Beautiful differentiation

Conal Elliott

LambdaPix

1 September, 2009 ICFP

Differentiation

Derivatives have many uses.

For instance,

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

There are three common differentiation techniques.

- Numeric
- Symbolic
- "Automatic" (*forward* & reverse modes)

What's a derivative?

For scalar domain:

$$d :: Scalar\ s \Rightarrow (s \rightarrow s) \rightarrow (s \rightarrow s)$$

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

What's a derivative?

For scalar domain:

$$d :: Scalar\ s \Rightarrow (s \rightarrow s) \rightarrow (s \rightarrow s)$$

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

What about non-scalar domains?
Return to this question later.

Aside: We can treat functions like numbers.

$$
\begin{array}{l}
\textbf{instance } \mathit{Num}\ \beta \Rightarrow \mathit{Num}\ (\alpha \to \beta) \textbf{ where} \\
\quad u + v = \lambda x \to u\ x + v\ x \\
\quad u * v = \lambda x \to u\ x * v\ x \\
\qquad\qquad \ldots \\
\textbf{instance } \mathit{Floating}\ \beta \Rightarrow \mathit{Floating}\ (\alpha \to \beta) \textbf{ where} \\
\quad \mathit{sin}\ \ u = \lambda x \to \mathit{sin}\ \ (u\ x) \\
\quad \mathit{cos}\ u = \lambda x \to \mathit{cos}\ (u\ x) \\
\qquad\qquad \ldots
\end{array}
$$

We can treat applicatives like numbers.

$$
\begin{array}{l}
\textbf{instance } \mathit{Num}\ \beta \Rightarrow \mathit{Num}\ (\alpha \to \beta) \textbf{ where} \\
\quad (+) = \mathit{liftA}_2\ (+) \\
\quad (*) = \mathit{liftA}_2\ (*) \\
\qquad \ldots \\
\textbf{instance } \mathit{Floating}\ \beta \Rightarrow \mathit{Floating}\ (\alpha \to \beta) \textbf{ where} \\
\quad \mathit{sin} = \mathit{fmap}\ \mathit{sin} \\
\quad \mathit{cos} = \mathit{fmap}\ \mathit{cos} \\
\qquad \ldots
\end{array}
$$

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:

$$\textbf{data}\ D\ \alpha = D\ \alpha\ \alpha$$

For instance,

$$\begin{aligned}
D\ a\ a' + D\ b\ b' &= D\ (a + b)\ (a' + b') \\
D\ a\ a' * D\ b\ b' &= D\ (a * b)\ (b' * a + a' * b) \\
sin\ (D\ a\ a') &= D\ (sin\ a)\ (a' * cos\ a) \\
sqrt\ (D\ a\ a') &= D\ (sqrt\ a)\ (a' / (2 * sqrt\ a)) \\
&\ldots
\end{aligned}$$

Scalar, first-order AD

Overload functions to work on function/derivative value pairs:

$$\textbf{data}\ D\ \alpha = D\ \alpha\ \alpha$$

For instance,

$$
\begin{aligned}
D\ a\ a' + D\ b\ b' &= D\ (a + b)\ (a' + b') \\
D\ a\ a' * D\ b\ b' &= D\ (a * b)\ (b' * a + a' * b) \\
sin\ (D\ a\ a') &= D\ (sin\ a)\ (a' * cos\ a) \\
sqrt\ (D\ a\ a') &= D\ (sqrt\ a)\ (a'\ /\ (2 * sqrt\ a)) \\
&\ldots
\end{aligned}
$$

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?

$$\textbf{data}\ D\ \alpha = D\ \alpha\ \alpha$$

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

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: $\forall u.\ \mathit{sin}\ (\mathit{toD}\ u) \equiv \mathit{toD}\ (\mathit{sin}\ u)$

Simplify each side:

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

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

How does a correct implementation arise?

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

Simplify each side:

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

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

Sufficient:

$$sin\ (D\ ux\ dux) = D\ (sin\ ux)\ (dux * cos\ ux)$$

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,

$$\begin{aligned} sin &= sin \bowtie cos \\ cos &= cos \bowtie \lambda x \rightarrow -sin\ x \\ sqrt &= sqrt \bowtie \lambda x \rightarrow recip\ (2 * sqrt\ x) \end{aligned}$$

Function overloadings make for prettier definitions.

$$
\begin{aligned}
&\textbf{instance } \mathit{Floating}\ \alpha \Rightarrow \mathit{Floating}\ (D\ \alpha)\ \textbf{where}\\
&\quad \mathit{exp} = \mathit{exp} \bowtie \mathit{exp}\\
&\quad \mathit{log} = \mathit{log} \bowtie \mathit{recip}\\
&\quad \mathit{sqrt} = \mathit{sqrt} \bowtie \mathit{recip}\ (2 * \mathit{sqrt})\\
&\quad \mathit{sin} = \mathit{sin} \bowtie \mathit{cos}\\
&\quad \mathit{cos} = \mathit{cos} \bowtie -\mathit{sin}\\
&\quad \mathit{acos} = \mathit{acos} \bowtie \mathit{recip}\ (-\mathit{sqrt}\ (1 - \mathit{sqr}))\\
&\quad \mathit{atan} = \mathit{atan} \bowtie \mathit{recip}\ (1 + \mathit{sqr})\\
&\quad \mathit{sinh} = \mathit{sinh} \bowtie \mathit{cosh}\\
&\quad \mathit{cosh} = \mathit{cosh} \bowtie \mathit{sinh}
\end{aligned}
$$

$$\mathit{sqr}\ x = x * x$$

Scalar, higher-order AD

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

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

Suffices to tweak the chain rule:

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

Most other definitions can then go through unchanged.
The derivations adapt.

What's a derivative – really?

For scalar domain:

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

What's a derivative – really?

For scalar domain:

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

Redefine: unique scalar s such that

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

What's a derivative – really?

For scalar domain:

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

Redefine: unique scalar s such that

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

Equivalently,

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

or

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

What's a derivative – really?

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

What's a derivative – really?

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

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

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

What's a derivative – really?

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

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

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

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*: $\alpha \multimap \beta$.

$$d :: (\mathit{Vector}\ s\ \alpha, \mathit{Vector}\ s\ \beta) \Rightarrow (\alpha \to \beta) \to (\alpha \to (\alpha \multimap \beta))$$

First-order AD:

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

Higher-order AD:

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

What's a linear map?

Preserves linear combinations:

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

What's a linear map?

Preserves linear combinations:

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

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

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

Memoized for efficiency.

What's a linear map?

Preserves linear combinations:

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

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

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

Memoized for efficiency.

Vectors, matrices, etc re-emerge as *memo-tries*.
Statically dimension-typed!

What's a basis?

$$\textbf{class } \mathit{Vector}\ s\ v \Rightarrow \mathit{HasBasis}\ s\ v \textbf{ where}$$
$$\quad \textbf{type } \mathit{Basis}\ v :: *$$
$$\quad \mathit{coord} :: v \rightarrow (\mathit{Basis}\ v \rightarrow s)$$
$$\quad \mathit{basisValue} :: \mathit{Basis}\ v \rightarrow v$$

```haskell
instance HasBasis Double Double where
  type Basis Double = ()
  coord s           = λ() → s
  basisValue ()     = 1

instance (HasBasis s u, HasBasis s v)
         ⇒ 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 overloadings:

$$\textbf{instance}\ \mathit{Num}\ \beta \Rightarrow \mathit{Num}\ (\alpha \rightarrow \beta)\ \textbf{where}$$
$$(+) = \mathit{liftA}_2\ (+)$$
$$(*) = \mathit{liftA}_2\ (*)$$
$$\ldots$$
$$\textbf{instance}\ \mathit{Floating}\ \beta \Rightarrow \mathit{Floating}\ (\alpha \rightarrow \beta)\ \textbf{where}$$
$$\mathit{sin} = \mathit{fmap}\ \mathit{sin}$$
$$\mathit{cos} = \mathit{fmap}\ \mathit{cos}$$
$$\ldots$$

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*? Require toD_x be a *natural transformation*:

$$fmap\ g \circ toD_x \equiv toD_x \circ fmap\ g$$

where

$$toD_x\ u = D\ (u\ x)\ (d\ u\ x)$$

Define *fmap* from this naturality condition.

Derive AD *naturally*

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

$$
fmap\ g\ (toD_x\ u) \equiv fmap\ g\ (D\ (u\ x)\ (d\ u\ x))
$$

Sufficient definition:

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

Similar derivation for $liftA_2$ (for $(+)$, $(*)$, etc).

Sufficient definition:

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

Oops. *d* doesn't have an implementation.

Solution A: Inline *fmap* for each *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*.