BCS Calculus — All Lecture Slides

EPITA — Calculus for Computer Scientists

Jim Newton

Table of Contents

Course Introduction

EPITA — Calculus for Computer Scientists

Jim Newton

Coverage

Orientation only — no handout chapter is covered here.

Presented alongside `CM-1-slides.tex` (Chapter 1) on the first day of class.

Course Objectives

- Understand why calculus is relevant to computer scientists: optimization in ML, numerical simulation, systems that evolve over time.
- CS-first framing: derivatives introduced as **symbolic reduction rules** before their limit definition — mirroring AST pattern-matching you already know.
- Identify the two central tools — the **derivative** and the **integral** — and their connection via the Fundamental Theorem of Calculus.
- Know the role of Python, NumPy, and SciPy: computational tools *and* a medium for building intuition.

Why Calculus for CS?

Machine Learning
Gradient descent minimises a loss function.

Algorithms
Complexity analysis: continuous approximations of discrete sums.

Systems
ODEs model queues, networks, heat dissipation.

Signal Processing
Fourier analysis integrates products of functions.

The key insight

We *model* discrete, jagged processes with smooth functions, apply calculus to the model, then interpret the result.

Error = numerical error + modelling error.

CS-First Framing

Traditional approach

Limits $\rightarrow$ derivatives $\rightarrow$ integrals

This course

Derivatives as reduction rules $\rightarrow$ limits (definition) $\rightarrow$ integrals

- You already know pattern-matching on abstract syntax trees.
- The derivative rules *are* pattern-matching on expression trees.
- Code and mathematics mirror each other — each differentiation rule becomes exactly one `case` branch in `derivative.py`.
- Foundations (limits, ε-δ) come *after* you are fluent with derivatives as a computational tool.

Types of Sessions

Type	Description
CM	*Cours Magistral* — 2 h lecture, new material
TP	*Travaux Pratiques* — hands-on computing (NumPy, SciPy)
HW	Written homework, pen-and-paper, **not graded** but exam is drawn from it
Oral Defense	**Group** project: `derivative.py`; graded by live oral exam
Intra	**Individual** assignments (e.g. `riemann_sum.py`); submitted to Forge
Exam	Single written final; drawn largely from HW problems

Grading

Component	Description	Weight
Final Exam	Written, covers entire course	40%
Oral Defense	Group project (`derivative.py`), individual oral exam	30%
Intra	Auto-graded programming assignments	15%
Quizzes	Unannounced, start of some CM sessions	10%
Summary	One-page summary of the course	5%
		100%

How to Read the Handout

Margin icons

★ **Essential** — must know for exam

∀ **Recall** — from another course

△ **Warning** — common mistake

ˆ **Enrichment** — optional deeper material

Colored boxes

Theorem, Definition, Example, Proof, Lemma, Corollary, Proposition, Listing, Activity

Each box type has a distinct background color. Skim the box type first, then decide how deeply to read.

The Two Big Ideas

The Derivative

Measures *instantaneous rate of change*.

$f'(x)$ = slope of tangent at x.

Tools: reduction rules, chain rule, implicit differentiation.

The Integral

Measures *total accumulation*.

$\int_a^b f(x)\,dx$ = signed area under curve.

Tools: Riemann sums, FTC, substitution, partial fractions.

The Fundamental Theorem of Calculus

These two tools are inverses of each other. This connection — profound and surprising — is the central result of the course.

CM-1: The Derivative as Reduction Rules

EPITA — Calculus for Computer Scientists

Jim Newton

Coverage

Preceded by `intro-slides.tex` (course overview) on the first day of class.

This lecture covers:

- Chapter 1: The Derivative as Reduction Rules — full, *except* composition (the Chain Rule gets its own chapter, Chapter 2, next time)

Chapter Objectives

By the end of this chapter you should be able to:

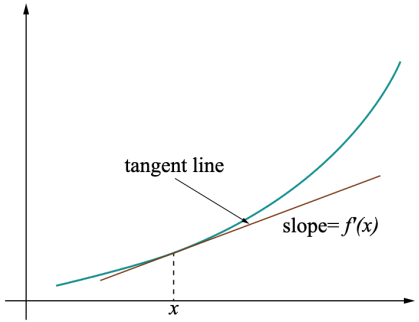
- Recall the derivatives of constants, the identity, x^n (Power Rule), e^x, sin, cos, and tan by hand; the full table, including ln and inverse trig, comes in Chapter 2.
- Use **linearity**, the product rule, and the quotient rule step by step. (The chain rule — for *compositions* — is Chapter 2.)
- Extending term-by-term differentiation from a finite sum to an *infinite* power series, as with e^x, is a tempting but unjustified leap — it needs its own theorem, not just linearity.

What is a Derivative?

Geometric intuition

At every point on the graph of a differentiable function f, there is a unique **tangent line**. The **slope** of that tangent line at x is the value of the derivative $\mathcal{D}_x f(x)$.

- If f is differentiable everywhere, $\mathcal{D} f$ is itself a function $\mathbb{R} \to \mathbb{R}$.
- We write $\mathcal{D}_x f(x) = m$ when we want to name the input variable x.
- We write $\mathcal{D} f$ when no particular variable is singled out.



CS Perspective: Discrete Meets Continuous

Natural objection

Network throughput, CPU usage, packet counts are *discrete* and discontinuous. How can calculus apply?

Answer: we model.

1. Replace the discrete reality with a smooth function (the model).
2. Apply calculus to the model — getting exact results.
3. Total error = numerical error + modelling error.

This modelling step is standard in engineering and applied science. It is the reason calculus is relevant to computer scientists.

Approximating the Derivative

If f represents a measurable process, approximate the derivative at x using two nearby measurements:

$$\mathcal{D}_x f(x) \approx \frac{f(x_2) - f(x_1)}{x_2 - x_1} \qquad (x_1 \approx x_2 \approx x)$$

Example: Acceleration from a speedometer

Speed at t_1: $50\,\text{km/h}$; speed at $t_2 = t_1 + 0.5\,\text{s}$: $60\,\text{km/h}$.

$$a \approx \frac{60 - 50}{0.5}\,\frac{\text{km/h}}{\text{s}} = 20\,\frac{\text{km}}{\text{h}\cdot\text{s}} \approx 5.56\,\frac{\text{m}}{\text{s}^2}$$

Unit conversion: multiply by $\frac{1000\,\text{m}}{\text{km}}$, $\frac{1\,\text{h}}{3600\,\text{s}}$ (both $= 1$).

Tangent Line — Common Misconceptions

Geometry vs. calculus

In geometry: a tangent to a *circle* touches it at exactly one point. This definition **breaks down** for general curves.

Tangent can touch infinitely often:
The line $y = 1$ is tangent to $y = \cos x$ at $x = 0$, yet meets the curve once per period.

Tangent can cross the curve:
The line $y = x$ is tangent to $y = \sin x$ at the origin and crosses it there.

Correct definition: the tangent line at a point is the *limiting position of a secant line* as the two points are brought together. (Made precise in Chapter 6.)

Differential Rules — Overview

We treat differentiation as a set of **reduction rules** (like pattern-matching on an AST).

Rule	**Formula**
Constant	$\mathcal{D}\, c = 0$
Identity	$\mathcal{D}_x x = 1$
Power	$\mathcal{D}_x x^n = n x^{n-1}$
Scalar	$\mathcal{D}\,(c\, g) = c\, \mathcal{D}\, g$
Sum	$\mathcal{D}\,(h + g) = \mathcal{D}\, h + \mathcal{D}\, g$
Product	$\mathcal{D}\,(h\, g) = h\, \mathcal{D}\, g + g\, \mathcal{D}\, h$
Quotient	$\mathcal{D}\,(h/g) = \dfrac{g\, \mathcal{D}\, h - h\, \mathcal{D}\, g}{g^2}$

One rule is missing on purpose: **Chain** (composition, $h \circ g$) — hard enough to deserve its own chapter,

Constant, Identity, and Power Rules

Constant Rule

$\mathcal{D}\, c = 0$ (a horizontal line has slope 0)

Identity Rule

$\mathcal{D}_x x = 1$ (the line $y = x$ has slope 1 everywhere)

Power Rule ($n \in \mathbb{R}$)

$$\mathcal{D}_x x^n = n\, x^{n-1}$$

Examples

$\mathcal{D}_x x^6 = 6x^5$ $\quad$ $\mathcal{D}_x x^{-6} = -6x^{-7}$ $\quad$ $\mathcal{D}_x x^{\sqrt{2}} = \sqrt{2}\, x^{\sqrt{2}-1}$ (for $x > 0$)

Linearity of the Derivative

The Scalar Rule and Sum Rule together say: differentiation is a **linear operator**.

$$\mathcal{D}(\alpha f + \beta g) = \alpha\,\mathcal{D}f + \beta\,\mathcal{D}g$$

Why this matters:

- Distribute $\mathcal{D}$ across a sum; pull constants out.
- Polynomials, Taylor series, Fourier series — all reduce to differentiating *one term at a time*.
- The definite integral is also linear — two instances of the same structural idea.

Derivative of a polynomial

$$\mathcal{D}_x(3x^4 - 5x^2 + x - 1) = 12x^3 - 10x + 1$$

Product Rule

Product Rule

$$\mathcal{D}(hg) = h\,\mathcal{D}\,g + g\,\mathcal{D}\,h$$

Special case (Scalar Rule): if $h = c$ constant, $\mathcal{D}(cg) = c\,\mathcal{D}\,g$.

Example: $\mathcal{D}_x(x^3 \cos x)$

Let $h(x) = x^3$, $g(x) = \cos x$. $\quad \mathcal{D}\,h = 3x^2, \quad \mathcal{D}\,g = -\sin x$.
$\mathcal{D}_x(x^3 \cos x) = x^3(-\sin x) + \cos x\,(3x^2) = 3x^2 \cos x - x^3 \sin x$

Quotient Rule

Quotient Rule

$$\mathcal{D}\,\frac{h}{g} = \frac{g\,\mathcal{D}\,h - h\,\mathcal{D}\,g}{g^2}$$

Valid wherever $g \neq 0$.

Proof: deferred to Chapter 2 — it needs the Chain Rule (write $h/g = h \cdot g^{-1}$, then Product Rule, then Chain Rule on g^{-1}). For now, just use the formula.

Derivative of $\tan x$

$$\mathcal{D}\tan = \mathcal{D}\,\frac{\sin}{\cos} = \frac{\cos\cdot\cos - \sin\cdot(-\sin)}{\cos^2} = \frac{\cos^2 + \sin^2}{\cos^2} = \sec^2$$

Special Functions: Derivative Table (excerpt)

$f(x)$	$\mathcal{D}_x f(x)$	Note
e^x	e^x	only function = its own derivative
$\sin x$	$\cos x$	
$\cos x$	$-\sin x$	
$\tan x$	$\sec^2 x$	via Quotient Rule
$\arctan x$	$\dfrac{1}{1+x^2}$	
$\arcsin x$	$\dfrac{1}{\sqrt{1-x^2}}$	
$\ln \lvert x \rvert$	$\dfrac{1}{x}$	

e^x: The Fixed Point of Differentiation

Key fact

$$\mathcal{D}_x e^x = e^x$$

e^x is the **only non-zero function** that equals its own derivative.

Why it matters: any process whose *rate of change is proportional to its current value* is described by e^x.

- Exponential growth and decay
- Compound interest
- Radioactive decay
- Algorithm analysis ($O(e^n)$ complexity)
- Solving ODEs: $y' = y$, $y(0) = 1 \Rightarrow y(t) = e^t$

Worked Example: Differentiating a Power Series

Several functions (e^x, $\sin x$, $\cos x$) are given by an **infinite** series, not a finite formula. Differentiate term by term, just like a polynomial (justified rigorously later; taken as given here).

e^x reproduces itself

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots \quad \Longrightarrow \quad \mathcal{D}_x e^x = 1 + x + \frac{x^2}{2!} + \cdots = e^x$$

e^{-x}: substitute first, then differentiate

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \cdots \quad \Longrightarrow \quad \mathcal{D}_x e^{-x} = -1 + x - \cdots = -e^{-x}$$

No Chain Rule needed — we substituted $-x$ into the series *before* differentiating.

Worked Example: A Companion Pair

Let $g(x) = e^x + e^{-x}$ and $h(x) = e^x - e^{-x}$. No new series work needed — just the Sum/Difference Rule on the two derivatives just computed:

$$\mathcal{D}\, g = \mathcal{D}\, e^x + \mathcal{D}\, e^{-x} = e^x - e^{-x} = h \qquad \mathcal{D}\, h = \mathcal{D}\, e^x - \mathcal{D}\, e^{-x} = e^x + e^{-x} = g$$

The pattern to remember

Differentiating g gives h; differentiating h gives back g — the two functions swap under differentiation. (Up to a factor of 2, these are $\cosh x$ and $\sinh x$ — the name isn't important here.)

You'll see this exact pattern again, with one added twist (alternating signs), for $\sin x$ later in the course.

Chapter 1 Summary

- The derivative is the **slope of the tangent line** — computed mechanically, without limits (for now).
- Five reduction rules cover compound expressions built without composition: Constant, Power, Scalar, Sum, Product, Quotient.
- **Linearity** is the key structural property: $\mathcal{D}(\alpha f + \beta g) = \alpha\,\mathcal{D}f + \beta\,\mathcal{D}g$ — and it extends to *infinite* sums (Taylor series), not just finite ones.
- e^x, e^{-x}, and $e^x \pm e^{-x}$: derivatives computed directly from their series, no new rules needed.
- Special functions worth memorizing: e^x, sin, cos, tan.

Coming next

Chapter 2: Composition and the Chain Rule — the one reduction rule we deliberately skipped, plus what it unlocks: a proof of the Quotient Rule and implicit differentiation.

CM-2: Composition and the Chain Rule

EPITA — Calculus for Computer Scientists

Jim Newton

Coverage

This lecture covers:

- Chapter 2: Composition and the Chain Rule — **full** (the Chain Rule, the proof of the Quotient Rule, the complete derivative table, implicit differentiation, and the derivative of arctan)

Chapter Objectives

- Compute the derivative of a **composition** $h(g(x))$ using the Chain Rule, in point-free, Newtonian, and Leibniz notation.
- Prove the Quotient Rule, using the Chain Rule together with the Power Rule.
- Apply **implicit differentiation** to find dy/dx when y is defined by a relation $F(x, y) = 0$ rather than an explicit formula.
- Prove the derivative of arctan using implicit differentiation.

The Chain Rule: Differentiating a Composition

Chapter 1 covered every reduction rule *except* one: what happens when you differentiate $h(g(x))$, a composition?

Chain Rule

$$\mathcal{D}(h \circ g) = (\mathcal{D}\, h \circ g) \cdot \mathcal{D}\, g \qquad \text{pointwise: } \mathcal{D}_x h(g(x)) = \mathcal{D}_x h(g(x)) \cdot \mathcal{D}_x g(x)$$

Example: $\mathcal{D}_x(x^3)^2$

Let $g(x) = x^3$, $h(t) = t^2$. $\quad \mathcal{D}\, h(t) = 2t, \quad \mathcal{D}\, g(x) = 3x^2$.
$\mathcal{D}_x h(g(x)) = 2(x^3) \cdot 3x^2 = 6x^5$ (agrees with the Power Rule directly).

The hard part isn't the formula — it's recognizing which function is "outer" (h) and which is "inner" (g).

Chain Rule: Three Notations

Point-free

$\mathcal{D}\,(h \circ g) = (\mathcal{D}\,h \circ g) \cdot \mathcal{D}\,g$ — no input variable named.

Newtonian

$[h(g(x))]' = h'(g(x)) \cdot g'(x)$

Leibniz

$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$, where $u = g(x)$, $y = h(u)$.

Leibniz notation is inherently pointwise — it always names variables and values, never functions — which is why it needs the auxiliary variable u and a substitution step at the end. It has no point-free form.

Proof of the Quotient Rule

Chapter 1 stated the Quotient Rule without proof. Now we have what it needs: Chain Rule + Power Rule.

$$\mathcal{D}\,\frac{h}{g} = \mathcal{D}\left(h\,g^{-1}\right) \overbrace{=}^{\text{Product Rule}} h\,\mathcal{D}\left(g^{-1}\right) + g^{-1}\mathcal{D}\,h \overbrace{=}^{\text{Chain + Power}} -h\,g^{-2}\mathcal{D}\,g + g^{-1}\mathcal{D}\,h$$

$$= \frac{\mathcal{D}\,h}{g} - \frac{h\,\mathcal{D}\,g}{g^2} = \frac{g\,\mathcal{D}\,h - h\,\mathcal{D}\,g}{g^2}$$

Key step: $\mathcal{D}\left(g^{-1}\right)$ is a composition ($t \mapsto t^{-1}$ applied to g), so it needs the Chain Rule — together with the Power Rule for $\mathcal{D}\left(t^{-1}\right) = -t^{-2}$.

The Complete Derivative Table

f	$f(x)$	$\mathcal{D}_x f(x)$
$h+g,\ h-g$	—	$\mathcal{D}_x h(x) \pm \mathcal{D}_x g(x)$
hg	—	$h(x)\mathcal{D}_x g(x) + g(x)\mathcal{D}_x h(x)$
$h \circ g$	$h(g(x))$	$\mathcal{D}_x h(g(x)) \cdot \mathcal{D}_x g(x)$
h/g	—	$\dfrac{g(x)\mathcal{D}_x h(x) - h(x)\mathcal{D}_x g(x)}{[g(x)]^2}$
$\sin, \cos, \tan$	—	$\cos x,\ -\sin x,\ \sec^2 x$
$\arcsin, \arccos, \arctan$	—	$\dfrac{1}{\sqrt{1-x^2}},\ \dfrac{-1}{\sqrt{1-x^2}},\ \dfrac{1}{1+x^2}$
$\exp,\ \ln\lvert x\rvert$	e^x, —	$e^x,\ \dfrac{1}{x}$

Every rule from Chapters 1 and 2 is now in hand — this is the complete reference table. The arctan entry is proved next.

Implicit Differentiation

When y is defined implicitly by $F(x, y) = 0$, differentiate both sides with respect to x, treating y as a function of x.

Example: circle $x^2 + y^2 = 1$

$$\frac{d}{dx}(x^2 + y^2) = \frac{d}{dx}(1) \;\Rightarrow\; 2x + 2y\,\frac{dy}{dx} = 0 \;\Rightarrow\; \frac{dy}{dx} = -\frac{x}{y}$$

Why it matters:

- Some curves have no explicit $y = f(x)$ formula.
- Works even when the Implicit Function Theorem only guarantees *existence* of y.
- Key application: deriving $(\arctan x)' = \frac{1}{1+x^2}$.

Deriving $(\arctan x)' = \dfrac{1}{1+x^2}$

Let $y = \arctan x$, so $\tan y = x$.
Differentiate both sides with respect to x:

$$\sec^2 y \cdot \frac{dy}{dx} = 1 \;\Rightarrow\; \frac{dy}{dx} = \cos^2 y$$

Now use $\tan y = x$ and the identity $1 + \tan^2 y = \sec^2 y$:

$$\cos^2 y = \frac{1}{\sec^2 y} = \frac{1}{1+\tan^2 y} = \frac{1}{1+x^2}$$

Result

$$\mathcal{D}_x \arctan x = \frac{1}{1+x^2}$$

CM-2 Summary

- Chain Rule: $\mathcal{D}_x h(g(x)) = \mathcal{D}_x h(g(x)) \cdot \mathcal{D}_x g(x)$ — point-free, Newtonian, and Leibniz notation.
- Quotient Rule, finally proved: $h/g = h\,g^{-1}$, then Product Rule + Chain Rule.
- The derivative table is now complete.
- Implicit differentiation: differentiate $F(x, y) = 0$, solve for dy/dx — works even without an explicit $y = f(x)$ formula.
- Chapter 2 is now fully covered, including the proof that $\mathcal{D}_x \arctan x = \frac{1}{1+x^2}$.

Coming next: CM-3

The Shape of a Function — using f' and f'' to sketch a graph without plotting it point by point.

CM-3: The Shape of a Function

EPITA — Calculus for Computer Scientists

Jim Newton

Coverage

This lecture covers:

- Chapter 3: The Shape of a Function — **full**

Chapter Objectives

- Use the sign of f' to determine where f is increasing, decreasing, or constant; locate local extrema from sign changes.
- Use the sign of f'' to determine concavity and identify inflection points.
- Apply the First and Second Derivative Tests to classify critical points.
- Find global extrema on a closed interval.

The Running Example

With every differentiation rule available, we turn to a new question: given a formula for f, what does its *graph* look like — without plotting it point by point?

The function we will reconstruct

$$f(x) = \tfrac{1}{8}x^4 + \tfrac{1}{6}x^3 - 4x^2 - 8x + 25$$

Strategy: read f's shape off f' and f'' — the only two pieces of information we are allowed to use.

Polynomial Form vs. Factored Form

$f(x)$ — polynomial form

$$f(x) = \tfrac{1}{8}x^4 + \tfrac{1}{6}x^3 - 4x^2 - 8x + 25$$

$f'(x)$ — both forms

$$\tfrac{1}{2}x^3 + \tfrac{1}{2}x^2 - 8x - 8$$
$$= \tfrac{1}{2}(x+4)(x-4)(x+1)$$

Roots: -4, -1, 4.

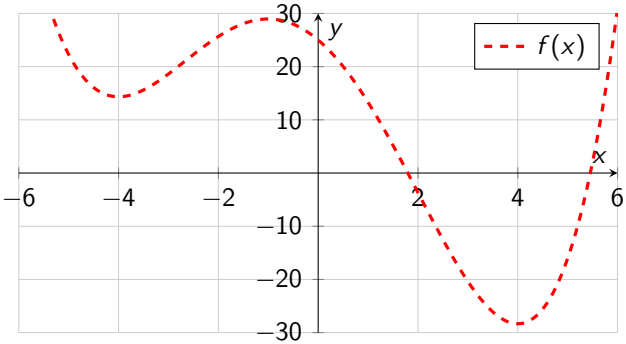
$f''(x)$ — both forms

$$\tfrac{3}{2}x^2 + x - 8$$
$$= \tfrac{1}{2}(3x+8)(x-2)$$

Roots: $-\tfrac{8}{3}$, 2.

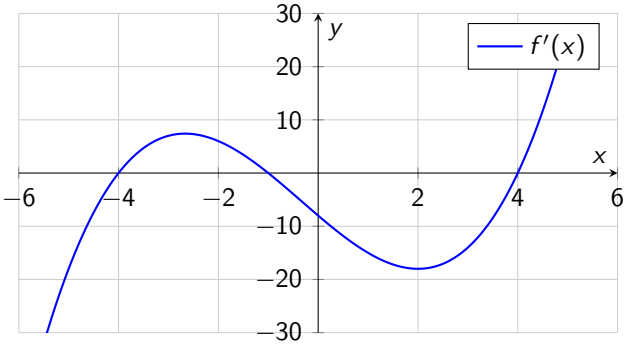
We never need to factor f itself — its shape comes entirely from the roots of f' and f'' above.

Reading f From f' and f''



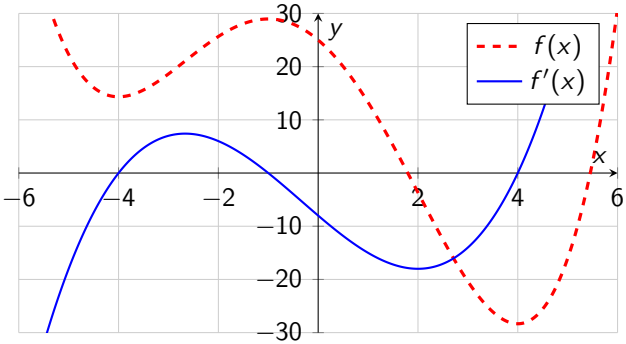
$f(x)$ — the quartic we want to sketch.

Reading f From f' and f''



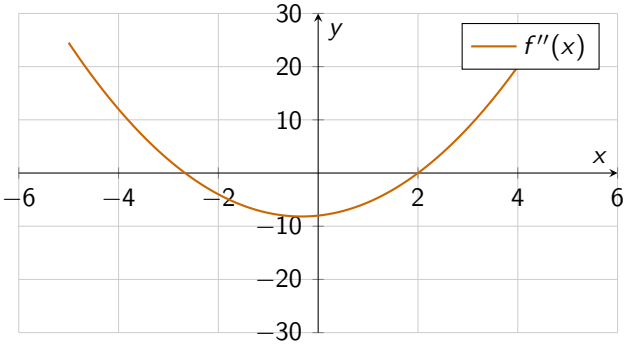
$f'(x)$ — where is it positive? negative? zero?

Reading f From f' and f''



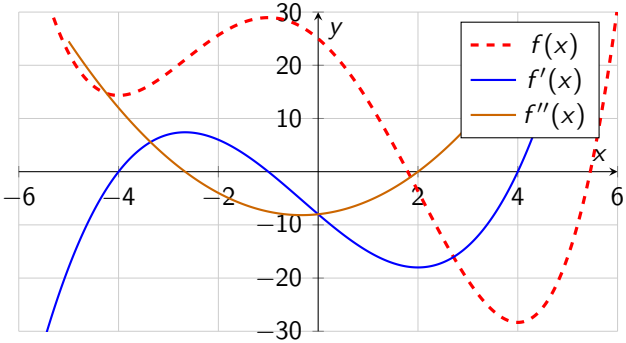
$f(x)$ **and** $f'(x)$ **together** — f increases exactly where $f' > 0$.

Reading f From f' and f''



$f''(x)$ — where is it positive? negative? zero?

Reading f From f' and f''



All three together — everything we need to sketch f, read directly off f' and f''.

Guiding Question: What is the Shape?

Derivative	**Sign**	**Conclusion**
$f'(x)$	> 0	f increasing on that interval
$f'(x)$	< 0	f decreasing on that interval
$f'(x)$	$= 0$	critical point (candidate for extremum)
$f''(x)$	> 0	f concave up (cup shape)
$f''(x)$	< 0	f concave down (cap shape)
$f''(x)$ changes sign		inflection point

Terminology

Extrema

A **local minimum**/**local maximum** — collectively, **extrema** (singular **extremum**).

Critical Point

A point c where $f'(c) = 0$ *or* $f'(c)$ does not exist.

Inflection Point

A point where f'' changes sign — i.e. where the concavity of f reverses.

A Critical Point Is Only a *Candidate*

$f(x) = x^2$

$f'(x) = 2x$, critical at $x = 0$.
$f''(0) = 2 > 0$: **genuine local min**.

$f(x) = x^3$

$f'(x) = 3x^2$, critical at $x = 0$.
f' never changes sign at 0: **no extremum at all**.

The lesson

$f'(c) = 0$ does not by itself imply a sign change. A critical point must always be *tested further*.

First and Second Derivative Tests

First Derivative Test

Suppose c is a critical point of f.

- f' changes $+ \to -$ at c: **local maximum**.
- f' changes $- \to +$ at c: **local minimum**.
- f' does not change sign at c: **neither**.

Second Derivative Test

If $f'(c) = 0$:

- $f''(c) > 0$: local **minimum**.
- $f''(c) < 0$: local **maximum**.
- $f''(c) = 0$: **inconclusive** — fall back on the First Derivative Test.

Worked Example: Sketching the Quartic

Step 1: $f'(x) = \frac{1}{2}x^3 + \frac{1}{2}x^2 - 8x - 8 = \frac{1}{2}(x+1)(x-4)(x+4)$

Critical points: $x = -4, -1, 4$.

Step 2: Sign chart of f':

	$(-\infty, -4)$	-4	$(-4, -1)$	-1	$(-1, 4)$	4	$(4, \infty)$
f'	$-$	0	$+$	0	$-$	0	$+$
f	$\searrow$	min	$\nearrow$	max	$\searrow$	min	$\nearrow$

Step 3: $f''(x) = \frac{3}{2}x^2 + x - 8 = \frac{1}{2}(3x+8)(x-2)$.

Confirming with the Second Derivative Test

c	sign change of f'	$f''(c)$	conclusion
-4	$- \to +$	$\frac{1}{2}(-4)(-6) = 12 > 0$	local minimum
-1	$+ \to -$	$\frac{1}{2}(5)(-3) = -7.5 < 0$	local maximum
4	$- \to +$	$\frac{1}{2}(20)(2) = 20 > 0$	local minimum

Both tests agree. Comparing the two local minima, $f(-4) = \frac{43}{3} \approx 14.3$ vs. $f(4) = -\frac{85}{3} \approx -28.3$: the **global** minimum is at $x = 4$.

Worked Example: $\sin x$

$$f(x) = \sin x, \qquad f'(x) = \cos x, \qquad f''(x) = -\sin x$$

Three curves every student already recognizes — read shape off them *by inspection*, no equation-solving needed.

- Critical points where $\cos x = 0$: $x = \frac{\pi}{2} + k\pi$ — alternating local max/min, matching the peaks and troughs of $\sin x$ itself.
- Inflection points where $\sin x = 0$: $x = k\pi$ — concavity flips exactly where the curve crosses the axis.

Since $f'' = -f$, the curve always bends back toward the axis — a feature special to sin and cos.

Worked Example: e^{2x} — No Features to Find

$$f(x) = e^{2x}, \qquad f'(x) = 2e^{2x}, \qquad f''(x) = 4e^{2x}$$

f' and f'' are both *positive multiples of f itself*, and $e^{2x} > 0$ always — so neither ever vanishes.

- No critical points: f is strictly increasing on all of $\mathbb{R}$.
- No inflection points: f is concave up on all of $\mathbb{R}$.

The point

"No features" is itself a conclusion to be *verified from the derivatives* — not an exception to the chapter's tools.

Global Extrema on a Closed Interval

Extreme Value Theorem

A continuous function on a closed interval $[a, b]$ attains its global maximum and minimum.

Algorithm (closed interval method):

1. Find all critical points inside (a, b).
2. Evaluate f at each critical point *and* at the endpoints a, b.
3. The largest value is the global max; the smallest is the global min.

Warning

On an *open* or unbounded interval, global extrema may not exist. Always state the domain.

Endpoints Can Win Even With No Critical Points

$f(x) = x + 1$ on $[0, 1]$

$f'(x) = 1$ everywhere — *no critical points at all.*

Yet f still has a global minimum and maximum on $[0, 1]$:

$$f(0) = 1 \text{ (global min)}, \qquad f(1) = 2 \text{ (global max)}$$

The lesson

Checking critical points is not enough — always check the endpoints too, and compare every value together.

CM-3 Summary

- $f' > 0$: increasing; $f' < 0$: decreasing; $f' = 0$ or undefined: critical point (only a *candidate* extremum).
- $f'' > 0$: concave up; $f'' < 0$: concave down; sign change: inflection.
- First/Second Derivative Tests classify critical points — when $f''(c) = 0$, fall back on the First Derivative Test.
- Global extrema on $[a, b]$: evaluate at critical points *and* endpoints — endpoints can win even with zero critical points.

Coming next: HW-1

Apply derivatives to optimize skip-list performance, and practice finding and classifying extrema.

HW-1: Derivatives by Hand & Skip-List Optimization

EPITA — Calculus for Computer Scientists

Jim Newton

Coverage

Chapter 3: HW-1 — Derivatives by Hand and Skip List Optimization — full (the chapter *is* this homework).

HW-1 Objectives

- Compute derivatives by hand (polynomial, trig, exponential, logarithm).
- Verify with your `derivative` implementation.
- Apply derivatives to a CS problem: model two-level skip-list search time.
- Optimize $T(k) = k + n/k$ to find optimal index size $k = \sqrt{n}$.
- Generalize to multi-level skip list; show optimal promotion probability $p = 1/e$.
- Understand that changing the cost model changes the optimal design.

Part 1: Derivatives by Hand

Practice applying Chapter 1 rules to functions not yet encountered:

Polynomials and powers:

$$\mathcal{D}_x 5x^3 - 2x + 7, \quad \mathcal{D}_x \frac{1}{\sqrt{x}}$$

Trig:

$$\mathcal{D}_x \sin x \cos x, \quad \mathcal{D}_x \tan^2 x$$

Chain rule:

$$\mathcal{D}_x e^{x^2}, \quad \mathcal{D}_x \sin(3x + 1)$$

Exponential and log:

$$\mathcal{D}_x 3e^x + \ln x$$

Verification: evaluate both your hand answer and your `derivative` output numerically at several points. They should agree to machine precision.

Two-Level Skip List: The Model

A two-level linked list with n elements and k "express" nodes.

Expected search cost

$$T(k) = k + \frac{n}{k}$$

First term: steps along express lane. Second term: expected steps in the base list per express gap.

Question: what value of k minimizes $T(k)$?

Method:

1. Differentiate $T(k)$ with respect to k.
2. Set $T'(k) = 0$ and solve.
3. Verify it is a minimum with the Second Derivative Test.

Two-Level Optimization

$$T(k) = k + nk^{-1} \;\Rightarrow\; T'(k) = 1 - \frac{n}{k^2}$$

Setting $T'(k) = 0$:

$$1 - \frac{n}{k^2} = 0 \;\Rightarrow\; k^2 = n \;\Rightarrow\; \boxed{k = \sqrt{n}}$$

Second Derivative Test: $T''(k) = \dfrac{2n}{k^3} > 0$ for $k > 0$ — confirms minimum.

Interpretation

$\sqrt{n}$ express nodes, each covering $\sqrt{n}$ base nodes. Total search cost: $T(\sqrt{n}) = 2\sqrt{n}$ — square root improvement.

Multi-Level Skip List

With multiple levels and promotion probability p, the expected search cost is:

$$T(p) = \frac{1}{p} \log_{1/p}(n) = \frac{-\ln n}{p \ln p}$$

Goal: minimize over $p \in (0, 1)$.

Steps:

1. Use change-of-base to write T in terms of natural logarithms.
2. Show minimizing T is equivalent to maximizing $-p \ln p$.
3. Differentiate $-p \ln p$, set equal to zero.

Multi-Level Result: $p = 1/e$

Maximize $g(p) = -p \ln p$:

$$g'(p) = -(\ln p + 1) = 0 \;\Rightarrow\; \ln p = -1 \;\Rightarrow\; \boxed{p = e^{-1} = \frac{1}{e} \approx 0.368}$$

Interpretation

Each element is promoted to the next level with probability $1/e \approx 37\%$. This balances: fewer levels (coarser search) vs. more steps per level (denser search).

Lesson

Changing the cost model changes the optimal design. Mathematical models must reflect the actual implementation cost structure.

HW-1 Summary

- Derivative rules are not just abstract — they solve real CS optimization problems.
- Two-level list: optimal express-lane size is $k = \sqrt{n}$ (Power Rule + algebra).
- Multi-level list: optimal promotion probability is $p = 1/e$ (logarithm derivative).
- Numerical verification with `derivative.py` builds confidence in both the code and the math.
- The cost model shapes the optimal answer — always question your model.

Coming next: Chapter 4

Symbolic differentiation via pattern matching on expression trees.

CM-4: Symbolic Differentiation via Pattern Matching

EPITA — Calculus for Computer Scientists

Jim Newton

Coverage

This lecture covers:

- Chapter 5: Symbolic Differentiation via Pattern Matching — partial (through Composition; testing is covered in CM-5)

Chapter Objectives

- Explain why the base class is called `PointFree`, and describe the ADT it roots: leaf nodes (`Numeric`, `Symbol`, `Identity`, `Power`, `Sin`, `Cos`, `Exp`) and binary nodes (`Sum`, `Difference`, `Product`, `Quotient`, `Compose`).
- Construct and evaluate expression ASTs using Python class instances and `match`/`case` pattern matching.
- Implement `derivative` by mapping each differentiation rule from Chapter 1 onto exactly one `case` branch.

Testing the derivative *numerically* is covered in CM-5.

Why This Chapter Exists

When the shape of the code mirrors the shape of the mathematics, two things happen at once: the code becomes easier to read and easier to verify, and the mathematics becomes easier to understand — sometimes the clearest path to grasping a concept (a definition, technique, rule, etc.) is to implement it.

— from the handout, Chapter 5

Code Mirrors Mathematics

The key insight

When code structure mirrors mathematical structure, both become easier to read, verify, and trust.

Mathematics

$$\mathcal{D}_x(f+g) = \mathcal{D}_x f + \mathcal{D}_x g$$

$$\mathcal{D}_x(f \cdot g) = f\mathcal{D}_x g + g\mathcal{D}_x f$$

$$\mathcal{D}_x(f \circ g) = (\mathcal{D}_x f \circ g) \cdot \mathcal{D}_x g$$

Python (`case` branches)

```
case Sum(f, g):
    return Sum(f.derivative(),
               g.derivative())
case Product(f, g):
    ...
case Compose(f, g):
    ...
```

One rule = one case branch. Correct rule $\Leftrightarrow$ correct branch.

Pattern Matching — A First Look

Before designing the full ADT, watch `match`/`case` dispatch on a *partial* `evaluate` — just the leaves:

```
class Numeric:
    __match_args__ = ('c',)
    def __init__(self, c): self.c = c
class Sin:    pass
class Cos:    pass
class Identity: pass

def evaluate(expr, x):
    match expr:
        case Numeric(c):    return c
        case Identity():    return x
        case Sin():         return sin(x)
        case Cos():         return cos(x)
```

Key idea

Point-Free vs. Pointwise

Pointwise (names x)

$\mathcal{D}_x \sin x = \cos x$

Point-free (no x)

$\mathcal{D} \sin = \cos$

`PointFree` objects ARE functions, not expressions

`Sin()` is *the sine function* — not $\sin(x)$. No free variable x ever appears; the input is supplied later, via `evaluate`.

Two traps

- A bare x becomes `Identity()`, never nothing.
- $3x^2$ is a *product*; $\sin(x^2)$ is a *composition*.

From Pointwise to Python

Pointwise	**Point-free**	**Python**
$f(x) = x$	id	`Identity()`
$f(x) = 3x^2$	$3 \cdot (\cdot)^2$	`Product(Numeric(3), Power(2))`
$f(x) = \sin(x^2)$	$\sin \circ (\cdot)^2$	`Compose(Sin(), Power(2))`
$f(x) = \sin(x)^2$	$(\cdot)^2 \circ \sin$	`Compose(Power(2), Sin())`
$f(x) = e^{-x}$	$\exp \circ (-1 \cdot \mathrm{id})$	`Compose(Exp(), Product(Numeric(-1), Identity()))`

A longer worked table, with every entry verified against `.evaluate()`, is in the handout.

A Word of Caution: "ADT" is Overloaded

Two different meanings, same acronym

Outside this course sequence, ADT often means *Abstract Data Type* — a type specified by the operations it supports (push, pop, enqueue, . . .), representation hidden from the client.

In Advanced Data Structures (Ch. 5, Abstract Syntax Trees) and here, ADT means *Algebraic Data Type* — a closed, enumerable set of variants, pattern-matched *exhaustively*.

Abstract Data Type asks *what can I do with this value?*
Algebraic Data Type asks *what shapes can this value take?*

The ADT: Leaf Nodes

Class	Abstract base	Math
`Numeric(c)`	`Constant`	constant $c \in \mathbb{R}$
`Symbol(name)`	`Constant`	named constant (π, e, ...)
`Identity()`	`Unary`	$x \mapsto x$
`Power(n)`	`Unary`	$x \mapsto x^n$
`Sin()`, `Cos()`, `Exp()`	`Unary`	sin, cos, exp

`Constant` and `Unary` are *abstract* base classes — never instantiated directly, but let one arm (`case Constant(c):`, `case Unary():`) match every leaf of that kind at once.

The ADT: Binary Nodes

Class	Math	Abstract base
`Sum(f, g)`	$f + g$	`BinaryOp`
`Difference(f, g)`	$f - g$	`BinaryOp`
`Product(f, g)`	$f \cdot g$	`BinaryOp`
`Quotient(f, g)`	f/g	`BinaryOp`
`Compose(f, g)`	$f \circ g$	`BinaryOp`

Building $\sin(x^2) + e^x$

```
Sum(Compose(Sin(), Power(2)), Exp())
```

evaluate: Completing the Pattern

Provided for you — not an assignment.

```
def evaluate(expr, x, env=None):
    if env is None: env = {}
    match expr:
        case Identity():    return x
        case Numeric(c):    return c
        case Symbol(name):  return env[name]
        case Power(n):      return x ** n
        case Sin():         return sin(x)
        case Sum(f, g):
            return evaluate(f, x, env) + evaluate(g, x, env)
        case Product(f, g):
            return evaluate(f, x, env) * evaluate(g, x, env)
        case Compose(f, g):
            return evaluate(f, evaluate(g, x, env), env)
```

Leaf branches return directly; binary branches recur into both children.

`Compose` is Not Just Another Binary Node

`Compose` is *associative* and has a two-sided identity, `Identity()`: together, the set of `PointFree` objects forms a *monoid* under composition.

Monoid

A set M with $\triangle$: $M \times M \to M$ such that:

1. $\triangle$ is associative.
2. Some $e \in M$ satisfies $e \triangle a = a \triangle e = a$ for all a.

`PointFree` has (at least) three independent monoids

Additive: ($+$, `Numeric(0)`) Multiplicative: ($\times$, `Numeric(1)`) Compositional: ($\circ$, `Identity()`)

f^n: Two Very Different Trees

`f ** n` constructs `Compose(Power(n), f)` — **not** a product of `f` with itself.

```
def __pow__(self, n):
    return Compose(Power(n), self)
```

`Sin() ** 100`
depth 1, 3 nodes — regardless of n

`naive_pow(Sin(), 100)`
depth 100, 201 nodes

Differentiating makes the gap worse: 7 nodes vs. 10,497 nodes, for the *same* function.

derivative: The Task

Implement `derivative(node)`: same shape as `evaluate`, one `case` branch per differentiation rule — but each branch returns a new `PointFree` tree instead of a number.

$f(x) = \sin x + \cos x \cdot \sin x$

$$f'(x) = \cos x + (-\sin x) \cdot \sin x + \cos x \cdot \cos x$$

`derivative(Sum(Sin(), Product(Cos(), Sin())))` must return exactly this tree.

No test file is provided. Designing your own test suite *is* part of this assignment.

CM-4 Summary

- `PointFree`: point-free functions as class instances; `match`/`case` replaces per-class OOP dispatch.
- `Constant`, `Unary`, `BinaryOp` group the ADT's leaves and internal nodes for exhaustive pattern matching.
- `Compose` makes `PointFree` a monoid — one of *three* independent monoid structures on the same set.
- `f ** n` composes with `Power(n)`: $O(1)$ tree size and depth, not $O(n)$ — keeping differentiation compact too.

Assigned now: the Group Project

Implementing `derivative.py` for every node type above is the Oral Defense group project — logistics and testing strategy covered next, in CM-5.

Coming Next: CM-5

Testing the Derivative

Why structural AST equality fails for real-valued functions, and why numerical testing (`functions_agree`) works instead — plus the Oral Defense logistics and Intra assignments — Chapter 6.

CM-5: Testing the Derivative

EPITA — Calculus for Computer Scientists

Jim Newton

Coverage

This lecture covers:

- Chapter 6: Testing `derivative`, Debugging Tools, and the Group Project — **full**

Chapter Objectives

- Explain why structural AST equality is too strict a correctness test for `derivative`, and design a numerical testing strategy instead.
- Use `print_ast`, `display`/`toLaTeX`, and `simplify` to diagnose why a computed derivative disagrees with an expected one.
- Explain what `gen` generates, and the difference between syntactic and semantic uniformity.
- Implement `derivative` for every node type in the ADT, and design and defend a test suite for it, as your group project.
- Implement `pattern_match.py` and `checks.py` individually.

The Testing Problem

How can you know whether `derivative` is correct?

The AST it computes is not the AST you would write by hand

The same mathematical function can be reached by different chains of rule applications, producing different-looking (but equally correct) trees.

A tempting but doomed idea

In Advanced Data Structures, `equiv_exprs` decided Boolean expression equivalence by exhaustive search — Boolean variables take only two values, so this is feasible.

Real-valued functions have an infinite domain. Exhaustive search is not an option. How do we adapt the idea?

Resolution: Structural vs. Numerical Equality

Structural vs. numerical equality

- **Structural:** compare ASTs after `.simplify()`.
- **Numerical:** evaluate both sides at sample points, compare within a tolerance.

Numerical equality sidesteps the infinite-domain problem: it never needs to prove equivalence for *every* input, only agreement at finitely many sample points, within tolerance. **If a structural test fails, check numerically before assuming the derivative is wrong.**

Debugging Tools — All Black Boxes

- `print_ast(expr)` — indented text diagram of the tree.
- `expr.toLaTeX()` + `display(expr)` — renders to LaTeX in your browser.
- `expr.simplify()` — folds constant arithmetic, cancels terms like `f - f`, sorts commutative operands into a canonical order.

You are not expected to know how any of these work internally — only that they exist and are safe to call.

gen: Randomly Generating Expressions

Provided for you — not an assignment.

```
def gen(size):
    if size == 1:
        return random.choice(leaves)
    pivot = random.randint(1, size - 1)
    left  = gen(pivot)
    right = gen(size - pivot)
    return random.choice([lambda: Sum(left, right),
                          lambda: Product(left, right),
                          ...])()
```

Deliberately does not use `match`/`case`

gen *builds* a tree from an `int`; it does not tear an existing tree down. There is no shape to match against.

Oral Defense: `derivative.py`

Group of 2–3. Implement `derivative` for every node type in the ADT — assigned last lecture (CM-4).

No test file provided

Designing and implementing your own test suite is part of the assignment. A hidden test suite also runs on submission and contributes to your grade — only pass/fail is shown.

Live defense

Explain your implementation choices and walk through your test suite.

30% of the final grade.

Intra Assignments

Two assignments, auto-graded via Forge/Intra, tests provided (no surprises), 15% of the final grade:

- `pattern_match.py` — `count_nodes` / `max_depth` / `distribute` (rewrites expressions via the distributive law).
- `checks.py` — `inf_norm`, `distance`, `functions_agree` — building blocks for your own `derivative` tests.

Graded individually, but work through them with your group.

CM-5 Summary

- Numerical testing sidesteps the AST-equality problem that exhaustive search cannot solve for real-valued functions; `.simplify()` helps when structural comparison is still worth trying.
- `print_ast`, `display`/`toLaTeX`, and `simplify` are black-box debugging tools — use them, no need to know how they work internally.
- `gen` builds random trees from an `int`; syntactic uniformity (same shape) is not the same as semantic uniformity (same function).
- Oral Defense (`derivative.py`, 30%) and two Intra assignments (`pattern_match.py`, `checks.py`, 15%) round out Chapter 6.

Coming next

L2: CM-6 — Definition of the Derivative (ε-δ limits, first-principles proofs).
L3: skips straight to CM-7 — The Riemann Integral.

CM-6: Definition of the Derivative

EPITA — Calculus for Computer Scientists

Jim Newton

Coverage

This lecture covers:

- Chapter 7: Definition of the Derivative — full
- L2 only. L3 does not have a CM-6 session — it continues straight from CM-5 to CM-7 (The Riemann Integral), and Chapter 7 remains optional reading for that track.

Chapter Objectives

- Recall the ε-δ definition of limit and the limit laws.
- Apply the Limit Cancellation Recipe to resolve $\frac{0}{0}$ forms by factoring (algebraic) or by trigonometric identity (transcendental).
- Define the derivative rigorously as a limit of a difference quotient.
- Connect the limit definition to the reduction rules already studied.
- Identify non-differentiable functions: corners, cusps, discontinuities.
- Re-derive the Sum and Product Rules from the limit definition.
- Define e^x via its ODE and derive the addition formula.
- Derive the derivative of sin from first principles.

The Limit: Recall

ε-δ definition

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

$$\forall\, \varepsilon > 0,\ \exists\, \delta > 0 \text{ such that } 0 < |x - a| < \delta \implies |f(x) - L| < \varepsilon$$

Continuity

f is **continuous at** a if $\lim_{x \to a} f(x) = f(a)$.

Key limit laws: sums, products, quotients, and compositions of continuous functions are continuous (where defined).

The Limit Cancellation Recipe

Pattern: resolving $\frac{0}{0}$ indeterminate forms

1. **Identify** — direct substitution gives $\frac{0}{0}$: both numerator and denominator vanish.
2. **Transform** — expose the vanishing factor:
   algebraic: factor (Factor Theorem); *transcendental*: apply a trig/analytic identity.
3. **Cancel** — remove the common factor. Valid because the limit guarantees $x \neq a$ (or $h \neq 0$) *throughout*.
4. **Evaluate** — apply the Substitution Rule, or quote a known limit.

Running example: $\lim_{x \to 1} \dfrac{x^2 - 1}{x - 1} \xrightarrow{\text{factor}} \lim_{x \to 1} (x + 1) = 2$

Every first-principles derivative proof in this chapter applies this recipe.

The Derivative as a Limit

Formal definition

The **derivative** of f at a is

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$$

provided this limit exists. If it does, f is **differentiable at** a.

Connection to Chapter 1:
Every reduction rule we used is a theorem provable from this definition. We were computing correctly without knowing why.

Notation:

$$f'(a), \quad \mathcal{D}_x f(x)\big|_{x=a}, \quad \frac{df}{dx}(a)$$

All mean the same thing.

Non-Differentiable Functions

A function may be continuous but not differentiable.

Feature	**Example**	**Reason**
Corner	$f(x) = \|x\|$ at $x = 0$	Left and right limits of $\frac{f(h)-f(0)}{h}$ differ
Cusp	$f(x) = x^{2/3}$ at $x = 0$	Derivative $\to \infty$
Discontinuity	step function	Limit does not exist
Vertical tangent	$f(x) = x^{1/3}$ at $x = 0$	$f'(0) = \pm\infty$

Key fact

Differentiable $\Rightarrow$ continuous. But continuous $\not\Rightarrow$ differentiable.

Re-deriving the Sum Rule

Theorem

If f and g are differentiable at a, then $(f+g)'(a) = f'(a) + g'(a)$.

Proof:

$$\begin{aligned}(f+g)'(a) &= \lim_{h\to 0} \frac{(f+g)(a+h) - (f+g)(a)}{h} \\ &= \lim_{h\to 0} \frac{f(a+h) - f(a)}{h} + \lim_{h\to 0} \frac{g(a+h) - g(a)}{h} \\ &= f'(a) + g'(a) \qquad \square\end{aligned}$$

The split uses the limit law for sums. The same technique proves the Scalar Rule.

Re-deriving the Product Rule

Key trick: add and subtract $f(a+h)g(a)$.

$$\begin{aligned}
(fg)'(a) &= \lim_{h\to 0} \frac{f(a+h)g(a+h) - f(a)g(a)}{h} \\
&= \lim_{h\to 0} \frac{f(a+h)g(a+h) - \mathbf{f(a+h)g(a)} + \mathbf{f(a+h)g(a)} - f(a)g(a)}{h} \\
&= \lim_{h\to 0} f(a+h) \cdot \frac{g(a+h) - g(a)}{h} + \lim_{h\to 0} g(a) \cdot \frac{f(a+h) - f(a)}{h} \\
&= f(a)\, g'(a) + g(a)\, f'(a)
\end{aligned}$$

(Used: continuity of f so $\lim_{h\to 0} f(a+h) = f(a)$.)

Transcendental Derivatives

$\mathcal{D}_x \sin x = \cos x$ (limit proof)
Uses the limits:

$$\lim_{h\to 0} \frac{\sin h}{h} = 1, \quad \lim_{h\to 0} \frac{\cos h - 1}{h} = 0$$

and the angle-addition formula
$\sin(x + h) = \sin x \cos h + \cos x \sin h$.

$\mathcal{D}_x e^x = e^x$ (by definition)

exp is *defined* as the unique function satisfying

$$\exp' = \exp, \qquad \exp(0) = 1.$$

The derivative is immediate. As a consequence:

$$e^{x+y} = e^x \cdot e^y.$$

CM-6 Summary

- The derivative is *defined* as a limit of a difference quotient.
- Every Chapter 1 rule is a theorem provable from this definition.
- Differentiable $\Rightarrow$ continuous, but not conversely (corners, cusps, vertical tangents).
- Every $\frac{0}{0}$ difference quotient is resolved by the Cancellation Recipe: **transform** (factor or identity), **cancel** (limit $\Rightarrow h \neq 0$), **evaluate**.
- sin's derivative uses the transcendental path: angle-addition formula exposes $\lim \sin h/h = 1$ and $\lim(\cos h - 1)/h = 0$.
- e^x is *defined* by $\exp' = \exp$, $\exp(0) = 1$: its derivative is immediate, and $e^{x+y} = e^x e^y$ follows from uniqueness.

Coming next: CM-7 (both tracks)

The Riemann Integral: accumulation as a limit of sums.

CM-7: The Riemann Integral

EPITA — Calculus for Computer Scientists

Jim Newton

Coverage

This lecture covers:

- Chapter 6: The Riemann Integral — full

Chapter Objectives

- Approximate area using left, right, midpoint, trapezoidal, and Simpson's rule sums.
- Define $\int_a^b f(x)\,dx$ as the limit of a Riemann sum.
- Apply properties: zero-width, reversal, splitting, linearity.
- Compare convergence rates of the five methods.
- Implement the five rules in Python (`riemann_sum.py`) and test against `scipy.integrate.quad`.

The Big Idea: Area as a Limit of Sums

Riemann sum with n uniform subintervals, width $\Delta x = (b-a)/n$

$$S_n = \sum_{k=0}^{n-1} f(x_k^*)\,\Delta x$$

where $x_k^* \in [x_k, x_{k+1}]$ is a sample point in each subinterval.

The definite integral

$$\int_a^b f(x)\,dx = \lim_{n\to\infty} S_n$$

provided this limit exists and is the same for every choice of sample points.

The integral measures **signed area**: regions below the x-axis contribute negatively.

The Five Summation Rules

Rule	**Sample point x_k^***	**Error**
Left sum	x_k (left endpoint)	$O(1/n)$
Right sum	x_{k+1} (right endpoint)	$O(1/n)$
Midpoint	$(x_k + x_{k+1})/2$	$O(1/n^2)$
Trapezoid	average of $f(x_k)$ and $f(x_{k+1})$	$O(1/n^2)$
Simpson's	parabola through x_j, x_{j+1}, x_{j+2}	$O(1/n^4)$

Simpson's rule (over a pair of subintervals)

$$S_n = \frac{\Delta x}{3} \sum_{j=0,2,\ldots}^{n-2} \left[f(x_j) + 4f(x_{j+1}) + f(x_{j+2}) \right]$$

Requires n even — reuses existing partition points, no new samples. Error $O(1/n^4)$: halving n reduces error by $\mathbf{16}\times$.

Convergence: Why Higher Order Wins

For the integral $\int_0^1 \frac{1}{1+x^2}\,dx = \pi/4$:

Method	$n = 10$	$n = 100$	$n = 1000$	$n = 10000$
Left sum	10^{-2}	10^{-3}	10^{-4}	10^{-5}
Trapezoid	10^{-3}	10^{-5}	10^{-7}	10^{-9}
Simpson's	10^{-6}	10^{-10}	10^{-14}	≈ 0

Practical lesson

Use Simpson's rule when accuracy matters. `scipy.integrate.quad` goes further: adaptive Gauss-Kronrod achieves $\approx 10^{-12}$ with far fewer evaluations.

Properties of the Definite Integral

Zero-width

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

Reversal

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

Splitting

$$\int_a^b f\,dx = \int_a^c f\,dx + \int_c^b f\,dx$$

Linearity

$$\int_a^b (\alpha f + \beta g)\,dx = \alpha \int_a^b f\,dx + \beta \int_a^b g\,dx$$

Linearity is the key to computing integrals of sums term by term — the same structure as linearity of the derivative.

Intra: Implementing `riemann_sum.py`

Five functions to implement, each with signature:

```
method(f, left, right, n)
```

- `left_sum(f, a, b, n)`
- `right_sum(f, a, b, n)`
- `mid_sum(f, a, b, n)`
- `trapezoid_sum(f, a, b, n)`
- `simpsons_sum(f, a, b, n)`

Forbidden imports

`numpy` and `scipy` may NOT appear in any submitted file. The grader does not have them installed. Build everything from `math` and pure Python.

`scipy.integrate.quad` as Testing Oracle

During local development (not in submitted files!), use `quad` to verify your implementations:

Testing strategy

For several integrands and intervals, check:

$$\left|\texttt{simpsons\_sum}(f, a, b, n) - \texttt{quad}(f, a, b)[0]\right| < \varepsilon$$

At $n = 1000$, Simpson's rule should agree with `quad` to $\approx 10^{-14}$.

Warning

Never `import scipy` in `riemann_sum.py`. Use it only in separate test/scratch scripts.

Chapter 6 Summary

- The definite integral is the limit of Riemann sums — a precise formalization of "area under the curve."
- Five rules differ in accuracy: left/right $O(1/n)$, midpoint/trapezoid $O(1/n^2)$, Simpson's $O(1/n^4)$.
- Properties — reversal, splitting, linearity — are proved from the limit definition using the corresponding sum properties.
- `riemann_sum.py` (Intra) builds these from scratch; `scipy.integrate.quad` is the benchmark during testing.

Coming next: Chapter 7 (TP-1)

NumPy: inverse trig functions, numerical integration, discovering antiderivatives empirically.

TP-1: NumPy — Integrals & Inverse Trig

EPITA — Calculus for Computer Scientists

Jim Newton

Coverage

This lecture covers:

- Chapter 7: NumPy — Integrals & Inverse Trig — full

Chapter Objectives

- Identify domains and ranges of the six inverse trig functions; explain why restricted domains are necessary.
- Graph arctan and interpret its horizontal asymptotes as $x \to \pm\infty$.
- Verify $(\arctan x)' = \frac{1}{1+x^2}$ numerically using finite differences.
- Use NumPy to evaluate definite integrals involving inverse trig functions.
- Discover antiderivatives empirically by comparing `np.cumsum` against known formulas.

NumPy: The Key Tool

Not in the standard library

Install once: `pip install numpy`. Import as `import numpy as np`. **Forbidden in Oral Defense and Intra submissions.**

Core idea: vectorized operations.

Python list (slow):

```
ys = [x**2 for x in xs]
```

NumPy (fast):

```
xs = np.linspace(0, 2, 5)
ys = xs**2
```

`np.linspace(a, b, n)`: n evenly spaced points from a to b.
Vectorized math: `np.sin`, `np.cos`, `np.arctan`, `np.exp`, ...

Inverse Trig: Restricted Domains

Trig functions are periodic $\Rightarrow$ not one-to-one $\Rightarrow$ no true inverse.
Solution: restrict to a canonical interval.

Function	**Restricted domain**	**Range of inverse**
arcsin	$[-\pi/2,\, \pi/2]$	$[-1,\, 1]$
arccos	$[0,\, \pi]$	$[-1,\, 1]$
arctan	$(-\pi/2,\, \pi/2)$	$\mathbb{R}$

Common mistake

$\arcsin(\sin x) = x$ only if $x \in [-\pi/2, \pi/2]$. Outside this range, `np.arcsin(np.sin(3.0))` $\neq$ `3.0`. This is not a bug — it is the definition of arcsin.

Key Features of arctan

- **Domain:** all of $\mathbb{R}$ (no restriction needed).
- **Range:** $(-\pi/2, \pi/2)$.
- **Asymptotes:** $\lim_{x\to+\infty} \arctan x = \pi/2$, $\lim_{x\to-\infty} \arctan x = -\pi/2$.
- **Odd:** $\arctan(-x) = -\arctan(x)$.
- **Slope at origin:** $(\arctan)'(0) = 1$.
- **S-shaped curve** between two horizontal asymptotes.

Antiderivative

$$\int \frac{dx}{1+x^2} = \arctan x + C$$

Special case:

$$\int_0^1 \frac{dx}{1+x^2} = \frac{\pi}{4}$$

Verifying the Derivative Numerically

Central difference approximation:

$$(\arctan)'(x) \approx \frac{\arctan(x+h) - \arctan(x-h)}{2h}, \quad h = 10^{-5}$$

Compare against the exact formula $\frac{1}{1+x^2}$ at 500 grid points.

Result

Maximum error $\approx 10^{-10}$ — consistent with $O(h^2)$ accuracy of central difference.

Caveat: h too small is also bad

For very small h, $\arctan(x+h) - \arctan(x-h)$ suffers floating-point cancellation. There is an optimal $h \approx 10^{-5}$; below that, the error *increases*.

π from an Integral

`np.trapezoid` approximates $\int_a^b f(x)\,dx$ from a sampled grid.

Computing $\pi/4$ numerically

```
xs = np.linspace(0, 1, 10000)
approx = np.trapezoid(1.0 / (1.0 + xs**2), xs)
```

Result: $0.7853981628\ldots$ vs. $\pi/4 = 0.7853981634\ldots$ — error $< 6 \times 10^{-10}$.

π emerges from integrating a purely algebraic function — no circles, no geometry.

Discovering Antiderivatives with `np.cumsum`

FTC: $F(x) = \int_0^x f(t)\,dt$ is an antiderivative of f.

Strategy: compute F numerically, plot it, recognise the shape.

Left Riemann cumulative sum

`F = np.cumsum(f) * dx` (where `dx = xs[1] - xs[0]`)

For $f(x) = \frac{1}{1+x^2}$: The cumulative sum matches $\arctan(x)$ to within $O(\Delta x)$.

The empirical confirmation

If a candidate G satisfies $G' = f$ and $G(0) = 0$, then $G = F$ exactly — no algebraic derivation needed.

Chapter 7 Summary

- NumPy enables vectorized computation: evaluate, differentiate, and integrate a function across a grid with no explicit loop.
- Inverse trig functions require restricted domains — library functions make silent choices; always check the range.
- The derivative $(\arctan x)' = \frac{1}{1+x^2}$ is verified numerically via central differences; the convergence rate confirms $O(h^2)$.
- $\int_0^1 \frac{dx}{1+x^2} = \pi/4$ is computed to nine decimal places with 10 000 trapezoid points.
- `np.cumsum` discovers antiderivatives empirically: compute $F(x)$, plot it, match against known functions.

Coming next: Chapter 8

The Fundamental Theorem of Calculus: the connection between derivatives and integrals.

CM-8: The Fundamental Theorem of Calculus

EPITA — Calculus for Computer Scientists

Jim Newton

Coverage

This lecture covers:

- Chapter 8: The Fundamental Theorem of Calculus — full

Chapter Objectives

- Explain the FTC using the **tank-and-flow-meter analogy**.
- State and apply FTC Part 1: differentiation of an accumulation function.
- State and apply FTC Part 2: evaluation via antiderivatives.
- Compute definite integrals as linear combinations using the antiderivative table.
- Expand a function as a Taylor series and differentiate or integrate term by term.
- (*Enrichment*) Recognize the definite integral as an inner product on a function space.

Intuition: Tank and Flow Meter

Method 1 (tank): measure the water level directly.

$$\Delta V = V(b) - V(a)$$

Method 2 (flow meter): integrate the flow rate $r(t)$.

$$\Delta V = \int_a^b r(t)\,dt$$

Both methods give the same answer.

The FTC in one sentence

The integral of a rate of change equals the net change.

Flow rate $r(t) > 0$: water enters.
Flow rate $r(t) < 0$: water leaves.
Net change can be zero even if both happen.

FTC Part 1: Differentiation of an Accumulation Function

Theorem (FTC Part 1)

If f is continuous on $[a, b]$, define the accumulation function

$$F(x) = \int_a^x f(t)\, dt.$$

Then F is differentiable on (a, b) and $F'(x) = f(x)$.

Meaning: integration and differentiation are inverse operations. The accumulation function is an antiderivative of the integrand.

Example

$$\frac{d}{dx} \int_0^x \frac{1}{1+t^2}\, dt = \frac{1}{1+x^2}$$

FTC Part 2: Evaluation via Antiderivatives

Theorem (FTC Part 2)

If f is continuous on $[a, b]$ and F is any antiderivative of f (i.e., $F' = f$), then

$$\int_a^b f(x)\,dx = F(b) - F(a) = \Big[F(x)\Big]_a^b$$

This is the engine of integral calculus. To evaluate a definite integral:

1. Find any antiderivative F of f (use the table).
2. Compute $F(b) - F(a)$.

Example

$$\int_0^{\pi} \sin x\,dx = [-\cos x]_0^{\pi} = -\cos\pi + \cos 0 = 1 + 1 = 2$$

Worked Examples Using FTC Part 2

$$\int_0^2 (3x^2 + 1)\, dx = [x^3 + x]_0^2 = 10$$

$$\int_1^e \frac{1}{x}\, dx = [\ln x]_1^e = 1$$

$$\int_0^1 e^x\, dx = [e^x]_0^1 = e - 1$$

$$\int_0^1 \frac{1}{1 + x^2}\, dx = [\arctan x]_0^1 = \frac{\pi}{4}$$

$$\int_0^{1/2} \frac{1}{\sqrt{1 - x^2}}\, dx = [\arcsin x]_0^{1/2} = \frac{\pi}{6}$$

Linearity and the Antiderivative Table

The integral is a **linear operator**:

$$\int_a^b \big(\alpha f + \beta g\big)\, dx = \alpha \int_a^b f\, dx + \beta \int_a^b g\, dx$$

Use this with the derivative table read backwards:

$f(x)$	$\int f(x)\, dx$
$x^n \ (n \neq -1)$	$\dfrac{x^{n+1}}{n+1} + C$
e^x	$e^x + C$
$\cos x$	$\sin x + C$
$\dfrac{1}{1+x^2}$	$\arctan x + C$
$\dfrac{1}{x}$	$\ln \lvert x \rvert + C$

Taylor Series and Term-by-Term Integration

Key series

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}, \qquad \sin x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}, \qquad \cos x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}$$

Differentiate term by term:

$$\frac{d}{dx}\left(\sum_{n=0}^{\infty} \frac{x^n}{n!}\right) = \sum_{n=1}^{\infty} \frac{n x^{n-1}}{n!} = \sum_{n=0}^{\infty} \frac{x^n}{n!} \quad \Rightarrow \quad (e^x)' = e^x$$

Integrate term by term similarly — the constant of integration pins the value at $x = 0$.

(Enrichment) Inner Product on Function Spaces

The definite integral defines an inner product on $C[0,1]$:

$$\langle f, g \rangle = \int_0^1 f(x)\, g(x)\, dx, \qquad \|f\| = \sqrt{\langle f, f \rangle}$$

This satisfies the four inner product axioms (non-negativity, symmetry, linearity, definiteness).

Connection to Linear Algebra

The L^2 inner product is the infinite-dimensional analogue of the dot product. Orthogonality, projections, and Fourier series all arise from this structure. The Gram-Schmidt process extends to function spaces.

Chapter 8 Summary

- FTC Part 1: $\frac{d}{dx}\int_a^x f(t)\,dt = f(x)$ — differentiation undoes integration.
- FTC Part 2: $\int_a^b f\,dx = F(b) - F(a)$ — integration reduces to antiderivative evaluation.
- Linearity of the integral + antiderivative table = machinery for computing most standard integrals.
- Taylor series can be differentiated and integrated term by term.
- The definite integral defines an inner product on function spaces.

Coming next: HW-2 (Chapter 9)

Practice: inner products, antiderivatives, Taylor series, and definite integrals.

HW-2: Computing Integrals

EPITA — Calculus for Computer Scientists

Jim Newton

Coverage

Chapter 9: HW-2 — Computing Integrals — full (the chapter *is* this homework).

HW-2 Objectives

- Evaluate the inner product of two functions on an interval and verify orthogonality.
- Compute antiderivatives by table lookup (linearity + known forms).
- Differentiate and integrate Taylor series term by term.
- Evaluate definite integrals using FTC Part 2.
- Verify answers numerically using `simpsons_sum` from `riemann_sum.py`.

Exercise: Inner Product on Function Spaces

The inner product on $C[0,1]$ is:

$$\langle f, g \rangle = \int_0^1 f(x)\, g(x)\, dx$$

Axioms to verify

1. Non-negativity: $\langle f, f \rangle \geq 0$
2. Definiteness: $\langle f, f \rangle = 0 \Rightarrow f = 0$ (subtle!)
3. Symmetry: $\langle f, g \rangle = \langle g, f \rangle$
4. Linearity: $\langle \alpha f + \beta g, h \rangle = \alpha \langle f, h \rangle + \beta \langle g, h \rangle$

Enrichment: Axiom 2 is subtle

A function that is 1 at exactly one point and 0 elsewhere has $\langle f, f \rangle = 0$ but is not identically zero. This is why mathematicians use Lebesgue integration.

Exercise: Antiderivatives by Table Lookup

Use linearity + the antiderivative table. No technique required.

Integral	**Answer**
$\int (3x^2 + 2x - 7)\,dx$	$x^3 + x^2 - 7x + C$
$\int (5e^x - 4\cos x)\,dx$	$5e^x - 4\sin x + C$
$\int \dfrac{3}{1+x^2}\,dx$	$3\arctan x + C$
$\int \left(\dfrac{1}{x} + e^x\right)\,dx$	$\ln\lvert x\rvert + e^x + C$
$\int \left(2\sin x + \dfrac{5}{\sqrt{1-x^2}}\right)\,dx$	$-2\cos x + 5\arcsin x + C$

Exercise: Taylor Series and Differentiation

Differentiating $\sin x$ term by term

$$\frac{d}{dx}\left(x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots\right) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots = \cos x$$

Confirms $(\sin x)' = \cos x$ via series.

Integrating $\cos x$ term by term

$$\int\left(1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots\right) dx = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots = \sin x + C$$

Setting $C = 0$ uses $\sin 0 = 0$.

e^x differentiates to itself

$$\frac{d}{dx}\sum_{n=0}^{\infty}\frac{x^n}{n!} = \sum_{n=1}^{\infty}\frac{x^{n-1}}{(n-1)!} = \sum_{n=0}^{\infty}\frac{x^n}{n!}$$

Exercise: Evaluating Definite Integrals

Apply FTC Part 2: find F, compute $F(b) - F(a)$.

Integral	**Antiderivative F**	**Value**
$\int_0^2 (3x^2 + 1)\,dx$	$x^3 + x$	10
$\int_0^\pi \sin x\,dx$	$-\cos x$	2
$\int_1^e \frac{1}{x}\,dx$	$\ln x$	1
$\int_0^1 e^x\,dx$	e^x	$e - 1$
$\int_0^1 \frac{1}{1+x^2}\,dx$	$\arctan x$	$\pi/4$

Numerical Verification with `simpsons_sum`

Verify your exact answers using your own Intra submission:

Example: verify $\int_0^1 \frac{1}{1+x^2}\,dx = \pi/4$

```
from riemann_sum import simpsons_sum
import math

val = simpsons_sum(lambda x:  1/(1+x**2), 0, 1, 1000)
print(abs(val - math.pi/4))   # expect < 10^-12
```

- $n = 1000$ even (Simpson's requires even n).
- Error scales as $O(1/n^4)$: doubling n reduces error by $16\times$.
- No `scipy` or `numpy` needed — only your `riemann_sum.py`.

HW-2 Summary

- Inner products on function spaces work like dot products — but integration replaces summation.
- Antiderivatives come from reading the derivative table backwards, combined with linearity.
- Taylor series can be differentiated and integrated term by term — this is how the identity $(\sin)' = \cos$ is confirmed algebraically.
- FTC Part 2 reduces definite integrals to antiderivative evaluation at two points.
- Numerical verification with `simpsons_sum` should agree with exact answers to $< 10^{-10}$.

Coming next: Chapter 10 (TP-2)

SciPy: `quad`, `minimize_scalar`, `solve_ivp`.

TP-2: Calculus with SciPy

EPITA — Calculus for Computer Scientists

Jim Newton

Coverage

This lecture covers:

- Chapter 10: Calculus with SciPy — full

Chapter Objectives

- Use `scipy.integrate.quad` to evaluate definite integrals numerically, including those with no closed-form antiderivative.
- Use Python's `timeit` module to compare the speed of `simpsons_sum` and `quad`; explain why adaptive methods with compiled backends win.
- Use `scipy.optimize.minimize_scalar` to find minima and maxima numerically; compare with analytical solutions.
- Preview `scipy.integrate.solve_ivp` for solving initial-value problems and connect ODEs to the FTC.

SciPy: The Next Tier

The progression

Intra (`riemann_sum.py`): built from scratch, $O(1/n^4)$ at best.

NumPy (`np.trapezoid`): vectorized uniform-grid integration.

SciPy (`quad`): adaptive solver, $\approx 10^{-12}$ accuracy automatically.

Not in the standard library — and forbidden in submissions

`pip install scipy`. Import only specific functions:

```
from scipy.integrate import quad, solve_ivp
from scipy.optimize import minimize_scalar
```

Never `import scipy` in Oral Defense or Intra submissions. The grader will fail.

scipy.integrate.quad: Adaptive Integration

Signature and return value

```
val, err = quad(f, a, b)
```

`val`: the integral; `err`: estimated absolute error.

How it works: adaptive Gauss-Kronrod quadrature — concentrates sample points where f varies most rapidly. Not a uniform grid.

Familiar integral:

```
quad(lambda x:  1/(1+x**2), 0, 1)
```

$\rightarrow$ `(0.785398163397448, 8.7e-15)`

Agrees with $\pi/4$ to 15 decimal places.

Infinite limits:

```
quad(lambda x:  np.exp(-x**2),
     0, np.inf)
```

$\rightarrow \sqrt{\pi}/2 \approx 0.8862\ldots$

∞ passed as `np.inf`.

Integrals with No Closed Form

The Gaussian integral

$$\int_0^\infty e^{-x^2}\,dx = \frac{\sqrt{\pi}}{2}$$

Cannot be derived using techniques in this course. `simpsons_sum` cannot handle it (infinite upper limit). `quad` handles both limitations automatically.

Why it matters:
Every normal distribution calculation reduces to integrals of e^{-x^2}. Machine learning, statistics, and signal processing all depend on this number.

The Gaussian integral is not a curiosity — it is everywhere in applied science.

Speed: quad vs simpsons_sum

Two sources of the gap:

1. **Algorithm**: quad uses ≈ 21 evaluations (adaptive); simpsons_sum uses $n+1$ (uniform).
2. **Implementation**: quad calls compiled Fortran (QUADPACK); simpsons_sum runs a Python loop.

n	error	ms/call
1 000	1.3×10^{-13}	0.87
10 000	9×10^{-16}	8.7
100 000	9×10^{-16}	87
quad (21 evals)		0.04

Take-away

quad reaches machine precision in **0.04** ms; simpsons_sum needs 87 ms at $n = 100\,000$ for the same accuracy. Speedup: $\approx$ **2000**$\times$.

scipy.optimize.minimize_scalar

Signature

`result = minimize_scalar(f, bounds=(a, b), method='bounded')`

`result.x`: minimiser; `result.fun`: minimum value.

To maximise: minimise $-f$.

Local vs. global minimum

Finds a *local* minimum inside the given bounds. Always plot first to choose bounds containing exactly one minimum.

Recovering the skip-list optimum (HW-1)

```
T = lambda p:  math.log(n) / (-p * math.log(p))
result = minimize_scalar(T, bounds=(0.01, 0.99), method='bounded')
```

$\rightarrow$ `result.x` $= 0.36787944\ldots = 1/e$ ✓

scipy.integrate.solve_ivp: ODEs

Initial-value problem (IVP)

$$\frac{dy}{dt} = g(t, y), \qquad y(t_0) = y_0$$

Know the rate of change at every point $\Rightarrow$ determine the function.

Connection to FTC: when g doesn't depend on y, this reduces to plain integration.

Exponential growth: $y' = y$, $y(0) = 1 \Rightarrow y(t) = e^t$

```
sol = solve_ivp(lambda t, y:  y, (0, 3), [1.0], dense_output=True)
```

Max error vs. `np.exp(ts)`: $\approx 10^{-8}$ — adaptive RK45 solver.

Logistic Growth: When Rate Depends on State

$$\frac{dy}{dt} = r\,y\left(1 - \frac{y}{K}\right), \qquad y(0) = y_0$$

r: growth rate; K: carrying capacity.

- When $y \ll K$: behaves like exponential growth $y' \approx ry$.
- When $y \approx K$: growth slows; y levels off at K.
- When $y > K$: y decreases back toward K.

S-shaped solution

The solution curve has the same qualitative shape as arctan — both are bounded, monotone, and approach their asymptote exponentially fast.

No elementary closed form: `solve_ivp` is the right tool.

Chapter 10 Summary

- `quad`: near machine precision, handles infinite limits and integrands with no elementary antiderivative.
- `quad` vs `simpsons_sum`: $\approx 2000\times$ faster at the same accuracy — adaptive strategy (21 evals) + compiled Fortran.
- `minimize_scalar`: numerical critical-point finder; confirmed skip-list optimum $p = 1/e$ to 8 decimal places.
- `solve_ivp`: solves ODEs $y' = g(t, y)$; reduces to `quad` when g doesn't depend on y.
- The SciPy contract: you provide a Python callable and parameters; the library chooses the algorithm and estimates its own error.

Coming next: Chapter 11

Techniques of integration: u-substitution, partial fractions, integration by parts.

CM-9: Integration by Substitution

EPITA — Calculus for Computer Scientists

Jim Newton

Coverage

This lecture covers:

- Chapter 13: Integration by Substitution — full (u-substitution and the trigonometric substitution $x = \sin\theta$)

Chapter Objectives

- Explain why antidifferentiation has no systematic algorithm like the derivative rules.
- Apply **u-substitution** by reversing the chain rule.
- Evaluate $\int \sqrt{1-x^2}\,dx$ both geometrically and via the substitution $x = \sin\theta$, and explain why the two methods agree.

Why Integration is Hard

Differentiation: finite set of rules, always terminates, always gives an elementary result.

Integration: no such finite algorithm exists.

Example: why the chain rule doesn't reverse cleanly

$\frac{d}{dx}[\sin(x^2)\cos(x^2)] =?$ requires the product rule + chain rule.

But given the result, finding the original function by pattern-matching is not systematic.

Many elementary functions have no elementary antiderivative

e^{-x^2}, $\frac{\sin x}{x}$, $\frac{1}{\ln x}$ — these look simple but cannot be expressed in terms of standard functions. Use `quad`.

U-Substitution: Reversing the Chain Rule

The idea

If $\int f(g(x))g'(x)\,dx$ has the form "outer composed with inner, times inner's derivative," let $u = g(x)$, $du = g'(x)\,dx$:

$$\int f(g(x))\,g'(x)\,dx = \int f(u)\,du$$

Examples

$$\int 2x\cos(x^2)\,dx: \quad u = x^2,\ du = 2x\,dx \ \Rightarrow\ \int \cos u\,du = \sin u + C = \sin(x^2) + C$$

$$\int \frac{x}{1+x^2}\,dx: \quad u = 1 + x^2,\ du = 2x\,dx \ \Rightarrow\ \tfrac{1}{2}\int \frac{du}{u} = \tfrac{1}{2}\ln|u| + C$$

U-Substitution: How to Spot It

Look for a function and its derivative as a factor.

Integrand	**Let $u =$**	**Result**
e^{3x}	$u = 3x$	$\frac{1}{3}e^{3x} + C$
$\sin(ax + b)$	$u = ax + b$	$-\frac{1}{a}\cos(ax + b) + C$
$\frac{f'(x)}{f(x)}$	$u = f(x)$	$\ln\lvert f(x)\rvert + C$
$f'(x)\,[f(x)]^n$	$u = f(x)$	$\frac{[f(x)]^{n+1}}{n+1} + C$

For definite integrals: change limits too. If $u = g(x)$, when $x = a$, $u = g(a)$; when $x = b$, $u = g(b)$.

Two Integrals That Look Almost Identical

$$\int \frac{dx}{\sqrt{1-x^2}} \qquad \text{vs.} \qquad \int \sqrt{1-x^2}\,dx$$

The first: already in the table

Since $\frac{d}{dx}[\arcsin x] = \frac{1}{\sqrt{1-x^2}}$, reading the derivative table right-to-left gives

$$\int \frac{dx}{\sqrt{1-x^2}} = \arcsin x + C\,.$$

The second: no table entry, no u-substitution

There is no inner function whose derivative also appears as a factor — u-substitution does not help. We need a new tool.

The Substitution $x = \sin\theta$

Why $\sin\theta$?

$\sqrt{1-x^2}$ only makes sense (as a real number) for $-1 \le x \le 1$ — exactly the range of $\sin\theta$.

Setup

Let $x = \sin\theta$ with $\theta \in [-\pi/2, \pi/2]$, so $dx = \cos\theta\, d\theta$. On this range $\cos\theta \ge 0$, so

$$\sqrt{1-x^2} = \sqrt{1-\sin^2\theta} = \sqrt{\cos^2\theta} = \cos\theta\,,$$

with no absolute value or branch choice needed.

Why restrict to $[-\pi/2, \pi/2]$? Outside that range $\cos\theta$ can be negative, and $\sqrt{\cos^2\theta}$ would flip sign — the clean cancellation above needs this range.

Applying the Substitution to Both Integrals

$\int dx/\sqrt{1-x^2}$: $\cos\theta$ cancels

$$\int \frac{dx}{\sqrt{1-x^2}} = \int \frac{\cos\theta\, d\theta}{\cos\theta} = \int d\theta = \theta + C = \arcsin x + C$$

Matches the table-lookup answer exactly.

$\int \sqrt{1-x^2}\, dx$: $\cos\theta$ multiplies instead

$$\int \sqrt{1-x^2}\, dx = \int \cos^2\theta\, d\theta = \tfrac{1}{2}\big(\theta + \sin\theta\cos\theta\big) + C = \tfrac{1}{2}\big(\arcsin x + x\sqrt{1-x^2}\big) + C$$

(power-reduction identity $\cos^2\theta = \frac{1}{2}(1+\cos 2\theta)$)

Same substitution, one line of algebra apart — yet one collapses immediately and the other needs an extra identity. Whether $\cos\theta$ lands in the denominator or the numerator is the whole difference.

CM-9 Summary

- Integration has no finite algorithm — many elementary functions have no elementary antiderivative.
- **U-substitution**: spot $f(g(x)) \cdot g'(x)$; let $u = g(x)$.
- **Special substitution** $x = \sin\theta$ turns $\sqrt{1-x^2}$ into $\cos\theta$, unlocking both $\int dx/\sqrt{1-x^2}$ and $\int \sqrt{1-x^2}\,dx$.
- Same substitution, two different outcomes — whether $\cos\theta$ cancels or multiplies decides everything.

Coming next: CM-9 (Chapter 14: Standard Techniques)

Partial fractions: decomposing a rational function and integrating each term. (Enrichment) integration by parts.

HW-3: Variable Data Transfer Rate

EPITA — Calculus for Computer Scientists

Jim Newton

Coverage

Chapter 12: HW-3 — Variable Data Transfer Rate — full (the chapter *is* this homework).

HW-3 Objectives

- Evaluate integrals using u-substitution, selecting the right substitution.
- Integrate rational functions by partial fraction decomposition: simple and repeated linear factors.
- (*Enrichment*) Apply integration by parts to products.
- **Capstone:** model a multi-phase data transfer process by integrating a piecewise rate function and compute total data transferred.

Practice: U-Substitution

For each integral, identify u, compute du, and evaluate.

Integral	**Substitution**
$\int \cos(3x+1)\,dx$	$u = 3x+1$
$\int xe^{x^2}\,dx$	$u = x^2$
$\int \frac{\ln x}{x}\,dx$	$u = \ln x$
$\int \sin^3 x \cos x\,dx$	$u = \sin x$
$\int \frac{e^x}{1+e^{2x}}\,dx$	$u = e^x$

Practice: Partial Fractions

Decompose and integrate each rational function.

Integral	**Form**
$\int \frac{dx}{x^2-1}$	Simple linear: $\frac{A}{x-1} + \frac{B}{x+1}$
$\int \frac{3x+2}{x^2+3x+2}\,dx$	Factor denominator first
$\int \frac{dx}{x^2(x+1)}$	Repeated factor: $\frac{A}{x} + \frac{B}{x^2} + \frac{C}{x+1}$
$\int \frac{x}{(x-1)^2}\,dx$	Repeated linear
$\int \frac{2x+1}{x^2+1}\,dx$	Irreducible quadratic ($\ln + \arctan$)

Capstone: Variable Data Transfer Rate

Problem setup: A data link has a time-varying transfer rate $r(t)$ (Megabits/s). The total data transferred over $[a, b]$ is:

$$D = \int_a^b r(t)\,dt$$

Three consecutive phases, each exercising a different technique:

Phase	**Rate** $r(t)$	**Technique**
Slow Start	$\dfrac{t}{\sqrt{t^2+1}},\ t \in [0, \sqrt{3}]$	u-substitution
Dual Bottleneck	$\dfrac{6}{(t+1)(t+3)},\ t \in [0, 1]$	partial fractions
Traffic Burst	$\dfrac{2}{t^2+1},\ t \in [0, 1]$	arctan (irreducible quadratic)

Integrate each phase separately, then sum (Interval Splitting Proposition).

Solving the Multi-Phase Problem

Phase 1 (u-sub: $u = t^2 + 1$, $du = 2t\,dt$):

$$\int_0^{\sqrt{3}} \frac{t}{\sqrt{t^2+1}}\,dt = \Big[\sqrt{t^2+1}\Big]_0^{\sqrt{3}} = \sqrt{4} - \sqrt{1} = 1$$

Phase 2 (partial fractions: $\frac{6}{(t+1)(t+3)} = \frac{3}{t+1} - \frac{3}{t+3}$):

$$\int_0^1 \frac{6}{(t+1)(t+3)}\,dt = \Big[3\ln(t+1) - 3\ln(t+3)\Big]_0^1 = 3\ln\tfrac{3}{2}$$

Phase 3 (irreducible quadratic $\to$ arctan):

$$\int_0^1 \frac{2}{t^2+1}\,dt = \Big[2\arctan t\Big]_0^1 = \frac{\pi}{2}$$

Total: $D = 1 + 3\ln\frac{3}{2} + \frac{\pi}{2} \approx 3.787$ Megabits.

Verify with `scipy.integrate.quad`

Once you have the exact answer, verify numerically:

Example: Phase 3

```
from scipy.integrate import quad

r3 = lambda t:  2 / (t**2 + 1)
val, err = quad(r3, 0, 1)
print(f"numerical:  {val:.10f}")
print(f"exact:  {np.pi/2:.10f}")
```

Agreement to 10+ decimal places confirms both the integral and your formula. Repeat for Phases 1 and 2, then sum all three for the total Megabits transferred.

HW-3 Summary

- U-substitution: identify the inner function and its derivative as a factor.
- Partial fractions: factor the denominator, determine constants by substitution.
- Piecewise integration: split at breakpoints, integrate each phase separately.
- The data transfer capstone shows how multiple techniques combine on a single real CS problem.
- Verify with `scipy.integrate.quad` — agreement to 10 places confirms both your formula and your arithmetic.

Coming next: Chapter 13

Debriefing: could you implement `integrate()`? The Risch algorithm and why `scipy.integrate.quad` is the practical answer.

CM-10: Standard Techniques

EPITA — Calculus for Computer Scientists

Jim Newton

Coverage

This lecture covers:

- Chapter 14: Standard Techniques — full (partial fractions, and enrichment integration by parts)

Chapter Objectives

- Decompose a rational function into **partial fractions** and integrate each term.
- Recognize and apply the three standard integral forms: $\ln|x+b|$, $\ln|x^2+b|$, and arctan.
- (*Enrichment*) Apply **integration by parts** to products of functions.

CM-8 covered u-substitution and the trigonometric substitution $x = \sin\theta$. Despite the name, partial fractions is one of the most commonly applied integration techniques in practice — not a rare trick.

Partial Fractions: The Setup

Goal: integrate a rational function $\dfrac{P(x)}{Q(x)}$ where $\deg P < \deg Q$.

Idea: decompose into simpler fractions with known antiderivatives.

Example: $\dfrac{1}{(x-1)(x+2)}$

$$\frac{1}{(x-1)(x+2)} = \frac{A}{x-1} + \frac{B}{x+2}$$

Multiply both sides by $(x-1)(x+2)$: $1 = A(x+2) + B(x-1)$.
Set $x = 1$: $A = 1/3$. Set $x = -2$: $B = -1/3$.

$$\int \frac{dx}{(x-1)(x+2)} = \tfrac{1}{3}\ln|x-1| - \tfrac{1}{3}\ln|x+2| + C$$

Three Standard Integral Forms

Every partial fraction decomposition reduces to these three:

Form 1: linear denominator

$$\int \frac{dx}{x+b} = \ln|x+b| + K$$

Form 2: quadratic denominator, numerator x

$$\int \frac{x\,dx}{x^2+b} = \tfrac{1}{2}\ln|x^2+b| + K$$

Form 3: sum of squares

$$\int \frac{dx}{x^2+b^2} = \frac{1}{b}\arctan\frac{x}{b} + K$$

Partial Fractions: Repeated and Quadratic Factors

Repeated linear factor $(x-a)^2$

$$\frac{P(x)}{(x-a)^2} = \frac{A}{x-a} + \frac{B}{(x-a)^2}$$

Irreducible quadratic x^2+bx+c (enrichment)

Complete the square: $x^2+bx+c = (x+b/2)^2 + (c-b^2/4)$.
Result involves arctan (Form 3).

Degree condition

If $\deg P \geq \deg Q$, perform polynomial long division first. The remainder has $\deg <$ $\deg Q$ — then apply partial fractions.

(Enrichment) Integration by Parts

The product rule $\frac{d}{dx}(uv) = u\frac{dv}{dx} + v\frac{du}{dx}$ rearranged:

$$\int u\,dv = uv - \int v\,du$$

Use when: integrand is a product of two unlike functions.

Examples

$$\int xe^x\,dx: \quad u = x,\ dv = e^x\,dx \ \Rightarrow\ xe^x - e^x + C$$

$$\int \ln x\,dx: \quad u = \ln x,\ dv = dx \ \Rightarrow\ x\ln x - x + C$$

LIATE rule of thumb

Choose u in order: **L**og, **I**nverse trig, **A**lgebraic, **T**rig, **E**xponential.

Chapter 14 Summary

- **Partial fractions**: decompose rational functions into $\ln|x+b|$, $\ln|x^2+b|$, and arctan terms.
- Handles simple linear, repeated linear, and (enrichment) irreducible quadratic denominators.
- (Enrichment) **Integration by parts**: $\int u\,dv = uv - \int v\,du$, guided by LIATE.
- When all else fails: `scipy.integrate.quad`.

Coming next: HW-3 (Chapter 15)

Apply u-substitution, partial fractions, and integration by parts to a multi-phase data transfer rate problem.

CM-11: Could You Implement `integrate()`?

EPITA — Calculus for Computer Scientists

Jim Newton

Coverage

This lecture covers:

- Chapter 13: Debriefing — Could You Implement `integrate()`? — full (first hour)
- Exam review — no handout chapter (second hour)

Chapter Objectives

- Explain why symbolic integration has no finite algorithm analogous to the derivative rules.
- Identify which integration problems are tractable by pattern matching and which are not.
- Describe at a high level how the Risch algorithm decides whether an elementary antiderivative exists.
- Articulate why `scipy.integrate.quad` is the practical tool of choice for definite integrals.
- *(Enrichment)* Identify the sets $\mathcal{A}$, $\mathcal{E}$, C^1 and the morphisms between them; explain why `.derivative()` is total on $\mathcal{E}$ while the antiderivative is only partial.

A Thought Exercise

You implemented `derivative()` on the `PointFree` AST.

Given an expression tree, it returns the derivative tree — mechanically, correctly, every time.

The question

Could you implement `integrate()` on `PointFree`?
How would you approach it? Which parts are tractable? Which are not?

derivative() — about 30 lines
Finite rules, always terminates,
always returns an elementary result.

integrate() — ???
Can the same approach work?
What would a `match`/`case` block look like?

What Would Be Tractable

Some cases mirror the derivative table directly:

Expression	**Antiderivative**
`Numeric(c)`	`Product(Numeric(c), Identity())`
`Power(n)`	`Quotient(Power(n+1), Numeric(n+1))`, $n \neq -1$
`Sin()`	`Product(Numeric(-1), Cos())`
`Cos()`	`Sin()`
`Exp()`	`Exp()`
`Sum(f, g)`	`Sum(integrate(f), integrate(g))`
`Difference(f, g)`	`Difference(integrate(f), integrate(g))`

Linearity works just as for differentiation: $\int(\alpha f \pm \beta g) = \alpha \int f \pm \beta \int g$.

Where It Breaks Down

Products of two non-constant functions

No general rule. Integration by parts sometimes helps:
$\int fg\,dx = f\int g\,dx - \int f'\int g\,dx\,dx$

But this only simplifies if the remaining integral is easier — which depends on the specific f and g in a way that is hard to decide algorithmically. **There is no product rule for integration.**

Composition: `Compose(f, g)`

u-substitution works *if* the integrand contains $g(x)$ and $g'(x)$ as a factor. Detecting this in an arbitrary AST requires comparing subtrees well beyond simple pattern matching.

Quotients: partial fractions

Only works for rational functions with deg(num) $<$ deg(denom). Requires polynomial factorisation — an algorithm in its own right.

The Deeper Obstruction

Some elementary functions have no elementary antiderivative

$$\int e^{-x^2}\,dx \qquad \int \frac{\sin x}{x}\,dx \qquad \int \frac{1}{\ln x}\,dx$$

These antiderivatives *exist* as real-valued functions but cannot be expressed as any finite combination of polynomials, exponentials, logarithms, and trigonometric functions.

- The `PointFree` ADT cannot represent these antiderivatives.
- No `integrate()` working within the ADT could return a correct answer.
- This is not a limitation of the implementation — it is a mathematical fact.

The Risch Algorithm

Computer algebra systems (SymPy, Mathematica, Maple) do implement symbolic integration.

The theoretical foundation: Risch algorithm (1969)

For a large class of elementary functions, the algorithm:

1. **Decides** whether an elementary antiderivative exists.
2. **Computes** it if one does.

It is correct and complete for that class.

But the complexity is enormous

The full Risch implementation runs to **thousands of lines** of carefully structured case analysis, requiring a deep theory of differential algebra.

Compare: `derivative()` is about **30 lines**.

Calculus via Large Language Models?

LLMs (ChatGPT, Claude) handle symbolic integration surprisingly well. Ask for $\int x^2 e^x \, dx$ and you usually get a correct, worked solution.

Why it works (often):
Training data contains thousands of worked integration examples. The model completes a pattern — it is not executing an algorithm.

Common integrals: very reliable.
Unusual integrals: reliability drops.

No certificate of correctness

The Risch algorithm either gives a provably correct answer or declares none exists. An LLM gives a confident-looking answer with no such guarantee.

Verification: differentiate the LLM's answer $F(x)$ and check $F'(x) = f(x)$ numerically at several sample points using `evaluate`.

Why `scipy.integrate.quad` is the Practical Answer

The asymmetry

Symbolic integration: powerful when it works, but *does not always work*.

Numerical integration (`quad`): works for *any continuous function*, regardless of whether a closed-form antiderivative exists.

- `quad` uses adaptive Gauss-Kronrod quadrature — not Riemann sums.
- Accuracy $\approx 10^{-12}$ automatically, with error estimate returned.
- Handles infinite limits: pass `np.inf`.
- The existence of `quad` is not a convenience shortcut — for many functions that arise in practice, it is the **only tool available**.

Pattern

Exact answer exists? Compute it with FTC, then verify with `quad`.
No exact answer? Use `quad` directly.

Set	What it contains	Size
$\mathcal{A}$	All ASTs from the `PointFree` grammar	Countably ∞, decidable
$\mathcal{E}$	Functions expressible by some AST	Countably ∞
$C^1(\mathbb{R})$	All differentiable functions	Uncountably ∞

$$\mathcal{E} \subsetneq C^1(\mathbb{R}) \subsetneq \{\text{all functions } \mathbb{R} \to \mathbb{R}\}$$

Why $\mathcal{A}$ is decidable

The grammar is context-free; you can enumerate all ASTs by depth and decide membership in finite time. Countability follows because each AST is a finite tree over a finite alphabet.

The Evaluation Map *(enrichment)*

$[\![\cdot]\!] : \mathcal{A} \twoheadrightarrow \mathcal{E}$ maps each AST to the function it computes (written eval in our handout; the standard PL notation is the *semantic bracket* $[\![\cdot]\!]$).

Properties of eval

- **Surjective** by definition of $\mathcal{E}$
- **Not injective**: `Add(Id,Id)` and `Prod(2,Id)` both compute $2x$
- **Ring homomorphism**: $[\![s +_{\mathcal{A}} t]\!] = [\![s]\!] + [\![t]\!]$

First Isomorphism Theorem

Let $s \sim t$ mean "s and t compute the same function."

$$\mathcal{A}/\!\sim \;\cong\; \mathcal{E}$$

The quotient is the ring of expressions modulo algebraic identity.

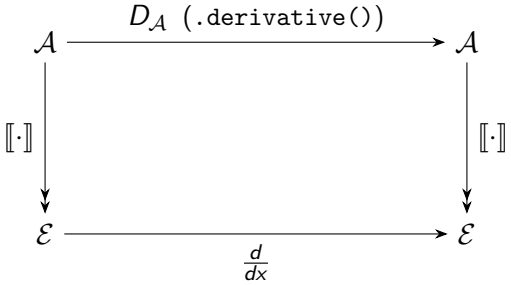
Differential rings

$(\mathcal{E}, +, \cdot, \frac{d}{dx})$ and $(C^1, +, \cdot, \frac{d}{dx})$ are *differential rings* — rings equipped with a derivation satisfying the Leibniz rule. $\mathcal{E}$ is a differential subring of C^1.

The Commutative Diagram *(enrichment)*

`.derivative()` and $\frac{d}{dx}$ are related by:

$$[\![D_{\mathcal{A}}(t)]\!] \;=\; \tfrac{d}{dx}([\![t]\!]) \quad \forall\, t \in \mathcal{A}$$



In algebraic language: $[\![\cdot]\!]$ is a surjective *differential ring homomorphism* from $(\mathcal{A}, D_{\mathcal{A}})$ to $(\mathcal{E}, \frac{d}{dx})$.

The soundness claim in one diagram: differentiating the AST and then evaluating gives the same result as evaluating and then differentiating.

The Partition and the Fundamental Asymmetry *(enrichment)*

$\ker(\frac{d}{dx}) = \mathbb{R}$ (constant functions), so $\frac{d}{dx}$ partitions $\mathcal{E}$ into antiderivative families $[f] = \{f + c \mid c \in \mathbb{R}\}$:

$$\mathcal{E}/\mathbb{R} \xrightarrow{\sim} \tfrac{d}{dx}(\mathcal{E}) \subsetneq \mathcal{E}$$

The strict inclusion is Liouville's theorem: $e^{x^2} \in \mathcal{E}$ but $e^{x^2} \notin \tfrac{d}{dx}(\mathcal{E})$.

Caveat: on a domain with n connected components, $\ker(\frac{d}{dx}) \cong \mathbb{R}^n$ (locally constant functions) — n independent constants, not one. Example: $\int \frac{1}{x}\,dx$ on $\mathbb{R} \setminus \{0\}$ needs independent C_1 (for $x < 0$) and C_2 (for $x > 0$).

Operator	**Map**	**Total?**
$D_{\mathcal{A}}$ (`.derivative()`)	$\mathcal{A} \to \mathcal{A}$	**Yes**
$\frac{d}{dx}$	$\mathcal{E} \to \mathcal{E}$	**Yes**
$\int$	$\mathcal{E} \to \mathcal{E}$	**No** — escapes $\mathcal{E}$

The derivative rules *close* under the grammar; the antiderivative rules do not. The Risch algorithm is precisely the decision procedure for whether $f \in \frac{d}{dx}(\mathcal{E})$.

Chapter 13 Summary

- `derivative()` is implementable with ≈ 30 lines because differentiation has a finite, complete set of rules.
- `integrate()` is fundamentally harder: products, compositions, and quotients resist mechanical treatment.
- Some elementary functions have *no* elementary antiderivative — the limitation is mathematical, not implementational.
- The Risch algorithm solves symbolic integration in theory; it requires thousands of lines and deep algebra theory.
- LLMs match patterns from training data — fast and often correct, but without a correctness guarantee. Always verify by differentiating.
- `scipy.integrate.quad` is the practical choice: works universally, near machine precision, with automatic error estimation.
- *(Enrichment)* $\mathcal{A}$, $\mathcal{E}$, C^1: three worlds linked by $[\![\cdot]\!]$; `.derivative()` and $\frac{d}{dx}$ satisfy $[\![D_{\mathcal{A}}(t)]\!] = \frac{d}{dx}([\![t]\!])$. Integration is partial because $\frac{d}{dx}(\mathcal{E}) \subsetneq \mathcal{E}$.