D4: Analysis and Topology

Structures, filters, limits, derivatives, integrals, and proof workflow

Summer School on Lean

July 18, Friday

Learning Goals

- read Mathlib analysis statements through their required structures;
- compare undergraduate epsilon language with Mathlib's filter language;
- read sequence, function, punctured, one-sided, and infinite limits as `Tendsto`;
- connect `ContinuousAt` and `Continuous` with preservation of limits;
- distinguish `deriv f x` from `HasDerivAt f c x`;
- recognize interval integrals and use a practical Mathlib proof workflow.

Four-Part Outline

Part I	**Structures** real numbers, type class assumptions, metric spaces, topology, neighborhoods
Part II	**Limits** epsilon-N, epsilon-delta, filters, `Tendsto`, continuity
Part III	**Derivatives** `deriv`, `HasDerivAt`, derivative rules and examples
Part IV	**Integrals** interval integrals, FTC map, Mathlib search and proof repair

Undergraduate Route and Mathlib Route

Undergraduate analysis	Mathlib analysis
start with $\mathbb{R}$, absolute values, and inequalities	start with a type carrying the needed structures
define limits by epsilon-N and epsilon-delta formulas	express limits uniformly with `Tendsto`
topology often appears later as an abstraction	topology and filters are infrastructure for analysis

Epsilon language is not discarded; it is recovered from `MetricSpace` statements.

Part I: Structures

Mathlib analysis is built from reusable structures.

- algebraic structure: operations and laws;
- order structure: inequalities, intervals, monotonicity;
- topological structure: neighborhoods, continuity, limits;
- metric structure: distance and epsilon-style statements;
- filters: a uniform language for convergence.

The point is not to memorize the hierarchy, but to read what a theorem needs.

The Real Line Carries Many Structures

```
import Mathlib

#check Real
#check abs
#check Set.Icc

#synth Field Real
#synth LinearOrder Real
#synth TopologicalSpace Real
#synth MetricSpace Real
```

- The real line is a familiar object with many registered instances.
- Type class search supplies algebraic, order, topological, and metric structure.
- Many analysis theorems are stated for abstract spaces, not only for `Real`.

Live Lean: `codeD4/D4_AnalyticStructures.lean`

Reading Structure Assumptions

A theorem may look abstract because it advertises the structures it needs.

Assumption	Mathematical meaning
`[TopologicalSpace X]`	open sets, neighborhoods, topological limits
`[MetricSpace X]`	distance, open balls, epsilon-style statements
`[LinearOrderedField X]`	ordered field algebra and inequalities
`[NormedAddCommGroup X]`	norm, distance from subtraction, normed-space analysis

For a concrete type such as `Real`, Lean usually finds these instances automatically.

Undergraduate Convergence Route

A first analysis course often follows this route:

$\mathbb{R}$ or $\mathbb{R}^n$

↓

absolute value or distance

↓

epsilon-neighborhoods and open balls

↓

epsilon-N and epsilon-delta definitions

↓

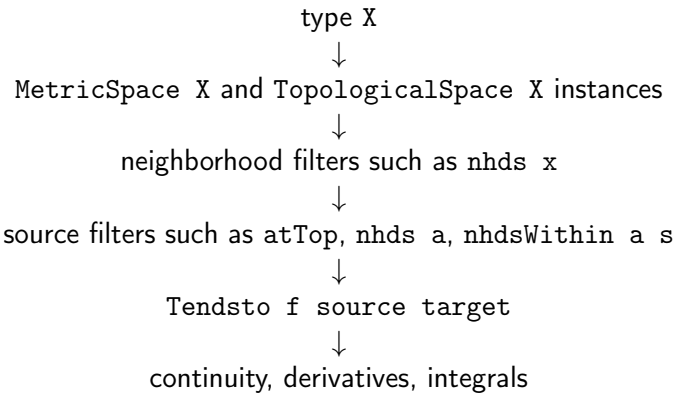
continuity, derivatives, integrals

↓

topology as an abstraction

Mathlib Convergence Route

Mathlib usually organizes the same ideas in the opposite direction:



Vocabulary Translation

Undergraduate language	Mathlib language	Reading
$\lvert x - y \rvert$ or distance	`dist x y`	distance between two points
epsilon open ball	`Metric.ball x eps`	points within radius eps
x near a	`nhds a`	neighborhood filter of a
$n \to \infty$	`atTop`	eventually for large indices
$x \to a$ within a set	`nhdsWithin a s`	restricted approach
$u_n \to L$	`Tendsto u ...`	sequence convergence
$\lim_{x \to a} f(x) = L$	`Tendsto f ...`	function limit

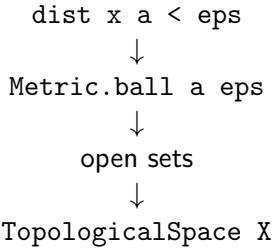
Metric Spaces: Distance and Balls

```
#check dist
#check Metric.ball
#check Metric.mem_ball
```

- A `MetricSpace X` supplies a distance function.
- In `Real`, `dist x a` corresponds to $|x - a|$.
- Epsilon-delta language is naturally metric-space language.

From Balls to Open Sets

In a metric space, open balls generate the usual topology.



Mathlib often states results at the topological level, then specializes them to metric spaces.

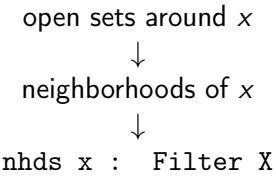
Topological Spaces

Object	Role in analysis
open sets	topological description of local information
neighborhoods	sets containing enough points near x
`nhds x`	filter of neighborhoods of x
`Continuous f`	preimages of opens are open, equivalently limits are preserved

Topology appears early in Mathlib because it supports many spaces beyond metric spaces.

Neighborhood Filters

The neighborhood filter `nhds x` packages all neighborhoods of x.



Informally, a property holds along `nhds x` if it holds sufficiently close to x.

What Is a Filter?

A filter is a way to say which sets are "large enough" or "close enough."

- For `atTop` on $\mathbb{N}$, large sets contain all sufficiently large numbers.
- For `nhds a`, large sets contain a neighborhood of a.
- For `nhdsWithin a s`, large sets contain points near a, but only inside s.

A property is eventually true along a filter if it holds on one of these large sets.

Filter Axioms in Natural Language

A collection of large sets should satisfy:

1. the whole space is large;
2. if A is large and $A \subseteq B$, then B is large;
3. if A and B are large, then $A \cap B$ is large.

These are exactly the rules we expect from phrases like:

sufficiently large, sufficiently close, eventually true.

Filter: Formal Definition

A filter F on a type α is a collection

$$F.\mathrm{sets} \subseteq \mathcal{P}(\alpha)$$

satisfying:

$$\begin{aligned}
&\alpha \in F.\mathrm{sets},\\
&x \in F.\mathrm{sets} \ \wedge \ x \subseteq y \Longrightarrow y \in F.\mathrm{sets},\\
&x \in F.\mathrm{sets} \ \wedge \ y \in F.\mathrm{sets} \Longrightarrow x \cap y \in F.\mathrm{sets}.
\end{aligned}$$

- `sets` is the collection of large sets.
- Mathlib allows a degenerate filter; nontriviality is expressed separately by `NeBot`.

Neighborhood Filter in Mathlib

The actual Mathlib definition of `nhds` is written using an infimum of principal filters, but the main reading lemma is:

```
#check nhds
#check mem_nhds_iff
```

$$s \in \mathcal{N}(x) \iff \exists t,\ t \subseteq s \ \wedge\ \mathrm{IsOpen}(t) \ \wedge\ x \in t.$$

- A neighborhood of x is any set containing an open set around x.
- This is the concrete topological meaning behind `nhds x`.

Eventually

```
eventually x in F, P x
```

means:

$$\{x \mid P(x)\} \in F$$

- Along `atTop`: Px holds for all sufficiently large x.
- Along `nhds a`: Px holds for all x sufficiently close to a.

Two Basic Eventually Lemmas

```
#check eventually_atTop
#check Metric.eventually_nhds_iff_ball
```

$$(\forall^{\mathrm{f}}\, n \text{ in } \mathtt{atTop},\ P(n)) \iff \exists N,\ \forall n \geq N,\ P(n).$$

$$(\forall^{\mathrm{f}}\, y \text{ in } \mathcal{N}(x),\ P(y)) \iff \exists \varepsilon > 0,\ \forall y \in B(x, \varepsilon),\ P(y).$$

Common Filters in Analysis

```
#check Filter
#check atTop
#check nhds
#check nhdsWithin
```

Filter	Natural-language reading
`atTop`	sufficiently large inputs
`nhds a`	sufficiently close to a
`nhdsWithin a s`	sufficiently close to a, while staying in s

The Route to Tendsto

`MetricSpace X` gives distance and balls

↓

`TopologicalSpace X` gives neighborhoods

↓

`nhds x : Filter X` describes closeness to x

↓

`Tendsto f source target` describes convergence

Part II: Limits

The goal of this part is to compare two languages:

- epsilon language: concrete and familiar;
- filter language: abstract and uniform;
- Mathlib usually proves limits by composing `Tendsto` lemmas.

Why Filters?

Situation	Traditional notation	Filter statement
sequence limit	$u_n \to L$	`Tendsto u atTop (nhds L)`
function limit	$x \to a$	`Tendsto f (nhds a) ...`
punctured limit	$x \to a,\ x \neq a$	`Tendsto f (nhdsWithin a s) ...`
right limit	$x \to a+$	source: `nhdsWithin a (Set.Ioi a)`
limit at infinity	$x \to +\infty$	`Tendsto f atTop (nhds L)`

The Shape of Tendsto

```
Tendsto f l1 l2
```

Piece	Meaning
`f`	the function being studied
`l1`	how inputs approach
`l2`	how outputs should approach

Do not read `Tendsto` as only a function limit; it is a general convergence statement.

Tendsto: Formal Definition

In Mathlib:

```
def Tendsto (f : alpha -> beta)
    (l1 : Filter alpha) (l2 : Filter beta) :=
  l1.map f <= l2
```

Natural-language reading:

every target-large set has source-large preimage.

Informally:

$$V \in l_2 \quad \Longrightarrow \quad f^{-1}(V) \in l_1.$$

Source Filter and Target Filter

input side	function	output side
`l1`	$\xrightarrow{f}$	`l2`
`atTop`	$\xrightarrow{u}$	`nhds L`
`nhds a`	$\xrightarrow{f}$	`nhds L`

A common beginner mistake is to put the output neighborhood on the source side.

Sequence Limits: Epsilon-N

Undergraduate statement:

$$u_n \to L \quad \Longleftrightarrow \quad \forall \varepsilon > 0,\ \exists N,\ \forall n \geq N,\ |u_n - L| < \varepsilon.$$

Metric-space version:

$$\forall \varepsilon > 0,\ \exists N,\ \forall n \geq N,\ \mathrm{dist}(u_n, L) < \varepsilon.$$

Sequence Limits: Filter Form

Mathlib statement:

```
Tendsto u atTop (nhds L)
```

Reading:

- `atTop`: the index n is eventually large;
- `nhds L`: the output is eventually inside every neighborhood of L;
- together: u_n tends to L.

Function Limits: Epsilon-Delta

Undergraduate statement:

$$\lim_{x \to a} f(x) = L$$

means:

$$\forall \varepsilon > 0,\ \exists \delta > 0,\ \forall x,\ |x - a| < \delta \Rightarrow |f(x) - L| < \varepsilon.$$

Function Limits: Metric Form

In a metric space:

$$\forall \varepsilon > 0,\ \exists \delta > 0,\ \forall x,\ \operatorname{dist}(x, a) < \delta \Rightarrow \operatorname{dist}(f(x), L) < \varepsilon.$$

This is the metric expansion of a topological/filter statement.

Function Limits: Filter Form

Mathlib statement:

```
Tendsto f (nhds a) (nhds L)
```

Reading:

- input side: x approaches a;
- output side: $f(x)$ approaches L;
- `Tendsto` packages the epsilon-delta quantifiers.

Punctured Limits

Undergraduate punctured condition:

$$0 < |x - a| \quad \text{and} \quad |x - a| < \delta.$$

Mathlib reading:

```
Tendsto f (nhdsWithin a s) (nhds L)
```

For a punctured limit, s is a set excluding the point a.

One-Sided and Infinite Limits

Situation	Source filter
$x \to a$ from the right	`nhdsWithin a (Set.Ioi a)`
$x \to a$ from the left	`nhdsWithin a (Set.Iio a)`
$x \to +\infty$	`atTop`
$x \to -\infty$	`atBot`
sequence $n \to \infty$	`atTop` on $\mathbb{N}$

The template `Tendsto f source target` stays the same.

Metric Equivalence: Target Neighborhood

```
#check Metric.tendsto_nhds
```

$$\texttt{Tendsto u l (nhds a)} \iff \forall \varepsilon > 0,\ \forall^{\mathrm{f}} x \text{ in } l,\ \operatorname{dist}(u(x), a) < \varepsilon.$$

With $l = \texttt{atTop}$, this becomes the usual epsilon-N language.

Metric Equivalence: Function Limits

```
#check Metric.tendsto_nhds_nhds
```

```
Tendsto f (nhds a) (nhds b)
```

is equivalent to:

$$\forall \varepsilon > 0,\ \exists \delta > 0,\ \forall x,\ \mathrm{dist}(x, a) < \delta \Rightarrow \mathrm{dist}(f(x), b) < \varepsilon.$$

Metric Equivalence: Restricted Limits

```
#check Metric.tendsto_nhdsWithin_nhds
#check Metric.continuousAt_iff
```

- `Metric.tendsto_nhdsWithin_nhds` gives the epsilon-delta form with $x \in s$.
- Punctured limits are modeled by choosing s to exclude the point.
- `Metric.continuousAt_iff` gives the epsilon-delta definition of continuity at a point.

Filter language is the general interface; epsilon language is the metric expansion.

In Lean: Inspect the Concepts

```
open Filter
open scoped Topology

#check Filter.Tendsto
#check atTop
#check nhds
#check nhdsWithin
#check tendsto_const_nhds
#check tendsto_id
```

- Use `#check` to learn the type of a concept or theorem.
- This is often the fastest way to read Mathlib analysis code.

Live Lean: `codeD4/D4_LimitsContinuity.lean`

In Lean: Compose `Tendsto` Lemmas

```
#check tendsto_const_nhds
#check tendsto_id
#check Tendsto.add
#check Tendsto.mul
#check Continuous.tendsto
```

- We usually do not unfold epsilon-delta definitions.
- We combine reusable theorems about `Tendsto`.
- Algebraic operations on limits are API lemmas.

Continuity at a Point

Undergraduate reading:

$$x \to a \quad \Longrightarrow \quad f(x) \to f(a).$$

Mathlib concepts:

```
ContinuousAt f a
```

```
Tendsto f (nhds a) (nhds (f a))
```

Continuity Everywhere

- `ContinuousAt f a`: continuity at one point.
- `Continuous f`: continuity at every point.
- Continuous functions preserve limits.
- Many proofs use continuity lemmas instead of proving limits directly.

Common examples:

- constant functions;
- identity function;
- sums and products of continuous functions;
- compositions of continuous functions.

Common Reading Mistakes

- Reversing source and target filters in `Tendsto`.
- Treating `nhds a` as a single set rather than a filter of neighborhoods.
- Forgetting that `nhdsWithin a s` restricts the approach to a set.
- Expecting every Mathlib proof to unfold epsilon-delta definitions.
- Ignoring typeclass assumptions such as `MetricSpace X`.

20-minute break

Part III: Derivatives

The goal is to read and use Mathlib's derivative API.

- undergraduate language: $f'(x) = c$;
- proof-friendly Mathlib language: `HasDerivAt f c x`;
- value-level Mathlib language: `deriv f x`;
- derivative calculations are assembled from reusable rules.

Undergraduate vs. Mathlib

Undergraduate notation	Mathlib expression
$f'(x) = c$	`HasDerivAt f c x`
derivative value	`deriv f x`
derivative function	`fun x => deriv f x`
derivative rule	theorem about `HasDerivAt`

The proposition `HasDerivAt` is often the better interface for proofs.

Two Interfaces

```
#check deriv
#check HasDerivAt
```

- `deriv f x` is a value.
- `HasDerivAt f c x` is a proposition.
- Rules such as sum, product, and chain rule usually produce `HasDerivAt` statements.

Live Lean: `codeD4/D4_Differentiation.lean`

What Does `HasDerivAt` Say?

Informally:

$$\texttt{HasDerivAt f c x} \quad \text{means} \quad f(x+h) = f(x) + ch + o(h).$$

For this lecture:

- read it as "the derivative of f at x is c";
- use API rules instead of unfolding the definition;
- keep the focus on theorem statements and proof composition.

Derivative Rule Map

basic derivatives

↓

identity and constant functions

↓

addition, subtraction, multiplication, scalar multiplication

↓

powers and polynomials

↓

chain rule and composed functions

Basic Derivative API

```
#check hasDerivAt_id
#check hasDerivAt_const
#check HasDerivAt.add
#check HasDerivAt.mul
#check HasDerivAt.comp
```

- Start from derivative facts for simple functions.
- Combine them with methods such as `.add`, `.mul`, and `.comp`.
- Use `simpa` to align the final expression with the goal.

Identity and Constant Functions

```
example (x : Real) :
    HasDerivAt (fun y : Real => y) 1 x := by
  simpa using (hasDerivAt_id x)

example (c x : Real) :
    HasDerivAt (fun _ : Real => c) 0 x := by
  simpa using (hasDerivAt_const (x := x) (c := c))
```

These are the derivative API's base cases.

Powers

```
#check hasDerivAt_pow
#check deriv_pow
```

Mathlib knows the usual rule:

$$\frac{d}{dx}x^n = nx^{n-1}.$$

In classroom examples, powers give good low-friction demonstrations.

Example: $x \mapsto x^2$

```
example (x : Real) :
    HasDerivAt (fun y : Real => y ^ 2) (2 * x) x := by
  simpa using (hasDerivAt_pow 2 x)
```

- The mathematical derivative is $2x$.
- The Lean proof delegates the calculation to `hasDerivAt_pow`.
- `simpa` performs the small normalization needed for the exact goal.

Example: $x \mapsto x^3 + x$

```
example (x : Real) :
    HasDerivAt (fun y : Real => y ^ 3 + y)
      (3 * x ^ 2 + 1) x := by
  have hpow :
      HasDerivAt (fun y : Real => y ^ 3) (3 * x ^ 2) x := by
    simpa using (hasDerivAt_pow 3 x)
  have hid : HasDerivAt (fun y : Real => y) 1 x := by
    simpa using (hasDerivAt_id x)
  simpa using hpow.add hid
```

From HasDerivAt to deriv

```
example (x : Real) :
    deriv (fun y : Real => y ^ 2) x = 2 * x := by
  simp

example (x : Real) :
    deriv (fun y : Real => y ^ 3 + y) x = 3 * x ^ 2 + 1 := by
  simp
```

- `deriv` is convenient for final value statements.
- For proof construction, `HasDerivAt` often gives more control.

Chain Rule Map

For $x \mapsto (x+1)^2$:

$$g(x) = x + 1, \text{ derivative } 1$$
$$\downarrow$$
$$h(z) = z^2, \text{ derivative } 2z$$
$$\downarrow$$
$$h(g(x)) = (x+1)^2$$
$$\downarrow$$
$$\text{chain rule: derivative } 2(x+1)$$

This is a good live search exercise rather than a memorization target.

Chain Rule Search Targets

```
#check HasDerivAt.comp
#check HasDerivAt.add
#check hasDerivAt_pow
#check hasDerivAt_id
#check hasDerivAt_const
```

- Ask: which theorem has the conclusion shape we need?
- If algebraic expressions differ syntactically, try `simpa` or a normalization tactic.
- This example models realistic Mathlib navigation.

Part IV: Integrals

This part gives a map of the integration API.

- interval integral notation;
- simple examples such as constant functions;
- the fundamental theorem of calculus as an API bridge;
- proof workflow for finding and repairing Mathlib proofs.

Why Integration Needs More Infrastructure

Differentiation can often be taught through pointwise rules.

Integration in Mathlib is built to cover:

- real and vector-valued functions;
- oriented intervals;
- measure-theoretic integration;
- integrability assumptions;
- fundamental theorem of calculus statements.

The lecture goal is to learn the map, not the whole measure theory.

Interval Integral Notation

```
#check intervalIntegral
#check IntervalIntegrable
```

Mathematical notation:

$$\int_a^b f(x)\,dx$$

Mathlib notation:

```
integral x in a..b, f x
```

Live Lean: `codeD4/D4_Integration.lean`

Oriented Intervals

The interval integral is oriented:

$$\int_a^b f(x)\,dx = -\int_b^a f(x)\,dx.$$

Important special case:

$$\int_a^a f(x)\,dx = 0.$$

This is why theorem names include words such as `same` and `symm`.

Basic Integral API

```
#check intervalIntegral.integral_same
#check intervalIntegral.integral_symm
#check intervalIntegral.integral_const
#check intervalIntegral.integral_add
```

- Use `#check` to inspect exact assumptions.
- Some linearity theorems include integrability hypotheses.
- Constant-function examples are good classroom entry points.

Example: Constant Function

```
example (a b c : Real) :
    (integral x in a..b, c) = (b - a) * c := by
  simpa using
    (intervalIntegral.integral_const (a := a) (b := b) (c := c))
```

This is a good first integral theorem because the statement matches the undergraduate formula.

Example: Same Endpoints

```
example (a : Real) (f : Real -> Real) :
    (integral x in a..a, f x) = 0 := by
  simp
```

The simplifier knows many basic interval-integral facts.

Linearity Map

Informal rules:

$$\int_a^b (f+g) = \int_a^b f + \int_a^b g, \qquad \int_a^b cf = c\int_a^b f.$$

Mathlib versions may require:

- `IntervalIntegrable f volume a b;`
- `IntervalIntegrable g volume a b;`
- scalar and normed-space structure.

Linearity API

```
#check intervalIntegral.integral_add
#check intervalIntegral.integral_sub
#check intervalIntegral.integral_const_mul
#check intervalIntegral.integral_mul_const
```

- The theorem statement tells you the hypotheses.
- Do not guess the assumptions; inspect them.
- This is typical for analysis APIs in Mathlib.

FTC I: Integral as an Antiderivative

Undergraduate shape:

$$F(x) = \int_a^x f(t)\,dt \quad \Longrightarrow \quad F'(x) = f(x).$$

Mathlib connects:

- continuity of f;
- the interval integral $x \mapsto \int_a^x f(t)\,dt$;
- a derivative statement such as `HasDerivAt`.

FTC I: API Entry Points

```
#check intervalIntegral.integral_hasDerivAt_right
#check intervalIntegral.deriv_integral_right
#check intervalIntegral.integral_hasDerivAt_left
```

- These are theorem names to inspect, not names to memorize.
- The exact theorem depends on endpoint direction and differentiability form.

FTC II: Integral of a Derivative

Undergraduate shape:

$$F' = f \quad \Longrightarrow \quad \int_a^b f(x)\,dx = F(b) - F(a).$$

In Lean, the statement must say precisely:

- where F has derivative f;
- which interval is used;
- which continuity or integrability hypotheses are available.

FTC II: API Entry Points

```
#check intervalIntegral.integral_eq_sub_of_hasDerivAt
#check intervalIntegral.integral_deriv_eq_sub
#check intervalIntegral.integral_deriv_eq_sub'
```

- Use `#check` before trying to apply an FTC theorem.
- The theorem name often reveals the conclusion shape.

Reading FTC Hypotheses

When reading an FTC theorem, ask:

1. Is the conclusion about `HasDerivAt`, `deriv`, or an integral equality?
2. Is the integral from a to x, or from x to a?
3. Does the theorem require continuity, differentiability, or interval integrability?
4. Are endpoint or interval assumptions hidden in the statement?

Proof Workflow

A practical Mathlib workflow:

1. read the goal and local context;
2. inspect definitions and theorem statements with `#check`;
3. search for API lemmas;
4. start with a small version of the target;
5. use Lean errors to repair theorem names and assumptions;
6. keep the final proof small and verified.

Live Lean: `codeD4/D4_ProofWorkflow.lean`

Workflow Example: Continuity to Limit

```
#check Continuous.tendsto
#check continuous_id
#check continuous_const

example (a c : Real) :
    Tendsto (fun x : Real => x + c) (nhds a) (nhds (a + c)) := by
  simpa using (continuous_id.add continuous_const).tendsto a
```

This shows how Part II's `Tendsto` statements arise from continuity.

Workflow Example: Derivative Repair

```
#check hasDerivAt_pow
#check HasDerivAt.add

example (x : Real) :
    HasDerivAt (fun y : Real => y ^ 2 + y) (2 * x + 1) x := by
  have hsq := hasDerivAt_pow 2 x
  have hid := hasDerivAt_id x
  simpa using hsq.add hid
```

If this fails, inspect the exact derivative expression and normalize the target.

AI Drafts and Lean Verification

AI is useful for drafts. Lean is the verifier. Mathlib search repairs the gap.

Good uses of AI:

- propose possible theorem names;
- suggest proof structure;
- explain errors in mathematical language;
- produce a first draft to be checked and corrected.

Suggested Exercises

- Explain `Tendsto u atTop (nhds L)` in epsilon-N language.
- Use `Metric.tendsto_nhds_nhds` to identify the epsilon-delta form of a function limit.
- Prove a simple continuity statement by composing known continuous functions.
- Prove or inspect the derivative of $x \mapsto x^2$ and $x \mapsto x^3 + x$.
- Compute the interval integral of a constant function.
- Repair a deliberately incomplete proof using `#check` and theorem search.

Live Lean: `codeD4/D4_Exercises.lean`