I.4 Dual Numbers

Int his chapter we introduce a mathematically beautiful alternative to divided differences for computing derivatives: *dual numbers*. As we shall see, these are a commutative ring that *exactly* compute derivatives, which when implemented in floating point give very high-accuracy approximations to derivatives. They underpin forward-mode automatic differentation.

Before we begin, we must be clear what a "function" is. Consider three possible scenarios:

1. *Black-box function*: this was considered last chapter where we can only input floating-point numbers

and get out a floating-point number. 2. *Generic function*: Consider a function that is a formula (or, equivalentally, a *piece of code*) that we can evaluate on arbitrary types, including custom types that we create. An example is a polynomial:

$$p(x) = p_0 + p_1 x + \cdots + p_n x^n$$

which can be evaluated for x in the reals, complexes, or any other ring. More generally, if we have a function defined in Julia that does not call any C libraries it can be evaluated on different types. 3. *Graph function*: The function is built by composing different basic "kernels" with known differentiability properties. We won't consider this situation in this module, though it is the model used by Python machine learning toolbox's like PyTorch and TensorFlow.

Definition 1 (Dual numbers) Dual numbers $\mathbb{D}$ are a commutative ring (over $\mathbb{R}$) generated by 1 and ϵ such that $\epsilon^2 = 0$. Dual numbers are typically written as $a + b\epsilon$ where a and b are real.

This is very much analoguous to complex numbers, which are a field generated by 1 and i such that $\mathrm{i}^2 = -1$. Compare multiplication of each number type:

$$\begin{aligned}(a + b\mathrm{i})(c + d\mathrm{i}) &= ac + (bc + ad)\mathrm{i} + bd\mathrm{i}^2 = ac - bd + (bc + ad)\mathrm{i} \\ (a + b\epsilon)(c + d\epsilon) &= ac + (bc + ad)\epsilon + bd\epsilon^2 = ac + (bc + ad)\epsilon\end{aligned}$$

And just as we view $\mathbb{R} \subset \mathbb{C}$ by equating $a \in \mathbb{R}$ with $a + 0\mathrm{i} \in \mathbb{C}$, we can view $\mathbb{R} \subset \mathbb{D}$ by equating $a \in \mathbb{R}$ with $a + 0\epsilon \in \mathbb{D}$.

1. Differentiating polynomials

Applying a polynomial to a dual number $a + b\epsilon$ tells us the derivative at a:

Theorem 1 (polynomials on dual numbers) Suppose p is a polynomial. Then

$$p(a+b\epsilon) = p(a) + bp'(a)\epsilon$$

Proof

It suffices to consider $p(x) = x^n$ for $n \geq 1$ as other polynomials follow from linearity.
We proceed by induction: The case $n = 1$ is trivial. For $n > 1$ we have

$$(a+b\epsilon)^n = (a+b\epsilon)(a+b\epsilon)^{n-1} = (a+b\epsilon)(a^{n-1} + (n-1)ba^{n-2}\epsilon) = a^n + bna^{n-1}\epsilon.$$

∎

Example 1 (differentiating polynomial) Consider computing $p'(2)$ where

$$p(x) = (x-1)(x-2) + x^2.$$

We can use Dual numbers to differentiating, avoiding expanding in monomials or rules of differentiating:

$$p(2+\epsilon) = (1+\epsilon)\epsilon + (2+\epsilon)^2 = \epsilon + 4 + 4\epsilon = 4 + \underbrace{5}_{p'(2)}\epsilon$$

2. Differentiating other functions

We can extend real-valued differentiable functions to dual numbers in a similar manner. First, consider a standard function with a Taylor series (e.g. $\cos$, $\sin$, $\exp$, etc.)

$$f(x) = \sum_{k=0}^{\infty} f_k x^k$$

so that a is inside the radius of convergence. This leads naturally to a definition on dual numbers:

$$\begin{aligned} f(a+b\epsilon) &= \sum_{k=0}^{\infty} f_k (a+b\epsilon)^k = f_0 + \sum_{k=1}^{\infty} f_k (a^k + ka^{k-1}b\epsilon) = \sum_{k=0}^{\infty} f_k a^k + \sum_k^{\infty} \\ &= f(a) + bf'(a)\epsilon \end{aligned}$$

More generally, given a differentiable function we can extend it to dual numbers:

Definition 2 (dual extension) Suppose a real-valued function f is differentiable at a. If

$$f(a+b\epsilon) = f(a) + bf'(a)\epsilon$$

then we say that it is a *dual extension at* a.

Thus, for basic functions we have natural extensions:

$$
\begin{aligned}
\exp(a+b\epsilon) &:= \exp(a) + b\exp(a)\epsilon \\
\sin(a+b\epsilon) &:= \sin(a) + b\cos(a)\epsilon \\
\cos(a+b\epsilon) &:= \cos(a) - b\sin(a)\epsilon \\
\log(a+b\epsilon) &:= \log(a) + \frac{b}{a}\epsilon \\
\sqrt{a+b\epsilon} &:= \sqrt{a} + \frac{b}{2\sqrt{a}}\epsilon \\
|a+b\epsilon| &:= |a| + b\,\mathrm{sign} a\,\epsilon
\end{aligned}
$$

provided the function is differentiable at a. Note the last example does not have a convergent Taylor series (at 0) but we can still extend it where it is differentiable.

Going further, we can add, multiply, and compose such functions:

Lemma 1 (product and chain rule) If f is a dual extension at $g(a)$ and g is a dual extension at a, then $q(x) := f(g(x))$ is a dual extension at a. If f and g are dual extensions at a then $r(x) := f(x)g(x)$ is also dual extensions at a. In other words:

$$
\begin{aligned}
q(a+b\epsilon) &= q(a) + bq'(a)\epsilon \\
r(a+b\epsilon) &= r(a) + br'(a)\epsilon
\end{aligned}
$$

Proof For q it follows immediately:

$$
\begin{aligned}
q(a+b\epsilon) &= f(g(a+b\epsilon)) = f(g(a) + bg'(a)\epsilon) \\
&= f(g(a)) + bg'(a)f'(g(a))\epsilon = q(a) + bq'(a)\epsilon.
\end{aligned}
$$

For r we have

$$
\begin{aligned}
r(a+b\epsilon) &= f(a+b\epsilon)g(a+b\epsilon) = (f(a) + bf'(a)\epsilon)(g(a) + bg'(a)\epsilon) \\
&= f(a)g(a) + b(f'(a)g(a) + f(a)g'(a))\epsilon = r(a) + br'(a)\epsilon.
\end{aligned}
$$

∎

A simple corollary is that any function defined in terms of addition, multiplication, composition, etc. of functions that are dual with differentiation will be differentiable via dual numbers.

Example 2 (differentiating non-polynomial)

Consider $f(x) = \exp(x^2 + \mathrm{e}^x)$ by evaluating on the duals:

$$
f(1+\epsilon) = \exp(1 + 2\epsilon + \mathrm{e} + \mathrm{e}\epsilon) = \exp(1+\mathrm{e}) + \exp(1+\mathrm{e})(2+\mathrm{e})\epsilon
$$

and therefore we deduce that

$$
f'(1) = \exp(1+\mathrm{e})(2+\mathrm{e}).
$$

3. Implementation as a special type

We now consider a simple implementation of dual numbers that works on general polynomials:

In [1]:
```
# Dual(a,b) represents a + b*ϵ
struct Dual{T}
    a::T
    b::T
end

# Dual(a) represents a + 0*ϵ
Dual(a::Real) = Dual(a, zero(a)) # for real numbers we use a + 0ϵ

# Allow for a + b*ϵ syntax
const ϵ = Dual(0, 1)

import Base: +, *, -, /, ^, zero, exp

# support polynomials like 1 + x, x - 1, 2x or x*2 by reducing to Dual
+(x::Real, y::Dual) = Dual(x) + y
+(x::Dual, y::Real) = x + Dual(y)
-(x::Real, y::Dual) = Dual(x) - y
-(x::Dual, y::Real) = x - Dual(y)
*(x::Real, y::Dual) = Dual(x) * y
*(x::Dual, y::Real) = x * Dual(y)

# support x/2 (but not yet division of duals)
/(x::Dual, k::Real) = Dual(x.a/k, x.b/k)

# a simple recursive function to support x^2, x^3, etc.
function ^(x::Dual, k::Integer)
    if k < 0
        error("Not implemented")
    elseif k == 1
        x
    else
        x^(k-1) * x
    end
end

# Algebraic operationds for duals
-(x::Dual) = Dual(-x.a, -x.b)
+(x::Dual, y::Dual) = Dual(x.a + y.a, x.b + y.b)
-(x::Dual, y::Dual) = Dual(x.a - y.a, x.b - y.b)
*(x::Dual, y::Dual) = Dual(x.a*y.a, x.a*y.b + x.b*y.a)

exp(x::Dual) = Dual(exp(x.a), exp(x.a) * x.b)
```

Out[1]:
```
exp (generic function with 15 methods)
```

We can also try it on the two polynomials as above:

In [2]:
```
f = x -> 1 + x + x^2
g = x -> 1 + x/3 + x^2
f(ϵ).b, g(ϵ).b
```

Out[2]:
```
(1, 0.3333333333333333)
```

The first example exactly computes the derivative, and the second example is exact up to the last bit rounding! It also works for higher order polynomials:

In [3]:
```
f = x -> 1 + 1.3x + 2.1x^2 + 3.1x^3
f(0.5 + ϵ).b - 5.725
```

Out[3]:
```
8.881784197001252e-16
```

It is indeed "accurate to (roughly) 16-digits", the best we can hope for using floating point.

We can use this in "algorithms" as well as simple polynomials. Consider the polynomial $1 + \ldots + x^n$:

In [4]:
```
function s(n, x)
    ret = 1 + x # first two terms
    for k = 2:n
        ret += x^k
    end
    ret
end
s(10, 0.1 + ϵ).b
```

Out[4]:
```
1.2345678999999998
```

This matches exactly the "true" (up to rounding) derivative:

In [5]:
```
sum((1:10) .* 0.1 .^(0:9))
```

Out[5]:
```
1.2345678999999998
```

Finally, we can try the more complicated example:

In [6]:
```
f = x -> exp(x^2 + exp(x))
f(1 + ϵ)
```

Out[6]:
```
Dual{Float64}(41.193555674716116, 194.362805189629)
```

What makes dual numbers so effective is that, unlike divided differences, they are not prone to disasterous growth due to round-off errors.