# 8. Algorithmic (automatic) differentiation

## Last time

- Systems of nonlinear equations
- Derivatives in higher dimensions
- Multidimensional Newton

## Goal for today

- Rules for derivatives
- Teach computer to follow rules: **algorithmic differentiation**
- Computing derivatives in higher dimensions

## Derivatives as rules

- Think about differentiating $f(x) = \text{sin}(x + (x + 1)(x^2 + 2))$

## Derivatives as rules

- Think about differentiating $f(x) = \sin(x + (x+1)(x^2+2))$
- We apply **rules** for derivatives of sums, products, compositions
- This feels like following an **algorithm**
- Can we get the computer to execute that for us?

## Derivatives as rules

- Think about differentiating $f(x) = \sin(x + (x + 1)(x^2 + 2))$
- We apply **rules** for derivatives of sums, products, compositions
- This feels like following an **algorithm**
- Can we get the computer to execute that for us?
- We only discuss **forward-mode algorithmic (automatic) differentiation**

## Derivatives (review)

- **Derivative = rate of change**
- Increasing $\iff$ derivative $> 0$

## Derivatives (review)

- **Derivative = rate of change**
- Increasing $\iff$ derivative $> 0$

- Tells us how function looks **locally** (close to a point)
- E.g. Use to analyze dynamics near a fixed point
- Crucial in root finding + optimization

## Derivatives II

- **Derivative** of $f : \mathbb{R} \to \mathbb{R}$ at $a$ is slope of **tangent line**
- **Tangent line**: “touches” graph of function at point
- Best affine approximation to $f$ near $a$.

## Derivatives II

- **Derivative** of $f : \mathbb{R} \to \mathbb{R}$ at $a$ is slope of **tangent line**
- **Tangent line**: “touches” graph of function at point
- Best affine approximation to $f$ near $a$.
- Notation: $f'(a)$, or $\left.\frac{df}{dx}\right|_a$ (if you must).

## Definition of derivative

- Definition of derivative of $f : \mathbb{R} \to \mathbb{R}$ at $a \in \mathbb{R}$:

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

- Intuition: Limit of slopes of **secant lines** (“rise over run”)

## Definition of derivative

- Definition of derivative of $f : \mathbb{R} \to \mathbb{R}$ at $a \in \mathbb{R}$:

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

- Intuition: Limit of slopes of **secant lines** (“rise over run”)

- How can we calculate derivatives numerically?

## How to *calculate* derivatives numerically

- Numerically: Cannot take limit $h \to 0$

## How to *calculate* derivatives numerically

- Numerically: Cannot take limit $h \to 0$
- So don't! Fix $h > 0$ to get **finite difference** approximation:

$$f'(a) \simeq \frac{f(a+h) - f(a)}{h}$$

## How to *calculate* derivatives numerically

- Numerically: Cannot take limit $h \to 0$

- So don't! Fix $h > 0$ to get **finite difference** approximation:

$$f'(a) \simeq \frac{f(a+h) - f(a)}{h}$$

- How good is this approximation? See later

## How to *calculate* derivatives numerically

- Numerically: Cannot take limit $h \to 0$
- So don't! Fix $h > 0$ to get **finite difference** approximation:

$$f'(a) \simeq \frac{f(a+h) - f(a)}{h}$$

- How good is this approximation? See later
- Do better: calculate derivatives exactly!

## Alternative viewpoint: Little-o notation

- Rewrite by hiding annoying limit:

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

## Alternative viewpoint: Little-o notation

- Rewrite by hiding annoying limit:

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

- So $f(a+h) = f(a) + hf'(a) + o(h)$

where $\frac{o(h)}{h} \to 0$ when $h \to 0$

## Alternative viewpoint: Little-o notation

- Rewrite by hiding annoying limit:

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

- So $\quad f(a+h) = f(a) + hf'(a) + o(h)$

  where $\frac{o(h)}{h} \to 0$ when $h \to 0$
- $o(h)$: "some function that goes to 0 *faster* than $h$"
- "Little-o notation"

## Derivative from $f(a+h)$

- We showed that *if* $f$ is differentiable at $a$ then

$$f(a+h) = f(a) + hf'(a) + o(h)$$

## Derivative from $f(a+h)$

- We showed that *if* $f$ is differentiable at $a$ then

$$f(a+h) = f(a) + hf'(a) + o(h)$$

- We also have the **converse**:
  *Suppose* $f(a+h) = A + Bh + o(h)$.
  *Then* $A = f(a)$ *and* $B = f'(a)$.

## Derivative from $f(a+h)$

- We showed that *if* $f$ is differentiable at $a$ then

$$f(a+h) = f(a) + hf'(a) + o(h)$$

- We also have the **converse**:
  *Suppose* $f(a+h) = A + Bh + o(h)$*.*
  *Then* $A = f(a)$ *and* $B = f'(a)$*.*

- Use to calculate derivatives: calculate $f(a+h)$
- Put in this form to extract derivative as coefficient of $h$

## Infinitesimals

- Simplify by thinking of “infinitesimal” perturbation $\epsilon$
- With $\epsilon^2 = 0$

## Infinitesimals

- Simplify by thinking of “infinitesimal” perturbation $\epsilon$
- With $\epsilon^2 = 0$

- Manipulate first-order Taylor expansions:
- $f(a + \epsilon) = f(a) + \epsilon f'(a)$

## Infinitesimals

- Simplify by thinking of “infinitesimal” perturbation $\epsilon$
- With $\epsilon^2 = 0$

- Manipulate first-order Taylor expansions:
- $f(a + \epsilon) = f(a) + \epsilon f'(a)$

- Expand $f(a + \epsilon) = A + B\epsilon$
- Coefficient of $\epsilon$ is derivative

## Sum rule for derivatives

- Let’s derive some of the rules of derivatives from calculus

## Sum rule for derivatives

- Let’s derive some of the rules of derivatives from calculus
- Sum of two functions:

$$(f+g)(x) := f(x) + g(x)$$

## Sum rule for derivatives

- Let's derive some of the rules of derivatives from calculus
- Sum of two functions:

$$(f+g)(x) := f(x) + g(x)$$

- Want $(f+g)'(a)$ so calculate $[f+g](a+\epsilon)$:

## Sum rule for derivatives

- Let's derive some of the rules of derivatives from calculus
- Sum of two functions:

$$(f+g)(x) := f(x) + g(x)$$

- Want $(f+g)'(a)$ so calculate $[f+g](a+\epsilon)$:

$$\begin{aligned}[f+g](a+\epsilon) &= f(a+\epsilon) + g(a+\epsilon) \\ &= [f(a)+\epsilon f'(a)] + [g(a)+\epsilon g'(a)] \\ &= [f(a)+g(a)] + [f'(a)+g'(a)]\epsilon\end{aligned}$$

- Hence $(f+g)'(a)$ = (coefficient of $\epsilon$) = $f'(a)+g'(a)$.

## Product rule for derivatives

- Product of two functions:

$$(f \cdot g)(x) := f(x) \cdot g(x)$$

(Here $\cdot$ is normal scalar multiplication)

## Product rule for derivatives

- Product of two functions:

$$(f \cdot g)(x) := f(x) \cdot g(x)$$

(Here · is normal scalar multiplication)

- Calculate $(f \cdot g)(a + \epsilon)$:

$$\begin{aligned}[f \cdot g](a + \epsilon) &= f(a + \epsilon) \cdot g(a + \epsilon) \\ &= [f(a) + \epsilon f'(a)] \cdot [g(a) + \epsilon g'(a)] \\ &= [f(a) \cdot g(a)] + [f(a)g'(a) + g(a)f'(a)]\epsilon.\end{aligned}$$

- Hence $(f \cdot g)'(a)$ = $f(a)g'(a) + g(a)f'(a)$.

## Derivatives from rules: Algorithmic differentiation

- For complicated function, execute each rule in turn
- E.g. $h(x) = 3x^2 + 2x = +(3 * (x * x), 2 * x)$

## Derivatives from rules: Algorithmic differentiation

- For complicated function, execute each rule in turn
- E.g. $h(x) = 3x^2 + 2x = +(3 * (x * x), 2 * x)$

- What information do we need for each function?

## Required information

- Fix point $a$ where taking derivatives
- For function $f$, need exactly *2* pieces of information:
  - value $f(a)$
  - derivative $f'(a)$
- Represent $f$ as

$$f \rightsquigarrow (f(a), f'(a))$$

## Required information

- Fix point $a$ where taking derivatives
- For function $f$, need exactly *2* pieces of information:
  - value $f(a)$
  - derivative $f'(a)$
- Represent $f$ as

$$f \rightsquigarrow (f(a), f'(a))$$

- How can we represent this in Julia?

## Representation in Julia

- Need to group together 2 pieces of information

## Representation in Julia

- Need to group together 2 pieces of information
- Could use tuple or vector etc.
- But want novel **behaviour**, i.e. rules for $+$ and $\times$.

## Representation in Julia

- Need to group together 2 pieces of information
- Could use tuple or vector etc.
- But want novel **behaviour**, i.e. rules for $+$ and $\times$.

- So *define a new type*
- Commonly called **dual number**

## Dual number type in Julia

- Make an immutable dual number type:

## Dual number type in Julia

- Make an immutable dual number type:

~~~
```julia
struct Dual{T<:Real}
    value::T
    deriv::T
end
```
~~~

. . .

- Recall: composite type is a template for box containing data
- `T` is a **type parameter**, any subtype of real numbers
- **`Dual(a, b)` corresponds directly to** $a + \epsilon b$

## Creating dual numbers

- Create dual number by calling constructor, as usual:

```
julia> Dual(3, 4)
Dual{Int64}(3, 4)
```

- Note that Julia automatically **infers** the type `T` as `Int64`

## Implementing arithmetic

- To implement arithmetic, import relevant functions:

```
import Base: +, *
```

- Add methods acting on objects of type `Dual`:

```
+(f::Dual, g::Dual) = Dual(f.value + g.value,
                           f.deriv + g.deriv)
```

## Differentiation using dual numbers

- Suppose have Julia function like `f(x) = x^2 + 2x`
- How can we differentiate $f$ at $a = 3$?

## Differentiation using dual numbers

- Suppose have Julia function like `f(x) = x^2 + 2x`
- How can we differentiate $f$ at $a = 3$?

- We need $\epsilon$ components for derivative
- So want to calculate $f(a + \epsilon) = f(a) + \epsilon f'(a)$

## Differentiation using dual numbers

- Suppose have Julia function like `f(x) = x^2 + 2x`
- How can we differentiate $f$ at $a = 3$?

- We need $\epsilon$ components for derivative
- So want to calculate $f(a + \epsilon) = f(a) + \epsilon f'(a)$

## Differentiation II

- Recall that $a + b\epsilon$ corresponds to `Dual(a, b)`
- So define `xx = Dual(a, 1)`
- And call `f(xx)`
- [Represents identity function $x \mapsto x$ at $x = a$, with derivative $1$].

## Higher-dimensional functions

- How calculate derivatives of higher-dimensional functions?
- Consider $f(x, y)$
- Suppose take dual numbers with *same* $\epsilon$:
- Set $x = a + c\epsilon$ and $y = b + d\epsilon$
- Calculate

$$f(a + c\epsilon, b + d\epsilon)$$

## Directional derivative

- For $f : \mathbb{R}^n \to \mathbb{R}$, derivative is

$$\lim_{\epsilon \to 0} \frac{1}{\epsilon} f(\mathbf{a} + \epsilon \mathbf{v})$$

- **Directional derivative** in direction **v**

## Directional derivative

- For $f : \mathbb{R}^n \to \mathbb{R}$, derivative is

$$\lim_{\epsilon \to 0} \frac{1}{\epsilon} f(\mathbf{a} + \epsilon \mathbf{v})$$

- **Directional derivative** in direction **v**
- In 2D, define

$$g(\epsilon) := f(a + v_1 \, \epsilon, b + v_2 \, \epsilon)$$

## Directional derivative II

- Expand $g(\epsilon) := f(a + v_1\,\epsilon, b + v_2\,\epsilon)$:

## Directional derivative II

- Expand $g(\epsilon) := f(a + v_1\,\epsilon, b + v_2\,\epsilon)$:

$$g(\epsilon) = f(a,b) + v_1\,\epsilon\,\frac{\partial f}{\partial x}(a,b) + v_2\,\epsilon\,\frac{\partial f}{\partial y}(a,b)$$

- Coefficient of $\epsilon$ is

$$v_1\,\frac{\partial f}{\partial x}(a,b) + v_2\,\frac{\partial f}{\partial y}(a,b) = \nabla f(a,b) \cdot \mathbf{v}$$

## Directional derivative II

- Expand $g(\epsilon) := f(a + v_1\,\epsilon, b + v_2\,\epsilon)$:

$$g(\epsilon) = f(a,b) + v_1\,\epsilon\,\frac{\partial f}{\partial x}(a,b) + v_2\,\epsilon\,\frac{\partial f}{\partial y}(a,b)$$

- Coefficient of $\epsilon$ is

$$v_1\,\frac{\partial f}{\partial x}(a,b) + v_2\,\frac{\partial f}{\partial y}(a,b) = \nabla f(a,b) \cdot \mathbf{v}$$

- How can we *calculate* this?

## Calculating directional derivatives

- `f(Dual(a, v₁), Dual(b, v₂)` calculates $\nabla f(a,b) \cdot \mathbf{v}$!

## Calculating directional derivatives

- `f(Dual(a, v₁), Dual(b, v₂))` calculates $\nabla f(a, b) \cdot$ **v**!

- **v** $= (1, 0)$ gives $\partial f / \partial x$
- **v** $= (0, 1)$ gives $\partial f / \partial y$

## Jacobian

- For $f : \mathbb{R}^2 \to \mathbb{R}^2$, have $f = (f_1, f_2)$ with

$$f_i(\mathbf{a} + \epsilon \mathbf{v}) = f_i(\mathbf{a}) + \epsilon \nabla f_i(\mathbf{a}) \cdot \mathbf{v}$$

- So

$$f(\mathbf{a} + \epsilon \mathbf{v}) = f(\mathbf{a}) + \epsilon \, Df(\mathbf{a}) \cdot \mathbf{v}$$

- $Df(\mathbf{a})$ is **Jacobian matrix** of partial derivatives $\frac{\partial f_i}{\partial x_j}$
- Coefficient of $\epsilon$ is Jacobian–vector product

## Summary

- We can implement rules to calculate derivatives
- By defining a new type and overloading operations
- Extends directly to derivatives of higher-dimensional functions