Jenna Zaidspiner
Topic: 2
Floating Point Arithmetic and Errors

Floating point numbers

In order to understand floating point arithmetic, we must define what a floating point number is. A floating point number is an integer value with a specified number of decimal places after it. For example, $-2.987, 5$, and 3.678239 are all floating point numbers with a different number of decimal places (a different amount of precision). There are 3 components to a floating point number, the base (β), precision (p), and exponent range ($[L, U]$, where L is the lower bound for the exponent and U is the upper). A number x is then written as

$$x = (d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + ... + \frac{d_{p-1}}{\beta^{p-1}})\beta^E$$

where $0 \le d_i \le \beta - 1, i = 0, ..., p-1, E \in [L, U]$. The mantissa is the value $d_0 d_1 ... d_{p-1}$. Numbers that aren't exactly representable in floating point are called *machine numbers.*

Floating point arithmetic

Addition and Subtraction

When adding and subtracting floating point numbers, you must shift the mantissas of the numbers so that the exponents of each number match. For example, to compute $1.4561 + 101.67$, we first write the problem as $1.4561 \times 10^0 + 1.0167 \times 10^2$ and note that the exponents do not match. Thus we shift the mantissa of the first number so that the new problem is

$$.014561 \times 10^2 + 1.0167 \times 10^2$$

We can then add the numbers together normally and we get 1.0313×10^2 or 103.13. If you were to calculate this number exactly, it would actually be 103.1261, but each number in the problem only had a precision of 5, so the solution must be precision 5 as well. This is where floating point errors come into play since the solution calculated is not exact. We will discuss the errors associated with each arithmetic operation in a later section, but for now it is important to understand the operations themselves.

Multiplication

Just like with addition and subtraction, you still must shift the mantissa. However, you do not shift it to make the exponent match, you shift it so that there is only one digit in front of the decimal place for each number. Then you simply multiply the significands (the part of the number that is before the exponent) and add the exponents. For example, if we want to calculate 1.234×208.5, we do the following steps:

$$\begin{aligned} 1.234 \times 10^0 * 2.085 \times 10^2 \\ = (1.234 * 2.085) \times (10^{0+2}) \\ = 2.57289 \times 10^2 \\ = 257.3 \end{aligned}$$

Notice that we have a loss of digits in the multiplication case as well since the answer must have the same precision as the starting values.

Division

Similar to multiplication, division shifts the mantissa so that there is one digit before the decimal, but you subtract the exponents from each other instead of adding them. For example, consider the problem $486.3 \div 31.41$. Then the problem goes as follows:

$$\begin{aligned} 4.863 \times 10^2 &\div 3.141 \times 10^1 \\ = (4.863 \div 3.141) &\times (10^{2-1}) \\ &= 1.548 \times 10^1 \\ &= 15.48 \end{aligned}$$

The result when you put this problem in a calculator has a higher precision answer than 4, but since the two numbers in the problem had precision 4, we had to round the answer to have precision 4 as well.

Floating point errors

Machine epsilon

Machine epsilon characterizes the accuracy of a floating point system, and is denoted ϵ_{mach}. The value of machine epsilon is dependent on the type of rounding that is being used in the problem. Maximum relative error also relates to machine epsilon. Relative error is the difference between an exact value and an approximation to it, divided by the exact value. The maximum amount of relative error that a problem can have is given by

$$|\frac{\mathrm{fl}(x) - x}{x}| \leq \epsilon_{mach}$$

where $\mathrm{fl}(x)$ is the value of x after correctly rounded floating point arithmetic, and x is a non-zero real number within the range of the floating point problem. Machine epsilon is inversely related to the number of digits in the mantissa; the higher precision a number has, the smaller machine epsilon will be, and therefore, the smaller the error will be. Thus, we can lessen floating point errors by increasing the precision of the numbers in the problem.

Round-off error

Round-off error is a common error associated with floating point arithmetic. There are 2 rounding rules that contribute to round-off error: *round-to-nearest*, and *chopping* (or *round-by-chop*).

1. **Round-to-nearest**

   Round-to-nearest means that you round the number, x, to the floating point number closest to x. In the event of a tie, you round to the floating point number whose last stored digit is even, which is also why this method is sometimes called "round-to-even". For example, consider the number 4.661 and we want a number with precision 2. Then using round-to-nearest, we get the value 4.7. Now consider the number 4.450. Since 5 is equally close to 0 and 10, we have a tie. The number is then rounded to the nearest even number, which is 4.4. For the number 4.451 however, we no longer have a tie, and the number is rounded to 4.5.

   Because there are more rules when applying round-to-nearest, it is computationally more expensive than using the chop method. However, this also means that round-to-nearest is more accurate and has less round-off error in general than chop. In terms of machine epsilon for round-to-nearest, $\epsilon_{mach} = \frac{1}{2}\beta^{1-p}$.

2. **Chop**

   The chop rounding method says that you should truncate the value after the $(p-1)$st digit, also called "round to zero". So, if we want a number with precision 2, we only have one digit after the decimal place and ignore the digits after it. Using the same values that we used in our examples for round-to-nearest: 4.661 rounds to 4.6, 4.450 becomes 4.4, and 4.451 also becomes 4.4. This rounding method is not as accurate as round-to-nearest since it always rounds towards 0 and does not take into account the digits after the $(p-1)$st digit. Therefore, if we want to improve accuracy of our outcome, we should use round-to-nearest over the chop method. For rounding by chop, $\epsilon_{mach} = \beta^{1-p}$.

We can see that ϵ_{mach} for the chop method is twice the value of machine epsilon for round-to-nearest, which is in line with our conclusion that it is better to use round-to-nearest for higher accuracy when rounding since we want a small ϵ_{mach}. Round-off error is the main error associated with floating point arithmetic. In the case of addition and subtraction, round-off error can cause there to be a loss of digits of the smaller number in the operation since we must shift the mantissas to match. So, if one number is exceedingly larger than the other number being added or subtracted, the mantissa of the smaller number may shift so far as to not have any impact on the result. For example, say we are adding 23 and .000000012 (or 1.2×10^{-8}). Both numbers have precision of 2, so the solution will as well in floating point arithmetic. Shifting the mantissa of the smaller number, we get

$$\begin{aligned}
2.3 \times 10^1 &+ .0000000012 \times 10^1 \\
&= 2.3000000012 \times 10^1 \\
&= 23.000000012 \\
&= 23
\end{aligned}$$

See that because the second number being added was much smaller, all of its digits got ignored in the addition. The same thing can happen with subtraction. While this case is a more extreme example, take a look at the example we used in the section on floating point arithmetic. The last digit got lost in the addition, and we were left with 103.13 instead of 103.1261. While this round-off is not as impactful, it still causes some error in calculation since floating point arithmetic did not calculate the exact solution. Here, floating point errors can be reduced by increasing the precision of the solution. If the solution allowed for more precision, say 10 decimal places, then the exact solution would be calculated in the arithmetic. The errors grow when the size difference between the numbers grow and the precision is not large enough to account for this difference; that is when loss of digits becomes a problem and errors occur.

In the case of multiplication, given the two numbers being multiplied have a precision of p, the result can have a precision of up to $2p$. This means that there will have to be a loss of digits due to the result having to be precision p. In the example in the section above, the true result was 257.289, but we need precision 4, so we rounded the result to be 257.3. Thus, the last 2 digits were lost and, using round-to-nearest, the 2 in the 10s place was rounded to a 3. Errors in the multiplication case grow as the difference in precision between the starting values and the result grows. So there will be more error in a result with $2p$ precision since more digits will be lost in rounding. Likewise, for division, the quotient of 2 p-digit numbers may contain more than p digits and lose digits in the result to round-off error.

Truncation errors

Truncation errors occur when we approximate a function using truncation; the error is the difference between the actual function and the truncated value of the function. These usually occur when we approximate functions that can be represented using an infinite series instead of a finite one. For example, let's consider the Taylor series of $\cos(x)$. This can be represented as the infinite series

$$\cos(x) \approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + ... + \sum_{n=4}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}$$

In order to use all of the terms of the Taylor series, of which there are an infinite amount, it would take an infinite amount of computation time. So, we will only use a finite amount of the terms to get an accurate approximation of cosine. Say we decide to only use the first 3 term (up to $n = 2$). Then we have

$$\cos(x) \approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!}$$

Then the truncation error is all of the terms after the third one, since we truncated the series to be a finite set of values with what we deem to be an appropriate amount of terms to yield an accurate result. Round-off error rounds the final digit in computation, while truncation error cute of the sequence/floating point number after a certain number of digits, not taking into account the following digits. Truncation errors will grow the more digits you leave out. For example, if we only used the first term of cosine, we would have that $\cos(x) \approx 1$, which is not sufficiently accurate. But if we allow more terms to be included in the computation, then we can get to a point where the remaining terms are sufficiently small that they do not have a significant impact on the approximation. Adding more terms, however, takes more computational time, so you must find the balance between how much accuracy you require to deem your result an accurate approximation and how much time you are willing to spend to get that accuracy.

Catastrophic cancellation

Catastrophic cancellation mainly occurs in the case of subtraction, when subtracting a relatively accurate approximation of two close floating point numbers results in a very bad approximation of their difference. This occurs because subtraction is inherently ill-conditioned at nearby inputs. Even if the inputs have a small relative error, the relative error of their approximate difference can majorly differ from the true value of their difference. Subtraction does not always cause catastrophic cancellation, but it can certainly amplify any floating point arithmetic errors that occurred in the calculation of the input values.

For example, say we want to find the largest root of the equation $y = ax^2 + bx + c$, where $a = 1, b = 25$, and $c = 1$. Then we have to solve $x^2 + 25x + 1 = 0$ for the largest root. Using floating point arithmetic to solve the quadratic equation, we have

$$\begin{aligned} x &= \frac{-b + \sqrt{b^2 - 4ac}}{2a} \\ &= \frac{-25 + \sqrt{25^2 - 4}}{2} \\ &= \frac{-25 + \sqrt{621}}{2} \\ &= \frac{-25 + 24.9}{2} \\ &= -.05 \end{aligned}$$

The true value of this calculation is actually $-.040064$. Thus, we calculate a relative error of $|\frac{-.05+.04}{-.04}| = .25$, which is a 25% error. Firstly, when we take the square root of 621 in the quadratic equation, we have a loss of digits due to round-off error. The value of the square root when we allow a higher precision is 24.9199. Then, because 24.9 is close in magnitude but opposite in sign to -25 we encounter catastrophic cancellation in this subtraction. The round-off error in the calculation of the square root contributed to the amplification of the error in the subtraction calculation later in the problem and subsequently the relative error calculation. This problem can be avoided by manipulating the quadratic equation. Multiplying the numerator and denominator of the quadratic equation by

$$\frac{-b - \sqrt{b^2 - 4ac}}{-b - \sqrt{b^2 - 4ac}}$$

gives us the new equation

$$x = \frac{2ac}{-b - \sqrt{b^2 - 4ac}}$$

The relative error using this new formula is only .1%. The manipulated equation makes it so that the 2 numbers of similar magnitude are being added instead of subtracted, so no catastrophic cancellation occurs.

. . . .

References

1. Wikipedia contributors. (2022, November 29). Floating-point arithmetic. Wikipedia. https://en.wikipedia.org/wiki/Floating-point_arithmetic

2. Wikipedia contributors. (2022b, November 29). Round-off error. Wikipedia. https://en.wikipedia.org/wiki/Round-off_error

3. ALAFF Catastrophic cancellation. (n.d.). https://www.cs.utexas.edu/users/flame/laff/alaff/a2appendix-catastrophic-cancellation.html

4. Stephen Bryngelson class notes, Fall 2022

5. Wikipedia contributors. (2022a, November 11). Catastrophic cancellation. Wikipedia. https://en.wikipedia.org/wiki/Catastrophic_cancellation