Formalization of Harmonic Analysis in Lean

by

Wouter Bosse

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Friday June 27, 2025 at 08:45 AM.

Student number:	5643325	
Project duration:	April 22, 2025 – June 27, 2025	
Thesis committee:	Dr. M.P.T. Caspers,	TU Delft, supervisor
	Dr. Y. Murakami,	TU Delft
Daily supervisor:	J. Reimann,	TU Delft

Laymen's summary

This thesis focuses on implementing a specific type of mathematical function, known as a Schwartz function, in a computer program called Lean. To do this, we must follow Lean's strict rules, which ensure that all mathematical steps are precise and verifiable by Lean. Schwartz functions are especially useful when considering the Fourier transform, a powerful tool used in areas like quantum mechanics, engineering, and signal processing. While Lean already includes a version of these functions, this project offers a more concrete, textbook-like approach that is easier to understand. This implementation may support future implementations of results in related mathematical areas such as distribution theory, which is still underdeveloped in Lean.

Abstract

The formalization of mathematics has become increasingly popular in recent years. Formalization refers to the expression of mathematical definitions and proofs within a precisely defined system of rules, allowing software programs to verify their correctness. This thesis formalizes Schwartz functions using the interactive theorem prover Lean, which provides automation and feedback during the construction of definitions and arguments. Due to their smoothness and rapid decay at infinity, the space of Schwartz functions forms a natural setting for the Fourier transform. While Lean's library already contains an abstract formalization of Schwartz functions, this thesis presents a more concrete, coordinate-based approach that aligns more closely with standard mathematical textbooks. The knowledge gained throughout this thesis can support future formalizations of results in distribution theory, a domain that is still underdeveloped in Lean.

Contents

1 Introduction

In the recent years, the *formalization* of mathematics has become increasingly popular. Formalization refers to the process of writing mathematical definitions and proofs in a precisely defined system of rules, called a *formal language.* This allows mathematics to be verified by digital *proof assistants*, software programs that can verify proofs automatically, rather than relying solely on peer review. These tools are not only able to verify correctness of mathematical results, but also provide immediate feedback, such as errors or suggestions for next proof steps. Notable breakthroughs include the formal verification of the four colour theorem, a follow-up to Peter Scholze's Fields Medal-winning work, and the prime number theorem [23] [12] [4].

This thesis makes use of the Lean interactive theorem prover, which is such a proof assistant [13]. Lean is based on a formal language referred to as *type theory* [16]. *Type theory* was developed in the first half of the twentieth century by Bertrand Russell (and others) to deal with paradoxes that arose in the set theory of the time [37]. It treats mathematical objects as *types* that govern how these objects can be used, allowing Lean to verify mathematical statements by checking types.

However, many areas of mathematics remain underdeveloped in Lean [29]. One of the main challenges is that formalizing definitions and proofs often requires a deep understanding of the underlying mathematical structures. This makes it harder to work on new formalizations and to make use of already formalized content.

This thesis focuses on the formalization of an important notion in *harmonic analysis*, a branch of mathematics that studies the representation of functions as frequencies. It has many important applications in areas like signal processing, quantum mechanics, and partial differential equations. The *Fourier transform* plays a central role in harmonic analysis. It is an operation that transforms functions from the time or spatial domain into the frequency domain, revealing the presence and strength of frequency components in the original function. In particular, this thesis formalizes *Schwartz functions* on $\mathbb{R}^n$. These functions are infinitely differentiable and rapidly decreasing at infinity, which makes them very well suited for Fourier transformation [25].

Schwartz functions have already been formalized in Lean in a more general setting [18]. This formalization uses an unconventional definition that avoids the use of partial derivatives and polynomials completely. However, we found that when learning about Schwartz functions, this approach obstructs intuitive understanding. It requires familiarity with more abstract mathematical objects before one can understand and make use of the formalization. We aim to provide a more concrete formalization of Schwartz functions that can be understood with an undergraduate background in analysis, by working directly with higher-order partial derivatives, polynomials, and coordinate-based domains.

The thesis is structured as follows. In Chapter 2, we provide an introduction to the mathematical theory needed to understand the formalization process, as well as the motivation for why the Schwartz functions are well-suited for Fourier transformation. Chapter 3 introduces Lean's foundations and key concepts. Then, Chapter 4 shows the formalization process. Finally, in Chapter 5 we provide a discussion of our formalization and offer recommendations for future work.

2 Mathematical theory

2.1 Preliminaries

2.1.1 Terminology and notation

Given $x = (x_1, ..., x_n) \in \mathbb{R}^n$, we set $|x| = (x_1, ..., x_n)^{1/2}$. A *multi-index* α is an ordered n-tuple of nonnegative integers. For a multi-index $\alpha = (\alpha_1, ..., \alpha_n)$, $\partial^\alpha f$ denotes the derivative $\partial_1^{\alpha_1}...\partial_n^{\alpha_n} f$ and $|\alpha| = \alpha_1 + ... + \alpha_n$ denotes its size. The number $|\alpha|$ indicates the *total order of differentiation* of $\partial^\alpha f$. The space of functions in $\mathbb{R}^n$ all of whose derivatives of order at most $N \in \mathbb{N}$ are continuous is denoted by $C^N(\mathbb{R}^n)$ and the space of *infinitely differentiable* (or *smooth*) functions on $\mathbb{R}^n$ by $C^\infty(\mathbb{R}^n)$. For a function $\phi : \mathbb{R}^n \to \mathbb{R}^m$, we write $\phi(h) = o(h)$ if $\lim_{h\to 0} \frac{||\phi(h)||}{||h||} = 0$. So, this means that ϕ becomes negligible compared to the norm of the input vector h. We say that f is little o. It is straightforward to show that the sum of two little-o functions is again little-o.

2.1.2 Structures

This section uses definitions from [33], [9], and [22]. While the reader may already be familiar with some of these definitions, they are included here to support the formalizations in later sections, particularly in Section 2.4 and Chapter 3.

We start by defining some concepts that will be used multiple times throughout this section.

Definition 2.1 (Binary Operation)**.** Let G be a set. A *binary operation* on G is a function $\circ : G \times G \to G$ that assigns to each pair $(a, b) \in G \times G$ the element $a \circ b \in G$. We denote the set G equipped with operation $\circ$ by $(G, \circ)$.

Definition 2.2. Let G be a set with a binary operation '$\circ$'. The following are important notions.

- *Associativity*: For all $a, b, c \in G$, $a \circ (b \circ c) = (a \circ b) \circ c$
- *Identity element* $\exists e \in G$ such that $e \circ a = a \circ e = a, \ \forall a \in G$
- *inverse element* $\forall a \in G$, there exists $a^{-1} \in G$ such that $a \circ a^{-1} = a^{-1} \circ a = e$.
- *Commutativity*: $\forall a, b \in G, a \circ b = b \circ a$.

Associativity, existence of an identity element and existence of an inverse element are called the *group axioms.* When we work with the binary operations addition and multiplication, we define the following.

- *Distributivity:* $\forall a, b, c \in G, a \cdot (b + c) = a \cdot b + a \cdot c$.

We begin by defining several fundamental algebraic structures that serve as the foundation for many others.

Definition 2.3 (Groups)**.** A group is a set G with an operation $G \times G \to G$, denoted $(a, b) \mapsto a \circ b$, such that the group axioms are met. A group G is called *commutative* or *abelian* if it satisfies commutativity.

Definition 2.4 (Fields)**.** A field is a set $\mathbb{K}$ equipped with two operations called addition and multiplication, such that $\mathbb{K}, +$ and $\mathbb{K}, \cdot$ are abelian groups and distributivity holds. The set of all above axioms combined is called the *field axioms.*

Example 2.5 (Real numbers)**.** The real numbers with the well-known operations of addition and multiplication form a field. Indeed, given $a \in \mathbb{R}$ we have:

- Addition and multiplication satisfy commutativity and associativity
- Additive inverse $-a$
- Multiplicative inverse $1/a \ \forall a \neq 0 \in \mathbb{R}$
- Additive and multiplicative identity $a + 0 = a$ and $a \cdot 1 = a$
- Multiplication distributes over addition

The following definition serves as the general structure used throughout the thesis.

Definition 2.6 (Vector spaces)**.** Let $\mathbb{K}$ be a field. A *vector space* V over $\mathbb{K}$ is a set equipped with

- an operation called *vector addition* $+ : V \times V \to V$, and
- an operation called *scalar multiplication* $\cdot : \mathbb{K} \times V \to V$,

such that $(V, +)$ is an abelian group, and the following properties are satisfied:

- *Associativity with scalar multiplication:* $a(bv) = (ab)v$
- *Identity element of scalar multiplication:* $1 \cdot v = v$, where $1 \in \mathbb{K}$
- *Distributivity of scalar multiplication over vector addition:* $a(u + v) = au + av$
- *Distributivity of scalar multiplication over field addition:* $(a+b)v = av+bv$

Example 2.7 ($\mathbb{R}^n$ is a vector space)**.** Let $\mathbb{R}^n$ be the set of all n-tuples $(x_1, \ldots, x_n)$ of real numbers. With component-wise addition and scalar multiplication, $\mathbb{R}^n$ forms a vector space over $\mathbb{R}$. Indeed, let $x, y \in \mathbb{R}^n$, then $x_i, y_i \in \mathbb{R}$ for all $i \in \{1, ..., n\}$ and

$$(x_1, ..., x_n) + (y_1, ..., y_n) = (x_1 + y_1, ..., x_n + y_n) \tag{1}$$

and similarly, for $a \in \mathbb{R}$ we have

$$a \cdot (x_1, ..., x_n) = (ax_1, ..., ax_n) \tag{2}$$

We say the vector space $\mathbb{R}^n$ is defined over the field $\mathbb{R}$.

We need the following definition to build generalization of a vector space.

Definition 2.8 (Ring)**.** A ring is a set $\mathcal{R}$ equipped with addition and multiplication such that $(\mathcal{R}, +)$ is an abelian group. Moreover, multiplication is associative, it has a multiplicative identity and multiplication distributes over addition.

The following is the generalization of vector space.

Definition 2.9 (Modules)**.** Suppose that $\mathcal{R}$ is a ring with multiplicative identity 1. A $\mathcal{R}-$module M is a vector space in which the field $\mathbb{K}$ is replaced by a ring.

Example 2.10 (Vector spaces are modules)**.** Let V be a vector space over the field $\mathbb{K}$, then V is a $\mathbb{K}-$module. Indeed, a vector space over a field satisfies all axioms of a $\mathbb{K}-$module, since the axioms of vector spaces and modules coincide in this case. And since $\mathbb{K}$ is a field, it is also a commutative ring. This follows directly from the axioms in the definition. Thus, a vector space over a field $\mathbb{K}$ is a $\mathbb{K}-$module.

Example 2.11. $\mathbb{R}^n$ is a $\mathbb{R}-$module. Indeed, in Example 2.7 we saw that $\mathbb{R}^n$ is a vector space with field $\mathbb{R}$. So by Example 2.10 we can conclude that $\mathbb{R}^n$ is a $\mathbb{R}-$module.

The following definition introduces a notion of size within structures, and plays an important role in the structures that follow.

Definition 2.12 (Norm)**.** A *norm* on a vector space V over a field $\mathbb{K}$ is a function $||\cdot|| : V \to \mathbb{R}_{\geqslant 0}$ satisfying the following properties for all $x, y \in V$ and scalars $a \in \mathbb{K}$.

- *Non-negativity:* $||x|| \geqslant 0$
- *Definiteness:* $||x|| = 0 \iff x = 0$
- *Homogeneity:* $||ax|| = |a| \cdot ||x||$
- *Triangle inequality:* $||x + y|| \leqslant ||x|| + ||y||$.

By combining the first fundamental structures in this section and the properties of the norm, we are able to construct structures suitable for Schwartz functions (see Section 3.3).

Definition 2.13 (Normed groups)**.** A *normed group* is an abelian group G with a norm $||\cdot|| : G \to \mathbb{R}_{\geqslant 0}$. Note that homogeneity is not defined in this setting.

Example 2.14 ($\mathbb{R}^n$ is a normed group)**.** Let $G = \mathbb{R}^n$ be the set of $n-$tuples of real numbers with as operation vector $(+)$ addition and the *Euclidean norm*:

$$||\mathbf{x}|| = \sqrt{x_1^2 + ... + x_n^2}.$$

Then $(\mathbb{R}^n, +, ||\cdot||)$ is a normed group. Indeed, it satisfies the abelian group properties:

- $\mathbb{R}^n$ is closed under addition
- Addition is associative and commutative
- The identity is $\mathbf{0}$
- Every $\mathbf{x}$ has inverse element $-\mathbf{x}$

Moreover, it satisfies all norm axioms:

- $||x|| \geqslant 0$
- $||x|| = 0 \iff x = 0$
- $||x + y|| \leqslant ||x|| + ||y||$. Taking squares on both sides and rewriting gives for the left hand side:

  $$||x + y||^2 = (x + y) \cdot (x + y) = x \cdot x + 2x \cdot y + y \cdot y = ||x||^2 + 2x \cdot y + ||y||^2.$$

  The right-hand side becomes:

  $$(||x|| + ||y||)^2 = ||x||^2 + 2||x|| \cdot ||y|| + ||y||^2.$$

  Subtracting $||x||^2 + ||y||^2$ from both sides yields:

  $$x \cdot y \leqslant ||x|| \cdot ||y||$$

  which is the *Cauchy-Schwarz inequality* (Lemma 3.3 of [9]).

Therefore, $\mathbb{R}^n$ is a normed group.

Definition 2.15 (Normed vector spaces)**.** A *normed vector space* is a vector space equipped with a norm.

Example 2.16 ($\mathbb{R}^n$ over $\mathbb{R}$ with Euclidean norm is a normed vector space)**.** Let $\mathbf{x} \in \mathbb{R}^n$ and $a \in \mathbb{R}$. Then $||a \cdot \mathbf{x}|| = |a| \cdot ||\mathbf{x}||$. Indeed,

$$||a \cdot \mathbf{x}|| = \sqrt{(ax_1)^2 + ... + (ax_n)^2} = \sqrt{a^2(x_1^2 + ... + x_n^2)} = |a|\sqrt{x_1^2 + ... + x_n^2} = |a| \cdot ||\mathbf{x}||.$$

So $\mathbb{R}^n$ satisfies all normed vector space axioms (by previous examples and homogeneity).

While norms provide a way to measure size of elements, the following definitions provides a way to measure distance between elements.

Definition 2.17 (Metric spaces)**.** A *metric space* is a pair (X, d) where X is a set and $d : X \times X \to \mathbb{R}_{\geqslant 0}$ is a function that satisfies the following for all $x, y, z \in X$: non-negativity, definiteness, the triangle inequality, and

- *Symmetry:* $d(x, y) = d(y, x)$.

d is called the *metric* or the *distance function.*

Example 2.18 (Euclidean space)**.** Let $X = \mathbb{R}^n$, the metric induced by the Euclidean norm $d(x, y) = ||x - y||$ gives rise to a metric space. More generally, normed vector spaces are automatically metric spaces when using the metric induced by the norm.

Some metrics behave differently from our intuitive understanding of distance. This should be accounted for when working with general notions of metric spaces (see Section 3.3).

Example 2.19 (Discrete metric)**.** Let X be any set. The metric

$$d(x, y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{if } x \neq y \end{cases}$$

is called the *discrete metric.*

We introduce the following to be able to abstract the idea of metric spaces. Moreover, the definition is used in Section 2.4.

Definition 2.20 (Open and closed sets)**.** Let (X, d) be a metric space.

- A set $U \subseteq X$ is *open* if for every $x \in U$, there exists $\epsilon > 0$ such that the open ball
$$B_\epsilon(x) = \{y \in X : d(x, y) < \epsilon\}$$
is contained in U.
- A set $F \subseteq X$ is *closed* if its complement $X \backslash F$ is open.

The following is an even broader framework than metric space and is central in the study of continuity [27]. We depend on this notion in Section 2.4 and Section 3.3.

Definition 2.21 (Topological spaces)**.** Let X be a set. A *topology* $\mathcal{T}$ on X is a collection of subsets of X, called open sets, such that:

- $\varnothing \in \mathcal{T}$ and $X \in \mathcal{T}$
- The union of any collection of sets in $\mathcal{T}$ is also in $\mathcal{T}$
- The intersection of any finite collection of sets in $\mathcal{T}$ is also in $\mathcal{T}$

A *topological space* is a pair $(X, \mathcal{T})$, where X is a set and $\mathcal{T} \subseteq \mathcal{P}(X)$ is a collection of subsets of X that satisfies the 3 axioms.

Example 2.22 (Metric spaces are topological spaces)**.** Given a metric space (X, d). Let $\mathcal{T}$ be collection of open sets in X. $\mathcal{T}$ satisfies the axioms in 2.21. Indeed,

- There exists no $x \in \varnothing$, so $\varnothing \in \mathcal{T}$. And $X \in \mathcal{T}$, since for every $x \in X$ and every $r > 0$, the open ball $B_r(x) \subseteq X$ by definition.

- Suppose we have a collection of open sets $\{U_i\}$ with indexes i. Let U be the union of all U_i. Then any point $x \in U$ lies in some U_i. Since U_i open there exists $r > 0$ such that $B_r(x) \subseteq U_i \subseteq U$. So U is open.

- Let D be a finite intersection of n open sets F_i. Then for $x \in F$ there exists radii $r_1, ..., r_n$ such that $B_{r_i} \subseteq F_i$. Let $r = \min\{r_1, ..., r_n\}$, then $B_r(x) \subseteq F_i$ for all i. So $B_r(x) \subseteq F$. Thus, the collection of open sets in a metric space is a topology. We conclude that every metric space defines a topological space.

2.2 Fréchet derivatives

Unless stated otherwise, all results from this subsection are from [10]. In classical analysis, the derivative of a function $f : \mathbb{R}^n \to \mathbb{R}$ is usually understood as the gradient which captures how the function changes locally. This thesis works with a more general version of the derivative called *Fréchet derivatives.* This is a coordinate-free approach to the best linear approximation of a function at a point, and it is also compatible with the structure of general normed vector spaces, not only $\mathbb{R}^n$. We use E and F to denote normed vector spaces (see Definition 2.15).

Definition 2.23. Let $\lambda : E \to F$ be a linear map with E and F normed vector spaces. λ is *bounded* if there exists $C \geqslant 0$ such that

$$||\lambda(x)||_F \leqslant C||x||_E, \quad \forall x \in E.$$

Definition 2.24. Let $f : U \to F$ be a function where U is an open subset of E. We say f is Fréchet differentiable at $x \in U$ if there exists a bounded and linear operator $A : E \to F$ such that

$$\lim_{h \to 0} \frac{||f(x+h) - f(x) - Ah||}{||h||} = 0 \tag{3}$$

We say that A is the derivative of f at x, denoted by $Df(x)$.

Remark 2.25. Since the Fréchet derivative is the best linear approximation of $f(x+h)$ near x, we can write

$$f(x+h) \approx f(x) + Ah,$$

or

$$f(x+h) = f(x) + Ah + o(h). \tag{4}$$

The Fréchet and the classical derivative have similar properties such as the chain rule, uniqueness, linearity of the derivative and more. In $\mathbb{R}^n$, the two notions coincide. We begin by proving some of these properties.

Theorem 2.26 (Uniqueness)**.** If $f : E \to F$ is Fréchet differentiable at x, then the mapping A in Definition (2.24) is uniquely defined.

Proof. Suppose A_1 and A_2 are two bounded linear maps that satisfy Equation 3. Then, for each $\epsilon > 0$ there exists a $\delta > 0$ such that

$$||f(x+h) - f(x) - A_i h|| < \epsilon ||h||, \;\; i \in \{1, 2\}$$

whenever $||h|| < \delta$. By the triangle inequality, $||A_1 h - A_2 h|| < 2\epsilon ||h||$ whenever $||h|| < \delta$. By linearity of A_1 and A_2, the difference $A_1 - A_2$ is linear. Thus, we can scale h arbitrarily. Indeed, let $h = tv$ for some $v \in E$, and scalar $t > 0$ such that $||tv|| < \delta$. Then,

$$||(A_1 - A_2)(tv)|| = |t| \cdot ||(A_1 - A_2)(v)|| \leqslant 2\epsilon |t| \cdot ||v||.$$

Dividing by $|t|$ yields

$$||(A_1 - A_2)(v)|| \leqslant 2\epsilon \cdot ||v||.$$

Since we chose $\epsilon > 0$ arbitrarily and the inequality holds for all $v \in \mathbb{R}^n$, we conclude that $A_1 = A_2$. □

Theorem 2.27 (Chain rule)**.** If f is differentiable at x and g is differentiable at $f(x)$, then $(g \circ f)'(x)$ is differentiable at x, and

$$(g \circ f)'(x) = g'(f(x)) \circ f'(x).$$

Proof. Let $h, k \in \mathbb{R}^n$. Define

$$\begin{aligned} F := g \circ f, A := f'(x), y :&= f(x), B := g'(y), \text{ and} \\ o_1(h) :&= f(x+h) - f(x) - Ah \\ o_2(k) :&= g(y+k) - g(k) - Ak \\ \phi(kh) :&= Ah + o_1(h) \end{aligned}$$

Note that by Equation (3) we have that $o_1(h)$ and $o_2(h)$ are little-o. We show that $F'(x) = BA$. We have

$$\begin{aligned} F(x+h) - F(x) - BAh &= g(f(x+h)) - g(f(x)) - BAh \\ &= g(f(x) + Ah + o_1(h)) - g(y) - BAh \\ &= g(y + \phi(h)) - g(y) - BAh \\ &= g(y) + B\phi(h) + o_2(\phi(h)) - g(y) - BAh \\ &= B[Ah + o_1(h)] + o_2(\phi(h)) - BAh \\ &= B(o_1(h)) + o_2(\phi(h)). \end{aligned}$$

Thus, it follows that

$$\begin{aligned} ||F(x+h) - F(x) - BAh|| &= ||B(o_1(h)) + o_2(\phi(h))|| \\ &\leqslant ||B|| \, ||o_1(h)|| + ||o_2(\phi(h))||. \end{aligned}$$

Now, since the sum of two little-o functions is again little-o, the conclusion follows. □

Indeed, the derivative behaves like as expected from the familiar definition. Let us check this by doing some simple examples.

Example 2.28 (Constant functions)**.** Let X, Y be normed vector spaces. Let $f : X \to Y$ be $f(x) = y_0$, where $y_0 \in F$ a fixed element. Thus f is a constant map and we would expect $f'(x) = 0$. Indeed,

$$lim_{h\to 0}\frac{||f(x+h) - f(x) - 0||}{||h||} = lim_{h\to 0}\frac{||0||}{||h||} = 0$$

.

Example 2.29. Let $X = Y = \mathbb{R}$. Let f be a function whose derivative in the usual sense at x is $\lambda \in \mathbb{R}$. Then the Fréchet derivative of f at x is the linear map $Ah = \lambda h$, because

$$lim_{h\to 0}\frac{|f(x+h) - f(x) - \lambda h|}{|h|} = lim_{h\to 0}|\frac{f(x+h) - f(x)}{h} - \lambda| = 0$$

Next, we introduce a theorem that helps us to work with the Fréchet derivative in the appropriate setting for Schwartz Functions.

Theorem 2.30. If f is bounded in a neighborhood of x and if a linear map A has the property in Equation (3), then A is a bounded linear map; in other words, A is the Fréchet derivative of f at x.

Proof. Choose $\delta > 0$ so that whenever $||h|| \leqslant \delta$ we will have

$$||f(x+h)|| \leqslant M$$

and

$$||f(x+h) - f(x) - Ah|| \leqslant ||h||.$$

Then, for $||h|| \leqslant \delta$, we have

$$||Ah|| \leqslant ||f(x+h) - f(x)|| + ||h|| \leqslant 2M + \delta.$$

For $||u|| \leqslant 1, ||\delta u|| \leqslant \delta$, whence

$$||A(u)|| = \frac{1}{\delta}||A(\delta u)|| \leqslant \frac{1}{\delta}(2M + \delta) = \frac{2M + \delta}{\delta}.$$

Thus $||A|| \leqslant (2M + \delta)/\delta < \infty$. □

2.3 Multilinear maps

We have seen in Section 2.2 that a Fréchet derivative is a bounded linear map. It is not surprising that higher-order derivatives are given by *multilinear maps*, which are linear in each argument separately. In the setting defined by $E = \mathbb{R}^n$ and $F = \mathbb{R}^m$, we have the following theorem [10].

Theorem 2.31. Let $f : \mathbb{R}^n \to \mathbb{R}^m$, and let $f_1, ..., f_m$ be the component functions of f. If all partial derivatives $\frac{\partial f_i}{\partial x_j}$, $i = 1, 2, .., m$, $j = 1, 2, ..., n$, exist in a neighborhood of x and are continuous at x, then $f'(x)$ exists, and

$$(f'(x)h)_i = \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j}(x) \cdot h_j, \;\; h = (h_1, ..., h_n)^T \in \mathbb{R}^n.$$

That is, the Fréchet derivative of f is given by the Jacobian matrix $J \in \mathbb{R}^{m\times n}$ of f at x, $J_{ij} = \frac{\partial f_i}{\partial x_j}(x)$.

Proof. Since for all i, the partial derivatives $\frac{\partial f_i}{\partial x_j}$ exist in a neighborhood of x and are continuous at x, each component function $f_i : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable at x. Hence, by standard results from multivariable analysis (see [17]), each f_i is Fréchet differentiable at x with gradient

$$\nabla f_i(x) = \Big(\frac{\partial f_i}{\partial x_1}(x), ..., \frac{\partial f_i}{\partial x_n}(x)\Big).$$

Therefore, for every $h \in \mathbb{R}^n$, we have

$$\sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j} \cdot h_j = f_i(x+h) - f_i(x) + o_i(h).$$

Rearranging terms gives

$$o_i(h) = f_i(x+h) - f_i(x) - \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j} \cdot h_j.$$

But, since $\lim_{h\to 0} \frac{|o_i(h)|}{||h||} = 0$ we have

$$\lim_{h\to 0} \frac{1}{||h||}\left| f_i(x+h) - f_i(x) - \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j} \cdot h_j \right| = 0. \tag{5}$$

By definition of the Euclidean norm

$$\frac{1}{||h||^2}||f(x+h) - f(x) - Jh||^2 = \frac{1}{||h||^2}\sum_{i=1}^{m}\left| f_i(x+h) - f_i(x) - \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j}(x) \cdot h_j \right|^2.$$

Since the left-hand side is non-negative and the right hand side converges to zero by (5), the result follows. □

This Jacobian matrix represents the coordinate representation of the Fréchet derivative, which is a linear map where each row corresponds to the gradient of a component function. Now, since the Fréchet derivative is the best linear approximation of f near some point $x \in \mathbb{R}^n$, we can write

$$f(x+h) \approx f(x) + J(x)(h).$$

We say that h is the *direction vector*, as it specifies the direction in which the change of the function is measured. When we choose h to be a standard basis vector, we obtain the partial derivatives. Indeed, let e_i be a standard basis vector given by $e_i = \begin{bmatrix} 0 & ... & 0 & 1 & 0 & ... & 0 \end{bmatrix}^T$ with the 1 on the ith place. Now applying the linear map to the e_i corresponds to the matrix vector product of the Jacobian and the basis vector. Thus,

$$Df(x)(e_i) = \frac{\partial f}{\partial x_i}(x).$$

Now, the second derivative is just the derivative of the map $Df(x)$. It represents the rate of change of $Df(x)(h)$ in some direction k, denoted $D^2 f(x)(h,k)$. This is a *bilinear map* from $\mathbb{R}^n \times \mathbb{R}^n$ onto $\mathbb{R}^m$; it is linear in each argument separately. In the case $f : \mathbb{R}^n \to \mathbb{R}$, $D^2 f(x)$ is called the *Hessian matrix*. We refer the reader to [27] for the details. If we take $h = e_i$ and $k = e_j$, we obtain

$$D^2 f(x)(e_i, e_j) = \frac{\partial^2 f}{\partial x_j \partial x_i}(x).$$

More generally, the kth derivative of f at x, denoted

$$D^k f(x) : \underbrace{\mathbb{R}^n \times \cdots \times \mathbb{R}^n}_{k \text{ times}} \to \mathbb{R}^m$$

represents a k-linear map; it is linear in each of the k arguments separately. Again, using standard basis vectors $e_1, ...e_k$ yields

$$D^k f(x)(e_1, ..., e_k) = \frac{\partial^k f}{\partial x_1 ... \partial x_k}(x).$$

Remark 2.32 (Notation)**.** Let $f : \mathbb{R}^n \to \mathbb{R}^m$. The first derivative is a linear map. We say $Df(x) \in \operatorname{Lin}(\mathbb{R}^n, \mathbb{R}^m)$. So, the function Df, which maps each point $x \in \mathbb{R}^n$ to its derivative, maps $\mathbb{R}^n$ to $\operatorname{Lin}(\mathbb{R}^n, \mathbb{R}^m)$. Similarly, for the second derivative we have $D^2 f(x) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^m$, and we write $D^2 f : \mathbb{R}^n \to \operatorname{Lin}(\mathbb{R}^n, \operatorname{Lin}(\mathbb{R}^n, \mathbb{R}^m))$. For $\operatorname{Lin}(\mathbb{R}^n, \operatorname{Lin}(\mathbb{R}^n, \mathbb{R}^m))$ we use the abbreviation $\operatorname{Lin}_2(\mathbb{R}^n \times \mathbb{R}^n, \mathbb{R}^m)$. Higher-order derivatives work similarly. The space of continuous linear maps is denoted $\mathcal{L}(\mathbb{R}^n, \mathbb{R}^m)$. Similar notation holds in this case. Lastly, the maps $D^k f(x)$ are sometimes denoted $f^{(k)}(x)$, which aligns with the notation often used in the literature.

We now work our way to *symmetry* of higher-order derivatives, which will become relevant in Section 3.4. Let us start with some definitions.

Definition 2.33. Let E and F be vector space over a field $\mathbb{K}$ and let $T : \underbrace{E \times ... \times E}_{\text{d times}} \to F$ be a $d-$linear map. Let S_d be the set of all possible permutations of $\{1, ..., d\}$. T is called *symmetric* if for all permutations $\sigma \in S_d$, we have

$$T(x_1, ..., x_n) = T(x_{\sigma(1)}, ..., x_{\sigma(d)}).$$

Definition 2.34 (Operator norm)**.** The continuous linear mappings between two normed vector spaces E and F form a vector space $\mathcal{L}(E,F)$. For $\phi \in \mathcal{L}(E,F)$ let

$$||\phi||_{\mathcal{L}(E,F)} = \sup_{||x||_E \leqslant 1} ||\phi(x)||_F.$$

With this norm, $\mathcal{L}(E,F)$ becomes a normed vector space.

Remark 2.35. Note that for $x \in E$, we have

$$||\phi(x)||_F \leqslant ||\phi||_{\mathcal{L}(E,F)} ||x||_E.$$

Indeed, define the unit vector $\vec{x} := \frac{x}{||x||_E}$. By definition of the supremum we have

$$||\phi(\vec{x})||_F \leqslant ||\phi||_{\mathcal{L}(E,F)}.$$

By multiplying both sides with $||x||_E$ we get

$$||\phi(x)||_F = ||\phi(||x||_E \cdot \vec{x})||_F = ||x||_E ||\phi(\vec{x})||_F \leqslant ||x||_E ||\phi||_{\mathcal{L}(E,F)} \tag{6}$$

We state the following lemma without proof (for details, the reader is referred to Corollary 3.2 of [11]).

Lemma 2.36 (Mean value inequality)**.** Let E and F be normed vector spaces, O an open subset of E and $f : O \to F$ differentiable on O. If the segment $[a,b] \subset O$, then

$$||f(b) - f(a)||_F \leqslant \sup_{x \in (a,b)} |f'(x)|_{\mathcal{L}(E,F)} ||b - a||_E.$$

We are now ready for the main results of this section. These are drawn from [11].

Theorem 2.37. Let E and F be normed vector spaces, O an open subset of E and f a mapping from O into F. If f is 2-differentiable in O and $a \in O$, then $f^{(2)}(a)$ is a symmetric bilinear mapping.

Proof. For $h, k \in E$ small, define

$$\Delta(h,k) = f(a+h+k) - f(a+k) - f(a+h) + f(a)$$

Then, by adding and subtracting $f'(a+h)(k)$ and $f'(a)(k)$ we get

$$\begin{aligned}\Delta(h,k) - f^{(2)}(a)(h,k) =& f(a+h+k) - f(a+k) - f'(a+h)k + f'(a)k \\ &- (f(a+h) - f(a)) \\ &+ (f'(a+h) - f'(a) - f^{(2)}(a)(h))k.\end{aligned}$$

Now fix h and set

$$H(k) = f(a+h+k) - f(a+k) - f'(a+h)k + f'(a)k.$$

The triangle inequality yields

$$\begin{aligned}||\Delta(h,k) - f^{(2)}(a)(h,k)||_F \leqslant{}& ||H(k) - H(0)||_F \\ &+ ||(f'(a+h) - f'(a) - f^{(2)}(a)(h))(k)||_F\end{aligned}$$

Since f is 2-differentiable in O, the function H is differentiable, and so is $H(k) - H(0)$. Therefore, the first term in the right-hand side can be bounded by applying Lemma 2.36. The second term is a bounded linear map, so by Equation (6) we get:

$$\begin{aligned}||\Delta(h,k) - f^{(2)}(a)(h,k)||_F \leqslant{}& ||H(k) - H(0)||_F \\ &+ ||(f'(a+h) - f'(a) \; - f^{(2)}(a)(h))(k)||_F \\ \leqslant{}& \sup_{0\leqslant\lambda\leqslant 1} |H'(\lambda k)|_{\mathcal{L}(E,F)}||k||_E \\ &+ |f'(a+h) - f'(a) - f^{(2)}(a)(h)|_{\mathcal{L}(E,F)}||k||_E\end{aligned}$$

To estimate $H'(\lambda k)$, consider the derivative of H in the direction $u = \lambda k$:

$$\begin{aligned}H'(u) &= f'(a+h+u) - f'(a+u) - f'(a+h) + f'(a) \\ &= f'(a+h+u) - f'(a) - f^{(2)}(a)(h+u) \\ &\quad - (f'(a+u) - f'(a) - f^{(2)}(a)u) \\ &\quad - (f'(a+h) - f'(a) - f^{(2)}(a)h),\end{aligned}$$

where we use the approximation terms for the derivatives in the second equality for the multivariable case (see Equation (4)). Now taking the norm yields

$$\begin{aligned}|H'(u)|_{\mathcal{L}(E,F)} \leqslant{}& |f'(a+h+u) - f'(a) - f^{(2)}(a)(h+u)|_{\mathcal{L}(E,F)} \\ &+ |(f'(a+u) - f'(a) - f^{(2)}(a)u)|_{\mathcal{L}(E,F)} \\ &+ |(f'(a+h) - f'(a) - f^{(2)}(a)h)|_{\mathcal{L}(E,F)}.\end{aligned}$$

Set $\epsilon > 0$. Since each term is the approximation error of the second order derivative and f' is differentiable by assumption for sufficiently small h, u, we have:

$$|H'(u)| \leqslant \epsilon||h+u||_E + \epsilon||u||_E + \epsilon||h||_E \leqslant 2\epsilon(||h||_E + ||u||_E).$$

We have

$$|H'(\lambda k)| \leqslant 2\epsilon(||h||_E + ||k||_E).$$

Thus,

$$\begin{aligned}||\Delta(h,k) - f^{(2)}(a)(h,k)||_F &\leqslant (2\epsilon(||h||_E + ||k||_E + \epsilon||h||_E)||k||_E \\ &= \epsilon(3||h||_E + 2||k||_E)||k||_E \\ &\leqslant 2\epsilon(||h||_E + ||k||_E)^2.\end{aligned}$$

By the triangle inequality,

$$||f^{(2)}(a)(h,k) - f^{(2)}(a)(k,h)||_F = ||f^{(2)}(a)(h,k) - \Delta(h,k)||_F$$
$$+||\Delta(h,k) - f^{(2)}(a)(k,h)||_F.$$

But, since $\Delta(k,h) = \Delta(h,k)$ we have

$$||f^{(2)}(a)(h,k) - f^{(2)}(a)(k,h)||_F \leqslant 4\epsilon(||h||_E + ||k||_E)^2.$$

Now, suppose $x, y \in E$. If $\alpha > 0$ small, then so are $h = \frac{\alpha}{2}\frac{x}{||x||_E}$ and $k = \frac{\alpha}{2}\frac{y}{||y||_E}$. But then

$$\frac{\alpha^2}{4||x||_2||y||_2}||f^{(2)}(a)(x,y) - f^{(2)}(a)(y,x)||_F \leqslant 4\epsilon\alpha^2,$$

or

$$||f^{(2)}(a)(x,y) - f^{(2)}(a)(y,x)||_F \leqslant 16\epsilon||x||_E||y||_E.$$

This completes the proof. □

Now, for the general case of symmetry for higher-order derivatives, we get the following corollary.

Theorem 2.38. Let E and F be normed vector spaces and O an open subset of E. Let $f : O \to F$ is k times differentiable in O and $a \in O$. Then $f^{(k)}(a)$ is symmetric.

Proof. We prove this by induction on k. The base case $k = 2$ is treated in Theorem 2.37. Suppose that the result is true up to a given k. Let us consider the case $k + 1$. Since f is $(k+1)$-differentiable at $a \in E$, $f^{(k)}$ is defined on a neighborhood V of a. By the induction hypothesis $f^{(k)}$ is a symmetric multilinear map from E^k into F for all x in V. Now since the derivative $f^{(k)}(a)$ maps into $\mathcal{L}_S(E^k; F)$ and this subspace is closed in $\mathcal{L}_k(E, F)$, Proposition 2.8 of [11] implies that the derivative $f^{(k)'}(a)$ also takes values in $\mathcal{L}_s(E^k, F)$. Thus the map is symmetric for every $x \in E$. So, if we fix the first variable of $f^{(k+1)}(a)$, the resulting $k-$linear map is symmetric. Now it remains to show that the map is symmetric in all $k+1$ arguments. Notice that $\mathcal{L}_{k+1}(E, F) = \mathcal{L}_2(E, \mathcal{L}_{k-1}(E, F))$. Set $g = f^{(k-1)}$, then $g^{(2)}(a) = f^{(k+1)}(a)$ and $g^{(2)}(a) \in \mathcal{L}_2(E, \mathcal{L}_{k-1}(E, F))$. By Theorem 2.37 we have

$$g^{(2)}(a)(x_1, x_2) = g^{(2)}(a)(x_2, x_1)$$

for $x_1, x_2 \in E$. But then

$$f^{(k+1)}(x_1, x_2, x_3, ..., x_{k+1}) = f^{(k+1)}(x_2, x_1, x_3, ..., x_{k+1})$$

for $x_1, x_2, x_3, ..., x_{k+1} \in E$. Suppose now that σ is a permutation of $k+1$ indices. Now, consider the expression

$$f^{(k+1)}(a)(x_{\sigma(1)}, ..., x_{\sigma(k+1)}).$$

If $\sigma(1) = 1$, then

$$\begin{aligned} f^{(k+1)}(a)(x_{\sigma(1)}, ..., x_{\sigma(k+1)}) &= f^{(k+1)}(a)(x_1, x_{\sigma(2)}, ..., x_{\sigma(k+1)}) \\ &= f^{(k+1)}(a)(x_1, x_2, ..., x_{k+1}). \end{aligned}$$

If $\sigma(1) \neq 1$, then by the induction hypothesis we can commute the last k variables

$$f^{(k+1)}(a)(x_{\sigma(1)}, ..., x_{\sigma(k+1)}) = f^{(k+1)}(a)(x_{\sigma(1)}, x_1, ..., x_k).$$

Now,by first commuting the first two variables and then a second commutation of the last k variables gets us

$$\begin{aligned} f^{(k+1)}(a)(x_{\sigma(1)}, x_1, ...) &= f^{(k+1)}(a)(x_1, x_{\sigma(1)}, ..., x_k) \\ &= f^{(k+1)}(a)(x_1, x_2, ..., x_{k+1}). \end{aligned}$$

The result follows by mathematical induction. □

2.4 Introduction to Schwartz functions

Unless stated otherwise, all results from this subsection are from [25]. We refer the reader unfamiliar with L^P spaces to [9]. In this section we introduce *Schwartz functions*. Schwartz functions are an important concept in functional analysis and the theory of distributions. Informally, these functions are defined by smoothness and their behaviour at infinity: they, along with all their derivatives, decay faster than the reciprocal of any polynomial at infinity. We will find out why this makes the space of all Schwartz functions very suitable for Fourier analysis. Let us start by defining Schwartz functions.

Definition 2.39. A complex-valued function $f \in C^{\infty}(\mathbb{R}^n)$ is called a Schwartz function if for every pair of multi-indices α and β there exists a positive constant $C_{\alpha,\beta}$ such that

$$\rho_{\alpha,\beta}(f) = \sup_{x \in \mathbb{R}^n} |x^{\alpha} \partial^{\beta} f(x)| = C_{\alpha,\beta} < \infty \tag{7}$$

The maps $\rho_{\alpha,\beta} : \mathcal{S}(\mathbb{R}^n) \to [0, \infty)$ are called the *Schwartz seminorms* of f. Here $\mathcal{S}(\mathbb{R}^n)$ is the set of all Schwartz functions on $\mathbb{R}^n$. When there is no ambiguity, we write $\mathcal{S}(\mathbb{R}^n)$ as $\mathcal{S}$.

We begin by proving a basic result about the Schwartz space [31].

Theorem 2.40. Let $m, n \in \mathbb{N}$ and $f \in \mathcal{S}(\mathbb{R}^m)$ $g \in \mathcal{S}(\mathbb{R}^n)$, then the function $f \times g$, i.e., $f(x_1, ..., x_m)g(x_{m+1}, ..., x_{m+n}) \in \mathcal{S}(\mathbb{R}^{m+n})$.

Proof. Note that

$$|(x, y)^{(\alpha,\beta)} \partial_x^{\gamma} \partial_y^{\mu} (f \times g)(x, y)| = |x^{\alpha} \partial^{\gamma} f(x)||y^{\beta} \partial^{\mu} g(y)|,$$

for every $(x, y) \in \mathbb{R}^m \times \mathbb{R}^n$,$f \in \mathcal{S}(\mathbb{R}^m), g \in \mathcal{S}(\mathbb{R}^n)$ and all multi-indices α, β. Thus, by combining the norms, we see that the resulting norm is finite. Therefore, $f \times g$ is in $\mathcal{S}(\mathbb{R}^{m+n})$ □

We now prove an unconventional characterization of the Schwartz functions, which will become relevant in Section 3.4.

Theorem 2.41 (Characterization)**.** Let $f : \mathbb{R}^n \to \mathbb{C}$, $f \in C^{\infty}(\mathbb{R}^n)$. Then f is a Schwartz function if and only if for every $k, n \in \mathbb{N}$, there exists $C_{k,n} > 0$, such that for all $x \in \mathbb{R}^n$ we have

$$||x||^k \cdot ||D^n f(x)|| \leqslant C_{k,n}.$$

The norm of $D^n f(x)$ is defined as in Definition 2.34.

Proof. '⇒' We want to bound $||x||^k \cdot ||D^n f(x)||$. By definition of the norm we have that

$$||D^n f(x)|| = \sup_{||v_1|| \leqslant 1, ..., ||v_n|| \leqslant 1} \left| D^n f(x)(v_1, ..., v_n) \right|,$$

with the Euclidean norm on the $v_i's$. Now, we can express each unit vector v_j as a linear combination of basis vectors: $v_j = \sum_{i=1}^n \xi_{ji} e_i$. By multilinearity, we have

$$\begin{aligned} D^n f(x)(v_1, ..., v_n) &= D^n f(x)\Big(\sum_{i_1=1}^n \xi_{1i} e_{i_1}, v_2, ..., v_n \Big) \\ &= \sum_{i_1=1}^n \xi_{1i} D^n f(x)(e_{i_1}, v_2, ..., v_n). \end{aligned}$$

Repeating this argument yields

$$\begin{aligned} D^n f(x)(v_1, ..., v_n) &= \sum_{i_1=1}^n \xi_{1i_1} ... \sum_{i_1=i}^n \xi_{ni_n} D^n f(x)(e_{i_1}, ..., e_{i_n}) \\ &= \sum_{i_1,...,i_n=1}^n \xi_{1i_1} ... \xi_{ni_n} D^n f(x)(e_{i_1}, ..., e_{i_n}). \end{aligned}$$

Each term $D^n f(e_{i_1}, ..., e_{i_n})$ equals $\partial^\beta f(x)$ for some $|\beta| = n$, depending on the multiplicities of $i_1, ..., i_n$. Since for all j, $||v_j||^2 \leqslant 1$ we have $\sum_{i=1}^n \xi_{ji}^2 \leqslant 1$. But then, $|\xi_{ji}| \leqslant 1$ and so,

$$\begin{aligned} ||D^n f(x)|| &= \sup_{||v_1|| \leqslant 1, ..., ||v_n|| \leqslant 1} \left| \sum_{i_1,...,i_n=1}^n \xi_{1i_1} ... \xi_{ni_n} \cdot \partial^\beta f(x) \right| \\ &\leqslant \sum_{i_1,...,i_n=1}^n \left| \partial^\beta f(x) \right| \\ &\leqslant n^n \cdot \max_{|\beta|=n} |\partial^\beta f(x)| < \infty. \end{aligned}$$

Here the last inequality holds as f is a Schwartz function. Moreover, we have

$$||x||^k \leqslant C_k \sum_{|\beta|=k} |x^\beta|.$$

Indeed, we can use the equivalence of norms on finite-dimensional vector spaces (see Theorem 1.34 of [36]). But then, by combining these results we get,

$$||x||^k \cdot ||D^n f(x)|| < \infty.$$

'$\Leftarrow$' Let α, β be arbitrary multi-indices and let $k = |\alpha|, n = |\beta|$. Note that

$$|x^\alpha| = \prod_{i=1}^n |x_i|^{\alpha_i} \leqslant \prod_{i=1}^n ||x||^{\alpha_i} = ||x||^{|\alpha|} = ||x||^k.$$

Moreover, $\partial^\beta f(x)$ is a component of the derivative $D^n f(x)$. Indeed, we have $\partial^\beta f(x) = D^n f(x)(\underbrace{e_1, ..., e_1}_{\beta_1 \text{ times}}, ..., \underbrace{e_n, ..., e_n}_{\beta_n \text{ times}})$. Thus, by definition of the operator norm we have:

$$\begin{aligned}|\partial^\beta f(x)| &= |D^n f(x)(e_1, ..., e_1, ..., e_n, ..., e_n)| \\ &\leqslant \sup_{||v_1|| \leqslant 1, ..., ||v_n|| \leqslant 1} ||D^n f(x)(v_1, ..., v_n)|| = ||D^n f(x)||.\end{aligned}$$

Now combining these two observations gives the bound

$$|x^\alpha \partial^\beta f(x)| \leqslant ||x||^k ||D^n f(x)|| \leqslant C$$

with $C > 0$. □

Example 2.42. The function $f(x) = e^{-|x|^2}$ is in $\mathcal{S}(\mathbb{R}^n)$. Indeed, f is smooth on all of $\mathbb{R}^n$ because exponentials are infinitely differentiable. Moreover, derivatives of f are combinations of polynomials times $e^{-|x|^2}$. Now, since $e^{-|x|^2}$ decays faster than any polynomial grows, this function is indeed a Schwartz function.

To develop the tools needed to prove continuity of the Fourier transform (Theorem 2.58), we show that the Schwartz space is a *topological vector space*. In fact, we can show that it is a *locally convex space*, which allows us to use boundedness to show continuity. We begin by stating definitions [14].

Definition 2.43 (Topological vector space)**.** A topological vector space (TVS) is a vector space X (Definition 2.6) X together with a topology $\mathcal{T}$ (Definition 2.21 such that with respect to this topology

1. The map of $X \times X \to X$ defined by $(x, y) \mapsto x + y$ is continuous,
2. The map op $\mathbb{K} \times X \to X$ defined by $(\alpha, x) \mapsto \alpha x$ is continuous,

where the vector space X is defined over the field $\mathbb{K}$ (Definition 2.4 and $\alpha \in \mathbb{K}$.

Definition 2.44 (Locally convex space)**.** A locally convex space is a TVS whose topology is defined by a family of seminorms ϱ such that $\bigcap_{\rho \in \varrho} \{x : \rho(x) = 0\} = \{0\}$.

Remark 2.45. Let X be a vector space and ϱ a family of seminorms. The topology defined ϱ consists of all sets that can be written as unions of sets of the form

$$\{x : p(x - x_0) < \epsilon\},$$

where $p \in \varrho, x_0 \in X$, and $\epsilon > 0$. We refer to [14] for details.

Example 2.46. We show that the Schwartz space with the topology defined by the countable family of Schwartz seminorms is a locally convex space. Firstly, note that addition in the seminorm topology is continuous. Let $f, g \in \mathcal{S}$ and α, β multi-indices. Then,

$$\rho_{\alpha,\beta}(f + g) = \sup_{x \in \mathbb{R}^n} \left| x^\alpha (\partial^\beta f(x) + \partial^\beta g(x)) \right|.$$

Using the triangle inequality, we obtain:

$$\rho_{\alpha,\beta}(f+g) = \sup_{x\in\mathbb{R}^n} \left|x^\alpha \partial^\beta (f+g)(x)\right| \leqslant \sup_{x\in\mathbb{R}^n} \left|x^\alpha \partial^\beta f(x)\right| + \sup_{x\in\mathbb{R}^n} \left|x^\alpha \partial^\beta g(x)\right|$$
$$\rho_{\alpha,\beta}(f) + \rho_{\alpha,\beta}(g).$$

Thus, we control the seminorm of $f+g$ by the sum of the seminorms of f and g. Moreover, multiplication is continuous. Indeed, let $\lambda \in \mathbb{R}$. Then:

$$\rho_{\alpha,\beta}(\lambda f) = |\lambda| \cdot \rho_{\alpha,\beta}(f).$$

Thus, the seminorm of λf scales continuously with λ. We conclude that both operations are continuous and thus that $\mathcal{S}$ is a topological vector space. It remains to show that

$$\bigcap_{\rho\in\varrho} \{f \in \mathcal{S} : \rho_{\alpha,\beta}(f) = 0\} = \{0\},$$

where ϱ is the family of seminorms. Note that ϱ is countable, because the set of all multi-indices is countable. We have, $\rho_{\alpha,\beta}(f) = 0$ if $\sup_{x\in\mathbb{R}^n} |x^\alpha \partial^\beta f(x)| = 0$, so $|x^\alpha \partial^\beta f(x)| = 0$ for all x. But then, $d^\beta f = 0$ for all β. So, then we must have that $f = 0$. We conclude that $\mathcal{S}$ is a locally convex space.

The space is equipped with the following notion of convergence.

Definition 2.47. Let f_k, f be in $\mathcal{S}(\mathbb{R}^n)$ for $k = 1, 2, ...$ We say that the sequence f_k converges to f in $\mathcal{S}(\mathbb{R}^n)$ if for all multi-indices α and β we have

$$\rho_{\alpha,\beta}(f_k - f) = \sup_{x\in\mathbb{R}^n} |x^\alpha(\partial^\beta(f_k - f))(x)| \to 0 \quad \text{as} \quad k \to \infty.$$

Now, we start by defining the Fourier transform in $\mathcal{S}$ and show that it behaves 'nicely' in this setting: it is a *homeomorphism* of the Schwartz class. That is, the Fourier transform maps the Schwartz space onto itself, is bijective, and it is a continuous linear mapping. This means the structure of the space is preserved under the transform [1].

Definition 2.48. Let $f \in \mathcal{S}$, we define for $\xi \in \mathbb{R}^n$

$$\hat{f}(\xi) = \int_{\mathbb{R}^n} f(x) e^{-2\pi i x\cdot\xi} dx.$$

We call $\hat{f}$ the Fourier transform of f.

A classic example is that the Gaussian is its own Fourier transform.

[1]When introducing the Fourier transform in L^p space convergence issues arise. Therefore, it fails to be a homeomorphism [25].

Example 2.49. Let $f(x) = e^{-\pi x^2}$[2]. The fact that f is smooth is trivial. Moreover, all derivatives of f are the product of a polynomial and a negative exponential. Therefore, $f \in \mathcal{S}$. By definition,

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \cdot \xi} dx = \int_{-\infty}^{\infty} e^{-\pi(x^2 + 2ix\cdot\xi)} dx.$$

Now, completing the square gives us

$$\hat{f}(\xi) = e^{\pi \xi^2} \int_{-\infty}^{\infty} e^{-\pi(x+i\xi)^2} dx.$$

Substitution of $u = x + i\xi$, $du = dx$ yields the integral term

$$\int_{-\infty}^{\infty} e^{-\pi u^2} du = 1.$$

Therefore,

$$\hat{f}(\xi) = e^{-\pi \xi^2},$$

We conclude that f is its own Fourier Transform.

We now prove some important properties of the Fourier transform. These proofs are adapted from Section 3.3 of [35].

Theorem 2.50. Let $f \in S$, $x, \xi \in \mathbb{R}^n, \alpha$ a multi-index. We have

1. $(\partial^\alpha f)^\wedge(\xi) = (2\pi i \xi)^\alpha \cdot \hat{f}(\xi)$
2. $(\partial^\alpha \hat{f})(\xi) = ((-2\pi i x)^\alpha f)^\wedge(\xi)$

Proof. Property (1): We have

$$(\partial^\alpha f)^\wedge(\xi) = \int_{\mathbb{R}^n} (\partial^\alpha f)(x) e^{-2\pi i x\cdot\xi} dx.$$

Now, since $f \in \mathcal{S}$, at the boundaries all terms vanish. We can apply integration by parts for each component α_j times with respect to x_j. Thus,

$$\begin{aligned}(\partial^\alpha f)^\wedge(\xi) &= (-1)^{|\alpha|} \int_{\mathbb{R}^n} f(x)(-2\pi i \xi)^\alpha e^{-2\pi i x\cdot\xi} dx \\ &= (2\pi i \xi)^\alpha \hat{f}(\xi)\end{aligned}$$

Property (2): We prove this using the (correct) assumption that the interchange of the derivative and integral is valid because $f \in \mathcal{S}$.

$$\begin{aligned}\partial^\alpha \hat{f}(\xi) &= \partial^\alpha \int_{\mathbb{R}^n} f(x) e^{-2\pi i x\cdot\xi} dx \\ &= \int_{\mathbb{R}^n} f(x) \cdot \partial^\alpha \big(e^{-2\pi i x\cdot\xi}\big), dx \\ &= \int_{\mathbb{R}^n} f(x) \cdot (-2\pi i x)^\alpha e^{-2\pi i x\cdot\xi}, dx \\ &= ((-2\pi i x)^\alpha f(x))^\wedge(\xi).\end{aligned}$$

[2]Sometimes refered to as the *Standardized Gaussian*

□

Remark 2.51. For a more rigorous proof of Property (2), we refer the reader to [25].

Property (1) tells us that taking a derivative in the spatial domain becomes multiplication in the frequency domain, while Property (2) shows the converse. With these properties, we are nearly ready to prove that the Fourier transform maps the Schwartz space onto itself. However, we still need the following lemma's.

Lemma 2.52. Given $f \in \mathcal{S}$, we have

$$||\hat{f}||_{L^\infty} \leqslant ||f||_{L^1}.$$

Proof. For all $\xi \in \mathbb{R}^n$, we have

$$|\hat{f}(\xi)| = \left| \int_{\mathbb{R}^n} f(x) \cdot e^{-2\pi i x \cdot \xi} dx \right| \leqslant \int_{\mathbb{R}^n} |f(x)| \cdot |e^{-2\pi i x \cdot \xi}| dx = \int_{\mathbb{R}^n} |f(x)| dx = ||f||_{L^1}.$$

Since,

$$||\hat{f}||_{L^\infty} = \sup_{\xi \in \mathbb{R}^n} |\hat{f}(\xi)| \leqslant ||f||_{L^1},$$

we have

$$||\hat{f}||_{L^\infty} \leqslant ||f||_{L^1}.$$

□

We follow 3.5 of [31] for the following lemma.

Lemma 2.53. $S(\mathbb{R}^n) \subset L^P(\mathbb{R}^n), \quad$ for $P \in [1, \infty]$.

Proof. Let $f \in \mathcal{S}$, $N \in \mathbb{N}$ and α, β multi-indices. By Definition 2.4, we have

$$|f(x)| = |(1 + |x|)^N \cdot \frac{|f(x)|}{(1 + |x|)^N} \leqslant \frac{\rho_{\alpha,0}(f)}{(1 + |x|)^N}$$

with $|\alpha| = N$. Now, since $\frac{\rho_{\alpha,0}(f)}{(1+|x|)^N}$ is dominated by $\frac{1}{(1+|x|)^N}$ which is an integrable function for $N > \frac{n}{p}$, we have $f \in L^P$, $P < \infty$. For $P = \infty$, note that since $f \in \mathcal{S}$ is continuous and decays at infinity, it is bounded. Thus,

$$||f||_\infty = \sup_{x \in \mathbb{R}^n} |f(x)| < \infty.$$

□

Now, we are ready to prove that the Fourier transformation maps the Schwartz space onto itself.

Theorem 2.54. Let $f \in S$, then $\hat{f} \in S$. That is,

$$\rho_{\alpha,\beta}(\hat{f}) = \sup_{\xi \in \mathbb{R}^n} |\xi^\alpha \partial^\beta \hat{f}(\xi)| < \infty.$$

Proof. Taking norms and rewriting using Theorem 2.50 yields

$$\begin{aligned}||x^\alpha(\partial^\beta \hat{f}(x))||_{L^\infty} &= \frac{(2\pi)^{|\beta|}}{(2\pi)^{|\alpha|}}\left|\left|(\partial^\alpha(x^\beta f(x)))^\wedge\right|\right|_{L^\infty} \\ &\leqslant \frac{(2\pi)^{|\beta|}}{(2\pi)^{|\alpha|}}\left|\left|(\partial^\alpha(x^\beta f(x)))\right|\right|_{L^1} < \infty,\end{aligned}$$

where the first inequality came from Lemma 2.52 and the last inequality from the fact that $(\partial^\alpha(x^\beta f(x))) \in \mathcal{S} \subset L^1$ (Lemma 2.53). □

It remains to show that the Fourier transform is bijective and continuous in the Schwartz space. We start by showing that it is bijective by stating that there exists an inverse Fourier transform.

Definition 2.55. Given a Schwartz function f, for all $x \in \mathbb{R}^n$ we define

$$\check{f}(x) = \hat{f}(-x).$$

We call the operation $f \to \check{f}$ the inverse Fourier transform.

The properties of the Fourier transform in Theorem 2.50 also hold for the Fourier inverse.

Theorem 2.56 (Fourier Inversion)**.** Given $f \in \mathcal{S}$ we have

$$(\hat{f})^\vee = f = (\check{f})^\wedge$$

The proof of this theorem is beyond the scope of this thesis, as it uses advanced concepts such as locally compact groups and Hausdorff spaces. See Theorem 2.2.14 of [25] for a proof.

In locally convex spaces, the topology is defined by the family seminorms. So instead of using the definition of continuity directly, we draw from the theorems of [32].

Theorem 2.57. Let α and β be multi-indices. Let X and Y be locally convex spaces with families of seminorms $\{\rho_\alpha\}$ and $\{d_\beta\}$. Then, a linear map $T : X \to Y$ is continuous if and only if for all multi-indices β, there are multi-indices $\alpha_1, \alpha_2, ..., \alpha_n$ and $C > 0$ such that

$$d_\beta(Tx) \leqslant C(\rho_{\alpha_1}(x) + ... + \rho_{\alpha_n}(x)).$$

We say that the output size of the linear map is controlled by the input size.

Theorem 2.58. Let α, β multi-indices. Let $f \in \mathcal{S}(\mathbb{R}^n)$. Then the Fourier transform $\hat{f} \in \mathcal{S}(\mathbb{R}^n)$ by Theorem 2.54. The map $f \mapsto \hat{f}$ is linear by Definition 2.48. Thus, it remains to show that the seminorms of the Fourier transform are controlled by some finite sum of Schwartz seminorms of the input function. That is,

$$||\hat{f}||_{\alpha,\beta} \leqslant C(||f||_{\alpha_1,\beta_1} + ... + ||f||_{\alpha_m,\beta_m}),$$

for some $C \in \mathbb{R}$, $m \in \mathbb{N}$, multi-indices $\alpha_1, ..., \alpha_m$ and $\alpha_1, ..., \alpha_m$.

Proof. We have

$$\begin{aligned}
\xi^\alpha \partial^\beta \hat{f}(\xi) &= \int_{\mathbb{R}^n} \xi^\alpha (-2\pi i x)^\beta e^{-2\pi i x\cdot\xi} f(x)dx \\
&= \int_{\mathbb{R}^n} \frac{1}{(-2\pi i)^{|\alpha|}} (\partial^\alpha) e^{-2\pi i x\cdot\xi} (-2\pi i x)^\beta f(x)dx \\
&= \frac{(-1)^{|\alpha|}}{(-2\pi i)^{|\alpha|}} \int_{\mathbb{R}^n} e^{-2\pi i x\cdot\xi} (\partial^\alpha (-2\pi i x)^\beta f(x))dx
\end{aligned}$$

The last inequality follows from repeated integration by parts, justified by the rapid decay of f at infinity. We use that $\int_{\mathbb{R}^n}(1+|x|^2)^{-k}dx < \infty$ for $k > \frac{n}{2}$, which can be shown by switching to polar coordinates. We have

$$\begin{aligned}
||\hat{f}||_{\alpha,\beta} = \sup_{\xi\in\mathbb{R}^n} \left|\xi^\alpha \partial^\beta f(\xi)\right| &\leqslant \frac{1}{(2\pi)^{|\alpha|}} \int_{\mathbb{R}^n} \left| \frac{(1+|x|^2)^{-k}}{(1+|x|^2)^{-k}} \partial^\alpha (-2\pi i x)^\beta f(x) \right| dx \\
&\leqslant \frac{1}{(2\pi)^{|\alpha|}} \left(\int_{\mathbb{R}^n} (1+|x|^2)^{-k} dx \right) \sup_{x\in\mathbb{R}^n} \{(1+|x|^2)^{-k} |\partial^\alpha (-2\pi i x)^\beta f(x)|\}
\end{aligned}$$

Now, we can use Leibniz's rule (see Proposition 4.4.3 of [20]) to conclude that there exist multi-indices α_j, β_j and constants c_j such that

$$||\hat{f}||_{\alpha,\beta} \leqslant \sum_{i=1}^{m} c_j ||f||_{\alpha_j,\beta_j}.$$

By Theorem 2.57, we conclude that the Fourier transform is continuous. □

We conclude that the Fourier transform is indeed a homeomorphism on the Schwartz space.

3 Introduction to Lean

Lean is an *interactive theorem prover*; it allows mathematicians (and others) to formalize mathematics in a programming language and receive immediate feedback from the system in the form of errors or suggestions. Leonardo de Moura, who announced Lean at Microsoft Research in 2015, states that this open-source project aims to support mathematical reasoning and the formal verification of mathematical claims (among other goals) [16]. A trusted logic-checking core called a *kernel* forces the user to justify every statement with prior verified results, ultimately based on axioms. However, Lean provides automated tools and methods to improve both efficiency and usability.

The kernel is the core component of the system. It is based on a version of *dependent type theory* called *Calculus of Inductive Constructions* with *inductive types* [15] [19]. This is a formal language that defines how expressions can be formed by a set of rules. Moreover, every expression has a *type* which determines the kind of object it represents. For instance, the natural numbers have a type, but so does a proof of a certain theorem. The kernel checks whether an expression is logically valid by ensuring that all expressions follow the rules of the formal language and that every expression has the 'right' type. Section 3.2.1 will explain in more detail how proving statements reduces to checking types.

The open-source nature of Lean allows for rapid developments in formalizing mathematics. This has resulted in the development of the *Mathlib* library [30]. This is a user-maintained library for Lean that contains definitions, lemmas and theorems, *tactics* and other programming infrastructure. The documentation of the library can be found in [28]; it is used extensively in this project.

Section 3.1 explores the background of type theory and introduces its role as a foundation for mathematics. Then, in Section 3.2 we will introduce some important definitions and concepts used in Lean. Thereafter, Section 3.3 discusses the formalization of differentiation in Lean. Finally, in Section 3.4, we will consider the current formalization of Schwartz functions in Lean.

3.1 Background of type theory

Around the turn of the twentieth century, there was an increasing concern with mathematical rigor. Cantor's paper *On a Property of the Collection of All Real Algebraic Numbers* introduced foundational concepts that formed the basis of set theory [8] [21]. This stage of set theory is now referred to as *Naive set theory.* Loosely speaking, it proposed that all mathematics could be formed using the notion of a *set* and an *element.* This theory is referred to as naïve as it allows the use of *the comprehension principle*, which states:

> Mathematical objects sharing a property form a set, of which they are the members.

However, it turned out that this rule of construction was too simple, leading to the construction of inconsistent sets. Indeed, several logical paradoxes arose, of which probably the most famous is Russell's Paradox [34]. Simply put, Russell's paradox defines a set R as the set of all sets x that are not members of themselves. Mathematically, we write

$$R = \{x | x \notin x\}.$$

Note that this set is a member of itself if and only if it is not a member of itself: a contradiction.

To avoid such paradoxes, two important foundational approaches were developed. Namely, a redefinition of set theory, and the development of type theory. We first briefly describe how set theory was redefined before considering type theory.

3.1.1 Zermelo-Fraenkel set theory

Two of the most important contributors in the field of set theory were Ernst Zermelo and Abraham Fraenkel. They restricted the 'freedom' of the construction of sets by defining axioms that explicitly tell us how sets can be formed. For example, the *schema of separation* roughly states that sets cannot be defined merely by a property. Instead, they must be subsets of existing sets, filtered by a property. Therefore, the set

$$\{x | x \notin x\}$$

is not defined in this system. What is allowed is a set like

$$T = \{x \in V | x \notin X\}$$

for some existing set V. In this case, no paradox arises. Indeed, we can only conclude that $T \in T$ if and only if $T \notin T$ under the additional assumption that $T \in V$. So the conclusion is $T \notin V$, which does not result in a paradox. Throughout the years, many other axiomatizations were introduced. This foundation of axiomatized set theory became known as Zermelo-Fraenkel set theory and it is to this day considered the most important foundation for mathematics.

3.1.2 Type theory

During the same time period, type theory was created as an alternative foundational system. Bertrand Russell tried to resolve his own paradox and together with Alfred North Whitehead, he proposed a theory of types in the 3-volume work *Principia Mathematica* [37]. It aims to avoid paradoxes by organizing all objects into a hierarchy of types. In this system, each object lives in a so-called *universe*, indicating its place in the hierarchy. A set can only contain elements from a lower universe, which prevents Russell's paradox. Indeed, the expressions $R \in R$ and $x \notin x$ are considered to be *ill-typed* in this system. This system has a rich history of development, which we will discuss in this thesis. However, some important developments such as *dependent type theory* and the *Curry-Howard Correspondence* will be briefly introduced in later sections. Moreover, we explain how type theory is used as the logical foundation in Lean.

3.2 Preliminaries

3.2.1 Types, proofs and tactics

Unless stated otherwise, this section follows the Mathematics in Lean 4 [6] and the Theorem Proving in Lean textbooks [2]. Dependent type theory is a flexible framework that allows both making assertions and proving them in the same system. As mentioned, every expression has its own type. We can check this type in Lean using the command `#check`. For example, typing `#check 5` returns `5 :` $\mathbb{N}$, so it has type $\mathbb{N}$. Type theory allows us to build new types out of others. For example $(0, 1)$ has type $\mathbb{N} \times \mathbb{N}$ and a function $f : \mathbb{N} \to \mathbb{N}$ has type $\mathbb{N} \to \mathbb{N}$.

As discussed in Section 3.1.2, type theory organizes all objects into a hierarchy of types. Types like $\mathbb{N}$ and $\mathbb{R}$ have type[3] `Type 0`. Moreover, `Type 0` itself has a type, namely `Type 1`. We can continue this process, which gives rise to Lean's infinite hierarchy: `Type 2`, `Type 3`, and so on. We say that a `Type` n lives in a *universe* of level n. Each universe `Type n` allows the use of types from lower universes. Lean does so by defining universe variables. When writing `universe u`, a universe level `u` is created, and a type living in that universe is denoted `Type u`. In most practical applications, universe handling is done automatically by Lean.

To prove assertions in dependent type theory, we start by introducing a type called `Prop`, which represents propositions. Similar to other types, we can build new propositions from existing ones. For example, the logical conjunction 'And' has the type `Prop → Prop → Prop`. The key idea is that for each element p of type `Prop`, a proof of p is simply an expression of type p. This reflects the idea of *dependent* type theory: types can depend on values. In this case, the type of a proof term depends on the proposition p being proved. Simply put, to prove something is the same as constructing an expression of that type. This way of thinking has been made explicit in the Curry-Howard correspondence, which treats propositions as types and proofs as expressions of those types [26]. As a result, verifying a proof reduces to checking that an expression has the correct type. The set of rules that governs the construction of such expressions are formalized in The Calculus of Constructions.

But how can we build expressions of desired types (proofs) in practice? In Example 3.1 we use `rfl`, which is short for reflexivity and has type $x = x$ for any x. `rfl` is an example of a proof term that Lean knows to be true. Due to the simplicity of the proposition, this proof term works directly.

Example 3.1. The expression $2 + 2 = 4$ has type `Prop`. An expression of type $2 + 2 = 4$ represents a proof of the corresponding proposition. The expression:

$$\texttt{theorem addition\_example} : 2 + 2 = 4 := \texttt{rfl},$$

has `type` $2 + 2 = 4$ and is thus a proof of this expression.

However, Lean also provides an interactive way. *Tactics* are instructions that tell Lean how to manipulate the expressions. For example, we can use the

[3]Lean uses the abbreviation `Type`

`RW` tactic to rewrite an expression in some other form or we can use the `apply` tactic to apply a certain theorem or lemma. There are many more tactics such as *have*, *intro* and *induction.* These are explained in more detail when there is the need to use them in this project.

Tactics like `RW` and `apply` take as input a verified statement (definition, lemma or theorem) and use it to manipulate the goal.

Example 3.2. For the statement 'if $x = y$, then $0 + x = y$' we can write

$$\begin{aligned} &\texttt{example} \quad (x\,y : \mathbb{N})(h : x = y) : 0 + x = y := \text{by}\\ &\qquad\qquad\quad \texttt{rw}[\text{Nat.zero\_add}]\\ &\qquad\qquad\quad \texttt{apply}[\text{h}]. \end{aligned}$$

Here, $(h : x = y)$ represents the assumption that $x = y$. The first rewrite uses `Nat.zero_add`, which is a known result of type `0+`$n = n$. Therefore, it rewrites the goal type $0 + x = y$ to $x = y$. Finally, by applying the assumption that $x = y$, we prove the goal.

3.2.2 Definitions, structures, lemmas and theorems

Definitions, lemmas and theorems in Lean are all declarations of types. When writing `def`, we define a constant or function of a certain type. However, lemmas and theorems are not so different. Indeed, they also declare a constant (the name of the lemma or theorem) and assign it a type, followed by a proof. This means that proving a theorem like $p \to q \to p$, is the same as defining a function of that type to Lean.

Example 3.3. When given two propositions p and q and the proofs that both are true, if we want to prove that p is true we can write

$$\begin{aligned} &\texttt{def p\_holds} \ (p\,q : \text{Prop})(hp : p)(hq : q) : p := hp\\ &\texttt{theorem p\_holds} \ (p\,q : \text{Prop})(hp : p)(hq : q) : p := hp. \end{aligned}$$

Both declerations are allowed by Lean and both have type p. However, we usually write `theorem` or `lemma` when the goal is to prove a statement.

3.2.3 Using Mathlib

`Mathlib` is a user-maintained library for Lean that contains these definitions, lemmas, and theorems as well as tactics and other programming infrastructure. A lot of the mathematics described in this section is formalized in Mathlib and this library is therefore used extensively in this project [30].

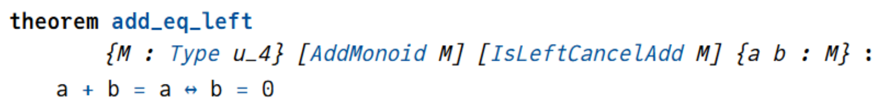
When taking definitions from the Mathlib library, it is important to understand the syntax used in their *type signatures.* A type signature specifies the input and output of a definition: it tells you what kind of values it takes and returns. By inspecting a type signature, we can see which arguments should be explicitly provided and which ones Lean infers automatically.

- Parentheses () denote explicit arguments.
- Curly braces are used for implicit arguments: Lean tries to infer them from context.
- Square brackets [] denote *typeclass arguments*. The idea of type classes is to make certain arguments implicit by automatically searching suitable instances from a user-defined database. This process is called typeclass resolution.

Example 3.4. The algebraic identity that in any *Additive monoid* M where addition is left-cancellable we have

$$a + b = a \iff b = 0 \quad \text{for } a, b \in M$$

is implemented in Mathlib as theorem `add_eq_left`. It has the following type signature [1]:



```
theorem add_eq_left
      {M : Type u_4} [AddMonoid M] [IsLeftCancelAdd M] {a b : M} :
   a + b = a ↔ b = 0
```

Figure 1: Documentation of add_eq_left in Mathlib

We will omit the formal definition of an additive monoid in this example. Let us break down this definition from Mathlib.

- `{M : Type u_4}` declares that M is a type that lives in univese `u_4`. The curly brackets indicate that Lean will try to infer it from context.
- `AddMonoid` and `IsLeftCancelAdd M` mean that M must be an additive monoid which is left-cancellable. When using this theorem in a specific context, Lean automatically uses typeclass inference to check that the used set M indeed has this structure.
- `{a b :  M}` says that variables a and b are implicitly provided, Lean will figure out which elements to use.

.

Remark 3.5. When using the above example with $a, b \in \mathbb{N}$, Lean knows from its inference database that $\mathbb{N}$ is indeed an additive monoid. Therefore, we do not have to prove this ourselves.

3.2.4 Finite numbers, coercions and structures

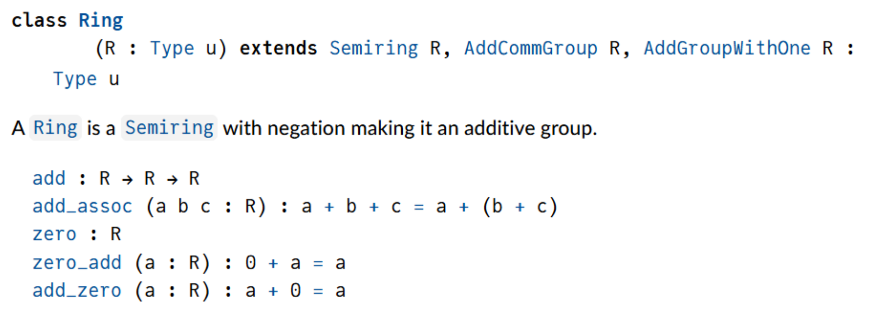
This section follows the Lean Language Reference Chapters 12 and 18 [13]. For any natural number n, the type `Fin` n is a type that contains all the natural numbers strictly less than n. Thus, `Fin` n has exactly n elements, namely

$\{0, 1, ..., n-1\}$. Therefore, it is very well suited to represent the valid indices of an array. Indeed, out-of-bound errors are not an issue. By defining a i : `Fin` n, we can access two things. Firstly, i is a natural number, which can be obtained from the command i`.val`. Secondly, $i < n$, which can be obtained by typing i`.is_lt`.

In Lean, a *coercion* is an implicit conversion from one type to another. This allows us to use a value of some type in a context where Lean expects some other type. For example, if we want to use i : `Fin` n as a natural number, then we can just write i instead of writing i`.val` every time because Lean automatically coerces i to a natural number.

In Mathlib many mathematical objects are implemented as *structures*. Structures are also declarations of types, but they can be seen as bundles of data. A *class* is a structure that allows Lean to use its typeclass inference on it automatically. Objects that are implemented as these structures allow for easy handling of its underlying datastructures such as proofs. It is best explained by an example.

Example 3.6 (Ring)**.** In Definition 2.8 we have seen that a set is called a ring if it is equipped with addition and multiplication that satisfy the ring axioms (Definition 2.8). In mathlib it is documented as follows [3]:



```
class Ring
      (R : Type u) extends Semiring R, AddCommGroup R, AddGroupWithOne R :
   Type u
```

A Ring is a Semiring with negation making it an additive group.

```
add : R → R → R
add_assoc (a b c : R) : a + b + c = a + (b + c)
zero : R
zero_add (a : R) : 0 + a = a
add_zero (a : R) : a + 0 = a
```

Figure 2: Documentation of a ring in Mathlib

Note that, for demonstration purposes only the first few lines of this documentation are shown. Let us walk through the implementation. The first line of code, the type signature, bundles the properties of a `Semiring`, `AddCommGroup` and `AddGroupWithOne` by using the extends command. By bundling these properties, R has the addition and multiplication operation and satisfies all the ring properties. Some of these properties are given in the last 5 lines of code. `add :` $R \to R \to R$, defines the addition operation. `add_assoc` $(a\, b\, c : R)\ : a + b + c = a + (b + c)$, is the additive associativity property of the ring, etc.

3.2.5 Inductive types

This section follows Chapter 7 of Theorem Proving in Lean [2]. We have seen that Lean's formal system consists of types like `Prop, Type 0, Type 1, ...`

and the construction of dependent types (with the use of constructors as $\to$ and $\times$). Moreover, we have seen the type `Nat`. This type, along with types such as lists and trees, belongs to a general family of type constructions known as *inductive types*.

To build these types, we use a set of *constructors*. Constructors describe how values of a certain type are built.

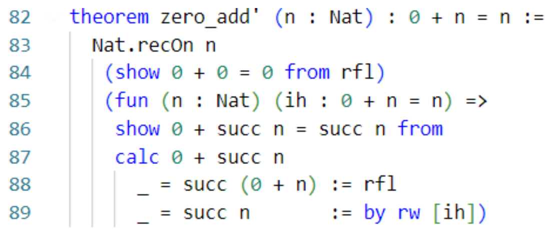
Example 3.7 (The natural numbers)**.** The natural numbers can be defined inductively by

$$\begin{aligned} \texttt{inductive}\ &\text{Nat where}: \\ &\mid \text{zero} : \text{Nat} \\ &\mid \text{succ} : \text{Nat} \to \text{Nat}, \end{aligned}$$

which consists of a base case and the recursive constructor. The base case does not take arguments, so it is defined right away. The recursive constructor (succ) can only be applied to the previously constructed `Nat`. Applying it to zero yields `succ zero : Nat`. Now, we are able to apply it to `succ zero`, which yields `succ (succ zero) : Nat` , etc. [4].

From Example 3.7 it becomes clear that for defining a function on the natural numbers (or any other inductive type) we have to tell Lean what the result should be for the base case and for the recursive step. This can be done by using a *recursor*: a construction that allows us to prove or define values step by step. For the natural numbers a recursor is `Nat.RecOn`.

Example 3.8. Say, we want to prove that $0 + n = n \quad \forall n \in \mathbb{N}$. We can write:



```
82  theorem zero_add' (n : Nat) : 0 + n = n :=
83    Nat.recOn n
84     (show 0 + 0 = 0 from rfl)
85     (fun (n : Nat) (ih : 0 + n = n) =>
86      show 0 + succ n = succ n from
87      calc 0 + succ n
88        _ = succ (0 + n) := rfl
89        _ = succ n       := by rw [ih])
```

Figure 3: Lean code for theorem zero_add'

`Nat.recOn` n tells us to proceed by recursion on n. Therefore, Lean asks for two goals: $0 + 0 = 0$ and $0 + (succ)\ n = \text{succ}\ n$ assuming the induction hypothesis $0 + n = n$. The first goal can be easily solved by using reflexivity (line 84). The second goal requires some more steps. The `calc` tactic allows us to write computations as we would on paper. First, by `rfl` we show that $0 + \text{succ}\ n = \text{succ}(0 + n)$ (line 88). Now, this equals $\text{succ}\ n$ by the induction hypothesis ih: $0 + n = n$ (line 85).

[4]The Natural Numbers Game can be played get familiar with this construction [7]

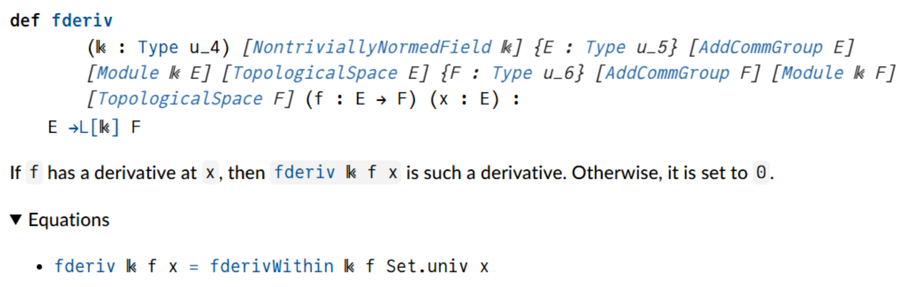
3.2.6 Currying

Currying is the process of transforming a function that takes multiple arguments into a sequence of functions, each taking a single argument. In Lean this is done by default. Indeed, the function $f : A \to B \to C$ is interpreted by Lean as $f : A \to (B \to C)$. That is, the function takes the argument A and returns a function of type $B \to C$. We say that f would be of *uncurried* form if $f : A \times B \to C$ takes as argument $(a, b) \in A \times B$ and returns an element of C. Tactics in Lean expect curried functions by default.

3.3 Differentiation in Lean

3.3.1 First order differentiation

Differentiation is formalized in Lean via Fréchet derivatives (see Definition 2.24). To understand how to work with differentiation in Lean, let us break down the Mathlib implementation of the Fréchet derivative [5].



```
def fderiv
        (𝕜 : Type u_4) [NontriviallyNormedField 𝕜] {E : Type u_5} [AddCommGroup E]
        [Module 𝕜 E] [TopologicalSpace E] {F : Type u_6} [AddCommGroup F] [Module 𝕜 F]
        [TopologicalSpace F] (f : E → F) (x : E) :
    E →L[𝕜] F
```

If `f` has a derivative at `x`, then `fderiv 𝕜 f x` is such a derivative. Otherwise, it is set to `0`.

▼ Equations

- `fderiv 𝕜 f x = fderivWithin 𝕜 f Set.univ x`

Figure 4: Documentation of Fréchet derivative in Mathlib

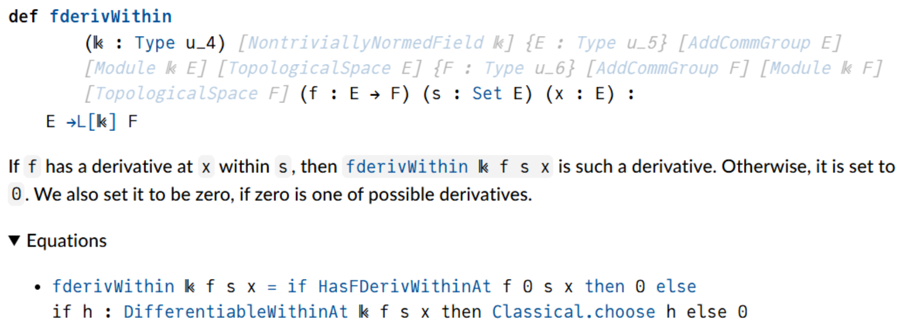
As shown in the type signature of the definition (4), Lean takes the following as input.

- An object $\mathbb{K}$ of type Type u_4
- A function $f : E \to F$ which is to be differentiated
- A point x at which the derivative is to be asserted

As in Example 3.4, this definition reveals that Lean automatically infers propositions from the input variables. For starters, from the input $\mathbb{K}$ Lean automatically infers if the input is a nontrivially normed field. For the domain and codomain E and F, Lean checks if they are abelian groups (AddCommGroup), $\mathbb{K}-$modules, and topological spaces (see Section 2.1.2). If these structures are not in place, Lean returns an error. If they are, the setting is suited for differentiation. Indeed, Definition 3 makes it clear that we have to be able to reason about subtraction, norms, division, and limits. This is justified by the structures in place:

- The domain E is:
  - an Abelian group, so $x + h$ is defined
  - a module over a nontrivially normed field, so scalar multiplication is defined
  - a topological space to allow limits.
- The codomain F is
  - an Abelian group, so $f(x + h) - f(x)$ is defined
  - a topological space to allow limits.
- The return type $E \to L[\mathbb{K}]F$:
  - requires that E and F are normed spaces, so norms are defined.

Now that we know what `fderiv` expects as input, let us consider how Lean determines the derivative. Below 'Equations' in Figure 4, we can see that Lean uses another definition called `fderivWithin` which checks if f has a derivative within some set S (in the case of providing a point x, $S = \{x\}$ or Set.univ x). It is documented as follows [5]:



```
def fderivWithin
      (𝕜 : Type u_4) [NontriviallyNormedField 𝕜] {E : Type u_5} [AddCommGroup E]
      [Module 𝕜 E] [TopologicalSpace E] {F : Type u_6} [AddCommGroup F] [Module 𝕜 F]
      [TopologicalSpace F] (f : E → F) (s : Set E) (x : E) :
   E →L[𝕜] F
```

If `f` has a derivative at `x` within `s`, then `fderivWithin 𝕜 f s x` is such a derivative. Otherwise, it is set to `0`. We also set it to be zero, if zero is one of possible derivatives.

▼ Equations

- `fderivWithin 𝕜 f s x = if HasFDerivWithinAt f 0 s x then 0 else`
  `if h : DifferentiableWithinAt 𝕜 f s x then Classical.choose h else 0`

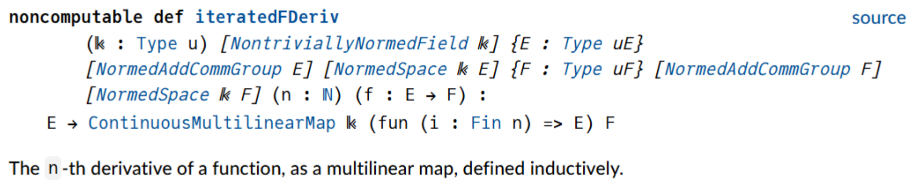
Figure 5: Documentation of `fderivwithin` in Mathlib

It uses `HasFDerivWithinAt` which returns True or False accordingly. When `HasFDerivWithinAt` returns False, `fderivWithin` is set to 0 as a placeholder. This way, it is not necessary to provide a proof of differentiability every time one refers to a derivative. When `HasFDerivWithinAt` returns True, Lean knows that there exists f' which satisfies (3). If 0 is one of the possible derivatives, it sets the derivative to 0. If it is not `Classical.choose`[5] is used to choose a Fréchet derivative, this derivative is thus not per se computed. By Theorem 2.26, we know that Lean chooses the unique derivative if f is differentiable and 0 if f is not differentiable.

[5]The definition of Fréchet derivative uses `Classical.choice` which relies on the classical axiom of choice. Although we are aware that this has been subject of historical debate, we will not explore these discussions in this thesis. In our finite context, the use of this axiom is not considered problematic.

3.3.2 Higher-order differentiation

Now that we have seen the implementation of Fréchet derivatives in Lean, we can move on to higher-order derivatives. These are also implemented in Lean and can be found in Mathlib under `iteratedFDeriv`. Let us unpack its definition [24].

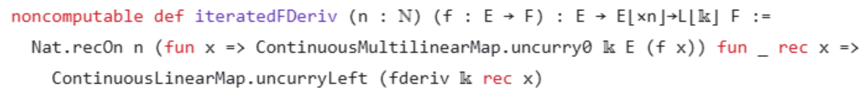


```
noncomputable def iteratedFDeriv                                               source
        (𝕜 : Type u) [NontriviallyNormedField 𝕜] {E : Type uE}
        [NormedAddCommGroup E] [NormedSpace 𝕜 E] {F : Type uF} [NormedAddCommGroup F]
        [NormedSpace 𝕜 F] (n : ℕ) (f : E → F) :
    E → ContinuousMultilinearMap 𝕜 (fun (i : Fin n) => E) F
```

The `n`-th derivative of a function, as a multilinear map, defined inductively.

Figure 6: Documentation of the iterated Fréchet derivative in Mathlib

As with **fderiv**, similar typeclass resolutions are done in this definition which we will not talk about here. However, there is a need for an additional input argument `n : N` which represents the order of differentiation. Now, there are two things worth noting here. First, the higher-order derivative is formalized as multilinear maps which aligns with the theory of Section 2.3. Indeed, we recognize the space of continuous linear maps from a normed space E to a normed space F in this type, which we denoted $\mathcal{L}^n(E, F)$. Second, it is worth looking at the inductive implementation of this definition. The documentation of Mathlib contains the following [24].



```
noncomputable def iteratedFDeriv (n : ℕ) (f : E → F) : E → E[×n]→L[𝕜] F :=
  Nat.recOn n (fun x => ContinuousMultilinearMap.uncurry0 𝕜 E (f x)) fun _ rec x =>
    ContinuousLinearMap.uncurryLeft (fderiv 𝕜 rec x)
```

Figure 7: Documentation of the iterated Fréchet derivative in Mathlib

The idea is straightforward, the '$n+1$'-th derivative is the derivative of the 'n'-th derivative. As explained in Section 3.2.5 and Section 3.2.6, Lean makes use of `Nat.recOn` to achieve this. The base case $n = 0$ defines the 0-th derivative of f at point x by returning the 0-multilinear map. The inductive case $n + 1$ computes the Fréchet derivative at x of the multilinear map $D^{n-1}f(x)$: (`Fin(n-1)` $\to E) \to L[\mathbb{K}]F$, here D is the derivative operator. This results in the map $E \to L[\mathbb{K}]F(($`Fin(n-1)` $\to E) \to L[\mathbb{K}]F)$, which is a linear map valued in linear maps. By uncurrying we get the desired result, which corresponds to $(D(D^{n-1}f(x))$. Taking this 'extra' Fréchet derivative again works as described in Section 3.3.

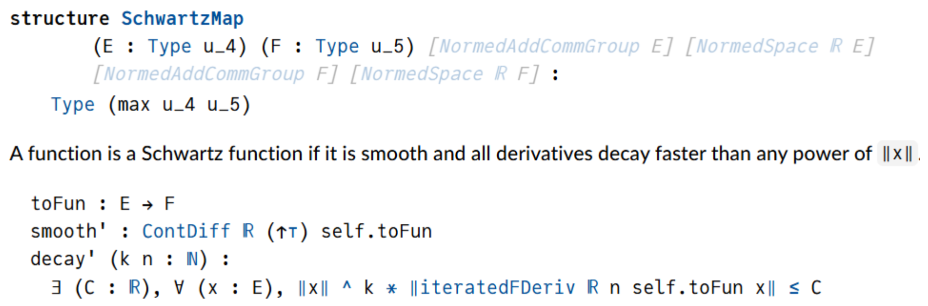
3.4 Schwartz functions in Lean

In contrast to the conventional definition of Schwartz maps as defined in Section 2.4, contributors to the Mathlib library have taken a different approach [18]. Let

$f : E \to F$ be a smooth function, where E and F are real normed vector spaces (Section 2.15) such that for all natural numbers k and n we have

$$||x^k|| * ||iteratedFDeriv\,\mathbb{R}\,n\,f\,x|| < C, \tag{8}$$

where C is a finite real number. This approach avoids the use of partial derivatives. These definitions are indeed equivalent in the special case that $E = \mathbb{R}^n$ and $F = \mathbb{C}$ (see Theorem 2.41). The documentation of Mathlib contains the following structure [18].



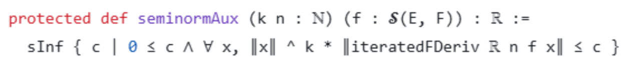
```
structure SchwartzMap
        (E : Type u_4) (F : Type u_5) [NormedAddCommGroup E] [NormedSpace ℝ E]
        [NormedAddCommGroup F] [NormedSpace ℝ F] :
    Type (max u_4 u_5)
```

A function is a Schwartz function if it is smooth and all derivatives decay faster than any power of ‖x‖.

```
  toFun : E → F
  smooth' : ContDiff ℝ (↑⊤) self.toFun
  decay' (k n : ℕ) :
    ∃ (C : ℝ), ∀ (x : E), ‖x‖ ^ k * ‖iteratedFDeriv ℝ n self.toFun x‖ ≤ C
```

Figure 8: Documentation of Schwartz maps in Mathlib

It is no surprise that Lean infers the same algebraic structures as in the derivatives sections, since the definition makes use of these derivatives. The seminorms are defined as follows [18]:



```
protected def seminormAux (k n : ℕ) (f : 𝓢(E, F)) : ℝ :=
  sInf { c | 0 ≤ c ∧ ∀ x, ‖x‖ ^ k * ‖iteratedFDeriv ℝ n f x‖ ≤ c }
```

Figure 9: Documentation of Schwartz seminorms in Mathlib

This corresponds with the smallest positive c such that

$$||x^k|| \cdot ||\texttt{iteratedFDeriv}\,\mathbb{R}\,n\,f\,x|| \leqslant c,$$

which completes the formalization of Schwartz functions in Mathlib.

4 Formalization of Schwartz functions on $\mathbb{R}^n$ in Lean

The formalization of Schwartz functions as shown in Figure 8 is suited to the general nature of the Mathlib library. It allows one to use this definition in different contexts and it is conformant with the abstract theory already implemented in Mathlib, such as the theory of topological vector spaces.

However, when learning Schwartz functions and for practical applications, the generality of the definition in Mathlib obstructs intuitive understanding and requires familiarity with more abstract mathematical objects first. This section aims to define Schwartz functions in a more concrete fashion which aligns with Definition 2.39.

The formalization process is as follows. In Section 4.1 we formalize the notion of multi-indices. This then allows for the product of monomials raised to some multi-index, Section 4.2. Section 4.3 defines partial derivatives in Lean and Section 4.4 finalizes the formalization.

4.1 Multi-indices

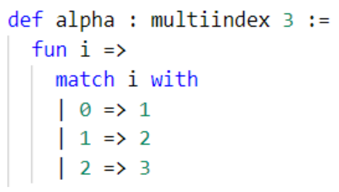
The formalization process of the Schwartz function begins with the simple definition of a multi-index. A multi-index α is an n-tuple of integers (Section 2.1.1), such as $\alpha = (\alpha_0, ..., a_{n-1})$ where $\alpha_i \geqslant 0$ for all i. This straightforward definition can be written in Lean as

```
def multi-index (n : Nat) := Fin n -> Nat.
```

This means that `multi-index n` is a function that takes an index `i : Fin n` and returns a natural number α_i, denoted in Lean by α `i`.

Note that we make use of the properties of `Fin n` as described in Section 3.2.4 which make it suitable for indexing an array. Indeed, the need for a proof that an element `i : Fin n` is strictly less than `n` ensures that out-of-bounds errors are impossible.

Example 4.1 (Multi-index)**.** Suppose that we want to define the multi-index $\alpha = (1, 2, 3)$ in Lean. In Lean we represent this as a function from `Fin 3` to $\mathbb{N}^3$ and can be coded as follows.



```
def alpha : multiindex 3 :=
  fun i =>
    match i with
    | 0 => 1
    | 1 => 2
    | 2 => 3
```

Figure 10: Lean code for the multi-index

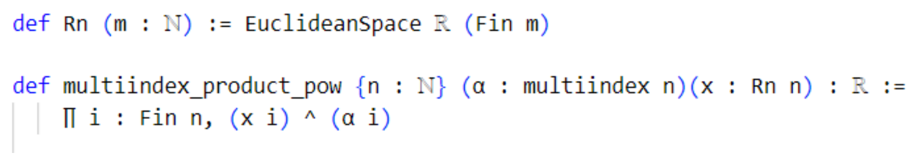
This example illustrates how our definition behaves like a function: all `i` $\in \{0, 1, 2\}$ with type `i : Fin 3` are mapped to a natural number $\{1, 2, 3\}$.[6]

4.2 Product of monomial raised to multi-index

With the notion of the multi-index formalized, we are ready to move on to the product of a polynomial raised to a multi-index

$$x^\alpha = x_0^{\alpha_0} ... x_{n-1}^{\alpha_{n-1}} = \prod_{i=0}^{n-1} x_i^{\alpha_i},$$

where $x \in \mathbb{R}^n$ and α a multi-index. Due to the definition of the multi-index, Lean can follow this definition closely:



```
def Rn (m : ℕ) := EuclideanSpace ℝ (Fin m)

def multiindex_product_pow {n : ℕ} (α : multiindex n)(x : Rn n) : ℝ :=
  ∏ i : Fin n, (x i) ^ (α i)
```

Figure 11: Lean code for a monomial raised to multi-index

For simplicity we define $\mathbb{R}^n$ as `Rn n`, where, similar to the definition of a multi-index, we use `Fin n` to map each coordinate of EuclideanSpace a value (this time in $\mathbb{R}$).

The definition for the monomial raised to a multi-index takes as input some multi-index α and some monomial `x : Rn n`. Note that `Fin n` allows us to index over dimensions and therefore make use of the product operator. Moreover, since the product is taken over $i \in$ `Fin n`, the product is well-defined due to the finiteness of `Fin n`.

4.3 Partial derivatives

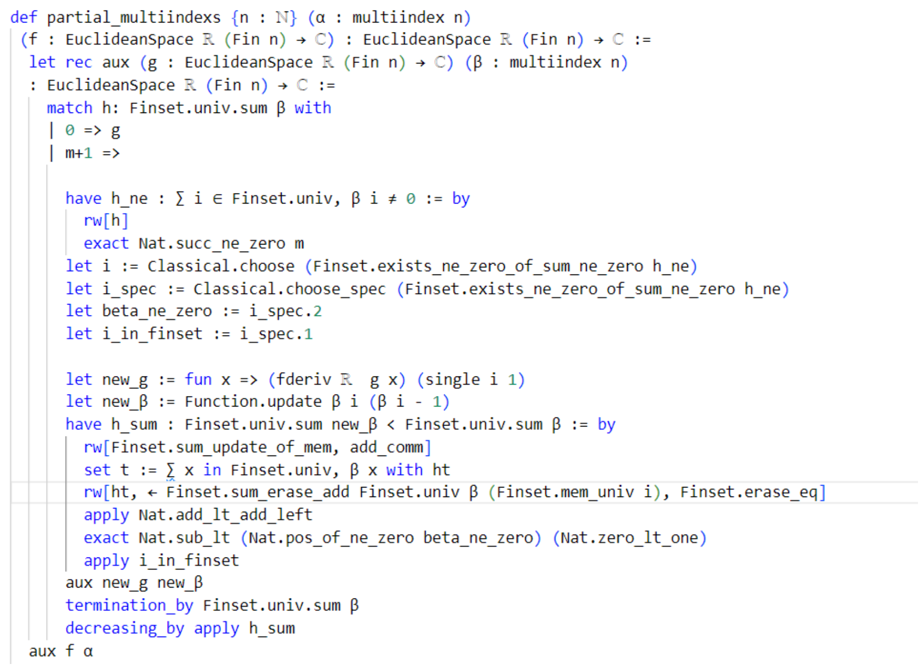
The next step in the formalization of Schwartz functions is defining partial derivatives related to the multi-indices. However, much like the Schwartz functions, multivariable calculus is generalised in Lean. Lean uses Fréchet derivatives introduced in Section 2.24, this way the derivatives are coordinate-free and are applicable for any normed vector space, not just $\mathbb{R}^n$. Therefore, this is a suitable way to define derivatives in Mathlib.

Thus, no direct definitions of partial derivatives are available in Mathlib, rather, partial derivatives are taken to be total Fréchet derivatives over a basis vector. The coordinate of the nonzero element in this vector determines the direction of the derivative. For instance, say we would like to write the partial derivative $\frac{\partial f}{\partial x_i}$ of a function $f : \mathbb{R}^n \to \mathbb{R}^m$, we could use the already defined

[6]There is shorter notation available in Mathlib, but this does not convey the point of implementation as a function as much.

Fréchet derivative and a basis vector. In Section 2.24 we have seen how to work with the predefined Fréchet derivative of Mathlib.

For our implementation of higher-order partial derivatives, we make use of a local recursive definition, which works similarly to the recursive definition in Section 3.2.5. We use the auxiliary function g to keep track of the current state of the differentiated function.



```
def partial_multiindexs {n : ℕ} (α : multiindex n)
 (f : EuclideanSpace ℝ (Fin n) → ℂ) : EuclideanSpace ℝ (Fin n) → ℂ :=
  let rec aux (g : EuclideanSpace ℝ (Fin n) → ℂ) (β : multiindex n)
  : EuclideanSpace ℝ (Fin n) → ℂ :=
    match h: Finset.univ.sum β with
    | 0 => g
    | m+1 =>

      have h_ne : ∑ i ∈ Finset.univ, β i ≠ 0 := by
        rw[h]
        exact Nat.succ_ne_zero m
      let i := Classical.choose (Finset.exists_ne_zero_of_sum_ne_zero h_ne)
      let i_spec := Classical.choose_spec (Finset.exists_ne_zero_of_sum_ne_zero h_ne)
      let beta_ne_zero := i_spec.2
      let i_in_finset := i_spec.1

      let new_g := fun x => (fderiv ℝ  g x) (single i 1)
      let new_β := Function.update β i (β i - 1)
      have h_sum : Finset.univ.sum new_β < Finset.univ.sum β := by
        rw[Finset.sum_update_of_mem, add_comm]
        set t := ∑ x in Finset.univ, β x with ht
        rw[ht, ← Finset.sum_erase_add Finset.univ β (Finset.mem_univ i), Finset.erase_eq]
        apply Nat.add_lt_add_left
        exact Nat.sub_lt (Nat.pos_of_ne_zero beta_ne_zero) (Nat.zero_lt_one)
        apply i_in_finset
      aux new_g new_β
      termination_by Finset.univ.sum β
      decreasing_by apply h_sum
  aux f α
```

Figure 12: Lean code for the partial derivative related to a multi-index

The recursive definition based on $\sum_i \beta_i$ is as follows:

- Base case: If $\sum_i \beta_i = 0$, the function returns unchanged. No derivatives are taken.

- Recursive step: If $\sum_i \beta_i \neq 0$, a coordinate i is chosen by `Classical.choose`. The directional derivative is taken with respect to direction i using `fderiv`. By evaluating this over *single* i 1 (the unit vector), the partial derivative $\frac{\partial g}{\partial x_i}$ is returned. Now, that element of the partial derivative is decremented by 1: `Function.update` $\beta\ i\ (\beta\ i - 1)$. Finally, the recursive call is made with the updated function and multi-index.

Furthermore, it can be seen that Lean asks for proofs of some basic facts that would implicitly be accepted by human readers in the recursive step. First of all, to be able to use `Classical.choose`, a hypothesis of the form

$$h : \exists x,\ p\ x$$

is needed. In our case:

$$\exists\, i \in \beta, \beta_i \neq 0.$$

This is achieved by using the `Finset.exists_ne_zero_of_sum_ne_zero` theorem defined in Mathlib which is the result of

$$\sum_{x\in S} f(x) \neq 0 \implies \exists\, a \in S, f\,(a) \neq 0,$$

for some finite set S. Now, to be able to use this result we first have to add (and prove) the hypothesis that

$$\texttt{h\_ne} : \sum_i \beta_i \neq 0.$$

This is done by telling Lean that since the sum equals $m + 1$, which is the successor of a nonnegative integer m, it must hold that the sum is not equal to 0. Indeed, `Nat.succ_ne_zero` does exactly that. Now, we are able to use `Classical.choose` to obtain an element of type `i: Fin n`. Additionally, by defining

$$\texttt{i\_spec} := \texttt{Classical.choose\_spec}$$

we are able to access all information about `i`. On the one hand that `i` : *Fin n*, and on the other hand that β `i` $\neq 0$. This is needed to provide Lean with the information that the recursive definition is terminated at some point. Indeed, we tell Lean that the termination variable is decreasing by the proof

$$h\_sum : \sum_i \beta_i^* < \sum_i \beta_i,$$

where β^* is the updated multi-index. Although this is trivial, we provide a proof here that follows the same steps as our Lean implementation. Let $S = \{1, ..., n\}$

$$\begin{aligned}
\sum_{i\in S} \beta_i^* < \sum_{i\in S} \beta_i &\iff \beta_i - 1 + \sum_{j\in S\setminus\{i\}} \beta_j < \sum_{i\in S} \beta_i \\
&\iff \sum_{j\in S\setminus\{i\}} \beta_j + (\beta_i - 1) < \sum_i \beta_i \\
&\iff \sum_{j\in S\setminus\{i\}} \beta_j + (\beta_i - 1) < \sum_{j\in S\setminus\{i\}} \beta_j + \beta_i \\
&\iff \beta_i - 1 < \beta_i \\
&\iff \beta_i - 1 < \beta_i + 0 \\
&\iff -1 < 0
\end{aligned}$$

which holds by the axioms of inequalities. Moreover, by splitting the element i of the set S, Lean aks creates a second goal to prove that this $i \in S$. We can easily deal with this by applying `i_in_finset` = `i_spec`.1, which we defined earlier.

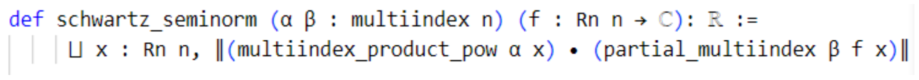
4.4 Schwartz functions

We are now ready to finalize the formalization of Schwartz functions $f : \mathbb{R}^n \to \mathbb{C}$ in Lean. We follow Section 2.4. Let $f : C^\infty(\mathbb{R}^n)$. If for all α, β multi-indices, we have

$$\rho_{\alpha,\beta}(f) = \sup_{x\in\mathbb{R}^{\ltimes}} |x^\alpha \partial^\beta f(x)| < \infty,$$

f is called a Schwartz function.

First, we define the seminorms, which follow easily from the tools we have created.

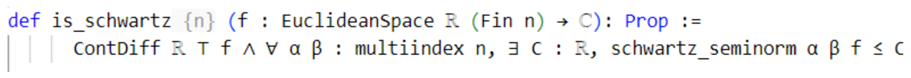


```
def schwartz_seminorm (α β : multiindex n) (f : Rn n → ℂ): ℝ :=
    ⨆ x : Rn n, ‖(multiindex_product_pow α x) • (partial_multiindex β f x)‖
```

Figure 13: Lean code for Schwartz seminorms

The definition takes as input multi-indices α and β and a function $f : \mathbb{R}^n \to \mathbb{C}$ and returns an expression of type $\mathbb{R}$, which is expected since we take a norm. The term $\bigsqcup$ x : Rn n is notation for the supremum over all $x \in \mathbb{R}^n$ and $||\cdot||$ denotes the Euclidean norm.

Finally, we have all components necessary to formalize the definition of a Schwartz function.



```
def is_schwartz {n} (f : EuclideanSpace ℝ (Fin n) → ℂ): Prop :=
    ContDiff ℝ ⊤ f ∧ ∀ α β : multiindex n, ∃ C : ℝ, schwartz_seminorm α β f ≤ C
```

Figure 14: Lean code for Schwartz functions

This definition takes as input a function $f : \mathbb{R}^n \to \mathbb{C}$, where Lean infers the dimension n from context (using the implicit argument $\{n\}$). First, the definition ensures that f is smooth, by explicitly requiring that `ContDiff` holds. This predicate takes as input the field $\mathbb{R}$, the differentiability order `⊤`(which denotes infinite differentiability), and the function f. Note that without this requirement, the definition would not work as intended. Indeed, since Lean sets the derivative of a non-differentiable function to 0, this could lead to the classification of non-smooth functions as Schwartz functions.

Furthermore, the definition states that for all multi-indices α, β the Schwartz seminorm is bounded. With this, we complete the formalization of Definition 2.4.

5 Discussion

In this thesis, we introduced Schwartz functions and motivated their relevance in Fourier analysis. Additionally, we provided both the necessary mathematical background and Lean-specific prerequisites to understand the current formalization of Schwartz functions in Lean. Finally, we presented a formalization of Schwartz functions using partial derivatives and polynomials in a coordinate-based setting.

The formalization process included formalizing monomials raised to multi-indices, higher-order partial derivatives, and Schwartz seminorms. Our definition aligns with the structure of introducing Schwartz functions in graduate textbooks such as [25], as it uses the conventional definition and builds mainly on the mathematical background introduced in undergraduate analysis courses. This motivated our concrete design approach: it allows one to learn about Schwartz spaces and differentiation in Lean without the barrier of Mathlib's more abstract implementation.

Due to this formalization process, we were forced to engage more deeply with the underlying algebraic structures and to view derivatives as multilinear maps. Combined with our study of the foundational framework in which mathematics is expressed in Lean, this resulted in a more coherent perspective on the mathematics involved.

However, our formalization comes with certain limitations. First, contributing to Mathlib requires the highest level of generality. This follows from Mathlib's design, which avoids having multiple overlapping definitions [29]. Our formalization would thus not be suitable for inclusion in Mathlib. Additionally, we make use of explicit coordinates and recursively defined partial derivatives, which results in layered definitions. This layering introduces additional complexity when interpreting or reusing the definitions.

Nonetheless, the mathematical and Lean-related background provided in this thesis could serve as background knowledge for future work in the formalization of *distribution theory*. This is a branch of mathematical analysis that generalizes the notion of functions. The Schwartz space is central in this theory, and at the time of writing, it is the only notion in distribution theory that has been formalized in Lean [28]. The formalization of *tempered distributions* would be a natural next step, as it depends on the Schwartz space.

References

[1] Jeremy Avigad, Leonardo de Moura, Simon Hudon, and Mario Carneiro. Basic.lean, 2019. URL: `https://github.com/leanprover-community/mathlib4/blob/f5e444c292cc4d1ce4feb9a859dbba4718bd8474/Mathlib/Algebra/Group/Basic.lean#L214-L214`.

[2] Jeremy Avigad, Leonardo de Moura, Soonho Kong, and Sebastian Ullrich. Theorem proving in lean 4 - theorem proving in lean 4. URL: `https://leanprover.github.io/theorem_proving_in_lean4/title_page.html`.

[3] Jeremy Avigad, Leonardo de Moura, Floris van Doorn, Yury Kudryashov, and Neil Strickland. Defs.lean, 2019. URL: `https://github.com/leanprover-community/mathlib4/blob/f5e444c292cc4d1ce4feb9a859dbba4718bd8474/Mathlib/Algebra/Ring/Defs.lean#L142-L143`.

[4] Jeremy Avigad, Kevin Donnelly, David Gray, and Paul Raff. A formally verified proof of the prime number theorem, 2006. URL: `https://arxiv.org/abs/cs/0509025`.

[5] Jeremy Avigad, Sébastien Gouëzel, and Yury Kudryashov. Fréchet derivative definitions in mathlib (lean theorem prover), 2019. URL: `https://github.com/leanprover-community/mathlib4/blob/master/Mathlib/Analysis/Calculus/FDeriv/Defs.lean`.

[6] Jeremy Avigad and Patrick Massot. Mathematics in lean — mathematics in lean 0.1 documentation, 2020. URL: `https://leanprover-community.github.io/mathematics_in_lean/index.html#`.

[7] Kevin Buzzard and The Lean Prover Community. The natural number game, 2024. URL: `https://github.com/leanprover-community/NNG4`.

[8] Georg Cantor. On a property of the class of all real algebraic numbers. *Crelle's Journal for Mathematics*, 77:258–262, 1874. URL: `https://mathdept.byu.edu/~grant/courses/cantor1874.pdf`.

[9] Neal L. Carothers. *Real Analysis*, pages 36–60, 342–350. Cambridge University Press, 2000. DOI: `https://doi.org//10.1017/cbo9780511814228`.

[10] Ward Cheney. *Calculus in Banach Spaces*, pages 87–97. Springer New York, NY, 2001. DOI: `https://doi.org/10.1007/978-1-4757-3559-8_3`.

[11] Rodney Coleman. *Calculus on Normed Vector Spaces*, pages 43, 65, 90–93. Springer New York, NY, 2012. DOI: `https://doi.org/10.1007/978-1-4614-3894-6`.

[12] Mathlib community. Completion of the liquid tensor experiment, 2022. URL: `https://leanprover-community.github.io/blog/posts/lte-final/`.

[13] The Lean Prover Community. The lean 4 reference manual, 2023. URL: `https://leanprover.github.io/lean4/doc/`.

[14] John B Conway. *A course in functional analysis*, pages 99–108. Springer, 2007. DOI: `https://doi.org/10.1007/978-1-4757-4383-8`.

[15] Thierry Coquand and Gérard Huet. The calculus of constructions. *Information and Computation*, 76(2):95–120, 1988. URL: url-https://www.sciencedirect.com/science/article/pii/0890540188900053.

[16] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). pages 378–388, 2015.

[17] Ben de Pagter, Wolter Groenevelt, and Bas Janssens. Lecture notes for analysis 2. Course notes, TU Delft, 2022. Unpublished manuscript.

[18] Moritz Doll and The Lean Prover Community. Schwartzspace, mathlib documentation, 2024. URL: `https://math.iisc.ac.in/~gadgil/PfsProgs25doc/Mathlib/Analysis/Distribution/SchwartzSpace.html#SchwartzMap`.

[19] Peter Dybjer. Inductive families. *Formal Aspects of Computing*, 6:440–465, 1994. DOI: `https://doi.org/10.1007/BF01211308`.

[20] Gerald B Folland. *Real analysis: modern techniques and their applications*, page 136. John Wiley And Sons, 1999. URL: `https://archive.org/details/real-analysis-modern-techniques-and-their-applications-gerald-b.-folland`.

[21] Dov M Gabbay, John Woods, and Akihiro Kanamori. *Handbook of the history of logic*, volume 6. Elsevier, 2004. URL: `https://dn790009.ca.archive.org/0/items/handbookofthehistoryoflogic`.

[22] Prof. Dr. Dion C Gijswijt. Lecture notes for algebra 1. Course notes, TU Delft, 2023. Unpublished manuscript.

[23] Georges Gonthier. A computer-checked proof of the four colour theorem, 2023. URL: `https://www2.tcs.ifi.lmu.de/~abel/lehre/WS07-08/CAFR/4colproof.pdf`.

[24] Sébastien Gouëzel. Ftaylorseries.lean, 2019. URL: `https://github.com/leanprover-community/mathlib4/blob/f5e444c292cc4d1ce4feb9a859dbba4718bd8474/Mathlib/Analysis/Calculus/ContDiff/FTaylorSeries.lean#L776-L779`.

[25] Loukas Grafakos. *Classical Fourier Analysis*, pages 104–113. Springer New York, NY, 3 edition, 2014. DOI: `https://doi.org/10.1007/978-1-4939-1194-3`.

[26] William A. Howard. The formulae-as-types notion of construction, 1980. URL: https://www.cs.cmu.edu/~crary/819-f09/Howard80.pdf.

[27] Vilmos Komornik. *Topology, Calculus and Approximation*, pages 117–140. Springer London, 2017. DOI : https://doi.org/10.1007/978-1-4471-7316-8_5.

[28] leanprover community. Github - leanprover-community/mathlib4: The math library of lean 4, 2021. URL: https://github.com/leanprover-community/mathlib4.

[29] The mathlib Community. The lean mathematical library. page 367–381, 2020. URL: https://doi.org/10.1145/3372885.3373824.

[30] The mathlib Community. The lean mathematical library, 2024. URL: https://github.com/leanprover-community/mathlib4.

[31] Dorina Mitrea. *The Schwartz Space and the Fourier Transform*, pages 89–115. Springer International Publishing, 2018. DOI: https://doi.org/10.1007/978-3-030-03296-8_3.

[32] Michael Reed and Barry Simon. *Methods of Modern Mathematical Physics: Functional analysis*, pages 125, 319–320. Elsevier, 1972. URL: https://archive.org/details/methods-of-modern-mathematical-physics-1-simon-reed.

[33] Walter Rudin. *Principles of mathematical analysis*, pages 24–41. McGRAW-HILL, 3 edition, 2018. URL: https://archive.org/details/principles-of-mathematical-analysis-walter-rudin/page/n5/mode/2up.

[34] Bertrand Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30:222, 1908. DOI: https://doi.org/10.2307/2369948.

[35] Robert S Strichartz. *A Guide to Distribution Theory and Fourier Transforms*, pages 31–34. World Scientific, 2003. URL: https://www.uaar.edu.pk/fs/books/29.pdf.

[36] Jan van Neerven. *Functional Analysis*, pages 18–19. Cambridge University Press, 2022. URL: https://fa.ewi.tudelft.nl/~neerven/FA/JvN-Functional_Analysis.pdf.

[37] Alfred North Whitehead and Bertrand Russell. *Principia mathematica*. Merchant Books, 2009. URL: https://archive.org/details/cu31924001575244.

A Lean code

```
import Mathlib.Topology.Basic
import Mathlib.Analysis.Fourier.FourierTransform
import Mathlib.Analysis.Distribution.SchwartzSpace


import Mathlib.Data.Real.Basic

import Mathlib.Data.Fintype.Basic
import Mathlib.Algebra.BigOperators.Group.Finset.Defs

import Mathlib.Analysis.Calculus.FDeriv.Basic
import Mathlib.Analysis.Calculus.ContDiff.Defs
import Mathlib.Analysis.Normed.Module.Basic
import Mathlib.Analysis.Calculus.ContDiff.FTaylorSeries
import Mathlib.Data.Fintype.Basic

import Mathlib.Data.Finsupp.Basic
import Mathlib.Analysis.Calculus.ContDiff.Basic

noncomputable section

def multi-index (n : ℕ) := Fin n → ℕ

def Rn (m : ℕ) := EuclideanSpace ℝ (Fin m)

def multi-index_product_pow {n : ℕ} (α : multi-index n)(x : Rn n) : ℝ :=
    ∏ i : Fin n, (x i) ^ (α i)

open EuclideanSpace

def partial_multi-indexs {n : ℕ} (α : multi-index n) (f :
    EuclideanSpace ℝ (Fin n) → ℂ) : EuclideanSpace ℝ (Fin n) → ℂ :=
  let rec aux (g : EuclideanSpace ℝ (Fin n) → ℂ) (β : multi-index n) :
    EuclideanSpace ℝ (Fin n) → ℂ :=
    match h: Finset.univ.sum β with
    | 0 => g
    | m+1 =>

      have h_ne : Σ i ∈ Finset.univ, β i ≠ 0 := by
        rw[h]
        exact Nat.succ_ne_zero m
      let i := Classical.choose (Finset.exists_ne_zero_of_sum_ne_zero
    h_ne)
      let i_spec := Classical.choose_spec
    (Finset.exists_ne_zero_of_sum_ne_zero h_ne)
      let beta_ne_zero := i_spec.2
      let i_in_finset := i_spec.1
      let new_g := fun x => (fderiv ℝ  g x) (single i 1)
      let new_β := Function.update β i (β i - 1)
      have h_sum : Finset.univ.sum new_β < Finset.univ.sum β := by
        rw[Finset.sum_update_of_mem]
        rw[add_comm]

        rw[← Finset.sum_erase_add Finset.univ β (Finset.mem_univ i)]
        rw [Finset.erase_eq]
        apply Nat.add_lt_add_left
        exact Nat.sub_lt (Nat.pos_of_ne_zero beta_ne_zero)
    (Nat.zero_lt_one)
        apply i_in_finset
      aux new_g new_β
      termination_by Finset.univ.sum β
      decreasing_by apply h_sum
  aux f α

def schwartz_seminorm (α β : multi-index n) (f : Rn n → ℂ): ℝ :=
   ⨆ x : Rn n, ‖(multi-index_product_pow α x) · (partial_multi-indexs β
    f x)‖

def is_schwartz (f : EuclideanSpace ℝ (Fin n) → ℂ): Prop :=
      ContDiff ℝ ⊤ f ∧ ∀ α β : multi-index n, ∃ C : ℝ,
    schwartz_seminorm α β f ≤ C
```