

Beautiful differentiation

Conal Elliott

LambdaPix

2009-09-01 & 2013-07-18

Aside: functions as numbers

Often done in math.

$$d(u + v) \equiv du + dv$$

$$d(u \cdot v) \equiv dv \cdot u + du \cdot v$$

$$d(-u) \equiv -du$$

$$d(e^u) \equiv du \cdot e^u$$

$$d(\log u) \equiv du/u$$

$$d(\sqrt{u}) \equiv du/(2 \cdot \sqrt{u})$$

$$d(\sin u) \equiv du \cdot \cos u$$

$$d(\cos u) \equiv du \cdot (-\sin u)$$

...

Often done in math.

$$\begin{aligned}
 d(u + v) &\equiv du + dv \\
 d(u \cdot v) &\equiv dv \cdot u + du \cdot v \\
 d(-u) &\equiv -du \\
 d(e^u) &\equiv du \cdot e^u \\
 d(\log u) &\equiv du/u \\
 d(\sqrt{u}) &\equiv du/(2 \cdot \sqrt{u}) \\
 d(\sin u) &\equiv du \cdot \cos u \\
 d(\cos u) &\equiv du \cdot (-\sin u) \\
 &\dots
 \end{aligned}$$

Can we *really* treat functions as numbers?

We can treat functions as numbers.

instance *Num* $\beta \Rightarrow \text{Num } (\alpha \rightarrow \beta)$ **where**

$$u + v = \lambda x \rightarrow u\ x + v\ x$$

$$u * v = \lambda x \rightarrow u\ x * v\ x$$

...

instance *Floating* $\beta \Rightarrow \text{Floating } (\alpha \rightarrow \beta)$ **where**

$$\sin\ u = \lambda x \rightarrow \sin\ (u\ x)$$

$$\cos\ u = \lambda x \rightarrow \cos\ (u\ x)$$

...

We can treat functions as numbers.

instance $Num\ \beta \Rightarrow Num\ (\alpha \rightarrow \beta)$ **where**

$$u + v = \lambda x \rightarrow u\ x + v\ x$$

$$u * v = \lambda x \rightarrow u\ x * v\ x$$

...

instance $Floating\ \beta \Rightarrow Floating\ (\alpha \rightarrow \beta)$ **where**

$$sin\ u = \lambda x \rightarrow sin\ (u\ x)$$

$$cos\ u = \lambda x \rightarrow cos\ (u\ x)$$

...

Note:

$$fmap\ h\ u \equiv \lambda x \rightarrow h\ (u\ x)$$

$$liftA_2\ h\ u\ v \equiv \lambda x \rightarrow h\ (u\ x)\ (v\ x)$$

We can treat functions as numbers, more elegantly.

instance *Num* $\beta \Rightarrow$ *Num* $(\alpha \rightarrow \beta)$ **where**

$(+)$ = *liftA₂* $(+)$

$(*)$ = *liftA₂* $(*)$

...

instance *Floating* $\beta \Rightarrow$ *Floating* $(\alpha \rightarrow \beta)$ **where**

sin = *fmap sin*

cos = *fmap cos*

...

We can treat functions as numbers, more elegantly.

instance *Num* $\beta \Rightarrow$ *Num* $(\alpha \rightarrow \beta)$ **where**

$(+)$ = *liftA₂* $(+)$

$(*)$ = *liftA₂* $(*)$

...

instance *Floating* $\beta \Rightarrow$ *Floating* $(\alpha \rightarrow \beta)$ **where**

sin = *fmap sin*

cos = *fmap cos*

...

where

fmap *h u* $\equiv \lambda x \rightarrow h (u x)$

liftA₂ *h u v* $\equiv \lambda x \rightarrow h (u x) (v x)$

We can treat applicatives as numbers.

instance (*Applicative* f , *Num* β) \Rightarrow *Num* (f β) **where**

$(+)$ = *liftA*₂ (+)

$(*)$ = *liftA*₂ (*)

...

instance (*Functor* f , *Floating* β) \Rightarrow *Floating* (f β) **where**

sin = *fmap* *sin*

cos = *fmap* *cos*

...

We can treat applicatives as numbers.

instance (*Applicative* f , *Num* β) \Rightarrow *Num* (f β) **where**

$(+)$ = *liftA₂* $(+)$

$(*)$ = *liftA₂* $(*)$

...

instance (*Functor* f , *Floating* β) \Rightarrow *Floating* (f β) **where**

sin = *fmap* *sin*

cos = *fmap* *cos*

...

where

instance *Applicative* $((\rightarrow)$ α)

instance *Applicative* $[]$

... *Tree*, *State* s , *Either* e , ...

Differentiation

Derivatives have many uses.

- ▶ optimization
- ▶ root-finding
- ▶ surface normals
- ▶ curve and surface tessellation
- ▶ ...

What's a derivative?

For scalar domain:

$$d :: \text{Scalar } s \Rightarrow (s \rightarrow s) \rightarrow (s \rightarrow s)$$

$$d f x = \lim_{\varepsilon \rightarrow 0} \frac{f(x + \varepsilon) - f x}{\varepsilon}$$

What's a derivative?

For scalar domain:

$$d :: \text{Scalar } s \Rightarrow (s \rightarrow s) \rightarrow (s \rightarrow s)$$

$$d f x = \lim_{\varepsilon \rightarrow 0} \frac{f(x + \varepsilon) - f x}{\varepsilon}$$

What about non-scalar domains?

Return to this question later.

What do we want in a technique?

- ▶ Simple to implement,
- ▶ simple to prove correct,
- ▶ convenient,
- ▶ accurate,
- ▶ efficient, *and*
- ▶ general.

There are three common differentiation techniques.

- ▶ Numeric (approximation)
- ▶ Symbolic
- ▶ “Automatic” (*forward* & *reverse* modes)

Numeric (approximation)

For small h ,

$$d f x \approx \frac{f(x + \varepsilon) - f x}{\varepsilon}$$

Simple but inaccurate.

We can improve accuracy while sacrificing simplicity.

Symbolic differentiation

$$d(u + v) \equiv du + dv$$

$$d(u \cdot v) \equiv dv \cdot u + du \cdot v$$

$$d(-u) \equiv -du$$

$$d(e^u) \equiv du \cdot e^u$$

$$d(\log u) \equiv du/u$$

$$d(\sqrt{u}) \equiv du/(2 \cdot \sqrt{u})$$

$$d(\sin u) \equiv du \cdot \cos u$$

$$d(\cos u) \equiv du \cdot (-\sin u)$$

$$d(\sin^{-1} u) \equiv du/\sqrt{1 - u^2}$$

$$d(\cos^{-1} u) \equiv -du/\sqrt{1 - u^2}$$

$$d(\tan^{-1} u) \equiv du/(u^2 + 1)$$

$$d(\sinh u) \equiv du \cdot \cosh u$$

$$d(\cosh u) \equiv du \cdot \sinh u$$

$$d(\sinh^{-1} u) \equiv du/\sqrt{u^2 + 1}$$

$$d(\cosh^{-1} u) \equiv -du/\sqrt{u^2 - 1}$$

What is automatic differentiation?

- ▶ Computes function & derivative values in tandem
- ▶ “Exact” method
- ▶ Numeric, not symbolic

Scalar, first-order AD

Overload functions to work on function/derivative value pairs:

data $D \alpha = D \alpha \alpha$

For instance,

$$D a a' + D b b' = D (a + b) (a' + b')$$

$$D a a' * D b b' = D (a * b) (b' * a + a' * b)$$

$$\sin (D a a') = D (\sin a) (a' * \cos a)$$

$$\text{sqrt} (D a a') = D (\text{sqrt} a) (a' / (2 * \text{sqrt} a))$$

...

Scalar, first-order AD

Overload functions to work on function/derivative value pairs:

data $D \alpha = D \alpha \alpha$

For instance,

$$D a a' + D b b' = D (a + b) (a' + b')$$

$$D a a' * D b b' = D (a * b) (b' * a + a' * b)$$

$$\sin (D a a') = D (\sin a) (a' * \cos a)$$

$$\text{sqrt} (D a a') = D (\text{sqrt} a) (a' / (2 * \text{sqrt} a))$$

...

Are these definitions correct?

What is automatic differentiation — really?

- ▶ *What* does AD mean?
- ▶ *How* does a correct implementation arise?
- ▶ *Where* else might these answers take us?

What does AD mean?

What does AD mean?

data $D \alpha = D \alpha \alpha$

$toD :: (\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow D \alpha)$
 $toD f = \lambda x \rightarrow D (f x) (d f x)$

Spec: *toD* combinations correspond to function combinations, e.g.,

$toD u + toD v \equiv toD (u + v)$
 $toD u * toD v \equiv toD (u * v)$
 $recip (toD u) \equiv toD (recip u)$
 $sin (toD u) \equiv toD (sin u)$
 $cos (toD u) \equiv toD (cos u)$

i.e., *toD* preserves structure.

How does a correct implementation arise?

How does a correct implementation arise?

Goal: $\forall u. \sin (toD\ u) \equiv toD (\sin u)$

How does a correct implementation arise?

Goal: $\forall u. \sin (toD u) \equiv toD (\sin u)$

Simplify each side:

$$\begin{aligned} \sin (toD u) &\equiv \sin \circ toD u \\ &\equiv \lambda x \rightarrow \sin (toD u x) \\ &\equiv \lambda x \rightarrow \sin (D (u x) (d u x)) \end{aligned}$$

$$\begin{aligned} toD (\sin u) &\equiv \lambda x \rightarrow D (\sin u x) \quad (d (\sin u) x) \\ &\equiv \lambda x \rightarrow D ((\sin \circ u) x) \quad ((d u * \cos u) x) \\ &\equiv \lambda x \rightarrow D (\sin (u x)) \quad (d u x * \cos (u x)) \end{aligned}$$

How does a correct implementation arise?

Goal: $\forall u. \sin (toD u) \equiv toD (\sin u)$

Simplify each side:

$$\begin{aligned} \sin (toD u) &\equiv \sin \circ toD u \\ &\equiv \lambda x \rightarrow \sin (toD u x) \\ &\equiv \lambda x \rightarrow \sin (D (u x) (d u x)) \end{aligned}$$

$$\begin{aligned} toD (\sin u) &\equiv \lambda x \rightarrow D (\sin u x) (d (\sin u) x) \\ &\equiv \lambda x \rightarrow D ((\sin \circ u) x) ((d u * \cos u) x) \\ &\equiv \lambda x \rightarrow D (\sin (u x)) (d u x * \cos (u x)) \end{aligned}$$

Sufficient:

$$\sin (D u x d u x) = D (\sin u x) (d u x * \cos u x)$$

Where else might these answers take us?

Where else might these answers take us?

In this talk:

- ▶ Prettier definitions
- ▶ Higher-order derivatives
- ▶ Higher-dimensional functions

Digging deeper — the scalar chain rule

$$d (g \circ u) x \equiv d g (u x) * d u x$$

For scalar domain & range. Variations for other dimensions.

Define and reuse:

$$(g \bowtie dg) (D ux dux) = D (g ux) (dg ux * dux)$$

For instance,

$$\mathit{sin} = \mathit{sin} \bowtie \mathit{cos}$$

$$\mathit{cos} = \mathit{cos} \bowtie \lambda x \rightarrow -\mathit{sin} x$$

$$\mathit{sqrt} = \mathit{sqrt} \bowtie \lambda x \rightarrow \mathit{recip} (2 * \mathit{sqrt} x)$$

Function overloadings make for prettier definitions.

instance *Floating* $\alpha \Rightarrow$ *Floating* ($D \alpha$) **where**

exp = *exp* \boxtimes *exp*

log = *log* \boxtimes *recip*

sqrt = *sqrt* \boxtimes *recip* ($2 * \text{sqrt}$)

sin = *sin* \boxtimes *cos*

cos = *cos* \boxtimes $-sin$

acos = *acos* \boxtimes *recip* ($-sqrt (1 - \text{sqr})$)

atan = *atan* \boxtimes *recip* ($1 + \text{sqr}$)

sinh = *sinh* \boxtimes *cosh*

cosh = *cosh* \boxtimes *sinh*

sqr $x = x * x$

Scalar, higher-order AD

Generate *infinite towers* of derivatives (Karczmarczuk 1998):

$$\mathbf{data} \ D \ \alpha = D \ \alpha \ (D \ \alpha)$$

Suffices to tweak the chain rule:

$$(g \bowtie dg) \quad (D \ ux_0 \ dux) = D \ (g \ ux_0) \ (dg \ ux_0 * dux) \quad \text{-- old}$$

$$(g \bowtie dg) \ ux@(D \ ux_0 \ dux) = D \ (g \ ux_0) \ (dg \ ux * dux) \quad \text{-- new}$$

Most other definitions can then go through unchanged.

The derivations adapt.

What's a derivative – really?

For scalar domain:

$$d f x = \lim_{\varepsilon \rightarrow 0} \frac{f(x + \varepsilon) - f x}{\varepsilon}$$

What's a derivative – really?

For scalar domain:

$$d f x = \lim_{\varepsilon \rightarrow 0} \frac{f(x + \varepsilon) - f x}{\varepsilon}$$

Redefine: unique scalar s such that

$$\lim_{\varepsilon \rightarrow 0} \frac{f(x + \varepsilon) - f x}{\varepsilon} - s \equiv 0$$

What's a derivative – really?

For scalar domain:

$$df_x = \lim_{\varepsilon \rightarrow 0} \frac{f(x + \varepsilon) - f(x)}{\varepsilon}$$

Redefine: unique scalar s such that

$$\lim_{\varepsilon \rightarrow 0} \frac{f(x + \varepsilon) - f(x)}{\varepsilon} - s \equiv 0$$

Equivalently,

$$\lim_{\varepsilon \rightarrow 0} \frac{f(x + \varepsilon) - f(x) - s \cdot \varepsilon}{\varepsilon} \equiv 0$$

or

$$\lim_{\varepsilon \rightarrow 0} \frac{f(x + \varepsilon) - (f(x) + s \cdot \varepsilon)}{\varepsilon} \equiv 0$$

What's a derivative – really?

$$\lim_{\varepsilon \rightarrow 0} \frac{f(x + \varepsilon) - (f(x) + s \cdot \varepsilon)}{\varepsilon} \equiv 0$$

What's a derivative – really?

$$\lim_{\varepsilon \rightarrow 0} \frac{f(x + \varepsilon) - (f(x) + s \cdot \varepsilon)}{\varepsilon} \equiv 0$$

Now generalize: unique *linear map* T such that:

$$\lim_{\varepsilon \rightarrow 0} \frac{|f(x + \varepsilon) - (f(x) + T \varepsilon)|}{|\varepsilon|} \equiv 0$$

What's a derivative – really?

$$\lim_{\varepsilon \rightarrow 0} \frac{f(x + \varepsilon) - (f(x) + s \cdot \varepsilon)}{\varepsilon} \equiv 0$$

Now generalize: unique *linear map* T such that:

$$\lim_{\varepsilon \rightarrow 0} \frac{|f(x + \varepsilon) - (f(x) + T \varepsilon)|}{|\varepsilon|} \equiv 0$$

Derivatives are linear maps.

Captures all “partial derivatives” for all dimensions.

See *Calculus on Manifolds* by Michael Spivak.

The chain rules all unify into one.

Generalize from

$$d(g \circ u) x \equiv d g (u x) * d u x$$

etc

The chain rules all unify into one.

Generalize from

$$d(g \circ u) x \equiv d g (u x) * d u x$$

etc to

$$d(g \circ u) x \equiv d g (u x) \circ d u x$$

Generalized derivatives

Derivative values are *linear maps*: $\alpha \multimap \beta$.

$$\begin{aligned} d &:: (\text{Vector } s \ \alpha, \text{Vector } s \ \beta) \\ &\Rightarrow (\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow (\alpha \multimap \beta)) \end{aligned}$$

Generalized derivatives

Derivative values are *linear maps*: $\alpha \multimap \beta$.

$$\begin{aligned} d &:: (\text{Vector } s \ \alpha, \text{Vector } s \ \beta) \\ &\Rightarrow (\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow (\alpha \multimap \beta)) \end{aligned}$$

First-order AD:

$$\mathbf{data} \ \alpha \triangleright \beta = D \ \beta \ (\alpha \multimap \beta)$$

Generalized derivatives

Derivative values are *linear maps*: $\alpha \multimap \beta$.

$$\begin{aligned} d &:: (\text{Vector } s \alpha, \text{Vector } s \beta) \\ &\Rightarrow (\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow (\alpha \multimap \beta)) \end{aligned}$$

First-order AD:

$$\mathbf{data} \alpha \triangleright \beta = D \beta (\alpha \multimap \beta)$$

Higher-order AD:

$$\begin{aligned} \mathbf{data} \alpha \triangleright^* \beta &= D \beta (\alpha \triangleright^* (\alpha \multimap \beta)) \\ &\approx \beta \times (\alpha \multimap \beta) \times (\alpha \multimap (\alpha \multimap \beta)) \times \dots \end{aligned}$$

What's a linear map?

Preserves linear combinations:

$$h(s_1 \cdot u_1 + \dots + s_n \cdot u_n) \equiv s_1 \cdot h u_1 + \dots + s_n \cdot h u_n$$

What's a linear map?

Preserves linear combinations:

$$h (s_1 \cdot u_1 + \dots + s_n \cdot u_n) \equiv s_1 \cdot h u_1 + \dots + s_n \cdot h u_n$$

Fully determined by behavior on *basis* of α , so

$$\mathbf{type} \ \alpha \multimap \beta = \mathit{Basis} \ \alpha \xrightarrow{M} \beta$$

Memoized for efficiency.

What's a linear map?

Preserves linear combinations:

$$h (s_1 \cdot u_1 + \dots + s_n \cdot u_n) \equiv s_1 \cdot h u_1 + \dots + s_n \cdot h u_n$$

Fully determined by behavior on *basis* of α , so

$$\mathbf{type} \ \alpha \rightarrow \beta = \mathit{Basis} \ \alpha \xrightarrow{M} \beta$$

Memoized for efficiency.

Vectors, matrices, etc re-emerge as *memo-tries*.

Statically dimension-typed!

What's a basis?

```
class Vector s v  $\Rightarrow$  HasBasis s v where  
  type Basis v :: *  
  coord      :: v  $\rightarrow$  (Basis v  $\rightarrow$  s)  
  basisValue :: Basis v  $\rightarrow$  v
```


instance *HasBasis Double Double* **where**

type *Basis Double* = ()

coord s = $\lambda() \rightarrow s$

basisValue () = 1

instance (*HasBasis s u, HasBasis s v*)

\Rightarrow *HasBasis s (u, v)* **where**

type *Basis (u, v)* = *Basis u* 'Either' *Basis v*

coord (u, v) = *coord u* 'either' *coord v*

basisValue (Left a) = (*basisValue a*, 0)

basisValue (Right b) = (0, *basisValue b*)

Automatic differentiation – naturally

Can we make AD even simpler?

Recall our function overloads:

instance *Num* $\beta \Rightarrow$ *Num* $(\alpha \rightarrow \beta)$ **where**

$(+)$ = *liftA₂* $(+)$

$(*)$ = *liftA₂* $(*)$

...

instance *Floating* $\beta \Rightarrow$ *Floating* $(\alpha \rightarrow \beta)$ **where**

sin = *fmap sin*

cos = *fmap cos*

...

These definitions are standard for *applicative functors*.

Could they work for *D*?

Automatic differentiation – *naturally*

Could we simply define AD via the standard

$$\sin = fmap \sin$$

etc? What is *fmap*?

Automatic differentiation – *naturally*

Could we simply define AD via the standard

$$\text{sin} = \text{fmap sin}$$

etc? What is *fmap*?

Require toD_x be a *natural transformation*:

$$\text{fmap } g \circ \text{toD}_x \equiv \text{toD}_x \circ \text{fmap } g$$

where

$$\text{toD}_x u = D (u x) (d u x)$$

Derive *fmap* from this naturality condition.

Derive AD *naturally*

$$\begin{aligned}
 \text{toD}_x (\text{fmap } g \ u) &\equiv \text{toD}_x (g \circ u) \\
 &\equiv D ((g \circ u) \ x) (d (g \circ u) \ x) \\
 &\equiv D (g (u \ x)) (d \ g (u \ x) \circ d \ u \ x)
 \end{aligned}$$

$$\text{fmap } g (\text{toD}_x \ u) \equiv \text{fmap } g (D (u \ x) (d \ u \ x))$$

Sufficient definition:

$$\text{fmap } g (D \ ux \ dux) = D (g \ ux) (d \ g \ ux \circ dux)$$

Similar derivation for *liftA₂* (for (+), (*), etc).

Sufficient definition:

$$\mathit{fmap} \ g \ (D \ ux \ dux) = D \ (g \ ux) \ (d \ g \ ux \circ dux)$$

Sufficient definition:

$$\mathit{fmap} \ g \ (D \ ux \ dux) = D \ (g \ ux) \ (d \ g \ ux \circ dux)$$

Oops. d doesn't have an implementation.

Sufficient definition:

$$\mathit{fmap} \ g \ (D \ ux \ dux) = D \ (g \ ux) \ (d \ g \ ux \circ dux)$$

Oops. d doesn't have an implementation.

Solution A: Inline fmap for each $\mathit{fmap} \ g$ and rewrite $d \ g$ to known derivative.

Sufficient definition:

$$\mathit{fmap} \ g \ (D \ ux \ dux) = D \ (g \ ux) \ (d \ g \ ux \circ dux)$$

Oops. d doesn't have an implementation.

Solution A: Inline fmap for each $\mathit{fmap} \ g$ and rewrite $d \ g$ to known derivative.

Solution B: Generalize *Functor* to allow non-function arrows, and replace functions by differentiable functions.

Conclusions

- ▶ Specification as a *structure-preserving semantic function*.
- ▶ Implementation *derived systematically* from specification.
- ▶ Prettier implementation via *functions-as-numbers*.
- ▶ *Infinite derivative towers* with nearly no extra code.
- ▶ Generalize to differentiation over *vector spaces*.
- ▶ Even simpler specification/derivation via *naturality*.