# Lean Continuity Lemmas

James Gibson

February 2022

## 1 Introduction

For my first LEAN project, my intent was to prove that for a continuous function defined on a closed interval of the reals, the image of the function is bounded from above and below and achieves its bounds. Due to time limitations and a target length of 200 lines of code, my end goal proved to be too ambitious for my current ability. However, I did manage to make a large amount of progress on my goal, proving sequential continuity and continuity are equivalent for any point in a closed interval on the reals and proving that a continuous function on a closed interval of the reals is bounded.

This document will serve to describe my thoughts on my coding experience: what I struggled with, what I enjoyed, and things I would change and redo given the opportunity.

## 2 Sequential continuity equivalent to Continuity

### 2.0.1 Overview

To start, we'll define what it means to be continuous on an interval $[a, b]$. LEAN's inbuilt continuity and tendsto are defined using filters which differ from the Epsilon-Delta continuity I am familiar with. For this reason, I'll define my own Epsilon-Delta versions of these concepts which I'll use exclusively in this project. Unfortunately, we'll lose the Mathlib API which has been created for these definitions, meaning we will have to write and prove more lemmas that we need that we could otherwise rely on Mathlib for. To avoid getting stuck up on individual lemmas, we will often use sorry to avoid unnecessary proofs so that we can make more progress on our goal.

Now that we have definitions for continuity and sequential limits, we can go about proving that sequential continuity holds if and only if the function is continuous. In lean, this lemma for a particular point in $[a, b]$ is as follows:

```
lemma I_cont_sql_cont_pt (a b : ℝ) (f : I a b → ℝ) (x : I a b):
I_pt_continuity a b f x ↔ ∀ (s : {seq : ℕ → I a b //
Itendsto a b seq x}), Rtendsto (f ∘ s) (f x)
```

In words, the left side simply means function f is continuous on interval $[a, b]$ at point x. The right side is the statement that for all s of the subtype of `ℕ → I a b` such that s tends to x, then the sequence `(f ∘ s)` tends to $fx$.

### 2.0.2 Forward implication

For forward implication of the proof is straightforward and didn't prove to be too difficult. The idea is to:

1. introduce an arbitrary s,
2. find a delta neighbourhood around x so that the image of this is within epsilon of fx,
3. find a sufficiently large N such that above this the sequence is within delta of x,
4. use the above to conclude that `(f ∘ s)` tends to $fx$.

I was able to complete this proof using only commands and tactics introduced in the first 3 weeks of the module so have little to comment on problems that I ran into.

### 2.0.3 Backwards implication

The backwards implication of the proof was much more challenging and proceeds as follows:

1. Assume the goal is not true and aim to find a contradiction
2. Use the negation of the goal to find a sequence $t_n$ which converges to x but $ft_n$ is at least epsilon away from $fx$ for all n
3. Specialise our starting hypothesis to this sequence
4. Use this to show there exists an N such that above this, $f(t_n)$ is within epsilon of $fx$
5. Derive a contradiction from 1) and 4).

This argument is much more complex that the forward implication and I needed to use many more commands than in the previous direction. One main difficulty I encountered was creating the sequence $t_n$ and then proving that $t_n$ tends to x. The way I approached this was creating by hand a sequence $s_n : \frac{1}{n}$ and then bounding by

$$|x - t_n \leq sn = \frac{1}{n}$$

A different way to approach this would be to take an arbitrary sequence which tends to 0 rather than sn. However to do this, you need a lemma that the subtype:

```
{seq : ℕ → I a b // Rtendsto a b seq 0 ∧ ∀ (n : ℕ), seq n > 0}
```

is non-empty. If we were using the conventional filter definitions of continuity and tendsto, this would be a result in Mathlib which we could use here. Since we are not though, this proof is made more complex since we need to prove that $s_n$ tendsto 0. This takes up a rather large amount of lines of this proof and I am not happy with how I approached this part and the final product-heavily relying on working forward with lots of have commands. With hindsight, I would prove that the subtype separately and then use an arbitrary $s_n$ from that subtype

A second problem I encountered was the generation of $t_n$. I was expecting this to be simple, simply passing the sequence to the hypothesis of the negation of the goal and hoping lean would recognise the sequence mapped to the desired type of the input and create a new sequence with the desired properties. This however didn't work and I consulted the Xena project discord server and was recommended to use the choose function. To do so, I needed to express a new hyp:

```
have h₁ : ∀ (n : ℕ), ∃ (y : I a b), |x.val - y.val| < sn n ∧ ε ≤ |f x - f y|
```

Doing so meant this was now in the format where choose can generate a new sequence:

```
exact h (sn n) nat.one\_div\_pos\_of\_nat.
```

# 3 Continuous function is bounded

### 3.0.1 Overview

We start by stating the prerequisite lemmas required for proving a continuous function is bounded; firstly, the Bolzano-Weierstrass Theorem. This wont be proved in this project since it could take a whole project to do on its own, rather we'll just admit we're sorry and move on.

We'll also define a lemma called nat_ceil, which takes in a real, var, and a natural, n, and gives a proof that `⌈var + n⌉₊ ≤ ⌈var⌉₊ + n`. I found proving this lemma very challenging and ultimately couldn't complete it but have left the skeleton that almost worked. My approach began by splitting into two cases, one where `var ≤ -n` and the other case being the opposite. The first case was simple since no coercion was needed.

The second case was much harder. The closest thing I got to working was coercing the both sides of the inequality to the reals, and proving if the inequality held in the reals, it would also hold in the naturals. This was required as `↑⌈a⌉₊ = ↑⌈a⌉`, the equality needed to turn our natural ceil into just a ceil, was coercing both sides to the reals. I needed to use this as the results for ceil are more useful to us than the results for the natural ceil. However, this created its own difficulties. Since $n$ is already being coerced to $\mathbb{R}$, coercing it again, i.e. `↑↑n`, was causing a deterministic timeout. Trying other methods to accomplish this faced similar or worse errors. Overall, my inability to sort out the proof of this

lemma is likely down to my inexperience and lack of knowledge of type theory-something I hope to improve in the future to better my ability to approach these sorts of lemmas.

### 3.0.2 Main lemma

We will work backwards from our goal, proving that its sufficient to show that if we have an upper and lower bound to $f$, then taking the max of the modulus of these bounds will bound f. Doing so is straightforward, and results only requires the fact that `b ≤ max a b`, which is already a result in Mathlib. We will only prove that $f$ is bounded above here, since bounding it from below follows from applying the prior to $-f$. I would have done so in my project but I need to balance my time between this project and my other modules. Proving $f$ is bounded from above is done by inducing a contradiction as follows:

1. Assume the negation of the goal
2. Use the negation to construct a sequence $t_n$ satisfying $n \leq ft_n$
3. This sequence is bounded so has a convergent subsequence, call this $t_n \circ q_n$
4. By sequential continuity, $f \circ t_n \circ q_n$ must tend to $fc$
5. Above some N, $f \circ t_n \circ q_n N$ must be strictly greater than $fc - 1$
6. `f (tn ((qn (⌈f (tn (qnN))⌉₊ + 2))))` is less than $fc + 1$ but is greater than or equal to $f \circ t_n \circ q_n N + 2$.
7. Observe that 6 implies $f \circ t_n \circ q_n N \leq fc - 1$
8. Clearly 5 and 7 contradicts so we have finished our proof.

This proof had many steps so took quite a long time to formalise in lean. Completing step 6 proved to be the most difficult step and I used lots of forward reasoning 'have' to get to the final result. In this step, I encountered the same problem from earlier where `(n : ℕ)` had to be coerced twice which was causing another deterministic timeout. I have left some skeleton of what I tried to provide a gist of what results these parts should rely on. Given more time and experience I'd like to revisit this part and try to either reduce it to a simpler argument or use the lean tactics to reduce the number of `have` commands in my proof.

# 4 Conclusion

To conclude, I'm happy and content with what I have produced. Although I suspect I didn't use lean's tactics as much as I could have, I think I am more comfortable about when and how to use the majority of them having done this project. In the future I want to become more familiar the obtain tactic since I'm aware it can reduce the amount of `have` and `cases` occurrences which would slim down my code significantly.