Journal of Humanistic Mathematics

Volume 12 | Issue 1 January 2022

# The Nature of Numbers: Real Computing

Bradley J. Lucier
*Purdue University (Emeritus)*

# The Nature of Numbers: Real Computing

Bradley J. Lucier

*Purdue University, West Lafayette, Indiana, USA*
lucier@purdue.edu

**Synopsis**

While studying the computable real numbers as a professional mathematician, I came to see the computable reals, and not the real numbers as usually presented in undergraduate real analysis classes, as the natural culmination of my evolving understanding of numbers as a schoolchild. This paper attempts to trace and explain that evolution. The first part recounts the nature of numbers as they were presented to us grade-school children. In particular, the introduction of square roots induced a step change in my understanding of numbers. Another incident gave me insight into the brilliance of Alan Turing in his paper introducing both the computable real numbers and his famous "Turing machine". The final part of this paper describes the computable real numbers in enough detail to supplement the usual undergraduate real analysis class. An appendix presents programs that implement the examples in the text.

## 1. Background

### 1.1. *"Numbers are your friends."*

I used to tell my kids that. Numbers are *my* friends, at least.

The protagonist of Roddy Doyle's novel *The Dead Republic* [3] relates how for a period in the 1950s and '60s an optimistic and confident Republic of Ireland set out to educate *every* child, building schools, developing curricula, and hiring teachers as needed. That remark resonated with my memories of grade school in Ontario (1961–68), a period that began with memorizing addition and multiplication tables and, influenced by the so-called New Math, ended with a few weeks doing all arithmetic calculations in base seven!

Numbers were both useful and fun in grade school. Numbers felt comfortable, like friends.[1]

My understanding of numbers got foggier in high school. Learning in university about Dedekind cuts, the uncountability of the reals, completeness, etc., fed a certain excited arrogance, a feeling that this rigor, like a warm mid-morning breeze, had blown away the fog. But I was wrong—some wisps remained.

By training and by nature I'm a Numerical Analyst, which means that I compute approximations to either numbers or functions, generally because you can't compute them exactly. I sometimes want to bound with complete accuracy the error in a computation. It's like being asked to write down $\sin(\pi/4)$ to 4 decimal places: 0.7071. The true value is $\sqrt{2}/2 = 0.70710678\ldots$, and the error in your answer is $\sqrt{2}/2 - 0.7071 = 0.00000678\ldots$ It doesn't matter what comes after "678" because, whatever it is, the error is less than 0.00005, so you've answered the question correctly.

Later in my career I was drawn to the question of *which numbers* can be computed to any given accuracy, and how to do it. Such numbers are the *computable* or *constructive* reals.

In my quest I found John Harrison's Ph.D. thesis, "Theorem Proving with the Real Numbers" [7]; Section 4.5 provides explicit algorithms for calculating with computable real numbers.[2]

In studying and programming the algorithms found there, I realized that this approach to the real numbers, the *computable* reals, is the natural continuation of the approach begun in grade school; further, that the fog in my head in high school was a quite natural result of diverting from the grade-school approach *without anyone having explicitly said so*; and that the "rigor" of university held false promise for someone who, like me, wants to compute answers with guaranteed accuracy.

[1] "Numbers befriended him throughout the day." [4, "The Fruit Crate", page 81]

[2] Harrison presents specialized versions of methods developed by Valérie Ménissier-Morain; see [10] and the references therein. Ménissier-Morain also presents alternate formulations of the computable/constructive real numbers.

### *1.2. Who's the audience?*

I first organized this material for presentations to the Math Club at Purdue University. But much of the material presented here (not Section 4.14) is accessible to motivated high school students with an interest in mathematics and computer programming. (You'll need the generally useful ability to ignore mention of inessential notions you might not yet understand.) Some of the examples come from a first course in calculus. Wikipedia articles on fields, ordered fields, real closed fields, etc., might offer starting points for further reading, while [2] provides an introduction to the field of computable analysis itself.

## 2. History

### *2.1. Prehistory*

Before I started Grade 1 at age six, I didn't know my letters. I may not have known my numerals. I probably had some ideas about numbers.

School taught us about whole numbers: adding and subtracting one-digit numbers in Grade 1; adding with carrying and subtracting with borrowing in Grade 2; multiplication flashcard contests in Grade 3. Everything was about manipulating the *digits* of numbers.

### *2.2. Fractions and decimals: Diverging*

Fractions and decimal numbers were developed between Grades 4 and 6, entirely separately from each other.

Notions of accuracy were introduced with decimal numbers around Grade 4. For example, 3.14 is a better approximation to $\pi = 3.14159\ldots$ than is 3.1; even so, 3.1 is good enough for certain purposes. Let's say that "we know $x$ to $n$ decimals" when we know $n$ correctly rounded decimal digits of $x$ to the right of the decimal point. Technically, the difference between $x$ and its "approximation to $n$ decimals" is no more than $\frac{1}{2}10^{-n}$.

We needed to adapt the operations taught in earlier grades to work with these new "approximations". For example, if you want to know the value of $2\pi$ to one decimal and you argue, "Well, $\pi$ is 3.1 to one decimal, so $\pi + \pi$ is $3.1+3.1 = 6.2$ to one decimal", then you'd be wrong. $2\pi = 6.283185\ldots$, so to one decimal accuracy, $2\pi$ is 6.3. You need to compute the intermediate result to two decimals to get the final answer to one decimal: $3.14 + 3.14 = 6.28$, which, to one decimal accuracy, is 6.3. To know a sum of two numbers correct to $n$ decimals, you need to know the summands correct to at least $n + 1$ decimals: that's the rule.

This process can *fail*, although I don't remember my Grade 4 teacher making a point of this. Here's an example of when this fails in the form of a challenge game.

Let's assume that I'm thinking of two numbers, $x$ and $y$, and I tell you that, to two decimal places, $x$ is 1.11 and $y$ is 1.14. In other words, $x$ could be bigger or smaller than 1.11, but it rounds to 1.11; similarly, while $y$ rounds to 1.14, it could be bigger or smaller than 1.14.

I challenge you to tell me $x + y$ to one decimal accuracy.

Given what you know, you can say for certain that $x + y$ is about 2.25, but it could be bigger than 2.25, in which case you should round to 2.3, or it could be less than 2.25, in which case you should round to 2.2. You just can't know, based on the information I gave you, whether $x + y$ is, to one decimal accuracy, 2.2 or 2.3.

You might think that getting more decimals of $x$ and $y$ could help, but no! Ask me for three decimals of $x$ and $y$, and I'll say 1.110 and 1.140. Again, $x + y$ is about 2.250, but you still don't know whether it's bigger or smaller than 2.25, so you don't know whether to round to 2.2 or 2.3. Ask me for four, and I'll say 1.1100 and 1.1400, confounding you again. If the numbers I'm thinking of are exactly $x = 111/100$ and $y = 114/100$ and I give you correctly rounded approximations of arbitrary length to $x$ and $y$, you'll *never* be able give me $x+y$ rounded correctly to one digit to the right of the decimal point.

We can conclude that it's not always possible to compute correctly rounded decimal sums given only correctly rounded decimals as data.

## *2.3. Fractions and (some) decimals: Converging*

When I was 12, the teacher wrote a list like this on the board: $2/5 = 0.4$, $2/3 = 0.\overline{6}$, $1/12 = 0.08\overline{3}$, $5/7 = 0.\overline{714285}$, where the line over a string of digits means that that string repeats indefinitely. She pointed out that rationals seem to have decimal expansions with strings of digits that eventually repeat, and then asked: *Are (a) fractions and (b) numbers with decimal expansions that eventually repeat, the same kind of numbers?*

That was an interesting question, and I answered yes, with the following reasoning.

To show that fractions have decimal expansions that eventually repeat (possibly with 0's at the end), I pointed out that when computing the decimal expansion of $p/q$ using long division, the remainder at each stage must be one of 0, 1, …, $q-1$.

Let's consider what happens after you've passed the unit digit of $p$, so that you're always pulling down the digit 0 as the next digit of $p$ in long division. If any of the remainders after this point are 0, then the computation ends and you've calculated the (finite) decimal expansion of $p/q$. Otherwise, infinitely many remainders take only finitely many values (between 1 and $q-1$, inclusive), so at least one of the remainders repeats; at that point the entire computation repeats and the string of digits in the quotient must repeat.

It was a bit harder to explain that a number whose decimal expansion eventually repeats is also a fraction. Let's do an example.

Consider $x = .345\overline{0234}$. Multiplying it by 1,000 will leave only the repeating part to the right of the decimal point: $1{,}000\,x = 345.\overline{0234}$. We can multiply again by 10,000 to put one string of repeating digits to the left of the decimal point, and subtract:

$$\begin{aligned} 10{,}000 \times 1{,}000\,x &= 3{,}450{,}234.\overline{0234} \\ 1{,}000\,x &= \phantom{3{,}450{,}}345.\overline{0234} \\ 9{,}999{,}000\,x &= 3{,}449{,}889. \end{aligned}$$

So $x = 3{,}449{,}889/9{,}999{,}000 = 383{,}321/1{,}111{,}000$. One can do something similar for any repeating decimal.

## *2.4. Square roots: A crisis*

A bit later, the teacher pointed out that the square of any even number $2k$, which is $(2k)^2 = 4k^2$, is also even; and the square of any odd number $2k+1$, which is $(2k+1)^2 = 4k^2 + 4k + 1 = 4(k^2 + k) + 1$, is odd. So if a square of a whole number, $p^2$, is even, then $p$ must itself be even.

She then claimed that there is no rational number $p/q$ in lowest form (meaning $p$ and $q$ have no common factor) with $(p/q)^2 = 2$.

For if there were, then $p^2 = 2q^2$ is even, so $p$ itself must be even, or $p = 2k$ for some $k$. Then $p^2 = 4k^2 = 2q^2$ or $q^2 = 2k^2$, so $q^2$ is even, so $q$, again, must be even. So $p$ and $q$ must both be even. In other words, they have 2 as a common factor, which contradicts our assumption that we've already *removed* any common factors from $p$ and $q$, because we assumed that $p/q$ was in lowest form.

Then the teacher said that, nonetheless, there is a number, written $\sqrt{2}$, whose square is 2.

She explained that this meant that if you laid out all the rational numbers on a horizontal line, with $x$ to the left of $y$ if $x < y$, then there would be a hole or "gap" where you'd expect the number whose square is 2 to be, and that $\sqrt{2}$ fills that gap.

I objected—rather vigorously, in fact!

Until now, every number was rational, so it could be represented as a finite string of digits (with some of those digits possibly repeating forever). The digits of $\sqrt{2}$ could not eventually repeat, because it isn't rational, so it was impossible to write down all its digits at once.

*How could we know that a number exists if we can't write down a representation of its string of digits?*[3]

But I could see immediately that $\sqrt{2}$ lies between 1 and 2, because $1^2 < 2 < 2^2$, so its unit digit is 1; and quick mental calculation showed $1.4^2 = 1.96 < 2 < 1.5^2 = 2.25$, so $\sqrt{2} = 1.4\ldots$ The principle was clear: you could compute as many digits of $\sqrt{2}$ as you liked, at the very least by trial and error.

After school I told my mother that the teacher had told us about square roots, but she hadn't given us an effective method to compute their digits, which bothered me. My mother told me that my father could show me how, and eventually he did (a separate story, see Section 3). The significant aspect of the computation that we'll use later is that if you know $2n$ decimals of $x$, you can compute $n$ decimals of $\sqrt{x}$.[4]

[3]James Propp proposed almost exactly the same question [13]: "Students should also be led to see that the question 'But how do we know that the square root of two really exists, if we can't write down all its digits or give a pattern for them?' is a fairly intelligent question. In what sense do we know that such a number exists?" His paper, which considers various properties related to completeness, does not, however, mention the computable real numbers.

[4]Sometimes you can get by with fewer digits of $x$, but we'll not go into all the details.

After I encountered square roots, I thought of numbers no longer as objects represented by finite strings of digits (with some possibly repeating), but as objects with an associated process that would compute whatever digits of the number you might need. You may never be able to write down all the digits at once, but you'd be able to compute as many as you need at a given time. Today, you'd implement that "process" as a computer program.

And if you couldn't compute an object's digits, it wasn't a number.

### *2.5. High school: Fog*

I didn't think much about the nature of numbers in high school, I was too busy learning about things like trigonometry: What was $\sin(1/4)$? $\arctan(1/2)$? I guess I didn't care, I just assumed they existed.

At the time, there was Grade 13 in Ontario, where one was taught "baby" calculus, so I learned about $e = 2.718281828459\ldots$, and how to compute both $\pi$ and $e$ using series. Series also helped with $\sin(1/4)$ and $\arctan(1/2)$.

I read a number of books from what is now called the *Anneli Lax New Mathematical Library*, published by the Mathematical Association of America and directed to high school students. There I learned about other representations of numbers (e.g., continued fractions [11]). But there was no overarching theory of what numbers were, and evidently I didn't think I needed one.

### *2.6. University: Rigor, but ...*

In the 1970s, all first-year science majors at the University of Windsor were required to take a course called *Basic Concepts of Mathematics*, developed by Elias Zakon, which covered some properties of the real numbers as a complete ordered field; see [16].

There it was, "the" real numbers.

The course showed that completeness implies the Archimedean property: that for all positive numbers $x$ and $y$, there is a natural number $n$ such that $n \times x > y$.

Cantor's diagonalization argument was used to prove that the real numbers could not be put in a list, so they could not be indexed by the natural numbers: $x_1, x_2, \ldots$ This argument went something like this:

Let's assume that we can put all real numbers strictly between 0 and 1 in a list, e.g.,

$$\begin{aligned} x_1 &= 0.8574025375628211\ldots \\ x_2 &= 0.2876463473845367\ldots \\ x_3 &= 0.7342236104231586\ldots \\ &\vdots \end{aligned}$$

Cantor then said: I'll specify a new number $\hat{x}$ whose first digit is different from the first digit of $x_1$, whose second digit is different from the second digit of $x_2$, and, in general, whose $n$th digit is different from the $n$th digit of $x_n$, $n = 1, 2, \ldots$

Here's one rule that works: if $d$ is the $n$th digit of $x_n$, then let the $n$th digit of $\hat{x}$ be 7 if $d < 5$ and 3 otherwise. If I repeat the previous list while underlining the $n$th digit of the $n$th number in the list, I get

$$\begin{aligned} x_1 &= 0.\underline{8}574025375628211\ldots \\ x_2 &= 0.2\underline{8}76463473845367\ldots \\ x_3 &= 0.73\underline{4}2236104231586\ldots \\ &\vdots \end{aligned}$$

and using this rule we'd get $\hat{x} = 0.337\ldots$ You can see it's not $x_1$, because its first digit differs from the first digit of $x_1$; similarly for $x_2$ and $x_3$; but the same is true for *any* $x_n$ in the list, so *this* $\hat{x} = 0.337\ldots$ cannot be in our list.[5]

So our assumption that we can put all the real numbers in a list is false. Any set that can be put in a list is said to be *countable*. The real numbers are *uncountable*.

The approach in *Basic Concepts of Mathematics* was completely axiomatic: one simply *assumed* that the set of real numbers exists. Later sections in the book showed, however, that one could *construct* such numbers by associating them with so-called *Dedekind cuts*, which constructed the real numbers from special pairs of sets of rational numbers.

There are a number of ways to define Dedekind cuts. For our purposes, a cut is a pair $(A, B)$, where $A$ and $B$ are nonempty sets of rational numbers

[5]Two numbers whose digits all differ from each other can be equal, e.g., $0.500\ldots = 0.499\ldots$, but that can't happen with numbers all of whose digits are 3 or 7.

for which (a) every rational number is contained in $A$ or $B$, and (b) for all $x$ in $A$ and $y$ in $B$, $x \le y$. If $A$ has a maximum $z$, then $(A, B)$ represents that rational number; similarly if $B$ has a minimum. If $A$ has no maximum and $B$ has no minimum, then there is a "gap" between $A$ and $B$, with all the rational numbers in $A$ strictly to the left of the rational numbers in $B$ on the number line. The pair $(A, B)$ is then associated with the real number in that gap. Heuristically, the real numbers fill in all the gaps between the rational numbers.

To get back to our example of $x = \sqrt{2}$, we can define $A$ and $B$ by

$$A = \{x = p/q \mid x < 0 \text{ or } x^2 < 2\} \text{ and } B = \{y = p/q \mid y > 0 \text{ and } y^2 > 2\}.$$

Then Dedekind *defined* $\sqrt{2} = (A, B)$; heuristically, $\sqrt{2}$ is the point in the gap between the rational numbers in $A$ and $B$.

By this point I had completely forgotten my early struggles with numbers and reveled in all the great stuff I was learning at university: that every vector space has a basis, that you can't measure the volume of some sets of points, etc.

Perhaps I should have retained some of my (literally) childish skepticism in my approach to numbers, because I did not suspect for a moment that, as we'll see later, Dedekind's approach introduces "numbers" *for which one cannot compute digits* (for any reasonable meaning of the word "compute").

## 3. Interlude: Computing Square Roots

### *3.1. History, Part I*

My teacher introduced square roots, but not a way to compute them except through trial and error, digit by digit.

I told my mother that I wasn't happy about this when I got home from school, and she said that my father, who had a small accounting office, knew how to compute square roots, and he could show me when he got home.

I was very excited about this, called him at work, and insisted that he tell me over the phone before he came home, which he eventually did: "For example, to compute the square root of 2, take a blank sheet of paper..." I'll give the reasoning behind the method here.

### 3.2. *But first: Watch a video!*

It's much easier to understand the procedure for computing square roots by hand if you see an example being done dynamically. Of the many videos on YouTube, I especially like the one by Tibees [14].

### 3.3. *Mathematical justification*

We'll say that we know the "integer square root" $x$ of a positive integer $y$ if $x^2 \le y < (x+1)^2$.

The question is, can we use $x$, the integer square root of $y$, to quickly learn the integer square root of a number that begins with the digits of $y$ but has two more digits, $y_1y_2$, adjoined at the end? The new number would be $100y + 10y_1 + y_2$. (This is how we define *adjoin*.)

The answer is yes. We can adjoin a single digit $x_1$ to the right of $x$, or $10x + x_1$. The new digit would be the biggest to satisfy

$$(10x + x_1)^2 \le 100y + 10y_1 + y_2,$$

or, expanding,

$$100x^2 + 20x \times x_1 + x_1^2 \le 100y + 10y_1 + y_2,$$

or, moving one term from the left side to the right and factoring,

$$(20x + x_1) \times x_1 \le 100(y - x^2) + 10y_1 + y_2.$$

Such a digit exists because the two-digit number $10y_1 + y_2 < 100$ and $y < (x+1)^2$ implies $y + 1 \le (x+1)^2$, so

$$(10x+0)^2 = 100x^2 \le 100y \le 100y + 10y_1 + y_2 < \text{(continued on next line)}$$
$$< 100y + 100 = 100(y+1) \le 100(x+1)^2 = (10x+10)^2.$$

So for $(10x + x_1)^2 \le 100y + 10y_1 + y_2$ to be satisfied, $x_1$ must satisfy $0 \le x_1 < 10$, i.e., it must be a digit.

### 3.4. *The algorithm*

The algorithm is roughly as follows: You keep track of $y - x^2$, your current square root "remainder", like the remainder in long division. Then you adjoin the next two digits of $y$ to the right of the remainder, computing $100(y - x^2) + 10y_1 + y_2$.

Along the left edge of the paper, you keep track of $2x$, where $x$ is your current answer. You then want to find the largest digit $x_1$ such that if you adjoin it to the right of $2x$ (forming $20x + x_1$) and multiply by $x_1$, subtracting the product from $100(y - x^2) + 10y_1 + y_2$ yields a new nonnegative remainder.

You then adjoin the new digit $x_1$ to the right of $x$, so the new answer is $10x + x_1$. To update the column on the left with twice the new answer, you need only double $x_1$ and add any carries this might generate (e.g., $2 \times 6 = 12$, with carry 1) into the current value of $20x$—you don't need to double all of $10x + x_1$ again.

And this works with any number, not just integers, because, well, it does.

In Figure 1, we use this method to compute the first digits of $\sqrt{2}$, beginning with $x = 1$.

$$\begin{array}{rl}
 & \;\;1.\;\;4\;\;1\;\;4 \\
 & \sqrt{2.\,00\,00\,00} \\
\boxed{1}\times\boxed{1} & \;\;\underline{1} \\
 & \;\;1\,00 \\
2\,\boxed{4}\times\boxed{4} & \;\;\;\;\underline{96} \\
 & \;\;\;\;\;\;4\,00 \\
2\,8\,\boxed{1}\times\boxed{1} & \;\;\;\;\;\;\underline{2\,81} \\
 & \;\;\;\;\;\;1\,19\,00 \\
2\,8\,2\,\boxed{4}\times\boxed{4} & \;\;\;\;\;\;\underline{1\,12\,96} \\
 & \;\;\;\;\;\;\;\;\;\;6\,04
\end{array}$$

Figure 1: Computing the square root of 2.

During the computation, the digits we've computed so far of $\sqrt{y}$ sit above the radical sign; the left column contains twice the number above the radical sign; and the last line is the current value of $y - x^2$.

The boxes contain the values of the new digit $x_1$ adjoined both to the right of $x$ (above the square root sign) and to the right of $2x$, in the left column. We subtract the product of $x_1$ and $20x + x_1$, the updated value on the left, from the old remainder to get the new remainder.

And because you must pull down two digits of $y$ to compute each new digits of $\sqrt{y}$, this algorithm proves our previous claim: If you know $2n$ digits of $y$ to the right of the decimal point, then one can calculate $n$ digits of $\sqrt{y}$ to the right of the decimal point.

## *3.5. History, Part II*

As I said, my father explained how to compute $\sqrt{2}$ to me *over the phone*. And it took a while. You can imagine my sheet of paper filling up as he explained what he was doing at work, with numbers "to the left", "up top", "down below", etc. At about the point I left the computation above, my father said "You should be able to see the pattern now" and I just sort of grunted. I was totally confused. The page was a messy mass of digits.

I didn't look forward to my father coming home that night, and, indeed, he right away asked to see the paper I had used for the computation. I remember him asking "What's this?" or something similar, perhaps a bit sharply (considering how much time he'd taken from his work that day). Then we sat together and he showed me again.

## *3.6. Explaining algorithms over the phone, and Alan Turing*

Years later, I remembered this incident while thinking about Alan Turing's 1936 paper [15] that introduced both his "machines" and the idea of "computable numbers".

Turing's famous minimal machine has an infinite tape divided into squares. At any time there is precisely one square of the tape in the "focus" of the machine. The machine can read what is in that square, write one of a finite set of symbols to that square, or can erase whatever symbol is on the square; after this the machine can move the tape one square to the left or to the right. The machine has an internal state, and can make decisions about what to do depending on what is written on the square directly in its view.

In his paper, Turing cites grade-school arithmetic on a two-dimensional grid as motivation. In grade school, as in this document, we use only finitely many symbols, beginning with the decimal digits 0 through 9. The way one lines up the digits, left-and-right and up-and-down, to compute $\sqrt{y} = \sqrt{2}$, for example, and changes the focus from the current computation at the bottom of the page back up to the top to get more digits of $y$, is similar to moving Turing's tape back and forth after adding symbols to it.

Turing's tape is infinite in the same way that our page would need to extend arbitrarily far to the left, right, and downward to compute arbitrarily many digits of $\sqrt{2}$.

Turing was teaching a machine to compute numbers in the same way my father was teaching me over the phone to compute square roots—I was the machine. And had he used chess notation to designate positions on a square grid (e.g., "Put '2' in the square in column P of row 4."), like the way Turing used a tape divided into boxes, my father would have succeeded.

## 4. The Computable Reals

### *4.1. What is computable?*

Because computers are ubiquitous in our lives, we'll take an informal and practical approach to computability: an object is computable if there can exist a computer program that is guaranteed to compute it in a finite number of steps.[6]

So pick a programming language and a character set in which to write programs. I generally prefer to write programs in Scheme, a member of the Lisp family of programming languages, and I'll use the Latin-1 character set, a 256-character subset of the complete Unicode character set.

### *4.2. The definition*

We start with my old grade-school idea: something is a number if and only if we can compute as many digits of it as we want.

But that notion isn't precise enough, so let's see what will work.

One issue is that we want to do operations on these numbers—add them, multiply them, etc. This means that *we have to be ready to accept as input to addition, multiplication, etc., what we're producing as outputs of those operations.*

[6] In the first half of the 20th century, many brilliant minds worked on the question of computability. Alan Turing came up with his *machines*; Alonzo Church invented the *lambda calculus*; a number of people developed *recursive function theory.* Church and Turing proved that all these theories defined the same class of functions; afterwards, the *Church–Turing thesis*, which claims that these three equivalent theories capture all informally specified "effectively calculable functions" was widely accepted. This is related to the notion of *Turing completeness.*

We've shown in a previous section by an example that it is not guaranteed (in all cases) to compute correctly rounded sums of numbers given only correctly rounded numbers as inputs (see Section 2.2).

So we have to weaken our definitions a bit: we say that a real number $x$ is *computable*[7] if and only if there can exist a computer program that computes for each positive integer $k$ (roughly, the number of digits we want to the right of the decimal point) an integer $N$ such that

$$|N - 10^k x| < 1, \text{ or equivalently } \left|\frac{N}{10^k} - x\right| < \frac{1}{10^k}. \tag{1}$$

A similar formula can be found in Section 4.5 of [7].

For example, if $x = \pi = 3.14159\ldots$ and $k = 2$, then $N = 314$ or $N = 315$ will work because both

$$|314 - 10^2\pi| = |-.15926535\ldots| < 1 \text{ and } |315 - 10^2\pi| = |.8407346\ldots\ldots| < 1.$$

The number $N/10^k$, 3.14 or 3.15 in our case, is our approximation to $x$ to $k$ decimal digits. We don't require that $N/10^k$ be a correctly rounded approximation; our expectations are weaker.

Note, however, that if $10^k x$ is an integer, then because $N$ is an integer and $|N - 10^k x| < 1$, then $N = 10^k x$ (they're both the same integer). In particular, if $x = 0$ then $N = 0$ for all $k \geq 0$.

We take "we know a computable real $x$" or "we're given a computable real $x$" to mean that we have in hand a program that for any $k$ will compute an $N$ that satisfies (1).

Given a computable real $x$ and nonnegative integer $k$, it will be useful (if an abuse of notation) to denote any $N$ that satisfies (1) by $x(k)$. In some cases $x(k)$ is unique, but usually $x(k)$ could be one of two adjacent integer values—in the example with $\pi$, $x(2) \in \{314, 315\}$. In general $x(k)$ is not a function, with a unique value for each $k$, but a *relation*, with two possible values for each $k$. *In everything that follows, it doesn't matter which value of $x(k)$ we use.* In fact, we'll see that $x(k)$ can take different values in different contexts.

[7] To simplify the presentation, we assume the existence of Dedekind's real numbers, only some of which are computable; this isn't necessary.

## *4.3. Background: Manipulating inequalities*

The expression that defines $x(k)$, $|x(k) - 10^k x| < 1$, involves (a) absolute values and (b) inequalities. To be helpful, we review here properties of inequalities and absolute values.

First, we need the fact that $|a| < b$ if and only if $-b < a < b$.

More importantly, we need the *triangle inequality*, $|a + b| \leq |a| + |b|$. If we want to show that $|a+b| < 1$, for example, we might first show that $|a| < 1/2$ and $|b| \leq 1/2$, so $|a + b| \leq |a| + |b| < 1/2 + 1/2 = 1$.

And we want to compute an *integer* $x(k)$ such that $|x(k) - 10^k x| < 1$. Our strategy will often be to find an intermediate result, another number $X$, for which we can show through two separate arguments that $|X - 10^k x| < 1/2$ *and* that we can compute an integer $x(k)$ for which $|x(k) - X| \leq 1/2$. Then subtracting and adding $X$ and using the triangle inequality gives

$$|x(k)-10^k x| = |(x(k)-X)+(X-10^k x)| \leq |x(k)-X|+|X-10^k x| < \frac{1}{2}+\frac{1}{2} = 1.$$

## *4.4. Simple consequences of the definition*

Because $|x(k) - 10^k x| < 1$, we have $-1 < 10^k x - x(k) < 1$ or $x(k) - 1 < 10^k x < x(k) + 1$. So, if $x(k) \geq 1$, we know that $10^k x > x(k) - 1 \geq 0$, so $x > 0$. If $x(k) \leq -1$, we know $x < 0$.

Similarly, if $x(k) > N$ for some integer $N$, then $10^k x > N$; so, if $x(k) > 1$ then $10^k x > 1$.

Finally, if, for some $k$, $|x(k) - y(k)| > 1$, then $x \neq y$.

## *4.5. Two useful functions*

We'll gather together two useful functions.

The first is the round operator, which takes any *rational* number $p/q$ and produces an integer

$$\text{round}\left(\frac{p}{q}\right) = N \quad \text{for } N - \frac{1}{2} \leq \frac{p}{q} < N + \frac{1}{2}.$$

By restricting the argument to be rational, we know grade-school methods to compute round, which gives a better bound on the error than we guarantee for computable real numbers:

$$\left|\operatorname{round}\left(\frac{p}{q}\right) - \frac{p}{q}\right| \leq \frac{1}{2} \text{ rather than } |x(k) - 10^k x| < 1.$$

Our second useful function involves square roots. We gave an algorithm (in Section 3.4) to find, for any nonnegative integer $y$, a nonnegative integer $x$ such that $x^2 \leq y < (x+1)^2$. We can conclude that this $x$ satisfies $x \leq \sqrt{y} < x+1$, which implies $|x - \sqrt{y}| < 1$, but we need to do a bit better.

We write $y = x^2 + r$, where $r$ is the remainder. In our new integer square root routine, which we'll call isqrt, we want to return $x$ if $y = x^2 + r < (x+1/2)^2 = x^2 + x + 1/4$ and to return $x+1$ if $y = x^2 + r > (x+1/2)^2 = x^2 + x + 1/4$. Since $x$, $y$, and $r$ are all integers, we see that

$$\operatorname{isqrt}(y) = \begin{cases} x+1, & r > x, \\ x, & \text{otherwise.} \end{cases}$$

So, for *integer* $y$, we can compute an integer $\operatorname{isqrt}(y)$ such that $\left|\operatorname{isqrt}(y) - \sqrt{y}\right| < 1/2$.

## *4.6. Rational numbers are computable*

To see that each rational number $x = p/q$ is computable, note that for any $k \geq 0$, $10^k x$ is rational and $|\operatorname{round}(10^k x) - 10^k x| \leq \frac{1}{2} < 1$, so the integer $x(k) = \operatorname{round}(10^k x)$ satisfies our definition (1).

## *4.7. The computable numbers are a field*

Given computable $x$ and $y$, is $z = x + y$ computable? What about $x \times y$? $1/x$ for $x \neq 0$? $\sqrt{x}$?

Let's start with addition. The question is: If $x$ and $y$ are computable, is $z = x + y$ computable? That is, for any $k$ can we compute an integer $z(k)$ such that $|z(k) - 10^k(x+y)| < 1$?

The answer is yes, and the method is more or less what was taught in Grade 4. First we compute $x(k+1)$ and $y(k+1)$ (which we can do by assumption, because $x$ and $y$ are computable), one more digit than we want to compute of $z$. Then

$$|x(k+1) - 10^{k+1}x| < 1 \text{ and } |y(k+1) - 10^{k+1}y| < 1.$$

Dividing each of these inequalities by 10 gives

$$\left|\frac{x(k+1)}{10} - 10^k x\right| < \frac{1}{10} \text{ and } \left|\frac{y(k+1)}{10} - 10^k y\right| < \frac{1}{10}.$$

So

$$\left|\frac{x(k+1)+y(k+1)}{10} - 10^k(x+y)\right| \\ \le \left|\frac{x(k+1)}{10} - 10^k x\right| + \left|\frac{y(k+1)}{10} - 10^k y\right| < \frac{2}{10}.$$

But $[x(k+1)+y(k+1)]/10$ is now *rational* so the integer

$$z(k) := \text{round}\left(\frac{x(k+1)+y(k+1)}{10}\right)$$

satisfies

$$\left|z(k) - \frac{x(k+1)+y(k+1)}{10}\right| \le \frac{1}{2}.$$

Adding the previous two inequalities gives

$$|z(k) - 10^k(x+y)| < \frac{2}{10} + \frac{1}{2} < 1, \text{ as required.}$$

Computing the negative of a computable real is trivial: If $|x(k) - 10^k x| < 1$ then for $y = -x$ we can take $y(k) = -x(k)$.

Explicit algorithms (in base two arithmetic instead of base 10) for multiplication and inverse ($1/x$ for nonzero $x$) are given in [7], Section 4.5. Beyond the following special case of of dividing a computable real $x$ by a positive integer $N$, we won't discuss them here.

We note that if $|x(k) - 10^k x| < 1$, $N > 1$, and $y = x/N$, then

$$\left|\frac{x(k)}{N} - 10^k \frac{x}{N}\right| < \frac{1}{N} \le \frac{1}{2},$$

and because $x(k)/N$ is *rational* we have $|\,\text{round}(x(k)/N) - x(k)/N| \le 1/2$. Combining these two inequalities gives $|\,\text{round}(x(k)/N) - 10^k y| < 1$, so we can take $y(k) = \text{round}(x(k)/N)$.

In fact the computable reals form a field, a sub-field of the real numbers.

### *4.8. Equality*

One of the axioms of a field says that "for each number $x \neq 0$, there exists a number called $x^{-1}$ such that $x \times x^{-1} = 1$." So it's important to know whether any given $x$ is zero, especially when one wants to compute $x^{-1}$ or $y/x$.

But this is not computably decidable for the set of computable reals. Here's why: Say we want to know whether a given $x$ is zero, so we ask for $x(0)$, i.e., an integer such that $|x(0) - x| < 1$. Let's say $x(0) = 0$, then by definition of $x(0)$ we can say only that $-1 < x < 1$. If we then compute $x(20)$ and find $x(20) = 0$, then we know that $-10^{-20} < x < 10^{-20}$, but we aren't any closer to knowing whether $x$ is precisely zero or not. One cannot write a program that is guaranteed to be able to tell me whether $x$ is zero after a finite number of steps.

Similarly if we want to know whether $x > 0$: If we happen to find a $k$ such that $x(k) > 0$, then we know that $x > 0$. If we continue to get $x(k) = 0$ for all the $k$s we try, we can't tell whether $x > 0$, $x < 0$, or $x = 0$.

It is also not computable to determine the equality of computable numbers in general, for $x = y$ if and only if $x - y = 0$, which we have seen is, in general, not computable. (This lack of computability manifests itself in programs that don't stop, if you ask them the wrong questions!) This can arise in ways that are perhaps not anticipated; for example, trying to compute $\mathrm{round}(x)$ when $x$ is computable (and not just rational) is, in general, not computable because $\mathrm{round}(x) = N$ if and only if

$$N - \frac{1}{2} \leq x < N + \frac{1}{2}$$

and neither of those inequalities is computably decidable.

The computable reals form an ordered field, but the ordering is not computable.

### *4.9. Square roots*

We can use the function isqrt to show that if $y$ is nonnegative and computable, then so is $x = \sqrt{y}$.

For any $k$ we need to compute an integer $x(k)$ such that $|x(k) - 10^k x| < 1$, or $|x(k) - 10^k \sqrt{y}| < 1$.

The analysis of the algorithm depends on the simple formula $a^2 - b^2 = (a-b)(a+b)$, and we proceed as follows:

First compute $y(2k)$, for which $|y(2k) - 10^{2k}y| < 1$.

If $y(2k) = 0$, then we know $|0 - 10^{2k}y| < 1$ so $\left|10^k\sqrt{y}\right| < 1$, so $x(k) = 0 = \operatorname{isqrt}(y(2k))$ works.

If $y(2k) = 1$, then $y > 0$ and $|1 - 10^{2k}y| < 1$. Using $a^2 = 1$ and $b^2 = 10^{2k}y$ we find

$$|1-10^{2k}y| = \left|1-10^k\sqrt{y}\right|\times\left|1+10^k\sqrt{y}\right| < 1 \text{ or } \left|1-10^k\sqrt{y}\right| < \frac{1}{\left|1+10^k\sqrt{y}\right|} < 1,$$

so $x(k) = 1 = \operatorname{isqrt}(y(2k))$ again works.

Otherwise, when $y(2k) > 1$, we have $10^{2k}y > 1$, so both $\sqrt{y(2k)} > 1$ and $10^k\sqrt{y} > 1$. With $a^2 = y(2k)$ and $b^2 = 10^{2k}y$ we see

$$|y(2k) - 10^{2k}y| = \left|\sqrt{y(2k)} - 10^k\sqrt{y}\right| \times \left|\sqrt{y(2k)} + 10^k\sqrt{y}\right| < 1,$$

or

$$\left|\sqrt{y(2k)} - 10^k\sqrt{y}\right| < \frac{1}{\left|\sqrt{y(2k)} + 10^k\sqrt{y}\right|} < \frac{1}{2}.$$

Because $y(2k)$ is an *integer*, $\left|\operatorname{isqrt}(y(2k)) - \sqrt{y(2k)}\right| < 1/2$ and $x(k) := \operatorname{isqrt}(y(2k))$ satisfies $\left|x(k) - 10^k\sqrt{y}\right| < 1$.

One can show that the computable reals form a *real closed field*.

## *4.10. In general, $x(k)$ is a relation, not a function*

We've previously noted that the condition $|x(k) - 10^k x| < 1$ allows one to return $N$ or $N+1$ for $x(k)$ when $N < 10^k x < N+1$. Sometimes, for a given $x$, different ways of computing $x(k)$ could lead to either result.

For example, let

$$x = \frac{\sqrt{2}}{2} = \sqrt{\frac{1}{2}}$$

and consider the two associated ways of computing the "digits" of this number, which we obtain by combining our programs for taking the square root of a computable number, converting a rational number to a computable number, and dividing a computable number by a positive integer:

- Convert 2 to a computable number, compute its square root, then divide by 2. Call this program $\bar{x}(k)$,

$$\bar{x}(k) = \text{round}\left(\frac{\text{isqrt}(\text{round}(2 \times 10^{2k}))}{2}\right).$$

- Convert 1/2 to a computable number and compute its square root directly. Call this program $\hat{x}(k)$,

$$\hat{x}(k) = \text{isqrt}\left(\text{round}\left(\frac{1}{2} \times 10^{2k}\right)\right).$$

We can see that $x = \bar{x} = \hat{x}$, and the computable number $z = \bar{x} - \hat{x}$ is zero. Computing $\bar{x}(k)$, $\hat{x}(k)$, and $z(k)$ for various values of $k$ quickly leads us to discover that

$$\bar{x}(13) = 7071067811866 \neq \hat{x}(13) = 7071067811865,$$

yet $z(13) = 0$, as it must, since $z$ is zero. We can conclude that

$$0.7071067811865 < \frac{\sqrt{2}}{2} < 0.7071067811866.$$

Here is a simple, concrete example, where two programs to compute the "digits" of a computable number give two different results for a specific $k$.

## *4.11. How many computable reals?*

Because we're using a character set of 256 elements, there are precisely $256^n$ strings of $n$ Latin-1 characters. We can put all possible (valid and invalid) computer programs in a list:

1. Start with listing all 256 length-one strings.
2. Add all $256^2 = 65{,}536$ length-two strings to the list.
3. Etc.

There are countably many finite-length strings of characters, so countably many computer programs.

Most of these programs are not even syntactically valid;[8] others never stop. Of those programs that run to completion, some print the string "hello, world",[9] while only some compute $x(k)$ for some computable real number $x$. So there are at most countably many computable real numbers, and most real numbers (in Dedekind's sense) are *not* computable.

Somewhat paradoxically, however, Cantor's diagonalization argument shows again that you cannot put all the computable real numbers between 0 and 1 in a list $x_1, x_2, x_3, \dots$ For if you could, you could compute $x_k(k)$, for example,

$$\begin{aligned} x_1(1) &= 8 \\ x_2(2) &= 28 \\ x_3(3) &= 734 \\ x_4(4) &= 0127 \\ &\vdots \end{aligned}$$

We can now compute a new real $\hat{x}$ whose digits are derived from the unit digits of $x_k(k)$ in the following way: If the unit digit of $x_k(k)$, $d$ say, satisfies $d < 5$, then set the $k$th digit of $\hat{x}$ to 7; otherwise, set it to 3. We take $\hat{x}(k)$ to be simply the first $k$ digits in this expansion.

By the way we constructed $\hat{x}$, we have $|\hat{x}(k) - 10^k\hat{x}| \leq 0.7777\ldots < 1$; furthermore, $|\hat{x}(k) - x_k(k)| \geq 2 > 1$, so we know that $\hat{x} \neq x_k$ for *every* $k$. So $\hat{x}$ is *not* in this list!

What is going on? There are no more computable reals than programs, and there are clearly only countably many programs, yet if we assume we can put all the computable reals into a list, we can always find another computable real *not* in that list!

What this construction shows is that *one cannot compute the list of computable reals*—one cannot write a computer program to pick out from the list of *all* possible programs those programs that represent the computable reals.[10]

[8]Except maybe in the programming language Perl; see [9].

[9]The first example program in Kernighan and Ritchie's 1978 tutorial *The C Programming Language* [8].

[10]Indeed, in his paper [15], Turing showed that one cannot write a program to determine whether a general program will halt, let alone whether it will halt and return an integer satisfying (1).

### *4.12. A real number that is not computable*

The previous section shows that we cannot computably tell which computer programs correspond to computable reals and which do not. This result allows us to specify a real number that is not computable.

To simplify, we assume this number $U$ lies between 0 and 1, and we specify its decimal digits. In particular, we set the $M$th digit of $U$ to be 3 if the $M$th program in our list of all possible programs is a program for a computable real, and 7 otherwise. Since one cannot computably decide for every program whether that program computes a computable real, the digits of $U$ are not computable. Therefore $U$ itself is not computable.

### *4.13. All useful numbers are computable*

To me this is almost a tautology—if you can't compute the value of a number to arbitrary accuracy, then that number isn't very useful!

Let's look specifically at $\pi$ and $e$, perhaps the two most famous "useful" numbers. Each can be defined in terms of convergent infinite series of rational numbers from calculus, for example,

$$\pi = 4\sum_{n=0}^{\infty} \frac{2}{(4n+1)(4n+3)} \text{ and } e = \sum_{n=0}^{\infty} \frac{1}{n!}.$$

Each of these series is of the form $A = \sum_{n=0}^{\infty} a_n$, with the properties that (a) the sequence of rational coefficients $\{a_n \mid n = 0, 1, 2, \ldots\}$ is itself computable (for any $N$ one can compute the first $N$ coefficients in a finite number of steps), and (b) there is a computable function $T(K)$, $K = 1, 2, \ldots$, (a *modulus of convergence*) returning nonnegative integers, such that the partial sums $A_N = \sum_{n=0}^{N} a_n$ satisfy[11] $|A_{T(K)} - A| < \frac{1}{K}$.

The limits of such series are themselves computable: for each $k$, let $K = 2 \times 10^k$ and $N = T(K)$; then

$$|A_N - A| < \frac{1}{2} \times 10^{-k} \text{ or } |10^k A_N - 10^k A| < \frac{1}{2}.$$

Now $10^k A_N$ is itself *rational*, so we find that $A(k) = \text{round}\big(10^k A_N\big)$ satisfies $|A(k) - 10^k A_N| \leq \frac{1}{2}$. Combining the two previous inequalities gives $|A(k) - 10^k A| < 1$, as needed.

[11]See Definition 3.1 in [1] or Definition 3.11 of [2] to find similar requirements.

An extension of this argument shows that if $x$ is a computable real within the radius of convergence of the power series

$$f(x) = \sum_{n=0}^{\infty} a_n x^n$$

with computable rational coefficients $a_n$ and a computable modulus of convergence, then $f(x)$ is also a computable real. So $\log x$, $e^x$, $\sin x$, $\cos x$, etc., are all computable functions.

## *4.14. All computable functions are continuous*

We prove here that all computable functions that are defined for all computable real arguments and return computable real results are continuous.[12]

To be more precise in this section, to every computable real number $x$ we associate one of Turing's "machines"[13] $X$, which, for each nonnegative integer argument $k$, stops after a finite number of steps and returns an integer $X(k)$ for which

$$\left|\frac{X(k)}{10^k} - x\right| < \frac{1}{10^k}, \quad k \geq 0. \tag{2}$$

(In this section $X$ is the machine and $X(k)$ is the value it returns.) For each computable $y$ there is a machine $Y$, etc.

So what can we learn from (2) for a fixed $K \geq 0$? First that

$$\frac{X(k)-1}{10^k} < x < \frac{X(k)+1}{10^k}, \quad k = 0, 1, \dots, K,$$

and the intersection of these intervals

$$\left(\frac{L_K}{10^K}, \frac{U_K}{10^K}\right) = \bigcap_{k=0}^{K} \left(\frac{X(k)-1}{10^k}, \frac{X(k)+1}{10^k}\right) \tag{3}$$

[12]Later material does not rely on this section, which is more technical than the others.

[13]You can think of $X$ as an algorithm, function, computer program, subroutine, procedure, closure, etc., depending on the terminology used by your favorite programming language. You can also think of a computer as a "universal" Turing machine that takes programs as representations of other Turing machines and executes them.

is nonempty—$x$ is in each of these intervals, after all. (Here $L$ is for *Lower* and $U$ is for *Upper*.)

Second, we note that if a computable real $y$ is also in the intersection, that is,

$$\frac{L_K}{10^K} < y < \frac{U_K}{10^K},$$

then

$$\frac{X(k)-1}{10^k} < y < \frac{X(k)+1}{10^k}, \quad k = 0, 1, \ldots, K.$$

This means that if $Y$ is a valid machine for $y$ (so $Y$ and $y$ satisfy (2)), then

$$\overline{Y}(k) = \begin{cases} X(k), & 0 \le k \le K, \\ Y(k), & K < k, \end{cases} \tag{4}$$

where we've now replaced $Y(k)$ with the value $X(k)$ for all $k \le K$, is *also* a valid machine for $y$; that is, $\overline{Y}$ and $y$ also satisfy (2).

We now consider the function $f(x)$. To say that $f(x)$ is computable means that there is a machine $F$ that has a slot into which one can plug in another machine $X$ associated with a computable real number $x$. As before, $X$ will take nonnegative integers as arguments and return integers. The machine $F$ after $X$ is plugged in specifies a *new* machine $F_X$,[14] which again takes nonnegative integers $k$ as arguments and returns integers $F_X(k)$ that satisfy

$$\left| \frac{F_X(k)}{10^k} - f(x) \right| < \frac{1}{10^k}, \quad k \ge 0. \tag{5}$$

The machine $F_X$, when applied to $k$, stops after a finite number of steps, during which it evaluates $X(\ell)$ for at most finitely many $\ell$; the number of evaluations of $X$, and the arguments $\ell$, may depend on $k$ and previous results that $X$ returns. $F_X$ cannot somehow look at the steps that $X$ takes in its computations: it can only receive the results $X(\ell)$.

We're now ready to show that we can make $f(y)$ as close to $f(x)$ as we like by specifying how close $y$ must be to $x$. In other words, if you give me a computable number $\epsilon > 0$ and a computable real $x$, I'll give you a (rational)

[14]We use subscript notation $F_X$ to represent the result of $F$ when applied to $X$ instead of functional notation $F(X)$ so we can write $F_X(k)$ instead of $F(X)(k)$.

number $\delta > 0$ such that $|f(x) - f(y)| < \epsilon$ for all computable $y$ for which $|x - y| < \delta$.

To begin, given $\epsilon > 0$, I can choose a $K \geq 0$ such that $1/10^K < \epsilon/4$. Given $x$ with associated machine $X$, I'm going to run the machine $F_X$ with the argument $K$ until it stops and returns $F_X(K)$ that satisfies

$$\left|\frac{F_X(K)}{10^K} - f(x)\right| < \frac{1}{10^K}. \tag{6}$$

If the machine $F_X$ when applied to $K$ *never* evaluates $X$ for *any* $k$, then $F_X(K) = F_Y(K)$ for any machine $Y$ for a computable real $y$. Using formula (6) for both $f(x)$ and $f(y)$ shows that

$$\begin{aligned} |f(x) - f(y)| &= \left|f(x) - \frac{F_X(K)}{10^k} + \frac{F_Y(K)}{10^k} - f(y)\right| \\ &\leq \left|f(x) - \frac{F_X(K)}{10^k}\right| + \left|\frac{F_Y(K)}{10^k} - f(y)\right| \\ &< \frac{1}{10^K} + \frac{1}{10^K} = \frac{2}{10^K} < \epsilon \end{aligned} \tag{7}$$

for *all* $y$.

Otherwise, let's denote by $K'$ the *largest* value of $k$ for which $F_X$ evaluates $X(k)$ when running the machine $F_X$ applied to $K$. We'll also compute the interval

$$(a, b) = \left(\frac{L_{K'}}{10^{K'}}, \frac{U_{K'}}{10^{K'}}\right) = \bigcap_{k=0}^{K'} \left(\frac{X(k) - 1}{10^k}, \frac{X(k) + 1}{10^k}\right)$$

from (3). Then for any $y$ in $(a, b)$, the machine $\overline{Y}$, defined by $\overline{Y}(k) = X(k)$ for $k = 0, \dots, K'$ and $\overline{Y}(k) = Y(k)$ for $k > K'$, is a valid machine for $y$ and $F_X(K) = F_{\overline{Y}}(K)$! Using (7) again, we see that

$$|f(x) - f(y)| < \frac{2}{10^K} < \epsilon, \tag{8}$$

but we don't yet have our $\delta$.

Both $a$ and $b$, the endpoints of our interval, are computable, with associated machines $A$ and $B$. So we can find $F_A(K)$ satisfying

$$\left|\frac{F_A(K)}{10^K} - f(a)\right| < \frac{1}{10^K},$$

with the same for $f(b)$.

The machine $F_A$ applied to $K$ evaluates $A$ for finitely many $m$, so let $M$ be the largest value for which $F_A$ evaluates $A$, and let $M' = \max(M, K')$. Then for any computable $y$ that satisfies $|y - a| < 1/10^{M'}$, the machine $\overline{Y}$ with

$$\overline{Y}(m) = \begin{cases} A(m), & 0 \le m \le M', \\ Y(m), & M' < m, \end{cases}$$

is valid for $y$[15] and satisfies $F_{\overline{Y}}(K) = F_A(K)$, so by (7), for all such $y$

$$|f(a) - f(y)| < \frac{2}{10^K}. \tag{9}$$

A similar argument shows that there is a nonnegative integer $N'$ such that for all computable $y$ satisfying $|y - b| < 1/10^{N'}$, $|f(y) - f(b)| < 2/10^K$.

Combining these inequalities shows that if

$$|y - x| < \min\Big(\frac{|b-a|}{4}, \frac{1}{10^{M'}}, \frac{1}{10^{N'}}\Big) = \delta,$$

we have:

1. If $a+\delta \le x \le b-\delta$, then for all $|y-x| < \delta$, $y \in (a,b)$, so $|f(x)-f(y)| < 2/10^K < \epsilon$ by (8).
2. If $a < x < a+\delta$, then for those $y \in (a, x+\delta)$, $|f(x)-f(y)| < 2/10^K < \epsilon$ by (8). And for those $y \in (x-\delta, a]$, we have by (9)
$$|f(y) - f(x)| < |f(y) - f(a)| + |f(a) - f(x)| < \frac{2}{10^K} + \frac{2}{10^K} < \epsilon.$$
3. A similar argument holds when $b - \delta < x < b$.

## 4.15. A real application

The routine that multiplies integers of arbitrary size in the Gambit implementation of Scheme [5] relies on computing Fast Fourier Transforms of vectors of complex floating-point numbers, and is based on a result of Colin Percival [12].

[15]This is because $a$ is an integer over a power of 10.

Such a routine needs a table of complex floating-point approximations to

$$e^{i\theta} = \cos(\theta) + i\sin(\theta), \quad \theta = \frac{j\pi}{2^N},\ j = 0, \dots, 2^{N-1},$$

for a positive integer $N$. Percival's paper requires very specific accuracy of the entries of this table.

I used my computable real routines both to build this table and to determine its accuracy.[16] I won't give details here, but I needed two things: (a) a routine to correctly round computable reals to floating-point numbers (which is, in general, noncomputable, but I was careful to call it only on irrational computable real arguments for which it is guaranteed to work) and (b) computable versions of the complex exponentials

$$e^{\pi i/2^k} = \cos\left(\frac{\pi}{2^k}\right) + i\sin\left(\frac{\pi}{2^k}\right), \quad k = 1, \dots, N,$$

for some integer $N$. When $k = 1$, $e^{\pi i/2^1} = i$, and subsequent values are computed using square roots; for example, for $k = 2$,

$$e^{\pi i/2^2} = \sqrt{e^{\pi i/2^1}} = \frac{\sqrt{2}}{2} + i\frac{\sqrt{2}}{2}, \text{ etc.}$$

If $a$ and $b$ are computable real numbers, we can find computable $x$ and $y$ such that $a + ib = (x + iy)^2$ as follows. We have

$$a + ib = (x + iy)^2 = x^2 - y^2 + i(2xy),$$

so $a = x^2 - y^2$ and $b = 2xy$, or $y = b/(2x)$. Using this expression for $y$, we get

$$a = x^2 - \frac{b^2}{4x^2} \text{ or } 4x^4 - 4ax^2 - b^2 = 0.$$

This is a quadratic in $x^2$, so the quadratic formula gives

$$x^2 = \frac{4a \pm \sqrt{16a^2 + 16b^2}}{8} = \frac{a}{2} \pm \frac{1}{2}\sqrt{a^2 + b^2}.$$

We choose the $+$ of $\pm$ because $x^2$ is nonnegative, and then another square root gives us $x$. Then we set $y = b/(2x)$ (when $x \neq 0$).

Beyond the operations that have already been given, we need only figure out how to multiply and divide computable real numbers; suitable algorithms can be found in Section 4.5 of [7].

[16]This can also easily be achieved with existing software libraries, e.g., MPFR [6].

## 5. Postscript

I won't argue that the current approach of teaching Real Analysis for math majors in university be replaced with the computable real approach — I don't know enough about the computable reals to make that argument. The computable approach leads to conclusions that students might generally find difficult; for example, the function $f(x) = 0$ for $x < 0$ and $f(x) = 1$ for $x \geq 0$ is not a computable function (because of the comparisons).

As I hope I've made clear, however, studying the computable reals had quite a number of benefits to me personally.

First, it gave me added perspective on the statement attributed to Leopold Kronecker that "God made the integers, all else is the work of man"—all operations here are on integers, with the rational numbers and the computable reals being built on top of such things. It gave me a better understanding of Turing's ideas. It integrated my evolving sense of number as a child with my professional understanding as an adult.

Second, it drove home to me how beautiful mathematical theories can be built by noncomputable or nonconstructive approaches, but with perhaps little practical significance. It seems in some cases to lead to almost a personal belief (a religious faith?) in things that cannot be constructed: that all vector spaces have bases, that every bounded linear functional on a closed subspace of a Banach space can be extended to the whole space, that some sets are not Lebesgue measurable. Beautiful (and sometimes strange) results—many with constructive counterparts [1]—that range from curiosities to the bases of even more grandiose and beautiful theories.

Finally, the computable real approach, with guaranteed accuracy of final results, helped me as a professional mathematician find accurate solutions to highly sensitive problems. Accurate final answers are definitely *not* guaranteed by the popular approach of fixing ahead of time the precision of intermediate results to 50 or 100 digits, say, and then hoping that this will guarantee 15 digits accuracy in the final result.

I *would* argue that students should be exposed to these ideas in real analysis class, in the same way that a digital electronics design class would have a laboratory component.

**Acknowledgments.** I thank Kyle Dahlin for carefully reading an early draft of this document, and Leonard Lipshitz, with whom I've discussed these issues over many years.

## References

[1] Errett Bishop and Douglas Bridges. *Constructive Analysis*, volume 279 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 1985.

[2] Vasco Brattka, Peter Hertling, and Klaus Weihrauch. A tutorial on computable analysis. In S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *New Computational Paradigms: Changing Conceptions of What Is Computable*, pages 425–491. Springer New York, New York, NY, 2008.

[3] Roddy Doyle. *The Dead Republic*. Jonathan Cape, London, 2010.

[4] Louise Erdrich. *The Night Watchman*. Harper, New York, NY, 2020.

[5] Marc Feeley. Gambit Scheme. `https://github.com/gambit/gambit`. Accessed 2022/01/09.

[6] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software*, 33(2), 2007. Article 13, 15 pages.

[7] John Robert Harrison. *Theorem Proving with the Real Numbers*. CPHC/BCS Distinguished Dissertations. Springer-Verlag London, Ltd., London, 1998.

[8] Brian W. Kernighan and Dennis W. Ritchie. *The C Programming Language*. Prentice Hall, Englewood Cliffs NJ, 1978.

[9] Colin McMillen. 93% of paint splatters are valid Perl programs. `https://www.famicol.in/sigbovik/`. Accessed 2022/01/09.

[10] Valérie Ménissier-Morain. Arbitrary precision real arithmetic: design and algorithms. *The Journal of Logic and Algebraic Programming*, 64(1):13–39, 2005.

[11] Carl Douglas Olds. *Continued Fractions*. Number 9 in the Anneli Lax New Mathematical Library. Mathematical Association of America, Washington, D.C., 1963.

[12] Colin Percival. Rapid multiplication modulo the sum and difference of highly composite numbers. *Mathematics of Computation*, 72(241):387–395, 2003.

[13] James Propp. Real analysis in reverse. *American Mathematical Monthly*, 120(5):392–408, 2013.

[14] Tibees. How to find the square root of a decimal number without a calculator, 2018. `https://www.youtube.com/watch?v=nAZvUnWbS8c`. Accessed 2022/01/09.

[15] Alan Mathison Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society. Second Series*, 42(3):230–265, 1936.

[16] Elias Zakon. *Basic Concepts of Mathematics*. The Trillia Group, West Lafayette, Indiana, 2001. `http://www.trillia.com/zakon1.html`. Accessed 2022/01/09.

## Appendix: Some routines for the computable reals

We use the programming language Scheme, and we *identify* the computable number $x$ with its procedure `x`. We'll not explain the language here, except to say that (a) all operations use *prefix notation*, the operator goes first, followed by the arguments, so if `x` is a procedure for a computable real $x$, the programming notation for $x(k)$ is `(x k)`, (b) `lambda` is notation to return a procedure, so the definition of the procedure `x` taking an argument `k` might begin `(lambda (k) ...)`, and (c) `exact-integer-sqrt`, when given a nonnegative integer $x$, returns two nonnegative integers $s$ and $r$ with $s^2 \leq x < (s+1)^2$ and $x = s^2 + r$. The names we give functions that operate on computable real numbers begin with `c-`. Comments begin with a semicolon. The procedures run under Gambit Scheme [5].
We first define the procedures round and isqrt from Section 4.5. Next we write a routine to convert rational numbers to computable numbers, from Section 4.6. Addition, negation, subtraction, and dividing by an integer are then defined as in Section 4.7. Our little library is completed with the routine for square root from Section 4.9.
Finally, we compute the example in Section 4.10, showing that $x(k)$ is a relation, not a function.

```
;;; Round a rational number to an integer
(define (round x)
  (floor (+ x 1/2)))
;;; Compute the integer square root of the nonnegative integer x
(define (isqrt x)
  (call-with-values
      (lambda ()
        (exact-integer-sqrt x))
    (lambda (s r)
      (if (< s r) (+ s 1) s))))

;;; Convert the rational number r to a computable number
(define (r->c r)
  (lambda (k) (round (* r (expt 10 k)))))
;;; Add x and y
(define (c-+ x y)
  (lambda (k) (round (/ (+ (x (+ k 1)) (y (+ k 1))) 10))))
;;; Negation and subtraction
(define (c-- x #!optional (y #f))
  (if (not y)
      (lambda (k) (- (x k)))   ;; Negate x
      (c-+ x (c-- y))))        ;; Add x to negative y
;;; Divide x by the nonzero integer N
(define (c-/-by-N x N)
  (lambda (k) (round (/ (x k) N))))
;;; Compute the square root of x
(define (c-sqrt x)
  (lambda (k) (isqrt (x (* 2 k)))))

;;; The example that illustrates that computable reals
;;; determine relations, not functions.
(define (test N K)
  (let ((x-bar (c-/-by-N (c-sqrt (r->c N)) N))
        (x-hat (c-sqrt (r->c (/ 1 N)))))
    (do ((k 0 (+ k 1)))
        ((= k K))
      (let ((x-bar_k (x-bar k))
            (x-hat_k (x-hat k)))
        (if (not (= x-bar_k x-hat_k))
            (pretty-print (list 'N= N 'k= k
                                'x-bar_k= x-bar_k 'x-hat_k= x-hat_k
                                'x-bar-x-hat_k= ((c-- x-bar x-hat) k))))))))
(test 2 15)
;;; The output is
;;; (N= 2 k= 13 x-bar_k= 7071067811866 x-hat_k= 7071067811865 x-bar-x-hat_k= 0)
```