Understanding Mooncake.jl

Mooncake.jl is a reverse-mode automatic differentiation framework for Julia which aims, in particular, to support mutation.

Notations

If x is a primal value, then $\dot{x}$ is like $\mathrm{d}x$, and $\overline{x}$ is like $\frac{\partial}{\partial x}$.

Pushforward, or directional derivative

Let $f : X \to Y$ be a function between normed spaces $(X, \|\cdot\|_X)$ and $(Y, \|\cdot\|_Y)$.

The directional derivative of f at x in the direction $\dot{x}$ is

$$\mathbb{D}f[x](\dot{x}) := \lim_{\varepsilon \to 0} \frac{f(x + \varepsilon \dot{x}) - f(x)}{\varepsilon} \tag{1}$$

where the limit of a function $h : \mathbb{R} \to Y$ is understood as to mean:

$$\lim_{\varepsilon \to 0} h(\varepsilon) = h_0 \iff \lim_{\varepsilon \to 0} \|h(\varepsilon) - h_0\|_Y = 0$$

You may prefer to write this as a Fréchet derivative:

$$\lim_{\|\dot{x}\|_X \to 0} \frac{\|f(x + \dot{x}) - f(x) - \mathbb{D}f[x](\dot{x})\|_Y}{\|\dot{x}\|_X} = 0$$

Or in Landau notation:

$$f(x + \dot{x}) = f(x) + \mathbb{D}f[x](\dot{x}) + \mathcal{O}(\dot{x})$$

Adjoints of linear operators

Adjoint A^* of a linear operator A:

$$\langle A(\dot{x}), \overline{y} \rangle = \langle \dot{x}, A^*(\overline{y}) \rangle$$

Define this inner product on the Hilbert space of "types", e.g., on tuples of numbers and vectors:

$$\left\langle (x, \vec{u}), (y, \vec{v}) \right\rangle = xy + \left\langle \vec{u}, \vec{v} \right\rangle$$

This just makes bookkeeping easier: we don't need to write all linear operators as matrices in order to find the adjoint.

Quick questions

- Is forward mode supported? E.g., to differentiate $f : \mathbb{R} \to \mathbb{R}^N$ in one pass.
  - Not currently, but it's in the works.
- Are `CoDual` elements the categorical dual of `Dual` numbers, as in https://higherlogics.blogspot.com/2020/05/dualcodual-numbers-for-forwardreverse.html?
  - No relation; `CoDual` is simply a primal value paired with a tangent (no distinction is drawn between vector spaces and dual vector spaces).
- How does the interface work? If I define a method on `ftype_data()`, how do I make it visible to `fdata()`, which is a *generated* function?
  - Only overload `tangent_type` is you want to. But it should work out-of-the-box with custom types.
- How do I make `tangent(fdata(x), rdata(x)) === x` for `x::MyType`?

Things to look into

- `test_tangent_consistency`, `test_data`
- How does the choice of inner product affect things?