An Introduction to Formal Real Analysis, Rutgers University, Fall 2025, Math 311H

Lecture 3: More fun with Sequences

Prof. Alex Kontorovich

This text is automatically generated by LLM from "Real Analysis, The Game", Lecture 3

1 More on sequences

In Lecture 2, we learned the definition of a sequence `a : ℕ→ℝ` converging to a limit `L`: for any tolerance $\varepsilon > 0$, there exists a time `N`, so that, for any point after that time, `n ≥N`, we are within the tolerance, `|a n - L| < ε`.

We also learned how to do something completely trivial with it, namely, show that the constant sequence converges, with that constant as its limit.

Let's step it up a notch, shall we?

2 Level 1: The Archimedean Property

The so-called Archimedean Property (which I think is originally due to Eudoxus, and appears already in Euclid's Elements Book V) is a fundamental property of the real numbers that captures the intuitive notion that there are no "infinitely large" or "infinitesimally small" positive real numbers.

More precisely, it states that no matter how small ε `> 0` is, there is always a natural number `N` so that `1 / N` is even smaller than ε (and of course positive). Equivalently, we can state it as: for any positive real number ε, there exists a natural number `N` such that `1 / ε < N`.

Why does this matter? The Archimedean Property is one of the most fundamental properties distinguishing the real numbers from other number systems. Without it, we could have "infinitely large" or "infinitesimally small" positive numbers, which would break most of calculus and analysis.

Our goal will be to prove the following:

Theorem (ArchProp): For any ε `: ℝ` with `0 < ε`, there exists `N : ℕ` such that `1 / ε < N`.

This is mathematically "obvious" to most people—if you have a positive number ε, no matter how small, you can always find a natural number large enough that `1 / ε` is smaller than it. But how do you actually formalize this in Lean?

2.1 The Natural Language Proof Strategy

First, let's think about this in natural language. The key insight is that we need to provide a specific natural number `N` that works.

A natural choice would be to use something related to the ceiling function. The ceiling function `x` $\mapsto\lceil$`x`$\rceil$ rounds any real number up to the nearest integer. However, there's a subtle issue here: the standard ceiling function takes values in integers $\mathbb{Z}$, but we need values in $\mathbb{N}$ (the natural numbers). These are *not* the same thing!!

Fortunately, Lean provides the "natural number ceiling function" written `x` $\mapsto\lceil$`x`$\rceil_+$, which takes any real number and returns a natural number. (You can write these symbols using `\lceil`, `\rceil`, and `\_+`. Or if you're lazy like me, just copy and paste them from elsewhere.) For negative inputs, this function returns `0`. For example, $\lceil$`-3.14`$\rceil_+$`= 0` and $\lceil$`3.14`$\rceil_+$`= 4`.

Now our strategy becomes clear:

- **Choice of N**: Use `N =` $\lceil$`1 / ε`$\rceil_+$`+ 1`

- **Why this works**: We have the "key inequality": `1 / ε` $\leq \lceil$ `1 / ε` $\rceil_+$, which holds by the definition of the ceiling function
- **Getting strict inequality**: Adding 1 gives us `1 / ε < ⌈1 / ε⌉₊ + 1`

2.2 The Lean Implementation Challenges

In Lean, the first two steps of our natural language proof work fine, but then we encounter the issue of **type coercion** ("casting" between different number types). We'll discuss this in more detail later, but again it has to do with the fact that $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$ are all different kinds of things, and we need to be able to move numbers up the "sophistication" heirarchy, with natural numbers being the simplest objects and the reals being the most complicated (so much so that we keep postponing their construction).

For example, notice that when we'll write our `have` statement to establish the key inequality:

`have fact : 1 / ε≤⌈1 / ε⌉₊ := by WhateverTheProofIs`

Lean will record it as:

`fact : 1 / ε≤↑⌈1 / ε⌉₊`

Notice the mysterious up arrow ↑. This represents a coercion function from natural numbers to real numbers:

↑ : $\mathbb{N} \to \mathbb{R}$

This is because $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$ are all **different** types in Lean's type system (and really, in mathematics, as we'll see when we construct the real numbers)! Even though we think of natural numbers as being "contained" in the real numbers, formally they are distinct types of things, and Lean needs explicit coercion functions to convert between them.

Think of it this way: the natural number `3 : ℕ` and the fraction `3 / 1 : ℚ` and the real number `3.000 : ℝ` are different objects that just happen to represent the same mathematical value.

The `push_cast` tactic helps manage these coercions, kind of like `ring_nf` but for casting instead of ring operations.

2.3 New Tools You'll Need

- $\lceil \cdot \rceil_+$: The natural number ceiling function
- `push_cast`: Tactic that handles coercions between number types

- `bound`: Solves many routine inequalities

The `bound` tactic can solve many "trivial" inequalities once the types are properly aligned.

2.4 Hint:

If you get stuck and don't see a Hint, try backtracking until you do.

2.5 Lean Proof

```
Statement ArchProp {ε : ℝ} (hε : 0 < ε) :
    ∃ (N : ℕ), 1 / ε < N := by
  use ⌈1 / ε⌉₊ + 1
  have fact : 1 / ε ≤ ⌈1 / ε⌉₊ := by bound
  push_cast
  bound
```

2.6 Natural Language Version

Let's compare now to the purely natural language proof:

2.6.1 Natural Language Proof of the Archimedean Property

Theorem: For any positive real number ε > 0, there exists a natural number N such that 1 / ε < N.

Proof: Let ε > 0 be given. We need to find a natural number N such that 1 / ε < N.

Use the value N = $\lceil$1 / $\varepsilon\rceil_+$+ 1, where $\lceil\cdot\rceil_+$ denotes the natural number ceiling function.

Since ε > 0, we have 1 / ε > 0. By the definition of the natural number ceiling function, we know that:

1 / $\varepsilon \leq \lceil$1 / $\varepsilon\rceil_+$

Now, since $\lceil$1 / $\varepsilon\rceil_+$ is a natural number and N = $\lceil$1 / $\varepsilon\rceil_+$+ 1, we have:

$\lceil$1 / $\varepsilon\rceil_+$ < $\lceil$1 / $\varepsilon\rceil_+$+ 1 = N

Combining these inequalities, we get that:

1 / $\varepsilon \leq \lceil$1 / $\varepsilon\rceil_+$ < N

Therefore, 1 / ε < N, which completes the proof.

Significance: The Archimedean Property is fundamental to analysis because it ensures that the real numbers have no "infinite" or "infinitesimal" elements. It guarantees that we can always find natural numbers large enough to dominate any given positive real number when we take their reciprocals. This property is essential for many limit processes and is equivalent to the completeness of the real numbers in certain formulations of real analysis.

2.7 Review of Common Pitfalls

- Don't use the regular ceiling function $\lceil \cdot \rceil$ - it returns integers, not natural numbers!
- **Watch out for casting issues** - if `bound` isn't working, try `push_cast` first
- The addition $\lceil$`1 / ε`$\rceil_+$`+ 1` happens in $\mathbb{N}$, then gets cast to $\mathbb{R}$ - this is why we need `push_cast`

Historical Note: While often attributed to Archimedes (c. 287-212 BCE), this property was likely known to Eudoxus (c. 408-355 BCE) and appears in Euclid's *Elements*. Archimedes used a version of this principle in his method of exhaustion, particularly in calculating areas and volumes by approximating them with polygons of increasing numbers of sides.

3 Level 2: Our First Real Limit

Congratulations! You've just proved the Archimedean Property. Now let's use it to prove something genuinely interesting: our first non-trivial limit.

3.1 The Goal: Proving that `1 / n` →`0`

We want to prove that the sequence `a(n) = 1 / n` converges to `0` as `n` approaches infinity. This is intuitively obvious—as `n` gets larger, `1 / n` gets smaller and approaches `0`. But how do we make this rigorous using the ε-`N` definition of limits?

Theorem: The sequence `a(n) = 1 / n` converges to `0`.

This might seem straightforward, but let's see it as a test of the definition.

3.2 Recall: The Definition of Sequential Convergence

A sequence `a : ℕ→ℝ` converges to a limit `L` (written `SeqLim a L`) if:

For every ε `> 0`, there exists `N : ℕ` such that for all `n ≥N`, `|a(n) - L| < ε`

In formal notation: ∀ ε `> 0, ∃N, ∀n ≥N, |a(n)- L| < ε`

For our specific case with `a(n) = 1 / n` and `L = 0`, this becomes: ∀ ε `> 0, ∃N, ∀n ≥N, |1 / n - 0| < ε`

3.3 The Natural Language Proof Strategy

Here's how we'll prove this step by step:

Step 1: Let ε `> 0` be given. (This will correspond to `intro ε hε`)

Step 2: We need to find `N` such that for all `n ≥N`, we have `1 / n < ε`.

Key insight: We need `1 / n < ε`, which is equivalent to `1 / ε < n` (since both sides are positive). So we need `n` to be larger than `1 / ε`. When our Engineer requests the tolerance of ε `= 1/100`, the Machinist replies, ok, I can do that, but I'll need `N = 1 / ε= 100` days in my factory.

Step 3: That's exactly why we developed the Archimedean Property! It tells us that there exists some natural number `N` such that `1 / ε < N`. Rather than reproving that from scratch, we can simply quote this fact; then we'll choose such an `N` and use it.

Step 4: Now let `n ≥N` be given. We have:

- `1 / ε < N` (by our choice of `N`)

- `N` $\leq$`n` (by assumption)
- Therefore: `1 / ε < N` $\leq$`n`, so `1 / ε < n`
- Taking reciprocals (and flipping the inequality): `1 / n < ε`

Step 5: Since `|1 / n - 0| = |1 / n| = 1 / n < ε`, we're done!

3.4 The Lean Implementation Challenges

3.4.1 Challenge 1: Cross-Multiplying Fractions

Our key step is showing that `1 / n < ε`. In paper mathematics, we'd simply cross-multiply to get `1 < n * ε`. But Lean is very careful about division by zero, so we can't just cross-multiply willy-nilly.

The Problem: We want to go from `1 / n < ε` to `1 < n * ε`, but this is only valid if `n > 0` and `ε > 0`.

Solution: The `field_simp` tactic handles this automatically! It will clear denominators by cross-multiplying, but only after it can verify that all denominators are positive (or at least non-zero).

3.4.2 Challenge 2: Linear Arithmetic

Once we've cleared the fractions, we need to combine various inequalities like:

- `1 / ε < N` (from the Archimedean Property)
- `N` $\leq$`n` (our assumption)
- `1 < n * ε` (our goal after clearing denominators)

For some reason, the `bound` tactic doesn't always handle these linear combinations well, especially when they involve multiplication by variables.

Solution: The `linarith` tactic is specifically designed for linear arithmetic. It can take a list of hypotheses and solve goals that follow from linear combinations of those hypotheses.

3.4.3 Challenge 3: Explicit Type Casting

Remember those mysterious up arrows ↑ from the last level? They're back! When we write `1 / n`, Lean sees this as `1 / ↑n` where `n` starts as a natural number but needs to be cast as a real number.

Sometimes we have to be specific about what type of casting to use. The expression `1 / ↑n` could be ambiguous—are we casting to integers, rationals, reals, or something else?

Solution: Instead of an up arrow, we can specify the casting explicitly with this syntax: `(n : ℝ)`. This tells Lean exactly what type we want to cast `n` to, in this case, the reals. This eliminates ambiguity and makes your proofs more precise.

3.4.4 Challenge 4: Casting in Tactic Applications

Sometimes you want to apply a tactic or theorem, but the types don't quite match because of casting issues. For example, you might have the hypothesis that `N ≤n` where `n, N` are naturals, but `bound` or `linarith` are searching for a proof that `(N : ℝ)≤(n : ℝ)` as reals.

Solution: The `exact_mod_cast` tactic is like `apply`, but it automatically handles the type coercions for you. If you're trying to prove `(N : ℝ)≤(n : ℝ)`, and you have the hypothesis `h : N ≤n` (as naturals), then you can write: `exact_mod_cast h`. Lean will look at `h` and realize, oh it's exactly what you're trying to prove, but just cast to a different number system; and it'll figure out the proof from there.

3.5 New Tools You'll Need

- `field_simp`: Clears denominators by cross-multiplying, but only when it can prove the denominators are non-zero. This is the key to handling fractional inequalities safely.

- **Arithmetic with inequalities**: You might also find the `linarith` tactic helpful. It is a very powerful, general tactic like `ring_nf`, but instead of proving algebraic *identities*, it proves *inequalities* involving "linear arithmetic" on the specified hypotheses. For example, if you have as hypotheses: h_1 `: X ≤Y`, h_2 `: 2 * Y ≤Z`, and your Goal is to prove that `2 * X ≤Z`, then simply calling `linarith [`h_1`,` h_2`]` will do the trick. So add as many inequality hypotheses to your Game Board

as you may need, and then call `linarith` on them to prove a Goal. I find that `linarith` is best called at the very end, when you've assembled all your facts and are ready to close a Goal (but it has many other uses as well, as you'll see).

- **Explicit casting** `(n : ℝ)`: Tells Lean exactly what type to use, eliminating ambiguity in expressions like `1 / n`. You only need to cast once in any expression, and Lean will automatically cast everything else. For example, you can say `(0 : ℝ)< N`, and Lean will figure out that the `0` you mean is a real number, and so, to compare that with `N`, the latter too must be cast to the reals.
- `exact_mod_cast`: Automatically handles type coercions between different number types ($\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$), when the Goal is *exactly* what you're trying to prove, just needs to get coerced.

3.6 Pro Tips for This Level

1. **Use** `change` to convert `SeqLim a 0` to its definition
2. **Apply the Archimedean Property** to get the existence of an `N`
3. **Use** `choose` to extract the `N` from the existential statement
4. **Establish positivity first** before using `field_simp`
5. **Work step by step** - don't try to do everything at once!

3.7 What Makes This Non-Trivial?

Unlike the constant sequence (which was essentially definitional), this proof requires:

- **The Archimedean Property** to find a suitable `N`
- **Careful type management** between $\mathbb{N}$ and $\mathbb{R}$
- **Positivity arguments** to handle division
- **Inequality manipulation** to connect our bounds

This is a perfect example of how even "obvious" mathematical facts require sophisticated machinery to prove rigorously!

3.8 Lean Proof

```
Statement OneOverNLimZero (a : ℕ → ℝ) (ha : ∀ n, a n = 1
    / n) :
SeqLim a 0 := by
  change ∀ ε > 0, ∃ N, ∀ n ≥ N, |a n - 0| < ε
  intro ε hε
  have f1 : ∃ (N : ℕ), 1 / ε < N := by apply ArchProp hε
  choose N eps_inv_lt_N using f1
  use N
  intro n n_ge_N
  ring_nf
  specialize ha n
  rewrite [ha]
  have f2 : |1 / (n : ℝ)| = 1 / n := by bound
  rewrite [f2]
  have f3 : 0 < 1 / ε := by bound
  have Npos : (0 : ℝ) < N := by linarith [f3,
     eps_inv_lt_N]
  have N_le_n : (N : ℝ) ≤ n := by exact_mod_cast n_ge_N
  have npos : (0 : ℝ) < n := by linarith [Npos, N_le_n]
  field_simp
  field_simp at eps_inv_lt_N
  have f4 : N * ε ≤ n * ε := by bound
  linarith [eps_inv_lt_N, f4]
```

3.9 Natural Language

As usual, compare to what's written in textbooks:

3.9.1 Natural Language Proof of `1 / n` →`0`

Theorem: The sequence `a(n) = 1 / n` converges to `0`.

Proof: Let ε `>` `0` be given. We need to find `N` : ℕ such that for all `n` ≥`N`, we have `|1 / n - 0|` < ε.

Since ε `>` `0`, we have `1 /` ε `>` `0`. By the Archimedean Property, there exists a natural number `N` such that `1 /` ε `<` `N`. We choose this value of `N`, and use it.

Now let `n` ≥`N` be given. We need to show that `|1 / n|` < ε.

Since `1 / ε < N` and `N ≤ n`, we have that `1 / ε < n`. Since both `1 / ε` and `n` are positive, taking reciprocals reverses the inequality: `1 / n < ε`.

Therefore, `|1 / n - 0| = |1 / n| = 1 / n < ε`, which completes the proof.

3.10 What We Just Accomplished

This proof demonstrates several key concepts:

1. **The power of the Archimedean Property**: Without it, we couldn't guarantee the existence of a suitable `N`.
2. The ε-`N` definition in action: We explicitly constructed `N` in terms of ε, showing the quantifier structure ∀ε, ∃`N`, ∀`n`.
3. **Rigorous inequality manipulation**: What seems "obvious" requires careful attention to positivity and type casting.
4. **The bridge between intuition and formality**: The intuitive idea that "`1 / n` gets arbitrarily small" becomes a precise mathematical statement.