Hack the Derivative!

Erik Taubeneck

GAMECHANGER

Software Engineer

October 20th, 2015
American University Math/Stat Dept Colloquium

Outline

Try It Yourself

The functions used through out this presentations can be installed with pip!

```
$ mkdir somewhere_new
$ cd somewhere_new
$ virtualenv venv
$ source venv/bin/activate
$ pip install hackthederivative
$ python
>>> from hackthederivative import *
```

Taking the Derivative

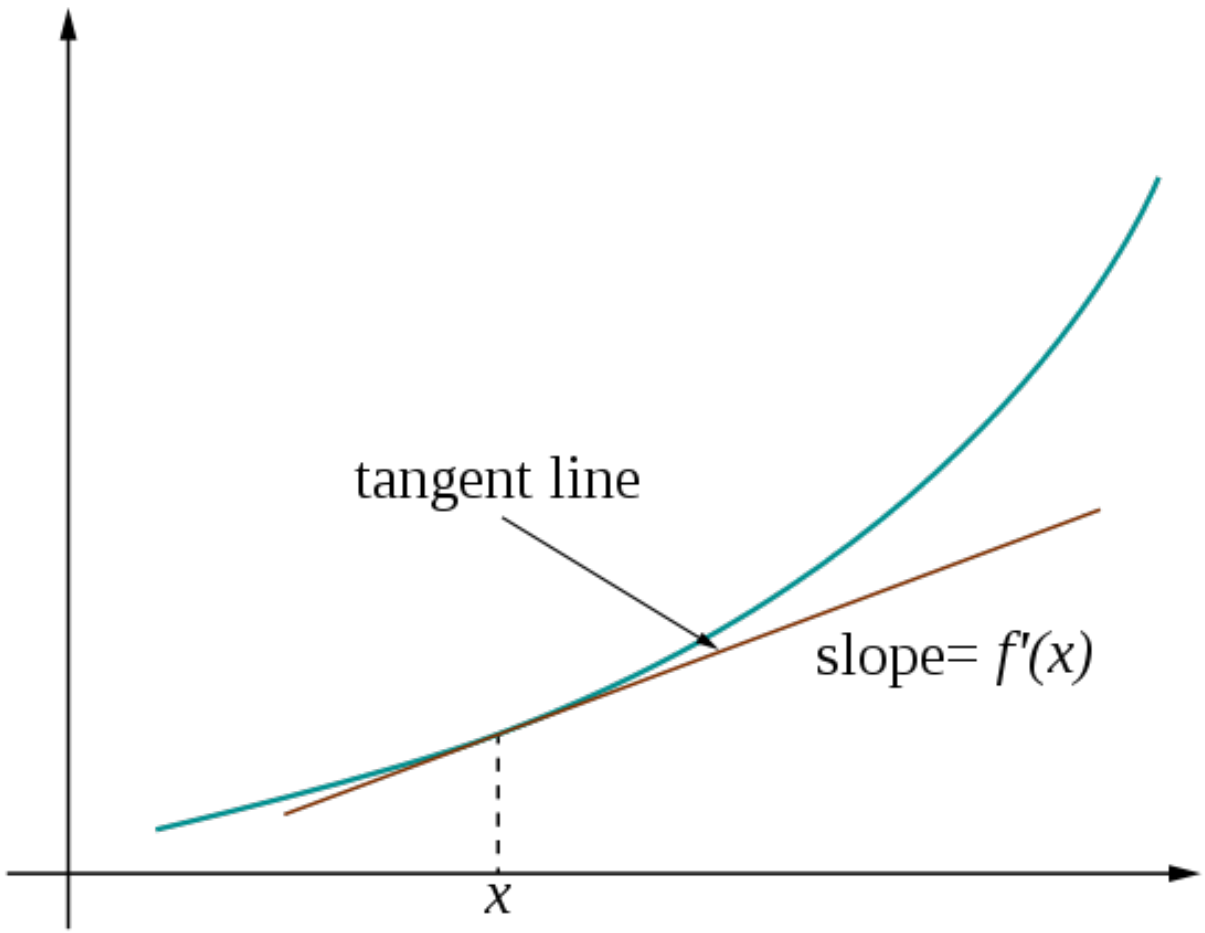
Let $f(x)$ be a function from $\mathbb{R} \to X \subseteq \mathbb{R}$.
e.g.

- $f(x) = x^2$
- $g(x) = sin(x)$
- $h(x) = 5x^3 - 6x + \frac{1}{x}$ [1]

[1] Note that $h(x)$ is not defined at $x = 0$.

Taking the Derivative

The derivative of $f(x)$, $f'(x)$, is a new function which tells us the *slope* of the tangent line that intersects f at any given x.



Taking the Derivative

Formally, the derivative of $f(x)$ at x is

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

Taking the Derivative

$\mathbb{E}^x$:

Let $f(x) = x^2$.

Taking the Derivative

$\mathbb{E}^x$:

Let $f(x) = x^2$. Then

$$f'(x) = \lim_{h \to 0} \frac{(x+h)^2 - x^2}{h}$$

Taking the Derivative

$\mathbb{E}^x$:

Let $f(x) = x^2$. Then

$$\begin{aligned} f'(x) &= \lim_{h \to 0} \frac{(x+h)^2 - x^2}{h} \\ &= \lim_{h \to 0} \frac{x^2 + 2xh + h^2 - x^2}{h} \end{aligned}$$

Taking the Derivative

$\mathbb{E}^x$:

Let $f(x) = x^2$. Then

$$
\begin{aligned}
f'(x) &= \lim_{h \to 0} \frac{(x+h)^2 - x^2}{h} \\
&= \lim_{h \to 0} \frac{x^2 + 2xh + h^2 - x^2}{h} \\
&= \lim_{h \to 0} \frac{2xh + h^2}{h}
\end{aligned}
$$

Taking the Derivative

$\mathbb{E}^x$:

Let $f(x) = x^2$. Then

$$
\begin{aligned}
f'(x) &= \lim_{h \to 0} \frac{(x+h)^2 - x^2}{h} \\
&= \lim_{h \to 0} \frac{x^2 + 2xh + h^2 - x^2}{h} \\
&= \lim_{h \to 0} \frac{2xh + h^2}{h} \\
&= \lim_{h \to 0} 2x + h
\end{aligned}
$$

Taking the Derivative

$\mathbb{E}^x$:

Let $f(x) = x^2$. Then

$$
\begin{aligned}
f'(x) &= \lim_{h \to 0} \frac{(x+h)^2 - x^2}{h} \\
&= \lim_{h \to 0} \frac{x^2 + 2xh + h^2 - x^2}{h} \\
&= \lim_{h \to 0} \frac{2xh + h^2}{h} \\
&= \lim_{h \to 0} 2x + h \\
&= 2x
\end{aligned}
$$

□

Finite Difference

Suppose we wish to find the derivative for some function $f(x)$ a x_0.

Finite Difference

Suppose we wish to find the derivative for some function $f(x)$ a x_0. One approximation is the finite difference method.

Finite Difference

Suppose we wish to find the derivative for some function $f(x)$ a x_0.
One approximation is the finite difference method.
Recall

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

Finite Difference

Suppose we wish to find the derivative for some function $f(x)$ a x_0.
One approximation is the finite difference method.
Recall

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

We can estimate

$$f'(x_o) \approx \frac{f(x_0+h) - f(x_0)}{h}$$

for some small h.

Finite Difference

Implemented in Python

```
In [1]: def finite_difference(f, x, h):
            return (f(x+h) - f(x))/h
```

Finite Difference

Implemented in Python

```
In [1]: def finite_difference(f, x, h):
            return (f(x+h) - f(x))/h

In [2]: f = lambda x: x**2

In [2]: finite_difference(f, 1.0, 0.00001)

Out [3]: 2.00001000001393
```

Finite Difference

But how do we choose h?

Finite Difference

But how do we choose h?

$$\begin{aligned} f(x+h) &= f(x) + \sum_{n=1}^{\infty} \frac{f^{(n)}(x)}{n!} h^n \\ &= f(x) + f'(x)h + \sum_{n=2}^{\infty} \frac{f^{(n)}(x)}{n!} h^n \end{aligned}$$

where $f^{(n)}$ is the n^{th} derivative of f.

Finite Difference

Solving for f', we get

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \sum_{n=1}^{\infty} \frac{f^{(n)}(x)}{n!} h^{(n-1)}$$

and thus the error of such approximation is

$$\mathcal{E}(h, x) = \sum_{n=1}^{\infty} \frac{f^{(n)}(x)}{n!} h^{(n-1)}$$

$$|\mathcal{E}(h, x_0)| \leq \sum_{n=1}^{\infty} \left| \frac{f^{(n)}(x)}{n!} \right| \left| h^{(n-1)} \right|$$

The right hand side is a strictly monotonically increasing function with value 0 when $h = 0$, so the limit of $\lim_{h \to 0} |\mathcal{E}(h, x_n)| = 0$.

Finite Difference

```
In [4]: import sys

In [5]: h = sys.float_info.min
```

Finite Difference

```
In [4]: import sys

In [5]: h = sys.float_info.min

In [6]: print finite_difference(f, 1.0, h)

Out [6]: 0.0
```

Uh oh.

Finite Difference

Hmm, 0.00001 seems pretty small, and worked well earlier. Maybe that will just work.

Finite Difference

Hmm, 0.00001 seems pretty small, and worked well earlier.
Maybe that will just work.

```
In [7]: finite_difference(f, 1.0e20, 0.00001)

Out [7]: 0.0
```

Nope, that can break too.
To figure out why, let's look closer at Floating Point numbers.

64 bits

A 64-bit computer can do exact operations on the integers modulo 18,446,744,073,709,551,616 (2^{64}).

Unsigned: $\{z \in \mathbb{Z} \mid 0 <= z < 2^{64} - 1\}$

Signed: $\{z \in \mathbb{Z} \mid -2^{63} <= z <= 2^{63} - 1\}$

Floating Point Numbers

```
In [1]: import sys

In [2]: sys.float_info.max
Out [2]: 1.7976931348623157e+308

In [3]: sys.float_info.min
Out [3]: 2.2250738585072014e-308
```

Floating Point Numbers

```
In [1]: import sys

In [2]: sys.float_info.max
Out [2]: 1.7976931348623157e+308

In [3]: sys.float_info.min
Out [3]: 2.2250738585072014e-308
```

For comparison:

- There are an estimated 10^{78} to 10^{82} atoms in the visible universe.

Floating Point Numbers

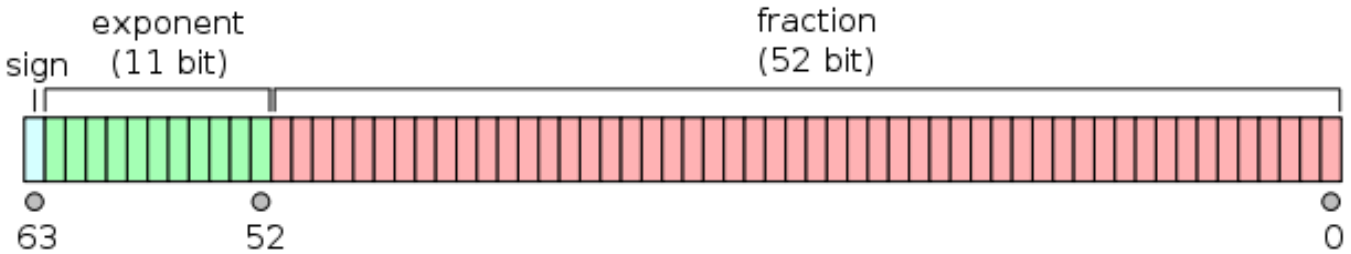
```
In [1]: import sys

In [2]: sys.float_info.max
Out [2]: 1.7976931348623157e+308

In [3]: sys.float_info.min
Out [3]: 2.2250738585072014e-308
```

For comparison:

- There are an estimated 10^{78} to 10^{82} atoms in the visible universe.
- Given a lottery that you played every millisecond from the Big Bang to now, if you were only expected to win once, the probability of winning would be $4 * 10^{-20}$.

Floating Point Numbers

A Float64, or a double-precision floating-point number, consists of



and represents the real number $x \in \mathbb{R}$ with *sign*, e, and b_i such that

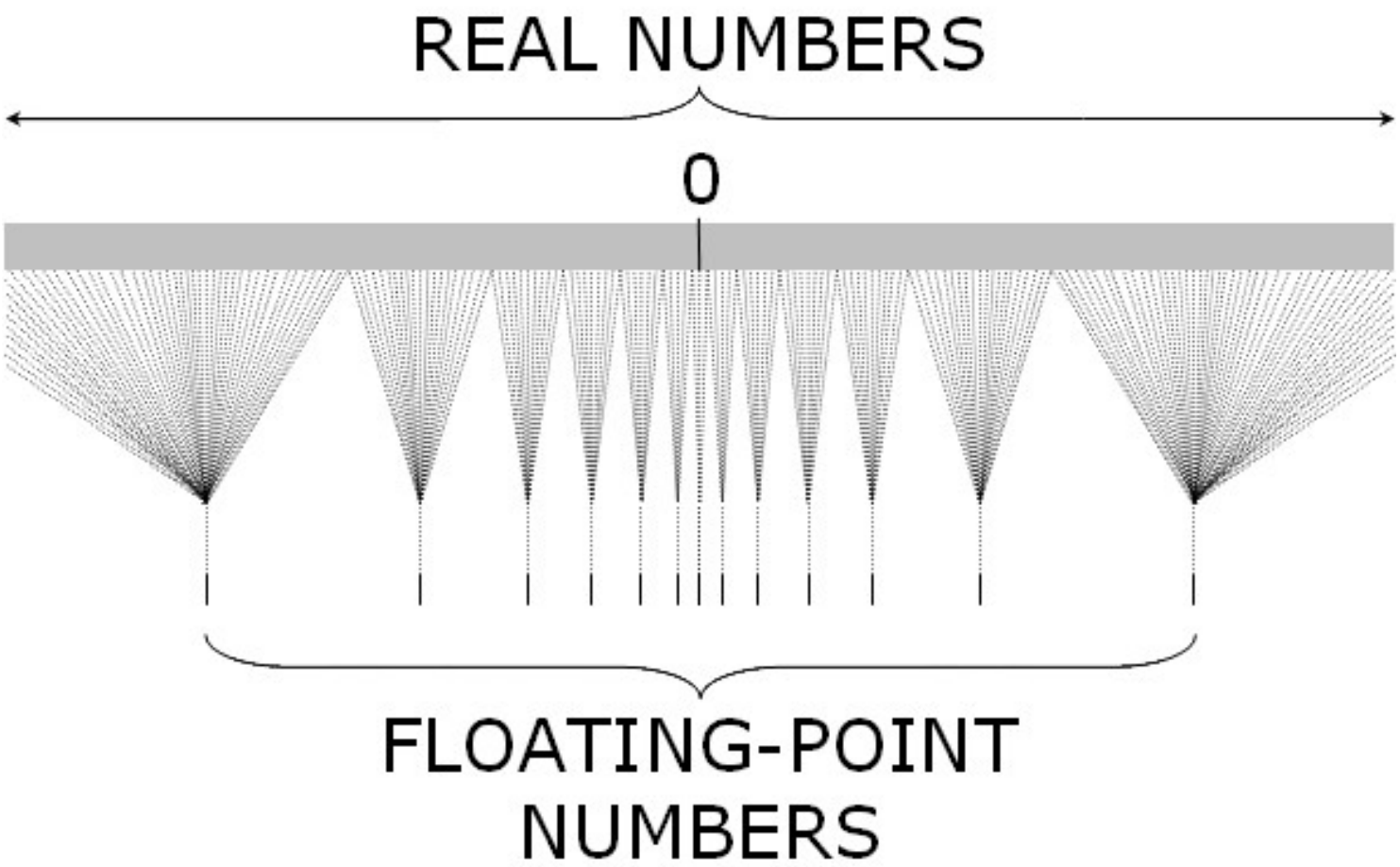
$$(-1)^{sign}(1.b_{51}b_{50}...b_0)_2 \ 2^{e-1023}$$

or (possibly more readable)

$$(-1)^{sign}\left(1+\sum_{i=1}^{52} b_{52-i}2^{-i}\right) \times 2^{e-1023}$$

Floating Point Numbers

The floating point numbers have non-constant gaps!



Floating Point Numbers

Multiplication is helped by Associativity and Commutativity

$$(5.916829373 \times 10^{23}) \times (7.208209342 \times 10^{-51})$$

Floating Point Numbers

Multiplication is helped by Associativity and Commutativity

$$\begin{aligned} & (5.916829373 \times 10^{23}) \times (7.208209342 \times 10^{-51}) \\ = \quad & 5.916829373 \times 10^{23} \times 7.208209342 \times 10^{-51} \end{aligned}$$

Floating Point Numbers

Multiplication is helped by Associativity and Commutativity

$$\begin{aligned} & (5.916829373 \times 10^{23}) \times (7.208209342 \times 10^{-51}) \\ = \quad & 5.916829373 \times 10^{23} \times 7.208209342 \times 10^{-51} \\ = \quad & 5.916829373 \times 7.208209342 \times 10^{23} \times 10^{-51} \end{aligned}$$

Floating Point Numbers

Multiplication is helped by Associativity and Commutativity

$$
\begin{aligned}
& (5.916829373 \times 10^{23}) \times (7.208209342 \times 10^{-51}) \\
= \quad & 5.916829373 \times 10^{23} \times 7.208209342 \times 10^{-51} \\
= \quad & 5.916829373 \times 7.208209342 \times 10^{23} \times 10^{-51} \\
= \quad & (5.916829373 \times 7.208209342) \times (10^{23} \times 10^{-51})
\end{aligned}
$$

Floating Point Numbers

Multiplication is helped by Associativity and Commutativity

$$
\begin{aligned}
& (5.916829373 \times 10^{23}) \times (7.208209342 \times 10^{-51}) \\
= \quad & 5.916829373 \times 10^{23} \times 7.208209342 \times 10^{-51} \\
= \quad & 5.916829373 \times 7.208209342 \times 10^{23} \times 10^{-51} \\
= \quad & (5.916829373 \times 7.208209342) \times (10^{23} \times 10^{-51})
\end{aligned}
$$

but this doesn't work for addition

Floating Point Numbers

Multiplication is helped by Associativity and Commutativity

$$\begin{aligned} & (5.916829373 \times 10^{23}) \times (7.208209342 \times 10^{-51}) \\ = \quad & 5.916829373 \times 10^{23} \times 7.208209342 \times 10^{-51} \\ = \quad & 5.916829373 \times 7.208209342 \times 10^{23} \times 10^{-51} \\ = \quad & (5.916829373 \times 7.208209342) \times (10^{23} \times 10^{-51}) \end{aligned}$$

but this doesn't work for addition

$$(5.916829373 \times 10^{23}) + (7.208209342 \times 10^{-51})$$

Floating Point Gaps

```
In [1]: import sys

In [2]: plus_epsilon_identity(x, eps):
            return x + eps == x

In [3]: eps = sys.float_info.min
```

Floating Point Gaps

```
In [1]: import sys

In [2]: plus_epsilon_identity(x, eps):
            return x + eps == x

In [3]: eps = sys.float_info.min

In [4]: plus_epsilon_identity(0.0, eps)
Out [4]: False

In [5]: plus_epsilon_identity(1.0, eps)
Out [5]: True
```

Floating Point Gaps

```
In [1]: import sys

In [2]: plus_epsilon_identity(x, eps):
            return x + eps == x

In [3]: eps = sys.float_info.min

In [4]: plus_epsilon_identity(0.0, eps)
Out [4]: False

In [5]: plus_epsilon_identity(1.0, eps)
Out [5]: True

In [6]: plus_epsilon_identity(1.0e20, 1.0)
Out [6]: True
```

Problems with the Derivative

If we choose h that is so small such that

$$float(x) + float(h) = float(x)$$

Problems with the Derivative

If we choose h that is so small such that

$$float(x) + float(h) = float(x)$$

then

$$\frac{f(x+h) - f(x)}{h} = \frac{f(x) - f(x)}{h} = \frac{0}{h} = 0$$

Problems with the Derivative

Similarly, if

$$float(f(x+h)) = float(f(x))$$

Problems with the Derivative

Similarly, if

$$float(f(x+h)) = float(f(x))$$

then

$$\frac{f(x+h) - f(x)}{h} = \frac{0}{h} = 0$$

Measuring the Gaps

```
In [1]: def eps(x):
            e = float(max(sys.float_info.min, abs(x)))
            while not plus_epsilon_identity(x, e):
                last = e
                e = e / 2.
            return last
```

Measuring the Gaps

```
In [1]: def eps(x):
            e = float(max(sys.float_info.min, abs(x)))
            while not plus_epsilon_identity(x, e):
                last = e
                e = e / 2.
            return last

In [2]: eps(1.0)
Out[2]: 2.22044604925031e-16
```

Measuring the Gaps

```
In [1]: def eps(x):
            e = float(max(sys.float_info.min, abs(x)))
            while not plus_epsilon_identity(x, e):
                last = e
                e = e / 2.
            return last

In [2]: eps(1.0)
Out[2]: 2.22044604925031e-16

In [3]: eps(1.0e13)
Out[3]: 0.0011102230246251565
```

Measuring the Gaps

```
In [1]: def eps(x):
            e = float(max(sys.float_info.min, abs(x)))
            while not plus_epsilon_identity(x, e):
                last = e
                e = e / 2.
            return last

In [2]: eps(1.0)
Out[2]: 2.22044604925031e-16

In [3]: eps(1.0e13)
Out[3]: 0.0011102230246251565

In [4]: eps(1.0e19)
Out[4]: 1110.2230246251565
```

Choosing Epsilon

Going back to our error estimation,

$$\mathcal{E}(h,x) = \sum_{n=1}^{\infty} \frac{f^{(n)}(x)}{n!} h^{(n-1)}$$

we can add in a term, $R(f,x,h)$ which accounts for the rounding error. The total error then becomes

$$\mathcal{E}(h,x) = \sum_{n=1}^{\infty} \frac{f^{(n)}(x)}{n!} h^{(n-1)} + R(f,x,h)$$

Choosing Epsilon

It turns out that a good estimate [2] is

$$h = \sqrt{u} * max(|x|, 1)$$

where $u = eps(1)$.

[2] http://www.karenkopecky.net/Teaching/eco613614/Notes_NumericalDifferentiation.pdf

Choosing Epsilon

It turns out that a good estimate [2] is

$$h = \sqrt{u} * max(|x|, 1)$$

where $u = eps(1)$.

```
In [1]: def finite_difference(f, x, h=None):
            if not h:
                h = sqrt(eps(1.0)) * max(abs(x), 1.0)
            return (f(x+h) - f(x))/h
```

[2] http://www.karenkopecky.net/Teaching/eco613614/Notes_NumericalDifferentiation.pdf

Testing It Out

```
In [2]: def error(f, df, x):
            return abs(finite_difference(f, x) - df(x))

In [3]: def error_rate(f, df, x):
            return error(f, df, x) / df(x)
```

Testing It Out

```
In [2]: def error(f, df, x):
            return abs(finite_difference(f, x) - df(x))

In [3]: def error_rate(f, df, x):
            return error(f, df, x) / df(x)

In [4]: f, df = lambda x:x**2, lambda x:2*x
```

Testing It Out

```
In [2]: def error(f, df, x):
            return abs(finite_difference(f, x) - df(x))

In [3]: def error_rate(f, df, x):
            return error(f, df, x) / df(x)

In [4]: f, df = lambda x:x**2, lambda x:2*x

In [5]: error_rate(f, df, 1.0)
Out[5]: 7.450580596923828e-09
```

Testing It Out

```
In [2]: def error(f, df, x):
            return abs(finite_difference(f, x) - df(x))

In [3]: def error_rate(f, df, x):
            return error(f, df, x) / df(x)

In [4]: f, df = lambda x:x**2, lambda x:2*x

In [5]: error_rate(f, df, 1.0)
Out[5]: 7.450580596923828e-09

In [6]: error_rate(f, df, 1.0e5)
Out[6]: 9.045761108398438e-09
```

Testing It Out

```
In [2]: def error(f, df, x):
            return abs(finite_difference(f, x) - df(x))

In [3]: def error_rate(f, df, x):
            return error(f, df, x) / df(x)

In [4]: f, df = lambda x:x**2, lambda x:2*x

In [5]: error_rate(f, df, 1.0)
Out[5]: 7.450580596923828e-09

In [6]: error_rate(f, df, 1.0e5)
Out[6]: 9.045761108398438e-09

In [7]: error_rate(f, df, 1.0e20)
Out[7]: 4.5369065472e-09
```

Testing It Out

```
In [8]: import math

In [9]: f, df = math.sin, math.cos
```

Testing It Out

```
In [8]: import math

In [9]: f, df = math.sin, math.cos

In [10]: error_rate(f, df, 1.0)
Out[10]: 1.2780011808656197e-08
```

Testing It Out

```
In [8]: import math

In [9]: f, df = math.sin, math.cos

In [10]: error_rate(f, df, 1.0)
Out[10]: 1.2780011808656197e-08

In [11]: error_rate(f, df, 1.0e10)
Out[11]: 1.002014830004253
```

Testing It Out

```
In [8]: import math

In [9]: f, df = math.sin, math.cos

In [10]: error_rate(f, df, 1.0)
Out[10]: 1.2780011808656197e-08

In [11]: error_rate(f, df, 1.0e10)
Out[11]: 1.002014830004253

In [12]: error_rate(f, df, 1.0e20)
Out[12]: 0.9999999999998509
```

Testing It Out

```
In [8]: import math

In [9]: f, df = math.sin, math.cos

In [10]: error_rate(f, df, 1.0)
Out[10]: 1.2780011808656197e-08

In [11]: error_rate(f, df, 1.0e10)
Out[11]: 1.002014830004253

In [12]: error_rate(f, df, 1.0e20)
Out[12]: 0.9999999999998509
```

We can do better

Testing It Out

```
In [8]: import math

In [9]: f, df = math.sin, math.cos

In [10]: error_rate(f, df, 1.0)
Out[10]: 1.2780011808656197e-08

In [11]: error_rate(f, df, 1.0e10)
Out[11]: 1.002014830004253

In [12]: error_rate(f, df, 1.0e20)
Out[12]: 0.9999999999998509
```

We can do better using Complex Analyis.

A Crash Course in Complex Analysis

- Def: $i = \sqrt{-1}$.

A Crash Course in Complex Analysis

- Def: $i = \sqrt{-1}$.
- Why do we need i???

A Crash Course in Complex Analysis

- Def: $i = \sqrt{-1}$.
- Why do we need i???
- Solve $x^2 + 1 = 0$

A Crash Course in Complex Analysis

- Def: $i = \sqrt{-1}$.
- Why do we need i???
- Solve $x^2 + 1 = 0$
- Def: $\mathbb{C} = \{x + iy \ \forall \ x, y \ \in \mathbb{R}^2\}$

A Crash Course in Complex Analysis

$\mathbb{E}^x$:

$$f(z) = z^2$$

A Crash Course in Complex Analysis

$\mathbb{E}^x$:

$$f(z) = z^2$$

Let $z = x + iy$. Then

$$f(z) = f(x + iy) = (x + iy)^2 = x^2 + 2ixy + i^2y^2 =$$

$$x^2 + 2ixy - y^2$$

A Crash Course in Complex Analysis

$\mathbb{E}^x$:

$$f(z) = z^2$$

Let $z = x + iy$. Then

$$f(z) = f(x + iy) = (x + iy)^2 = x^2 + 2ixy + i^2y^2 =$$

$$x^2 + 2ixy - y^2$$

We define

$$u(x, y) = \Re(f(x + iy)) \text{ and } v(x, y) = \Im(f(x + iy))$$

A Crash Course in Complex Analysis

$\mathbb{E}^x$:

$$f(z) = z^2$$

Let $z = x + iy$. Then

$$f(z) = f(x + iy) = (x + iy)^2 = x^2 + 2ixy + i^2y^2 =$$

$$x^2 + 2ixy - y^2$$

We define

$$u(x, y) = \Re(f(x + iy)) \text{ and } v(x, y) = \Im(f(x + iy))$$

In our example

$$u(x, y) = x^2 - y^2 \text{ and } v(x, y) = 2xy$$

A Crash Course in Complex Analysis

$$u(x, y) = x^2 - y^2 \text{ and } v(x, y) = 2xy$$

We can now take 4 different derivatives.

$$\frac{\partial u}{\partial x} = 2x \qquad \frac{\partial v}{\partial x} = 2y$$
$$\frac{\partial u}{\partial y} = -2y \qquad \frac{\partial v}{\partial y} = 2x$$

A Crash Course in Complex Analysis

$$u(x, y) = x^2 - y^2 \text{ and } v(x, y) = 2xy$$

We can now take 4 different derivatives.

$$\frac{\partial u}{\partial x} = 2x \qquad \frac{\partial v}{\partial x} = 2y$$
$$\frac{\partial u}{\partial y} = -2y \qquad \frac{\partial v}{\partial y} = 2x$$

Note that:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \qquad \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}$$

Cauchy-Riemann Equations

These are the Cauchy-Riemann equations and hold for all analytic functions on $\mathbb{C}$! [3]

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \qquad \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}$$

[3] See your favorite Complex Analysis textbook or Dr. Casey for a proof.

Time for the Hack!

Given $f : \mathbb{C} \to \mathbb{C}$ such that $f : \mathbb{R} \to X \subseteq \mathbb{R}$, we can rewrite

$$f(z) = f(x + iy) = u(x, y) + iv(x, y)$$

Time for the Hack!

Given $f : \mathbb{C} \to \mathbb{C}$ such that $f : \mathbb{R} \to X \subseteq \mathbb{R}$, we can rewrite

$$f(z) = f(x + iy) = u(x, y) + iv(x, y)$$

Now, for all $\bar{z} \in \mathbb{R}$, we have $y = 0$, and so

$$f(\bar{z}) = f(x, 0) = u(x, 0) + iv(x, 0)$$

Time for the Hack!

Since $f : \mathbb{R} \to X \subseteq \mathbb{R}$, $f(\bar{z}) \in \mathbb{R}$ for all $\bar{z} \in \mathbb{R}$. Therefore

$$v(x, 0) = 0$$

Time for the Hack!

Since $f : \mathbb{R} \to X \subseteq \mathbb{R}$, $f(\bar{z}) \in \mathbb{R}$ for all $\bar{z} \in \mathbb{R}$. Therefore

$$v(x, 0) = 0$$

and so

$$f(\bar{z}) = f(x, 0) = u(x, 0) + iv(x, 0) = u(x, 0)$$

Time for the Hack!

Since $f : \mathbb{R} \to X \subseteq \mathbb{R}$, $f(\bar{z}) \in \mathbb{R}$ for all $\bar{z} \in \mathbb{R}$. Therefore

$$v(x, 0) = 0$$

and so

$$f(\bar{z}) = f(x, 0) = u(x, 0) + iv(x, 0) = u(x, 0)$$

We want to estimate $\frac{df}{\partial x}$. Since for all $z \in \mathbb{R}$, $f = u$,

$$\frac{df}{\partial x} = \frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}$$

by the Cauchy-Riemann equations.

Time for the Hack!

Again, using the definition of the derivative

$$\frac{\partial v}{\partial y} = \lim_{h \to 0} \frac{v(x, y + h) - v(x, y)}{h}$$

Time for the Hack!

Again, using the definition of the derivative

$$\frac{\partial v}{\partial y} = \lim_{h \to 0} \frac{v(x, y+h) - v(x, y)}{h}$$

Since $y = 0$ for all $\bar{z} \in \mathbb{R}$,

$$\frac{\partial v}{\partial y} = \lim_{h \to 0} \frac{v(x, h) - v(x, 0)}{h}$$

Time for the Hack!

Recall that $v(x, 0) = 0$, so

$$\frac{\partial v}{\partial y} = \lim_{h \to 0} \frac{v(x, h)}{h}$$

Time for the Hack!

Recall that $v(x, 0) = 0$, so

$$\frac{\partial v}{\partial y} = \lim_{h \to 0} \frac{v(x, h)}{h}$$

Now, to estimate $\frac{df}{\partial x} = \frac{\partial v}{\partial y}$, for a very small h

$$\frac{df}{\partial x} \approx \frac{v(x, h)}{h} = \frac{\Im(f(x + ih))}{h}$$

Time for the Hack!

We've now replaced

$$\frac{f(x+h)-f(x)}{h}$$

with

$$\frac{\Im(f(x+ih))}{h}.$$

Note that we have eliminated the floating point addition!

Complex Step Finite Difference

Implemented in Python

```
In [1]: import sys

In [2]: def complex_step_finite_diff(f, x):
            h = sys.float_info.min
            return (f(x+h*1.0j)).imag / h
```

Complex Step Finite Difference

Implemented in Python

```
In [3]: f, df = lambda x:x**2, lambda x:2*x
```

Complex Step Finite Difference

Implemented in Python

```
In [3]: f, df = lambda x:x**2, lambda x:2*x

In [4]: complex_step_finite_diff(f, 1.0)
Out[4]: 2.0
```

Complex Step Finite Difference

Implemented in Python

```
In [3]: f, df = lambda x:x**2, lambda x:2*x

In [4]: complex_step_finite_diff(f, 1.0)
Out[4]: 2.0

In [5]: complex_step_finite_diff(f, 1.0e10)
Out[5]: 20000000000.0
```

Complex Step Finite Difference

Implemented in Python

```
In [3]: f, df = lambda x:x**2, lambda x:2*x

In [4]: complex_step_finite_diff(f, 1.0)
Out[4]: 2.0

In [5]: complex_step_finite_diff(f, 1.0e10)
Out[5]: 20000000000.0

In [6]: complex_step_finite_diff(f, 1.0e20)
Out[6]: 2e+20
```

Testing It Out

```
In [7]: def cerror(f, df, x):
            return abs(complex_step_finite_diff(f, x) -
                df(x))

In [8]: def cerror_rate(f, df, x):
            return cerror(f, df, x) / x
```

Testing It Out

```
In [7]: def cerror(f, df, x):
            return abs(complex_step_finite_diff(f, x) -
                df(x))

In [8]: def cerror_rate(f, df, x):
            return cerror(f, df, x) / x

In [9]: cerror_rate(f, df, 1.0)
Out[9]: 0.0
```

Testing It Out

```
In [7]: def cerror(f, df, x):
            return abs(complex_step_finite_diff(f, x) -
                df(x))

In [8]: def cerror_rate(f, df, x):
            return cerror(f, df, x) / x

In [9]: cerror_rate(f, df, 1.0)
Out[9]: 0.0
In [10]: cerror_rate(f, df, 1.0e10)
Out[10]: 0.0
```

Testing It Out

```
In [7]: def cerror(f, df, x):
            return abs(complex_step_finite_diff(f, x) -
                df(x))

In [8]: def cerror_rate(f, df, x):
            return cerror(f, df, x) / x

In [9]: cerror_rate(f, df, 1.0)
Out[9]: 0.0
In [10]: cerror_rate(f, df, 1.0e10)
Out[10]: 0.0
In [11]: cerror_rate(f, df, 1.0e20)
Out[11]: 0.0
```

Testing It Out

```
In [7]: def cerror(f, df, x):
            return abs(complex_step_finite_diff(f, x) -
                df(x))

In [8]: def cerror_rate(f, df, x):
            return cerror(f, df, x) / x

In [9]: cerror_rate(f, df, 1.0)
Out[9]: 0.0
In [10]: cerror_rate(f, df, 1.0e10)
Out[10]: 0.0
In [11]: cerror_rate(f, df, 1.0e20)
Out[11]: 0.0
```

Testing It Out

```
In [11]: import cmath

In [11]: f, df = cmath.sin, cmath.cos
```

Testing It Out

```
In [11]: import cmath

In [11]: f, df = cmath.sin, cmath.cos

In [12]: cerror_rate(f, df, 1.0)
Out[12]: 0.0
```

Testing It Out

```
In [11]: import cmath

In [11]: f, df = cmath.sin, cmath.cos

In [12]: cerror_rate(f, df, 1.0)
Out[12]: 0.0

In [13]: cerror_rate(f, df, 1.0e10)
Out[13]: 0.0
```

Testing It Out

```
In [11]: import cmath

In [11]: f, df = cmath.sin, cmath.cos

In [12]: cerror_rate(f, df, 1.0)
Out[12]: 0.0

In [13]: cerror_rate(f, df, 1.0e10)
Out[13]: 0.0

In [14]: cerror_rate(f, df, 1.0e20)
Out[14]: 0.0
```

It's Not Perfect, But It's Pretty Good

```
In [15]: cerror_rate(f, df, 1.0e5)
Out[15]: .110933124815182e-16
```

It's Not Perfect, But It's Pretty Good

```
In [15]: cerror_rate(f, df, 1.0e5)
Out[15]: .110933124815182e-16

In [16]: f = lambda x: cmath.exp(x) / cmath.sqrt(x)
In [17]: df = lambda x: (cmath.exp(x)* (2*x -
    1))/(2*x**(1.5))
```

It's Not Perfect, But It's Pretty Good

```
In [15]: cerror_rate(f, df, 1.0e5)
Out[15]: .110933124815182e-16

In [16]: f = lambda x: cmath.exp(x) / cmath.sqrt(x)
In [17]: df = lambda x: (cmath.exp(x)* (2*x -
    1))/(2*x**(1.5))

In [18]: cerror_rate(f, df, 1.0)
Out[18]: 0.0
```

It's Not Perfect, But It's Pretty Good

```
In [15]: cerror_rate(f, df, 1.0e5)
Out[15]: .110933124815182e-16

In [16]: f = lambda x: cmath.exp(x) / cmath.sqrt(x)
In [17]: df = lambda x: (cmath.exp(x)* (2*x -
    1))/(2*x**(1.5))

In [18]: cerror_rate(f, df, 1.0)
Out[18]: 0.0
In [19]: cerror_rate(f, df, 1.0e2)
Out[19]: 1.1570932160552342e-16
```

It's Not Perfect, But It's Pretty Good

```
In [15]: cerror_rate(f, df, 1.0e5)
Out[15]: .110933124815182e-16

In [16]: f = lambda x: cmath.exp(x) / cmath.sqrt(x)
In [17]: df = lambda x: (cmath.exp(x)* (2*x -
    1))/(2*x**(1.5))

In [18]: cerror_rate(f, df, 1.0)
Out[18]: 0.0
In [19]: cerror_rate(f, df, 1.0e2)
Out[19]: 1.1570932160552342e-16
In [20]: cerror_rate(f, df, 5.67)
Out[20]: 1.2795419601231268e-16
```

It's Not Perfect, But It's Pretty Good

```
In [15]: cerror_rate(f, df, 1.0e5)
Out[15]: .110933124815182e-16

In [16]: f = lambda x: cmath.exp(x) / cmath.sqrt(x)
In [17]: df = lambda x: (cmath.exp(x)* (2*x -
    1))/(2*x**(1.5))

In [18]: cerror_rate(f, df, 1.0)
Out[18]: 0.0
In [19]: cerror_rate(f, df, 1.0e2)
Out[19]: 1.1570932160552342e-16
In [20]: cerror_rate(f, df, 5.67)
Out[20]: 1.2795419601231268e-16
In [21]: cerror_rate(f, df, 1.0e3)
Out [21]: OverflowError: math range error
```

Limits

- Must satisfy Cauchy Riemann equations, so does not work for abs or a function with $<$ or $>$ in it.
- Only works for functions from $\mathbb{R} \to X \subseteq \mathbb{R}$.

Other Approaches

Automatic Differentiation

- All computer programs perform a sequence of elementary arithmetic
- Using the chain rule repeatedly, derivative of arbitrary order can be computed

Thanks!

- Follow me on twitter: @taubeneck
- Fork me on Github: eriktaubeneck
- Blog: `http://skien.cc/`
- Slides: `http://slides.skien.cc/hack-the-derivative-au.pdf`
- Long Form Article in Code Words: `https://codewords.recurse.com/issues/four/hack-the-derivative`
- Other Presentations: `http://slides.skienc.cc`