# Models as Code
# Differentiable Programming with Julia

Dr. Viral B. Shah and Dr. Elliot Saba

Prepared for MSR. June 3, 2019.

# Software 2.0: Who will own the platform?

Yann LeCun
Director of AI Research,
Facebook

OK, Deep Learning has outlived its usefulness as a buzz-phrase.

Deep Learning est mort. Vive **Differentiable Programming**!

Andrej Karpathy
Director of AI,
Tesla

**Software 2.0:** Write a rough skeleton of the code, and use the computational resources at our disposal to search this space for a program that works.

Chris Lattner
Senior Director,
TensorFlow, TPU
(Google)

**Julia** is another great language with an open and active community. They are currently investing in machine learning techniques. The Julia community shares many common values as with our project.

# 3 Posters at NeurIPS 2018 (MLSys Workshop)

Julia on TPUs

Differentiable Programming — Zygote

flux — Deep Learning

# Zygote.jl: General Purpose Automatic Differentiation

```
function foo(W, Y, x)
    Z = W * Y
    a = Z * x
    b = Y * x
    c = tanh.(b)
    r = a + c
    return r
end
```

Zygote

```
function ∇foo(W, Y, x)
    Z = W * Y
    a = Z * x
    b = Y * x
    c, 𝒥tanh = ∇tanh.(b)
    a + c, function (Δr)
        Δc = Δr, Δa = Δr
        (Δtanh, Δb) = 𝒥tanh(Δc)
        (ΔY, Δx) = (Δb * x', Y' * Δb)
        (ΔZ = Δa * x', Δx += Z' * Δa)
        (ΔW = ΔZ * Y', ΔY = W' * ΔZ')
        (nothing, ΔW, ΔY, Δx)
    end
end
```

*M. Innes. Don't Unroll Adjoint: Differentiating SSA-Form Programs ([arXiv:1810.07951](arXiv:1810.07951))*

# Celeste.jl: Julia at Peta-scale

Cori: 650,000 cores. 1.3M threads. 60 TB of data.

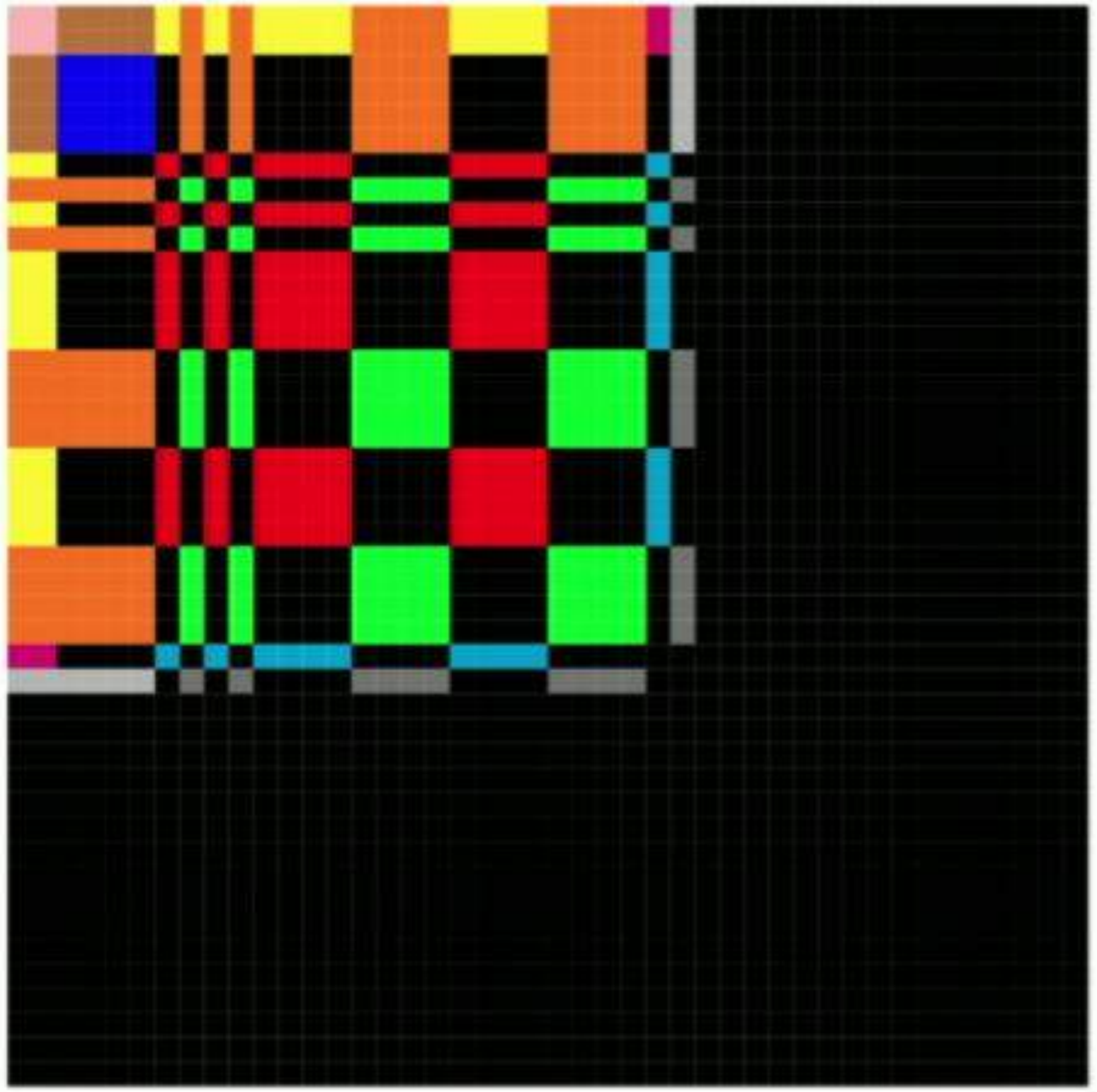
Most light sources are near the detection limit.

**Cataloging the Visible Universe through Bayesian Inference at Petascale**

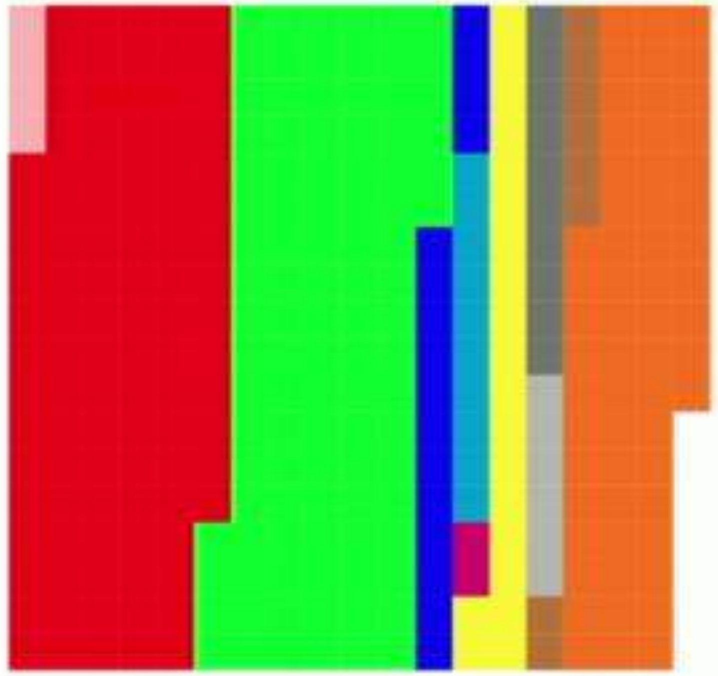
Jeffrey Regier*, Kiran Pamnany†, Keno Fischer‡, Andreas Noack§, Maximilian Lam*, Jarrett Revels§, Steve Howard¶, Ryan Giordano¶, David Schlegel‖, Jon McAuliffe¶, Rollin Thomas‖, Prabhat‖

*Department of Electrical Engineering and Computer Sciences, University of California, Berkeley
†Parallel Computing Lab, Intel Corporation
‡Julia Computing
§Computer Science and AI Laboratories, Massachusetts Institute of Technology
¶Department of Statistics, University of California, Berkeley
‖Lawrence Berkeley National Laboratory

# Celeste: Custom sparsity patterns and storage



**Matrix structure**



**Storage**

# CUDAnative.jl: Native code generation on GPUs

```
function vadd(gpu, a, b, c)
    i = threadIdx().x + blockDim().x *
      ((blockIdx().x-1) + (gpu-1) *
gridDim().x)
    @inbounds c[i] = a[i] + b[i]
    return
end

a, b, c = (CuArray(...) for _ in 1:3)
@cuda threads=length(a) vadd(1, a, b, c)
```

Provides:

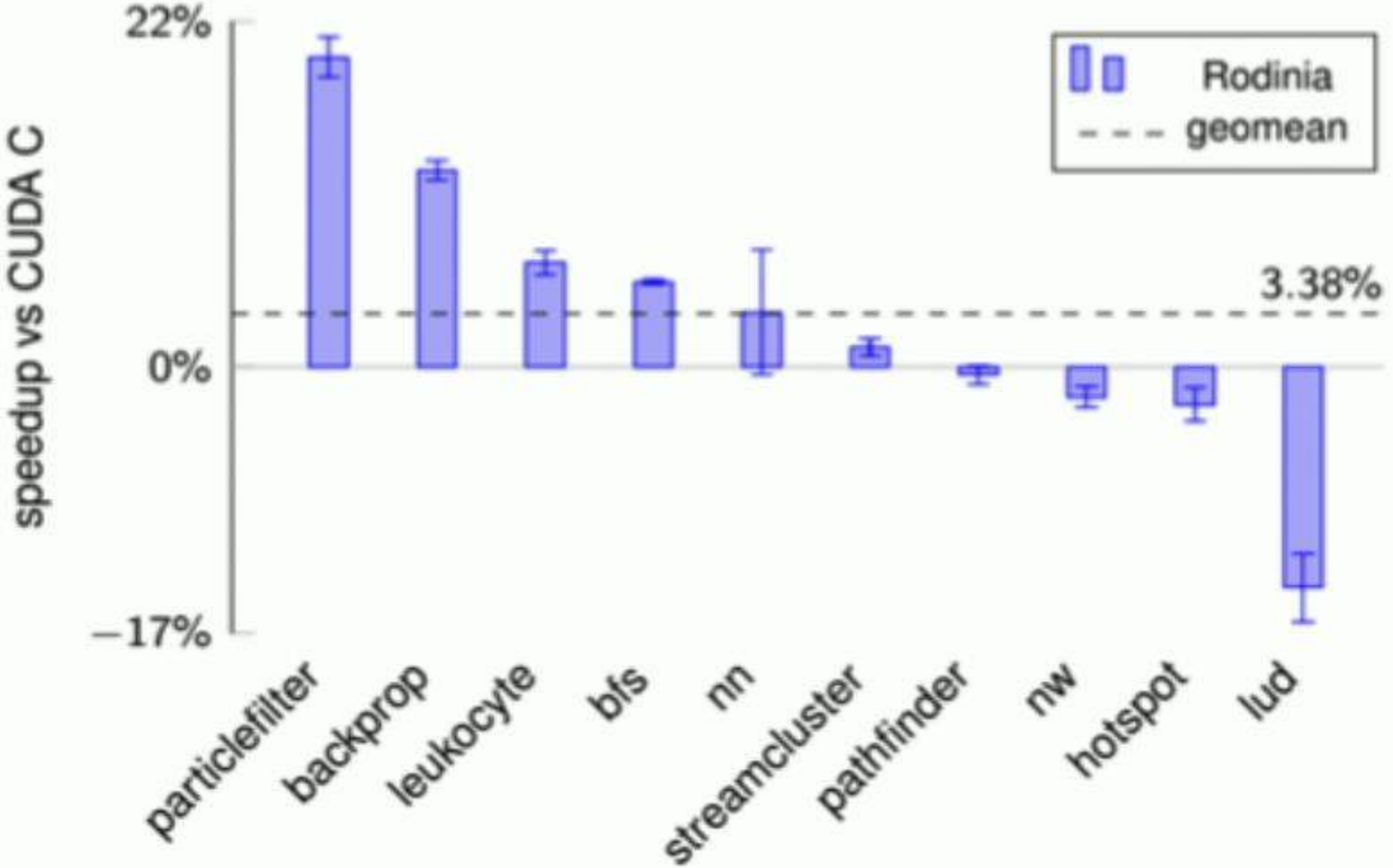
- CUDA intrinsics
- SPMD Programming model
- GPU memory management

...

Besard et al. *Effective Extensible Programming: Unleashing Julia on GPUs* (arXiv:1712.03112)

```
julia> @device_code_ptx @cuda vadd(1, a, a, a)
//
// Generated by LLVM NVPTX Back-End
//

.visible .entry ptxcall_vadd_23(
          .param .u64 ptxcall_vadd_23_param_0,
          .param .align 8 .b8
ptxcall_vadd_23_param_1[16],
          .param .align 8 .b8
ptxcall_vadd_23_param_2[16],
          .param .align 8 .b8
ptxcall_vadd_23_param_3[16]
)
{
          mov.u32   %r1, %tid.x;
          mov.u32   %r2, %ntid.x;
          mov.u32   %r3, %ctaid.x;
          ...
}
```

# CUDAnative.jl: As fast as CUDA C



# Julia runs on Google TPUs

Performance on par with TensorFlow on TPUs

Scales to pods (512 TPU cores - 4.3 $PF_{16}$/s on ResNet50)


*Fischer et al. Automatic Full Compilation of Julia Programs and ML Models to Cloud TPUs* (*[arXiv:1810.09868](https://arxiv.org/abs/1810.09868)*)

# Google.ai Lead Jeff Dean on Julia

**Jeff Dean**
@JeffDean

Following

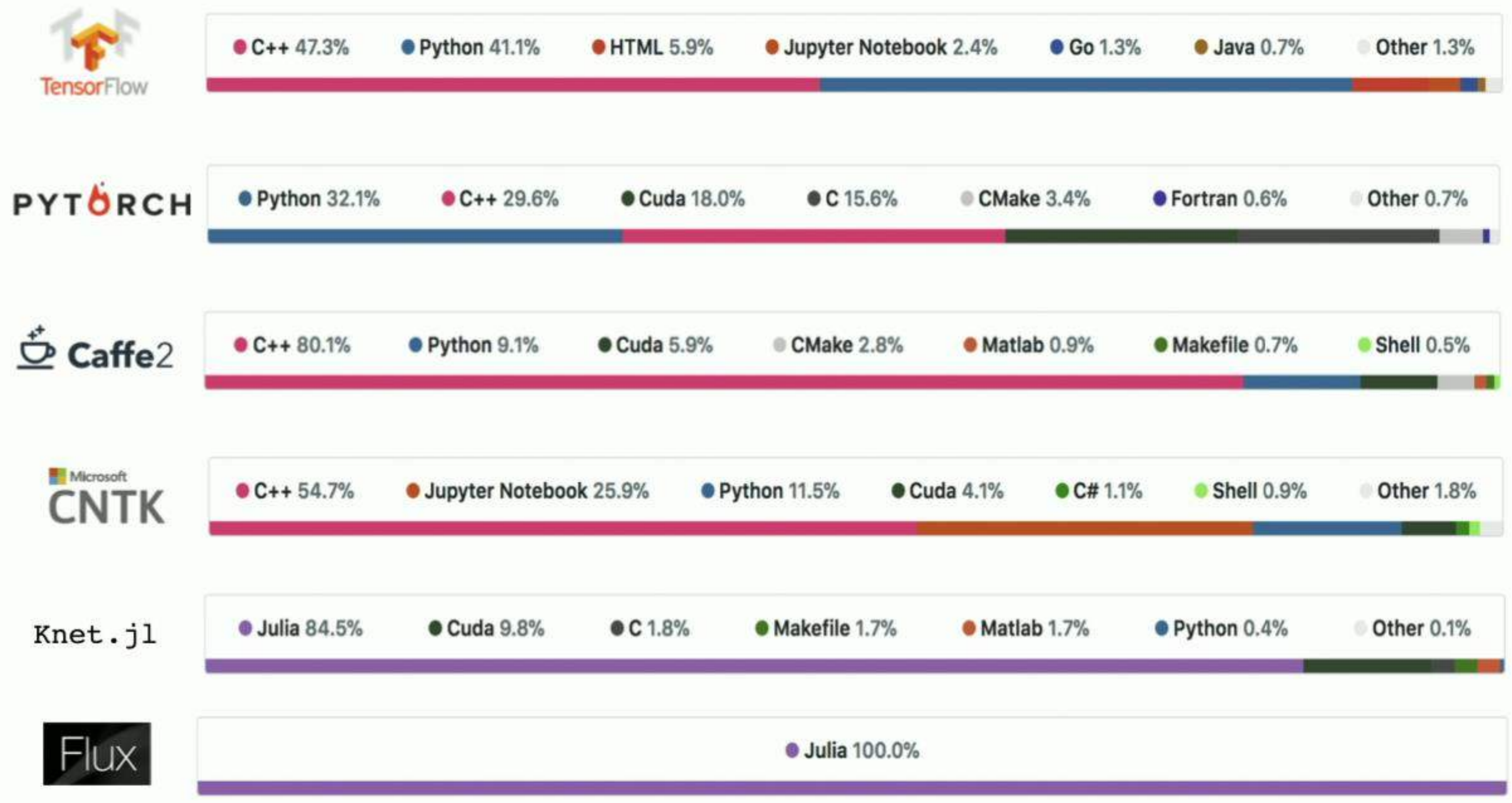
Julia + TPUs = fast and easily expressible ML computations!

> **Keno Fischer** @KenoFischer
> Our new paper today: arxiv.org/abs/1810.09868. Compile your #julialang code straight to @Google's #CloudTPU. Must go faster! We'll have an (alpha quality) repo up soon for people to start playing with this.

6:23 AM - 24 Oct 2018

**240** Retweets **617** Likes

6 240 617


TensorFlow
C++ 47.3%
Python 41.1%
HTML 5.9%
Jupyter Notebook 2.4%
Go 1.3%
Java 0.7%
Other 1.3%
PYTORCH
Python 32.1%
C++ 29.6%
Cuda 18.0%
C 15.6%
CMake 3.4%
Fortran 0.6%
Other 0.7%
Caffe2
C++ 80.1%
Python 9.1%
Cuda 5.9%
CMake 2.8%
Matlab 0.9%
Makefile 0.7%
Shell 0.5%
Microsoft CNTK
C++ 54.7%
Jupyter Notebook 25.9%
Python 11.5%
Cuda 4.1%
C# 1.1%
Shell 0.9%
Other 1.8%
Knet.jl
Julia 84.5%
Cuda 9.8%
C 1.8%
Makefile 1.7%
Matlab 1.7%
Python 0.4%
Other 0.1%
Flux
Julia 100.0%

# Best in class packages in many domains

Differential Equations

Graph Processing

Data Science

Image Processing

flux

Deep Learning

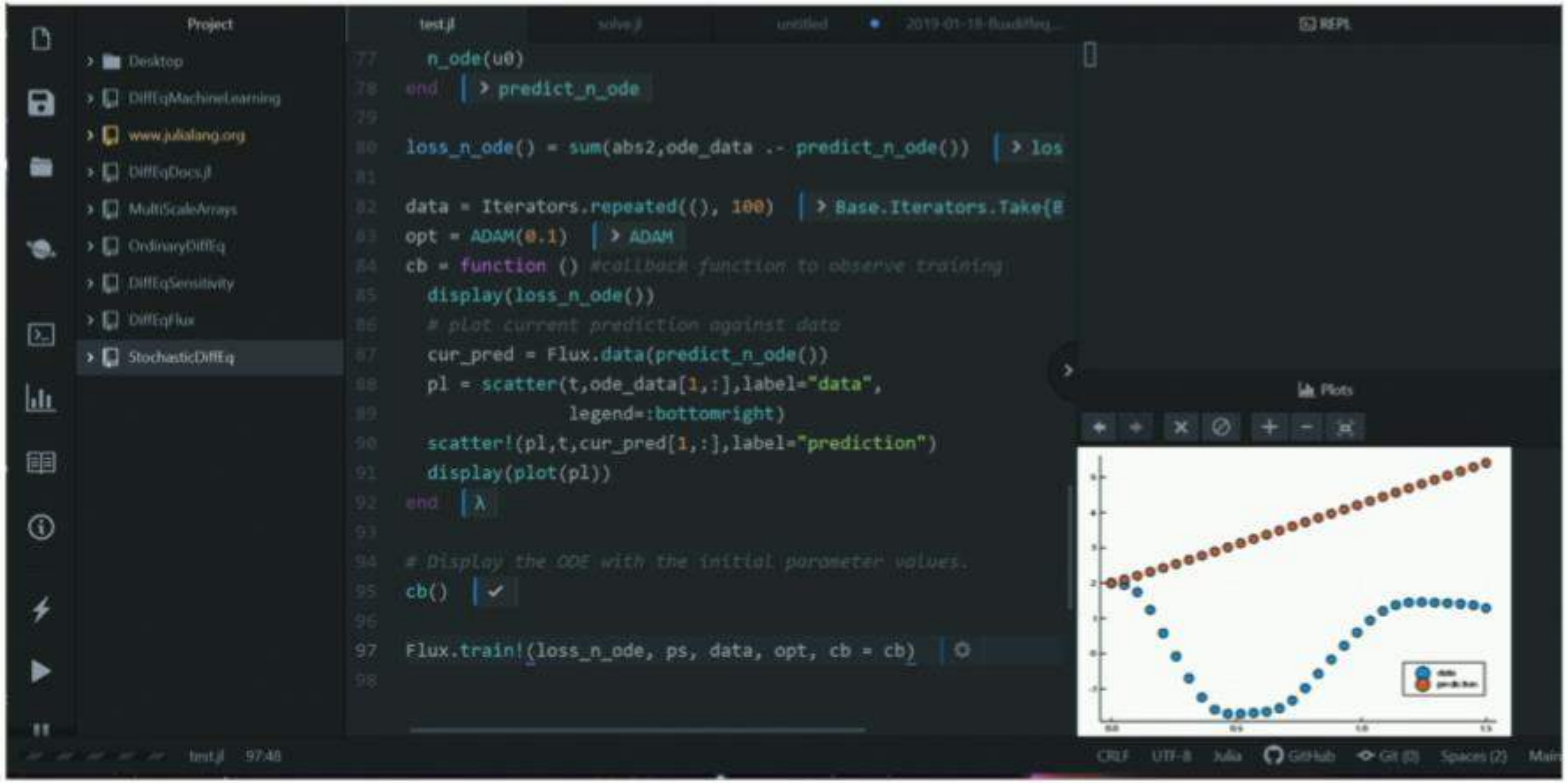
JUMP

Operations Research

Signal Processing

Computational Biology

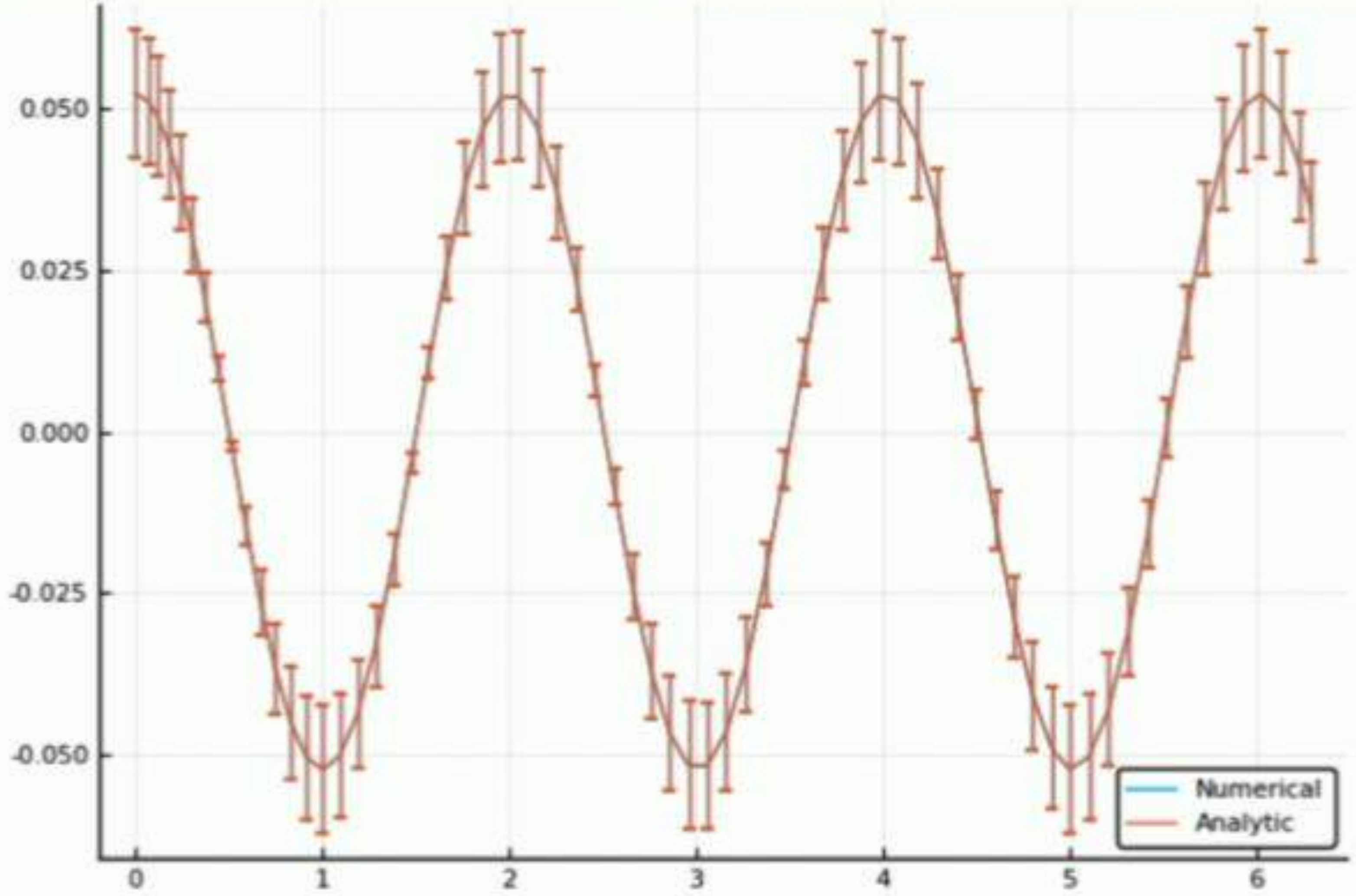
# Composability: DifferentialEquations.jl + Flux.jl (Neural ODEs)




Rackauckas et al. *DiffEqFlux.jl - A Julia Library for Neural Differential Equations (arXiv:1902.02376)*

# Composability: DifferentialEquations.jl + Measurements.jl

```julia
g = 9.79 ± 0.02 # Gravitational constants
L = 1.00 ± 0.01 # Length of the pendulum

# Initial speed & angle, time span
u₀ = [0 ± 0, π/60 ± 0.01]
tspan = (0.0, 6.3)

# Define the problem
function pendulum(du, u, p, t)
    θ  = u[1]
    dθ = u[2]
    du[1] = dθ
    du[2] = -(g/L)*θ
end

# Pass to solvers
prop = ODEProblem(pendulum, u₀, tspan)
sol = solve(prob, Tsit5(), reltol = 1e-6)

# Analytic solution
u = u₀[2] .* cos.(sqrt(g/L) .* sol.t)
```



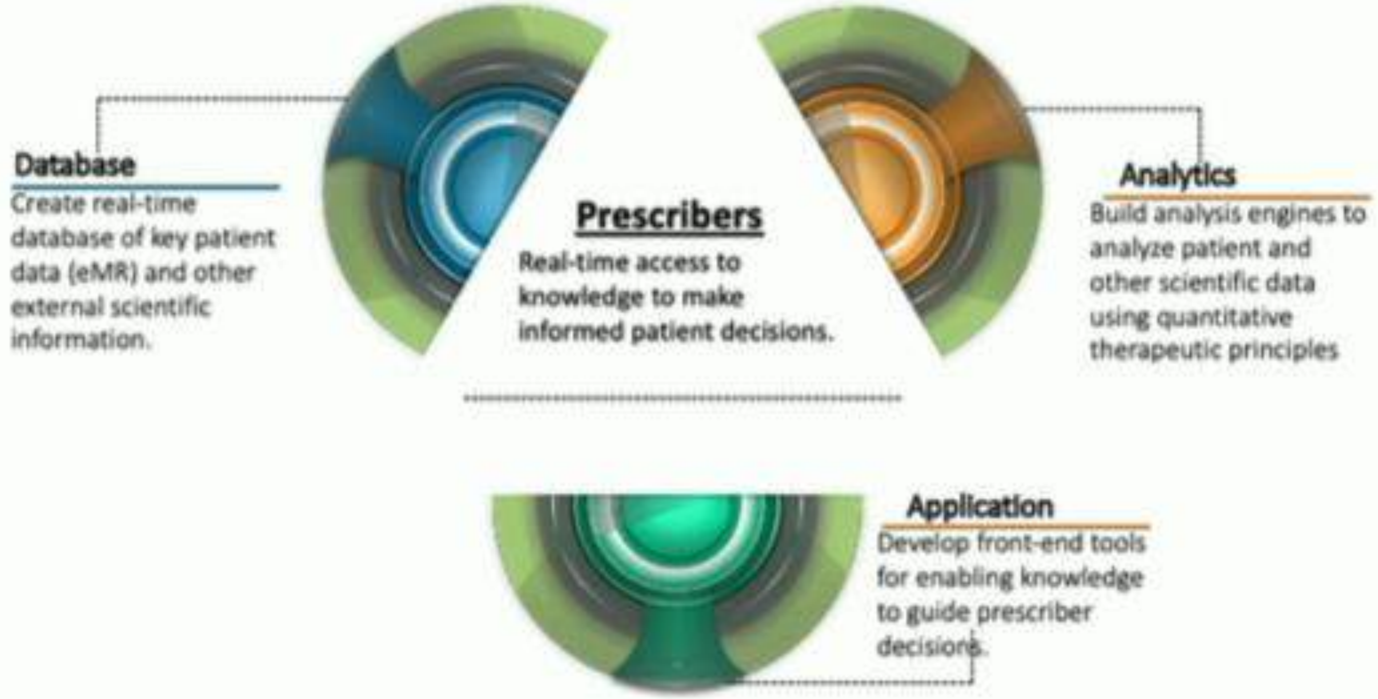

*Rackauckas et al. DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia. 2017. ([Journal of Open Research Software]())*

*Giordano. Uncertainty propagation with functionally correlated quantities ([arXiv:1610.08716]())*

# Personalized Medicine in Partnership with UMB

Joga Gobburu
Professor, School of Pharmacy (UMB)
ex-Director, Division of Pharmacometrics, US FDA

Vijay Ivaturi
Professor, School of Pharmacy (UMB)




- Accurate personalized drug dosage calculation
- Clinical trial at Johns Hopkins with Vancomycin
- Demonstrate savings. Average cost of stay is $25,000. 20% savings expected.
- FDA approval and roll-out

# Neural SDEs in Finance
## The future for optimizing portfolios of hundreds of thousands of options

Chris Rackauckas et al.

- Semilinear Parabolic Form

$$\frac{\partial u}{\partial t}(t,x) + \frac{1}{2}\mathrm{Tr}\left(\sigma\sigma^{\mathrm{T}}(t,x)(\mathrm{Hess}_x u)(t,x)\right) + \nabla u(t,x)\cdot\mu(t,x) + f\left(t,x,u(t,x),\sigma^{\mathrm{T}}(t,x)\nabla u(t,x)\right) = 0 \quad [1]$$

Then the solution of Eq. 1 satisfies the following BSDE (cf., e.g., refs. 8 and 9):

$$u(t,X_t) - u(0,X_0) = -\int_0^t f\left(s,X_s,u(s,X_s),\sigma^{\mathrm{T}}(s,X_s)\nabla u(s,X_s)\right)ds + \int_0^t [\nabla u(s,X_s)]^{\mathrm{T}}\sigma(s,X_s)\,dW_s. \quad [3]$$

- u via:

$$l(\theta) = \mathbb{E}\left[\left|g(X_{t_N}) - \hat{u}\left(\{X_{t_n}\}_{0\le n\le N},\{W_{t_n}\}_{0\le n\le N}\right)\right|^2\right].$$

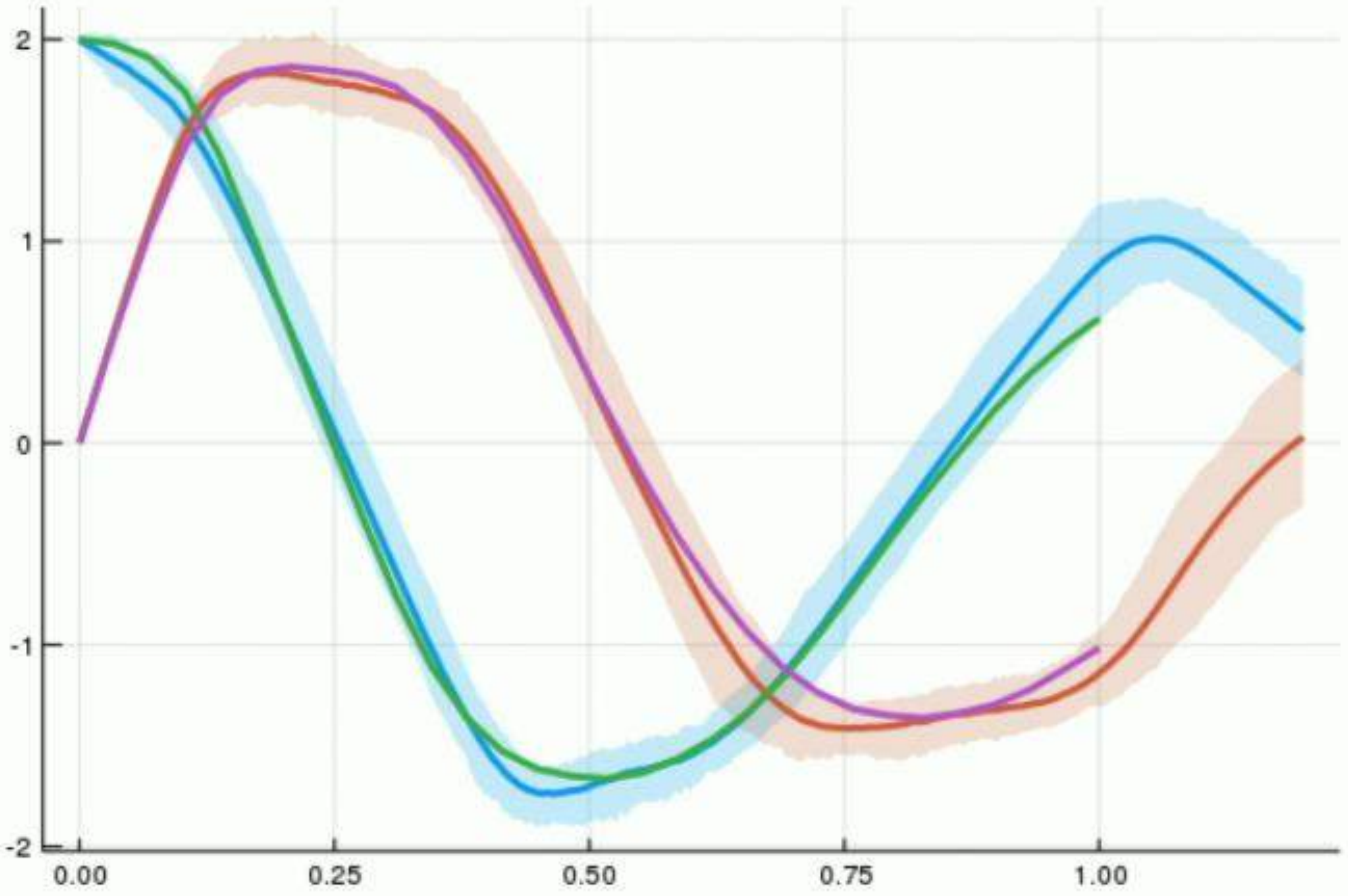
*Solving high-dimensional partial differential equations using deep learning, 2018, PNAS, Han, Jentzen, E*

- Financial Quants optimize portfolios through PDEs: Hamilton-Jacobi-Bellman, Nonlinear Black-Scholes.
- High dimensional PDEs are unsolvable by traditional mechanisms.
- New (2018) idea: transform it into a Backwards stochastic differential equation with a neural network inside of it.
- New computational challenge: get neural networks working inside of fancy mathematical (adaptive high order etc.) stochastic differential equation libraries.

We solve this by differentiating an existing library!

# Neural SDEs in action

## Neural SDEs in action: nonlinearly extrapolate time series with error bounds




See DiffEqFlux.jl Blog Post
For Code Examples

https://github.com/FluxML/model-zoo

Julia is the only language with fast SDE solvers

And

Julia is the only language which can differentiate all of its differential equation solvers.

The next generation of tools for quants
Is in Julia!

# Fixing Boston's school buses with route optimization



## U.S. NEWS

THE NUMBERS | By Jo Craven McGinty

# How Do You Fix School Bus Routes? Call MIT

A trio of MIT researchers recently tackled a tricky vehicle-routing problem when they set out to improve the efficiency of the Boston Public Schools bus system.

Last year, more than 30,000 students rode 650 buses to 230 schools at a cost of $120 million.

In hopes of spending less this year, the school system offered $15,000 in prize money in a contest that challenged competitors to reduce the number of buses.

The winners—Dimitris Bertsimas, co-director of MIT's Operations Research Center and doctoral students Arthur Delarue and Sebastien Martin—devised an algorithm that drops as many as 75 bus routes.

The school system says the plan, which will eliminate

weeks to complete.

"They have been doing it manually many years," Dr. Bertsimas said. "Our whole running time is in minutes. If things change, we can re-optimize."

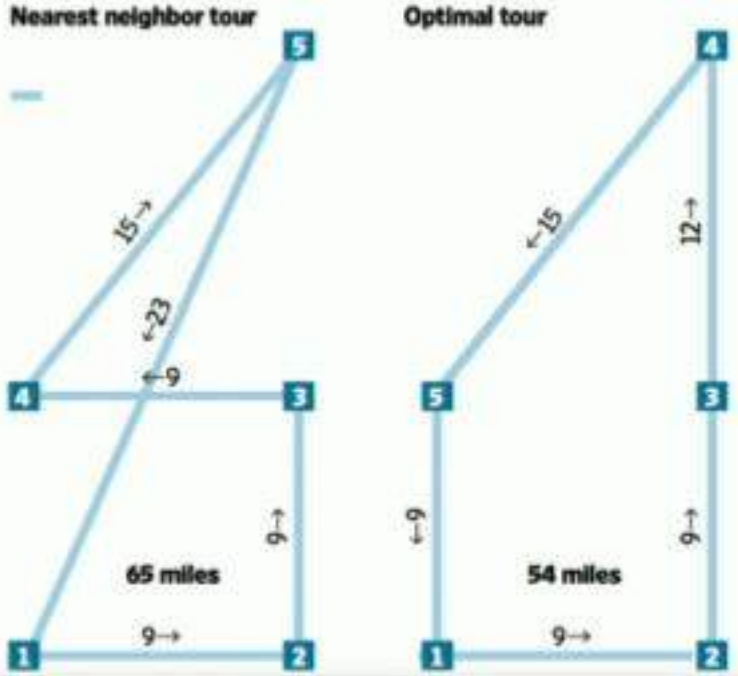
The task of plotting school-bus routes resembles the classic math exercise known as the Traveling Salesman Problem, where the goal is to find the shortest path through a series of cities, visiting each only once, before returning home.

Intuitively, traveling to the closest destination first and then the next closest after that until the tour ends would seem to guarantee the most efficient route.

But in practice, the nearest-neighbor solution rarely produces the shortest route. In one 42-city example that starts in Phoenix, following the nearest-neighbor ap-

### Road Test

Route planners often grapple with some form of the Traveling Salesman Problem where the solution is the shortest route that passes through each city once before returning home. Traveling to the nearest neighbor seems logical, but usually isn't optimal.




matics at the Limits of Computation."

One of the largest routes Dr. Cook has optimized stopped at 49,603 historic sites in the U.S. "It took 178.9 computing years to solve, running on a 310-processor computing cluster from March through November 2016," he said.

Algorithms help speed up the work. An iPhone app Dr. Cook helped develop can calculate the optimal route for a 50-city tour in seconds.

But the Boston Public Schools conundrum was more complex than the basic Traveling Salesman Problem.

The MIT researchers had to optimize multiple routes that accounted for traffic, different-size buses, students with special needs such as wheelchair access, and staggered school days that start at 7:30 a.m., 8:30 a.m. or 9:30 a.m.

They first paired clusters

overall system," Dr. Bertsimas said.

Individual students may be on a bus for more or less time than last year, but in keeping with school-system rules, no bus trip should last more than one hour, Mr. Delarue said.

Whether the plan will work as predicted remains to be seen. A previous effort to automate the system failed in 2011 when buses following routes created with software ran perpetually late.

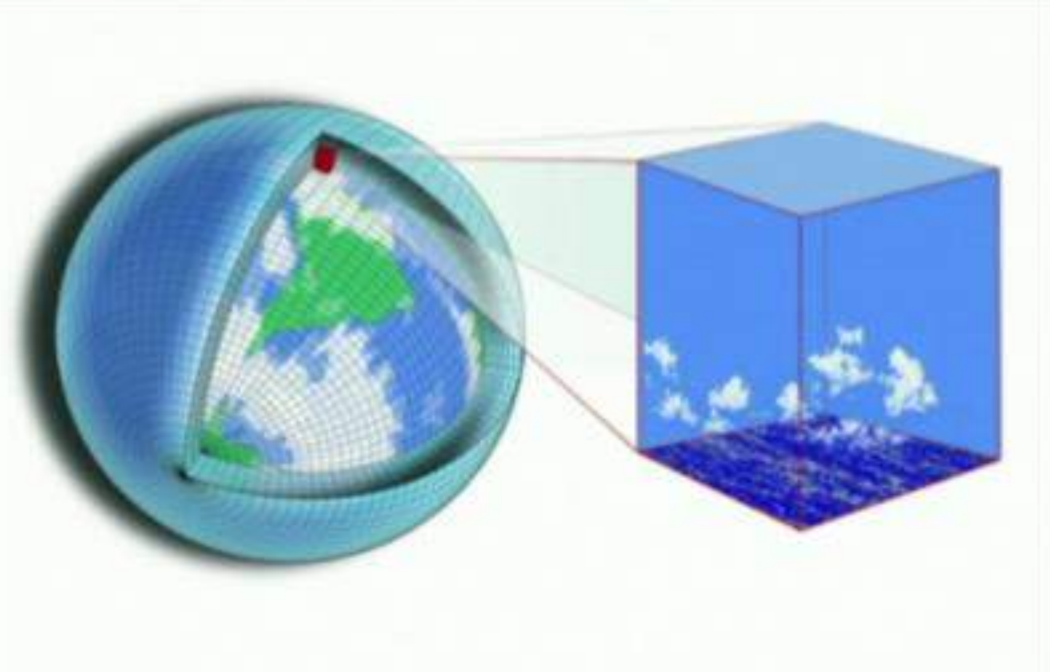
To avert similar problems this year, the school system's transportation staff vetted the MIT routes, making tweaks as needed.

"We wanted to make sure we were not picking up students on small streets with big buses," said John Hanlon, chief of operations for Boston Public Schools, noting one adjustment his staff has

# Climate modeling and Energy Optimization

MIT News
ON CAMPUS AND AROUND THE WORLD

Browse or Search



FULL SCREEN

MIT professors Raffaele Ferrari and John Marshall, along with colleagues from Caltech, NASA's Jet Propulsion Lab, and the Naval Postgraduate School, envision a revolution in climate modeling using data assimilation and machine learning.

Image courtesy of Caltech and Tapio Schneider

## New climate model to be built from the ground up

Scientists and engineers will collaborate in a new Climate Modeling Alliance to advance climate modeling and prediction.

MIT News
ON CAMPUS AND AROUND THE WORLD

Browse or Search


FULL SCREEN

New MIT research shows that, unless steady, continuous carbon-free sources of electricity are included in the mix, costs of decarbonizing the electrical system could be prohibitive and end up derailing attempts to mitigate the most severe effects of global climate change.

Image: Chelsea Turner

## Study: Adding power choices reduces cost and risk of carbon-free electricity

To curb greenhouse gas emissions, nations, states, and cities should aim for a mix of fuel-saving, flexible, and highly reliable sources.

# How to implement all that?

Easy, generic, fast: *pick two*

Common approaches

- Heroic C++ template code plus Python/R wrapper
- Problem-specific compilers

# Key idea

*Language for describing what to specialize on.*

- Design descriptive types for the domain at hand
- Write methods for whatever cases you can handle
- Compiler generates specializations
- These need to be *de-coupled*

# Sliding scale of specialization

| | |
|---|---|
| `Array` | Some kind of array |
| `Array{Int}` | Element type known to be Int |
| `Array{Int,2}` | … and 2-dimensional |
| `Array{<:Any,2}` | … or unknown element type |
| `Array{<:Real,2}` | … or unknown, but `Real`, element type |
| `SArray{(2,3),Float64,2}` | 2d `Float64` with dimensions 2x3 |

# Other special matrix types

- `Diagonal`
- `UniformScaling`
- `Symmetric, Hermitian`
- `LowerTriangular, UpperTriangular`
- `Bidiagonal, Tridiagonal, SymTridiagonal`
- `Adjoint, Transpose`

# Example: one-hot vector type

```
struct OneHotVector <: AbstractVector{Bool}
    index::Int
    len::Int
end

size(xs::OneHotVector)                    = (xs.len,)

getindex(xs::OneHotVector, i::Integer) = i == xs.index

A::AbstractMatrix * b::OneHotVector      = A[:, b.index]
```

# Other special matrix types

- `Diagonal`
- `UniformScaling`
- `Symmetric, Hermitian`
- `LowerTriangular, UpperTriangular`
- `Bidiagonal, Tridiagonal, SymTridiagonal`
- `Adjoint, Transpose`

# Representing layers of VGG19 neural net

```
ImmutableChain{Tuple{Conv{typeof(relu),Array{Float32,4},Array{Float32,1},(1, 1),(1, 1),(1, 1)},
Conv{typeof(relu),Array{Float32,4},Array{Float32,1},(1, 1),(1, 1),(1, 1)},
Metalhead.f42,Conv{typeof(relu),Array{Float32,4},Array{Float32,1},(1, 1),(1, 1),(1, 1)},
Conv{typeof(relu),Array{Float32,4},Array{Float32,1},(1, 1),(1, 1),(1, 1)},
Metalhead.f43,Conv{typeof(relu),Array{Float32,4},Array{Float32,1},(1, 1),(1, 1),(1, 1)},
Conv{typeof(relu),Array{Float32,4},Array{Float32,1},(1, 1),(1, 1),(1, 1)},
Conv{typeof(relu),Array{Float32,4},Array{Float32,1},(1, 1),(1, 1),(1, 1)},
Conv{typeof(relu),Array{Float32,4},Array{Float32,1},(1, 1),(1, 1),(1, 1)},
Metalhead.f44,Conv{typeof(relu),Array{Float32,4},Array{Float32,1},(1, 1),(1, 1),(1, 1)},
Conv{typeof(relu),Array{Float32,4},Array{Float32,1},(1, 1),(1, 1),(1, 1)},
Conv{typeof(relu),Array{Float32,4},Array{Float32,1},(1, 1),(1, 1),(1, 1)},
Conv{typeof(relu),Array{Float32,4},Array{Float32,1},(1, 1),(1, 1),(1, 1)},
Metalhead.f45,Conv{typeof(relu),Array{Float32,4},Array{Float32,1},(1, 1),(1, 1),(1, 1)},
Conv{typeof(relu),Array{Float32,4},Array{Float32,1},(1, 1),(1, 1),(1, 1)},
Conv{typeof(relu),Array{Float32,4},Array{Float32,1},(1, 1),(1, 1),(1, 1)},
Conv{typeof(relu),Array{Float32,4},Array{Float32,1},(1, 1),(1, 1),(1, 1)},
Metalhead.f46,Metalhead.f47,Dense{typeof(relu),LinearAlgebra.Adjoint{Float32,Array{Float32,2}},Array{F
loat32,1}},Dense{typeof(relu),LinearAlgebra.Adjoint{Float32,Array{Float32,2}},Array{Float32,1}},Dense{
typeof(identity),LinearAlgebra.Adjoint{Float32,Array{Float32,2}},Array{Float32,1}},typeof(softmax)}}
```

# Recent work: type system formalization

**Julia Subtyping: a Rational Reconstruction**

F. Zappa Nardelli, J. Belyakova, A. Pelenitsyn, B. Chung, J. Bezanson, J. Vitek

OOPSLA 2018

Paper: https://www.di.ens.fr/~zappa/projects/lambdajulia/

$$\frac{}{E \vdash t \ <: \ \mathtt{Any} \ \vdash E}\ [\textsc{top}]$$

$$\frac{}{E \vdash a \ <: \ a \vdash E}\ [\textsc{refl}]$$

$$\frac{E \vdash a_1 \ <: \ a'_1 \ \vdash E_1 \quad .. \quad E_{n-1} \vdash a_n \ <: \ a'_n \ \vdash E_n \qquad consistent(E_n)}{E \vdash \mathtt{Tuple}\{a_1, .., a_n\} \ <: \ \mathtt{Tuple}\{a'_1, .., a'_n\} \ \vdash E_n}\ [\textsc{tuple}]$$

$$\frac{t' = lift\_union(\mathtt{Tuple}\{a_1, .., a_n\}) \qquad E \vdash t' \ <: \ t \vdash E'}{E \vdash \mathtt{Tuple}\{a_1, .., a_n\} \ <: \ t \vdash E'}\ [\textsc{tuple\_lift\_union}]$$

$$\frac{t' = unlift\_union(\mathtt{Union}\{t_1, .., t_n\}) \qquad E \vdash t \ <: \ t' \ \vdash E'}{E \vdash t \ <: \ \mathtt{Union}\{t_1, .., t_n\} \vdash E'}\ [\textsc{tuple\_unlift\_union}]$$

$$\frac{E \vdash t_1 \ <: \ t \vdash E_1 \quad .. \quad reset\_occ_E(E_{n-1}) \vdash t_n \ <: \ t \vdash E_n}{E \vdash \mathtt{Union}\{t_1, .., t_n\} \ <: \ t \vdash max\_occ_{E_1..E_n}(E_n)}\ [\textsc{union\_left}]$$

$$\frac{\exists j. E \vdash t \ <: \ t_j \vdash E'}{E \vdash t \ <: \ \mathtt{Union}\{t_1, .., t_n\} \vdash E'}\ [\textsc{union\_right}]$$

$$\frac{n \leq m \quad E_0 = add(\mathsf{Barrier}, E) \qquad \forall\, 0 < i \leq n.\ \ E_{i-1} \vdash a_i \ <: \ a'_i \ \vdash E'_i \ \wedge \ E'_i \vdash a'_i \ <: \ a_i \vdash E_i}{E \vdash name\{a_1, .., a_m\} \ <: \ name\{a'_1, .., a'_n\} \vdash del(\mathsf{Barrier}, E_n)}\ [\textsc{app\_inv}]$$

$$\frac{name\{\top_1, .., \top_m, ..\} \ <: \ t'' \in tds \qquad E \vdash t''[a_1/\top_1 .. a_m/\top_m] \ <: \ t' \ \vdash E'}{E \vdash name\{a_1, .., a_m\} \ <: \ t' \ \vdash E'}\ [\textsc{app\_super}]$$

$$\frac{add({}^{L}\top_{t_1}^{t_2}, E) \vdash t \ <: \ t' \ \vdash E'}{E \vdash t \ \mathsf{where}\ t_1 <: \top <: t_2 \ <: \ t' \ \vdash del(\top, E')}\ [\textsc{l\_intro}]$$

$$\frac{add({}^{R}\top_{t_1}^{t_2}, E) \vdash t \ <: \ t' \ \vdash E' \qquad consistent(\top, E')}{E \vdash t \ <: \ t' \ \mathsf{where}\ t_1 <: \top <: t_2 \vdash del(\top, E')}\ [\textsc{r\_intro}]$$

# Recent work: type system formalization

**Julia Subtyping: a Rational Reconstruction**

F. Zappa Nardelli, J. Belyakova, A. Pelenitsyn, B. Chung, J. Bezanson, J. Vitek

OOPSLA 2018

Paper: https://www.di.ens.fr/~zappa/projects/lambdajulia/

$$\frac{}{E \vdash t \ <: \ \texttt{Any} \ \vdash E}\ [\textsc{top}] \qquad \frac{}{E \vdash a \ <: \ a \vdash E}\ [\textsc{refl}]$$

$$\frac{E \vdash a_1 \ <: \ a_1' \ \vdash E_1 \ \ .. \ \ E_{n-1} \vdash a_n \ <: \ a_n' \ \vdash E_n \quad consistent(E_n)}{E \vdash \texttt{Tuple}\{a_1, \ .., a_n\} \ <: \ \texttt{Tuple}\{a_1', \ .., a_n'\} \ \vdash E_n}\ [\textsc{tuple}]$$

$$\frac{t' = lift\_union(\texttt{Tuple}\{a_1, \ .., a_n\}) \quad E \vdash t' \ <: \ t \vdash E'}{E \vdash \texttt{Tuple}\{a_1, \ .., a_n\} \ <: \ t \ \vdash E'}\ [\textsc{tuple\_lift\_union}]$$

$$\frac{t' = unlift\_union(\texttt{Union}\{t_1, \ .., t_n\}) \quad E \vdash t \ <: \ t' \ \vdash E'}{E \vdash t \ <: \ \texttt{Union}\{t_1, \ .., t_n\} \ \vdash E'}\ [\textsc{tuple\_unlift\_union}]$$

$$\frac{E \vdash t_1 \ <: \ t \ \vdash E_1 \ \ .. \ \ reset\_occ_E(E_{n-1}) \vdash t_n \ <: \ t \ \vdash E_n}{E \vdash \texttt{Union}\{t_1, \ .., t_n\} \ <: \ t \ \vdash max\_occ_{E_1..E_n}(E_n)}\ [\textsc{union\_left}]$$

$$\frac{\exists j. \, E \vdash t \ <: \ t_j \ \vdash E'}{E \vdash t \ <: \ \texttt{Union}\{t_1, \ .., t_n\} \ \vdash E'}\ [\textsc{union\_right}]$$

$$\frac{n \leq m \quad E_0 = add(\text{Barrier}, E) \quad \forall\, 0 < i \leq n. \ \ E_{i-1} \vdash a_i \ <: \ a_i' \ \vdash E_i' \ \wedge \ E_i' \vdash a_i' \ <: \ a_i \ \vdash E_i}{E \vdash name\{a_1, \ .., a_m\} \ <: \ name\{a_1', \ .., a_n'\} \ \vdash del(\text{Barrier}, E_n)}\ [\textsc{app\_inv}]$$

$$\frac{name\{\top_1, \ .., \top_m, \ ..\} \ <: \ t'' \in tds \quad E \vdash t''[a_1/\top_1 \,..\, a_m/\top_m] \ <: \ t' \ \vdash E'}{E \vdash name\{a_1, \ .., a_m\} \ <: \ t' \ \vdash E'}\ [\textsc{app\_super}]$$

$$\frac{add({}^{L}\top_{t_1}^{t_2}, E) \vdash t \ <: \ t' \ \vdash E'}{E \vdash t \ \textsf{where} \ t_1 <: \top <: t_2 \ <: \ t' \ \vdash del(\top, E')}\ [\textsc{l\_intro}]$$

$$\frac{add({}^{R}\top_{t_1}^{t_2}, E) \vdash t \ <: \ t' \ \vdash E' \quad consistent(\top, E')}{E \vdash t \ <: \ t' \ \textsf{where} \ t_1 <: \top <: t_2 \ \vdash del(\top, E')}\ [\textsc{r\_intro}]$$

$$\frac{search(\mathsf{T}, E) = {}^{L}\mathsf{T}_{l}^{u} \quad E \vdash u <: t \vdash E'}{E \vdash \mathsf{T} <: t \vdash E'}\ [\text{L\_LEFT}]$$

$$\frac{search(\mathsf{T}, E) = {}^{L}\mathsf{T}_{l}^{u} \quad E \vdash t <: l \vdash E'}{E \vdash t <: \mathsf{T} \vdash E'}\ [\text{L\_RIGHT}]$$

$$\frac{search(\mathsf{T}_1, E) = {}^{R}{\mathsf{T}_1}_{l_1}^{u_1} \quad search(\mathsf{T}_2, E) = {}^{L}{\mathsf{T}_2}_{l_2}^{u_2} \quad outside(\mathsf{T}_1, \mathsf{T}_2, E) \Rightarrow E \vdash u_2 <: l_2 \vdash E' \quad E \vdash u_1 <: l_2 \vdash E''}{E \vdash \mathsf{T}_1 <: \mathsf{T}_2 \vdash upd({}^{R}{\mathsf{T}_1}_{\mathtt{Union}\{\mathsf{T}_1, l_1\}}^{u_1}, E')}\ [\text{R\_L}]$$

$$\frac{search(\mathsf{T}, E) = {}^{R}\mathsf{T}_{l}^{u} \quad E \vdash l <: t \vdash E'}{E \vdash \mathsf{T} <: t \vdash upd({}^{R}\mathsf{T}_{l}^{t}, E')}\ [\text{R\_LEFT}]$$

$$\frac{search(\mathsf{T}, E) = {}^{R}\mathsf{T}_{l}^{u} \quad (is\_var(t) \wedge search(t, E) = {}^{L}\mathsf{S}_{l_1}^{u_1}) \Rightarrow \neg outside(\mathsf{T}, \mathsf{S}, E) \quad E \vdash t <: u \vdash E'}{E \vdash t <: \mathsf{T} \vdash upd({}^{R}\mathsf{T}_{\mathtt{Union}\{l, t\}}^{u}, E')}\ [\text{R\_RIGHT}]$$

$$\frac{\neg is\_var(a_1) \quad E \vdash typeof(a_1) <: t_2 \vdash E'}{E \vdash \mathtt{Type}\{a_1\} <: t_2 \vdash E'}\ [\text{TYPE\_LEFT}]$$

$$\frac{is\_kind(t_1) \quad is\_var(t_2) \quad E \vdash \mathtt{Type}\{\mathsf{T}\}\ \mathbf{where}\ \mathsf{T} <: \mathtt{Type}\{t_2\} \vdash E'}{E \vdash t_1 <: \mathtt{Type}\{t_2\} \vdash E'}\ [\text{TYPE\_RIGHT}]$$

$$\frac{add(\text{Barrier}, E) \vdash a_1 <: a_2 \vdash E' \quad E' \vdash a_2 <: a_1 \vdash E''}{E \vdash \mathtt{Type}\{a_1\} <: \mathtt{Type}\{a_2\} \vdash del(\text{Barrier}, E'')}\ [\text{TYPE\_TYPE}]$$

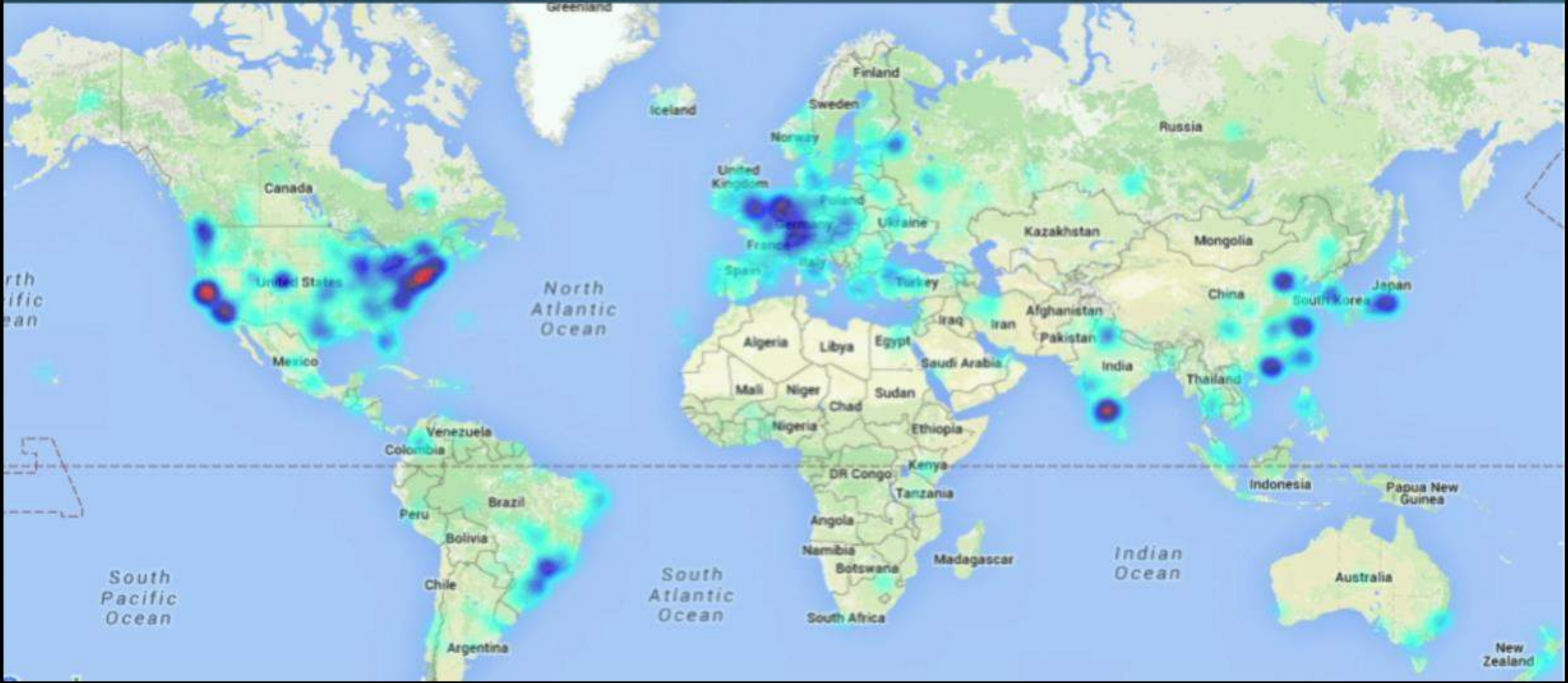
# Exploring novel data types: `BFloat16`

- New numeric type used for machine learning on TPUs
- 8 mantissa bits, 8 exponent bits
- Efficient Julia implementation is <100 LOC
- Harmonic sum in floating point (Source: Nick Higham's blog)

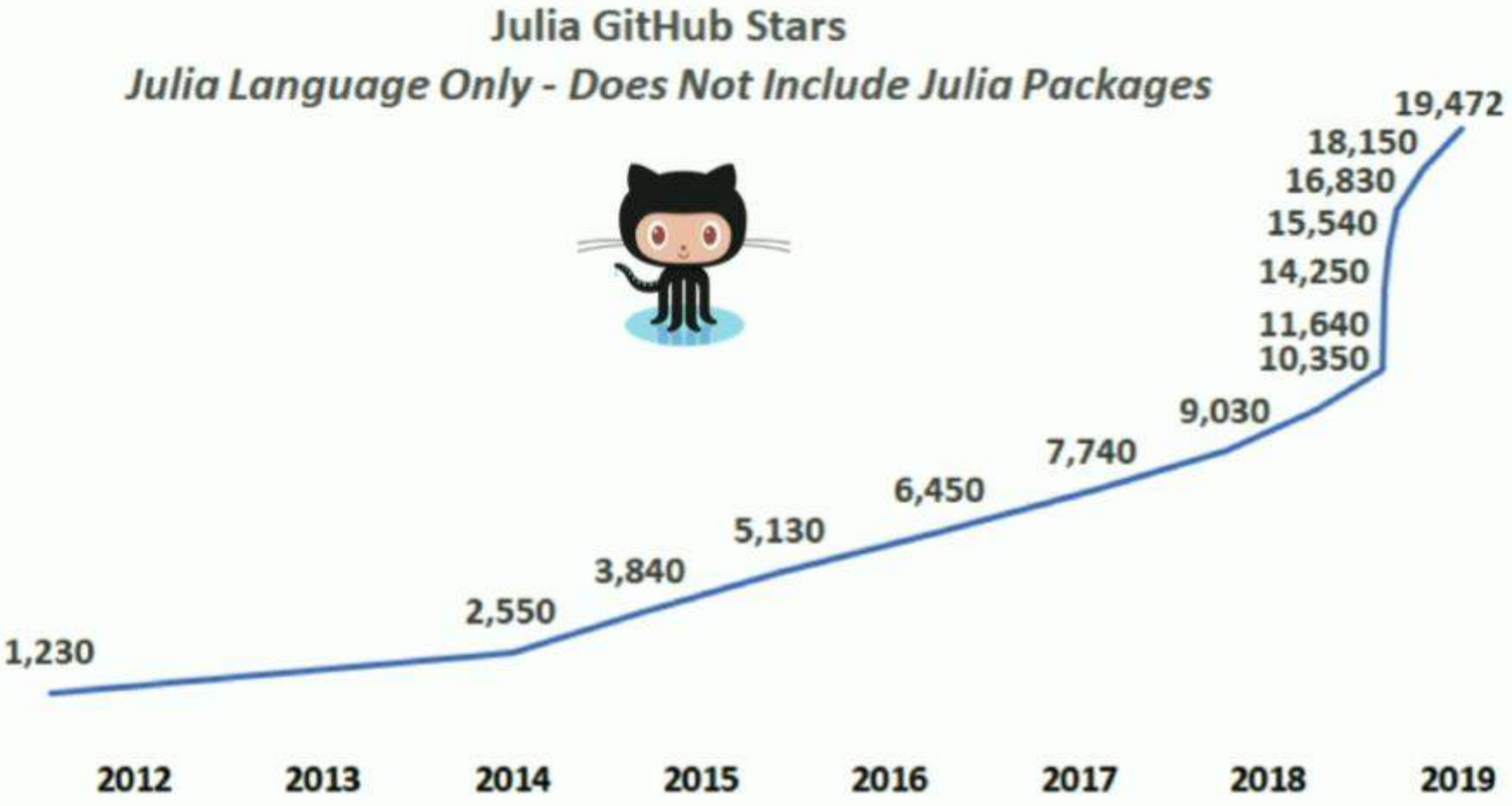
| Arithmetic | Computed Sum | Number of terms |
|---|---|---|
| bfloat16 | 5.0625 | 65 |
| fp16 | 7.0859 | 513 |
| fp32 | 15.404 | 2097152 |
| fp64 | 34.122 | $2.81 \cdots \times 10^{14}$ |

# A Global Community

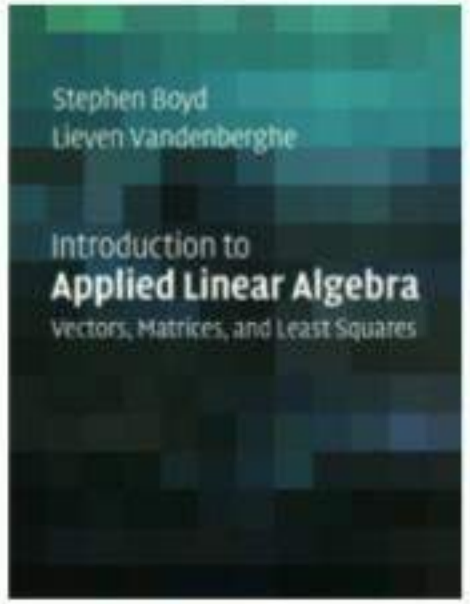
## Over 3 Million Downloads. 2,500 Packages.

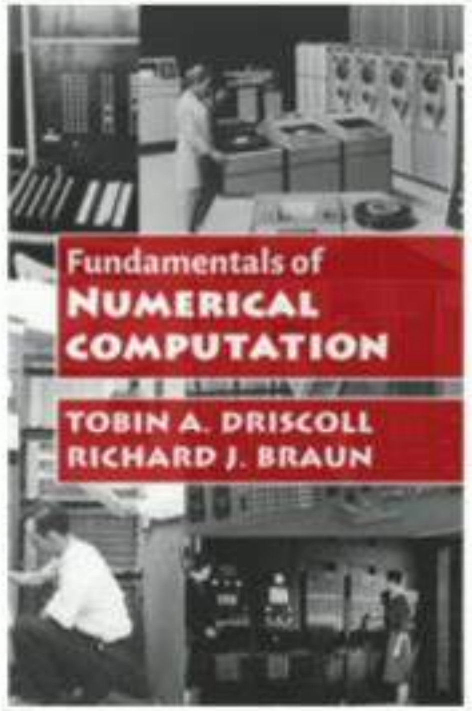


# A Growing Community



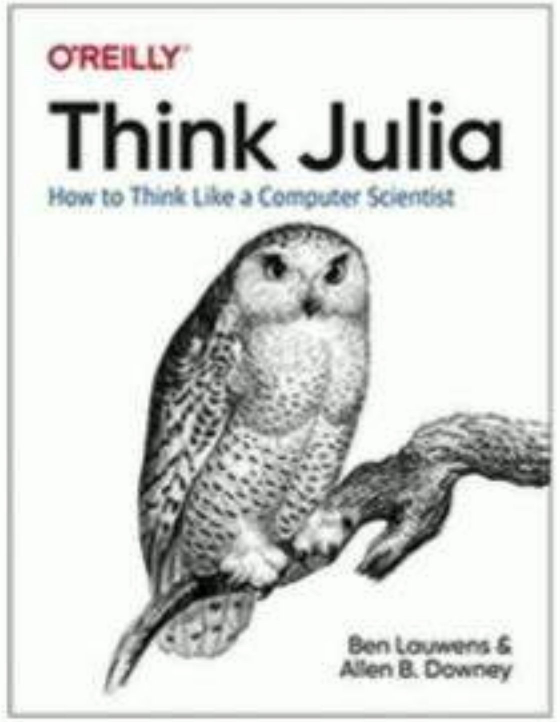
# Books

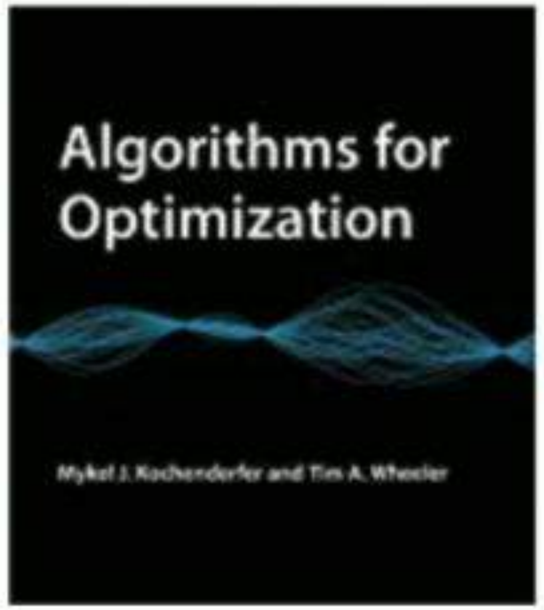

Stephen Boyd
Lieven Vandenberghe
Introduction to
Applied Linear Algebra
Vectors, Matrices, and Least Squares


Fundamentals of
NUMERICAL COMPUTATION
TOBIN A. DRISCOLL
RICHARD J. BRAUN

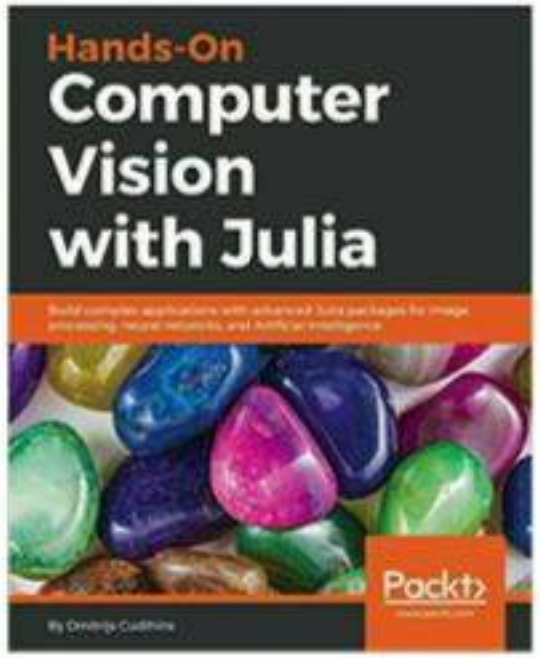

Ivo Balbaert, Avik Sengupta
Malcolm Sherrington
Julia: High Performance Programming
Learning Path
Packt›


Data Science with Julia
Paul D. McNicholas
Peter A. Tait
CRC Press

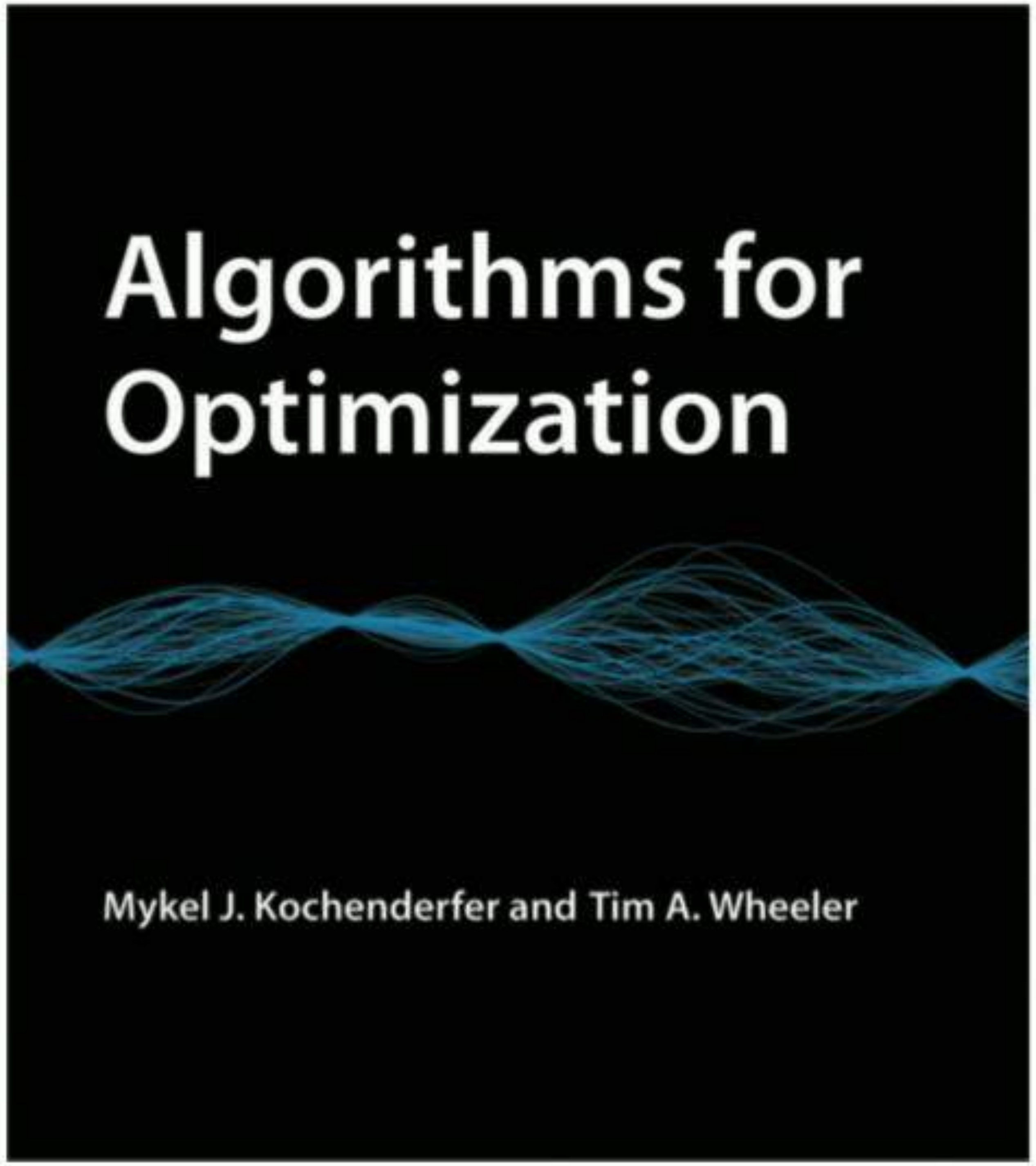

O'REILLY
Think Julia
How to Think Like a Computer Scientist
Ben Lauwens &
Allen B. Downey


Algorithms for
Optimization
Mykel J. Kochenderfer and Tim A. Wheeler


JULIA
PROGRAMMING
for
OPERATIONS RESEARCH
A Primer on Computing


Hands-On
Computer
Vision
with Julia
Packt›

# Books


Algorithms for Optimization
Mykel J. Kochenderfer and Tim A. Wheeler

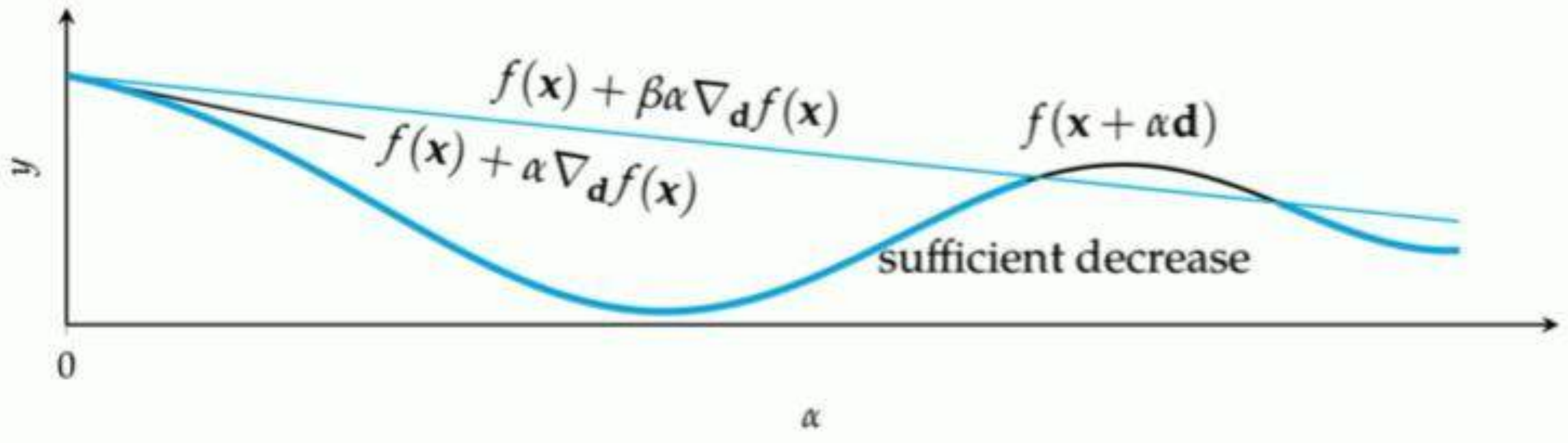



Figure 4.1. The sufficient decrease condition, the first Wolfe condition, can always be satisfied by a sufficiently small step size along a descent direction.

If $\mathbf{d}$ is a valid descent direction, then there must exist a sufficiently small step size that satisfies the sufficient decrease condition. We can thus start with a large step size and decrease it by a constant reduction factor until the sufficient decrease condition is satisfied. This algorithm is known as *backtracking line search*[6] because of how it backtracks along the descent direction. Backtracking line search is shown in figure 4.2 and implemented in algorithm 4.2. We walk through the procedure in example 4.2.

[6] Also known as *Armijo line search*, L. Armijo, "Minimization of Functions Having Lipschitz Continuous First Partial Derivatives," *Pacific Journal of Mathematics*, vol. 16, no. 1, pp. 1–3, 1966.

```
function backtracking_line_search(f, ∇f, x, d, α; p=0.5, β=1e-4)
    y, g = f(x), ∇f(x)
    while f(x + α*d) > y + β*α*(g⋅d)
        α *= p
    end
    α
end
```

Algorithm 4.2. The backtracking line search algorithm, which takes objective function `f`, its gradient `∇f`, the current design point `x`, a descent direction `d`, and the maximum step size `α`. We can optionally specify the reduction factor `p` and the first Wolfe condition parameter `β`.

```
function direct(f, a, b, ϵ, k_max)
    g = reparameterize_to_unit_hypercube(f, a, b)
    intervals = Intervals()
    n = length(a)
    c = fill(0.5, n)
    interval = Interval(c, g(c), fill(0, n))
    add_interval!(intervals, interval)
    c_best, y_best = copy(interval.c), interval.y

    for k in 1 : k_max
        S = get_opt_intervals(intervals, ϵ, y_best)
        to_add = Interval[]
        for interval in S
            append!(to_add, divide(g, interval))
            dequeue!(intervals[min_depth(interval)])
        end
        for interval in to_add
            add_interval!(intervals, interval)
            if interval.y < y_best
                c_best, y_best = copy(interval.c), interval.y
            end
        end
    end

    return rev_unit_hypercube_parameterization(c_best, a, b)
end
```

Algorithm 7.8. DIRECT, which takes the multidimensional objective function `f`, vector of lower bounds `a`, vector of upper bounds `b`, tolerance parameter `ϵ`, and number of iterations `k_max`. It returns the best coordinate.

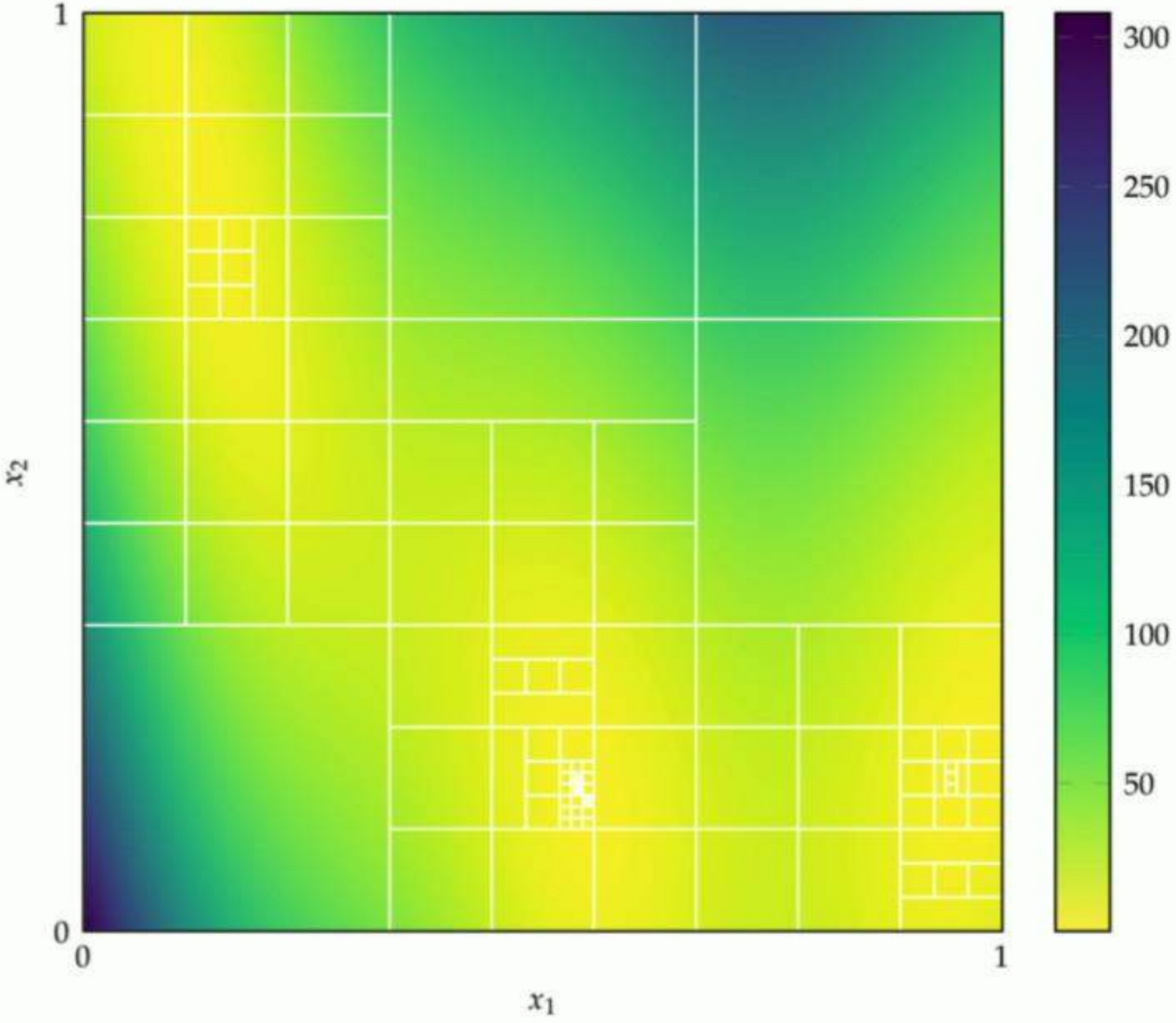



Figure 7.20. The DIRECT method after 16 iterations on the Branin function, appendix B.3. Each cell is bordered by white lines. The cells are much denser around the minima of the Branin function, as the DIRECT method procedurally increases its resolution in those regions.

# Some of the Universities Teaching Julia

BROWN UNIVERSITY
CORNELL UNIVERSITY FOUNDED A.D. 1865
EMORY UNIVERSITY
NORTHEASTERN UNIVERSITY BOSTON, MASSACHUSETTS
PSU PENN STATE UNIVERSITY
PURDUE
SZKOŁA GŁÓWNA HANDLOWA W WARSZAWIE
Stanford University
THE UNIVERSITY OF EDINBURGH
UNIVERSITY of GLASGOW
UNIVERSITY of SOUTH FLORIDA 1956
UNIVERSITE PAUL SABATIER TOULOUSE III
UNIVERSITY OF CONNECTICUT 1881
FIDE SPLENDET ET SCIENTIA
THE UNIVERSITY OF CALIFORNIA BERKELEY 1868
Western UNIVERSITY · CANADA
ARIZONA STATE UNIVERSITY 1885
MŰEGYETEM 1782
ILLINOIS STATE UNIVERSITY 1857
CUNY THE CITY UNIVERSITY OF NEW YORK
EPFL ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
MIT
TOKYO METROPOLITAN UNIVERSITY
首都大学東京
UCLA
AGH

# James H. Wilkinson Prize for Numerical Software

Jeff Bezanson, Stefan Karpinski, Viral Shah (2019)

> For the development of Julia, an innovative environment for the creation of high-performance tools that enable the analysis and solution of computational science problems.
>
> Julia allows researchers to write high-level code in an intuitive syntax and produce code with the speed of production programming languages. Julia has been widely adopted by the scientific computing community for application areas that include astronomy, economics, deep learning, energy optimization, and medicine.
>
> In particular, the Federal Aviation Administration has chosen Julia as the language for the next generation airborne collision avoidance system.

# JuliaCon 2019 in Baltimore

Sponsorship opportunities open

*A main goal in designing a language should be to plan for growth. The language must start small, and the language must grow as the set of users grows.*

Guy Steele, “Growing a language”, 1998

# An Introduction to Zygote: Linear Regression

In this notebook, we will define Linear Regression in Zygote from scratch, showing how easy it is to take derivatives of custom code.

In [1]:

```julia
# Initialize environment in current directory, to load
import Pkg; Pkg.activate(@__DIR__); Pkg.instantiate()
using Zygote, LinearAlgebra
```

This example will showcase how we do a simple linear fit with Zygote, making use of complex datastructures, a home-grown stochastic gradient descent optimizer, and some good old-fashioned math. We start with the problem statement: We wish to learn the mapping `f(X) -> Y`, where `X` is a matrix of vector observations, `f()` is a linear mapping function and `Y` is a vector of scalar observations.

Because we like complex objects, we will define our linear regression as the following object:

In [2]:

```julia
# LinearRegression object, containing multiple fields, some of which will be learned.
mutable struct LinearRegression
    # These values will be implicitly learned
    weights::Matrix
    bias::Float64

    # These values will not be learned
    name::String
end
```

localhost:8889/notebooks/Zygote%20Linear%20Regression.ipynb

jupyter Zygote Linear Regression Last Checkpoint: 12 hours ago (unsaved changes)

This example will showcase how we do a simple linear fit with Zygote, making use of complex datastructures, a home-grown stochastic gradient descent optimizer, and some good old-fashioned math. We start with the problem statement: We wish to learn the mapping `f(X) -> Y`, where `X` is a matrix of vector observations, `f()` is a linear mapping function and `Y` is a vector of scalar observations.

Because we like complex objects, we will define our linear regression as the following object:

In [2]:

```julia
# LinearRegression object, containing multiple fields, some of which will be learned.
mutable struct LinearRegression
    # These values will be implicitly learned
    weights::Matrix
    bias::Float64

    # These values will not be learned
    name::String
end

LinearRegression(nparams, name) = LinearRegression(randn(1, nparams), 0.0, name)
```

Out[2]: LinearRegression

We will define two verbs to act upon a `LinearRegression` object; `predict()`, to perform the linear regression, and `loss()` to measure the $\ell_2$ norm between a target and our current prediction.

In [3]:

```julia
# Our linear regression looks very familiar; w*X + b
function predict(model::LinearRegression, X)
    return model.weights * X .+ model.bias
end
```

Chrome File Edit View History Bookmarks People Window Help Charged 11:05 Mon 03

localhost:8889/notebooks/Zygote%20Linear%20Regression.ipynb

jupyter Zygote Linear Regression Last Checkpoint: 12 hours ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Help Trusted Julia 1.2.0-rc1

Run Code

In [8]:

```julia
# Now we begin our "training loop", where we take examples from `X`,
# calculate loss with respect to the corresponding entry in `Y`, find the
# gradient upon our model, update the model, and continue.  Before we jump
# in, let's look at what `Zygote.gradient()` gives us:
model = LinearRegression(size(X, 1), "Example")

# Calculate gradient upon `model` for the first example in our training set
grads = Zygote.gradient(model) do m
    return loss(m, X[:,1], Y[1])
end
```

Out[8]:

```
(Base.RefValue{Any}((weights = [-1.192026114794742 -0.482340240894484 1.5586551748882012 -0.5581118308574825], bias =
1.0, name = nothing)),)
```

The `grads` object is a Tuple containing one element per argument to `gradient()`, so we take the first one to get the gradient upon `model`:

In [16]:

```julia
grads = grads[1]
```

Out[16]:

```
Base.RefValue{Any}((weights = [0.597277958228296 -1.41050189509322 0.5269798784518928 0.24937720677402483], bias = -
1.0, name = nothing))
```

Because our LinearRegression object is mutable, the gradient holds a reference to it, which we peel via `grads[]`:

In [17]:

```julia
grads = grads[]
```

Out[17]:

```
(weights = [0.597277958228296 -1.41050189509322 0.5269798784518928 0.24937720677402483], bias = -1.0, name = nothing)
```

Chrome File Edit View History Bookmarks People Window Help Charged 11:06 Mon 03

localhost:8889/notebooks/Zygote%20Linear%20Regression.ipynb

jupyter Zygote Linear Regression Last Checkpoint: 12 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Help Trusted Julia 1.2.0-rc1

Run Code

```julia
weights_gt = [1.0, 2.7, 0.3, 1.2]
bias_gt = 0.4

# Generate a dataset of many observations
X = randn(length(weights_gt), 10000)
Y = weights_gt * X .+ bias_gt

# Add a little bit of noise to `X` so that we do not have an exact solution,
# but must instead do a least-squares fit:
X .+= 0.001.*randn(size(X))
```

Out[6]:

```
4×10000 Array{Float64,2}:
 -1.19203    1.91616     0.843466  …  -1.57857    0.246021  -0.597997
 -0.48234    1.32798    -0.899407      0.043473   1.44814   -0.89648
  1.55866    0.0718622   1.89751       0.994072   0.638415  -0.273722
 -0.558112  -1.68074     0.954623      0.442166  -0.187531  -0.92237
```

In [8]:

```julia
# Now we begin our "training loop", where we take examples from `X`,
# calculate loss with respect to the corresponding entry in `Y`, find the
# gradient upon our model, update the model, and continue.  Before we jump
# in, let's look at what `Zygote.gradient()` gives us:
model = LinearRegression(size(X, 1), "Example")

# Calculate gradient upon `model` for the first example in our training set
grads = Zygote.gradient(model) do m
    return loss(m, X[:,1], Y[1])
end
```

Out[8]:

```
(Base.RefValue{Any}((weights = [-1.192026114794742 -0.482340240894484 1.5586551748882012 -0.5581118308574825], bias =
1.0, name = nothing)),)
```

localhost:8889/notebooks/Zygote%20Linear%20Regression.ipynb

jupyter Zygote Linear Regression Last Checkpoint: 12 hours ago (autosaved)

File Edit View Insert Cell Kernel Help Trusted Julia 1.2.0-rc1

```julia
LinearRegression(nparams, name) = LinearRegression(randn(1, nparams), 0.0, name)
```

Out[4]: LinearRegression

We will define two verbs to act upon a `LinearRegression` object; `predict()`, to perform the linear regression, and `loss()` to measure the $\ell_2$ norm between a target and our current prediction.

In [5]:

```julia
# Our linear regression looks very familiar; w*X + b
function predict(model::LinearRegression, X)
    return model.weights * X .+ model.bias
end

# Our "loss" that must be minimized is the l2 norm between our current
# prediction and our ground-truth Y
function loss(model::LinearRegression, X, Y)
    return norm(predict(model, X) .- Y, 2)
end

```

Out[5]: loss (generic function with 1 method)

In [6]:

```julia
# Our "ground truth" values (that we will learn, to prove that this works)
weights_gt = [1.0, 2.7, 0.3, 1.2]'
bias_gt = 0.4

# Generate a dataset of many observations
X = randn(length(weights_gt), 10000)
Y = weights_gt * X .+ bias_gt
```

Chrome File Edit View History Bookmarks People Window Help Charged 11:06 Mon 03

localhost:8889/notebooks/Zygote%20Linear%20Regression.ipynb

jupyter Zygote Linear Regression Last Checkpoint: 12 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Help Trusted Julia 1.2.0-rc1

Run Code

```
 1.55866    0.0718622   1.89751     0.994072   0.638415  -0.273722
-0.558112  -1.68074     0.954623    0.442166  -0.187531  -0.92237
```

In [8]:

```
# Now we begin our "training loop", where we take examples from `X`,
# calculate loss with respect to the corresponding entry in `Y`, find the
# gradient upon our model, update the model, and continue.  Before we jump
# in, let's look at what `Zygote.gradient()` gives us:
model = LinearRegression(size(X, 1), "Example")

# Calculate gradient upon `model` for the first example in our training set
grads = Zygote.gradient(model) do m
    return loss(m, X[:,1], Y[1])
end
```

Out[8]:

```
(Base.RefValue{Any}((weights = [-1.192026114794742 -0.482340240894484 1.5586551748882012 -0.5581118308574825], bias =
1.0, name = nothing)),)
```

The `grads` object is a Tuple containing one element per argument to `gradient()`, so we take the first one to get the gradient upon `model`:

In [16]:

```
grads = grads[1]
```

Out[16]:

```
Base.RefValue{Any}((weights = [0.597277958228296 -1.41050189509322 0.5269798784518928 0.24937720677402483], bias = -
1.0, name = nothing))
```

Because our LinearRegression object is mutable, the gradient holds a reference to it, which we peel via `grads[]`:

In [17]:

```
grads = grads[]
```

# Zygote.jl: General Purpose Automatic Differentiation

```
function foo(W, Y, x)
    Z = W * Y
    a = Z * x
    b = Y * x
    c = tanh.(b)
    r = a + c
    return r
end
```

Zygote

```
function ∇foo(W, Y, x)
    Z = W * Y
    a = Z * x
    b = Y * x
    c, 𝒥tanh = ∇tanh.(b)
    a + c, function (Δr)
        Δc = Δr, Δa = Δr
        (Δtanh, Δb) = 𝒥tanh(Δc)
        (ΔY, Δx) = (Δb * x', Y' * Δb)
        (ΔZ = Δa * x', Δx += Z' * Δa)
        (ΔW = ΔZ * Y', ΔY = W' * ΔZ')
        (nothing, ΔW, ΔY, Δx)
    end
end
```

*M. Innes. Don't Unroll Adjoint: Differentiating SSA-Form Programs (arXiv:1810.07951)*

# Celeste.jl: Julia at Peta-scale

Cori: 650,000 cores. 1.3M threads. 60 TB of data.

Most light sources are near the detection limit.

**Cataloging the Visible Universe through Bayesian Inference at Petascale**

Jeffrey Regier[*], Kiran Pamnany[†], Keno Fischer[‡], Andreas Noack[§], Maximilian Lam[*], Jarrett Revels[§], Steve Howard[¶], Ryan Giordano[¶], David Schlegel[||], Jon McAuliffe[¶], Rollin Thomas[||], Prabhat[||]

[*]*Department of Electrical Engineering and Computer Sciences, University of California, Berkeley*
[†]*Parallel Computing Lab, Intel Corporation*
[‡]*Julia Computing*
[§]*Computer Science and AI Laboratories, Massachusetts Institute of Technology*
[¶]*Department of Statistics, University of California, Berkeley*
[||]*Lawrence Berkeley National Laboratory*

# Zygote.jl: General Purpose Automatic Differentiation

```
function foo(W, Y, x)
    Z = W * Y
    a = Z * x
    b = Y * x
    c = tanh.(b)
    r = a + c
    return r
end
```

Zygote

```
function ∇foo(W, Y, x)
    Z = W * Y
    a = Z * x
    b = Y * x
    c, 𝒥tanh = ∇tanh.(b)
    a + c, function (Δr)
        Δc = Δr, Δa = Δr
        (Δtanh, Δb) = 𝒥tanh(Δc)
        (ΔY, Δx) = (Δb * x', Y' * Δb)
        (ΔZ = Δa * x', Δx += Z' * Δa)
        (ΔW = ΔZ * Y', ΔY = W' * ΔZ')
        (nothing, ΔW, ΔY, Δx)
    end
end
```

*M. Innes. Don't Unroll Adjoint: Differentiating SSA-Form Programs (arXiv:1810.07951)*

Chrome File Edit View History Bookmarks People Window Help 11:11 Mon 03

localhost:8889/notebooks/Zygote%20Linear%20Regression.ipynb

jupyter Zygote Linear Regression Last Checkpoint: 12 hours ago (autosaved)

File Edit View Insert Cell Kernel Help Trusted Julia 1.2.0-rc1

```julia
# gradient upon our model, update the model, and continue.  Before we jump
# in, let's look at what `Zygote.gradient()` gives us:
model = LinearRegression(size(X, 1), "Example")

# Calculate gradient upon `model` for the first example in our training set
grads = Zygote.gradient(model) do m
    return loss(m, X[:,1], Y[1])
end
```

Out[8]: (Base.RefValue{Any}((weights = [-1.192026114794742 -0.482340240894484 1.5586551748882012 -0.5581118308574825], bias = 1.0, name = nothing)),)

The `grads` object is a Tuple containing one element per argument to `gradient()`, so we take the first one to get the gradient upon `model`:

In [16]:
```julia
grads = grads[1]
```

Out[16]: Base.RefValue{Any}((weights = [0.597277958228296 -1.41050189509322 0.5269798784518928 0.24937720677402483], bias = -1.0, name = nothing))

Because our LinearRegression object is mutable, the gradient holds a reference to it, which we peel via `grads[]`:

In [17]:
```julia
grads = grads[]
```

Out[17]: (weights = [0.597277958228296 -1.41050189509322 0.5269798784518928 0.24937720677402483], bias = -1.0, name = nothing)

We now get a `NamedTuple` so we can now do things like `grads.weights`. Note that while `weights` and `bias` have gradients, `name` just naturally has a gradient of `nothing`, because it was not involved in the calculation of the output loss.

Chrome File Edit View History Bookmarks People Window Help Charged 11:11 Mon 03

localhost:8889/notebooks/Zygote%20Linear%20Regression.ipynb

jupyter Zygote Linear Regression Last Checkpoint: 12 hours ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help Trusted Julia 1.2.0-rc1

Run Code

```
# Calculate gradient upon `model` for the first example in our training set
grads = Zygote.gradient(model) do m
    return loss(m, X[:,1], Y[1])
end
```

Out[8]:
```
(Base.RefValue{Any}((weights = [-1.192026114794742 -0.482340240894484 1.5586551748882012 -0.5581118308574825], bias = 1.0, name = nothing)),)
```

The `grads` object is a Tuple containing one element per argument to `gradient()`, so we take the first one to get the gradient upon `model`:

In [9]:
```
grads = grads[1]
```

Out[9]:
```
Base.RefValue{Any}((weights = [-1.192026114794742 -0.482340240894484 1.5586551748882012 -0.5581118308574825], bias = 1.0, name = nothing))
```

Because our LinearRegression object is mutable, the gradient holds a reference to it, which we peel via `grads[]`:

In [17]:
```
grads = grads[]
```

Out[17]:
```
(weights = [0.597277958228296 -1.41050189509322 0.5269798784518928 0.24937720677402483], bias = -1.0, name = nothing)
```

We now get a `NamedTuple` so we can now do things like `grads.weights`. Note that while `weights` and `bias` have gradients, `name` just naturally has a gradient of `nothing`, because it was not involved in the calculation of the output loss.

In [18]:
```
grads.weights
```

Out[18]:
```
1×4 Array{Float64,2}:
```

Chrome File Edit View History Bookmarks People Window Help

localhost:8889/notebooks/Zygote%20Linear%20Regression.ipynb

jupyter Zygote Linear Regression Last Checkpoint: 12 hours ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help Trusted Julia 1.2.0-rc1

Run Code

In [9]:
```
grads = grads[1]
```

Out[9]:
```
Base.RefValue{Any}((weights = [-1.192026114794742 -0.482340240894484 1.5586551748882012 -0.5581118308574825], bias =
1.0, name = nothing))
```

Because our LinearRegression object is mutable, the gradient holds a reference to it, which we peel via `grads[]`:

In [10]:
```
grads = grads[]
```

Out[10]:
```
(weights = [-1.192026114794742 -0.482340240894484 1.5586551748882012 -0.5581118308574825], bias = 1.0, name = nothin
g)
```

We now get a `NamedTuple` so we can now do things like `grads.weights`. Note that while `weights` and `bias` have gradients, `name` just naturally has a gradient of `nothing`, because it was not involved in the calculation of the output loss.

In [18]:
```
grads.weights
```

Out[18]:
```
1×4 Array{Float64,2}:
 0.597278  -1.4105  0.52698  0.249377
```

Next, we will define an update rule that will allow us to modify the weights of our model according to the gradients, using the simplest gradient descent update rule. We'll then run a training loop to update our weights with the loss from the training set, as we would expect:

In [21]:
```
# Let's define
function sgd_update!(model::LinearRegression, grads, η = 0.001)
    model.weights .-= η .* grads.weights
```

localhost:8889/notebooks/Zygote%20Linear%20Regression.ipynb

jupyter Zygote Linear Regression Last Checkpoint: 12 hours ago (unsaved changes)

Out[10]:
```
(weights = [-1.192026114794742 -0.482340240894484 1.5586551748882012 -0.5581118308574825], bias = 1.0, name = nothin
g)
```

We now get a `NamedTuple` so we can now do things like `grads.weights`. Note that while `weights` and `bias` have gradients, `name` just naturally has a gradient of `nothing`, because it was not involved in the calculation of the output loss.

In [11]:
```julia
grads.weights
```

Out[11]:
```
1×4 Array{Float64,2}:
 -1.19203  -0.48234  1.55866  -0.558112
```

Next, we will define an update rule that will allow us to modify the weights of our model according to the gradients, using the simplest gradient descent update rule. We'll then run a training loop to update our weights with the loss from the training set, as we would expect:

In [21]:
```julia
# Let's define
function sgd_update!(model::LinearRegression, grads, η = 0.001)
    model.weights .-= η .* grads.weights
    model.bias -= η * grads.bias
end
```

Out[21]:
```
sgd_update! (generic function with 2 methods)
```

In [22]:
```julia
# Now let's do that for each example in our training set:
@info("Running train loop for $(size(X,2)) iterations")
for idx in 1:size(X, 2)
    grads = Zygote.gradient(m -> loss(m, X[:, idx], Y[idx]), model)[1][]
```

localhost:8889/notebooks/Zygote%20Linear%20Regression.ipynb

jupyter Zygote Linear Regression Last Checkpoint: 12 hours ago (unsaved changes)

In [11]:
```
grads.weights
```

Out[11]:
```
1×4 Array{Float64,2}:
 -1.19203  -0.48234  1.55866  -0.558112
```

Next, we will define an update rule that will allow us to modify the weights of our model according to the gradients, using the simplest gradient descent update rule. We'll then run a training loop to update our weights with the loss from the training set, as we would expect:

In [21]:
```
# Let's define
function sgd_update!(model::LinearRegression, grads, η = 0.001)
    model.weights .-= η .* grads.weights
    model.bias -= η * grads.bias
end
```

Out[21]:
```
sgd_update! (generic function with 2 methods)
```

In [22]:
```
# Now let's do that for each example in our training set:
@info("Running train loop for $(size(X,2)) iterations")
for idx in 1:size(X, 2)
    grads = Zygote.gradient(m -> loss(m, X[:, idx], Y[idx]), model)[1][]
    sgd_update!(model, grads)
end
```

```
┌ Info: Running train loop for 10000 iterations
└ @ Main In[22]:2
```

In [23]:
```
weights_gt
```

Chrome File Edit View History Bookmarks People Window Help Charged 11:12 Mon 03

localhost:8889/notebooks/Zygote%20Linear%20Regression.ipynb

jupyter Zygote Linear Regression Last Checkpoint: 12 hours ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Help Trusted Julia 1.2.0-rc1

Run Code

In [12]:
```
# Let's define
function sgd_update!(model::LinearRegression, grads, η = 0.001)
    model.weights .-= η .* grads.weights
    model.bias -= η * grads.bias
end
```

Out[12]: sgd_update! (generic function with 2 methods)

In [22]:
```
# Now let's do that for each example in our training set:
@info("Running train loop for $(size(X,2)) iterations")
for idx in 1:size(X, 2)
    grads = Zygote.gradient(m -> loss(m, X[:, idx], Y[idx]), model)[1][]
    sgd_update!(model, grads)
end
```

```
┌ Info: Running train loop for 10000 iterations
└ @ Main In[22]:2
```

In [23]:
```
weights_gt
```

Out[23]:
```
1×4 Adjoint{Float64,Array{Float64,1}}:
 1.0  2.7  0.3  1.2
```

In [24]:
```
model.weights
```

Out[24]:
```
1×4 Array{Float64,2}:
 0.999031  2.69703  0.301551  1.20023
```

In [25]:
```
bias_gt
```

Chrome File Edit View History Bookmarks People Window Help Charged 11:12 Mon 03

localhost:8889/notebooks/Zygote%20Linear%20Regression.ipynb

jupyter Zygote Linear Regression Last Checkpoint: 12 hours ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Help Trusted Julia 1.2.0-rc1

Run Code

In [12]:

```julia
# Let's define
function sgd_update!(model::LinearRegression, grads, η = 0.001)
    model.weights .-= η .* grads.weights
    model.bias -= η * grads.bias
end
```

Out[12]: sgd_update! (generic function with 2 methods)

In [*]:

```julia
# Now let's do that for each example in our training set:
@info("Running train loop for $(size(X,2)) iterations")
for idx in 1:size(X, 2)
    grads = Zygote.gradient(m -> loss(m, X[:, idx], Y[idx]), model)[1][]
    sgd_update!(model, grads)
end
```

In [23]:

```julia
weights_gt
```

Out[23]:

```
1×4 Adjoint{Float64,Array{Float64,1}}:
 1.0  2.7  0.3  1.2
```

In [24]:

```julia
model.weights
```

Out[24]:

```
1×4 Array{Float64,2}:
 0.999031  2.69703  0.301551  1.20023
```

In [25]:

```julia
bias_gt
```

Out[25]: 0.4

Chrome File Edit View History Bookmarks People Window Help Charged 11:12 Mon 03

localhost:8889/notebooks/Zygote%20Linear%20Regression.ipynb

jupyter **Zygote Linear Regression** Last Checkpoint: 12 hours ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Help Trusted Julia 1.2.0-rc1

Run Code

In [12]:
```julia
# Let's define
function sgd_update!(model::LinearRegression, grads, η = 0.001)
    model.weights .-= η .* grads.weights
    model.bias -= η * grads.bias
end
```

Out[12]:
```
sgd_update! (generic function with 2 methods)
```

In [13]:
```julia
# Now let's do that for each example in our training set:
@info("Running train loop for $(size(X,2)) iterations")
for idx in 1:size(X, 2)
    grads = Zygote.gradient(m -> loss(m, X[:, idx], Y[idx]), model)[1][]
    sgd_update!(model, grads)
end
```

```
┌ Info: Running train loop for 10000 iterations
└ @ Main In[13]:2
```

In [23]:
```julia
weights_gt
```

Out[23]:
```
1×4 Adjoint{Float64,Array{Float64,1}}:
 1.0  2.7  0.3  1.2
```

In [24]:
```julia
model.weights
```

Out[24]:
```
1×4 Array{Float64,2}:
 0.999031  2.69703  0.301551  1.20023
```

In [25]:
```julia
bias_gt
```

Chrome File Edit View History Bookmarks People Window Help Charged 11:12 Mon 03

localhost:8889/notebooks/Zygote%20Linear%20Regression.ipynb

jupyter Zygote Linear Regression Last Checkpoint: 12 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Help Trusted Julia 1.2.0-rc1

Run Code

```
    model.bias -= η * grads.bias
end
```

Out[12]: sgd_update! (generic function with 2 methods)

In [13]:

```
# Now let's do that for each example in our training set:
@info("Running train loop for $(size(X,2)) iterations")
for idx in 1:size(X, 2)
    grads = Zygote.gradient(m -> loss(m, X[:, idx], Y[idx]), model)[1][]
    sgd_update!(model, grads)
end
```

```
┌ Info: Running train loop for 10000 iterations
└ @ Main In[13]:2
```

In [23]:

```
weights_gt
```

Out[23]:

```
1×4 Adjoint{Float64,Array{Float64,1}}:
 1.0  2.7  0.3  1.2
```

In [24]:

```
model.weights
```

Out[24]:

```
1×4 Array{Float64,2}:
 0.999031  2.69703  0.301551  1.20023
```

In [25]:

```
bias_gt
```

Out[25]: 0.4

Chrome File Edit View History Bookmarks People Window Help Charged 11:12 Mon 03

localhost:8889/notebooks/Zygote%20Linear%20Regression.ipynb

jupyter Zygote Linear Regression Last Checkpoint: 12 hours ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help Trusted Julia 1.2.0-rc1

Run Code

Out[12]: sgd_update! (generic function with 2 methods)

In [13]:
```
# Now let's do that for each example in our training set:
@info("Running train loop for $(size(X,2)) iterations")
for idx in 1:size(X, 2)
    grads = Zygote.gradient(m -> loss(m, X[:, idx], Y[idx]), model)[1][]
    sgd_update!(model, grads)
end
```

```
┌ Info: Running train loop for 10000 iterations
└ @ Main In[13]:2
```

In [14]:
```
weights_gt
```

Out[14]:
```
1×4 Adjoint{Float64,Array{Float64,1}}:
 1.0  2.7  0.3  1.2
```

In [24]:
```
model.weights
```

Out[24]:
```
1×4 Array{Float64,2}:
 0.999031  2.69703  0.301551  1.20023
```

In [25]:
```
bias_gt
```

Out[25]: 0.4

In [26]:
```
model.bias
```

Out[26]: 0.39800000000000035

Chrome File Edit View History Bookmarks People Window Help Charged 11:13 Mon 03

localhost:8889/notebooks/Zygote%20Linear%20Regression.ipynb

jupyter Zygote Linear Regression Last Checkpoint: 12 hours ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Help Trusted Julia 1.2.0-rc1

Run Code

```
end
```

```
┌ Info: Running train loop for 10000 iterations
└ @ Main In[13]:2
```

In [14]:
```
weights_gt
```

Out[14]:
```
1×4 Adjoint{Float64,Array{Float64,1}}:
 1.0  2.7  0.3  1.2
```

In [15]:
```
model.weights
```

Out[15]:
```
1×4 Array{Float64,2}:
 1.00142  2.70157  0.300252  1.20033
```

In [16]:
```
bias_gt
```

Out[16]:
```
0.4
```

In [17]:
```
model.bias
```

Out[17]:
```
0.39800000000000003
```

Chrome File Edit View History Bookmarks People Window Help Charged 11:13 Mon 03

localhost:8889/notebooks/Zygote%20Optimizing%20Raytracer.ipynb

jupyter Zygote Optimizing Raytracer Last Checkpoint: 9 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help Trusted Julia 1.2.0-rc1

Run Code

# Zygote Continued: A Differentiable Raytracer

We demonstrate in this notebook differentiating through a raytracer

In [1]:

```julia
# Initialize environment in current directory, to load
import Pkg; Pkg.activate(@__DIR__); Pkg.instantiate()
using RayTracer, Zygote, Flux, Images, Statistics, Interact
```

```
┌ Info: activating environment at `~/src/msr_talk/Project.toml`.
└ @ Pkg.API /Users/sabae/tmp/julia-build/julia-release-1.2/usr/share/julia/stdlib/v1.2/Pkg/src/API.jl:564
  Updating registry at `~/.julia/registries/General`
  Updating git-repo `https://github.com/JuliaRegistries/General.git`
┌ Warning: Some registries failed to update:
│     — /Users/sabae/.julia/registries/General — failed to fetch from repo
└ @ Pkg.Types /Users/sabae/tmp/julia-build/julia-release-1.2/usr/share/julia/stdlib/v1.2/Pkg/src/Types.jl:1171
```

In [2]:

```julia
width = 200
height = 200

# Static camera configuration
cam = Camera(
    # Center
```

localhost:8889/notebooks/Zygote%20Optimizing%20Raytracer.ipynb

jupyter Zygote Optimizing Raytracer Last Checkpoint: 9 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Help — Trusted — Julia 1.2.0-rc1

In [2]:

```julia
width = 200
height = 200

# Static camera configuration
cam = Camera(
    # Center
    Vec3(0.0f0, 0.0f0, -5.0f0),
    # Target
    Vec3(0.0f0, 0.0f0,  0.0f0),
    # Up
    Vec3(0.0f0, 1.0f0, 0.0f0),
    # Field of View
    45.0f0,
    # Focus
    1.0f0,
    # Resolution
    width, height,
)
origin, direction = get_primary_rays(cam)

function render(scene, light)
    packed_image = raytrace(origin, direction, scene, light, origin, 0)
    array_image = reshape(hcat(packed_image.x, packed_image.y, packed_image.z), (width, height, 3, 1))
    return array_image
end

function showimg(img)
    return colorview(RGB, permutedims(img[:,:,:,1], (3,2,1)))
end
```

Chrome File Edit View History Bookmarks People Window Help Charged 11:13 Mon 03

localhost:8889/notebooks/Zygote%20Optimizing%20Raytracer.ipynb

jupyter Zygote Optimizing Raytracer Last Checkpoint: 9 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Help Trusted Julia 1.2.0-rc1

Run Code

In [*]:

```
# Initialize environment in current directory, to load
import Pkg; Pkg.activate(@__DIR__); Pkg.instantiate()
using RayTracer, Zygote, Flux, Images, Statistics, Interact
```

```
┌ Info: activating environment at `~/src/msr_talk/Project.toml`.
└ @ Pkg.API /Users/sabae/tmp/julia-build/julia-release-1.2/usr/share/julia/stdlib/v1.2/Pkg/src/API.jl:564
```

```
Updating registry at `~/.julia/registries/General`
Updating git-repo `https://github.com/JuliaRegistries/General.git`
```

```
┌ Info: Recompiling stale cache file /Users/sabae/.julia/compiled/v1.2/RayTracer/sUryZ.ji for RayTracer [60dacb86-48ff-11e9-0f01-03ab8794bbc9]
└ @ Base loading.jl:1240
```

In [2]:

```
width = 200
height = 200

# Static camera configuration
cam = Camera(
    # Center
    Vec3(0.0f0, 0.0f0, -5.0f0),
    # Target
    Vec3(0.0f0, 0.0f0,  0.0f0),
    # Up
    Vec3(0.0f0, 1.0f0, 0.0f0),
    # Field of View
    45.0f0,
    # Focus
```

Chrome File Edit View History Bookmarks People Window Help Charged 11:14 Mon 03

localhost:8889/notebooks/Zygote%20Optimizing%20Raytracer.ipynb

jupyter **Zygote Optimizing Raytracer** Last Checkpoint: 9 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Help Trusted Julia 1.2.0-rc1

Run Code

In [2]:

```
width = 200
height = 200

# Static camera configuration
cam = Camera(
    # Center
    Vec3(0.0f0, 0.0f0, -5.0f0),
    # Target
    Vec3(0.0f0, 0.0f0,  0.0f0),
    # Up
    Vec3(0.0f0, 1.0f0, 0.0f0),
    # Field of View
    45.0f0,
    # Focus
    1.0f0,
    # Resolution
    width, height,
)
origin, direction = get_primary_rays(cam)

function render(scene, light)
    packed_image = raytrace(origin, direction, scene, light, origin, 0)
    array_image = reshape(hcat(packed_image.x, packed_image.y, packed_image.z), (width, height, 3, 1))
    return array_image
end

function showimg(img)
    return colorview(RGB, permutedims(img[:,:,:,1], (3,2,1)))
end
```

Chrome File Edit View History Bookmarks People Window Help

Charged 11:17 Mon 03

localhost:8889/notebooks/Zygote%20Optimizing%20Raytracer.ipynb

jupyter **Zygote Optimizing Raytracer** Last Checkpoint: 9 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help

Trusted | Julia 1.2.0-rc1

Run | Code

```julia
        loss = mean((zeroonenorm(image_rendered) .- zeroonenorm(image_gt)).^2)

        # Show the current loss
        @show loss

        # Return this loss as what is to be minimized
        return loss
    end

    # Update our light position and triangle color based upon those gradients
    #update!(opt, scene[1].material.color.color, grads[1][1].material.color.color)
    update!(opt, light.position, grads[2].position)

    if i % 10 == 1
        @info "$i iterations completed"
        display(showimg(render(scene, light)))
    end
end

```

In [29]:

```julia
light_gt
```

Out[29]:

```
PointLight{Float32}(Vec3{Array{Float32,1}}(Float32[1.0], Float32[1.0], Float32[0.0]), 20000.0f0, Vec3{Array{Float32,
1}}(Float32[3.6], Float32[3.0], Float32[-10.0]))
```

In [30]:

```julia
light
```

Out[30]:

```
PointLight{Float32}(Vec3{Array{Float32,1}}(Float32[1.0], Float32[1.0], Float32[0.0]), 20000.0f0, Vec3{Array{Float32,
1}}(Float32[4.18166], Float32[2.4189723], Float32[-9.362359]))
```

Chrome File Edit View History Bookmarks People Window Help Charged 11:18 Mon 03

localhost:8889/notebooks/Zygote%20Optimizing%20Raytracer.ipynb

jupyter Zygote Optimizing Raytracer Last Checkpoint: 9 hours ago (unsaved changes)

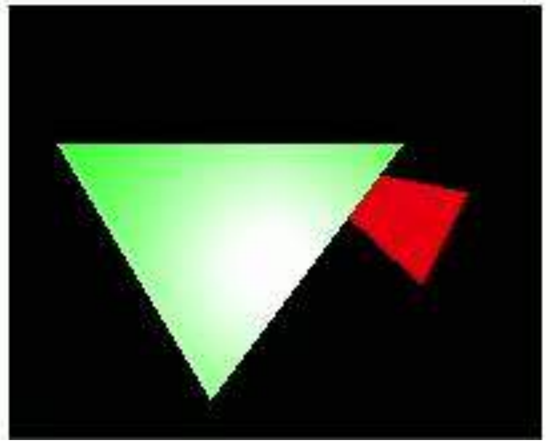
Logout

File Edit View Insert Cell Kernel Help Trusted Julia 1.2.0-rc1

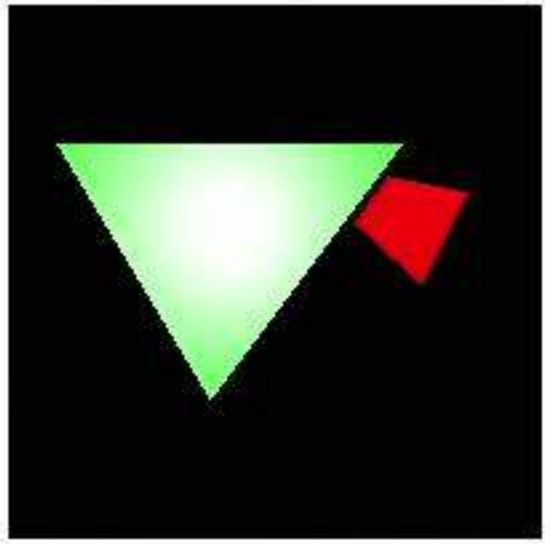
Run Code

```julia

        # Show the current loss
        @show loss

        # Return this loss as what is to be minimized
        return loss
    end

    # Update our light position and triangle color based upon those gradients
    #update!(opt, scene[1].material.color.color, grads[1][1].material.color.color)
    update!(opt, light.position, grads[2].position)

    if i % 10 == 1
        @info "$i iterations completed"
        display(showimg(render(scene, light)))
    end
end

```

```
loss = 0.003516449f0
```



```
loss = 0.0032816264f0
loss = 0.0032044f0
loss = 0.0031147993f0
loss = 0.0030166926f0
loss = 0.0029100836f0
loss = 0.002786581f0
```



```
┌ Info: 11 iterations completed
└ @ Main In[5]:26
```

```
loss = 0.0026639346f0
loss = 0.002522308f0
loss = 0.0023874852f0
loss = 0.0022445768f0
loss = 0.0020946297f0
loss = 0.0019481931f0
loss = 0.001802167f0
loss = 0.0016537956f0
```

Chrome File Edit View History Bookmarks People Window Help Charged 11:20 Mon 03

localhost:8889/notebooks/Zygote%20Optimizing%20Raytracer.ipynb

jupyter **Zygote Optimizing Raytracer** Last Checkpoint: 9 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Help Trusted Julia 1.2.0-rc1

Run Code

In [5]:

```julia
opt = ADAM(0.1)
image_gt = render(scene_gt, light_gt)
showimg(image_gt)

for i in 1:51
    # Take gradient of the following function
    grads = gradient(scene, light) do S, L
        # First, render according to our current light and scene
        image_rendered = render(S, L)

        # Normalize
        loss = mean((zeroonenorm(image_rendered) .- zeroonenorm(image_gt)).^2)

        # Show the current loss
        @show loss

        # Return this loss as what is to be minimized
        return loss
    end

    # Update our light position and triangle color based upon those gradients
    #update!(opt, scene[1].material.color.color, grads[1][1].material.color.color)
    update!(opt, light.position, grads[2].position)

    if i % 10 == 1
        @info "$i iterations completed"
        display(showimg(render(scene, light)))
    end
end

```

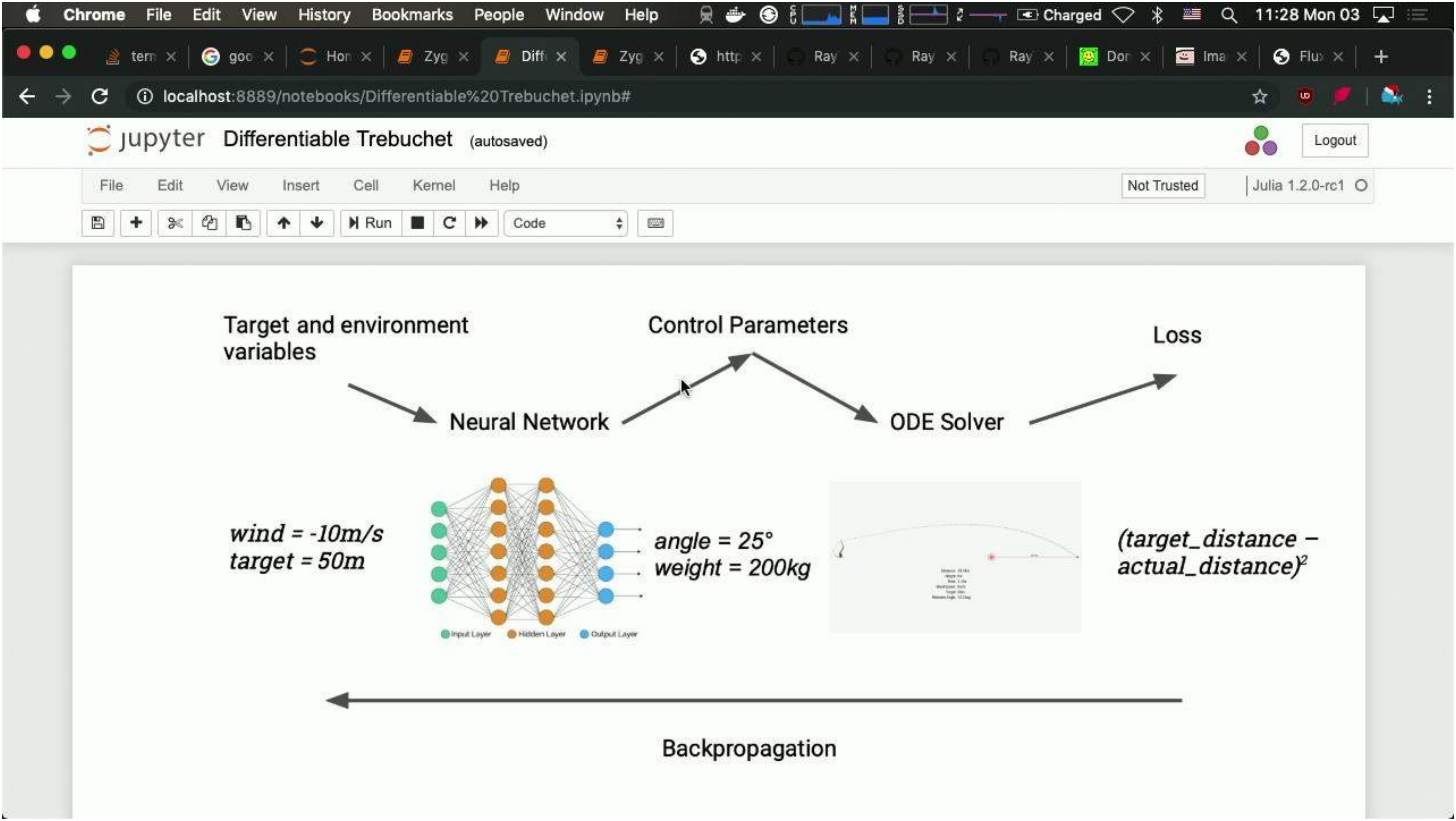
# Zygote.jl: General Purpose Automatic Differentiation

```
function foo(W, Y, x)
    Z = W * Y
    a = Z * x
    b = Y * x
    c = tanh.(b)
    r = a + c
    return r
end
```

Zygote

```
function ∇foo(W, Y, x)
    Z = W * Y
    a = Z * x
    b = Y * x
    c, 𝒥tanh = ∇tanh.(b)
    a + c, function (Δr)
        Δc = Δr, Δa = Δr
        (Δtanh, Δb) = 𝒥tanh(Δc)
        (ΔY, Δx) = (Δb * x', Y' * Δb)
        (ΔZ = Δa * x', Δx += Z' * Δa)
        (ΔW = ΔZ * Y', ΔY = W' * ΔZ')
        (nothing, ΔW, ΔY, Δx)
    end
end
```

*M. Innes. Don't Unroll Adjoint: Differentiating SSA-Form Programs (arXiv:1810.07951)*


localhost:8889/notebooks/Differentiable%20Trebuchet.ipynb#
jupyter Differentiable Trebuchet (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Not Trusted
Julia 1.2.0-rc1
Run
Code
Target and environment variables
Control Parameters
Loss
Neural Network
ODE Solver
wind = -10m/s
target = 50m
Input Layer
Hidden Layer
Output Layer
angle = 25°
weight = 200kg
(target_distance – actual_distance)²
Backpropagation

jupyter Differentiable Trebuchet (autosaved)

Logout

File Edit View Insert Cell Kernel Help Not Trusted Julia 1.2.0-rc1

Run Code

Backpropagation

## Loading your Trebuchet

Today we practice the ancient medieval art of throwing stuff. First up, we load our trebuchet simulator, Trebuchet.jl.

In [1]:

```julia
# Initialize environment in current directory, to load
import Pkg; Pkg.activate(@__DIR__); Pkg.instantiate()
using Trebuchet
```

```
┌ Info: activating environment at `~/src/msr_talk/Project.toml`.
└ @ Pkg.API /Users/sabae/tmp/julia-build/julia-release-1.2/usr/share/julia/stdlib/v1.2/Pkg/src/API.jl:564

  Updating registry at `~/.julia/registries/General`
  Updating git-repo `https://github.com/JuliaRegistries/General.git`

┌ Info: Precompiling Trebuchet [98b73d46-197d-11e9-11eb-69a6ff759d3a]
└ @ Base loading.jl:1242
```

We can see what the trebuchet looks like, by explicitly creating a trebuchet state, running a simulation, and visualising the trajectory.

In [2]:

```julia
t = TrebuchetState()
simulate(t)
```

We can see what the trebuchet looks like, by explicitly creating a trebuchet state, running a simulation, and visualising the trajectory.

In [*]:
```
t = TrebuchetState()
simulate(t)
visualise(t)
```

For training and optimisation, we don't need the whole visualisation, just a simple function that accepts and produces numbers. The `shoot` function just takes a wind speed, angle of release and counterweight mass, and tells us how far the projectile got.

In [3]:
```
function shoot(wind, angle, weight)
    Trebuchet.shoot((wind, Trebuchet.deg2rad(angle), weight))[2]
end
```

Out[3]: `shoot (generic function with 1 method)`

In [4]:
```
shoot(0, 30, 400)
```

Out[4]: `98.60072421662711`

It's worth playing with these parameters to see the impact they have. How far can you throw the projectile, tweaking only the angle of release?

There's actually a much better way of aiming the trebuchet. Let's load up a machine learning library, Flux, and see what we can do.

In [5]:
```
pathof(Trebuchet)
```

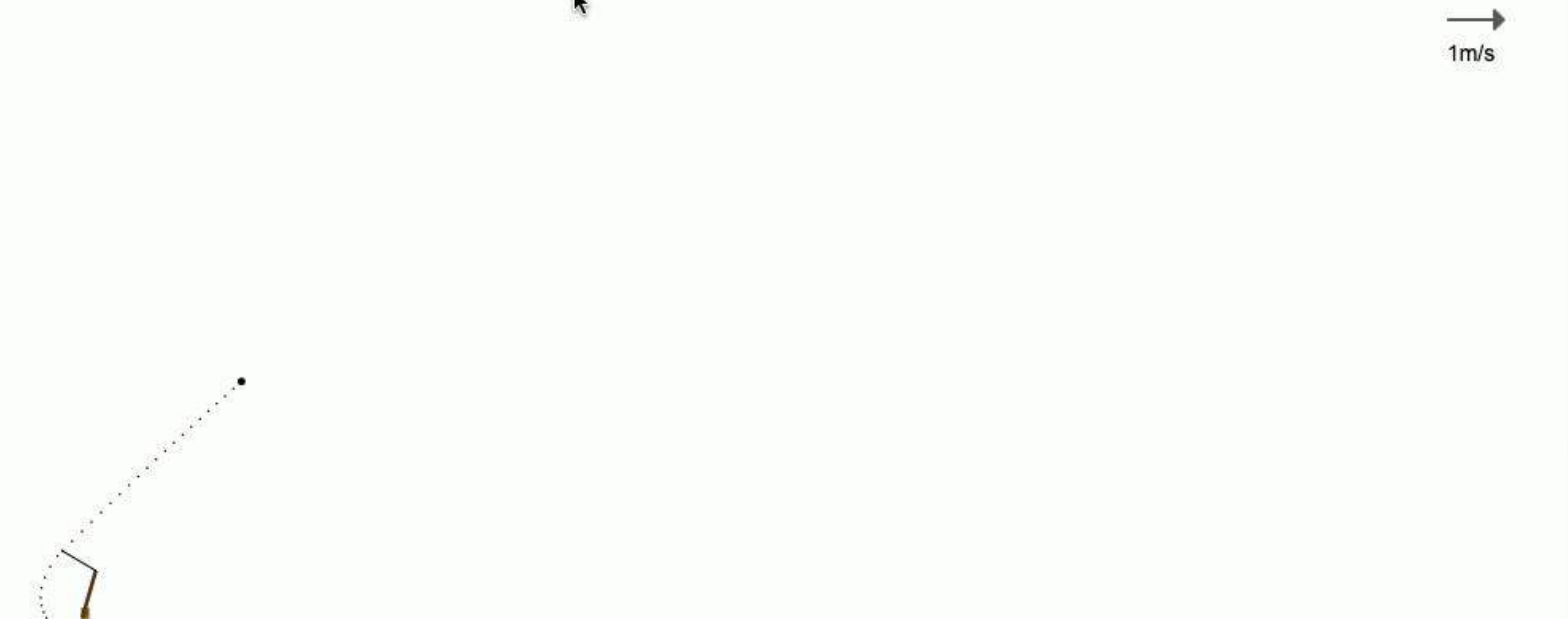
Out[5]: `"/Users/sabae/.julia/packages/Trebuchet/dU16T/src/Trebuchet.jl"`

We can see what the trebuchet looks like, by explicitly creating a trebuchet state, running a simulation, and visualising the trajectory.

In [2]:

```
t = TrebuchetState()
simulate(t)
visualise(t)
```

Out[2]:



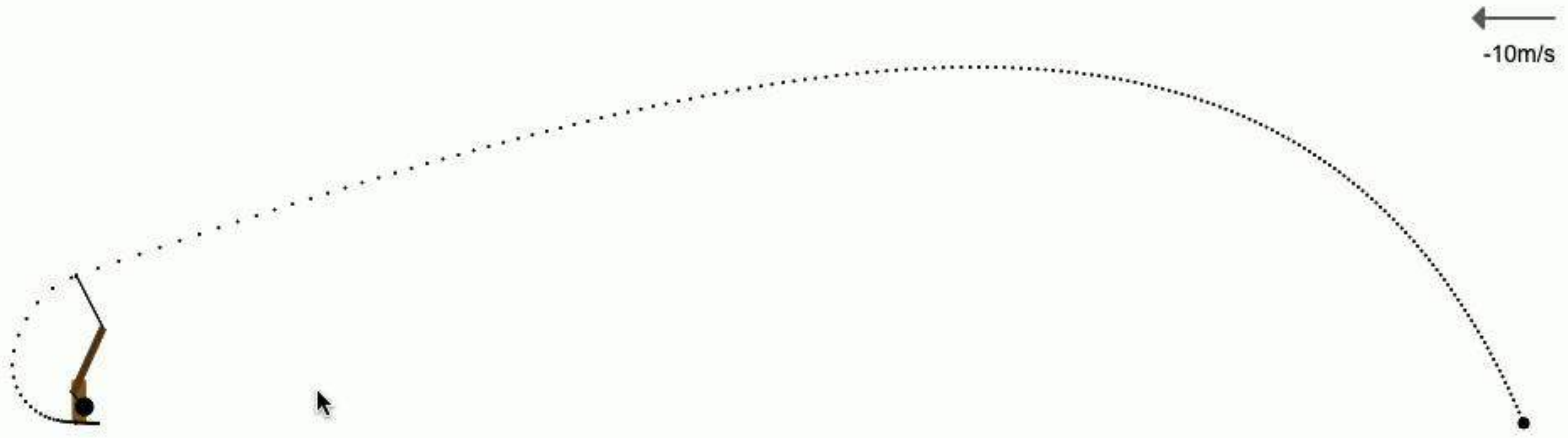
localhost:8889/notebooks/Differentiable%20Trebuchet.ipynb#

File Edit View Insert Cell Kernel Help — Not Trusted — Julia 1.2.0-rc1

In [22]:

```julia
t = TrebuchetState(release_angle = deg2rad(19), wind_speed = -10)
simulate(t)
visualise(t)
```

Out[22]:




| | |
|---|---|
| Distance | 0m |
| Height | 0m |
| Time | 0m |
| Wind Speed | -10m/s |
| Release Angle | 19deg |

Chrome File Edit View History Bookmarks People Window Help

localhost:8889/notebooks/Differentiable%20Trebuchet.ipynb#

File Edit View Insert Cell Kernel Help

Not Trusted Julia 1.2.0-rc1

In [24]:

```
t = TrebuchetState()
simulate(t)
visualise(t, 50)
```

Out[24]: