Comparing Coherent Differentiation (CD) and AD

Workshop on Differentiable Programming — 29, 30 June 2022

Thomas Ehrhard
IRIF, CNRS and Université Paris Cité

Derivatives

If E, F are, say, Banach spaces, $U \subseteq E$ open and $f : U \to F$, a derivative of f is a function

$$f' : U \to \mathcal{L}(E, F)$$

such that for any $x \in U$ there is $V_x \subseteq E$ open and $h_x : V_x \to F$ such that $0 \in V_x$, $x + V_x \subseteq U$ and,

$$\forall u \in V_x \quad f(x+u) = f(x) + f'(x) \cdot u + \|u\| h_x(u)$$

where

$$\|h_x(u)\| \underset{\to 0}{\longrightarrow} 0\,.$$

If it exists, f' is unique.

The chain rule

We can consider f' as a function

$$U \times E \to F$$

and then we can define the "tangent function"

$$\begin{aligned} \mathsf{T}f : U \times E &\to F \times F \\ (x, u) &\mapsto (f(x), f'(x) \cdot u) \end{aligned}$$

Fact (Chain rule)

If the (open) domain of g contains $f(U)$ then

$$\mathsf{T}(g \circ f) = \mathsf{T}g \circ \mathsf{T}f \,.$$

Generalizes quite well to manifolds: *Tangent Categories*.

Gradient

If E and F are finite dimensional, we can assume that they are Euclidian spaces, that is they are given together with a scalar product (positive-definite inner product)

$$\langle \_ \mid \_ \rangle : E \times E \to \mathbb{R}$$

which induces a canonical isomorphism

$$\eta_E : E \to E^* \quad \eta(x)(y) = \langle x \mid y \rangle$$

Then $\|x\| = \sqrt{\langle x \mid x \rangle}$.

Transpose

If $t \in \mathcal{L}(E, F)$ then $t^* \in \mathcal{L}(F^*, E^*)$ and hence

$$t^\top = {\eta_E}^{-1}\, t^*\, \eta_F \in \mathcal{L}(F, E)$$

$$F \xrightarrow{\eta_F} F^* \xrightarrow{t^*} E^* \xrightarrow{{\eta_E}^{-1}} E$$

characterized by

$$\forall x \in E\, \forall y \in F \quad \langle t \cdot x \mid y \rangle = \langle x \mid t^\top \cdot y \rangle\,.$$

So if $f : U \to F$ where $U \subseteq E$ open has a derivative $f' : U \to \mathcal{L}(E, F)$, we can define

$$f'(x)^\top \in \mathcal{L}(F, E)$$

When $F = \mathbb{R}$ we can define

$$\nabla f(x) = f'(x)^\top \cdot 1 \in E$$

the gradient vector field on U, characterized locally by

$$f(x + u) = f(x) + \langle u \mid \nabla f(x) \rangle + o(\|u\|)$$

Fact

If $y \in E$ with $\|y\| = 1$ then

$$\arg\max_{\|z\|=1} \langle z \mid y \rangle = y$$

So if we look for a small u in E (say $\|u\| = \varepsilon$) such as $f(x+u)$ is as large as possible, it is a good idea to take

$$u = \varepsilon \frac{\nabla f(x)}{\|\nabla f(x)\|} .$$

In concrete AD applications in AI:

- f is described as a program
- the dimension d of E is *very large* (typically several billions), say $E = \mathbb{R}^d$ with its canonical inner product
- $F = \mathbb{R}$.

And one wants to compute $\nabla f(x)$ as efficiently as possible, which is the same thing as $f'(x) \in \mathcal{L}(E, \mathbb{R})$ up to the η_E iso:

$$f'(x) \cdot u = \sum_{i=1}^{d} u_i \frac{\partial f(x)}{\partial x_i} = \langle u \mid \nabla f(x) \rangle$$

$$\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_i} \right)_{i=1}^{d}$$

A simple implementation

A simplified AD in the typed λ-calculus, in Brunel-Mazza-Pagani approach (in forward style): Λ_{AD}.

Data types:

$$A, B, \cdots := \mathsf{R} \mid A \times B \mid A \Rightarrow B$$

Notation: $A^n = \overbrace{A \times \cdots \times A}^{n}$

- Typing context: $\Gamma = (x_1 : A_1, \ldots, x_n : A_n)$
- Typing judgment: $\Gamma \vdash t : A$

for terms t that we describe now together with typing rules.

Basic ingredient: sets $(\mathbf{Sig}_e^d)_{d \in \mathbb{N}, e \in \mathbb{N} \setminus \{0\}}$ of function symbols, and for each $f \in \mathbf{Sig}_e^d$, its interpretation as a function $\overline{f} : \mathbb{R}^d \to \mathbb{R}^e$.

Terms and typing

$$\frac{i \in \{1, \ldots, n\}}{(x_j : A_j)_{j=1}^n \vdash x_i : A_i} \qquad \frac{f \in \mathbf{Sig}_e^d}{\Gamma \vdash f : \mathrm{R}^d \Rightarrow \mathrm{R}^e}$$

$$\frac{\Gamma \vdash t_1 : A_1 \qquad \Gamma \vdash t_2 : A_2}{\Gamma \vdash \langle t_1, t_2 \rangle : A_1 \times A_2} \qquad \frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash \mathrm{pr}_i\, t : A_i}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A \cdot t : A \Rightarrow B} \qquad \frac{\Gamma \vdash s : A \Rightarrow B \qquad \Gamma \vdash t : A}{\Gamma \vdash (s)t : B}$$

$$\frac{\Gamma \vdash t : A \Rightarrow A}{\Gamma \vdash \mathrm{fix}(t) : A}$$

The signature

For instance we can take

- $\mathbf{Sig}_1^0$ the set of all $\underline{r}$ for $r \in \mathbb{R}$, with $\overline{\underline{r}} = r$ etc.
- ifp $\in \mathbf{Sig}_1^3$,

$$\overline{\mathrm{ifp}}(r, r_1, r_2) = \begin{cases} r_1 & \text{if } r > 0 \\ r_2 & \text{otherwise} \end{cases}$$

- softmax $\in \mathbf{Sig}_k^k$
- relu, tanh, $\cdots \in \mathbf{Sig}_1^1$
- etc.

Operational semantics

Transformation rules for these expressions:

- λ-calculus rules:

$$(\lambda x : A \cdot s)t \to s\,[t/x]$$
$$\mathrm{pr}_i\langle t_1, t_2\rangle \to t_i$$
$$\mathrm{fix}(t) \to (t)\mathrm{fix}(t)$$

- The so called "δ-rules": if $f \in \mathbf{Sig}_e^d$ and $r_1, \ldots, r_d \in \mathbb{R}$ then

$$\underline{f}(\underline{r_1}, \ldots, \underline{r_d}) \to \underline{f(r_1, \ldots, r_d)}$$

  for instance $\underline{\cos}(\underline{\pi}) \to \underline{-1}$.

Specific evaluation strategies can be implemented by means of *abstract machines*.

Formal differentiation

If $\vdash t : \mathsf{R}^d \Rightarrow \mathsf{R}^e$ we want a differential

$$\vdash t' : \mathsf{R}^d \times \mathsf{R}^d \Rightarrow \mathsf{R}^e$$

It will be much more convenient to have a chain-rule compatible operation

$$\vdash \mathcal{T}t : \mathsf{R}^d \times \mathsf{R}^d \Rightarrow \mathsf{R}^e \times \mathsf{R}^e$$

BMP homomorphic differentiation

It is a syntactic transformation $\mathcal{T}$ from the language to itself.

Basic assumption

For each symbol $f \in \mathbf{Sig}_e^d$ with $d, e > 0$ there is a symbol $f' \in \mathbf{Sig}_e^{d+d}$.

Then we define

- Type transformation: $\mathcal{T}(\mathrm{R}^d) = \mathrm{R}^{2d}$ and $\mathcal{T}(A \Rightarrow B) = (\mathcal{T}A \Rightarrow \mathcal{T}B)$.
- $\mathcal{T}(A \times B) = \mathcal{T}A \times \mathcal{T}B$
- A term transformation such that

$$(x_1 : A_1, \ldots, x_n : A_n) \vdash t : B$$
$$\Rightarrow (x_1 : \mathcal{T}A_1, \ldots, x_n : \mathcal{T}A_n) \vdash \mathcal{T}t : \mathcal{T}B$$

The definition is straightforward:

- $\mathcal{T}x = x$
- for $c \in \mathbf{Sig}_1^0$, $\mathcal{T}(c) = \langle c, \underline{0} \rangle$ Reals are at the same time the values on which we compute and the coefficients of the matrices.
- for $f \in \mathbf{Sig}_e^d$ with $d, e > 0$,
  $\mathcal{T}f = \lambda x : \mathrm{R}^d \times \mathrm{R}^d \cdot \langle f(\mathrm{pr}_1 x), f'(x) \rangle$
- $\mathcal{T}\langle t_1, t_2 \rangle = \langle \mathcal{T}t_1, \mathcal{T}t_2 \rangle$
- $\mathcal{T}(\mathrm{pr}_i t) = \mathrm{pr}_i(\mathcal{T}t)$
- $\mathcal{T}(\lambda x : A \cdot t) = \lambda x : \mathcal{T}A \cdot \mathcal{T}t$
- $\mathcal{T}(s)t = (\mathcal{T}s)\mathcal{T}t$.

Fact (easy)

If $s \to t$ then $\mathcal{T}s \to^* \mathcal{T}t$

Denotation

In a recent paper by Mazza and Pagani it is strongly suggested that this language can be interpreted in a cartesian closed category $\mathcal{C}$. This category contains $\mathbb{R}$ as an object.

Morphisms should be partially defined functions which are differentiable "almost" everywhere.

- With any type A we associate an object $[\![A]\!]$ of $\mathcal{C}$
- and if $(x_i : A_i)_{i=1}^n \vdash t : B$ then $[\![t]\!] \in \mathcal{C}([\![A_1]\!] \times \cdots \times [\![A_n]\!], [\![B]\!])$

Main features

- If $s \to t$ then $[\![s]\!] = [\![t]\!]$
- If $x : \mathsf{R}^d \vdash t : \mathsf{R}^e$ so that $[\![t]\!] \in \mathcal{C}(\mathbb{R}^d, \mathbb{R}^e)$ and $[\![\mathcal{T}\, t]\!] \in \mathcal{C}(\mathbb{R}^{2d}, \mathbb{R}^{2e})$; then

$$[\![\mathcal{T}\, t]\!] = \mathsf{T}[\![t]\!]$$

Consequence: the $\mathcal{T}$ syntactic construct computes *almost everywhere* the "true" differential (gradient) of t.

Actually they manage to prove this without building the model.

Linearity in AD

Differentiation is linearization: if $f : E \to F$ then

$$\mathsf{T}f : \mathsf{T}E = E \times E \to \mathsf{T}F = F \times F$$

the second component is linear.

But in AD this is true only at ground type: we have

$$[\![\mathcal{T}A]\!] = \mathsf{T}[\![A]\!]$$

only when $A = \mathsf{R}^d$.

Question

Can we understand $\mathcal{T}t$ as a kind of derivative at higher types as well?
For instance when $\vdash t : (\mathsf{R} \Rightarrow \mathsf{R}) \Rightarrow \mathsf{R}$ we have

$$\vdash \mathcal{T}t : (\mathsf{R}^2 \Rightarrow \mathsf{R}^2) \Rightarrow \mathsf{R}^2$$

whereas the differential of t should rather be of type

$$(\mathsf{R} \Rightarrow \mathsf{R}^2) \Rightarrow \mathsf{R}^2$$

since $(\mathsf{R} \Rightarrow \mathsf{R})^2 = (\mathsf{R} \Rightarrow \mathsf{R}^2)$

. . . probably not straightforwardly

The derivative wrt. f of t such that $f : \mathsf{R} \Rightarrow \mathsf{R} \vdash t : \mathsf{R}$ involves in general the derivative of f, for instance if

$$t = (f)(f)\underline{42}$$

then (with $f, h : \mathsf{R} \Rightarrow \mathsf{R}$)

$$t'(f) \cdot h = (h)(f)\underline{0} + f'(f(\underline{0})) \cdot h(\underline{42})$$

depending linearly on the function h.

Whereas in AD

$$\mathcal{T}t(f) = (f)(f)(\underline{42}, \underline{1})$$

where now f has type $\mathsf{R}^2 \Rightarrow \mathsf{R}^2$.

DiLL and CD

Origins of DiLL

In the 1960's Christopher Stachey promotes a *mathematical semantics* of programs: what function does a program compute?

Meaningful also for stateful programs, seen as functions:

$$\text{machine state} \to \text{machine state}$$

In 1969 Christopher Strachey meets Dana Scott: they invent *Denotational Semantics*.

Denotational Semantics (DS)

Types are interpreted as lattices (or more general *domains* with order-completeness properties $\rightsquigarrow$ Domain Theory).

The order relation of these domains reflects the degree of definiteness of partial data.

Program $\mapsto$ monotone and Scott continuous function, that is $f(\sup_{n\in\mathbb{N}} x_n) = \sup_{n\in\mathbb{N}} f(x_n)$.

Scott continuity accounts for the finiteness of computations.

Scott continuity $\Leftrightarrow$ continuity for the Scott topology.

So the standard viewpoint on denotational semantics was mainly *topological*.

Linear Logic: algebraic viewpoint on DS

Girard's LL (1986) reflects the fact that denotational models have an underlying linear structure featuring operations very similar to those of *linear algebra*: tensor product, direct product, linear function space, dual etc.

Of course such models have also non linear morphisms.

The *exponential* modality of LL explains the connection between the linear and the non-linear worlds (categories).

Basic principle: we can forget that a function is linear, this is *dereliction*.

Differentiation in LL

Differential LL axiomatizes the converse operation:

$$\text{dereliction} : \text{linear} \to \text{non-linear}$$
$$\text{differentiation} : \text{non-linear} \to \text{linear}$$

reformulating the standard laws of the differential calculus.

Then differentiation becomes a very general *logical* operation.

Until recently DiLL was strongly non-deterministic, there was a deduction rule

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash A}{\Gamma \vdash A}\ (+)$$

apparently required to take into account the Leibniz rule $(uv)' = u'v + uv'$.

Coherent differentiation

Very recently we have developed a new approach which doesn't require this rule anymore.

Leads to a differential λ-calculus Λ_{CD} which has some similarities with Λ_{AD}.

Linearity in Λ_{CD}

We don't need to have a ground type of real numbers.

LL linearity is in some sense intrinsic, it does not rely on a specific choice of coefficients. Coefficients are not a data-type.

And so is differentiation in LL and in Λ_{CD}: it is agnostic as to coefficients. Differentiation is orthogonal to data-types.

Types of Λ_{CD}

$$A, B, \cdots := \mathsf{D}^d \iota \mid A \Rightarrow B$$

where d is an abitrary element of $\mathbb{N}$.

The ground type ι is the type of integers. We could also have a type of booleans and many more discrete data types (recursive types).

Then one extends D to all types

$$\mathrm{D}(\mathrm{D}^d\iota) = \mathrm{D}^{d+1}\iota \qquad\qquad \mathrm{D}(A \Rightarrow B) = (A \Rightarrow \mathrm{D}B)$$

Intuition

DA is the type of pairs (u, v) with $u, v : A$ and $u + v : A$.

$\mathrm{D}A \neq A \times A$ in general: in Λ_{CD} one deals with situations where this sum $u + v$ does not always exist: we drop the $(+)$ rule of DiLL.

So an $u : \mathrm{D}^d A$ should be thought of as a balanced tree with 2^d leaves labeled by elements $u_1, \ldots, u_{2^d} : A$ such that $\sum_{i=1}^{2^d} u_i : A$

Term syntax: very close to that of Λ_{AD}, 3 kinds of construct

- λ-calculus
- arithmetics
- differentiation and tree management.

$$M, N, \cdots := x \mid \lambda x : A \cdot M \mid (M)N \mid \mathsf{Y}M$$
$$\mid \underline{n} \mid \mathsf{ifz}^d(M, P, Q) \mid \mathsf{succ}^d(M) \mid \cdots$$
$$\mid \mathsf{D}M \mid \pi_i^d(M) \mid \iota_i^d(M) \mid \theta^d(M) \mid \mathsf{c}_I^d(M) \mid 0^A \mid M + N$$

The exponents d express at which depth in the tree $u : \mathsf{D}^e A$ (with $e \geq d$) the corresponding construct should be applied.

Ordinary typing rules

The λ-calculus rules are as in Λ_{AD}.

The arithemtic rules must take depth into account, for instance

$$\frac{}{\Gamma \vdash \underline{n} : \iota} \qquad \frac{\Gamma \vdash M : \mathsf{D}^d\iota \quad \Gamma \vdash P : A \quad \Gamma \vdash Q : A}{\Gamma \vdash \mathsf{ifz}^d(M, P, Q) : \mathsf{D}^d A}$$

Some differential / tree typing rules

$$\frac{\Gamma \vdash M : A \Rightarrow B}{\Gamma \vdash \mathsf{D}M : \mathsf{D}A \Rightarrow \mathsf{D}B}$$

Intuitively $\mathsf{D}M$ maps (x, u) to $(M(x), M'(x) \cdot u)$ exactly as $\mathcal{T}t$ in Λ_{AD} *at ground types*. Here differentiation makes sense at all types.

$$\frac{\Gamma \vdash M : \mathsf{D}^{d+2}A}{\Gamma \vdash \theta^d(M) : \mathsf{D}^{d+1}A}$$

Intuitively, if $M : \mathsf{D}^2A$ represents $((u_{00}, u_{01}), (u_{10}, u_{11}))$ then $\theta^0(M) : \mathsf{D}A$ represents $(u_{00}, u_{01} + u_{10})$

$$\frac{\Gamma \vdash M : \mathsf{D}^d A}{\Gamma \vdash \iota_0^d(M) : \mathsf{D}^{d+1} A}$$

If $M : A$ represents u then $\iota_0^0(M)$ represents $(u, 0)$.

Similarly for $\iota_1^d(M)$ on the other side.

Dually

$$\frac{\Gamma \vdash M : \mathsf{D}^{d+1} A}{\Gamma \vdash \pi_i^d(M) : \mathsf{D}^d A}$$

implements the obvious projections for $i = 0, 1$.

The most puzzling rule is perhaps

$$\frac{\Gamma \vdash M : \mathsf{D}^{d+l+2}A}{\Gamma \vdash \mathsf{c}_l^d(M) : \mathsf{D}^{d+l+2}A}$$

which implements a *circular permutation* of length $l+2$ at depth d in the access words in the tree represented by M.

Example

If $d = 0$, $l = 1$ and $M : \mathsf{D}^3 A$ represents

$$(((u_{000}, u_{001}), (u_{010}, u_{011})), ((u_{100}, u_{101}), (u_{110}, u_{111})))$$

then $\mathsf{c}_1^0(M)$ represents

$$(((u_{000}, u_{100}), (u_{001}, u_{101})), ((u_{010}, u_{110}), (u_{011}, u_{111})))$$

Cf the *standard flip* in tangent categories.

The differential reduction

Assuming $\Gamma, x : A \vdash M : B$

$$\mathsf{D}(\lambda x : A \cdot M) \to \lambda x : \mathsf{D}A \cdot \partial(x, M)$$

where $\partial(x, M)$ is an operation defined by induction on M such that $\Gamma, x : \mathsf{D}A \vdash \partial(x, M) : \mathsf{D}B$.

Remark

The definition of $\partial(x, M)$ is homomorphic wrt. the structure of terms, very much like the definition of $\mathcal{T}$ in Λ_{AD} though slightly more complicated!

Sums induced by Leibniz are not performed immediately, their position are marked by the construct $\theta^d(\_)$.

Major difference wrt. the definition of $\frac{\partial M}{\partial x} \cdot x$ in the differential λ-calculus, which is an inefficient symbolic differentiation.

Some cases of the def. of $\partial(x, M)$

- $\partial(x, x) = x$
- $\partial(x, y) = \iota_0^0(y)$
- $\partial(x, \lambda y : A \cdot M) = \lambda y : \mathrm{D}A \cdot \partial(x, M)$
- $\partial(x, (M)N) = (\theta^0(\mathrm{D}\partial(x, M)))\partial(x, N)$

Indeed if $\Gamma, x : A \vdash M : B \Rightarrow C$ and $\Gamma, x : A \vdash N : B$

$$\dfrac{\dfrac{\dfrac{\Gamma, x : \mathrm{D}A \vdash \partial(x, M) : B \Rightarrow \mathrm{D}C}{\Gamma, x : \mathrm{D}A \vdash \mathrm{D}\partial(x, M) : \mathrm{D}B \Rightarrow \mathrm{D}^2 C}}{\Gamma, x : \mathrm{D}A \vdash \theta^0(\mathrm{D}\partial(x, M)) : \mathrm{D}B \Rightarrow \mathrm{D}C} \qquad \Gamma, x : \mathrm{D}A \vdash \partial(x, N) : \mathrm{D}B}{\Gamma, x : \mathrm{D}A \vdash \partial(x, (M)N) : \mathrm{D}C}$$

Notice that $(\mathrm{D}B \Rightarrow \mathrm{D}^2 C) = \mathrm{D}^2(\mathrm{D}B \Rightarrow C)$.

Remark

Actually $(\mathrm{D}, \theta^0(\_), \iota_0^0(\_))$ is a strong monad.

Works also for fixpoints.

- $\partial(x, \mathsf{Y}M) = \mathsf{Y}\theta^0(\mathsf{D}\partial(x, M))$

Assuming $\Gamma, x : A \vdash M : B \Rightarrow B$ so that $\Gamma, x : A \vdash \mathsf{Y}M : B$, we have

$$\dfrac{\dfrac{\dfrac{\Gamma, x : \mathsf{D}A \vdash \partial(x, M) : B \Rightarrow \mathsf{D}B}{\Gamma, x : \mathsf{D}A \vdash \mathsf{D}\partial(x, M) : \mathsf{D}B \Rightarrow \mathsf{D}^2 B}}{\Gamma, x : \mathsf{D}A \vdash \theta^0(\mathsf{D}\partial(x, M)) : \mathsf{D}B \Rightarrow \mathsf{D}B}}{\Gamma, x : \mathsf{D}A \vdash \mathsf{Y}\theta^0(\mathsf{D}\partial(x, M)) : \mathsf{D}B}$$

The $\mathsf{c}_l^d(M)$ construct is very important, for instance

- $\partial(x, \mathsf{D}M) = \mathsf{c}_0^0(\mathsf{D}\partial(x, M))$

typed as follows, assuming that $\Gamma, x : A \vdash M : B \Rightarrow C$ so that $\Gamma, x : A \vdash \mathsf{D}M : \mathsf{D}B \Rightarrow \mathsf{D}C$

$$\dfrac{\dfrac{\Gamma, x : \mathsf{D}A \vdash \partial(x, M) : B \Rightarrow \mathsf{D}C}{\Gamma, x : \mathsf{D}A \vdash \mathsf{D}\partial(x, M) : \mathsf{D}B \Rightarrow \mathsf{D}^2C}}{\Gamma, x : \mathsf{D}A \vdash \mathsf{c}_0^0(\mathsf{D}\partial(x, M)) : \mathsf{D}B \Rightarrow \mathsf{D}^2C}$$

Why do basic operations act at depth d?

This is required by the definition of $\partial(x, M)$.

The case of the successor.

- $\partial(x, \mathsf{succ}^d(M)) = \mathsf{succ}^{d+1}(\partial(x, M))$

Assuming $\Gamma, x : A \vdash M : \mathsf{D}^d\iota$ we have:

$$\frac{\Gamma, x : \mathsf{D}A \vdash \partial(x, M) : \mathsf{D}^{d+1}\iota}{\Gamma, x : \mathsf{D}A \vdash \mathsf{succ}^{d+1}(\partial(x, M)) : \mathsf{D}^{d+1}\iota}$$

Reflects the linearity of $\mathsf{succ}(\_)$.

Remark

All these typing and reduction rules are justified (and actually inspired) by a general categorical semantics which has many instances which are well known models of LL.

- Relational semantics, finiteness spaces, profunctors (actually all models of the old DiLL), but also:
- Girard's coherence spaces and hypercoherences with multiset exponential.
- Non-uniform coherence spaces (Bucciarelli and E.).
- Probabilistic coherence spaces (PCS, Danos and E.).
- What about game models?

What is the meaning of this differential?

In the PCS model, **Bool** is interpreted as the set of all probability sub-distributions on $\{\mathbf{t}, \mathbf{f}\}$.

$M : \mathbf{Bool} \times \mathbf{Bool} \to \mathbf{Bool}$ *e.g.* is seen as a subdistribution transformer and as such is a *very regular funtion*: typically analytic.

More precisely $M(x, y)$ is analytic at x, y s.t. $x_{\mathbf{t}} + x_{\mathbf{f}} < 1$ and $y_{\mathbf{t}} + y_{\mathbf{f}} < 1$. For such x, y, and abitrary u, one can compute

$$\frac{\partial M(x, y)}{\partial y} \cdot u \quad \text{algebraically linear in } u$$

The Λ_{CD} computes exactly this kind of derivative.

A $\mathbf{Bool} \to \mathbf{Bool}$ example

Consider the recursive program M:

$$x : \mathbf{Bool} \vdash M : \mathbf{Bool} \qquad M = \mathrm{if}(x, \mathrm{if}(x, M, \mathbf{t}), \mathrm{if}(x, \mathbf{f}, M))$$

then $f = [\![M]\!]_{x:\mathbf{Bool}}$ in the model of probabilistic coherence spaces is the "least" function $f : \mathrm{P}(\mathbf{Bool}) \to \mathrm{P}(\mathbf{Bool})$ such that

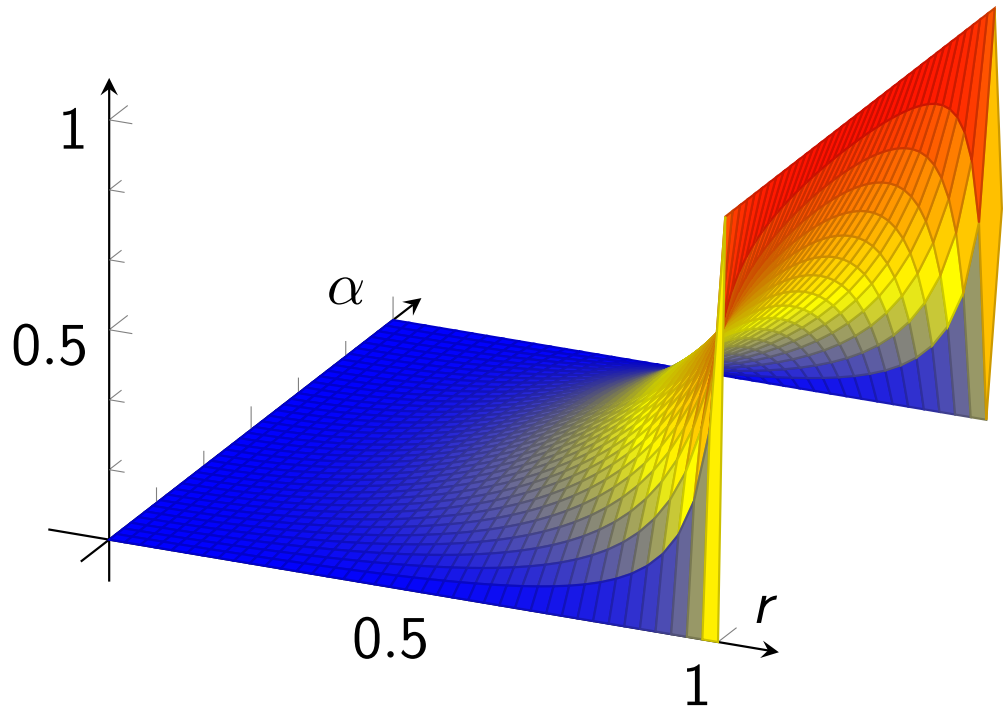
$$f(x) = (x_{\mathbf{t}}^2 + x_{\mathbf{f}}^2) f(x) + x_{\mathbf{t}} x_{\mathbf{f}} (\mathbf{t} + \mathbf{f})$$

$$f(x) = \begin{cases} 0 & \text{if } x_{\mathbf{t}} = 1 \text{ or } x_{\mathbf{f}} = 1 \\ \frac{x_{\mathbf{t}} x_{\mathbf{f}}}{1 - x_{\mathbf{t}}^2 - x_{\mathbf{f}}^2} (\mathbf{t} + \mathbf{f}) & \text{otherwise} \end{cases}$$

$\mathrm{P}(\mathbf{Bool})$: the space of proba subdistr. on $\{\mathbf{t}, \mathbf{f}\}$.

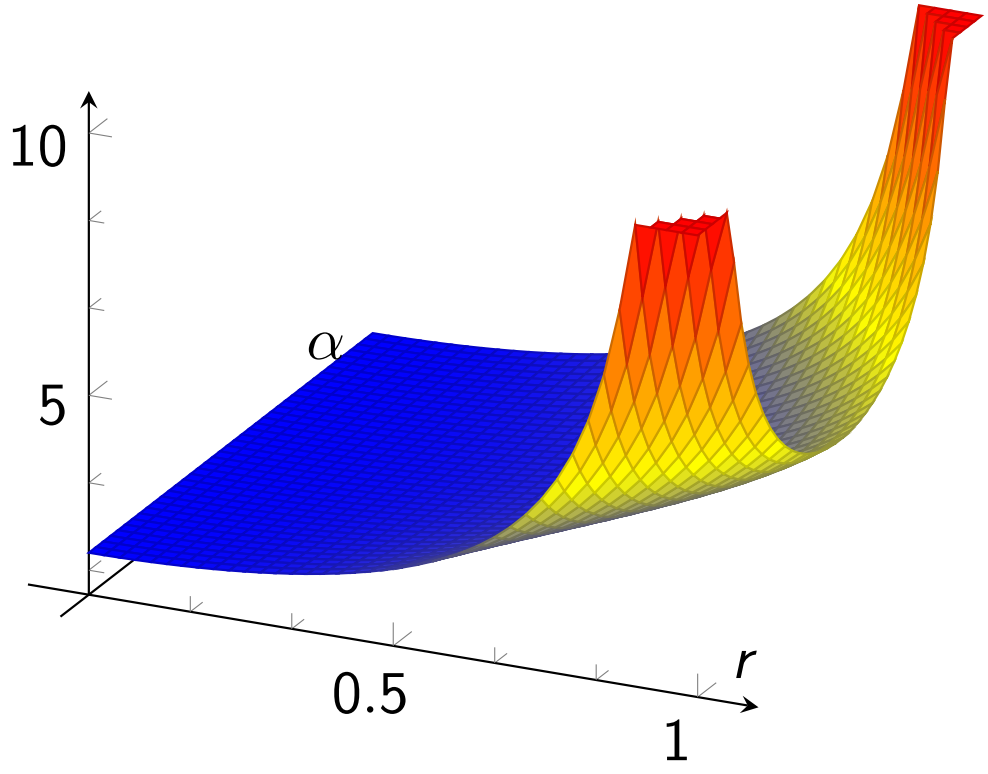
Convergence probability of f: with $r\alpha\mathbf{t} + r(1-\alpha)\mathbf{f} = x \in \mathrm{P}(\mathbf{Bool})$,

$$g(r,\alpha) = f(r\alpha\mathbf{t} + r(1-\alpha)\mathbf{f})_{\mathbf{t}} + f(r\alpha\mathbf{t} + r(1-\alpha)\mathbf{f})_{\mathbf{f}}$$



For $r < 1$ we can see $g(r, \_)$ as an analytic approximation of $g(1, \_)$.

$\min(10, r\frac{\partial g}{\partial r}/g) = \min(10, 2/(1 - r^2(1 - 2\alpha + 2\alpha^2)))$:



Fact

$r\frac{\partial g}{\partial r}/g$ = expectation of the number of uses of x, conditioned by termination.

Differences: the role of $\mathbb{R}$

- Semantically, extending Λ_{CD} to continuous data-types ($\mathbb{R}$) is not straightforward: the interpretation of R is the "cone" of finite positive measures on $\mathbb{R}$, not the real line itself.
- Example of a major difference between Λ_{AD} and Λ_{CD}: in the latter

$$\mathsf{ifz} : \mathsf{N} \,\&\, (X \,\&\, X) \to X$$

  is bilinear, in the former, ifp is not even continuous (replacing N with $\mathbb{R}$).
- In CD coefficients are not values. Linearity has clearly not the same status in both settings though the difference is not completely clear yet. The BMP backpropagation approach to AD is based on LL as well!

Similarity: syntaxes look close

- Formally Λ_{AD} and Λ_{CD} look very close! We can hope to use the homomorphic Λ_{CD} syntax to extend Λ_{AD} to higher types, and conversely to import backpropagation ideas from Λ_{AD} to Λ_{CD}.
- More specifically: Λ_{CD} provides an evaluation mechanism based on an environment-free Krivine machine whose states are (δ, M, s) where M is a term, s is a stack and δ is a sequence of bits. The diff/tree constructions are instructions for handling δ. Could such a mechanism make sense in AD?

Can CD meet AD?