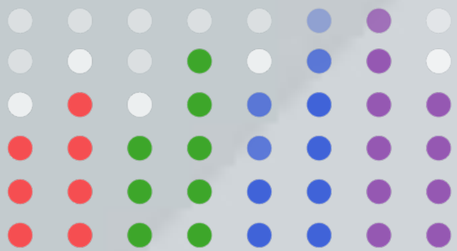
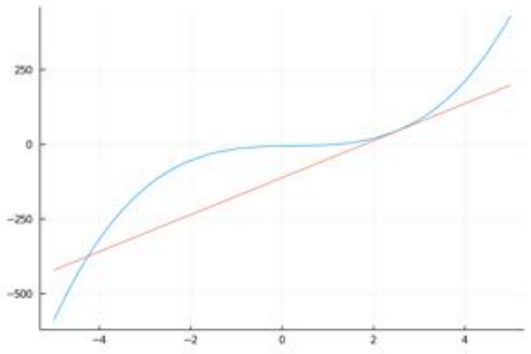


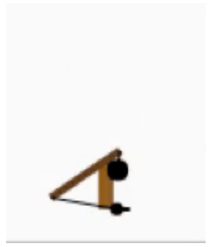
The impact of differentiable programming: how ∂P is enabling new science in Julia

Matt Bauman, Julia Computing

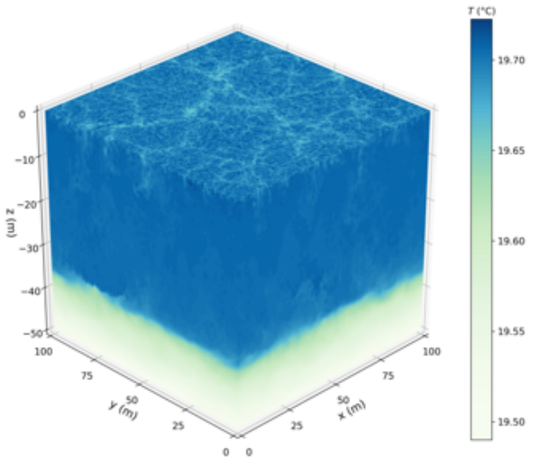
Agenda



- A review of derivatives



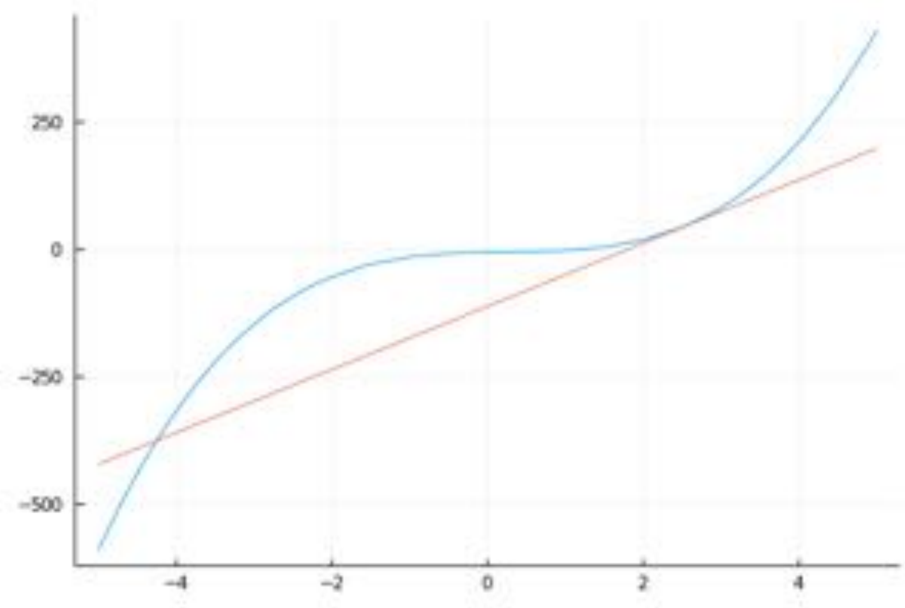
- Throwing things:
the inverse problem



- The real science

Derivatives

```
julia> f(x) = 4x^3 - 3x^2 + 2x - 5
f (generic function with 1 method)
```




```
julia> ∂f(x) = 12x^2 - 6x + 2
∂f (generic function with 1 method)

julia> ∂f(2.5)
62.0
```

Derivatives

```
julia> function g(x)
           if x < 0
               print("Enter function name: ")
               getfield(Base, Symbol(readline()))(x)
           else
               4x^3 - 3x^2 + 2x - 5
           end
       end
g (generic function with 1 method)

julia> g(2)
19
```

Derivatives

```
julia> function g(x)
           if x < 0
               print("Enter function name: ")
               getfield(Base, Symbol(readline()))(x)
           else
               4x^3 - 3x^2 + 2x - 5
           end
       end
g (generic function with 1 method)

julia> g'(2)
38
```

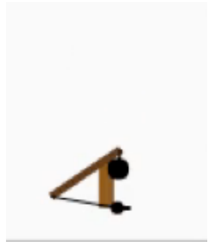
How to aim a trebuchet

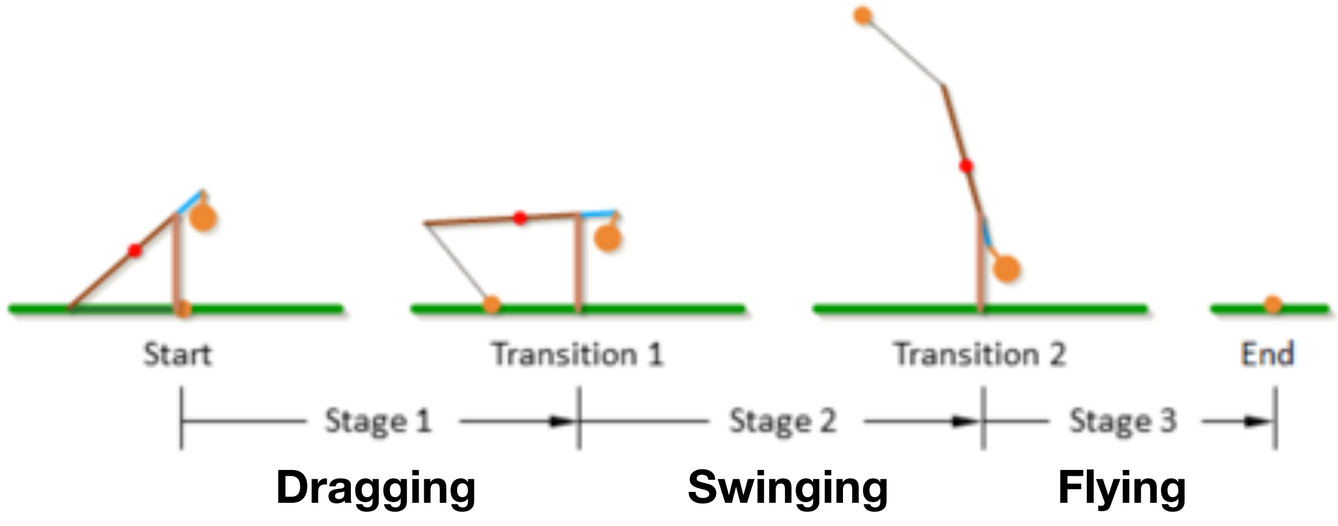
220 kg counterweight
60° release angle
100 m to target

How to aim a trebuchet

220 kg counterweight
55° release angle
100 m to target

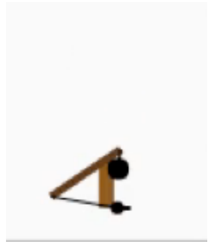
How to simulate a trebuchet

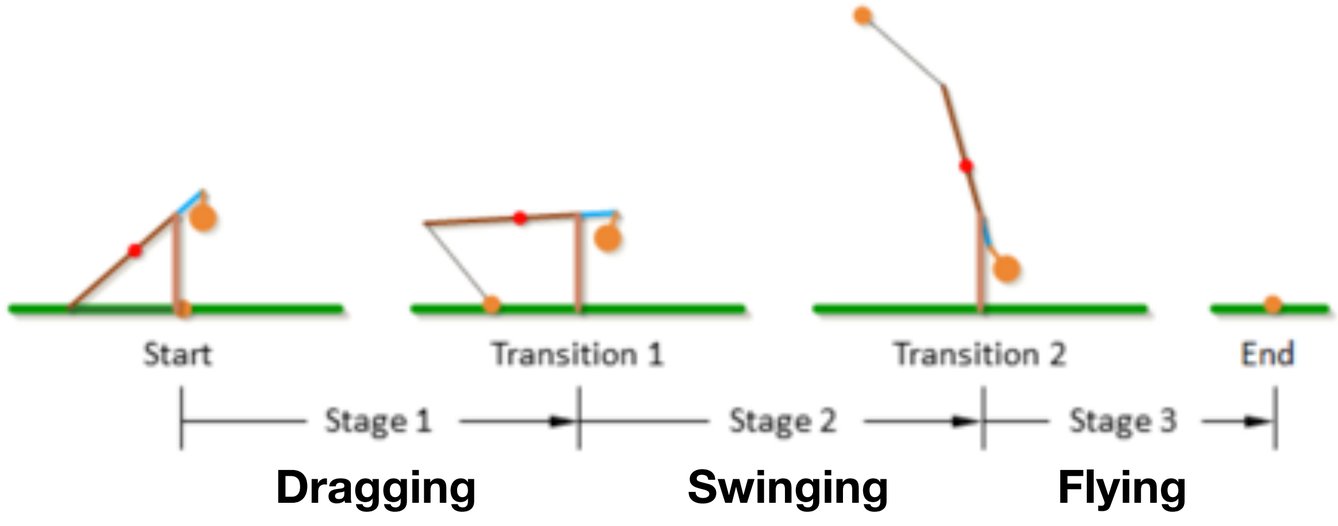


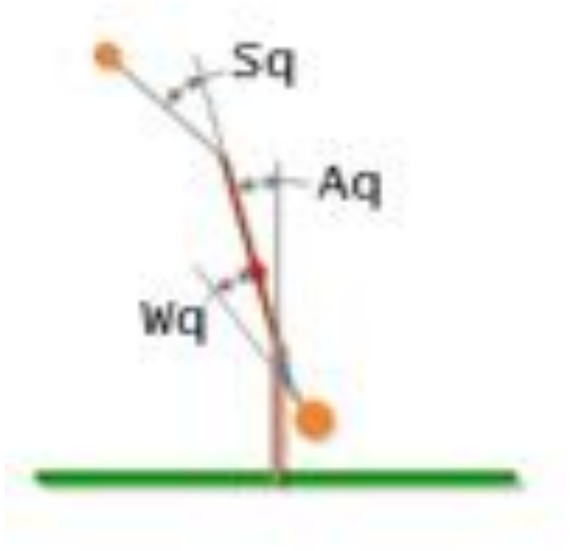


http://www.virtualtrebuchet.com/

How to simulate a trebuchet







$$r_1 = GL_{Acg}m_A\sin(A_q) + Gm_P\left(L_{Al}\sin(A_q) + L_S\sin(A_q + S_q)\right) - Gm_W\left(L_{As}\sin(A_q) + L_W\sin(A_q + W_q)\right)$$
$$-L_{Al}L_Sm_P\sin(S_q)\left(A_w^2 - (A_w + S_w)^2\right) - L_{As}L_Wm_W\sin(W_q)\left(A_w^2 - (A_w + W_w)^2\right)$$

$$r_2 = -L_Wm_W\left(G\sin(A_q + W_q) + L_{As}\sin(W_q)A_w^2\right)$$

$$r_3 = L_Sm_P\left(G\sin(A_q + S_q) - L_{Al}\sin(S_q)A_w^2\right)$$

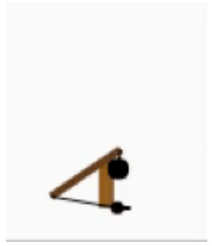
$$A_q'' = -\frac{r_1M_{22}M_{33} - r_2M_{12}M_{33} - r_3M_{13}M_{22}}{M_{13}M_{22}M_{31} - M_{33}\left(M_{11}M_{22} - M_{12}M_{21}\right)}$$

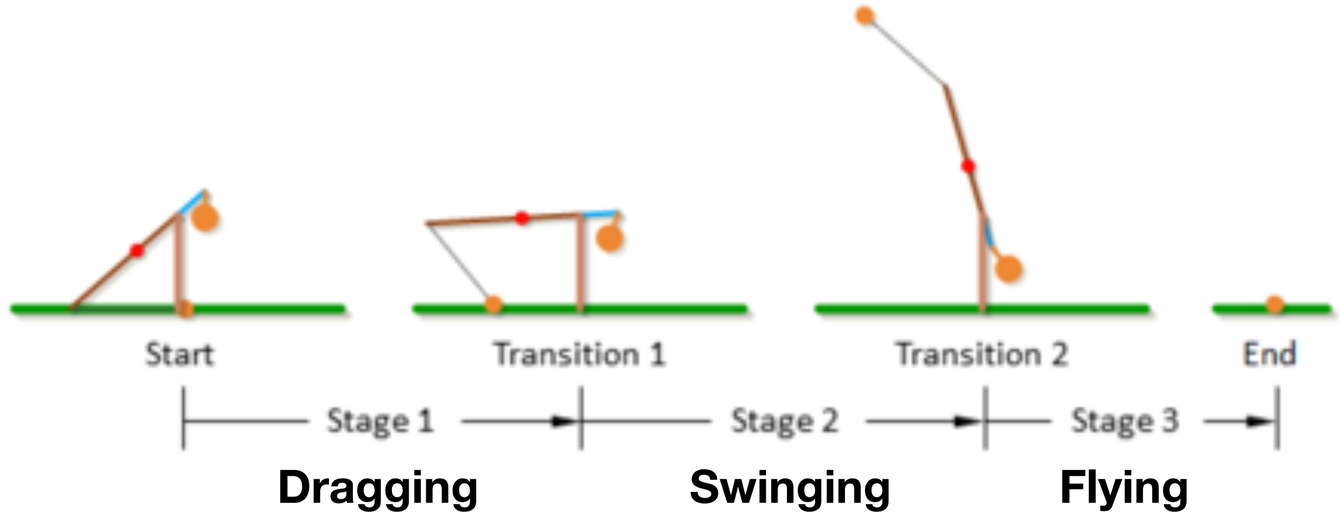
$$W_q'' = \frac{r_1M_{21}M_{33} - r_2\left(M_{11}M_{33} - M_{13}M_{31}\right) - r_3M_{13}M_{21}}{M_{13}M_{22}M_{31} - M_{33}\left(M_{11}M_{22} - M_{12}M_{21}\right)}$$

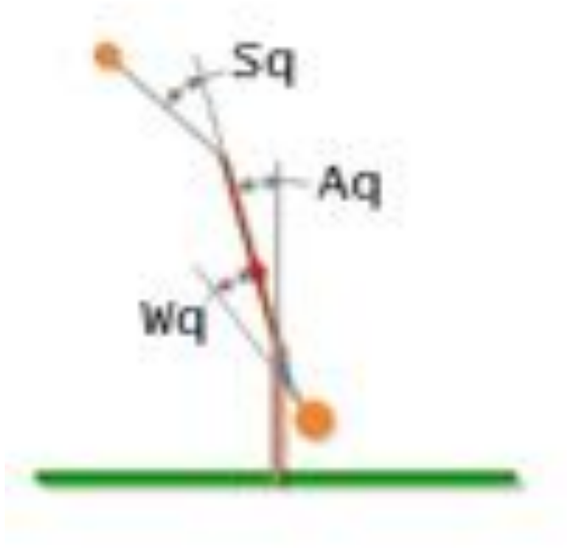
$$S_q'' = \frac{r_1M_{22}M_{31} - r_2M_{12}M_{31} - r_3\left(M_{11}M_{22} - M_{12}M_{21}\right)}{M_{13}M_{22}M_{31} - M_{33}\left(M_{11}M_{22} - M_{12}M_{21}\right)}$$

http://www.virtualtrebuchet.com/

How to simulate a trebuchet








```
(∂Trebuchet) pkg> add Trebuchet
  Resolving package versions...
Updating `~/Projects/∂P/∂Trebuchet/Project.toml`
  [98b73d46] + Trebuchet v0.2.1

julia> using Trebuchet

julia> Trebuchet.shoot((5, deg2rad(55), 220))[2]
93.53880897080582
```

http://www.virtualtrebuchet.com/

How to $\frac{\partial}{\partial x}$ a trebuchet

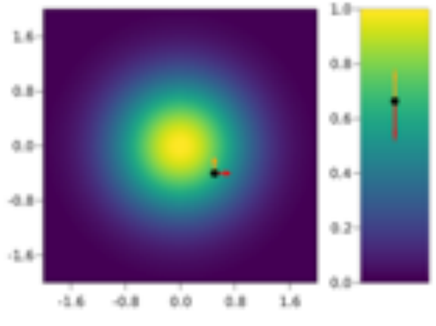
```
julia>          shoot( [5, 55, 220])

93.53880897080582
```

```
julia>          shoot'( [5, 55, 220])

3-element Array{Float64,1}:
  4.296867939687003
 -1.4268043075186865
  0.07697967188505561
```

How to $\frac{\partial}{\partial x}$ a trebuchet

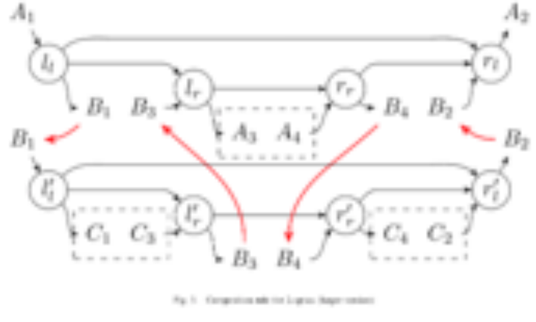
```
julia> @btime shoot($[5, 55, 220])
  2.081 ms (68313 allocations: 1.58 MiB)
93.53880897080582

julia> @btime shoot'($[5, 55, 220])
  2.551 ms (71902 allocations: 3.70 MiB)
3-element Array{Float64,1}:
  4.296867939687003
 -1.4268043075186865
  0.07697967188505561
```

Part I



Differential Geometry

II

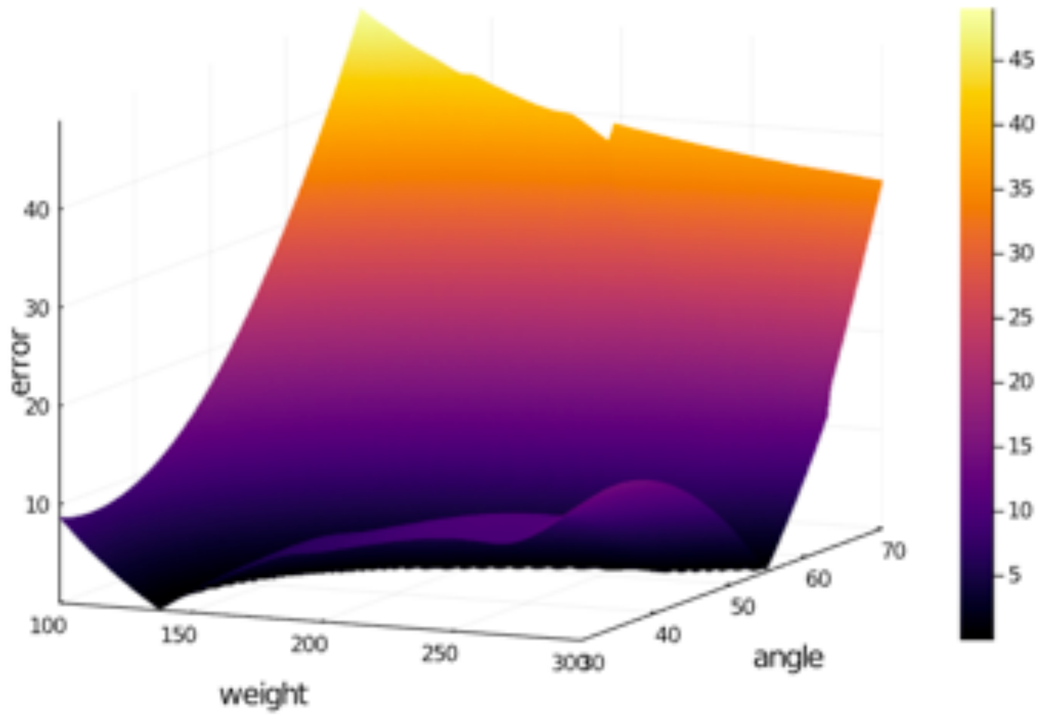


Optical Constructions

III

Compiler Support

How to aim a trebuchet

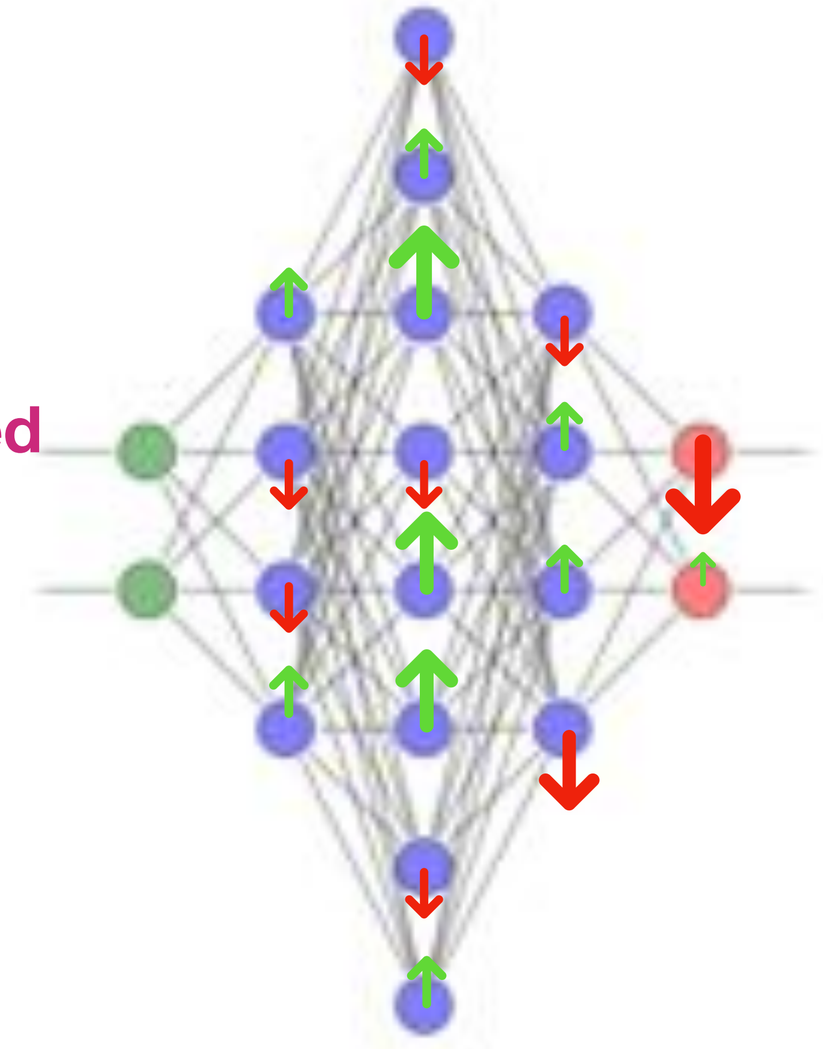



```
# Gradient descent to a single target:
function target!(f, args; N = 10, η = 0.1)
    for _ in 1:N
        grads = f'(args)
        args .-= η .* grads
    end
    return args
end

julia> args = target!(x->(shoot([5, x...])-100)^2,
                      [55., 220.])
2-element Array{Float64,1}:
  50.12142987617123
 220.28075748369307

julia> shoot(5, args...)
99.79941172730138
```

How to *quickly* aim a trebuchet

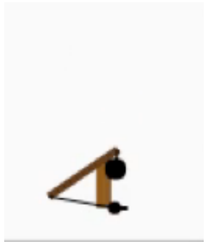
wind speed

target loc



weight →

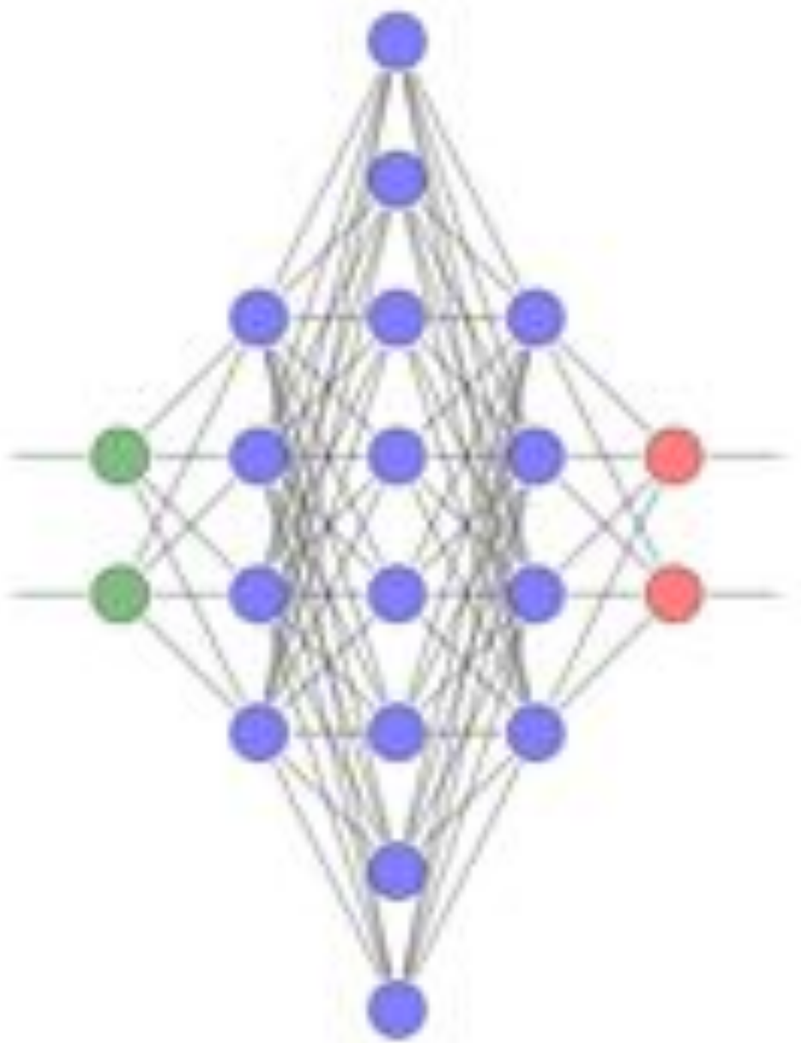
angle →



→ error from arbitrary target

How to *quickly* aim a trebuchet

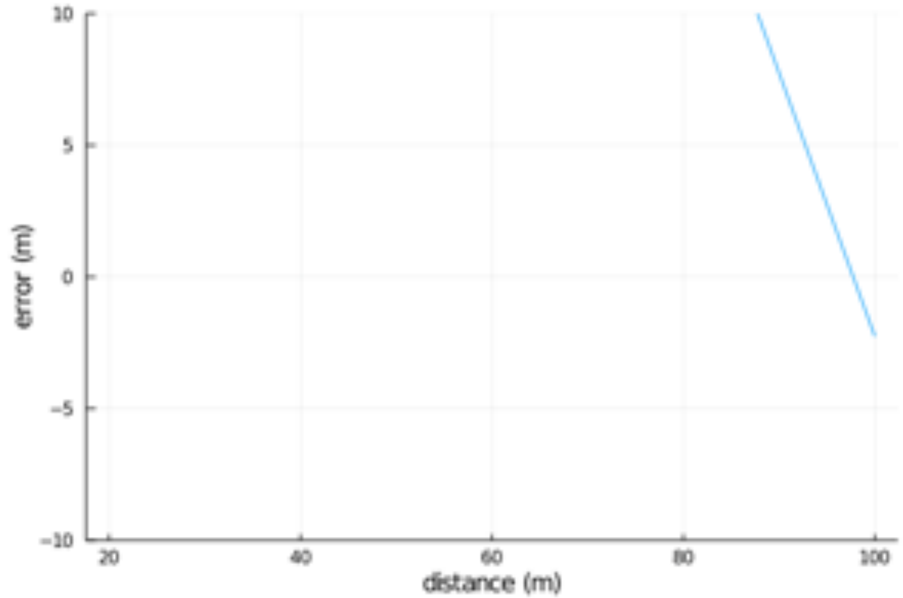
```
julia> aim(5, 100)
(49.65935546993644, 211.10317108580074)

julia> shoot(5, 49.66, 211.1)
99.62600675189528
```



- Once trained, this really is fast:

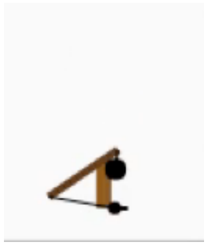
```
julia> @btime aim($5, $100)
  1.733 μs (18 allocations: 2.58 KiB)
(49.65935546993644, 211.10317108580074)
```

- Even the training is fast (under a minute)

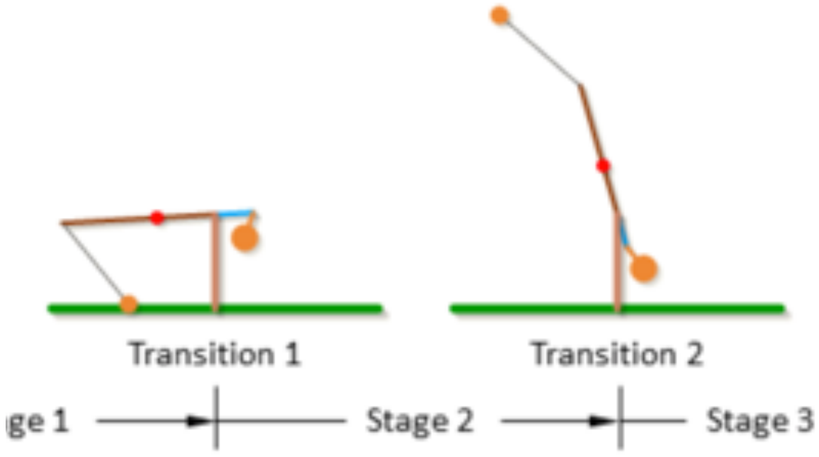


- And fun games can be had with corruption of inputs and controls to add robustness

How to *understand* a trebuchet



- What if I didn't know one of the terms in our system?
- But could describe most everything else?



$$r_1 = GL_{Acg}m_A\sin(A_q) + Gm_P\left(L_{Al}\sin(A_q) + L_S\sin(A_q + S_q)\right) - Gm_W\left(L_{As}\sin(A_q) + L_W\sin(A_q + W_q)\right)$$
$$-L_{Al}L_Sm_P\sin(S_q)\left(A_w^2 - (A_w + S_w)^2\right) - L_{As}L_Wm_W\sin(W_q)\left(A_w^2 - (A_w + W_w)^2\right)$$

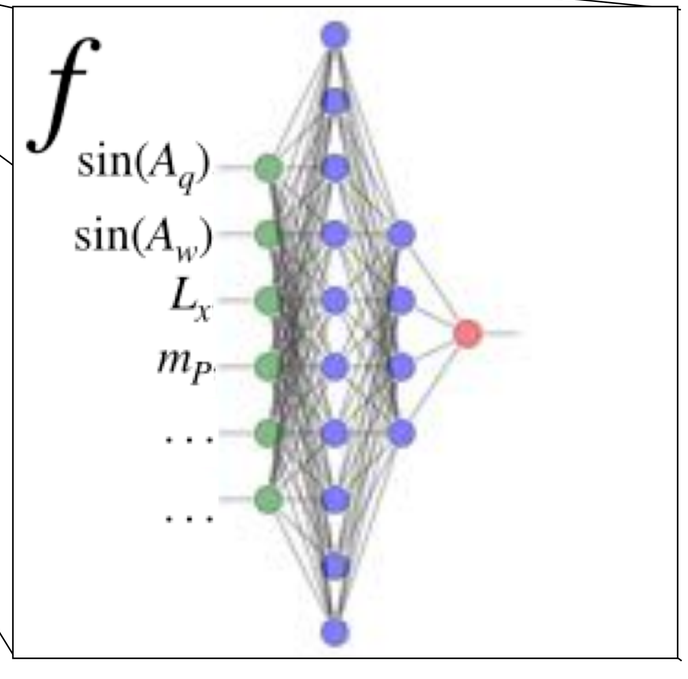
$$r_2 = -L_Wm_W\left(G\sin(A_q + W_q) + L_{As}\sin(W_q)A_w^2\right)$$

$$r_3 = L_Sm_P\left(G\sin(A_q + S_q) - \right)$$

$$A_q'' = -\frac{r_1M_{22}M_{33} - r_2M_{12}M_{33} - r_3M_{13}M_{22}}{M_{13}M_{22}M_{31} - M_{33}\left(M_{11}M_{22} - M_{12}M_{21}\right)}$$

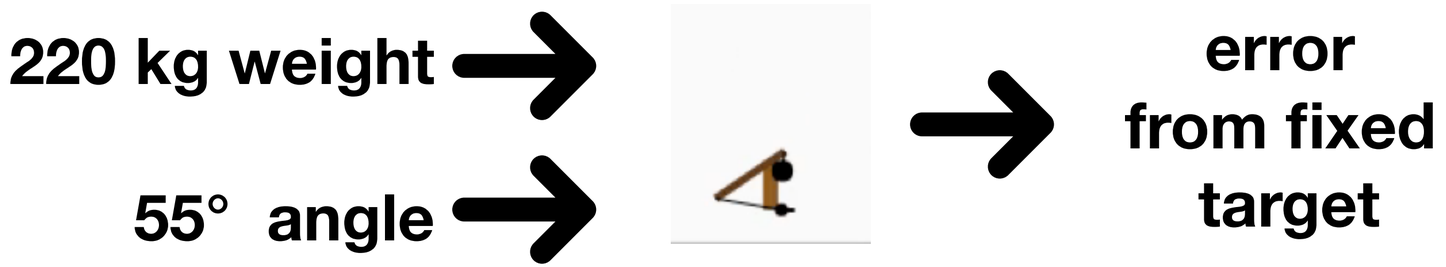
$$W_q'' = \frac{r_1M_{21}M_{33} - r_2\left(M_{11}M_{33} - M_{13}M_{31}\right) - r_3M_{13}M_{21}}{M_{13}M_{22}M_{31} - M_{33}\left(M_{11}M_{22} - M_{12}M_{21}\right)}$$

$$S_q'' = \frac{r_1M_{22}M_{31} - r_2M_{12}M_{31} - r_3\left(M_{11}M_{22} - M_{12}M_{21}\right)}{M_{13}M_{22}M_{31} - M_{33}\left(M_{11}M_{22} - M_{12}M_{21}\right)}$$



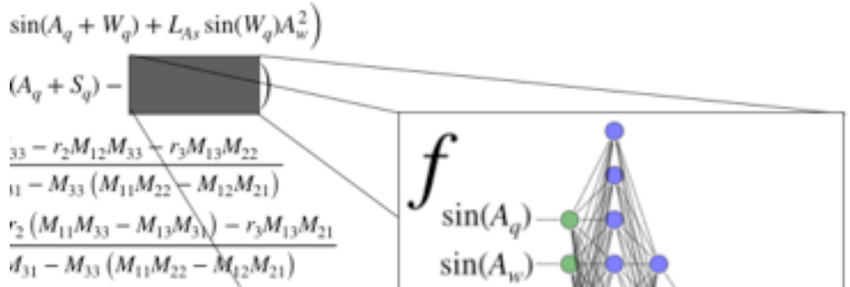
A derivative three ways

- Derivative with respect to the control parameters



- Derivative with respect to an approximation network

- Derivative with respect to a subset of the model itself

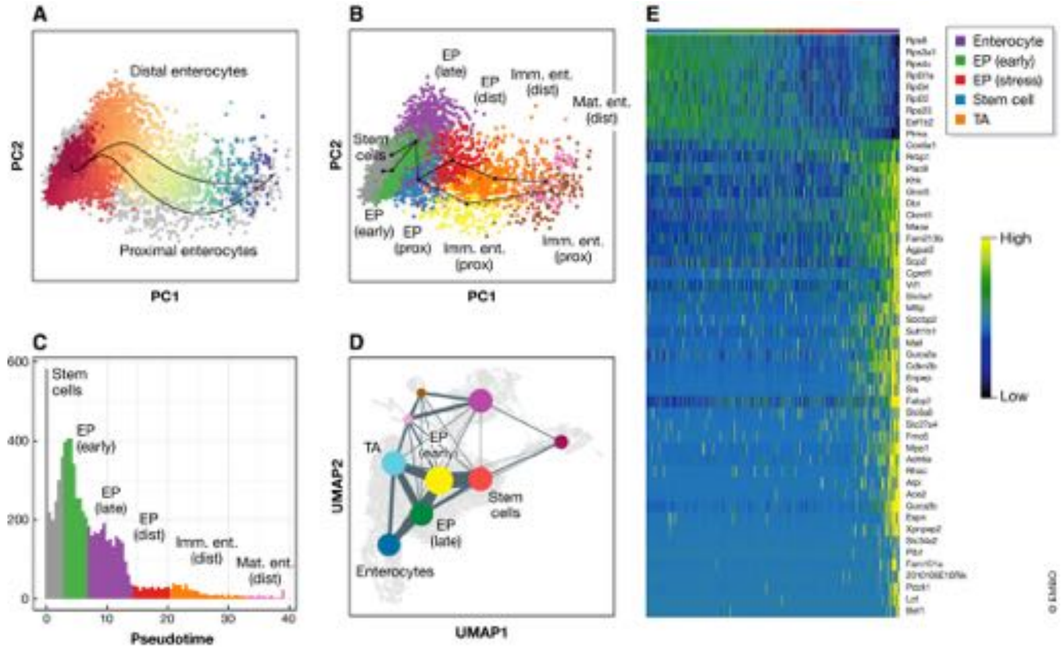


Differentiable Programming is going to disrupt Scientific Modeling and Simulation

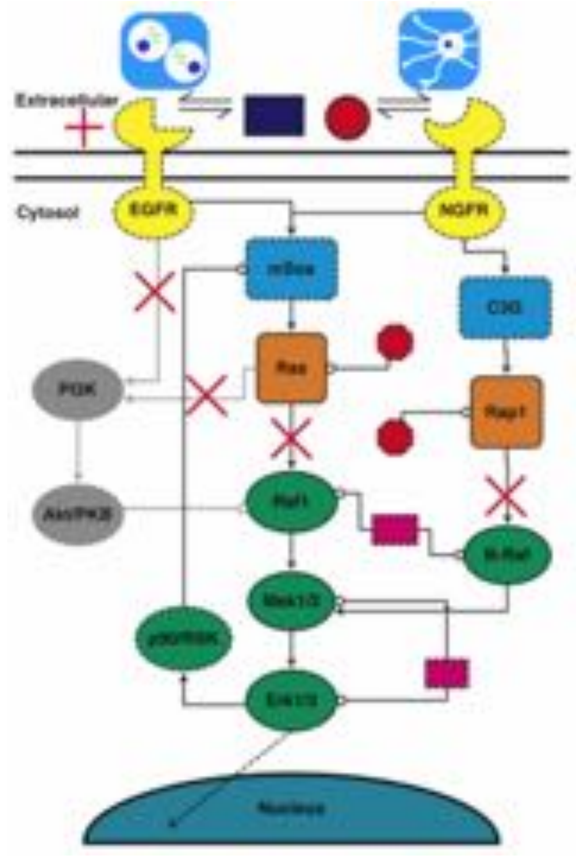
Ongoing work by Julia Computing and others in pharmaceuticals, Engineering, Chemistry, Manufacturing, Batteries, Climate, …

Scientific ML is model-based and data-efficient

Modern experimental procedures quickly captures terabytes of data



But “big data” often pales in comparison to the knowledge embedded into each model

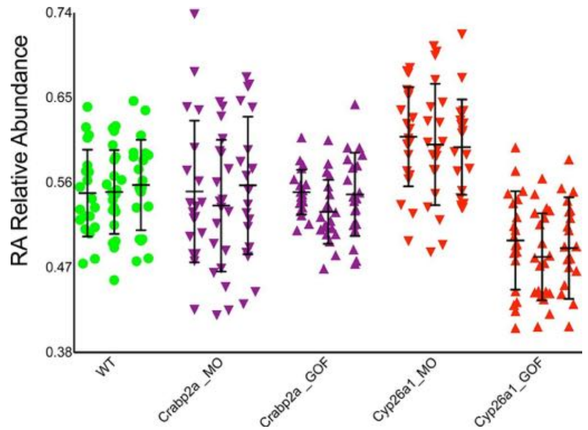


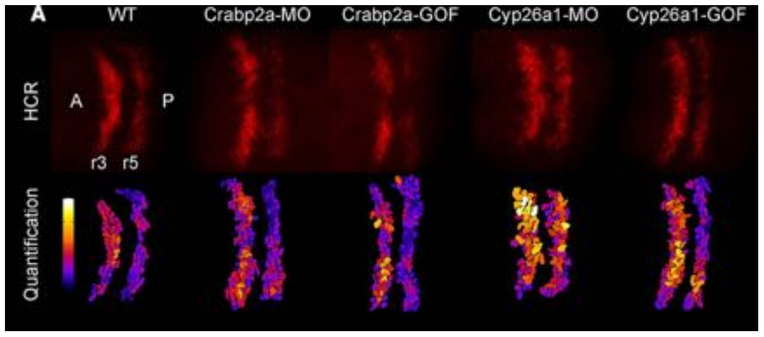
How do we simultaneously use both sources of knowledge?

Universal differential equations

Universal Differential Equations for Scientific Machine Learning (*arXiv:2001.04385*)

Deep Learning discovers systems models from data





Find neural networks so the model matches the data,
then find the equations which implies new chemical reactions and pharmacologically-relevant systems

$$d[RA_{out}] = (\beta - b[RA_{out}] + c[RA_{in}])dt,$$

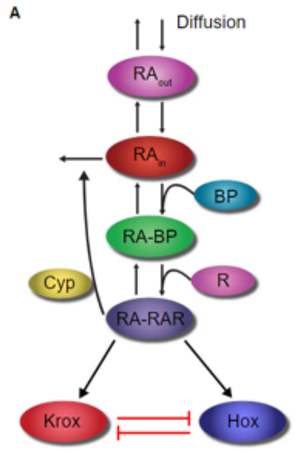
$$d[RA_{in}] = \left(b[RA_{out}] + \delta[RA - BP] - \left(\gamma[BP] + \eta + \text{NN} - c\right)[RA_{in}]\right)dt + \sigma dW_t,$$

$$d[\mathrm{RA-BP}] = (\gamma[BP][RA_{in}] + \lambda[BP][RA - RAR] - \text{NN} - BP])dt,$$

$$d[RA - RAR] = (\text{NN} - \lambda[BP][RA - RAR])dt,$$

$$d[RAR] = (\text{NN} + \lambda[BP][RA - RAR] - r[RAR])dt,$$

$$d[BP] = (a - \lambda[BP][RA - RAR] - \gamma[BP][RA_{in}] + (\delta + \nu[RAR])[RA - BP] - u[BP])dt,$$



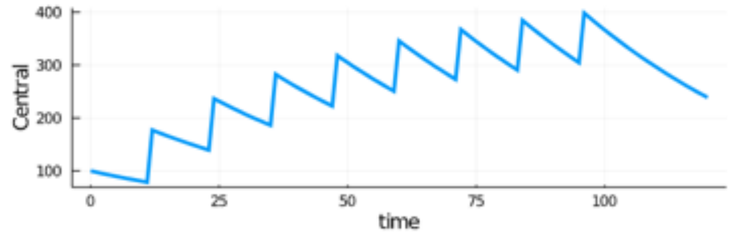
Improve QSP and PBPK productivity

pumas AI

Beyond Deterministic Models in Drug Discovery and Development (Trends in Pharmacological Sciences)

DeepNLME in Practice: Automated Discovery of Dynamics

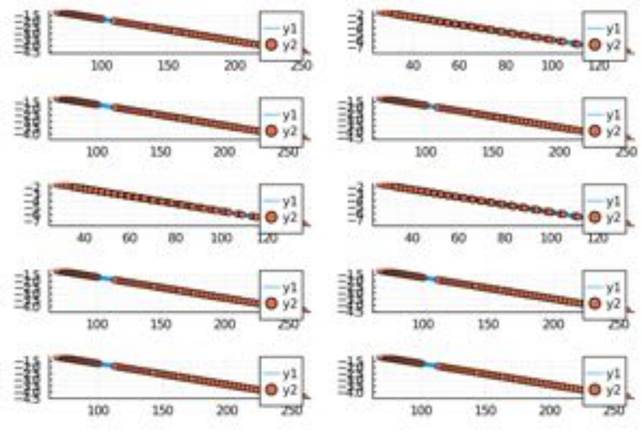
Use all prior scientific knowledge (dosage regimens, covariate relations, etc.)
Train neural networks to maximize population likelihoods (LaplaceI, FOCEI, NaivePooled)
Then predict PK(/PD) profiles of new patients



Or let Pumas discover the mechanistic model:

```
basis = Basis([u,u^2,u^3,cos(u),sin(u),tan(u),u/(1+u)],[u]) | 7 dimensional basis in ["u"]
opt = STRRidge(1e-4) | STRRidge{Float64}(0.0001)
λs = exp10.(-10:0.1:2) | 121-element Array{Float64,1}:
Ψ = SInDy(Xmat, Ymat, basis, λs, maxiter = 1000, opt = opt) | Sparse Identification Result with 1 active terms.
latexify.(sum(Ψ.equations.basis)) | L"$p_1 * u$"
```

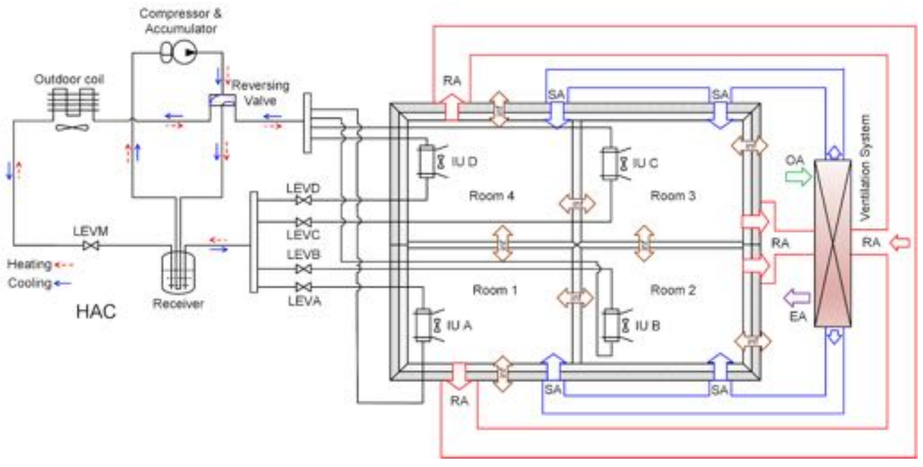
and showcase diagnostics demonstrating the predicted equation is reliable!



pumas AI

Accelerated building energy efficiency models

- Automation of model order reduction on DAEs via neural DAE surrogate dimensional reductions
- Interaction with component-based modeling to allow for generating accelerated building models with transferred learning components

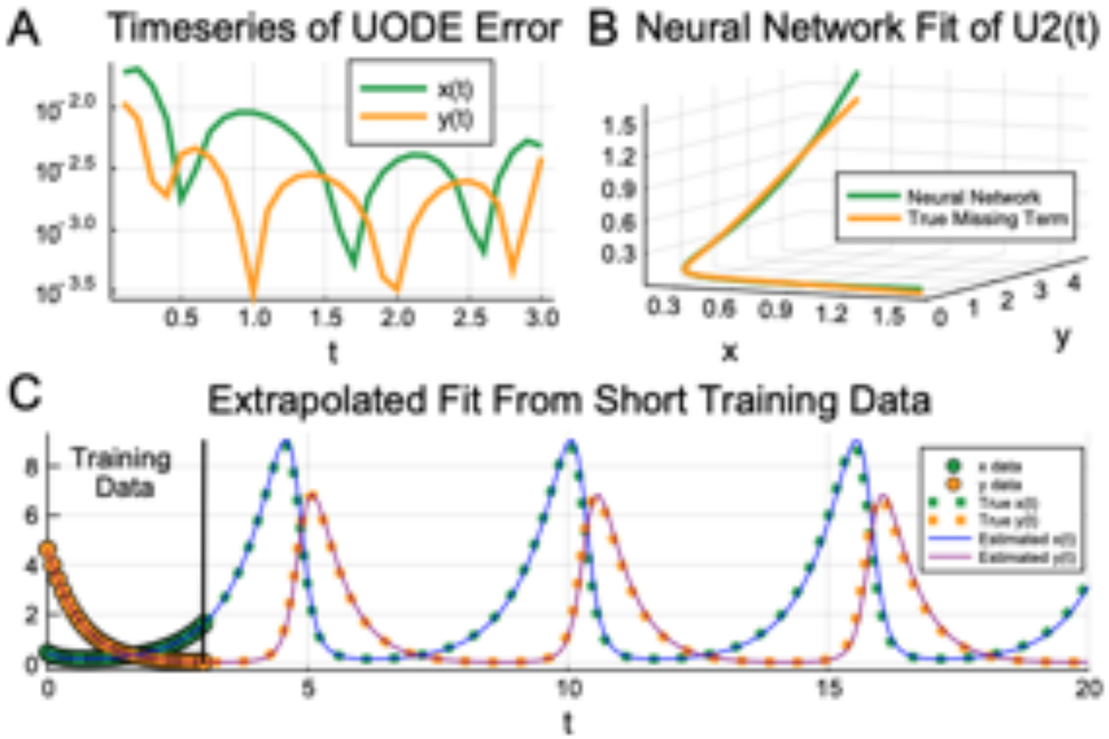


Simulations of physical systems involve a combination of compiler passes, ODE / DAE solvers, and machine learning.

Differentiable Programming ties all these parts together.

Scientific Machine Learning

https://sciml.ai



Noteworthy new capabilities

- Combine Science and Machine Learning
- Comprehensive Differential Equation Solvers
- GPU acceleration
- MTK.jl: A DSL for modeling and simulation
- Automatic differentiation

Accelerate Extremely Stiff Equations

Accelerating Simulation of Stiff Nonlinear Systems using Continuous-Time Echo State Networks
[NeurIPS 2020 ML4Eng Workshop, NeurIPS 2020]

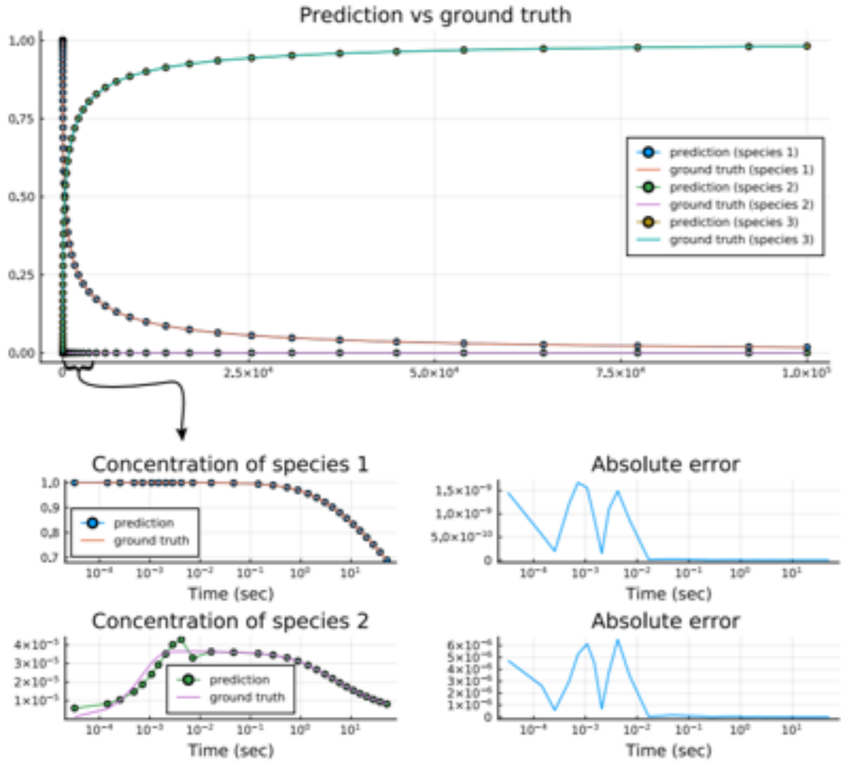


Figure 1: **Training performance on the the Robertson equations**. We see that our network is able to reproduce the dynamics, capturing the slow (second row) and fast transients (third row) with low error

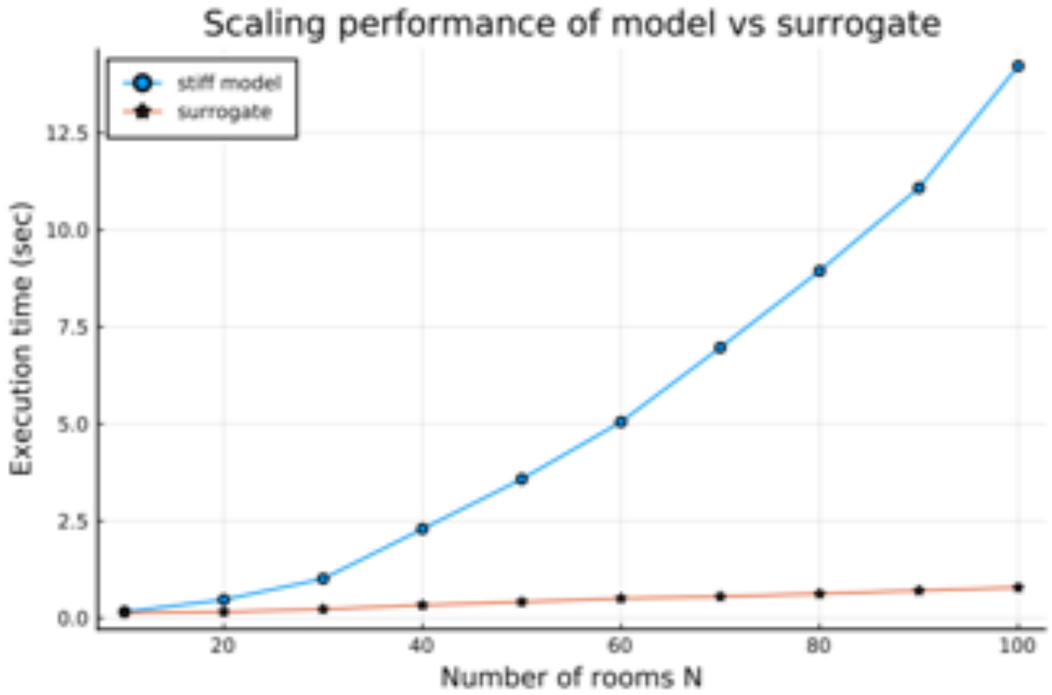
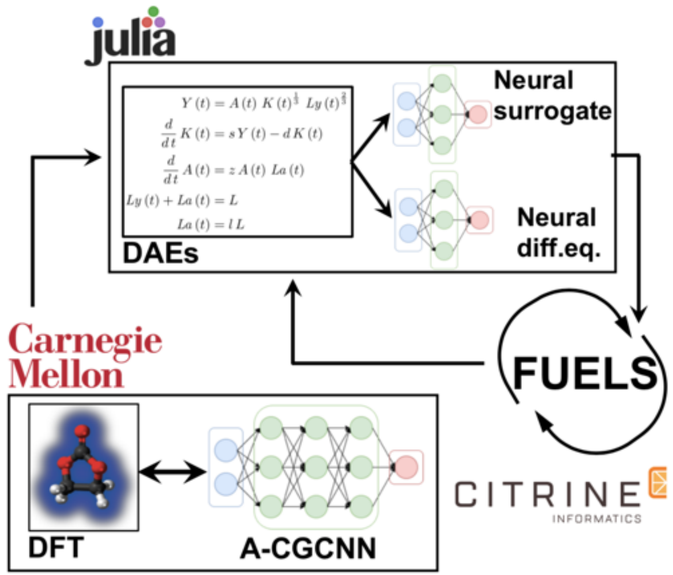
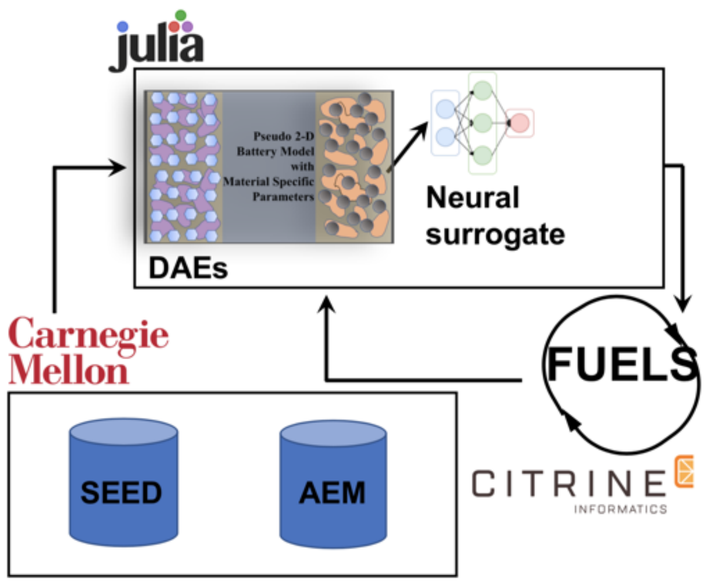


Figure 4: **Scaling performance of surrogate on heating system.** Our surrogate's runtime grows in near-constant time while the benchmark problem shows exponential growth in runtime.

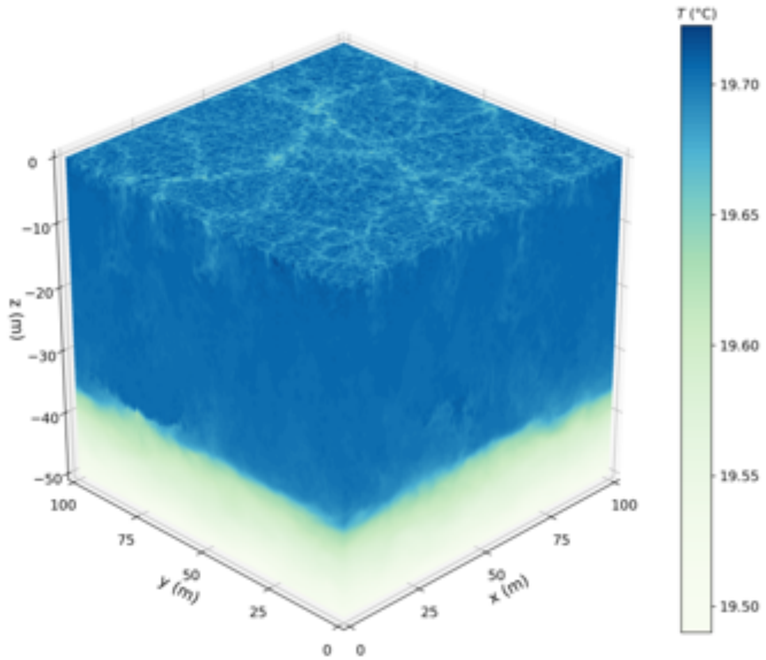


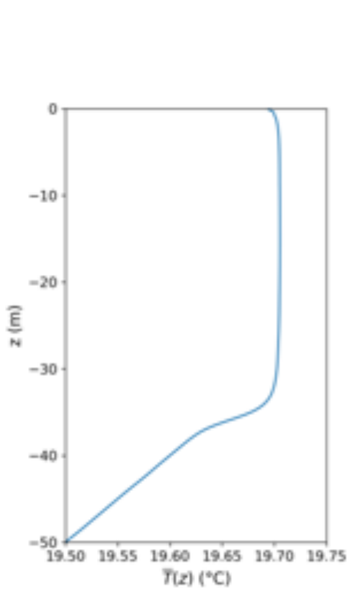


Optimization of materials for battery-powered aircraft

- GPU accelerate small (30) DAE battery models
- Utilize neural surrogates for global sensitivities
- Automatically refine equations from data
- Use these models to identify material properties
- Propose optimal experimental design
- Closed loop: direct collaboration with materials scientists, restructure model with new data

Automated Climate Parameterizations





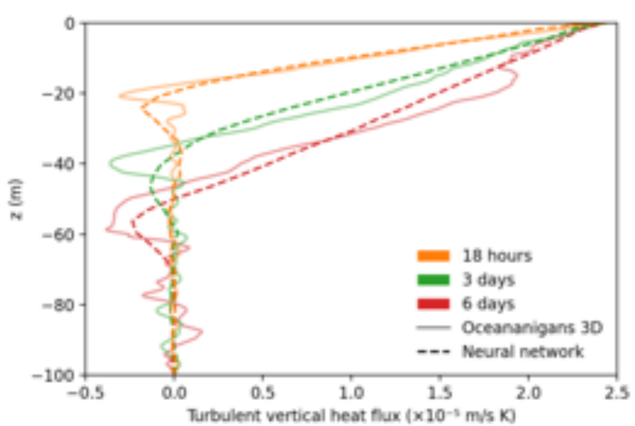
$$\nabla \cdot \mathbf{u} = 0$$

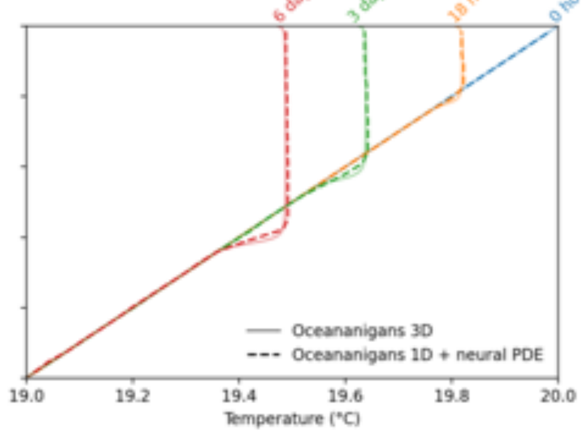
$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \mathrm{Pr}\nabla^2 \mathbf{u} + b\hat{z}$$

$$\frac{\partial b}{\partial t} + \mathbf{u} \cdot \nabla b = \nabla^2 b + Fe^z$$

Approximate only the vertical flux

$$\left(\frac{\partial}{\partial t} + \mathbf{u}.\nabla - \nabla^2\right)c' - \frac{\partial}{\partial z}\overline{w'c'} = -w'\frac{\partial \overline{c}}{\partial z}$$

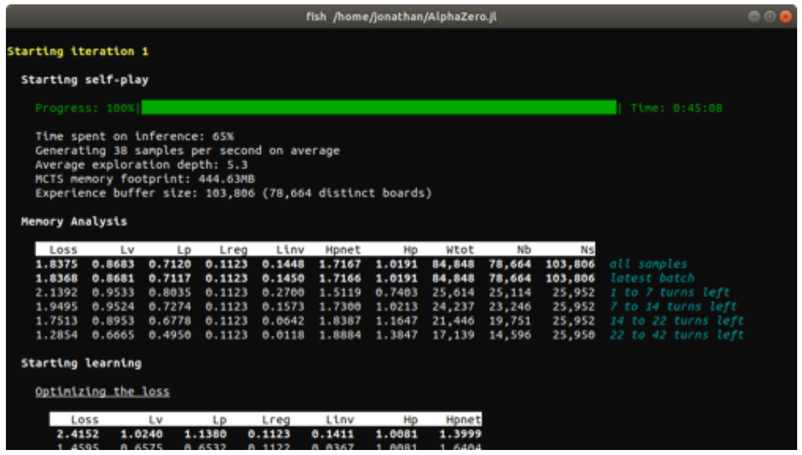


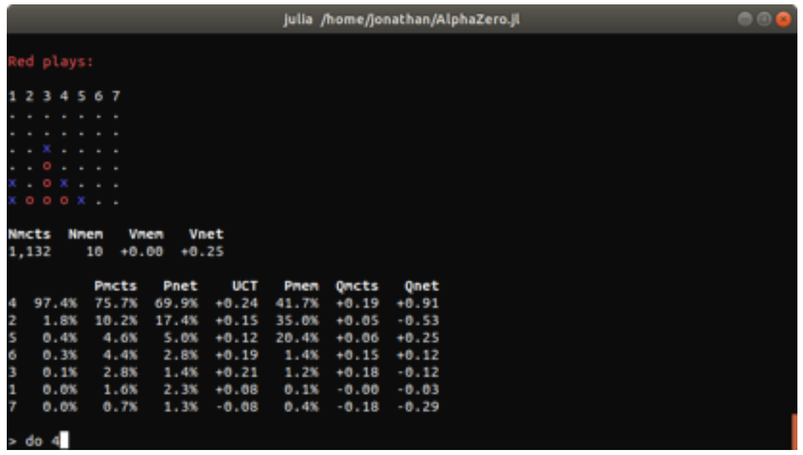


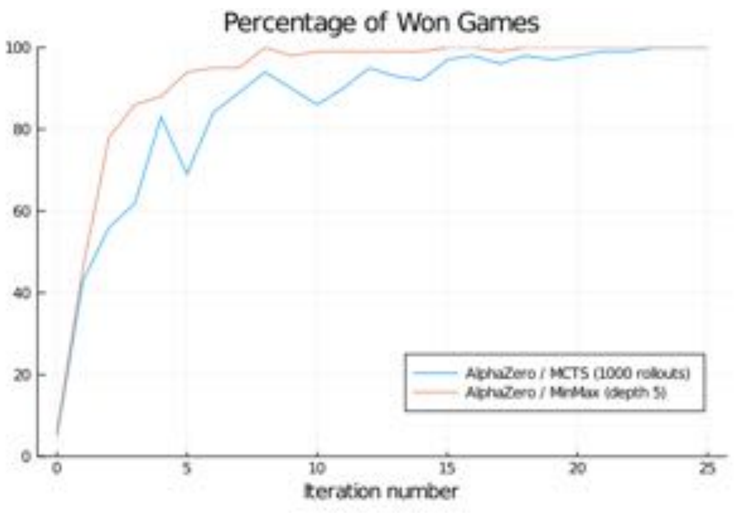
High fidelity 15,000x acceleration over direct
3D simulation. Traditionally done by hand!

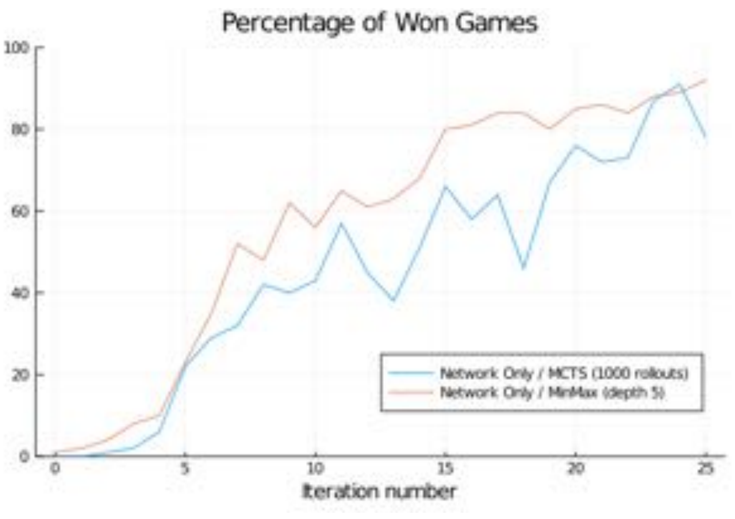
Capturing missing physics in climate model parameterizations using neural differential equations
Ali Ramadhan, John Marshall, Andre Souza, Gregory LeClaire Wagner, Manvitha Ponnapati, Christopher Rackauckas

Reinforcement Learning with AlphaZero.jl







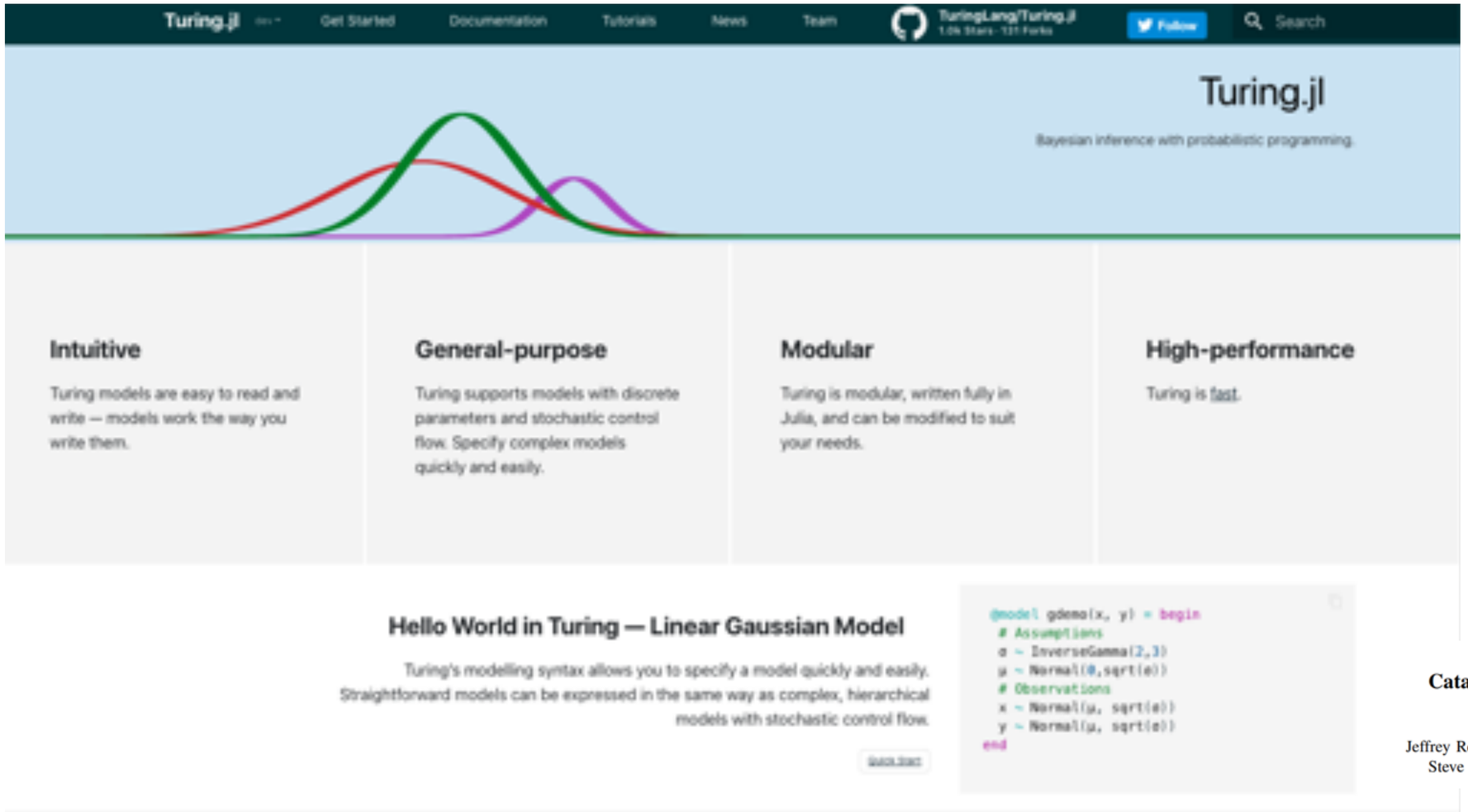


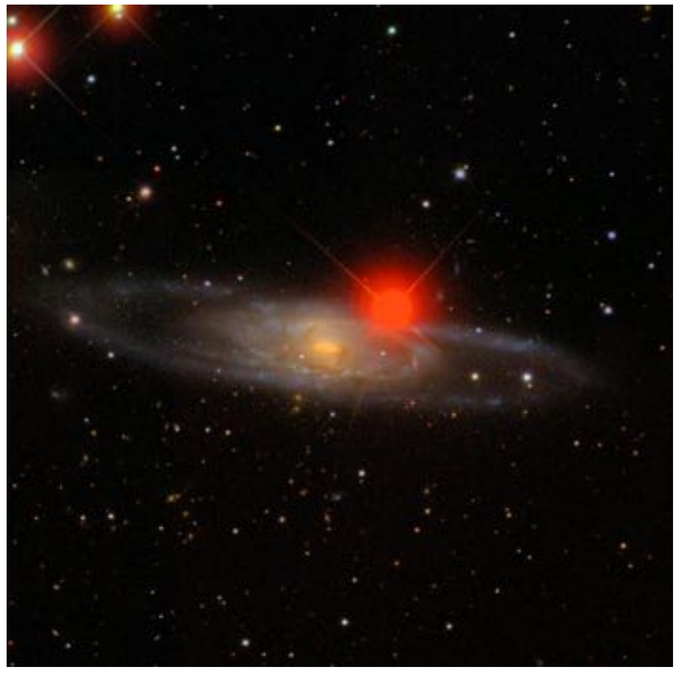
Jonathan Laurent
https://github.com/jonathan-laurent/AlphaZero.jl

- **Simple**: Core algorithm is 2000 lines of Julia
- **Extensible**: Generic interfaces allow new games and learning
- **Fast**: 10-100x faster than other high-level alternatives
- **Scalable**: Combines distributed, multi-threaded and multi-GPU parallelism

Celeste was our first peta-scale Bayesian Inference application

Turing.jl makes Probabilistic Programming available more broadly





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‖

Thank You

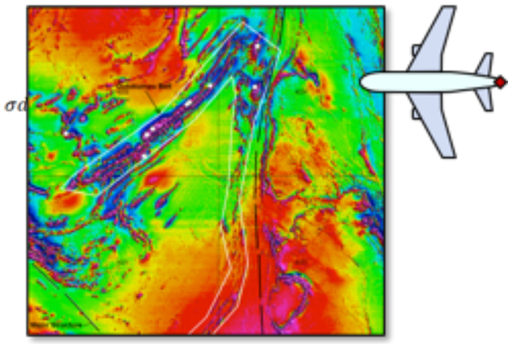
$$d[RA_{out}] = (\beta - b[RA_{out}] + c[RA_{in}])dt,$$

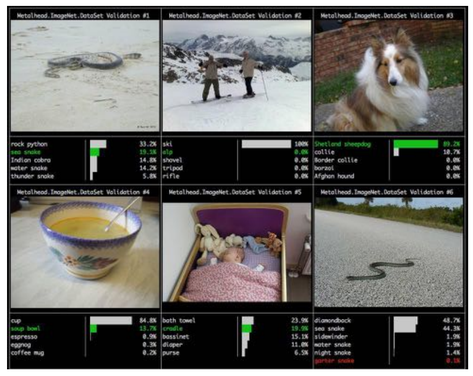
$$d[RA_{in}] = \left(b[RA_{out}] + \delta[RA - BP] - \left(\gamma[BP] + \eta + \quad - c\right)[RA_{in}]\right)dt + \sigma d$$

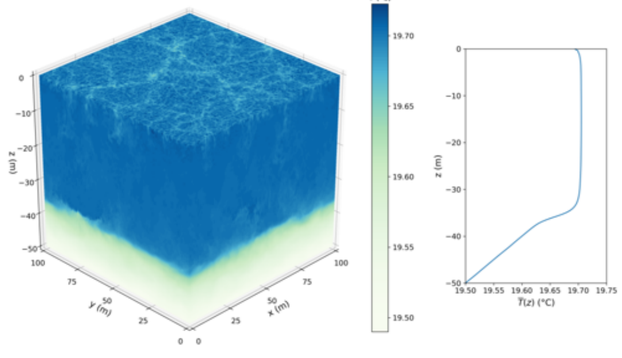
$$d[\mathrm{RA - BP}] = (\gamma[BP][RA_{in}] + \lambda[BP][RA - RAR] - \quad - BP])dt,$$

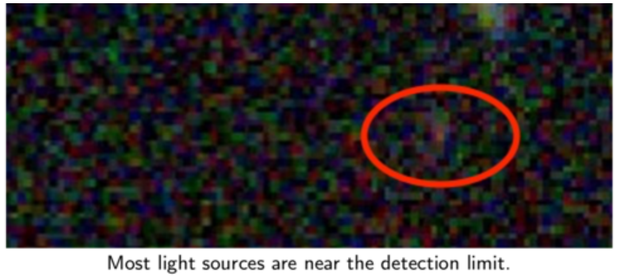
$$d[RA - RAR] = (\quad] - \lambda[BP][RA - RAR])dt,$$

$$d[RAR] = (\quad AR] + \lambda[BP][RA - RAR] - r[RAR])dt,$$

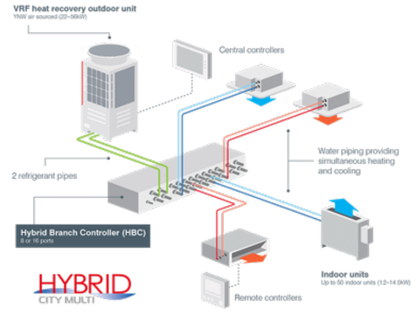








Most light sources are near the detection limit.



- Electrification, batteries, climate change and the pandemic are reshaping entire industries
- Differentiable Programming is at the heart of improving the pace of innovation
- In Julia, ∂P is being driven by real-world applications in modeling and simulation
- These advances are accelerating product design for engineers and scientists
- In short, Julia is building a language ecosystem and compiler toolchain for modern science