Computable analysis via fast converging Cauchy sequences

Chris Wong

September 12, 2016

Preface

In classical mathematics, one often proves a result without constructing the object in question. The archetypal *non-constructive* proof would be the result by Jarden (1953):

Theorem 0.0.1. *A power of an irrational number to an irrational exponent may be rational.*

Proof. $\sqrt{2}^{\sqrt{2}}$ is either rational or irrational. If it is rational, our statement is proved. If it is irrational, $(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = (\sqrt{2})^2 = 2$ proves our statement. $\square$

What makes this non-constructive is the claim that we can decide whether or not any number is rational. While the proof handles both cases, it does not show which of these is correct.[1] The crux of the issue is the *law of excluded middle*, which states that $A \vee \neg A$ for *any* proposition A. Constructive mathematicians reject this law in general, as it yields no information on the construction of either A or $\neg A$.

(Note that this law may still hold in specific cases. For example, given any natural number n we can still determine which of $n = 1$ or $n \neq 1$ is true. A predicate which has this property is called *decidable*.)

As with many areas of mathematics, the theory preceded its application. Constructive proofs can be seen as computer programs, a fact known as the Curry–Howard correspondence (Wadler, 2015). In particular, the study of *constructive analysis*—real numbers, convergence and continuity in a constructive setting—can often lead to novel, proven-correct numerical algorithms. It is this relationship between constructive analysis and computation that will be explored in this thesis.

Past treatments, such as the aptly named *Constructive Analysis* (Bishop and Bridges, 1985), have often favored a more theoretical approach; using a definition of the reals that is mathematically simple but computationally inefficient. The typical definition of Cauchy sequences includes a *modulus function* μ, which maps an error threshold ε to an index $n := \mu(\varepsilon)$. A naïve implementation of Cauchy sequences would then comprise two steps: compute the index n, then compute the element x_n at said index. But as we will see later, the overhead of separating these steps can eclipse the running time of the steps themselves.

We propose an alternative system based on *fast converging Cauchy sequences* (FCCS), which combines the modulus and indexing functions into a single operation. Furthermore, since a FCCS converges exponentially to its limit point, to compute another digit of precision only requires a handful of extra operations. The theory of FCCS then becomes a practical, efficient implementation of exact real arithmetic.

In Chapter 1 we discuss different representations of real numbers, and the advantages and disadvantages thereof. In Chapter 2 we introduce the definition of FCCS, along with associated operations and proofs. Chapter 3 evaluates the performance of this system, and justifies the requirement of fast convergence. Finally, the Appendix comprises a sample implementation of FCCS written in the Haskell programming language.

[1] It can be shown that $\sqrt{2}^{\sqrt{2}}$ is indeed irrational by the Gelfond–Schneider theorem; this can then be used to complete the original proof, provided there exists a constructive proof of Gelfond–Schneider as well.

0.1 Computability

A number is *computable* when it can be approximated to arbitrary precision by a (finite, terminating) computer program. The program itself can be written as either a Turing machine, a lambda calculus expression, or in any other equivalent model of computation. A function $\mathbb{R} \to \mathbb{R}$ is computable when any finite prefix of its result can be computed from a finite prefix of its argument. For the purposes of this project, we will consider all real numbers to be computable unless stated otherwise.

Chapter 1

Representations of real numbers

There are many possible ways to represent real numbers in a computational setting. While this thesis will primarily cover Cauchy sequences, other common representations are worth mentioning as well.

1.1 Decimal expansion

One naive approach is to represent a real number by a sequence of digits in some base b. This base can be any integer greater than 1, but is usually set to 2 or 10. In the latter case, this is our usual *decimal expansion*.

Formally, a non-negative real number $r = a_0.a_1a_2a_3\ldots$ is defined as the sum

$$r = \sum_{k=0}^{\infty} \frac{a_k}{b^k},$$

where a_0 forms the integer part, and $0 \leq a_1, a_2, \ldots < b$ form the fractional part. If the expansion ends in zeroes, then it is said to be *finite*.

While this representation is easy to understand, it has drawbacks that make it hard to use in practice. First, it is not unique: the decimal expansions $0.999\ldots$ and $1.000\ldots$ are two different representations of 1. We can preserve uniqueness by removing sequences that end in nines, but that leads to issues elsewhere. For example, suppose $x = 0.090909\ldots$ and $y = 0.909090\ldots$ are real numbers. While we could be tempted to conclude that $x + y = 1$, this may not be the case. It may turn out that after a million digits, x contains a '0' instead of a '9', resulting in a sum that is less than 1. In fact, it would take an infinite number of steps to be sure whether the leading digit of $x + y$ is '1' or '0'.

This problem touches on a fundamental weakness of the decimal representation. Since computing more digits can only *increase* the approximation, not decrease it, each term then forms a lower bound to the true value. In other words, if we know that the number x starts with the digit 1, then we also know that $x \geq 1$. But as the addition example shows, it is undecidable in general whether or not a real number satisfies such a bound. For this reason, decimal expansions are not suited to computable analysis.

1.2 Cauchy sequences

An alternative formulation is *Cauchy sequences*, a sequence of rational numbers which become "arbitrarily close" to each other as the sequence progresses. In the classical definition, a sequence is Cauchy when for any bound $\varepsilon > 0$, there is an index N after which any pair of points differ by less than ε.

For the purposes of computation, we also require that the sequence is *fast converging*; that is, the sequence converges exponentially to its limit point. Formally, we require that there is some fixed *modulus function* μ such that for any natural numbers m and n, $\left|x_{\mu(m)} - x_{\mu(m)+n}\right| < 1/2^m$. This restriction lets us find a suitable approximation in reasonable time.

Other formulations also require the denominator of each element to be a successive power of 2. This restriction simplifies the implementation, at the cost of further complicating the theory. The system designed by Gowland and Lester (2000) uses this technique. To avoid duplicating their work, we do not apply this idea in our implementation.

1.3 Continued fractions

A *(simple) continued fraction* is a possibly-infinite sequence of integers $[a_0; a_1, a_2, \ldots, a_n]$, where $a_1, a_2, \ldots$ are positive, which represents the real number

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_n}}}}\,.$$

This representation has many useful properties. It is unique (Prime.mover, 2013); it is rational if and only if the sequence is finite; the sequences for e, $\sqrt{n}$ for any integer n, and the golden ratio ϕ are easy to compute. Compared to Cauchy sequences it has two drawbacks: the underlying theory is not as easy to understand; and its implementations are slower in practice.

Chapter 2

Fast converging Cauchy sequences

For the remainder of this thesis, we will discuss *fast converging Cauchy sequences* (FCCS). These differ from the usual definition in that they must converge exponentially to their limit point. Despite this restriction, FCCS are equivalent to Cauchy sequences—at least in a mathematical sense—and hence form a model for the (computable) real numbers.

Definition 2.0.1 (Simpson (2009))**.** A fast converging Cauchy sequence (FCCS) $x : \mathbb{N} \to \mathbb{Q}$ has the property that

$$\forall m, n \in \mathbb{N} \quad |x_m - x_{m+n}| < \frac{1}{2^m} \ .$$

Proposition 2.0.2. *Every fast converging Cauchy sequence is a Cauchy sequence, and vice versa.*

Proof. ($\Rightarrow$) Let x be a FCCS, $\varepsilon > 0$ arbitrary. Let m be the least natural number such that $1/2^m \leq \varepsilon$. Then for all $n \in \mathbb{N}$, $|x_m - x_{m+n}| < 1/2^m \leq \varepsilon$.

($\Leftarrow$) Let x be a Cauchy sequence. Define $y_m := x_{N(1/2^m)}$ where $N(\varepsilon)$ is the index where all subsequent points differ by less than ε. Then y is a FCCS. □

2.1 Arithmetic and closure properties

With Cauchy sequences, we define arithmetic operations simply by working pointwise on each element. But with FCCS, we need to do extra work to preserve the fast convergence property. Take addition, for example: when two approximations are added together, their errors are added as well. In general, this means that the error of the result is *twice* that of the inputs. Hence the addition routine must shift the inputs one step to the right, halving the error to compensate.

Proposition 2.1.1 (Addition)**.** *Let x, y be* FCCS.
The sum $x + y$, defined by $(x+y)_k := x_{k+1} + y_{k+1}$, is a FCCS.

Proof. Let $m, n \in \mathbb{N}$ be arbitrary.
Then

$$\begin{aligned} |(x+y)_m - (x+y)_{m+n}| &= |(x_{m+1} + y_{m+1}) - (x_{m+n+1} + y_{m+n+1})| \\ &= |x_{m+1} - x_{m+n+1} + y_{m+1} - y_{m+n+1}| \end{aligned}$$

$$\begin{aligned}&\le |x_{m+1} - x_{m+n+1}| + |y_{m+1} - y_{m+n+1}| \\ &< \frac{1}{2^{m+1}} + \frac{1}{2^{m+1}} \\ &= \frac{1}{2^m} \ .\end{aligned}$$ □

Proposition 2.1.2 (Negation)**.** *If x is a* FCCS*, then $(-x)_k := -(x_k)$ is a* FCCS.

Proof. Let $m, n \in \mathbb{N}$ be arbitrary.

Then
$$|(-x)_m - (-x)_{m+n}| = |-x_m - (-x_{m+n})| = |x_m - x_{m+n}| < \frac{1}{2^m} \ .$$ □

Corollary 2.1.3 (Subtraction)**.** $(x-y)_k := (x+(-y))_k = x_{k+1} - y_{k+1}$ *is a* FCCS.

Multiplication is more subtle. Unlike with addition, the shift under multiplication is not constant. A larger input would necessitate a larger shift, since the multiplication operation scales the error up by a corresponding amount.

For example, consider the sequences $x := (1, \frac{1}{2}, \frac{1}{4}, \ldots)$ and $y := (16, 16, \ldots)$. It can be shown that x and y are FCCS; but if we multiply them pointwise, the result $(16, 8, 4, \ldots)$ is not fast converging. To recover fast convergence, this sequence must be shifted not once, but $\log_2 16 = 4$ times.

Proposition 2.1.4 (Multiplication)**.** $(xy)_k := x_{a(k)} \cdot y_{b(k)}$ *is a* FCCS*, where*

$$\begin{aligned}a(k) &= k + 1 + \phi(y_0) \\ b(k) &= k + 1 + \phi(x_0) \\ \phi(z) &= \max\{0, \lceil \log_2(|z|+1) \rceil\} \ .\end{aligned}$$

Proof. Given $m, n \in \mathbb{N}$, we need to choose $a, b : \mathbb{N} \to \mathbb{N}$ such that

$$\begin{aligned}|(xy)_m - (xy)_{m+n}| &= \left|x_{a(m)}y_{b(m)} - x_{a(m+n)}y_{b(m+n)}\right| \\ &= \left|x_{a(m)}y_{b(m)} - x_{a(m)}y_{b(m+n)} + x_{a(m)}y_{b(m+n)} - x_{a(m+n)}y_{b(m+n)}\right| \\ &\le \left|x_{a(m)}y_{b(m)} - x_{a(m)}y_{b(m+n)}\right| + \left|x_{a(m)}y_{b(m+n)} - x_{a(m+n)}y_{b(m+n)}\right| \\ &= \left|x_{a(m)}\right|\left|y_{b(m)} - y_{b(m+n)}\right| + \left|y_{b(m+n)}\right|\left|x_{a(m)} - x_{a(m+n)}\right| \\ &< \frac{1}{2^m} \ .\end{aligned}$$

For this inequality to hold, it is sufficient to show that

$$\left|x_{a(m)}\right|\left|y_{b(m)} - y_{b(m+n)}\right| < \frac{1}{2^{m+1}}$$

and

$$\left|y_{b(m+n)}\right|\left|x_{a(m)} - x_{a(m+n)}\right| < \frac{1}{2^{m+1}} \ .$$

To complete the proof, we observe that $\left|x_{a(m)}\right| < |x_0| + 1$ and $\phi(x_0) \ge \log_2(|x_0|+1)$. Using these facts:

$$\left|x_{a(m)}\right|\left|y_{b(m)} - y_{b(m+n)}\right| < (|x_0|+1) \cdot \frac{1}{2^{b(m)}}$$

$$
\begin{aligned}
&= \frac{|x_0|+1}{2^{m+1+\phi(x_0)}} \\
&\leq \frac{|x_0|+1}{2^{m+1+\log_2(|x_0|+1)}} \\
&= \frac{|x_0|+1}{2^{m+1}\cdot(|x_0|+1)} \\
&= \frac{1}{2^{m+1}} \ .
\end{aligned}
$$

Similarly,

$$
\begin{aligned}
|y_{a+b}||x_a - x_{a+b}| &< (|y_0|+1)\cdot\frac{1}{2^{m+1+\log_2(|y_0|+1)}} \\
&= \frac{1}{2^{m+1}} \ .
\end{aligned}
$$

□

To derive the final arithmetic operation, division, we must introduce the idea of (in-)equality first.

2.2 Ordering

Constructive analysis presents unique challenges in defining a total order on the reals. Without additional axioms, $a < b$ and $a \geq b$ are not full inverses; we are forced to favor one over the other. Many classical concepts, such as the sign function or equality (!), fall apart in a constructive setting. In this section we will define these concepts in terms of FCCS, and explore the consequences in terms of computability.

Definition 2.2.1 (Equality). A pair of FCCS x and y are equal, written $x = y$, when

$$\forall n \in \mathbb{N} \quad |x_{n+1} - y_{n+1}| \leq \frac{1}{2^n} \ .$$

x is not equal to y, written $x \neq y$, when $\neg(x = y)$.

Note that in the definition of equality, we compare the terms x_{n+1} and y_{n+1}, not x_n and y_n. This shifting can be justified with an example. Suppose x and y are sequences where $x := (0, \frac{1}{8}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \ldots)$ and $y := (1, \frac{7}{8}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \ldots)$. It can be shown that x and y are fast converging. Clearly x and y should be equal. But since $|x_1 - y_1| = 3/4 > 1/2^1$, without the shift we would be forced to deem them unequal. With the shift in place, the modified bound $|x_1 - y_1| \leq 1/2^0$ holds as required.

An important result of constructive analysis is that equality of real numbers is undecidable—that is, there is no algorithm that can determine whether two arbitrary numbers are equal or not. This is a consequence of the Halting Problem (Turing, 1936): since the execution of a computer program can be encoded as a real number, deciding equality would solve the Halting Problem as well.

Proposition 2.2.2. *Equality on* FCCS *is undecidable.*

Proof. Let P be an arbitrary computer program. Define the FCCS x as follows.

Given an arbitrary index n, run the program P for at most n steps. If the program terminates in $p \leq n$ steps, then define $x_n := 1/2^p$. Otherwise, set $x_n := 1/2^n$. In other words, $x \to 1/2^p \neq 0$ if and only if P terminates.

Suppose there is an algorithm for deciding whether $x = 0$. Then we also have an algorithm that decides whether an arbitrary program terminates. Hence the Halting Problem is solved, and we have a contradiction. □

Corollary 2.2.3. *The sign function*

$$\operatorname{sgn}(x) := \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & x > 0 \end{cases}$$

is not computable.

Proof. If the sign function was computable, then we could determine whether x and y are equal by evaluating $\operatorname{sgn}(x-y)$. This contradicts Proposition 2.2.2. □

In particular, constructive analysis forces a distinction between constructive and non-constructive inequality. A statement that two numbers are *not equal* is just a negation of the statement that they are equal. But for two numbers to be *apart*, we must provide an index after which the sequences will never come closer together. While it is easy to show that apartness implies not-equality, the converse requires an additional axiom: Markov's principle.

Definition 2.2.4 (Ordering)**.** We say that $x < y$ when there exists $c \in \mathbb{N}$ such that

$$\forall k \in \mathbb{N} \quad y_{c+k} - x_{c+k} > \frac{1}{2^c} \ .$$

$x \leq y$ is equivalent to $\neg(y < x)$.

Definition 2.2.5 (Apartness)**.** x and y are apart when $x < y$ or $y < x$.

Lemma 2.2.6 (Apartness from zero)**.** *Suppose there exists $n \in \mathbb{N}$ such that $|x_n| > 1/2^n$. Then either $x < 0$ or $x > 0$.*

Proof. If $|x_n| > 1/2^n$, then either $x_n > 1/2^n$ or $-x_n > 1/2^n$. Assume the former case.

Applying the definition of a FCCS, for all $k \in \mathbb{N}$

$$\begin{aligned} |x_n - x_{n+k}| < \frac{1}{2^n} \quad &\Rightarrow \quad x_n - x_{n+k} < \frac{1}{2^n} \\ &\Rightarrow \quad x_n - \frac{1}{2^n} < x_{n+k} \ . \end{aligned}$$

Let $c \geq n$ be the least natural number such that $1/2^c \leq x_n - 1/2^n$. Then

$$x_{c+k} > x_n - \frac{1}{2^n} \geq \frac{1}{2^c}$$

Since k is arbitrary, this shows that $x > 0$. With a similar argument, it can be shown that the other case implies $x < 0$. □

Definition 2.2.7 (Markov's principle)**.** Suppose $x : \mathbb{N} \to X$ is a sequence, and P is a decidable predicate on X (that is, as noted in the preface, $P(a) \vee \neg P(a)$ for any $a \in X$). Then

$$\neg \forall n\, P(x_n) \ \Rightarrow \ \exists n\, \neg P(x_n) \ .$$

Proposition 2.2.8 (Apartness)**.** *If $x \neq y$, then by Markov's principle x is apart from y.*

Proof. Let $z := x - y$. Applying Markov's principle,

$$\neg\left(\forall n \in \mathbb{N} \quad |x_{n+1} - y_{n+1}| \le \frac{1}{2^n}\right) \quad \Rightarrow \quad \exists n \in \mathbb{N} \quad |x_{n+1} - y_{n+1}| > \frac{1}{2^n} \ .$$

Hence $|z_n| = |x_{n+1} - y_{n+1}| > 1/2^n$, and by Lemma 2.2.6, either $z < 0$ or $z > 0$. From that we conclude $x < y$ or $x > y$ respectively. Therefore x and y are apart. □

Operationally, Markov's principle represents an unbounded search. The operations described thus far have all been bounded. Even in the case of multiplication, to compute $(x \cdot y)_n$ we only need to check four elements: x_0, y_0, $x_{a(n)}$, and $y_{b(n)}$. While the indices a and b can be large, they are determined solely by the initial approximations x_0 and y_0. But when finding evidence of apartness, we must inspect each element of the sequence until this evidence is found. There is no guarantee as to how many elements we need to inspect, other than that it will be found eventually. This unboundedness is what justifies the use of Markov's principle.

For example, consider the sequence x whose first N terms are zero, with the rest defined as $1/2^N$. Then x is not equal to zero, but it would take N steps to show that it is apart. Furthermore, if the input were in fact equal, then the algorithm would search forever for evidence that does not exist. Since equality of real numbers is undecidable, there cannot be an algorithm that has a better worst case.

Finally, we note that though the law of trichotomy is undecidable, the related notions of absolute value and minimum/maximum are not.

Proposition 2.2.9 (Absolute value)**.** *If x is a* FCCS*, then $|x|_n := |x_n|$ is a* FCCS.

Proof. For any $m, n \in \mathbb{N}$,

$$\begin{aligned} \left||x|_m - |x|_{m+n}\right| &= \left||x_m| - |x_{m+n}|\right| \\ &\le |x_m - x_{m+n}| \\ &< \frac{1}{2^m} \end{aligned}$$

by the reverse triangle inequality. □

Lemma 2.2.10 (Maximum as absolute value)**.** *If x and y are rational numbers, then*

$$\max(x,y) = \frac{x + y + |x-y|}{2} \ .$$

Proof. If $x \ge y$, then $|x - y| = x - y$. Hence

$$\frac{x + y + (x-y)}{2} = x = \max(x,y) \ .$$

On the contrary, if $x < y$ then $|x-y| = -(x-y)$. Hence

$$\frac{x + y - (x-y)}{2} = y = \max(x,y) \ .$$ □

Proposition 2.2.11 (Maximum and minimum)**.** *Let x and y be* FCCS*. Then the maximum, defined by* $\max(x,y)_n := \max(x_{n+1}, y_{n+1})$*, is a* FCCS*. Furthermore, the minimum* $\min(x,y) := -\max(-x,-y)$ *is also a* FCCS.

Proof. Let $m, n \in \mathbb{N}$ be arbitrary.

Then by Lemma 2.2.10,

$$\begin{aligned}
|\max(x,y)_m - \max(x,y)_{m+n}| &= |\max(x_{m+1}, y_{m+1}) - \max(x_{m+n+1}, y_{m+n+1})| \\
&= \frac{1}{2}\big|x_{m+1} + y_{m+1} + |x_{m+1} - y_{m+1}| - x_{m+n+1} - y_{m+n+1} - |x_{m+n+1} - y_{m+n+1}|\big| \\
&\leq \frac{1}{2}\left(|x_{m+1} - x_{m+n+1}| + |y_{m+1} - y_{m+n+1}| + |x_{m+1} - x_{m+n+1}| + |y_{m+1} - y_{m+n+1}|\right) \\
&< \frac{1}{2}\left(\frac{1}{2^{m+1}} + \frac{1}{2^{m+1}} + \frac{1}{2^{m+1}} + \frac{1}{2^{m+1}}\right) \\
&= \frac{1}{2^m}\ .
\end{aligned}$$

The fast convergence of $\min(x,y)$ then follows from Proposition 2.1.2. □

2.3 Division

To derive the reciprocal, and with it division, observe that x^{-1} is defined only when x is not zero. Hence intuitively, some constructive evidence that a number is not zero should assist in computing its reciprocal. Indeed this is this the case: the lower bound provided by apartness translates to an upper bound on the result.

Proposition 2.3.1 (Reciprocal)**.** *Let x be a* FCCS *where $x \neq 0$. Then $\left(x^{-1}\right)_k := (x_{k+2c})^{-1}$ is a* FCCS*, where c is defined as in Lemma 2.2.6.*

Proof. By the definition of a FCCS, we know that

$$|x_{m+2c} - x_{m+n+2c}| < \frac{1}{2^{m+2c}}\ ;$$

And by Lemma 2.2.6, we know that

$$\begin{aligned}
|x_{m+2c}| > \frac{1}{2^c} \ \wedge\ |x_{m+n+2c}| > \frac{1}{2^c} &\ \Rightarrow\ |x_{m+2c}||x_{m+n+2c}| > \frac{1}{2^{2c}} \\
&\ \Rightarrow\ \frac{1}{|x_{m+2c}||x_{m+n+2c}|} < 2^{2c}\ .
\end{aligned}$$

Therefore

$$\begin{aligned}
\left|(x_{m+2c})^{-1} - (x_{m+n+2c})^{-1}\right| &= \left|\frac{x_{m+2c} - x_{m+n+2c}}{x_{m+2c} \cdot x_{m+n+2c}}\right| \\
&= \frac{|x_{m+2c} - x_{m+n+2c}|}{|x_{m+2c}||x_{m+n+2c}|} \\
&< \frac{1}{2^{m+2c}} \cdot 2^{2c} \\
&= \frac{1}{2^m}\ .
\end{aligned}$$

□

Corollary 2.3.2 (Division)**.** $(x/y)_k := \left(x \cdot y^{-1}\right)_k$ *is a* FCCS.

2.4 Extensionality

Not all operations on FCCS apply to the computable reals that they represent. For example, consider the indexing operator $\iota_0 : \text{FCCS} \to \mathbb{Q}$, $\iota_0(x) := x_0$. While this is a valid function on FCCS, it is not clear how this can be seen as a function on $\mathbb{R}$. After all, for any real number there are many different FCCS that can represent it, all of which could yield different answers under ι_0.

The problem here is that ι_0 is not *well-defined* as a function on real numbers; in other words, it is not *extensional* as a function of FCCS. A function is extensional when it maps equal inputs to equal outputs, regardless of how the inputs are represented underneath. In the context of FCCS, this means that the function only concerns itself with what a sequence converges to, not the approximations along the way. Now it is easy to see how ι_0 is not extensional: if we let $x_n := 1/2^n$ and $y_n := -1/2^n$, then $x = y = 0$ but $\iota_0(x) \neq \iota_0(y)$. Intuitively, ι_0 breaks the abstraction of a "real number," exposing the Cauchy sequence representation underneath.

Most operations are extensional, including the arithmetic operations defined above. We will prove that addition is extensional below; the other operations can be verified using a similar argument. For brevity their extensionality will be assumed without proof.

Definition 2.4.1 (Extensionality)**.** A function $f : \text{FCCS} \to \text{FCCS}$ is extensional when $x = y$ implies $f(x) = f(y)$.

Proposition 2.4.2. *Addition is extensional.*

Proof. Let x, y, z be arbitrary FCCS where $x = y$.

Then for any $n \in \mathbb{N}$,

$$\begin{aligned} |(x+z)_n - (y+z)_n| &= |(x_{n+1} + z_{n+1}) - (y_{n+1} + z_{n+1})| \\ &= |x_{n+1} - y_{n+1}| \\ &\leq \frac{1}{2^{n+1}} \leq \frac{1}{2^n} \end{aligned}$$

Hence $x + z = y + z$ and addition is extensional in its first parameter.

The dual $z + x = z + y$ follows from commutativity. Hence addition is extensional. □

2.5 Exponentials and trigonometric functions

With elementary arithmetic under our belt, we can turn our attention to more advanced functions such as the exponential map $\exp(x)$.

Recall that the exponential can be defined as

$$\exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$$

In particular, this definition is useful because when x is small, each term is exponentially smaller than the one before it. Hence the sequence of partial sums can be easily adapted to a FCCS.

Proposition 2.5.1 (Exponential on $|x| \leq 1$)**.** *Let x be a* rational *number where $-1 \leq x \leq 1$. Then the exponential* $\exp_{[-1,1]}(x)$*, defined by*

$$\left[\exp_{[-1,1]}(x)\right]_n := \sum_{k=0}^{n+1} \frac{x^k}{k!},$$

is a FCCS.

Proof. Let $m, n \in \mathbb{N}$ be arbitrary.

It can be shown that $k! \leq 2^{k-1}$ for any $k \geq 1$.

Then

$$\begin{aligned}\left|\exp_{[-1,1]}(x)_m - \exp_{[-1,1]}(x)_{m+n}\right| &= \left|\sum_{k=0}^{m+1} \frac{x^k}{k!} - \sum_{k=0}^{m+n+1} \frac{x^k}{k!}\right| \\ &= \sum_{k=m+2}^{m+n+1} \frac{\left|x^k\right|}{k!} \\ &\leq \sum_{k=m+2}^{m+n+1} \frac{1}{k!} \\ &\leq \sum_{k=m+2}^{m+n+1} \frac{1}{2^{k-1}} \\ &= \sum_{k=m+1}^{m+n} \frac{1}{2^k} \\ &< \frac{1}{2^m}\ .\end{aligned}$$

□

We can then extend this definition to all rational numbers using the property $\exp(x+y) = \exp(x)\exp(y)$.

Corollary 2.5.2 (Exponential on $\mathbb{Q}$)**.** *Let x be a rational number. Then the exponential* $\exp(x)$*, defined by*

$$\exp(x) := \begin{cases} \exp_{[-1,1]}(x) & -1 \leq x \leq 1 \\ \exp(x/2)^2 & \textit{otherwise} \end{cases}$$

is a FCCS.

Proof. Since x is rational, it is decidable whether it lies in the interval $[-1,1]$. Moreover, by the Archimedean property $\exp(x)$ is guaranteed to terminate. The fast convergence of $\exp(x)$ then follows. □

The trigonometric functions sin, cos, and tan can be defined in a similar way.

Proposition 2.5.3 (Trigonometric functions)**.** *Let x be a rational number such that* $-1 \leq x \leq 1$*. Then the functions* $\cos_{[-1,1]}(x)$ *and* $\sin_{[-1,1]}(x)$*, defined by*

$$\left[\cos(x)_{[-1,1]}\right]_n := \sum_{k=0}^{n+1} \begin{cases} (-1)^{\lfloor k/2 \rfloor} \cdot \dfrac{x^k}{k!} & k \text{ even} \\ 0 & \text{otherwise} \end{cases}$$

and

$$\left[\sin(x)_{[-1,1]}\right]_n := \sum_{k=0}^{n+1} \begin{cases} (-1)^{\lfloor k/2 \rfloor} \cdot \dfrac{x^k}{k!} & k \text{ odd} \\ 0 & \text{otherwise} \end{cases}$$

are FCCS*. Using the double angle formulae* $\cos(2x) = 2\cos^2 x - 1$ *and* $\sin(2x) = 2\sin x \cos x$*, these functions can be extended to all of* $\mathbb{Q}$*.*

$\tan(x)$ *is then defined as* $\sin(x)/\cos(x)$*.*

Proof. Observe that in the definitions of $\cos_{[-1,1]}$ and $\sin_{[-1,1]}$, the magnitude of each term is less than or equal to that of $\exp_{[-1,1]}$. Therefore we can use a similar argument to prove the convergence of the former. □

2.6 Cantor's diagonal argument

One important result in set theory is the uncountability of the real line: that is, there is no bijective mapping that relates $\mathbb{N}$ and $\mathbb{R}$. This result applies within constructive analysis as well, if we restrict the theorem to the computable reals (or FCCS) instead.

A typical proof of this result uses Cantor's *diagonal argument*. Given any sequence of (computable) real numbers, we can construct a number that is apart from every number in the list. Hence there cannot be an enumeration of all real numbers—if there were, then the diagonal argument would yield one that it missed. The computable reals are then an example of a set that is *countable* but not *enumerable*: every program can be written as a natural number, but there is no algorithm that can list every terminating program.

Furthermore, since this argument is constructive, it can be implemented as a computer program as well. The Haskell implementation includes an enumeration of the rationals, which on application of this argument results in an arbitrary irrational number.

Proposition 2.6.1 (Cantor's diagonal argument)**.** *Let $x : \mathbb{N} \to$ FCCS be a sequence of fast converging Cauchy sequences such that $x = (x^{(0)}, x^{(1)}, \ldots)$. Then there exists a* FCCS $\operatorname{cantor}(x)$ *that is apart from every element in x.*

Proof. We want to define a sequence of rational half-open intervals $[a_n, b_n) : \mathbb{N} \to \mathbb{Q} \times \mathbb{Q}$ such that the following properties hold:

(C1) $|a_n - b_n| \leq 1/4^n$;

(C2) $[a_{n+1}, b_{n+1}) \subset [a_n, b_n)$;

(C3) $x^{(n)} \notin [a_n, b_n)$.

for all $n \in \mathbb{N}$.

Define $a_0 := x_0^{(0)} - 2$, $b_0 := x_0^{(0)} - 1$. For $n > 0$, suppose $[a_n, b_n)$ satisfies the properties above. Then define a_{n+1}, b_{n+1} by the equation

$$[a_{n+1}, b_{n+1}) := \begin{cases} [a_n, a_n + h_n) & x_{2n+2}^{(n)} \geq c_n \\ [b_n - h_n, b_n) & x_{2n+2}^{(n)} < c_n \end{cases}$$

where $c_n := (a_n + b_n)/2$ and $h_n := (b_n - a_n)/4 = 1/4^{n+1} = 1/2^{2n+2}$. Clearly (C1) and (C2) hold, as we shrink the interval by a factor of $1/4$ on each step.

Proof of (C3). Clearly $x^{(0)} \notin [a_0, b_0)$. For the inductive case, if $x_{2n+2}^{(n)} \geq c_n$ then $x^{(n)} \geq c_n - 1/2^{2n+2} = c_n - h_n = a_n + h_n$. Using a similar argument, if $x_{2n+2}^{(n)} < c_n$ then $x^{(n)} < b_n - h_n$. Hence (C3) holds.

Now define $[\operatorname{cantor}(x)]_n := a_{\lfloor n/2 \rfloor}$. Then for any $m, n \in \mathbb{N}$,

$$\begin{aligned} |[\operatorname{cantor}(x)]_m - [\operatorname{cantor}(x)]_{m+n}| &= \left|a_{\lfloor m/2 \rfloor} - a_{\lfloor (m+n)/2 \rfloor}\right| \\ &< \frac{1}{4^{\lfloor m/2 \rfloor}} \\ &\leq \frac{1}{2^m} \end{aligned}$$

shows that $\operatorname{cantor}(x)$ is a FCCS.

By (C3) and Lemma 2.2.6, this is apart from every element of x. $\square$

Chapter 3

Implementation notes

There are many different ways to implement fast converging Cauchy sequences on a computer. One aspect in which an implementation can differ is in how it treats the *modulus function*.

A modulus function $\mu(\varepsilon) : \mathbb{Q} \to \mathbb{N}$ of a sequence returns the index of an element with error at most ε. In the context of FCCS, we can simplify the definition by requiring $\varepsilon = 1/2^n$ for some natural n; then the modulus function can take this n directly.

The FCCS operations described thus far handle the modulus function implicitly—they shift the sequence such that μ is just the identity function. But it is possible to avoid shifting at all, and modify μ instead. In the attached code, the `Cauchy'` type implements this idea.

To decide which approach to use, the author ran benchmarks to compare the performance of each method. The two methods were tasked with evaluating two expressions:

- A "simple" expression, $(e \cdot e + e)/e$, where $e = \exp 1$ is Euler's constant; and
- A "complicated" expression, which is e added to itself 50 times.

Both expressions were evaluated within an error of $1/2^{50}$. The benchmarks were run using the `criterion` library on an Intel Core i7 laptop.

Benchmark	No modulus		With modulus	
	Mean (ms)	Std. dev. (μs)	Mean (ms)	Std. dev. (μs)
Simple	3.41	11.1	3.51	264
Complicated	369	85.0	706	598

It is interesting to note that while there is little difference in the "simple" benchmark, the no-modulus-function method is almost twice as fast in the "complicated" benchmark. This is likely because the modulus function variant involves two function calls instead of one. Over many arithmetic operations, these extra calls can cause a significant slowdown.

Appendix A

Haskell code

A.1 Cauchy.hs

```haskell
{-# OPTIONS_GHC -Wno-name-shadowing #-}

module Cauchy where


import Control.Applicative
import Data.Ratio
import Numeric.Natural


newtype Cauchy = Cauchy (Natural -> Rational)


instance Num Cauchy where
    Cauchy x + Cauchy y = Cauchy (\n -> x (1+n) + y (1+n))
    Cauchy x - Cauchy y = Cauchy (\n -> x (1+n) - y (1+n))
    Cauchy x * Cauchy y = Cauchy (\n -> x (1+ky+n) * y (1+kx+n))
      where
        kx = log2 (abs (x 0) + 1)
        ky = log2 (abs (y 0) + 1)
    negate (Cauchy x) = Cauchy (negate . x)
    abs (Cauchy x) = Cauchy (abs . x)
    signum = error "signum is undecidable"
    fromInteger n = Cauchy (\_ -> fromInteger n)


instance Fractional Cauchy where
    fromRational x = fromInteger (numerator x) / fromInteger (denominator x)
    recip x'@(Cauchy x) = Cauchy (\n -> recip $ x (n + 2*c))
```

```haskell
      where
        c = findNonZero x'


instance Show Cauchy where
    showsPrec p x = showsPrec p (toDouble $ x # 64) . ("..." ++)


-- Cauchy sequence with a modulus of convergence.
--
-- The second argument is a /modulus function/, μ : ℕ → ℕ, which
-- for any k gives the least index where the difference is less than
-- 1/2^μ(k).
data Cauchy' = Cauchy' (Natural -> Rational) (Natural -> Natural)


instance Num Cauchy' where
    Cauchy' x u + Cauchy' y v = Cauchy'
        (liftA2 (+) x y)
        (\r -> max (u (1+r)) (v (1+r)))
    Cauchy' x u - Cauchy' y v = Cauchy'
        (liftA2 (-) x y)
        (\r -> max (u (1+r)) (v (1+r)))
    Cauchy' x u * Cauchy' y v = Cauchy'
        (liftA2 (*) x y)
        (\r -> max (u (1+ky+r)) (v (1+kx+r)))
      where
        kx = log2 (abs (x 0) + 1)
        ky = log2 (abs (y 0) + 1)
    negate (Cauchy' x u) = Cauchy' (negate . x) u
    abs (Cauchy' x u) = Cauchy' (abs . x) u
    signum = error "signum is undecidable"
    fromInteger n = Cauchy' (\_ -> fromInteger n) (\_ -> 0)


instance Fractional Cauchy' where
    fromRational x = fromInteger (numerator x) / fromInteger (denominator x)
    recip x'@(Cauchy' x u) = Cauchy' (recip . x) (\r -> u (r + 2*c))
      where
        c = findNonZero x'


-- Returns max{0, ⌈log₂ x⌉}, where x is a rational number.
log2 :: Rational -> Natural
log2 x
    | x <= 0 = error "log2: negative argument"
    | otherwise = go x 0
  where
    go y acc
        | y <= 1 = acc
        | otherwise = acc `seq` go (y / 2) (1 + acc)


-- Given an x which does not converge to zero, find the lowest index c where
-- |x_{c+k}| > 1/2^c for every k.
findNonZero :: Index a => a -> Natural
findNonZero x = go 0
  where
    go n
        | e <= 0 = go $! n + 1
        | otherwise = log2 (recip e)
      where
        e = abs (x # n) - (1 % 2^n)


-- Find the exponential of a rational number.
expC :: Rational -> Cauchy
expC x
    | -1 <= x && x <= 1 = exp01 x
    | otherwise = let y = expC (x / 2) in y * y
  where
    exp01 x = Cauchy $ \n -> sum [ x^k / realToFrac (factorial k) | k <- [0 ..
     ↪  n+1] ]


-- Like expC, but returns a Cauchy' instead.
expC' :: Rational -> Cauchy'
expC' x
    | -1 <= x && x <= 1 = exp01 x
    | otherwise = let y = expC' (x / 2) in y * y
  where
    exp01 x = Cauchy'
        (\n -> sum [ x^k / realToFrac (factorial k) | k <- [0 .. n] ])
        (\r -> r+1)


-- Find the sine of a rational number.
sinC :: Rational -> Cauchy
sinC x
    | -1 <= x && x <= 1 = sin01 x
    | otherwise = 2 * sinC (x / 2) * cosC (x / 2)
  where
```

```haskell
    sin01 x = Cauchy $ \n -> sum [
        (-1)^(k `quot` 2) * x^k / realToFrac (factorial k) |
        k <- [1, 3 .. n+1] ]


-- Find the cosine of a rational number.
cosC :: Rational -> Cauchy
cosC x
    | -1 <= x && x <= 1 = cos01 x
    | otherwise = let c = cosC (x / 2) in 2 * c * c - 1
  where
    cos01 x = Cauchy $ \n -> sum [
        (-1)^(k `quot` 2) * x^k / realToFrac (factorial k) |
        k <- [0, 2 .. n+1] ]


-- Find the tangent of a rational number.
tanC :: Rational -> Cauchy
tanC x = sinC x / cosC x


-- Super simple factorial function.
factorial :: Natural -> Natural
factorial n = product [2..n]


-- Given a sequence of numbers, find a number that is apart from all of them.
cantor :: (Natural -> Cauchy) -> Cauchy
cantor xs = Cauchy $ \n -> let (a, _) = step (n `quot` 2) in a
  where
    step :: Natural -> (Rational, Rational)
    step 0 = (x00 - 2, x00 - 1)
      where
        x00 = xs 0 # 0
    step n
        | xn >= c = (a, a + h)
        | otherwise = (b - h, b)
      where
        (a, b) = step (n - 1)
        c = (a + b) / 2
        h = (b - a) / 4
        xn = xs n # 2 * n + 2


class Index a where
    (#) :: a -> Natural -> Rational
```

```haskell
infixl 1 #

instance Index Cauchy where
    Cauchy x # n = x n

instance Index Cauchy' where
    Cauchy' x u # n = x (u n)


toDouble :: Real a => a -> Double
toDouble = realToFrac
```

A.2 CalkinWilf.hs

```haskell
module CalkinWilf (
    calkinWilf,
    calkinWilf',
    ) where


-- https://www.cs.ox.ac.uk/jeremy.gibbons/publications/rationals.pdf


calkinWilf :: [Rational]
calkinWilf = iterate next 1


next :: Rational -> Rational
next x = recip (fromInteger n + 1 - y)
  where
    (n, y) = properFraction x


calkinWilf' :: [Rational]
calkinWilf' = 0 : concatMap (\x -> [-x, x]) calkinWilf
```

A.3 CauchyBenchmarks.hs

```haskell
-- Cauchy sequence benchmarks
--
-- Run them like so:
-- > cabal install criterion
-- > ghc CauchyBenchmarks.hs
-- > ./CauchyBenchmarks -o cauchy-benchmarks.html

```

```haskell

{-# LANGUAGE RankNTypes #-}


import Criterion.Main
import Data.List
import Data.Proxy
import Numeric.Natural

import Cauchy


class (Fractional a, Index a) => CauchyImpl a where
    euler :: a

instance CauchyImpl Cauchy where
    euler = expC 1

instance CauchyImpl Cauchy' where
    euler = expC' 1


main :: IO ()
main = defaultMain [
    benchmark "simple" $ (euler * euler + euler) / euler,
    benchmark "complicated" $ sum [fromInteger n * euler | n <- [1 .. 50]]
    ]
  where
    benchmark :: String -> (forall a. CauchyImpl a => a) -> Benchmark
    benchmark label expr = bgroup label [
        bench "single-function" (nf (# 50) (expr :: Cauchy)),
        bench "double-function" (nf (# 50) (expr :: Cauchy'))
        ]
```

Bibliography

E. Bishop and D. Bridges. *Constructive analysis*. Grundlehren der mathematischen Wissenschaften. Springer-Verlag, 1985. ISBN 9783540150664. URL https://books.google.co.nz/books?id=kxzvAAAAMAAJ.

P. Gowland and D. Lester. The correctness of an implementation of exact arithmetic. In *4th Conference on Real Numbers and Computers, 2000, Dagstuhl, 125*, volume 140, 2000.

D. Jarden. A simple proof that a power of an irrational number to an irrational exponent may be rational. *Scr. Math*, 19:229, 1953.

S. Marlow. Haskell 2010 language report, 2010. URL https://www.haskell.org/report/node/2010/haskell.html.

Prime.mover. Uniqueness of simple infinite continued fraction. *ProofWiki*, Sept. 2013. URL https://proofwiki.org/w/index.php?title=Uniqueness_of_Simple_Infinite_Continued_Fraction&oldid=159386.

S. G. Simpson. *Subsystems of second order arithmetic*, volume 1. Cambridge University Press, 2009.

A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *J. of Math*, 58 (345-363):5, 1936.

P. Wadler. Propositions as types. *Commun. ACM*, 58(12):75–84, Nov. 2015. ISSN 0001-0782. doi: 10.1145/2699407. URL http://doi.acm.org/10.1145/2699407.