INNER PRODUCT SPACES IN LEAN

MARTIN SKILLETER

ABSTRACT. In this article we discuss the implementation of inner product spaces (and subsequently, Hilbert spaces) in the mathematical computing language Lean. We outline the evolution of said implementation as problems arose and how these problems shed light on important subtleties relating to the theory of inner product spaces. Specifically, we consider the fundamental differences in the implementations of inner product spaces over $\mathbb{R}$ and $\mathbb{C}$, and how these implementations could not be generalised into abstract fields.

1. INTRODUCTION

The theory of inner product spaces is well-documented and is often taught at an undergraduate level in mathematics. The purpose of this paper is not to expound upon this, but rather to discuss the process of implementing this theory in the mathematical computing language Lean.

Lean is a burgeoning language in the field of fully rigorous proof-checking. It was originally developed to allow for aggressive program-optimisation techniques and is now used by mathematicians to check the completeness of mathematical proofs. As a result of its rigorous nature, Lean as a proof-checker is extremely pedantic, and this caused many problems in the optimisation of proofs in Lean.

We began this project with the following goals in mind:

(1) To formalise the axioms of an inner product space
(2) To prove that an inner product naturally induces a norm. As a by product of this, to prove the famous Cauchy-Schwarz inequality
(3) To prove the Jordan Von-Neumann Theorem: a normed vector space is an inner product space if and only if it satisfies the parallelogram law
(4) To implement the theory of Hilbert spaces; complete, possibly infinite-dimensional inner product spaces over $\mathbb{R}$ and $\mathbb{C}$

Ultimately, we restricted our attentions specifically to inner product spaces over $\mathbb{R}$, although the basics of inner product spaces over $\mathbb{C}$ were implemented (this is discussed in "The axioms of an inner product spaces"). Goal (4) was not completed, although progress was made towards proving the Riesz-Representation Theorem. It is worth noting that all code written in the course of this project is marked as "noncomputable" because it relies on the construction of the real numbers. Because these are equivalence classes of (infinite) Cauchy sequences, it is impossible to compare two real numbers and so no real calculations can be made.

We now discuss each of the above goals in turn.

2. The axioms of an inner product space

Here we will present a motivation for the theory of inner product spaces. Inner product spaces are vector spaces V equipped with a function $(\cdot,\cdot) : V \times V \to \mathbb{F}$, where $\mathbb{F}$ is either $\mathbb{R}$ or $\mathbb{C}$. This function (the so-called "inner product") must satisfy the following axioms:

(i) Conjugate symmetry: $(\forall x, y \in V)\ (x, y) = \overline{(y, x)}$
(ii) Linearity: $(\forall x, y, z \in V)(\forall \lambda \in \mathbb{F})\ (\lambda x + y, z) = \lambda(x, z) + (y, z)$
(iii) Positive Definiteness: $(\forall x \in V \setminus \{0\})\ (x, x) > 0$

A simple (and motivating) example of an inner product is the dot product on $\mathbb{R}^n$, given by $(x, y) = \sum_{i=1}^{n} x_i y_i$. This is the inner product most people will be familiar with, as it underpins the concept of angles in Euclidean geometry.

Note that in the definition of positive definiteness, we are implicitly making use of the fact that the imaginary component of the inner product of a vector with itself is always zero (this is a simple consequence of conjugate symmetry). This means that we may interpret (x, x) as an element of $\mathbb{R}$ (and hence make use of the natural ordering on $\mathbb{R}$), even if $\mathbb{F} = \mathbb{C}$. While a human mathematician may not make much of this subtlety, such a distinction is of vital importance in Lean.

Mathematicians study inner products because they allow for the generalisation of the geometric concepts of lengths and angles. We can interpret the concept of orthogonality by saying that x is orthogonal to y if $(x, y) = 0$. This allows for a proof of Pythagoras' Theorem in a more general setting, which we will discuss later. They also naturally induce a norm, which allows for a concept of distance and length in an abstract vector space (this is the main topic of exposition in Section 3.)

Initially, these axioms were implemented in Lean as follows:

```
class inner_product_space (α : Type*) [add_comm_group α]
[vector_space ℂ α] extends has_inner_product α :=
(conj_symm : ∀ (x y : α), inner_product x y = conj (inner_product y x))
(linearity : ∀ (x y z : α), ∀ (a : ℂ), inner_product (a • x + y) z =
a * inner_product x z + inner_product y z)
(pos_def : ∀ (x : α), x ≠ 0 → ∃ (r : ℝ), r = inner_product x x ∧ r > 0)
```

Comparing these with the above axioms shows only one key difference: the use of an existential quantifier in the positive definiteness axiom. At first glance, this should not cause a serious problem when proving theorems in Lean. However, such a definition proved almost impossible to work with for the following reason.

A key theory in the study of inner product spaces is that an inner product induces a norm via the definition:

$$\|x\| = \sqrt{(x, x)}$$

Initially, even defining this norm proved to be impossible; with the original definition of positive definiteness, the natural way to define such a norm would have been as the square root of the real number guaranteed to exist. However, retrieving a value from inside an existential quantifier is quite difficult, because Lean has no way of knowing what such a value would be. This obstacle was overcome by the use of a Lean module called "classical". This is the implementation of classical mathematics in Lean and the "classical.some" function was almost exactly what we needed: given an existential quantifier and a proposition, a value of the correct type is generated,

as well as a hypothesis that this value satisfies the given proposition. We were then able to define the norm as the square root of the real value given by "classical.some".

When we began to prove some simple lemmas, however, a fundamental flaw in our implementation reared its head. We had never guaranteed the uniqueness of such a real value, and so our vectors could have multiple lengths! Several possible solutions to this were put forward, among these the idea of implementing a partial order on the complex numbers, so that no coercion from the real numbers was required. It was ultimately decided that any solution would require at least some change to the initial axioms, so the original definition of positive-definiteness was removed and replaced with the following:

$$(\mathrm{pos\_def} : \forall\, (x : \alpha),\ x \neq 0 \to (\mathrm{inner\_product}\ \mathrm{x}\ \mathrm{x}).\mathrm{re} > 0)$$

This definition has one key advantage over the original: it specifies what real number satisfies the coercion used in our original definition (the real part of the inner product itself). This axiom proved to be more useful when defining and proving lemmas because we now had an explicit value to work with. Additionally, it meant that we no longer had to import the "classical" module into our files. Although many places in the MathLib standard library make use of "classical", it is common practise to restrict imports where possible.

It was while defining these axioms that the first key distinction between the complex numbers and the real numbers came to light. While the axiom of "linearity" was almost identical (only requiring the substitution of $\mathbb{C}$ to $\mathbb{R}$), the axiom of conjugate symmetry had another important distinction. In the real numbers, conjugation is the identity automorphism and so this axiom became

$$(\mathrm{conj\_symm} : \forall\, (x\ y : \alpha),\ \mathrm{inner\_product}\ \mathrm{x}\ \mathrm{y} = \mathrm{inner\_product}\ \mathrm{y}\ \mathrm{x})$$

Both conjugation and the identity map are what is known as involutions, meaning they are their own inverses. While a proof of this fact for conjugation is given in MathLib under the name "conj_conj", constantly applying it to simplify computations that would be trivial to a human mathematician became tiresome. This, and other instances of calculations like it, was a major factor in the decision to restrict attention to the case of inner product spaces over the real numbers. Any proof that is valid in $\mathbb{R}$ almost certainly generalises to $\mathbb{C}$, but it is more difficult to convince Lean of this fact because of implicit coercions between the real and complex numbers. As an example of this, we proved the famous Cauchy-Schwarz inequality ($|(x, y)| \leq \|x\|\|y\|$) when proving that an inner product naturally induced a norm. Over $\mathbb{R}$, this proof was 50 lines long. Over the complex numbers, the proof was never completed. However, even the draft version numbered at nearly 100 lines because of mathematical manipulations that most human mathematicians would consider trivial.

The final point worth noting is that the theory could not be generalised to include both $\mathbb{R}$ and $\mathbb{C}$ (it is standard practise to treat these two cases independently). At a basic level, this is because $\mathbb{R}$ and $\mathbb{C}$ have very different algebraic properties (the only automorphism on $\mathbb{R}$ is the identity map, for example) and because $\mathbb{R}$ is ordered, while $\mathbb{C}$ is not. Nevertheless, we considered generalising the results here with the theory of sesquilinear forms. These are bilinear maps with the properties of linearity and conjugate symmetry defined above. However, they lack the property of positive definiteness that we desire in much of our theory. In particular, a sesquilinear form need not induce a norm on a vector space, so we abandoned these.

3. Inner products and norms

Over the course of the project, a large amount of time was spent focused on proving that an inner product space naturally induced a norm. Normed spaces were already implemented in Lean's MathLib standard library. They were built as "towers of extensions": first, we prove that an inner product space is a metric space, then a normed group and finally, a normed space. However, each of these instances individually extend other classes. For example, before "metric_space" can be instantiated, a instance of "has_dist" must be created. "dist" is the distance function on the metric space, and so we needed to prove that it satisfies the axioms of a metric before creating the instance of "metric_space".

It was at this time that the problem of class instance resolution began to appear. When stating a theorem or lemma in Lean, users are able to make use of a feature called implicit variables to avoid having to explicitly state every instance that should be required; the trade-off for this is that Lean needs to be able to find the required instances itself. By using the "@" symbol, these implicit variables can be provided explicitly. When initially proving that inner products spaces were metric spaces and normed groups, we created specific instances of these classes. As more and more of these instances built up, Lean's automatic instance resolver began to struggle to find the instances required to resolve implicit variables. For example, the real numbers $\mathbb{R}$ are implemented in MathLib and are obviously a commutative group under addition. However, the Lean parser would often find itself in endless loops as it tried to find this instance, because "normed_space" is implemented as requiring an instance of the field over the vector space to be an additive commutative group. Lean's thought process was then "I need an instance of $\mathbb{R}$ as an additive commutative group. Because α is a normed space over $\mathbb{R}$, it must be that $\mathbb{R}$ is an additive commutative group. But α can't be a normed space over $\mathbb{R}$ unless $\mathbb{R}$ is an additive commutative group, so I'd better go find that instance".

Our solution to this problem was to change all instances into definitions (using the keyword "def"). As an example, the declaration of an instance of "metric_space" looked like:

```
def ip_space_is_metric_space : metric_space α :=
{dist_self := ip_dist_self, eq_of_dist_eq_zero := ip_eq_of_dist_eq_zero,
dist_comm := ip_dist_comm, dist_triangle := ip_dist_triangle}
```

The main benefit was that the Lean parser no longer recognised these as instances and so could not get trapped in the endless loop described above. The downside, however, is that any theorems that required an instance of "metric_space" needed to have this provided explicitly. In places, this became too cumbersome to work around, so a local attribute was declared (this means that inside a section, an instance can be declared that will not be visible to any theorems outside the section). In the file "orthonormal_bases.lean", a section named "norm_known" was created for the sole purpose of letting the Lean parser find the instance of "normed_space" for us. Aside from this small workaround, no solution to this problem has been found.

Another problem faced was interacting with the MathLib library. Because users often download a local copy of MathLib, developers try to keep size and compile time low for obvious reasons. This means that many obvious or trivial lemmas are either overlooked or intentionally left out. For example, there is no lemma in the real number library that says $r^2 = |r|^2$, which turned out to be used in multiple places

when proving that the norm on an inner product space respects scalar multiplication. Additionally, no instance of "has_norm" for the real numbers was ever created. It is possible that this is because multiple norms can be put on the real numbers, each of which induces a different topology, but the absolute value norm is widely accepted as being the canonical choice. Although small decisions like these did not largely inconvenience us, they meant that time had to be spent in proving facts unrelated to inner product spaces.

The main problem when using the MathLib library was our own. MathLib lemmas are named using a convention that we found unintuitive. Supposedly, the purpose of this is to allow users to guess the name of a lemma if they know its content. However, if you cannot hazard such a guess, this means trawling through the MathLib library in search of a relevant proof term. Scott Morrison's "library_search" tactic, which searches MathLib to find a lemma when given the content, helped alleviate this problem somewhat. However, it has restrictions (such as difficulty interacting with implicit arguments) which means that it does not always find such a lemma name. As a solution to this problem, we began to keep a separate document listing the names and content of lemmas used in MathLib. Although this grew in length over the course of the project, it was still a more useful index than MathLib.

4. The Jordan Von-Neumann Theorem

When starting this project, we were perhaps overly ambitious when establishing goals. We had initially hoped to prove the Spectral Theorem for real inner product spaces: a inner product space α has an orthonormal T-eigenbasis if and only if T is a self-adjoint (a.k.a. symmetric) operator. As we did not, for instance, fully define the Gram-Schmidt Procedure, this was out of our reach. A famous result we did prove, however, was the Jordan Von-Neumann Theorem.

Theorem 4.1. *Let α be a normed vector space. Then α is an inner product space if and only if it satisfies the parallelogram law ($\forall(x\ y : \alpha)\ \|x+y\|^2 + \|x-y\|^2 = 2\|x\|^2 + 2\|y\|^2$).*

Because we were unfamiliar with this theorem prior to the start of this project, we followed the paper "The Jordan Von-Neumann Theorem" published online[2].

There are two parts to proving this. The first is to prove that the norm naturally induced by an inner product space satisfies this equality. This is a simple exercise in expanding the definition of the norm and then utilising linearity of the inner product (it's worth noting that this is another example of the proof being much simplified in the case of a real inner product space. The proof terms generated were nearly identical, but because of manipulations involving the imaginary components of complex numbers, the proof in the complex case was almost twice as long and had no additional substance).

Conversely, we also had to show that a norm satisfying this law induces an inner product. In the case of an inner product space over the real numbers, the inner product is defined as $(x, y) = \frac{1}{4}(\|x+y\|^2 - \|x-y\|^2)$. To prove that this is, in fact, an inner product, we had to check that it satisfied the axioms given in the Introduction. Interestingly, these axioms must be proved in a particular order. We began by proving conjugate symmetry, which was a simple exercise in manipulations using Lean. Positive definiteness follows almost immediately from the non-negativity of a norm.

The most difficult of the axioms to prove was linearity. Rather than proving the scalar multiplication and addition in one step, we proved each of these separately and then combined them at the end. We first proved the additivity of the inner product $(\forall(x\ y\ z : \alpha), (x+y, z) = (x, z) + (y, z))$. This requires the following lemma:

Lemma 4.2.

$$\forall(x\ y\ z : \alpha), \|x+y+z\|^2 = \|x+y\|^2 + \|x+z\|^2 + \|y+z\|^2 - \|x\|^2 - \|y\|^2 - \|z\|^2$$

Additivity then follows fairly easily from this. More difficult is the proof that the inner product respects scalar multiplication $(\forall(x\ z : \alpha), \forall(a : \mathbb{R}), (a \cdot x, z) = a * (x, z))$. We showed this by building upwards through the levels of the scalars: we proved that it held for the natural numbers by induction. It then followed for the integers by applying the "cases" tactic and treating the cases where it is nonnegative and negative separately. Next, we proved that it held for the rational numbers by taking the numerator and denominator and treating them as integers, meaning it followed from the previous lemma.

It was at this point that we had to deviate from the standard proof. Normally, the result is proven for the real numbers by using the density of the rational numbers to choose a convergent sequence. The previous result is then applied and we prove the equality using the uniqueness of limits in metric spaces. However, sequences are not

really used in MathLib. Instead, the choice was made to rely on a different notion of convergence, that of filters. We were unfamiliar with this concept, and this was the first instance of truly new mathematical content in the project.

5. Orthogonality and Orthonormal Bases

A major area of study in the theory of inner product spaces is the concept of orthogonality. This allows us to generalise the concept of angles to arbitrary vector spaces. We say that two vectors x and y are orthogonal if $(x, y) = 0$. A simple consequence of this definition is Pythagoras' Theorem. Although this is generally encountered in the area of trigonometry, it generalises as follows:

Theorem 5.1. *Let $x\ y : \alpha$ be orthogonal vectors. Then $\|x + y\|^2 = \|x\|^2 + \|y\|^2$.*

The proof of this is a simple matter of expanding the definitions of the norm and orthogonality. This example is illustrative of how quite low-level theory (such as the theory of right-angled triangles) can be generalised into abstract settings.

Having proven that an inner product induces a norm (and defined orthogonality along the way), our next goal was to build the theory of orthonormal bases. A set is orthonormal if the vectors in the set are pairwise orthogonal and each vector has norm 1. We began this study by defining an orthogonal set as

```
def orthog_set (L : set α) : Prop :=
∀(a b ∈ L), a ≠ b ⟹ a perp b
```

and proceeded to prove some basic results about such sets. For example, we proved that normalising an orthogonal basis gives an orthonormal basis. This result was used to build towards the definition of the Gram-Schmidt Procedure, an algorithm which takes a linearly independent set of vectors as an input and returns an orthonormal set with the same span. The version of the Gram-Schmidt Procedure commonly encountered is defined for finite sets and for obvious reasons, this does not generalise well to potentially infinite sets. We will describe our solution to this now, so let S be a countable (potentially infinite) set. We defined a function "gram_partial" that, when given a natural number n, runs the Gram-Schmidt Procedure on the first n elements of S. We then defined the main function, "gram_schimdt", as the union over all $n \in \mathbb{N}$ of "gram_partial S n". It is worth noting that this definition is only well-defined on countable sets (otherwise, there are no "first n vectors). However, our purpose in defining the Gram-Schmidt Procedure was to prove that any Hilbert space has a countable orthonormal basis, so this was acceptable.

The next main goal was to define and prove results about the orthogonal complement (sometimes called the perp space) of a set $S \subseteq \alpha$. For example, a perp space is always closed (in the topological sense), regardless of the properties of S. One method for proving this is to show that perp S contains all of its limit point, by taking a limiting sequence and proving that the limit is in perp S. Although we considered this method, we decided to take a different approach because we found an unfamiliar proof (also to avoid working with filters when it was unnecessary).

We begin by proving that the perp space of a singleton set $\{x\}$ is closed by writing it as the kernel of the linear map $\lambda\ y, (x, y)$. Using the equivalence of boundedness and continuity (discussed in the next section), we could then use the Cauchy-Schwarz Inequality to prove that this map was continuous and so its kernel was closed. Finally, we wrote the perp space of S as perp $S = \bigcap_{x \in S}$ perp $\{x\}$. One interesting side-effect of this process was learning about the tactic "unfold_coes", which reveals the

definitions of coercions in Lean. Had we known about this earlier, it is possible that the theory of inner product spaces over $\mathbb{C}$ might have been more fruitful. This is certainly an avenue worth exploring in the future.

It is not immediately obvious that a perp space always forms a vector subspace of α, but proving this was our next goal. To do this in Lean, we had to create an instance of "subspace $\mathbb{R}$ (perp S)", which required defining addition and scalar multiplication. We wished to do this in the obvious way (simply consider the elements of the perp space as vectors in α and add them), but Lean's syntax presented a problem: we had a proof that the perp space was closed under addition, but how did we tell Lean to apply it? Scott Morrison solved this with the following definition:

```
instance perp_has_add : has_add (perp S) := ⟨⟨λ x y, ⟨x.val + y.val,
@perp_add_closed _ _ _ _ _ S _ _ x.property y.property⟩⟩
```

The section "x.val+y.val" represents adding the two vectors in α, while the proof term below shows that the result still lies in perp S. Even once we knew this syntax, we could find no examples of it in the MathLib standard library. Because Lean is a very niche language, there are few online resources available and this is an illustrative example of ways in which syntactic problems can interfere with theorem proving.

For S a closed subspace of α, the following result can be proven.

```
theorem proj_exists_unique (x : α) :
∃!(y : α), y ∈ S ∧ (‖x - y‖ = Inf{r | ∃(z : S), r = ‖x - z‖})
```

Informally, this means that every vector $x : \alpha$ has a unique 'closest point' in S. A useful feature of this vector (call it y) is that y is orthogonal to $x - y$. Since $x = y + (x - y)$, this means that any vector can be uniquely decomposed into orthogonal vectors, one in S and the other in the perp space. More precisely, α is the direct sum of S and its perp space, written $\alpha = S \oplus \text{perp } S$, for any closed subspace S.

This proves to be a very useful decomposition, because we can define the orthogonal projection onto S. In Lean, this was implemented as:

```
def orthog_proj (x : α) := classical.some (exists_of_exists_unique
(@proj_exists_unique α _ _ _ _ _ S _ _ h x))
```

A main result that are not immediately obvious from the above definition is that "orthog_proj" is linear. We could also, equivalently, have defined the orthogonal projection as any map with kernel orthogonal to its image, and we would ideally have liked to have proven this was equivalent. Even proving that the map was linear was challenging, because the function value has no real property aside from being the closest point to the input. Proving any results at all (such as additivity or scalar multiplicativity, or even that the function acts as the identity on S) turned out to be matters of infuriating arithmetic manipulations. Although Lean has a tactic called "Ring" to assist with manipulations that can be solved directly from the ring axioms, there is no analagous tactic for fields.

Although the theory of orthogonal projections is not yet complete, we used some "sorried" lemmas ("sorry" is a keyword in Lean that allows a user to mark a proof as unfinished while still allowing Lean to compile the code) to mark theorems we would like to have proven before turning to some more important results. Foremost among these is the Riesz-Representation Theorem:

Theorem 5.2. *Let $T : \alpha \to \mathbb{R}$ be a continuous (equiv. bounded) linear map. Then there exists a unique vector $x : \alpha$ such that $T(y) = (x, y)$.*

We implemented this in Lean as:

```
@is_bounded_linear_map ℝ _ α (ip_space_is_normed_space) _ _ f
⟹ (∃(x : α), is_riesz_rep f x)
```

The above contains an example of implicit arguments needing to be given explicitly, as can be seen from the use of the @ symbol. This goes back to the discussion earlier, where we could not provide an instance of "normed_space" outright without causing instance resolution errors. This meant that we had to manually provide it each time it was needed. The underscores in the definition above indicate that Lean should be able to infer the argument or instance from other information it is given.

This was a theorem that we had only seen in the finite-dimensional context (recall that the dimension of a vector space is the number of elements in any basis for that space). In this setting, all linear functionals are continuous, and so we neglected this condition when stating the theorem. Using the Cauchy-Schwarz Inequality, we were then able to prove that all linear functionals were bounded and hence continuous. This is entirely untrue in the infinite-dimensional context (it can be shown that every infinite dimensional inner product space has an unbounded linear functional) and is a good example of needing to have a full understanding of the mathematics before trying to write an implementation in Lean.

Before defining the Riesz-Representation Theorem, we had thought to implement bounded linear operators and the operator norm in Lean because we could not find this anywhere in MathLib. We wished to show that bounded linear operators form a normed vector space under pointwise addition and scalar multiplication. We had, in fact, nearly finished this when we discovered that bounded linear operators were implemented in MathLib (amusingly, this happened because we mistyped the name of our instance. When no error appeared, we followed the mistyped name to its definition in MathLib and found "bounded_linear_maps.lean").

This exercise was not a total loss, however, because we were able to slightly modify and reuse code from this when proving facts about orthogonal projections. Both the operator norm and the orthogonal projection are defined in terms of the infimum of a set. Because there are so few lemmas in MathLib relating to infima, there is only one proof technique. Once we realised this, we were able to return to the Git repository and recycle our proofs about bounded linear operators.

6. Hilbert Spaces

The final goal on our list was to implement the mathematical theory of Hilbert spaces.

A Hilbert space is an inner product space that is complete, meaning any Cauchy sequence (the distance between terms of the sequence gets arbitrarily small) converges. As we said in the section relating to the Jordan Von-Neuman Theorem, sequences are not commonly used in MathLib. Instead, we made use of filters via the following definition:

```
(complete : ∀{f : filter α}, (@cauchy α (@α_uniform_space α _ _ _ _))
f ⟹ ∃ x, f ≤ @nhds α (@α_topological_space α _ _ _ _) x)
```

Not much is yet complete for Hilbert spaces, but we will now outline some major theorems and how we intend to implement them in Lean. For sake of brevity, let H be fixed as a Hilbert space.

The first theorem, mentioned above, is that H has a countable orthonormal basis. Here, we are using basis in a different sense to the finite-dimensional case. Not every vector $x : H$ is in the span of such a basis; rather, it can be approximated arbitrarily closely by finite linear combinations of vectors in H (finite linear combinations are dense in H). Finite linear combinations are implemented in MathLib as finitely supported functions. $f : H \to \mathbb{R}$ is said to be finitely supported if f is nonzero at only finitely many points. On the points where f is nonzero, it may take arbitrary values. One interesting side note is that this is not the only meaning of a finite support. Alternatively, it can mean that the measure of the support of f is finite, which can allow for uncountably infinite supports. This was our original misunderstanding of the definition, and we had to read through the way the definition was used in several proofs before we understood the true implementation.

We expect that defining such a basis, and proving its existence, will be quite challenging because $\epsilon-\delta$ style analytical arguments are often quite difficult to rigorise. We do have an example of giving one in the "bounded_linear_maps.lean" file in MathLib, where such an argument is given to prove the equivalence of boundedness and continuity for linear operators. If we progress far enough to give this proof, we will likely us this as a template.

After defining this (and possibly "sorrying" the proof of its existence), we will proceed to define the Hilbert space

$$l^2(\mathbb{N}) = \{(x_1, x_2, x_3, ...) : \sum_{i=1}^{\infty} |x_i|^2 < \infty, x_i \in \mathbb{R}\}$$

and prove that it is actually a Hilbert space. This goal seems more achievable, but we will have to learn more about how (absolutely convergent) series are implemented in Lean. Our decision to implement $l^2(\mathbb{N})$ might be considered strange, as it is quite uncommon to include examples of algebraic structures in Lean. However, Hilbert spaces are quite unique in that all infinite-dimensional Hilbert spaces are "unitarily equivalent". Unitary equivalence is the isomorphism in the algebraic structure of Hilbert spaces and for all intents and purposes, two unitarily equivalent Hilbert spaces are interchangeable.

A unitary equivalence is a bijective linear map that preserves the norm (implemented in Lean as):

```
structure unitary_operator extends linear_map ℝ α β :=
(bijective : bijective to_fun)
(norm_preserving : ∀(x : α), ∥to_fun x∥ = ∥x∥)
```

A small result we showed was that, as a consequence of the Polarisation Identity, unitary operators also preserve the inner product.

Showing that all infinite-dimensional Hilbert spaces are unitarily equivalent is a non-trivial result that relies heavily on the basis defined above. We intend to build towards Parseval's Equality (an infinite-dimensional analogue of Pythagoras' Theorem) and collect results along the way.

7. Conclusions & Avenues for Further Research

Over the course of this project, we managed to define an inner product space, prove that an inner product naturally induces a norm, prove the Jordan Von-Neumann Theorem and start building towards Hilbert spaces and orthonormal bases. In the three weeks remaining in the project, we would ideally finish proving results about orthogonal projections and formally show that an inner product space can be decomposed as the direct sum of a closed subspace and its perp space. We also hope to define $l^2(\mathbb{N})$ and prove some basic results about Hilbert spaces.

If there is time, we would like to return to previous files (particularly "orthonormal_bases.lean") and replace instances of "simp" with explicit rewrites. Although these files do not compile as slowly as others in MathLib, compiling all six at once can take upwards of 5 minutes.

Although it is not discussed above, we also proved that the Cartesian product (equiv. direct sum) of two inner product spaces was an inner product space, with the inner product induced in a natural way. We would like to prove the analogue of this for tensor products, because the universal property of tensor products (any bilinear map from $V \times W$ to Z factors uniquely through a linear map $V \otimes W$ to Z) would be a useful result for other mathematicians to be able to apply. However, this would involve unravelling the definition of a tensor product as implemented in MathLib. As we are somewhat unfamiliar with the theory of tensor products in their own right, we will likely leave this.

Despite our discussion regarding sesquilinear forms in the Introduction, we would still like to generalise the theory of inner product spaces if possible. There is a natural generalisation of an inner product called an outer product, but the theory of this relies quite heavily on matrices. As matrices are not totally supported in MathLib, this should likely be left until later (although support for matrices could be added as progress was made on outer products).

Finally, if possible, we would likely to complete the analagous theory for inner product spaces over the complex numbers. This is less important, because there is current work being done to add Hermitian inner forms to MathLib (this was another factor in the decision to focus on the real numbers). However, this implementation may neglect interesting pieces of mathematics such as Hilbert spaces, orthonormal bases and the Jordan Von-Neumann Theorem, and so we would like to have our own implementation if possible.

8. References

[1] Jeremy Avigad, Leonardo de Moura, and Soonho Kong. *Theorem proving in Lean*. Microsoft Research, https://leanprover.github.io/tutorial/tutorial.pdf, 2015.
[2] Dartmouth College. The Jordan-Von Neumann Theorem.