Bolt.jl - The Differentiable Einstein-Boltzmann Solver

Cosmology from Home 2022

Jamie Sullivan w/ Zack Li & Marius Millea

Berkeley UNIVERSITY OF CALIFORNIA

DOE CSGF

BERKELEY CENTER *for* COSMOLOGICAL PHYSICS

Overview

1. Who needs *another* Boltzmann code?

Overview

1. Who needs *another* Boltzmann code?
2. Design of Bolt.jl

Overview

1. Who needs *another* Boltzmann code?
2. Design of Bolt.jl
3. What Bolt can do

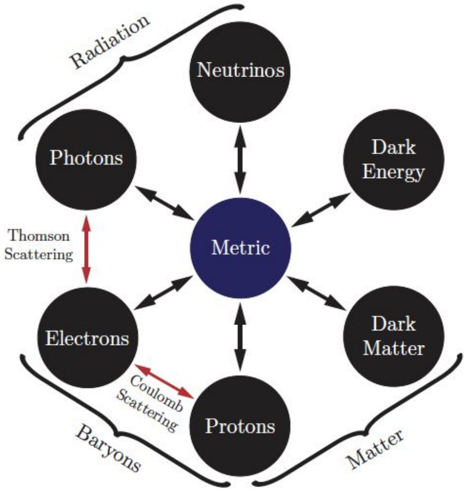
1. Why another Boltzmann code?

The Cornerstone of Cosmology

LSS and the CMB models evolve perturbations described by GR and the Boltzmann eqn. (Einstein-Boltzmann system)

Accurate modeling requires solving this (stiff) ODE system

LSS and the CMB models *require* Boltzmann codes



Ma&Bertschinger95, Baumann

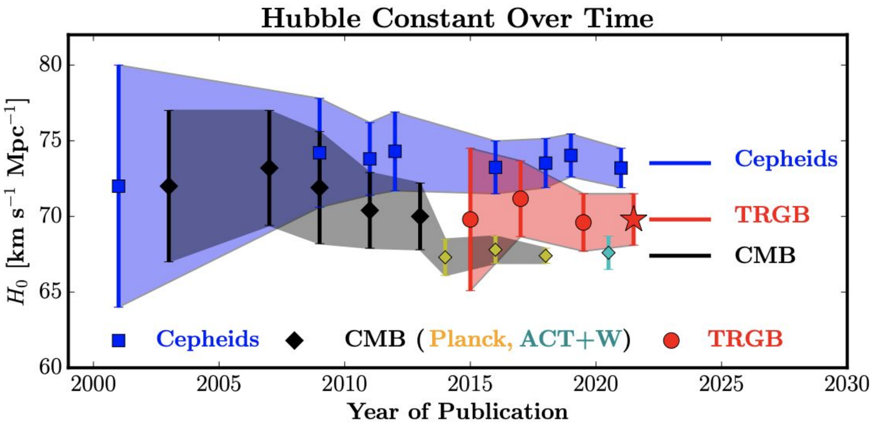
A Time of Tension

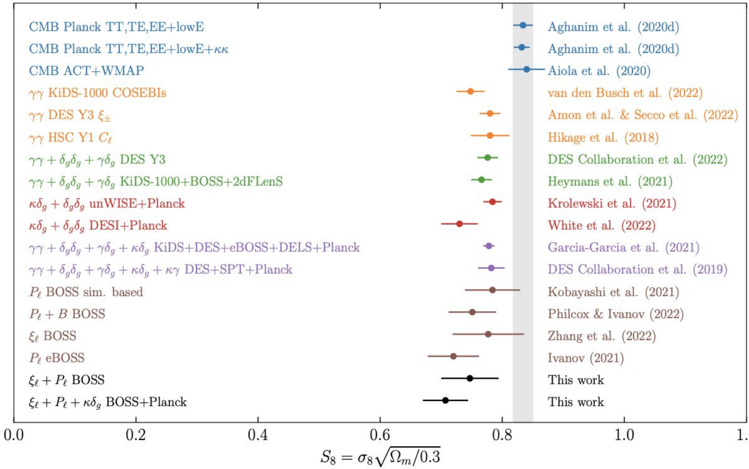
The data are telling us LCDM may not be enough!

Need to consider *model extensions*

In practice, this means:

1. more model parameters
2. editing source code





Freedman21, Chen++22

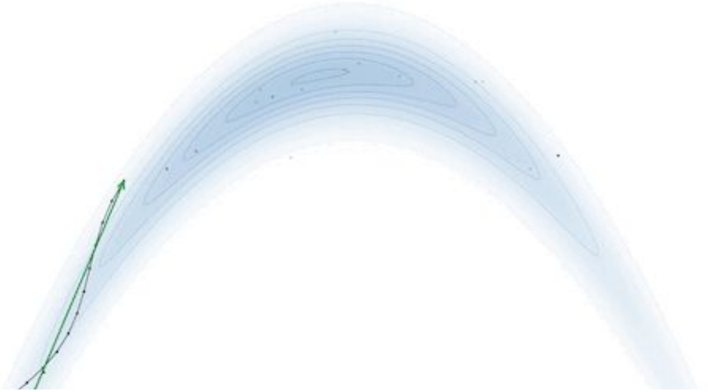
A Need for Differentiable Models

More parameters means high-dimensional inference

Gradient-based methods make this tractable

Must use model gradients, even when not analytic!

Can make an approximation*,
or use a differentiable model**



*e.g. Mancini++21, DeRose++21, Colas++20, Wibking++20 **e.g. Modi++20, Modi++21, Hearin++22

Beyond State of the Art

CAMB/CLASS are incredible feats of scientific software engineering

However:

- neither is differentiable
- navigating the source is a time commitment

The future is differentiable and easy to modify

- **-> Bolt.jl**

Blas++11, Lewis++00

2. Bolt.jl

Bolt.jl Design

Goals:

1. Differentiability
2. Rapid physics prototyping

Design aspects:

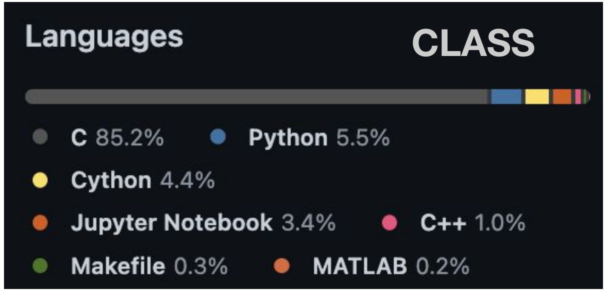
1. Julia & automatic differentiation (AD)
2. Minimal approximations

Julia

Solves the 2-language problem

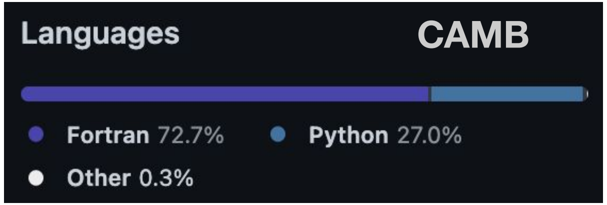
Multiple dispatch reduces code rewrites

Rich ODE and AD libraries (SciML)



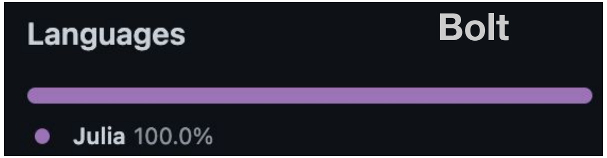
Languages CLASS

● C 85.2% ● Python 5.5%
● Cython 4.4%
● Jupyter Notebook 3.4% ● C++ 1.0%
● Makefile 0.3% ● MATLAB 0.2%



Languages CAMB

● Fortran 72.7% ● Python 27.0%
● Other 0.3%



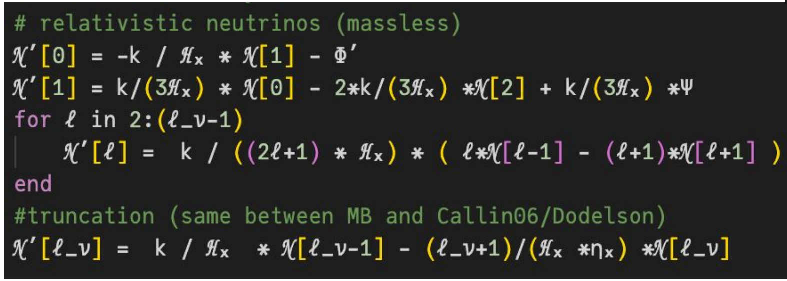
Languages Bolt

● Julia 100.0%

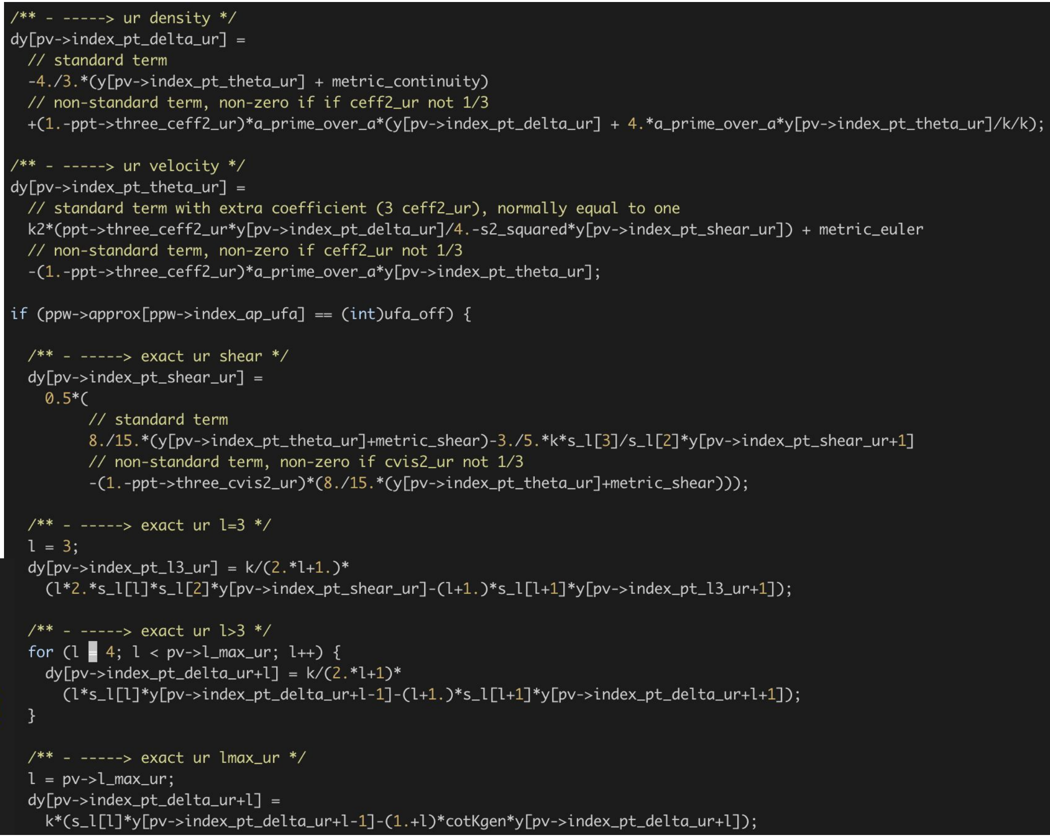
julialang.org, sciml.ai, github.com/lesgourg/class_public, github.com/cmbant/CAMB

Julia - Readability

Both codes compute the same thing - which would you rather read?



```
# relativistic neutrinos (massless)
𝒩′[0] = -k / ℋₓ * 𝒩[1] - Φ′
𝒩′[1] = k/(3ℋₓ) * 𝒩[0] - 2*k/(3ℋₓ) *𝒩[2] + k/(3ℋₓ) *Ψ
for ℓ in 2:(ℓ_ν-1)
    𝒩′[ℓ] =  k / ((2ℓ+1) * ℋₓ) * ( ℓ*𝒩[ℓ-1] - (ℓ+1)*𝒩[ℓ+1] )
end
#truncation (same between MB and Callin06/Dodelson)
𝒩′[ℓ_ν] =  k / ℋₓ  * 𝒩[ℓ_ν-1] - (ℓ_ν+1)/(ℋₓ *ηₓ) *𝒩[ℓ_ν]
```

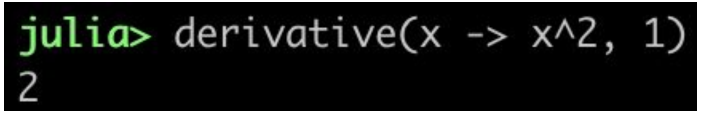


```
/** - -----> ur density */
dy[pv->index_pt_delta_ur] =
  // standard term
  -4./3.*(y[pv->index_pt_theta_ur] + metric_continuity)
  // non-standard term, non-zero if if ceff2_ur not 1/3
  +(1.-ppt->three_ceff2_ur)*a_prime_over_a*(y[pv->index_pt_delta_ur] + 4.*a_prime_over_a*y[pv->index_pt_theta_ur]/k/k);

/** - -----> ur velocity */
dy[pv->index_pt_theta_ur] =
  // standard term with extra coefficient (3 ceff2_ur), normally equal to one
  k2*(ppt->three_ceff2_ur*y[pv->index_pt_delta_ur]/4.-s2_squared*y[pv->index_pt_shear_ur]) + metric_euler
  // non-standard term, non-zero if ceff2_ur not 1/3
  -(1.-ppt->three_ceff2_ur)*a_prime_over_a*y[pv->index_pt_theta_ur];

if (ppw->approx[ppw->index_ap_ufa] == (int)ufa_off) {

  /** - -----> exact ur shear */
  dy[pv->index_pt_shear_ur] =
    0.5*(
         // standard term
         8./15.*(y[pv->index_pt_theta_ur]+metric_shear)-3./5.*k*s_l[3]/s_l[2]*y[pv->index_pt_shear_ur+1]
         // non-standard term, non-zero if cvis2_ur not 1/3
         -(1.-ppt->three_cvis2_ur)*(8./15.*(y[pv->index_pt_theta_ur]+metric_shear)));

  /** - -----> exact ur l=3 */
  l = 3;
  dy[pv->index_pt_l3_ur] = k/(2.*l+1.)*
    (l*2.*s_l[l]*s_l[2]*y[pv->index_pt_shear_ur]-(l+1.)*s_l[l+1]*y[pv->index_pt_l3_ur+1]);

  /** - -----> exact ur l>3 */
  for (l = 4; l < pv->l_max_ur; l++) {
    dy[pv->index_pt_delta_ur+l] = k/(2.*l+1)*
      (l*s_l[l]*y[pv->index_pt_delta_ur+l-1]-(l+1.)*s_l[l+1]*y[pv->index_pt_delta_ur+l+1]);
  }

  /** - -----> exact ur lmax_ur */
  l = pv->l_max_ur;
  dy[pv->index_pt_delta_ur+l] =
    k*(s_l[l]*y[pv->index_pt_delta_ur+l-1]-(1.+l)*cotKgen*y[pv->index_pt_delta_ur+l]);
```

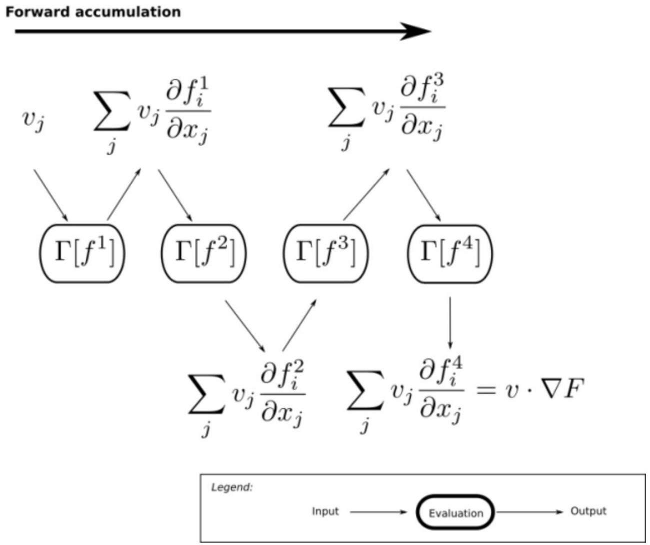
github.com/lesgourg/class_public

Automatic Differentiation

Chain rule for code!



```
julia> derivative(x -> x^2, 1)
2
```


Forward accumulation
v_j
$\sum_j v_j \frac{\partial f_i^1}{\partial x_j}$
$\sum_j v_j \frac{\partial f_i^3}{\partial x_j}$
$\Gamma[f^1]$
$\Gamma[f^2]$
$\Gamma[f^3]$
$\Gamma[f^4]$
$\sum_j v_j \frac{\partial f_i^2}{\partial x_j}$
$\sum_j v_j \frac{\partial f_i^4}{\partial x_j} = v \cdot \nabla F$
Legend:
Input
Evaluation
Output

Forward mode AD with dual numbers

$$f(a + \sum_{i=1}^{N} b_i \epsilon_i) = f(a) + f'(a) \sum_{i=1}^{N} b_i \epsilon_i$$

AD through stiff ODE solver

Approximation-free

We *choose* to minimize the use of approximations in Bolt

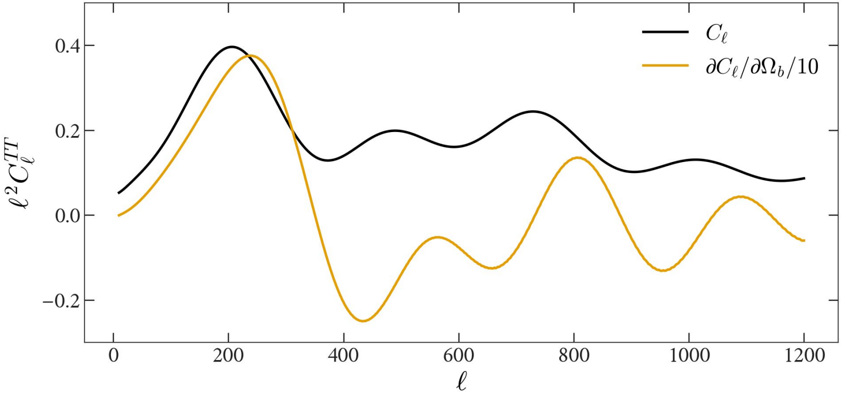
No TCA, UFA, massive neutrino fluid approximations

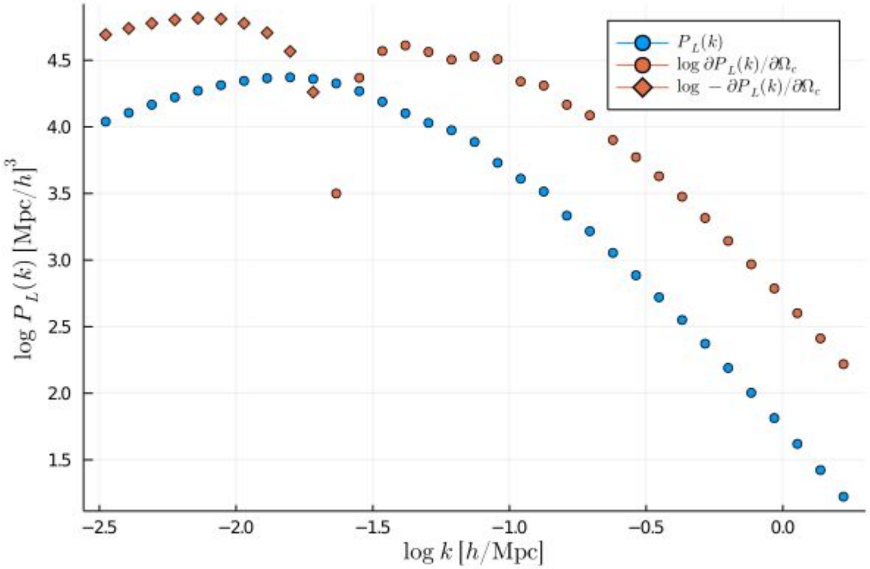
Enable user-driven extensions beyond vanilla LCDM

3. Bolt.jl in action

Gradients for the CMB and LSS

Exact gradients of the CMB anisotropy, linear matter power spectra



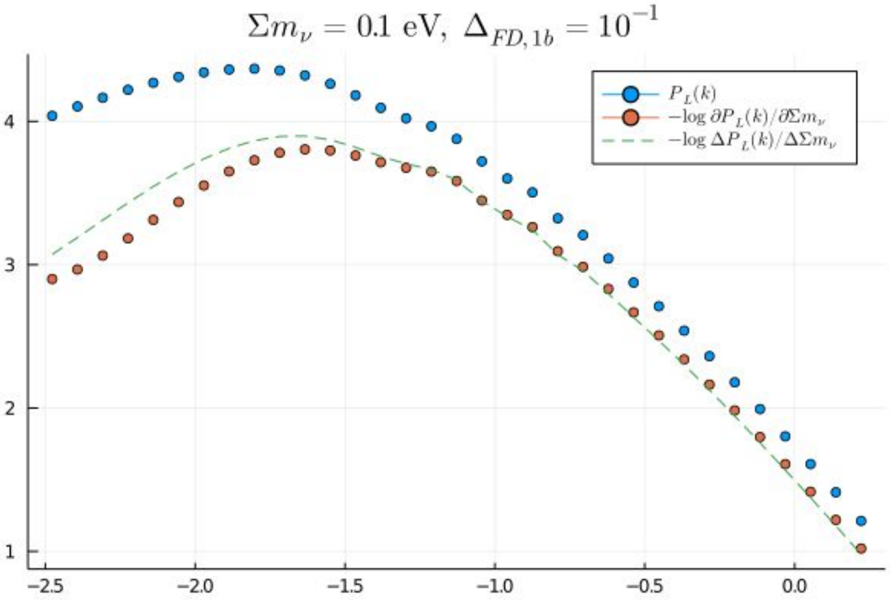


AD >> FD

Finite difference (FD) gives gradient approximations

Well-known that FD does not perform well for neutrino mass

Exact AD derivatives solve this problem!

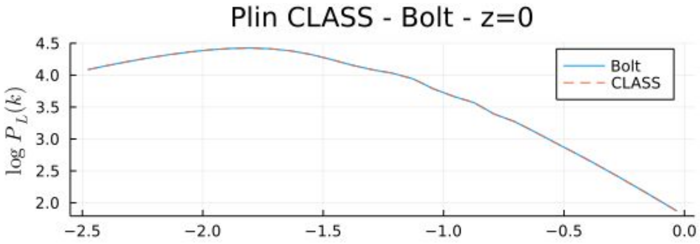


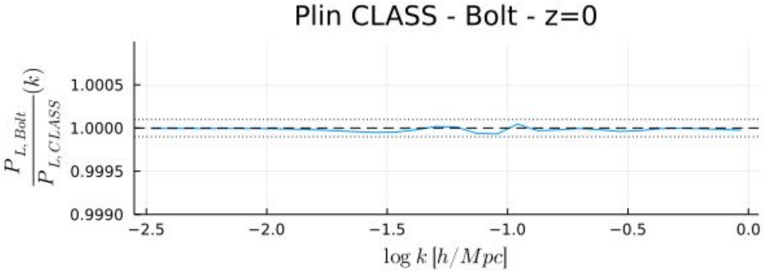
Agreement at the 0.1% Level

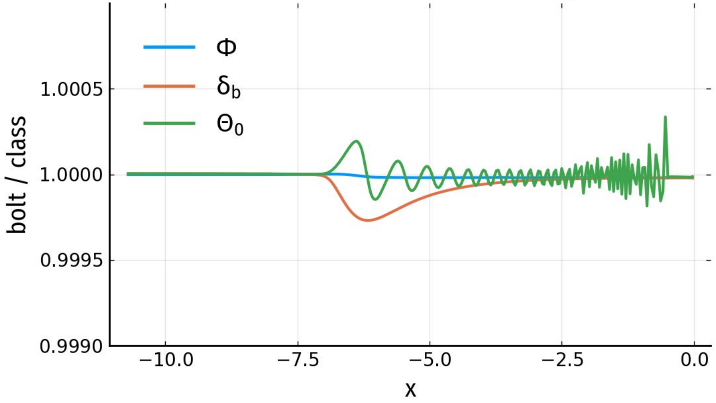
CLASS/CAMB are consistent at the 0.1% level in CMB and matter power spectra

Bolt joins* this exclusive club!

We agree on the perturbations for individual k-modes too







*see also, Moser++22

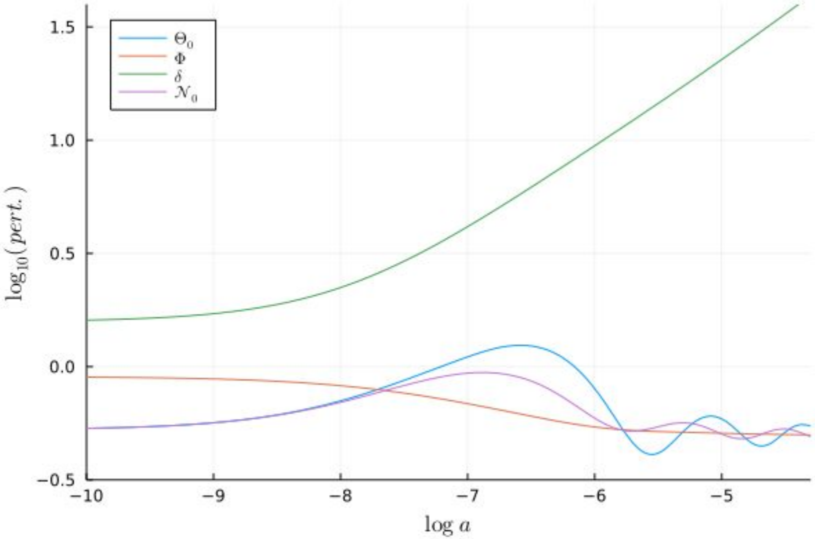
Bolt.jl runs on GPU!

First Boltzmann code that can run on GPU

In the right setting, can lead to enormous performance gains

Minimal code rewriting for GPU with CUDA.jl, Adapt.jl

These perturbations came out of a GPU!



Current & Future Work

Explore reverse mode AD

Improve performance, especially on GPU

Accelerate likelihoods with gradient-based sampling

Complement existing differentiable models (e.g. N-body)

Explore new physics extensions!

Summary

Differentiability indispensable for high-dimensional inference

Bolt.jl is:

- the **first** differentiable Boltzmann code
- consistent with CLASS/CAMB at the **<0.1% level**
- the first Boltzmann code **on GPU!**

Lack of approximations and readability **help you the cosmologist** to quickly add new physics!