4 Mathlib-Analysis

Since the definitions of mathematical concepts in analysis are implemented based on concepts of functional analysis and even algebra structures in `mathlib`, this chapter inevitably involves such concepts that are hard to understand. Therefore, we need to first supplement these concepts. Functional analysis is a branch of mathematics that studies linear operators and their properties in infinite-dimensional spaces. To better understand functional analysis, we begin with the familiar concept of limits in Euclidean space, then gradually generalize to metric spaces, and finally introduce the notions of normed spaces and Banach spaces.

4.1 The Real Numbers and the Extended Real Number System

In certain concepts of analysis, positive or negative infinity may appear. For example, in metric spaces, the distance from a point to a set (the distance from a point to the empty set is defined as positive infinity), or in convex analysis, where the extended real number system and indicator functions of sets are used. In this section, we briefly introduce the types related to real numbers in mathlib: `ENNReal` and `EReal`. `ENNReal` refers to the nonnegative extended real number system, which includes all nonnegative real numbers together with positive infinity.

Definition 4.1.1 `ENNReal`

```
def ENNReal := WithTop ℝ≥0
  deriving Zero, Top, AddCommMonoidWithOne, SemilatticeSup,
    DistribLattice, Nontrivial
```

`EReal` refers to the extended real number system, which includes all real numbers as well as positive and negative infinity.

Definition 4.1.2 `EReal`

```
def EReal := WithBot (WithTop ℝ)
  deriving Bot, Zero, One, Nontrivial, AddMonoid, PartialOrder,
    AddCommMonoid
```

When we perform operations involving inequalities over the real numbers, we can use `linarith` to let `Lean` automatically carry out the inference for us. However, for the extended real number system such as `EReal`, the situation can often be much more complicated, since it may involve comparisons or even operations with positive and negative infinity. These operations can sometimes even be quite counterintuitive:

Example 4.1.1

```
example : (1:EReal) + Bot.bot = Bot.bot := rfl
example : (Top.top:EReal) * 0  = 0 := mul_eq_zero_of_right Top.top rfl
example : (Top.top:EReal) + (Bot.bot:EReal) = Bot.bot := rfl
example : (Top.top : EReal) - (Top.top : EReal) = Bot.bot := rfl
example : EReal.toReal Bot.bot = 0 := rfl
```

Therefore, when proving inequalities involving `EReal`, we generally need to perform a case analysis on the

situations involving positive and negative infinity. Then, we convert the types to the real numbers and use inequalities over the real numbers to complete the proof. In Lean, there are functions defined for converting between different number types:

Definition 4.1.3 `EReal.toReal`

```
def toReal : EReal → ℝ
  | Bot.bot => 0
  | Top.top => 0
  | (x : ℝ) => x
```

One can also use the tactic `lift` to accomplish this. Try the following exercise using `lift`:

Example 4.1.2

```
example {x y: EReal} (hx : x + y > 1) : x > 1 - y := by sorry
```

4.2 Euclidean Space

Whether in high school mathematics or mathematical analysis, the mathematical concepts involved are basically discussed within Euclidean space. Let us first review the definition of Euclidean space and its related properties.

Definition 4.2.1 ***n*-dimensional Euclidean space**

The n-dimensional Euclidean space is the set of all n-dimensional real vectors:

$$\mathbb{R}^n = \{(x_1, x_2, \ldots, x_n) \mid x_i \in \mathbb{R}\}.$$

If $x \in \mathbb{R}^n$, you can define this fact as

```
variable {n : Type _}[Fintype n]{x : EuclideanSpace ℝ n}
```

Euclidean space is a very special kind of space. On a Euclidean space, we can define the Euclidean distance:

$$\|x - y\| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}.$$

Euclidean norm:

$$\|x\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}.$$

Euclidean inner product:

$$\langle x, y \rangle = \sum_{i=1}^{n} x_i y_i.$$

You can also define a Euclidean space by defining a finite dimensional inner product space on $\mathbb{R}$:

Example 4.2.1

```
variable {E : Type*} [AddCommGroup E] [Module ℝ E] [FiniteDimensional ℝ
   E] [SeminormedAddCommGroup E] [InnerProductSpace ℝ E]
```

4.3 Limits

One of the most important concepts in mathematical analysis is the limit. The usual definitions of limits are based on the ε-N or ε-δ language. We first review the definition of limits in Euclidean space.

4.3.1 Limits in Euclidean Space

Definition 4.3.1 Limit of a sequence in Euclidean space
Let $\{x_k\}$ be a sequence in $\mathbb{R}^n$. If there exists $x \in \mathbb{R}^n$ such that for any $\varepsilon > 0$, there exists $N \in \mathbb{N}$ with the property that for all $k > N$,

$$\|x_k - x\| < \varepsilon,$$

then the sequence $\{x_k\}$ is said to converge to x, denoted by

$$\lim_{k\to\infty} x_k = x.$$

However, in `mathlib`, the familiar definition of limits is not based on the ε-N language, but rather on the more general concept of filters. Intuitively, a filter is a system of "collections of sufficiently large sets" used to characterize the notion of "approaching a certain point."

4.3.2 Filters

Definition 4.3.2 Filter
A filter F on a type α is a family of subsets of α satisfying the following conditions:

(i) The whole set Ω (universe) of type α is in F;

(ii) If $s \in F$ and $s \subseteq t$, then $t \in F$;

(iii) If $s, t \in F$, then $s \cap t \in F$.

We demonstrate two examples here.

Example 4.3.1 `atTop`

$$F = \{\{n \mid n \geq N\}, N \in \mathbb{N}\} \tag{1}$$

Elements of such a filter is $\{n \mid n \geq 0\}, \{n \mid n \geq 1\}, \cdots$. In `mathlib`, it is defined by the Principal Filter:

```
def atTop [Preorder α] : Filter α := inf a, P (Ici a)
```

This definition can be somewhat difficult to understand, so we will not elaborate here. However, it is possible to verify that the above definition (1) is consistent with the one in `mathlib`:

```
example : {n:ℕ | n > 2} ∈ atTop := by
  simp [atTop]
  exact mem_iInf_of_mem (Nat.succ 2) fun ⦃a⦄ a ↦ a

example : {n:ℕ | n ≥ 2} ∈ atTop := by
  simp [atTop]
  exact mem_iInf_of_mem 2 fun ⦃a⦄ a ↦ a
```

Example 4.3.2 `nhds`

As an example, consider neighborhoods in $\mathbb{R}$. This filter is defined as follows:

```
irreducible_def nhds (x : X) : Filter X :=
  inf s ∈ { s : Set X | x ∈ s ∧ IsOpen s }, P s
```

A set is called a neighborhood of x if it contains an open set around x, and all neighborhoods of x form a filter. For an explanation of why neighborhoods are defined in this way, you can refer to Kevin Buzzard's answer here: https://leanprover.zulipchat.com/#narrow/channel/116395-maths/topic/Don't.20understand.20the.20definition.20of.20neighborhoods.20filter

One can verify that, on $\mathbb{R}$, all intervals containing a point are indeed neighborhoods of that point:

```
#check Ioo_mem_nhds
#check Ioc_mem_nhds
#check Ico_mem_nhds
#check Icc_mem_nhds

example (x : ℝ) (r : ℝ) (hr : r > 0) : Ioo (x-r) (x+r) ∈ nhds x := by
  refine Ioo_mem_nhds ?_ ?_
  <;> linarith

example (x : ℝ) (r : ℝ) (hr : r > 0) : Ioo (x-r) (x+r+1) ∈ nhds x := by
  refine Ioo_mem_nhds ?_ ?_
  <;> linarith

example (x : ℝ) (r : ℝ) (hr : r > 0) : Ioc (x-r) (x+r+1) ∈ nhds x := by
  refine Ioc_mem_nhds ?_ ?_
  <;> linarith
```

If U is a neighborhood of each point of a set s then it is a neighborhood of s: it contains an open set containing s:

```
theorem exists_open_set_nhds {U : Set X} (h : ∀ x ∈ s, U ∈ nhds x) : ∃ V
    : Set X, s ⊆ V ∧ IsOpen V ∧ V ⊆ U :=
  ⟨interior U, fun x hx => mem_interior_iff_mem_nhds.2 <| h x hx,
     isOpen_interior, interior_subset⟩
```

Though the neighborhood is not the open sets, but one can still find the finite cover theorem using **nhds**:

```
theorem IsCompact.elim_nhds_subcover (hs : IsCompact s) (U : X → Set X)
   (hU : ∀ x ∈ s, U x ∈ nhds x) :
    ∃ t : Finset X, (∀ x ∈ t, x ∈ s) ∧ s ⊆     x ∈ t, U x :=
  (hs.elim_nhds_subcover_nhdsSet hU).imp fun _ h ↦ h.imp_right
     subset_of_mem_nhdsSet
```

Next, we introduce some concepts to motivate the filter-based definition of limits. First, we define the preimage of a set under a function $f : X \to Y$.

Definition 4.3.3 preimage

```
def preimage (f : α → β) (s : Set β) : Set α := {x | f x ∈ s}
/-- `f ⁻¹' s` denotes the preimage of `s : Set β` under the function `f :
    α → β`. -/
infixl:80 " ⁻¹' " => preimage
```

For example, given a real sequence $x : \mathbb{N} \to \mathbb{R}$, the preimage of the interval $(-1, 1)$ under this sequence is the set of all indices n such that $x_n \in (-1, 1)$:

Example 4.3.3

```
example : x ⁻¹' (Ioo (-1) 1) = {n:ℕ | x n ∈ (Ioo (-1) 1)} := by rfl
```

The preimage of a composition equals the composition of the preimages, which follows directly from the definition.

Example 4.3.4

```
theorem preimage_comp {s : Set γ} : g ∘ f ⁻¹' s = f ⁻¹' (g ⁻¹' s) := rfl
```

Next, we define a new filter, called the forward mapping of a filter `Filter X` along a function $f : X \to Y$, denoted by F^* (defined as `Filter.map f F`; note that its type is `Filter Y`):

Definition 4.3.4 **Forward mapping**

```
#check Filter.map x atTop
example (V : Set ℝ): V ∈ Filter.map x atTop ↔ x⁻¹' V ∈ atTop :=mem_map
```

Since filters themselves are also sets, we can define an order relation on them:

Example 4.3.5

```
theorem le_def : f ≤ g ↔ ∀ x ∈ g, x ∈ f := Iff.rfl
```

Consider the limit $\lim_{n\to\infty} x_n = x$. This essentially describes two notions of "approaching," which can be characterized by two filters. In `mathlib`, the definition of limit is given by

Definition 4.3.5 **Limit**

```
def Tendsto (f : α → β) (l₁ : Filter α) (l₂ : Filter β) :=
  l₁.map f ≤ l₂
```

Example 4.3.6

Still take the sequence x_n as an example, $\lim_{n\to\infty} x_n = x$ is by definition

```
example (x0 : ℝ): Tendsto x atTop (nhds x0) ↔ Filter.map x atTop ≤ nhds
    x0 := by rfl
```

according to Example 4.3.4, we have the following derivation:

$$\begin{aligned}
\texttt{Tendsto x atTop (nhds x0)} &\iff \texttt{Filter.map x atTop} \le \texttt{nhds x0} \\
&\iff \forall\ \texttt{V} \in \texttt{nhds x0},\ \texttt{V} \in \texttt{Filter.map x atTop} \\
&\iff \forall\ \texttt{V} \in \texttt{nhds x0},\ \texttt{x⁻¹' V} \in \texttt{atTop} \\
&\iff \forall\ \varepsilon > 0,\ \texttt{x⁻¹' (Ioo (x0-}\varepsilon\texttt{) (x0+}\varepsilon\texttt{))} \in \texttt{atTop} \\
&\iff \forall\ \varepsilon > 0, \exists\ \texttt{N} : \mathbb{N}, \forall\ \texttt{n} > \texttt{N}, \texttt{x n} \in \texttt{(Ioo (x0-}\varepsilon\texttt{) (x0+}\varepsilon\texttt{))}
\end{aligned}$$

The equivalence between `nhds` and a specific open interval comes from the concept of filter basis. The open intervals $(x_0 - \varepsilon, x_0 + \varepsilon)$ forms a filter basis of `nhds`.

One can construct a sub-sequence by a strict monotone mapping:

Example 4.3.7 Sub-sequence

```
example (a : ℝ) (hx : Tendsto x atTop (nhds a)) (φ : ℕ → ℕ) (hφ:
    StrictMono φ): Tendsto (x ∘ φ) atTop (nhds a) := by sorry
```

You can check how the other operations are implemented and proved in `mathlib` through the concept of filters:

Example 4.3.8 Addition and multiplication

```
example {a b: ℝ} {x' : ℕ → ℝ} {y' : ℕ → ℝ} (hf : Tendsto x' atTop (nhds
    a)) (hg : Tendsto y' atTop (nhds b)) : Tendsto (fun n => (x' n + y' n
    )) atTop (nhds (a + b)) := by exact Tendsto.add hf hg

example {a b: ℝ} {x' : ℕ → ℝ} {y' : ℕ → ℝ} (hf : Tendsto x' atTop (nhds
    a)) (hg : Tendsto y' atTop (nhds b)) : Tendsto (fun n => (x' n) * (y'
     n)) atTop (nhds (a * b)) := by exact Tendsto.mul hf hg
```

Besides limits, filters can also be used to describe something that will eventually happen:

Example 4.3.9

```
example (P : ℕ → Prop) : (∀ᶠ n in atTop, P n) ↔ {n | P n} ∈ atTop :=
    .rfl
```

4.4 Metric Spaces

For general spaces, a key question is how to define a notion similar to the Euclidean norm to measure the distance between two points. To generalize the concept of limits from Euclidean spaces to more general spaces, we introduce the notion of a *distance function* and the definition of metric spaces.

Definition 4.4.1 Distance Function
Let X be a nonempty set. A function

$$d : X \times X \to [0, \infty)$$

is called a distance (or metric) on X if for all $x, y, z \in X$, the following hold:

(i) $d(x, x) = 0 \iff x = 0$;

(ii) $d(x, y) = d(y, x)$ (symmetry);

(iii) $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality).

(iv) $d(x, y) = 0 \iff x = y$.

(X, d) is called a metric space. In `mathlib`, `MetricSpace` is extended from `PseudoMetricSpace`.

```
class Dist (α : Type*) where
  dist : α → α → ℝ

class PseudoMetricSpace (α : Type u) : Type u extends Dist α where
  dist_self : ∀ x : α, dist x x = 0
  dist_comm : ∀ x y : α, dist x y = dist y x
  dist_triangle : ∀ x y z : α, dist x z ≤ dist x y + dist y z
  /-- Extended distance between two points -/
    ...

class MetricSpace (α : Type u) extends PseudoMetricSpace α : Type u
   where
  eq_of_dist_eq_zero : ∀ {x y : α}, dist x y = 0 → x = y
```

Definition 4.4.2 Limit of a sequence in a metric space
Let $\{x_k\}$ be a sequence in a metric space (X, d). If there exists $x \in X$ such that for every $\varepsilon > 0$, there exists $N \in \mathbb{N}$ such that for all $k > N$,

$$d(x_k, x) < \varepsilon,$$

then the sequence $\{x_k\}$ is said to converge to x, denoted by

$$\lim_{k \to \infty} x_k = x.$$

This definition is a theorem proved to be equivalent to that defined by the Filter in `mathlib`.

Example 4.4.1

```
variable {X : Type*} [MetricSpace X]
example (u : ℕ → X) (a : X): Tendsto u atTop (nhds a) ↔ ∀ ε > 0, ∃ N, ∀
    n ≥ N, dist (u n) a < ε := by exact Metric.tendsto_atTop
```

Definition 4.4.3 Cauchy sequence in a metric space
Let $\{x_k\}$ be a sequence in a metric space (X, d). If for every $\varepsilon > 0$, there exists $N \in \mathbb{N}$ such that for all $m, n > N$,

$$d(x_m, x_n) < \varepsilon,$$

then $\{x_k\}$ is called a Cauchy sequence.

In `mathlib`, you can find this definition in

Example 4.4.2

```
example (u : ℕ → X): CauchySeq u ↔ ∀ ε > 0, ∃ N, ∀ m ≥ N, ∀ n ≥ N, dist
    (u m) (u n) < ε := Metric.cauchySeq_iff
```

Definition 4.4.4 Complete metric space
A metric space (X, d) is called complete if every Cauchy sequence converges in X.

This is proved in `mathlib` in the following theorem

Example 4.4.3

```
example :(∀ (u : ℕ → X), CauchySeq u → ∃ a, Tendsto u atTop (nhds a)) →
    CompleteSpace X := Metric.complete_of_cauchySeq_tendsto
```

In mathematical analysis, many students have learned the Cauchy convergence theorem: in Euclidean spaces, convergent sequences and Cauchy sequences are equivalent. This is essentially because Euclidean spaces are complete metric spaces.

4.5 Normed Spaces

Although metric spaces are quite general, in functional analysis, we often need spaces where vectors support "addition" and "scalar multiplication." Therefore, we introduce norms on linear spaces.

Definition 4.5.1 Norm
Let X be a linear space. A function

$$\|\cdot\| : X \to [0, \infty)$$

is called a norm on X if for all $x, y \in X$ and scalars α, the following hold:

(i) $\|x\| = 0 \iff x = 0$;

(ii) $\|\alpha x\| = |\alpha| \cdot \|x\|$;

(iii) $\|x+y\| \leq \|x\| + \|y\|$.

Definition 4.5.2 Normed (linear) space

If X is a linear space equipped with a norm $\|\cdot\|$, then $(X, \|\cdot\|)$ is called a normed space. In `mathlib`, `NormedSpace` is extended from a `Module` (In algebra, `Module` is something similar to the linear space, or vector space):

```
class NormedAddCommGroup (E : Type*) extends Norm E, AddCommGroup E,
    MetricSpace E where
  dist := fun x y => ‖x - y‖
  /-- The distance function is induced by the norm. -/
  dist_eq : ∀ x y, dist x y = ‖x - y‖ := by aesop

class NormedSpace (k : Type*) (E : Type*) [NormedField k] [
    SeminormedAddCommGroup E]
     extends Module k E where
  norm_smul_le : ∀ (a : k) (b : E), ‖a • b‖ ≤ ‖a‖ * ‖b‖
```

One can declare a `NormedSpace` by

Example 4.5.1

```
variable {E : Type*} [NormedAddCommGroup E] [NormedSpace ℝ E]
```

You can check the properties of norm by

Example 4.5.2

```
example (x : E) : 0 ≤ ‖x‖ := norm_nonneg x
example {x : E} : ‖x‖ = 0 ↔ x = 0 := norm_eq_zero
example (x y : E) : ‖x + y‖ ≤ ‖x‖ + ‖y‖ := norm_add_le x y
example (a : ℝ) (x : E) : ‖a • x‖ = |a| * ‖x‖ := norm_smul a x
```

Using the norm, we can naturally define a distance by

$$d(x, y) = \|x - y\|.$$

Therefore, a normed space is naturally a metric space.

Example 4.5.3

```
example : MetricSpace E := by infer_instance
```

Definition 4.5.3 Banach space

A complete normed linear space is called a Banach space. You can define a Banach Space by

```
variable {E : Type*} [NormedAddCommGroup E] [NormedSpace ℝ E] [
    CompleteSpace E]
```

In the theory of functional analysis, we have the following famous result:

Theorem 4.5.4
Every finite-dimensional normed linear space is complete. This theorem can be proved by `infer_instance`.

```
example [FiniteDimensional ℝ E] : CompleteSpace E := by infer_instance
```

4.6 Inner Product Space

Definition 4.6.1 Inner product space
Let V be a vector space over $\mathbb{F}$, where $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$. An *inner product* on V is a function

$$\langle \cdot, \cdot \rangle : V \times V \to \mathbb{F}$$

satisfying the following axioms for all $x, y, z \in V$, and $\alpha \in \mathbb{F}$:

1. **Linearity in the first argument**:

$$\langle \alpha x + y, z \rangle = \alpha \langle x, z \rangle + \langle y, z \rangle$$

   (For complex spaces, this is sesquilinearity: linear in the first argument, conjugate linear in the second.)

2. **Conjugate symmetry**:

$$\langle x, y \rangle = \overline{\langle y, x \rangle}$$

3. **Positive-definiteness**:

$$\langle x, x \rangle \geq 0 \quad \text{and} \quad \langle x, x \rangle = 0 \iff x = 0$$

A vector space equipped with an inner product is called an **inner product space**.

```
class InnerProductSpace (k : Type*) (E : Type*) [RCLike k] [
    SeminormedAddCommGroup E] extends
  NormedSpace k E, Inner k E where
  /-- The inner product induces the norm. -/
  norm_sq_eq_inner : ∀ x : E, ‖x‖ ^ 2 = re (inner x x)
  /-- The inner product is *hermitian*, taking the `conj` swaps the
      arguments. -/
  conj_symm : ∀ x y, conj (inner y x) = inner x y
  /-- The inner product is additive in the first coordinate. -/
  add_left : ∀ x y z, inner (x + y) z = inner x z + inner y z
  /-- The inner product is conjugate linear in the first coordinate. -/
  smul_left : ∀ x y r, inner (r      x) y = conj r * inner x y
```

One can declare an inner product space by

Example 4.6.1

```
variable {E : Type*}[SeminormedAddCommGroup E][InnerProductSpace ℝ E]
```

and define a local notation of the inner product by

```
local notation "⟪" a₁ ", " a₂ "⟫" => @inner ℝ _ _ a₁ a₂
```

4.7 Open and Closed Sets

In a metric space, an open set is defined as follows:

Definition 4.7.1 Open and Closed Sets
Let (X, d) be a metric space. A set O is called open if for every $x \in O$, there exists an open ball contained in the set, that is,

$$\exists r > 0, \quad B(x, r) \subseteq O.$$

Here,

$$B(x, r) = \{y \in X \mid d(x, y) < r\}.$$

A closed set is a set whose complement is open.

```
#check Metric.ball
#check Metric.closedBall
example (s : Set X) : IsOpen s ↔ ∀ x ∈ s, ∃ ε > 0, Metric.ball x ε ⊆ s
   := Metric.isOpen_iff

#check IsClosed
example {s : Set X} : IsClosed s ↔ IsOpen ( s  ) :=
   isOpen_compl_iff.symm
```

We have mentioned in subsection 4.3 that a set is called a neighborhood of x if it contains an open set around x. Hence all the metric balls centered at x will be in `nhds x`.

Example 4.7.1

```
example {x : X} {s : Set X} : s ∈ nhds x ↔ ∃ ε > 0, Metric.ball x ε ⊆ s
   := Metric.nhds_basis_ball.mem_iff
example {x : X} {s : Set X} : s ∈ nhds x ↔ ∃ ε > 0, Metric.closedBall x
   ε ⊆ s := Metric.nhds_basis_closedBall.mem_iff
```

4.8 Continuous Functions on a Metric Space

In `mathlib`, `Continuous` is defined between topological spaces:

Definition 4.8.1 Continuous functions

A function between topological spaces is continuous if the preimage of every open set is open.

```
structure Continuous (f : X → Y) : Prop where
  /-- The preimage of an open set under a continuous function is an open
      set. Use `IsOpen.preimage`
  instead. -/
  isOpen_preimage : ∀ s, IsOpen s → IsOpen (f ⁻¹' s)
```

Definition 4.8.2 Hilbert Space

A complete inner product space is called a Hilbert space.

```
variable {E : Type*}[SeminormedAddCommGroup E][InnerProductSpace ℝ E][
    CompleteSpace E]
```

Definition 4.8.3 Continuous at a point

Continuity at a point is defined by the limit.

```
def ContinuousAt (f : X → Y) (x : X) :=
  Tendsto f (nhds x) (nhds (f x))

def ContinuousWithinAt (f : X → Y) (s : Set X) (x : X) : Prop :=
  Tendsto f (nhds[s] x) (nhds (f x))

def ContinuousOn (f : X → Y) (s : Set X) : Prop :=
  ∀ x ∈ s, ContinuousWithinAt f s x
```

This definition coincides that defined by $\varepsilon - \delta$ language:

Example 4.8.1

```
example {X Y : Type*} [MetricSpace X] [MetricSpace Y] (f : X → Y) (a : X
    ) :
     ContinuousAt f a ↔ ∀ ε > 0, ∃ δ > 0, ∀ {x}, dist x a < δ → dist (f x
        ) (f a) < ε := Metric.continuousAt_iff
```

And the definition defined on topology is equivalent to the one defined by limit:

Example 4.8.2

```
example {X Y : Type*} [MetricSpace X] [MetricSpace Y] (f : X → Y) :
  Continuous f ↔ ContinuousOn f univ := by
    exact continuous_iff_continuousOn_univ
```

You can try the tactic `continuity` to prove goals including continuity:

Example 4.8.3

```
variable {X : Type*}[MetricSpace X]
example {f : ℝ → X} (hf : Continuous f) : Continuous fun x : ℝ ↦ f (x ^
   2 + x) := by continuity
```

You can also use `continuity?` to show the details:

Example 4.8.4

```
variable {X : Type*}[MetricSpace X]
example {f : ℝ → X} (hf : Continuous f) : Continuous fun x : ℝ ↦ f (x ^
   2 + x) := by
  apply Continuous.comp'
  · simp_all only
  · apply Continuous.add
    · apply Continuous.pow
      apply continuous_id'
    · apply continuous_id'
```

4.9 Continuous Linear Operators

Let E and F be two normed linear spaces with norms $\|\cdot\|_E$ and $\|\cdot\|_F$ respectively.

Definition 4.9.1 Linear Operator
A linear operator $T : E \to F$ on a field $\mathbb{K}$ is an operator satisfying

(i) $\forall x, y \in E,\ T(x+y) = T(x) + T(y)$.

(ii) $\forall x \in E, c \in \mathbb{K},\ T(c \cdot x) = c \cdot T(x)$.

Definition 4.9.2 Continuous Linear Operator
Given a sequence $\{x_n\} \subset E$, a linear operator $T : E \to F$ is called **continuous** if

$$x_n \to x_0 \implies Tx_n \to Tx_0.$$

In `mathlib`, a continuous linear operator between two normed spaces E and F on $\mathbb{R}$ is denoted as

```
variable (f : E →L[ℝ] F)
```

The linearity and continuity can be checked in the following example:

Example 4.9.1

```
example : Continuous f := f.cont
example (x y : E) : f (x + y) = f x + f y := f.map_add x y
example (a : ℝ) (x : E) : f (a     x) = a     f x := f.map_smul a x
```

Equivalently, one can define continuity as:

Definition 4.9.3 Continuous Linear Operator
A linear map $T : E \to F$ is **continuous** if

$$\lim_{x \to 0} \|T(x)\|_F = 0.$$

Definition 4.9.4 Bounded Linear Operator
If there exists a constant $C > 0$ such that

$$\|T(x)\|_F \leq C\|x\|_E, \quad \forall x \in E,$$

then T is called a bounded linear operator.

Theorem 4.9.5 Equivalence between boundedness and continuity
Let E, F be normed linear spaces. Then a linear operator T is bounded if and only if it is continuous. This can be proved by a tactic `continuity` in `lean`. One can also check `continuity?` to obtain details.

```
example : Continuous f ↔ IsBoundedLinearMap ℝ f := by
  constructor
  · exact fun a ↦ ContinuousLinearMap.isBoundedLinearMap f
  · exact fun a ↦ ContinuousLinearMap.continuous f
```

The operator norm of continuous linear operators is defined by

Definition 4.9.6 Operator norm

```
def opNorm (f : E →SL[σ₁₂] F) :=
  sInf { c | 0 ≤ c ∧ ∀ x, ‖f x‖ ≤ c * ‖x‖ }
```

```
example (x : E) : ‖f x‖ ≤ ‖f‖ * ‖x‖ := f.le_opNorm x
```

4.10 Derivatives and Differentials on Normed Linear Spaces

Definition 4.10.1 Fréchet Differentiability and Fréchet Derivative
Let X, Y be normed linear spaces, and $U \subset X$. A function $f : U \to Y$ is **Fréchet differentiable** at $x_0 \in U$ if there exists a continuous linear operator $L : X \to Y$ such that

$$\lim_{\substack{h \to 0 \\ x_0 + h \in U}} \frac{\|f(x_0 + h) - f(x_0) - L(h)\|}{\|h\|} = 0. \tag{2}$$

Equivalently,

$$f(x_0 + h) = f(x_0) + L(h) + o(\|h\|),$$

and L is called the Fréchet derivative of f at x_0, denoted by

$$Df(x_0) = L.$$

In `mathlib`, it is defined as

```
structure HasFDerivAtFilter (f : E → F) (f' : E →L[ℝ] F) (x : E) (L :
    Filter E) : Prop where
  of_isLittleO :: isLittleO : (fun x' => f x' - f x - f' (x' - x)) =o[L]
      fun x' => x' - x
```

The key components of F-derivative in `mathlib` is a continuous linear map and a filter. The continuous linear map serves as the derivative, while the filter helps describes the little o notation, which means you are approaching a point.

If you want to consider the derivative on a point of a subset of the whole space, you need to restrict all the points computed on this subset, that is, the subset U in (2). Consequently, the filter should consist of the intersections between neighborhoods and the subset, that is `nhds[U] x`. If you want to consider the on a point of the whole space, then the constraint on U can be removed. This leads to the following two definitions.

Definition 4.10.2 Has Frechet derivatives at a point

```
def HasFDerivWithinAt (f : E → F) (f' : E →L[ℝ] F) (s : Set E) (x : E)
    := HasFDerivAtFilter f f' x (nhds[s] x)
def HasFDerivAt (f : E → F) (f' : E →L[ℝ] F) (s : Set E) (x : E) :=
    HasFDerivAtFilter f f' x (nhds x)
```

With this Frechet derivative, we can define differentiable of a function. As what we have mentioned above, `mathlib` has two versions on differentiability, one is for a subset, and the other one is for the entire space.

Definition 4.10.3 Differentiability at a point

A function f is differentiable at a point x within a set s if it admits a derivative there (possibly non-unique).

```
def DifferentiableWithinAt (f : E → F) (s : Set E) (x : E) :=
  ∃ f' : E →L[ℝ] F, HasFDerivWithinAt f f' s x
```

A function f is differentiable at a point x if it admits a derivative there (possibly non-unique).

```
def DifferentiableAt (f : E → F) (x : E) :=
  ∃ f' : E →L[ℝ] F, HasFDerivAt f f' x
```

Definition 4.10.4 Definitions of the linear operators

If f has a derivative at x within s, then `fderivWithin` is such a derivative. Otherwise, it is set to 0. If x is isolated in s, we take the derivative within s to be zero for convenience.

```
irreducible_def fderivWithin (f : E → F) (s : Set E) (x : E) : E →L[ℝ] F
    :=
  if nhds[s \ {x}] x = Bot.bot then 0 else
  if h : ∃ f', HasFDerivWithinAt f f' s x then Classical.choose h else 0
```

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

```
irreducible_def fderiv (f : E → F) (x : E) : E →L[ℝ] F :=
  if h : ∃ f', HasFDerivAt f f' x then Classical.choose h else 0
```

Definition 4.10.5 Differentiability on a set

`DifferentiableOn` means that f is differentiable within s at any point of s.

```
def DifferentiableOn (f : E → F) (s : Set E) :=
  ∀ x ∈ s, DifferentiableWithinAt ℝ f s x
```

`Differentiable` means that f is differentiable at any point.

```
def Differentiable (f : E → F) :=
  ∀ x, DifferentiableAt ℝ f x
```

Example 4.10.1

If f is differentiable on `univ`, then you can find the equivalence between the two definitions above:

```
example : DifferentiableOn ℝ f univ ↔ Differentiable ℝ f :=
   differentiableOn_univ
```

The remaining problem is, what is definition of derivative? Derivative is the special case that $E = \mathbb{K}$, meaning that $f : \mathbb{K} \to F$ is a one-dimensional function. Then you can define the following similar definitions:

Definition 4.10.6 Derivatives

f has the derivative f' at the point x as x goes along the filter L.

```
#check HasDerivAtFilter
```

f has the derivative f at the point x within the subset s.

```
#check HasDerivWithinAt
```

f has the derivative f' at the point x.

```
#check HasDerivAt
```

Derivative of f at the point x within the set s, if it exists.

```
#check derivWithin
```

Derivative of f at the point x, if it exists.

```
#check deriv
```

If f is defined on an inner product space, one can define the gradient of a function. You can check the definitions of gradient, which is also derived from Frechet derivative.

Example 4.10.2

```
#check HasGradientAtFilter

#check HasGradientWithinAt
#check HasGradientAt

#check gradientWithin
#check gradient
```