Automatic differentiation with JAX

Hans Dembinski

TU Dortmund, Germany

What is JAX

- For vector-valued functions expressible in NumPy code ($\mathcal{R}^m \to \mathcal{R}^n$)

What is JAX

- For vector-valued functions expressible in NumPy code ($\mathcal{R}^m \to \mathcal{R}^n$)
    - Compute derivatives with automatic differentiation (gradient, Jacobian, Hessian, ...)

What is JAX

- For vector-valued functions expressible in NumPy code ($\mathcal{R}^m \to \mathcal{R}^n$)
  - Compute derivatives with automatic differentiation (gradient, Jacobian, Hessian, ...)
  - JIT compile with XLA (Accelerated Linear Algebra) into optimised code for CPU,GPU,TPU

What is JAX

- For vector-valued functions expressible in NumPy code ($\mathcal{R}^m \to \mathcal{R}^n$)
  - Compute derivatives with automatic differentiation (gradient, Jacobian, Hessian, ...)
  - JIT compile with XLA (Accelerated Linear Algebra) into optimised code for CPU,GPU,TPU
    - XLA also used by TensorFlow, Julia, PyTorch, Nx

What is JAX

- For vector-valued functions expressible in NumPy code ($\mathcal{R}^m \to \mathcal{R}^n$)
  - Compute derivatives with automatic differentiation (gradient, Jacobian, Hessian, . . .)
  - JIT compile with XLA (Accelerated Linear Algebra) into optimised code for CPU,GPU,TPU
    - XLA also used by TensorFlow, Julia, PyTorch, Nx
    - XLA compiler uses LLVM to generate low-level code for CPU,GPU

What is JAX

- For vector-valued functions expressible in NumPy code ($\mathcal{R}^m \to \mathcal{R}^n$)
  - Compute derivatives with automatic differentiation (gradient, Jacobian, Hessian, ...)
  - JIT compile with XLA (Accelerated Linear Algebra) into optimised code for CPU,GPU,TPU
    - XLA also used by TensorFlow, Julia, PyTorch, Nx
    - XLA compiler uses LLVM to generate low-level code for CPU,GPU
  - Vectorise code

What is JAX

- For vector-valued functions expressible in NumPy code ($\mathcal{R}^m \to \mathcal{R}^n$)
  - Compute derivatives with automatic differentiation (gradient, Jacobian, Hessian, . . .)
  - JIT compile with XLA (Accelerated Linear Algebra) into optimised code for CPU,GPU,TPU
    - XLA also used by TensorFlow, Julia, PyTorch, Nx
    - XLA compiler uses LLVM to generate low-level code for CPU,GPU
  - Vectorise code
- Successor of autograd

What is JAX

- For vector-valued functions expressible in NumPy code ($\mathcal{R}^m \to \mathcal{R}^n$)
  - Compute derivatives with automatic differentiation (gradient, Jacobian, Hessian, . . .)
  - JIT compile with XLA (Accelerated Linear Algebra) into optimised code for CPU,GPU,TPU
    - XLA also used by TensorFlow, Julia, PyTorch, Nx
    - XLA compiler uses LLVM to generate low-level code for CPU,GPU
  - Vectorise code
- Successor of autograd
- Mirrors NumPy API

What is JAX

- For vector-valued functions expressible in NumPy code ($\mathcal{R}^m \to \mathcal{R}^n$)
  - Compute derivatives with automatic differentiation (gradient, Jacobian, Hessian, ...)
  - JIT compile with XLA (Accelerated Linear Algebra) into optimised code for CPU,GPU,TPU
    - XLA also used by TensorFlow, Julia, PyTorch, Nx
    - XLA compiler uses LLVM to generate low-level code for CPU,GPU
  - Vectorise code
- Successor of autograd
- Mirrors NumPy API
- Powered by Google

Why JAX?

- Easy to learn if you already know NumPy

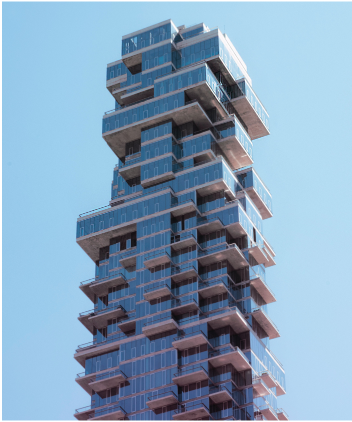

Photo by Ena Begcevic on Unsplash

Why JAX?

- Easy to learn if you already know NumPy
- Laser-focussed on one task

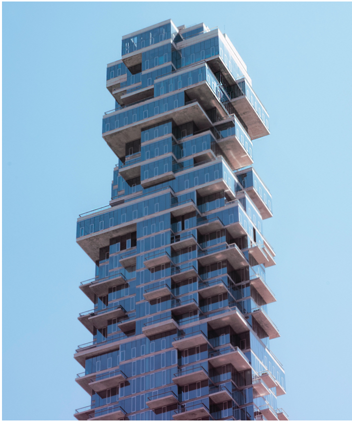

Photo by Ena Begcevic on Unsplash

Why JAX?

- Easy to learn if you already know NumPy
- Laser-focussed on one task
- Great low-level building block to build your own custom solutions

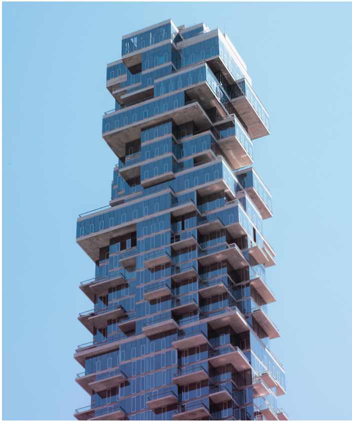

Photo by Ena Begcevic on Unsplash

Why JAX?

- Easy to learn if you already know NumPy
- Laser-focussed on one task
- Great low-level building block to build your own custom solutions
- Supported and further developed by Google

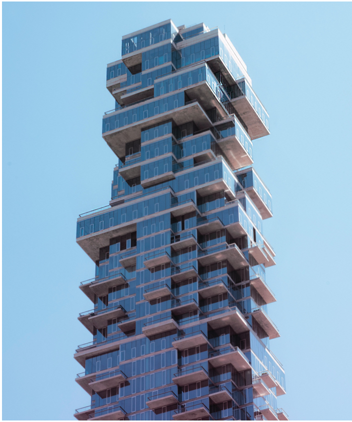

Photo by Ena Begcevic on Unsplash

Why JAX?

- Easy to learn if you already know NumPy
- Laser-focussed on one task
- Great low-level building block to build your own custom solutions
- Supported and further developed by Google
- Replacement for Numba? Not yet... (will come back to that)

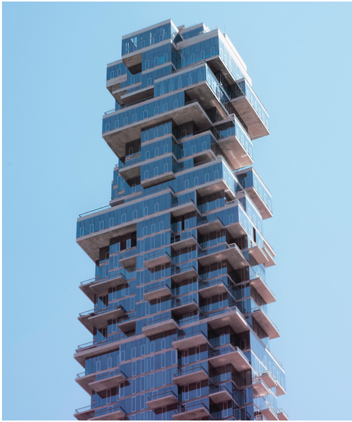

Photo by Ena Begcevic on Unsplash

Source material and further reading

- Boris Settinger's gentle introduction to automatic differentiation

Source material and further reading

- Boris Settinger's gentle introduction to automatic differentiation
  - Great Jupyter notebook with animations, references to prior work (go read this)

Source material and further reading

- Boris Settinger's gentle introduction to automatic differentiation
  - Great Jupyter notebook with animations, references to prior work (go read this)
- Baydin et al., *Automatic differentiation in machine learning: a survey*, Journal of Machine Learning Research, 18(153):1–43, 2018

Source material and further reading

- Boris Settinger's gentle introduction to automatic differentiation
  - Great Jupyter notebook with animations, references to prior work (go read this)
- Baydin et al., *Automatic differentiation in machine learning: a survey*, Journal of Machine Learning Research, 18(153):1–43, 2018
- JAX's beginner- and poweruser-friendly documentation

Derivatives in statistics are everywhere

- Minimisation with Newton-Raphson method (e.g. Minuit's Migrad algorithm), in 1-dim:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

Derivatives in statistics are everywhere

- Minimisation with Newton-Raphson method (e.g. Minuit's Migrad algorithm), in 1-dim:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

- Error propagation for vector-valued function $\vec{y} = f(\vec{x})$ that maps from $\mathcal{R}^m \to \mathcal{R}^n$

$$C_y = J \, C_x \, J^T$$

  with $J_{ij} = \partial y_i / \partial x_j$

Three ways to compute a derivative

- $\mathcal{R} \to \mathcal{R}$

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

Three ways to compute a derivative

- $\mathcal{R} \to \mathcal{R}$

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

- $\mathcal{R}^m \to \mathcal{R}^n$:

$$\frac{\partial \vec{y}}{\partial x_j} = \lim_{h \to 0} \frac{f(\vec{x} + \vec{e}_j h) - f(\vec{x})}{h}$$

Three ways to compute a derivative

- $\mathcal{R} \to \mathcal{R}$

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

- $\mathcal{R}^m \to \mathcal{R}^n$:

$$\frac{\partial \vec{y}}{\partial x_j} = \lim_{h \to 0} \frac{f(\vec{x} + \vec{e}_j h) - f(\vec{x})}{h}$$

- Three ways to compute derivatives on a computer

Three ways to compute a derivative

- $\mathcal{R} \to \mathcal{R}$

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

- $\mathcal{R}^m \to \mathcal{R}^n$:

$$\frac{\partial \vec{y}}{\partial x_j} = \lim_{h \to 0} \frac{f(\vec{x} + \vec{e}_j h) - f(\vec{x})}{h}$$

- Three ways to compute derivatives on a computer
  - Numerical differentiation

Three ways to compute a derivative

- $\mathcal{R} \to \mathcal{R}$

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

- $\mathcal{R}^m \to \mathcal{R}^n$:

$$\frac{\partial \vec{y}}{\partial x_j} = \lim_{h \to 0} \frac{f(\vec{x} + \vec{e}_j h) - f(\vec{x})}{h}$$

- Three ways to compute derivatives on a computer
  - Numerical differentiation
  - Symbolic differentiation

Three ways to compute a derivative

- $\mathcal{R} \to \mathcal{R}$

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

- $\mathcal{R}^m \to \mathcal{R}^n$:

$$\frac{\partial \vec{y}}{\partial x_j} = \lim_{h \to 0} \frac{f(\vec{x} + \vec{e}_j h) - f(\vec{x})}{h}$$

- Three ways to compute derivatives on a computer
  - Numerical differentiation
  - Symbolic differentiation
  - Automatic differentiation

Numerical differentiation

$$\frac{\partial y}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \approx \frac{f(x+h) - f(x)}{h} + O(h^2)$$

(It is almost always better to use the symmetric version $\frac{\partial y}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$)

- If you choose h small enough, then error is negligible

Numerical differentiation

$$\frac{\partial y}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \approx \frac{f(x+h) - f(x)}{h} + O(h^2)$$

(It is almost always better to use the symmetric version $\frac{\partial y}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$)

- If you choose h small enough, then error is negligible
- Choose h too small and you get numerical issues `1e16 - 1 == 1e16`

Numerical differentiation

$$\frac{\partial y}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \approx \frac{f(x+h) - f(x)}{h} + O(h^2)$$

(It is almost always better to use the symmetric version $\frac{\partial y}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$)

- If you choose h small enough, then error is negligible
- Choose h too small and you get numerical issues `1e16 - 1 == 1e16`
- **Pro**

Numerical differentiation

$$\frac{\partial y}{\partial x} = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h} \approx \frac{f(x+h) - f(x)}{h} + O(h^2)$$

(It is almost always better to use the symmetric version $\frac{\partial y}{\partial x} \approx \frac{f(x+h)-f(x-h)}{2h}$)

- If you choose h small enough, then error is negligible
- Choose h too small and you get numerical issues `1e16 - 1 == 1e16`
- **Pro**
  - Works on any $f(\vec{x})$, even if $f(\vec{x})$ calls into foreign (C++) code

Numerical differentiation

$$\frac{\partial y}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \approx \frac{f(x+h) - f(x)}{h} + O(h^2)$$

(It is almost always better to use the symmetric version $\frac{\partial y}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$)

- If you choose h small enough, then error is negligible
- Choose h too small and you get numerical issues `1e16 - 1 == 1e16`
- **Pro**
  - Works on any $f(\vec{x})$, even if $f(\vec{x})$ calls into foreign (C++) code
  - *Pro-ish*: Requires $2 \times m$ function evaluations for $\mathcal{R}^m \to \mathcal{R}^n$

Numerical differentiation

$$\frac{\partial y}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \approx \frac{f(x+h) - f(x)}{h} + O(h^2)$$

(It is almost always better to use the symmetric version $\frac{\partial y}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$)

- If you choose h small enough, then error is negligible
- Choose h too small and you get numerical issues `1e16 - 1 == 1e16`
- **Pro**
  - Works on any $f(\vec{x})$, even if $f(\vec{x})$ calls into foreign (C++) code
  - *Pro-ish*: Requires $2 \times m$ function evaluations for $\mathcal{R}^m \to \mathcal{R}^n$
- **Contra**

Numerical differentiation

$$\frac{\partial y}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \approx \frac{f(x+h) - f(x)}{h} + O(h^2)$$

(It is almost always better to use the symmetric version $\frac{\partial y}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$)

- If you choose h small enough, then error is negligible
- Choose h too small and you get numerical issues `1e16 - 1 == 1e16`
- **Pro**
  - Works on any $f(\vec{x})$, even if $f(\vec{x})$ calls into foreign (C++) code
  - *Pro-ish*: Requires $2 \times m$ function evaluations for $\mathcal{R}^m \to \mathcal{R}^n$
- **Contra**
  - Difficult to chose h optimally/safely when nothing is known about $f(\vec{x})$

Numerical differentiation

$$\frac{\partial y}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \approx \frac{f(x+h) - f(x)}{h} + O(h^2)$$

(It is almost always better to use the symmetric version $\frac{\partial y}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$)

- If you choose h small enough, then error is negligible
- Choose h too small and you get numerical issues `1e16 - 1 == 1e16`
- **Pro**
  - Works on any $f(\vec{x})$, even if $f(\vec{x})$ calls into foreign (C++) code
  - *Pro-ish*: Requires $2 \times m$ function evaluations for $\mathcal{R}^m \to \mathcal{R}^n$
- **Contra**
  - Difficult to chose h optimally/safely when nothing is known about $f(\vec{x})$
  - Lots of research on doing this, but it remains complicated and fragile

Numerical differentiation

$$\frac{\partial y}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \approx \frac{f(x+h) - f(x)}{h} + O(h^2)$$

(It is almost always better to use the symmetric version $\frac{\partial y}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$)

- If you choose h small enough, then error is negligible
- Choose h too small and you get numerical issues `1e16 - 1 == 1e16`
- **Pro**
  - Works on any $f(\vec{x})$, even if $f(\vec{x})$ calls into foreign (C++) code
  - *Pro-ish*: Requires $2 \times m$ function evaluations for $\mathcal{R}^m \to \mathcal{R}^n$
- **Contra**
  - Difficult to chose h optimally/safely when nothing is known about $f(\vec{x})$
  - Lots of research on doing this, but it remains complicated and fragile
  - Accuracy well below machine precision

Symbolic differentiation

- Calculating derivatives is mechanical, we know all rules
  $(f(x) + g(x))' = f'(x) + g'(x),\ (x^a)' = a\,x^{a-1},\ (\sin(x))' = \cos(x),\ \ldots$

Symbolic differentiation

- Calculating derivatives is mechanical, we know all rules
  $(f(x) + g(x))' = f'(x) + g'(x),\ (x^a)' = a\, x^{a-1},\ (\sin(x))' = \cos(x),\ \ldots$
- Computer algebra systems (CAS) can apply these rules, e.g. SymPy

```
from sympy import diff
from sympy.abc import x
diff(3 * x ** 2 + 2 * x, x) # returns 6*x + 2
```

Symbolic differentiation

- Calculating derivatives is mechanical, we know all rules
  $(f(x)+g(x))' = f'(x)+g'(x)$, $(x^a)' = a\,x^{a-1}$, $(\sin(x))' = \cos(x)$, $\ldots$
- Computer algebra systems (CAS) can apply these rules, e.g. SymPy

```
from sympy import diff
from sympy.abc import x
diff(3 * x ** 2 + 2 * x, x) # returns 6*x + 2
```

- **Pro**

Symbolic differentiation

- Calculating derivatives is mechanical, we know all rules
  $(f(x)+g(x))' = f'(x)+g'(x),\ (x^a)' = a\,x^{a-1},\ (\sin(x))' = \cos(x),\ \ldots$
- Computer algebra systems (CAS) can apply these rules, e.g. SymPy

```
from sympy import diff
from sympy.abc import x
diff(3 * x ** 2 + 2 * x, x) # returns 6*x + 2
```

- **Pro**
  - Great if you need to know formula, CAS can simplify expression

Symbolic differentiation

- Calculating derivatives is mechanical, we know all rules
  $(f(x)+g(x))' = f'(x)+g'(x)$, $(x^a)' = a\,x^{a-1}$, $(\sin(x))' = \cos(x)$, ...
- Computer algebra systems (CAS) can apply these rules, e.g. SymPy

```
from sympy import diff
from sympy.abc import x
diff(3 * x ** 2 + 2 * x, x) # returns 6*x + 2
```

- **Pro**
  - Great if you need to know formula, CAS can simplify expression
  - Compute once (slow), evaluate many times (fast)

Symbolic differentiation

- Calculating derivatives is mechanical, we know all rules
  $(f(x) + g(x))' = f'(x) + g'(x),\ (x^a)' = a\,x^{a-1},\ (\sin(x))' = \cos(x),\ \ldots$
- Computer algebra systems (CAS) can apply these rules, e.g. SymPy

```
from sympy import diff
from sympy.abc import x
diff(3 * x ** 2 + 2 * x, x) # returns 6*x + 2
```

- **Pro**
  - Great if you need to know formula, CAS can simplify expression
  - Compute once (slow), evaluate many times (fast)
  - Exact within machine precision

Symbolic differentiation

- Calculating derivatives is mechanical, we know all rules
  $(f(x) + g(x))' = f'(x) + g'(x)$, $(x^a)' = a\,x^{a-1}$, $(\sin(x))' = \cos(x)$, $\ldots$
- Computer algebra systems (CAS) can apply these rules, e.g. SymPy

```
from sympy import diff
from sympy.abc import x
diff(3 * x ** 2 + 2 * x, x) # returns 6*x + 2
```

- **Pro**
  - Great if you need to know formula, CAS can simplify expression
  - Compute once (slow), evaluate many times (fast)
  - Exact within machine precision
- **Contra**

Symbolic differentiation

- Calculating derivatives is mechanical, we know all rules
  $(f(x)+g(x))' = f'(x)+g'(x)$, $(x^a)' = a\,x^{a-1}$, $(\sin(x))' = \cos(x)$, $\ldots$
- Computer algebra systems (CAS) can apply these rules, e.g. SymPy

```
from sympy import diff
from sympy.abc import x
diff(3 * x ** 2 + 2 * x, x) # returns 6*x + 2
```

- **Pro**
  - Great if you need to know formula, CAS can simplify expression
  - Compute once (slow), evaluate many times (fast)
  - Exact within machine precision
- **Contra**
  - Only works if $f(\vec{x})$ is analytical and symbolically representable

Symbolic differentiation

- Calculating derivatives is mechanical, we know all rules
  $(f(x)+g(x))' = f'(x)+g'(x)$, $(x^a)' = a\,x^{a-1}$, $(\sin(x))' = \cos(x)$, ...
- Computer algebra systems (CAS) can apply these rules, e.g. SymPy

```
from sympy import diff
from sympy.abc import x
diff(3 * x ** 2 + 2 * x, x) # returns 6*x + 2
```

- **Pro**
  - Great if you need to know formula, CAS can simplify expression
  - Compute once (slow), evaluate many times (fast)
  - Exact within machine precision
- **Contra**
  - Only works if $f(\vec{x})$ is analytical and symbolically representable
  - Does not work if function does computer things, `if`, `while`, ...

Automatic differentiation

- Alternative approach to derivatives using hyperreal numbers

Automatic differentiation

- Alternative approach to derivatives using hyperreal numbers
  - Extension of real numbers to include infinite and **infinitesimal** quantities

Automatic differentiation

- Alternative approach to derivatives using hyperreal numbers
  - Extension of real numbers to include infinite and **infinitesimal** quantities
  - That sounds cool (or scary) but is actually very simple

Automatic differentiation

- Alternative approach to derivatives using hyperreal numbers
  - Extension of real numbers to include infinite and **infinitesimal** quantities
  - That sounds cool (or scary) but is actually very simple
  - Physicists have intuitively used that for a long time

Automatic differentiation

- Alternative approach to derivatives using hyperreal numbers
  - Extension of real numbers to include infinite and **infinitesimal** quantities
  - That sounds cool (or scary) but is actually very simple
  - Physicists have intuitively used that for a long time
  - Can be easily implemented on computer

Automatic differentiation

- Alternative approach to derivatives using hyperreal numbers
  - Extension of real numbers to include infinite and **infinitesimal** quantities
  - That sounds cool (or scary) but is actually very simple
  - Physicists have intuitively used that for a long time
  - Can be easily implemented on computer
- **Pro**

Automatic differentiation

- Alternative approach to derivatives using hyperreal numbers
  - Extension of real numbers to include infinite and **infinitesimal** quantities
  - That sounds cool (or scary) but is actually very simple
  - Physicists have intuitively used that for a long time
  - Can be easily implemented on computer
- **Pro**
  - Works (in principle) for any pure function one can write in code

Automatic differentiation

- Alternative approach to derivatives using hyperreal numbers
  - Extension of real numbers to include infinite and **infinitesimal** quantities
  - That sounds cool (or scary) but is actually very simple
  - Physicists have intuitively used that for a long time
  - Can be easily implemented on computer
- **Pro**
  - Works (in principle) for any pure function one can write in code
  - Exact within machine precision

Automatic differentiation

- Alternative approach to derivatives using hyperreal numbers
  - Extension of real numbers to include infinite and **infinitesimal** quantities
  - That sounds cool (or scary) but is actually very simple
  - Physicists have intuitively used that for a long time
  - Can be easily implemented on computer
- **Pro**
  - Works (in principle) for any pure function one can write in code
  - Exact within machine precision
  - About same cost as (lowest order) numerical differentiation

Automatic differentiation

- Alternative approach to derivatives using hyperreal numbers
  - Extension of real numbers to include infinite and **infinitesimal** quantities
  - That sounds cool (or scary) but is actually very simple
  - Physicists have intuitively used that for a long time
  - Can be easily implemented on computer
- **Pro**
  - Works (in principle) for any pure function one can write in code
  - Exact within machine precision
  - About same cost as (lowest order) numerical differentiation
- **Contra**

Automatic differentiation

- Alternative approach to derivatives using hyperreal numbers
  - Extension of real numbers to include infinite and **infinitesimal** quantities
  - That sounds cool (or scary) but is actually very simple
  - Physicists have intuitively used that for a long time
  - Can be easily implemented on computer
- **Pro**
  - Works (in principle) for any pure function one can write in code
  - Exact within machine precision
  - About same cost as (lowest order) numerical differentiation
- **Contra**
  - Fails if $f(\vec{x})$ calls into foreign (C++) code

Motivation of hyperreal numbers

- Start with a Taylor series

$$f(x+\epsilon) = f(x) + f'(x)\epsilon + \frac{1}{2}f''(x)\epsilon^2 + \ldots$$

Motivation of hyperreal numbers

- Start with a Taylor series

$$f(x+\epsilon) = f(x) + f'(x)\epsilon + \frac{1}{2}f''(x)\epsilon^2 + \dots$$

- Now define $\epsilon^2 = 0$, this truncates series to exact result

$$f(x+\epsilon) = f(x) + f'(x)\epsilon$$

Motivation of hyperreal numbers

- Start with a Taylor series

$$f(x+\epsilon) = f(x) + f'(x)\epsilon + \frac{1}{2}f''(x)\epsilon^2 + \ldots$$

- Now define $\epsilon^2 = 0$, this truncates series to exact result

$$f(x+\epsilon) = f(x) + f'(x)\epsilon$$

- Note that you can *read off* derivative as factor in front of ϵ

Motivation of hyperreal numbers

- Start with a Taylor series

$$f(x+\epsilon) = f(x) + f'(x)\epsilon + \frac{1}{2}f''(x)\epsilon^2 + \dots$$

- Now define $\epsilon^2 = 0$, this truncates series to exact result

$$f(x+\epsilon) = f(x) + f'(x)\epsilon$$

- Note that you can *read off* derivative as factor in front of ϵ
- Turns out one can use this to algebraically compute derivatives

Hyperreal numbers

- Think of hyperreal number $x + y\epsilon$ analog to complex number $x + yi$

Hyperreal numbers

- Think of hyperreal number $x + y\epsilon$ analog to complex number $x + yi$
  - $\epsilon^2 = 0$ instead of $i^2 = -1$

Hyperreal numbers

- Think of hyperreal number $x + y\epsilon$ analog to complex number $x + yi$
  - $\epsilon^2 = 0$ instead of $i^2 = -1$
  - $\mathrm{st}(\dots)$ extracts real number, analog to $\mathrm{Re}(\dots)$ for complex number

Hyperreal numbers

- Think of hyperreal number $x + y\epsilon$ analog to complex number $x + yi$
  - $\epsilon^2 = 0$ instead of $i^2 = -1$
  - $\mathrm{st}(\ldots)$ extracts real number, analog to $\mathrm{Re}(\ldots)$ for complex number
- Laws for addition, multiplication, inverse, power can be easily derived

Hyperreal numbers

- Think of hyperreal number $x + y\epsilon$ analog to complex number $x + yi$
  - $\epsilon^2 = 0$ instead of $i^2 = -1$
  - $\mathrm{st}(\ldots)$ extracts real number, analog to $\mathrm{Re}(\ldots)$ for complex number
- Laws for addition, multiplication, inverse, power can be easily derived
  - $(x + a\epsilon) + (y + b\epsilon) = (x + y) + (a + b)\epsilon$

Hyperreal numbers

- Think of hyperreal number $x + y\epsilon$ analog to complex number $x + yi$
  - $\epsilon^2 = 0$ instead of $i^2 = -1$
  - $\mathrm{st}(\dots)$ extracts real number, analog to $\mathrm{Re}(\dots)$ for complex number
- Laws for addition, multiplication, inverse, power can be easily derived
  - $(x + a\epsilon) + (y + b\epsilon) = (x + y) + (a + b)\epsilon$
  - $(x + a\epsilon)(y + b\epsilon) = xy + (xb + ya)\epsilon$

Hyperreal numbers

- Think of hyperreal number $x + y\epsilon$ analog to complex number $x + yi$
  - $\epsilon^2 = 0$ instead of $i^2 = -1$
  - $\mathrm{st}(\dots)$ extracts real number, analog to $\mathrm{Re}(\dots)$ for complex number
- Laws for addition, multiplication, inverse, power can be easily derived
  - $(x + a\epsilon) + (y + b\epsilon) = (x + y) + (a + b)\epsilon$
  - $(x + a\epsilon)(y + b\epsilon) = xy + (xb + ya)\epsilon$
  - $\dfrac{1}{x + a\epsilon} = \dfrac{x - a\epsilon}{(x + a\epsilon)(x - a\epsilon)} = \dfrac{x - a\epsilon}{x^2} = \dfrac{1}{x} - \dfrac{a}{x^2}\epsilon$

Hyperreal numbers

- Think of hyperreal number $x + y\epsilon$ analog to complex number $x + yi$
  - $\epsilon^2 = 0$ instead of $i^2 = -1$
  - $\mathrm{st}(\dots)$ extracts real number, analog to $\mathrm{Re}(\dots)$ for complex number
- Laws for addition, multiplication, inverse, power can be easily derived
  - $(x + a\epsilon) + (y + b\epsilon) = (x + y) + (a + b)\epsilon$
  - $(x + a\epsilon)(y + b\epsilon) = xy + (xb + ya)\epsilon$
  - $\dfrac{1}{x + a\epsilon} = \dfrac{x - a\epsilon}{(x + a\epsilon)(x - a\epsilon)} = \dfrac{x - a\epsilon}{x^2} = \dfrac{1}{x} - \dfrac{a}{x^2}\epsilon$
  - $(x + a\epsilon)^n = (x + a\epsilon)(x + a\epsilon)\cdots = x^n + nx^{n-1}a\epsilon$

Hyperreal numbers

- Think of hyperreal number $x + y\epsilon$ analog to complex number $x + yi$
  - $\epsilon^2 = 0$ instead of $i^2 = -1$
  - $\mathrm{st}(\dots)$ extracts real number, analog to $\mathrm{Re}(\dots)$ for complex number
- Laws for addition, multiplication, inverse, power can be easily derived
  - $(x + a\epsilon) + (y + b\epsilon) = (x + y) + (a + b)\epsilon$
  - $(x + a\epsilon)(y + b\epsilon) = xy + (xb + ya)\epsilon$
  - $\dfrac{1}{x + a\epsilon} = \dfrac{x - a\epsilon}{(x + a\epsilon)(x - a\epsilon)} = \dfrac{x - a\epsilon}{x^2} = \dfrac{1}{x} - \dfrac{a}{x^2}\epsilon$
  - $(x + a\epsilon)^n = (x + a\epsilon)(x + a\epsilon)\cdots = x^n + nx^{n-1}a\epsilon$
- With these rules can go on to compute polynomials, transcendental functions expressible as infinite series of polynomials and so on

- We can program these hyperreal numbers into computers

- We can program these hyperreal numbers into computers
- Any code that works on real numbers also works on hyperreal numbers

- We can program these hyperreal numbers into computers
- Any code that works on real numbers also works on hyperreal numbers
- Derivatives are computed automatically as a side-product

- We can program these hyperreal numbers into computers
- Any code that works on real numbers also works on hyperreal numbers
- Derivatives are computed automatically as a side-product
- Also works if code uses computer things like `if`, `while`, ...