Numerical Analysis Final Project Write-Up

1. A problem with floating point arithmetic

Floating point numbers can be frustrating to work with sometimes. They are good enough, fast, and computationally inexpensive, but can be inaccurate. For example, they do not keep many of the expected properties of real arithmetic such as associativity. If you compute $10^{100} + 1 - 10^{100}$, infinite precision arithmetic would tell you the answer is 1 but floating point numbers will tell you it is 0. Other times, they produce results that seem nonsensical in infinite precision arithmetic. For example, in Python, $0.1 + 0.2 = 0.30000000000000004$. With these shortcomings in mind, can we find a better system? We could try doing arithmetic with rational numbers by representing a number as a pair of 2 unbounded integers. However, if each of the input numerators and denominators contains n bits, then the output number is represented as roughly $4n$ bits, around as big as the total size of the inputs. As expressions become deeply nested, this can compound quickly. Overall, this is not a practical solution to the problem.

Instead, we could consider doing arithmetic with computable real numbers. Informally, a real number is *computable* if there is an algorithm that prints the decimal digits of the number to any desired accuracy. From this definition, it isn't very difficult to see that rational numbers are computable numbers. For example, the long division computes the decimal digits of a rational number up to any desired precision. However, it is much less obvious that there are computable irrational numbers. Familiar numbers like π and e are computable because they can be computed from infinite series to desired precision. If you want more decimal places, just add more terms of the series. This leads naturally to the question: are there incomputable numbers? It turns out that the answer is yes. In fact, almost every real number is incomputable because there are only countably many possible programs. Fortunately for us, most of the numbers that we care about for scientific computing are computable. It is highly unlikely that people who aren't invested in computability theory have ever seen an incomputable number in their life. With this is mind, we can focus our attention on just the computable reals. This leads to another question: are the computable reals a good approximation for any arbitrary real number? It turns out that the answer is yes because the computable reals are dense in the reals. Informally, a set A is *dense* in B if elements of A are arbitrarily close to elements of B. The most common example of this is that the rationals are dense in the reals. A common way to understand this is to consider that between any two real numbers, you can find infinitely many rationals. Elements of the set $\{\frac{m}{2^n} \in \mathbb{Q} \mid m, n \in \mathbb{Z}\}$ are called *dyadic rationals.* Equivalently, they are the real numbers that have a terminating binary representation. It turns out that the dyadic rationals are dense in the rationals. A consequence of this is that real numbers (and therefore computable reals) can be arbitrarily well approximated by dyadic rationals, which is amazing for finite computations in binary. To represent a real number, we must first define a computable function. Informally, a function f is *computable* if there is an algorithm that produces the value of $f(x)$ on input x. Addition is a computable function over the integers for example. With this, we can represent a computable (real) number x in binary by associating it with a computable function $f_x(n)$ that takes in a specified nonnegative integer tolerance n and outputs the nearest unbounded integer m such that the difference between x and $\frac{f_x(n)}{2^n}$ is less than the error tolerance. More explicitly, $|f_x(n) \cdot 2^{-n} - x| < 2^{-n}$. The way in which this is implemented is up to the language being used, but it always can be done for computable reals (at least based on the theory).

Without going through the gory details, the computable reals form a field. This means that we can do arithmetic with them to arbitrary precision, which is a massive improvement over floating point arithmetic. Additionally, they keep the nice properties of infinite precision arithmetic with real numbers including but not limited to associativity. After establishing these strengths, we must unfortunately expose the downfall of using computable reals instead of floating point numbers. Put simply, it may take a large amount of memory to actually compute certain numbers to a desired accuracy. The golden ratio φ, a computable real, is notoriously difficult to approximate by rationals. Another shortcoming of this approach takes the form of undecidability. Loosely, a relation or property is *decidable* if there is an algorithm that decides its truth in a finite amount of time. It turns out that it is not decidable whether two given computable reals are equal, or if one is greater than the other. To understand why, consider two different irrational numbers and a program that checks each decimal place for equality. The moment the program detects a difference, the numbers are declared unequal, but what if it takes longer than the heat death of the universe to actually find the first decimal place where they differ? Up until that point, the numbers are equal as far as the computer is concerned. Of course, some numbers are easy to compare, but given two arbitrary numbers, we cannot tell for certain in a finite amount of time. So if you were to type "$1 - 1$" into a computer, it could only tell you "$1 - 1$ is within a rounding error of $0.0000\ldots$," even though we know it is 0 Unfortunately, both of these flaws are massive and undermine the benefit of completely accurate arithmetic.

2. A solution

To address these issues, we can restrict our attention to the real numbers that can be represented by what are called "calculator operations." These include the four basic arithmetic operations, square roots, the sin, cos, and tan trigonometric functions and their inverses, and the exponential and natural log functions. These are the kinds of numbers we care about for scientific computing anyways, so this is a reasonable restriction, even if we can't reach every computable real this way. Remarkably, it is decidable whether a constant described by a system of equations over the complex numbers is equal to zero unless the problem contains a counterexample to Schanuel's conjecture, which is highly unlikely. Schanuel's conjecture states that given any set of n complex numbers $\{z_1, \ldots, z_n\}$ that are linearly independent over $\mathbb{Q}$, the field extension $\mathbb{Q}(z_1, ..., z_n, e^{z_1}, ..., e^{z_n})$ has transcendence degree at least n over $\mathbb{Q}$. This conjecture proves a that large class of numbers are transcendental, like $e + \pi$ and $\sin(e)$. The previously referenced equations can take the following two forms: a multivariate polynomial is equal to zero, or $e^x = y$. Fortunately for us, all of the "calculator operations" can be represented this way. For example the complex definition of sine is $\sin(z) = \frac{e^{iz} - e^{-iz}}{2i}$. With this background, we can finally arrive at the solution. It combines exact rational arithmetic and "calculator operation" computable real numbers. A real number (replacing a float) can be defined with 3 fields: a rational number (represented by the standard two BigIntegers), a computable real number (the dyadic rational approximation), and a property (essentially storing if we have a special number like π). The represented real number is the product of the rational factor and the computable real factor. For example, numbers like $\frac{\sqrt{2}}{2}$ are represented exactly as $\frac{1}{2} \cdot \sqrt{2}$. The property field has 2 types of values, a tag with a rational x, or a simple irrationality tag. Essentially, it just tells you whether you have a recognized form of a real number. To name a few, we have $\pi, \sqrt{x}, e^x, \ln(x), \sin(\pi x)$, and $\tan(\pi x)$. When adding two numbers whose computable real factors are the same, you just add the rational factors and preserve the computable real factor, just as you expect in infinite precision arithmetic. For example, $(1+\sqrt{2})+(\frac{1}{2}+\sqrt{2}) = (\frac{3}{2}+\sqrt{2})$. If you add two numbers of different types like $\ln(2)$ and $\sqrt{137}$, then we don't have a special tag like above, but just that the number is irrational. In this case, the stored number gets the irrationality tag.

Overall, this system of finite-precision arithmetic strikes a decent balance between accuracy and speed. Like the system of computable real arithmetic, the expected properties of infinite precision arithmetic hold. Additionally, we recover some decidability with our decision to restrict which computable reals we work with. Finally, we have some symbolic manipulation for common values with calculator operations, which leads to more exact answers. However, even with the combined approach, this system is still noticeably slower than floating point arithmetic. Performance and convergence are a large component of numerical analysis, so if speed is important, this system might not be what you need. Additionally, not all of the numbers we care about can be represented exactly in this system. More complicated formulas such as $\pi^2 - \pi^2 = 0$ will not be displayed so nicely. This system of real numbers is not a complete replacement for floating point numbers. It has some advantages, but its real purpose is to serve as an alternative system to floating point numbers that has different strengths and weaknesses. Depending on the context, either system could be better. For example, this real number system can be used when there is less calculation and greater accuracy is desired. However, if you need a massive amount of flops, floating point numbers will probably still be better. For example, Google implemented this system to the android calculator app to get more accurate answers. Given that the point of a calculator is to be as accurate as possible, this is the perfect place for such an implementation.

To conclude, this system of arbitrary precision arithmetic is very fascinating and demonstrates interesting applications of lesser known disciplines of math such as computability theory. It is certainly worth considering as an alternative to floating point numbers in certain contexts.

3. References

- Paper I used
  - `https://dl.acm.org/doi/pdf/10.1145/3385412.3386037`
- Summary of paper
  - `https://chadnauseam.com/coding/random/calculator-app`
- Decidability of solving calculator operation equations
  - `https://dl.acm.org/doi/pdf/10.1145/190347.190429`