ECLAIR: a complete toolbox for cosmological inference

Stéphane Ilić

IJCLab, Orsay

Atelier Outils de l'action Dark Energy
@ IAP, 05/06/2023

Introduction: back in 2016-2017

Several “CosmoBoxes” on the market :

- CosmoMC
- MontePython
- CosmoSIS
- Cobaya
- …

…but none entirely satisfying for my needs

Introduction

My “needs”:

- Juggling with many cosmological models (and as many Boltzmann solvers)
- Non trivial exploration of parameter space (priors, constraints…)
- A (relatively) big cluster to exploit

Generalized Dark Matter (GDM, Hu 1998)

- Defined for FLRW, linear perturbations
- Background: (non-zero) equation of state $w(\tau)$
- Perturbations: sound speed $c_s^2(\tau, k)$ & viscosity $c_{vis}^2(\tau, k)$
- Standard eqs. for density contrast & velocity divergence
- Continuity & Euler eqs. : requires closure equations (here by Hu):

$$\Pi_{\rm g} \equiv \frac{\delta P_{\rm g}}{\bar{\rho}_{\rm g}} = c_a^2 \delta_{\rm g} + (c_s^2 - c_a^2)\hat{\Delta}_{\rm g}^{\rm rest\ frame} \qquad \dot{\Sigma}_{\rm g} = -3\mathcal{H}\Sigma_{\rm g} + \frac{4}{1+w} c_{\rm vis}^2 \hat{\Theta}_{\rm g}^{\rm Newt.}$$

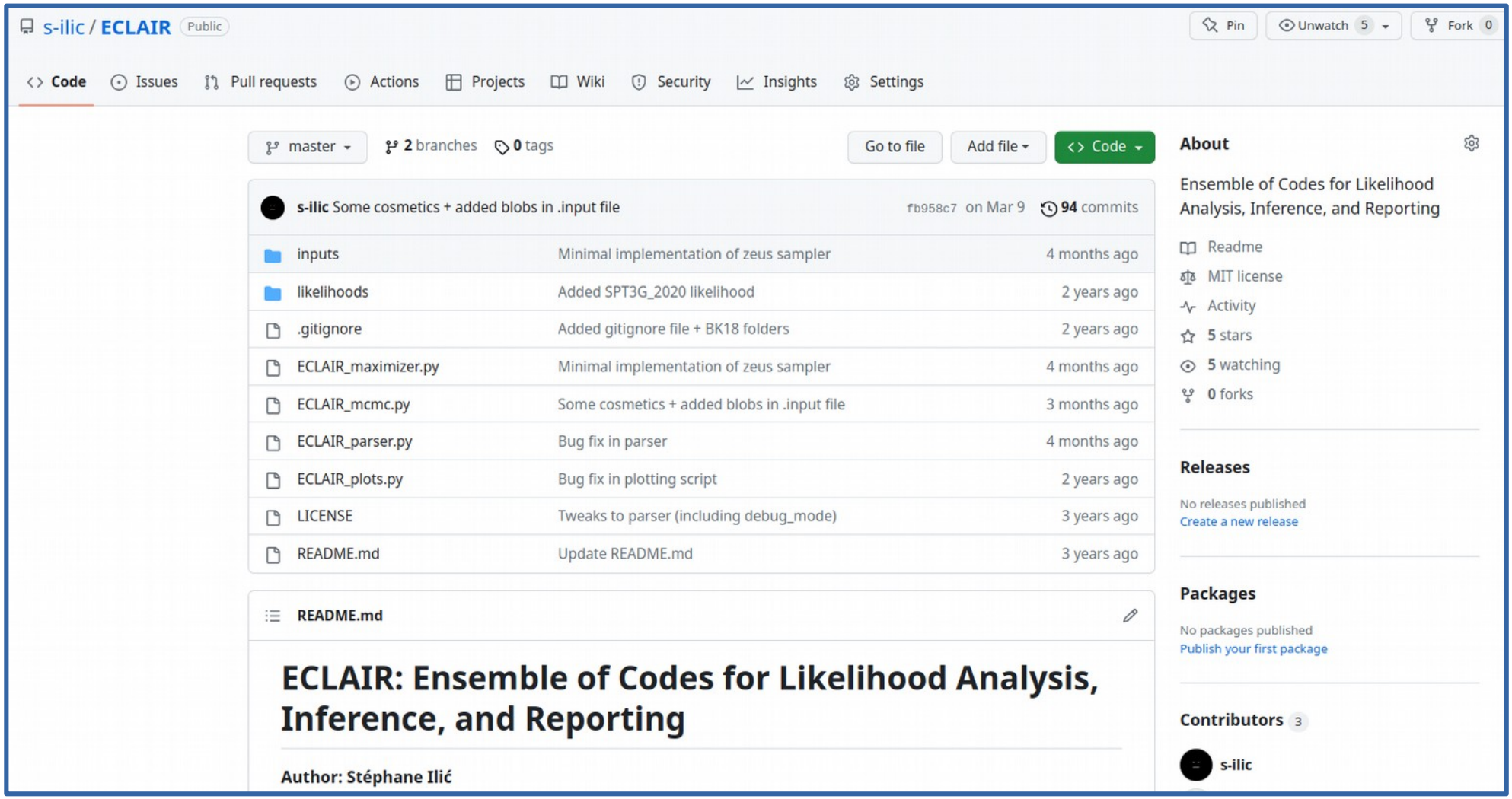
$$\left(c_a^2 = \frac{\dot{\bar{P}}_g}{\dot{\bar{\rho}}_g} = w - \frac{\dot{w}}{3\mathcal{H}(1+w)} \right)$$

Introduction

My “needs”:

- Juggling with many cosmological models (and as many Boltzmann solvers)
- Non trivial exploration of parameter space (priors, constraints…)
- A (relatively) big cluster to exploit

Introducing : ECLAIR



github.com/s-ilic/ECLAIR

Introducing : ECLAIR

- Two (fairly) short files in Python: main (~200 lines) & parser (500)
- Human-readable/tweakable, well-commented (I hope !)

Introducing : ECLAIR

```
==========================
== Free MCMC parameters ==
==========================

-------------------------------------------------------------------------------
-- Syntax should be "type par_name start min max width" where:                --
-- * "type": "var_class" if a CLASS parameter, otherwise "var"                --
-- * "par_name": parameter name (as known in CLASS or likelihoods)            --
-- * "start": mean of Gaussian used to initialize walkers positions           --
-- * "min/max": hard bounds (i.e. prior) on parameter value                   --
-- * "width": std dev of Gaussian used to initialize walkers positions        --
-- Note:                                                                      --
-- If an array of parameter needs to be passed to CLASS, append "_val_N" to   --
-- to each parameter where the integer N marks their position in the array    --
-- Example: "m_ncdm_val_0", "m_ncdm_val_1" for the masses of 2 neutrinos      --
-------------------------------------------------------------------------------

# Class parameters
var_class  omega_b        0.02222   0.      0.1    0.0001
var_class  omega_cdm      0.1197    0.      0.3    0.002
var_class  H0             67.0      45.0    90.0   0.1
var_class  tau_reio       0.076     0.01    0.8    0.01
var_class  ln10^{10}A_s   3.096     2.0     4.0    0.01
var_class  n_s            0.977     0.8     1.2    0.01

# Likelihood parameters
var  A_planck  1.  0.9  1.1  0.002
```

Introducing : ECLAIR

```
==========================
== Priors on parameters ==
==========================

--------------------------------------------------------------------------------
-- Uniform and Gaussian priors are implemented, with respective syntax:        --
-- * uni_prior     par_name  lower   upper                                      --
-- * gauss_prior   par_name  mean    stddev                                     --
--------------------------------------------------------------------------------

gauss_prior  A_planck  1.  0.0025
```

Introducing : ECLAIR

```
======================
== Fixed parameters ==
======================

-------------------------------------------------------------------------------
-- Syntax should be "type  par_name  value" where:                            --
-- * "type" : "fix_class" if a CLASS parameter, otherwise "fix"               --
-- * "par_name": parameter name (as known in CLASS or likelihoods)            --
-- * "value": parameter value (can be a string)                               --
-------------------------------------------------------------------------------

# CLASS parameters
fix_class  output          tCl,pCl,lCl
fix_class  lensing         yes
fix_class  l_max_scalars   2508
fix_class  "non linear"    halofit
fix_class  modes           s
fix_class  N_ur            2.0328
fix_class  N_ncdm          1
fix_class  m_ncdm          0.06
```

Introduction

My “needs”:

- Juggling with many cosmological models (and as many Boltzmann solvers)
- Non trivial exploration of parameter space (priors, constraints…)
- A (relatively) big cluster to exploit

Introducing : ECLAIR

- Two (fairly) short files in Python 2/3 : main (~200) & parser (~500)
- Human-readable/tweakable, well-commented (I hope !)
- Working with any CLASS variant, no modification required

Introducing : ECLAIR

```
===========
== CLASS ==
===========

------------------------------------------------------------------------------
-- Select the version of CLASS to be used (give name of Python wrapper)      --
------------------------------------------------------------------------------
which_class  classy
```

Introducing : ECLAIR

- Two (fairly) short files in Python 2/3 : main (~200) & parser (~500)
- Human-readable/tweakable, well-commented (I hope !)
- Working with any CLASS variant, no modification required
- Growing number of likelihoods/datasets implemented (easy to add new ones)

Introducing : ECLAIR

```
=================
== Likelihoods ==
=================

-------------------------------------------------------------------------------
-- Syntax should be "likelihood name_of_likelihood"                           --
-- For a detailed list of available likelihoods, see likelihoods.md in        --
-- "likelihoods" folder                                                       --
-------------------------------------------------------------------------------
likelihood Planck2018_highTTTEEElite
```

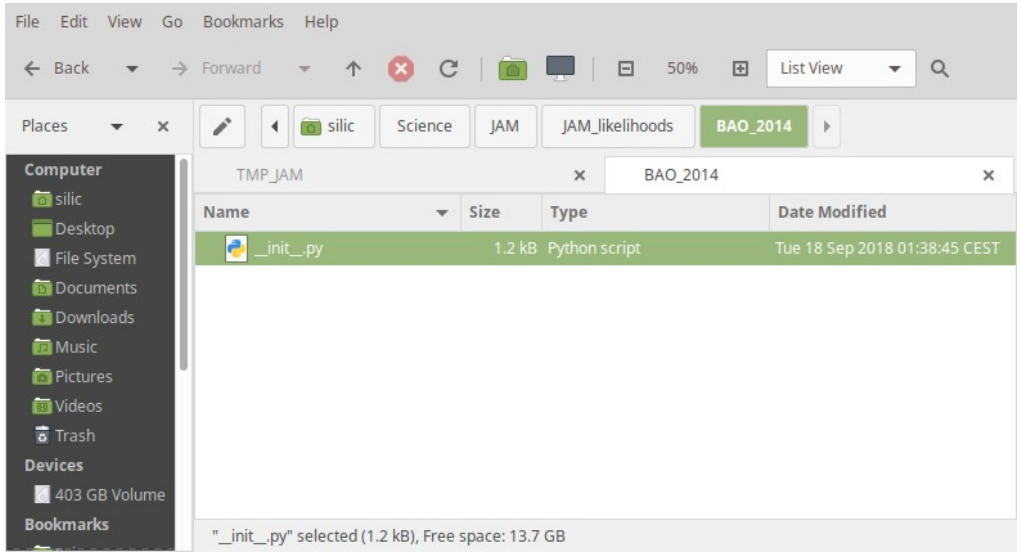
master — ECLAIR / likelihoods /

s-ilic Added SPT3G_2020 likelihood

Name

- ..
- ACTPol_lite_DR4_all
- ACTPol_lite_DR4_for_Planck
- ACTPol_lite_DR4_onlyEE
- ACTPol_lite_DR4_onlyTE
- ACTPol_lite_DR4_onlyTT
- BAO_2014
- BAO_2018
- BK15
- BK18
- H0_F20
- H0_HST
- H0_R18
- H0_R19
- JLA
- Pantheon
- Planck2015_highTT
- Planck2015_highTTTEEE
- Planck2015_highTTTEEElite
- Planck2015_highTTlite
- Planck2015_lensT
- Planck2015_lensTP
- Planck2015_lowTEB
- Planck2015_lowTT
- Planck2018_highTT
- Planck2018_highTTTEEE
- Planck2018_highTTTEEElite
- Planck2018_highTTlite
- Planck2018_lensCMBdep
- Planck2018_lensCMBmarg
- Planck2018_lowBB
- Planck2018_lowEE
- Planck2018_lowEEBB
- Planck2018_lowTT
- SPT3G_2020

Introducing : ECLAIR




```python
import numpy as np

### BAO "2014" data (used in Planck 2015 as ext. data)
def get_loglike(class_input, likes_input, class_run):
    lnl = 0.
    rs = class_run.rs_drag()
    # 6DF from 1106.3366
    z, data, error = 0.106, 0.327, 0.015
    da = class_run.angular_distance(z)
    dr = z / class_run.Hubble(z)
    dv = (da**2. * (1 + z)**2. * dr)**(1. / 3.)
    theo = rs / dv
    lnl += -0.5 * (theo - data)**2. / error**2.
    # BOSS LOWZ & CMASS DR10&11 from 1312.4877
    z, data, error = 0.32, 8.47, 0.17
    da = class_run.angular_distance(z)
    dr = z / class_run.Hubble(z)
    dv = (da**2. * (1 + z)**2. * dr)**(1. / 3.)
    theo = dv / rs
    lnl += -0.5 * (theo - data)**2. / error**2.
    z, data, error = 0.57, 13.77, 0.13
    da = class_run.angular_distance(z)
    dr = z / class_run.Hubble(z)
    dv = (da**2. * (1 + z)**2. * dr)**(1. / 3.)
    theo = dv / rs
    lnl += -0.5 * (theo - data)**2. / error**2.
    # SDSS DR7 MGS from 1409.3242
    z, data, error = 0.15, 4.47, 0.16
    da = class_run.angular_distance(z)
    dr = z / class_run.Hubble(z)
    dv = (da**2. * (1 + z)**2. * dr)**(1. / 3.)
    theo = dv / rs
    lnl += -0.5 * (theo - data)**2. / error**2.
    # Return log(like)
    return lnl
```

Introduction

My “needs”:

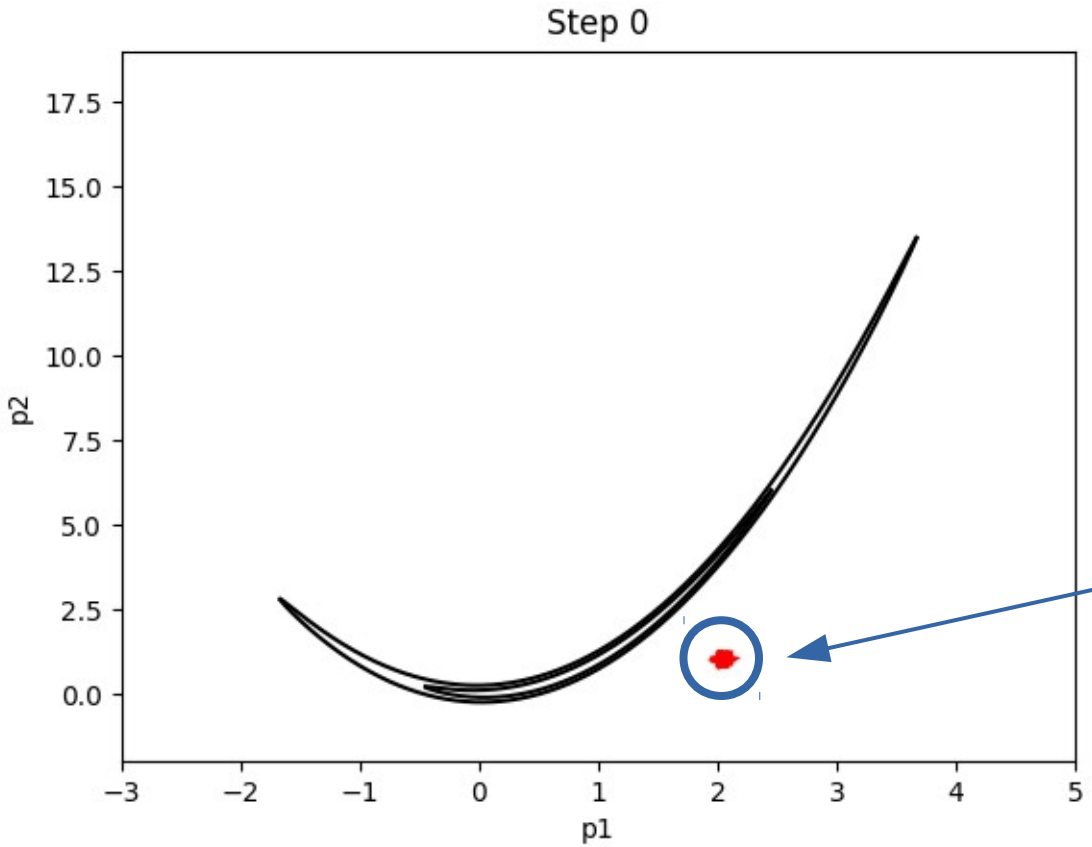
- Juggling with many cosmological models (and as many Boltzmann solvers)
- Non trivial exploration of parameter space (priors, constraints…)
- A (relatively) big cluster to exploit

Introducing : ECLAIR

- Two (fairly) short files in Python 2/3 : main (~200) & parser (~500)
- Human-readable/tweakable, well-commented (I hope !)
- Working with any CLASS variant, no modification required
- Growing number of likelihoods/datasets implemented (easy to add new ones)
- MCMC algorithm : Affine-Invariant Ensemble sampling

Ensemble sampling

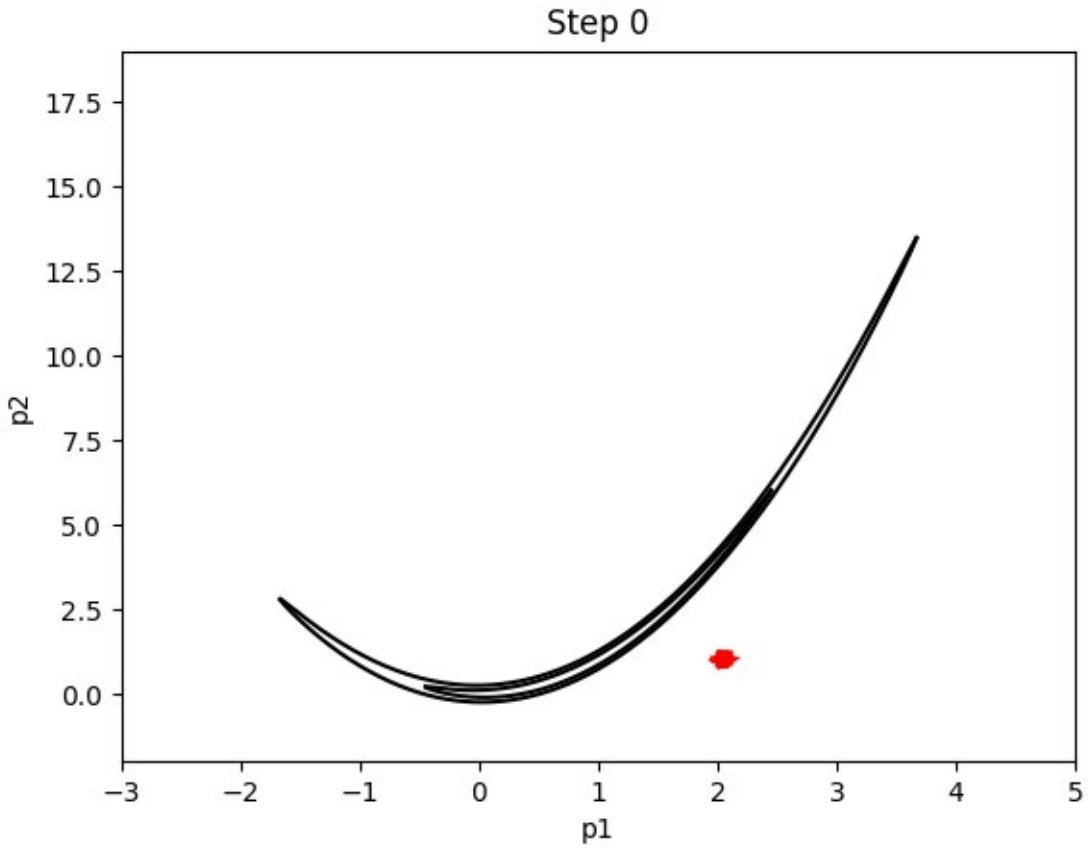
$$\pi(x) \propto \exp\left(\frac{-(x_1 - x_2)^2}{2\epsilon} - \frac{(x_1 + x_2)^2}{2}\right)$$



Collection of "walkers" initialized at random positions

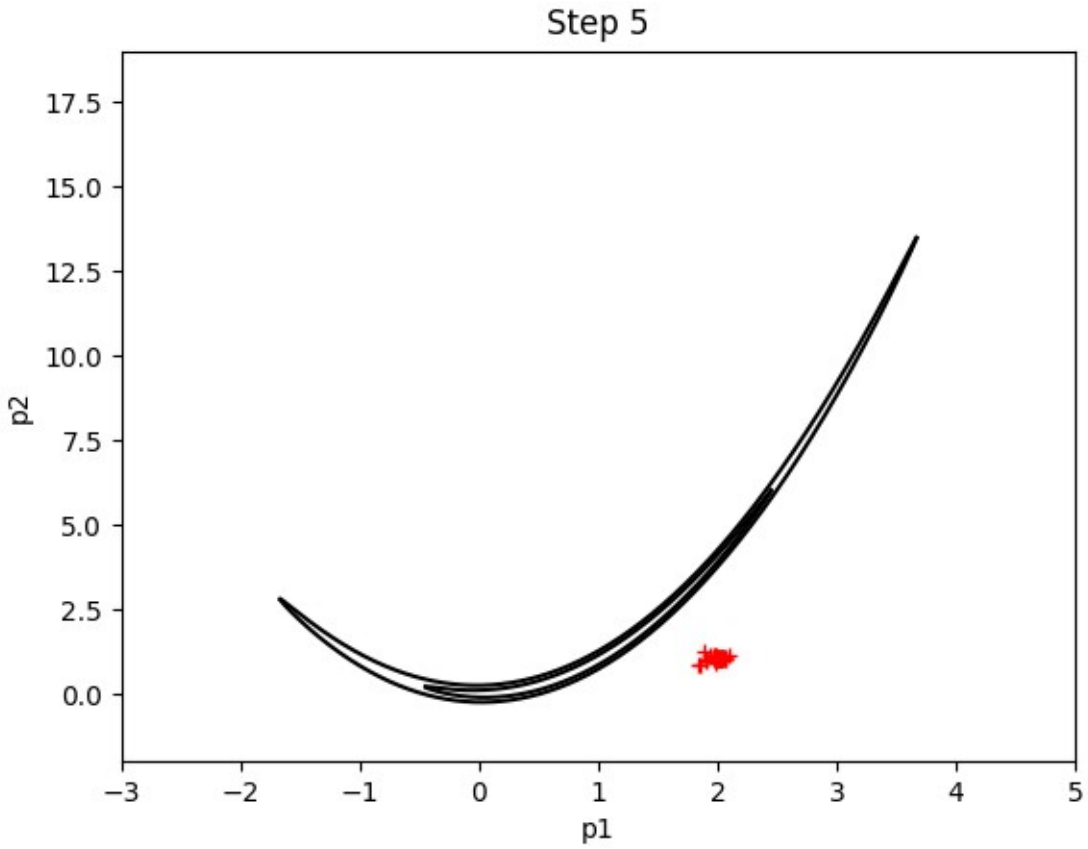
Ensemble sampling

$$\pi(x) \propto \exp\left(\frac{-(x_1 - x_2)^2}{2\epsilon} - \frac{(x_1 + x_2)^2}{2}\right)$$



Ensemble sampling

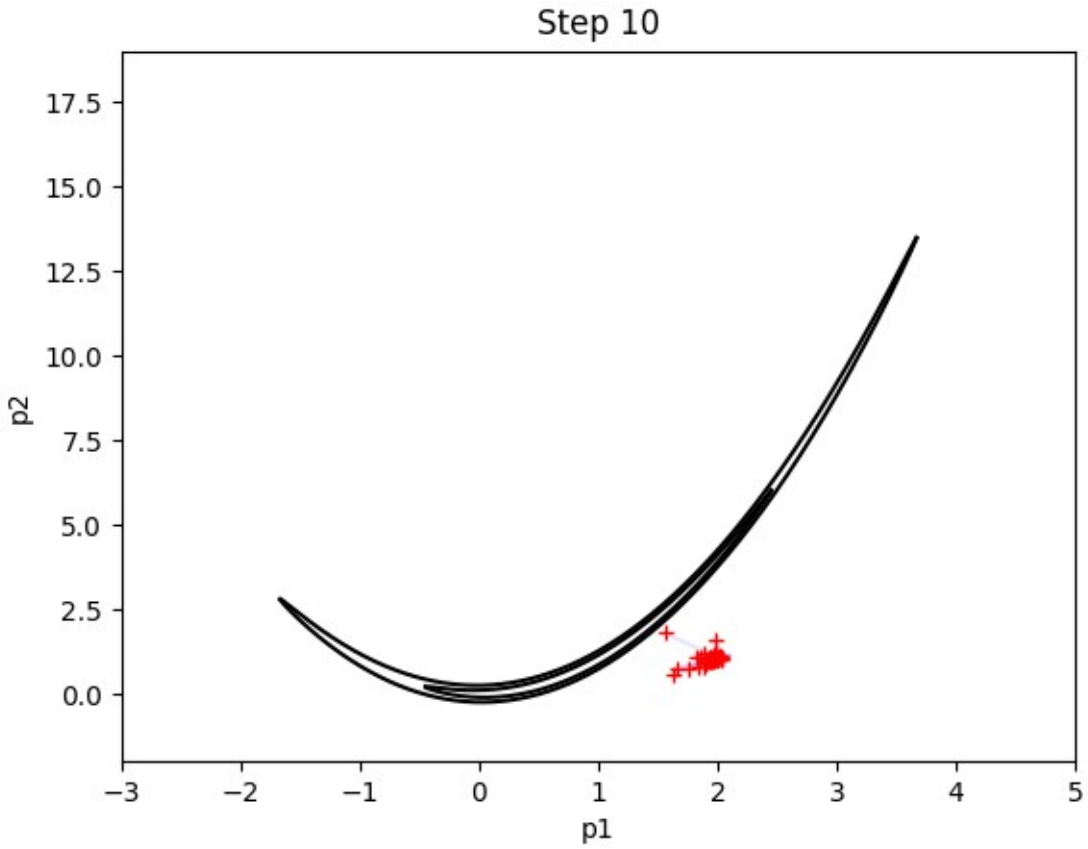
$$\pi(x) \propto \exp\left(\frac{-(x_1 - x_2)^2}{2\epsilon} - \frac{(x_1 + x_2)^2}{2}\right)$$



"Walkers" quickly spread throughout parameter space, using each other's position to propose jumps

Ensemble sampling

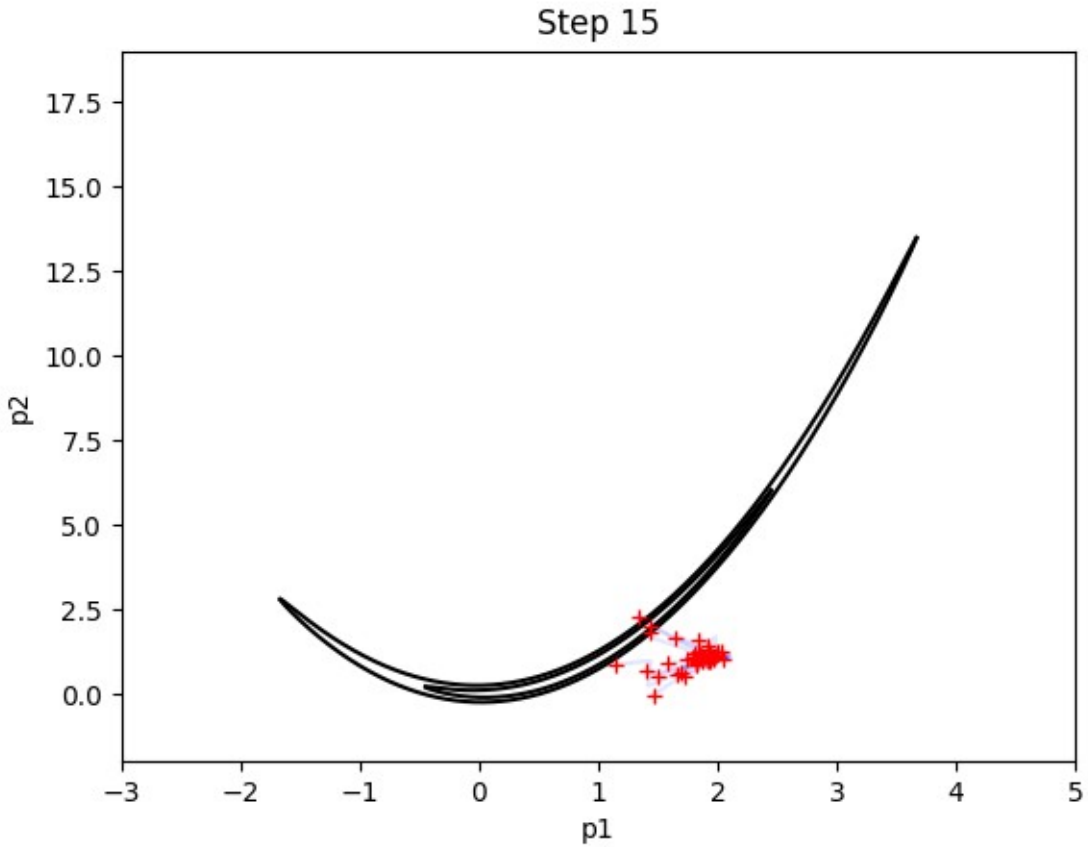
$$\pi(x) \propto \exp\left(\frac{-(x_1 - x_2)^2}{2\epsilon} - \frac{(x_1 + x_2)^2}{2}\right)$$



"Walkers" quickly spread throughout parameter space, using each other's position to propose jumps

Ensemble sampling

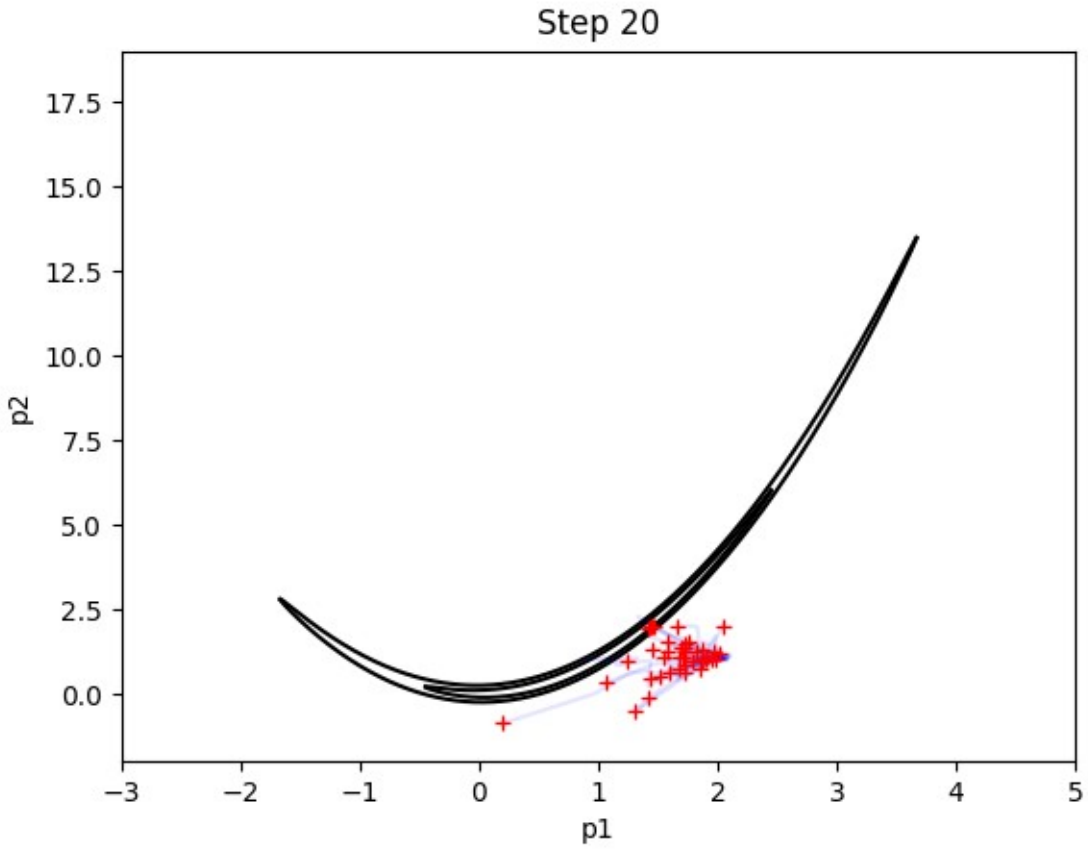
$$\pi(x) \propto \exp\left(\frac{-(x_1 - x_2)^2}{2\epsilon} - \frac{(x_1 + x_2)^2}{2}\right)$$



"Walkers" quickly spread throughout parameter space, using each other's position to propose jumps

Ensemble sampling

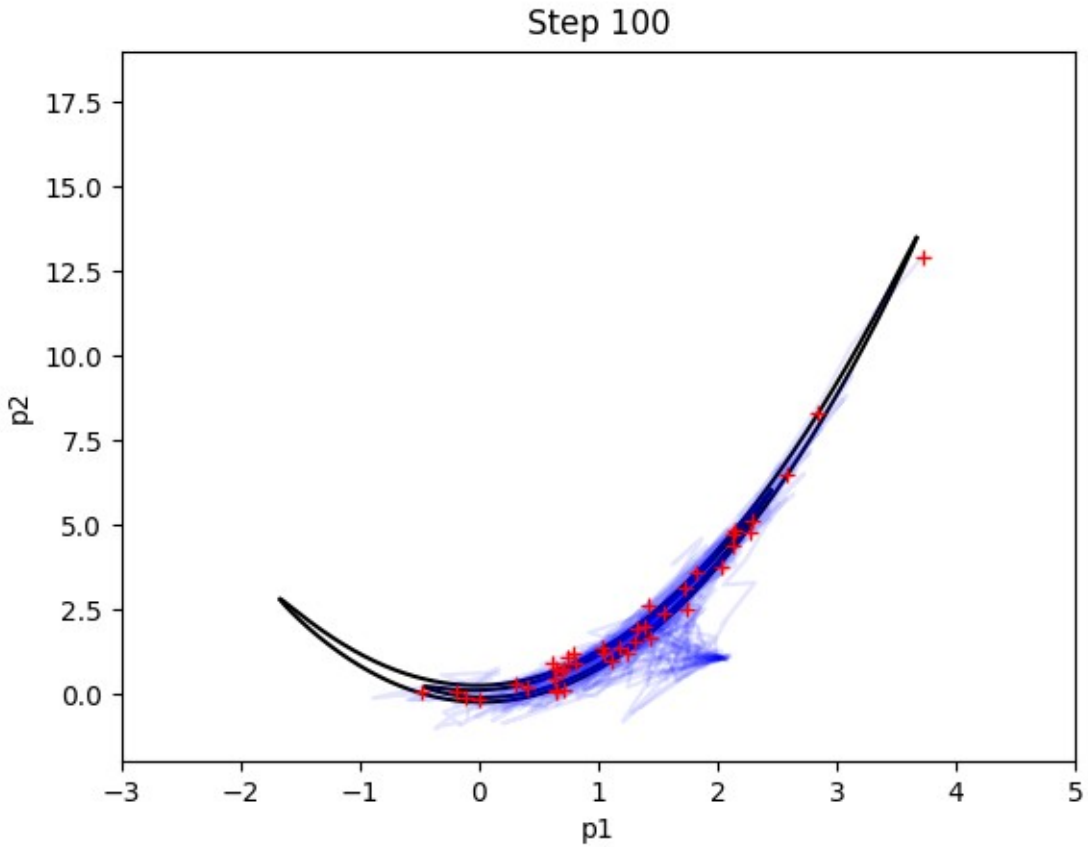
$$\pi(x) \propto \exp\left(\frac{-(x_1 - x_2)^2}{2\epsilon} - \frac{(x_1 + x_2)^2}{2}\right)$$



"Walkers" quickly spread throughout parameter space, using each other's position to propose jumps

Ensemble sampling

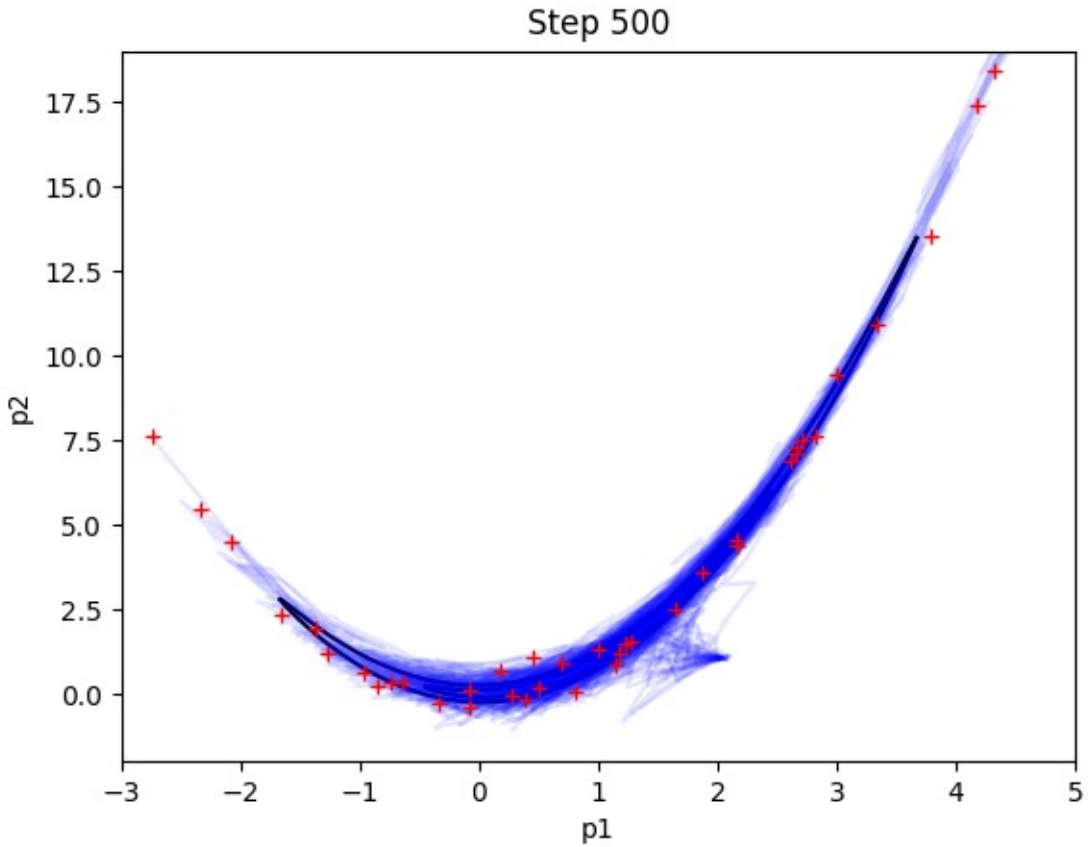
$$\pi(x) \propto \exp\left(\frac{-(x_1 - x_2)^2}{2\epsilon} - \frac{(x_1 + x_2)^2}{2}\right)$$



...and end up sitting in the "interesting" region of parameter space

Ensemble sampling

$$\pi(x) \propto \exp\left(\frac{-(x_1 - x_2)^2}{2\epsilon} - \frac{(x_1 + x_2)^2}{2}\right)$$



A single snapshot of walkers positions
=
A representative sample of the posterior distribution

Introducing : ECLAIR

```
==========
== MCMC ==
==========

------------------------------------------------------------
-- Select the MCMC sampler to be used (default "emcee") --
------------------------------------------------------------------------------
-- Possible choices:                                                        --
--   * "emcee" (https://emcee.readthedocs.io)                               --
--   * "zeus" (https://zeus-mcmc.readthedocs.io)                            --
------------------------------------------------------------------------------
#which_sampler  emcee
```

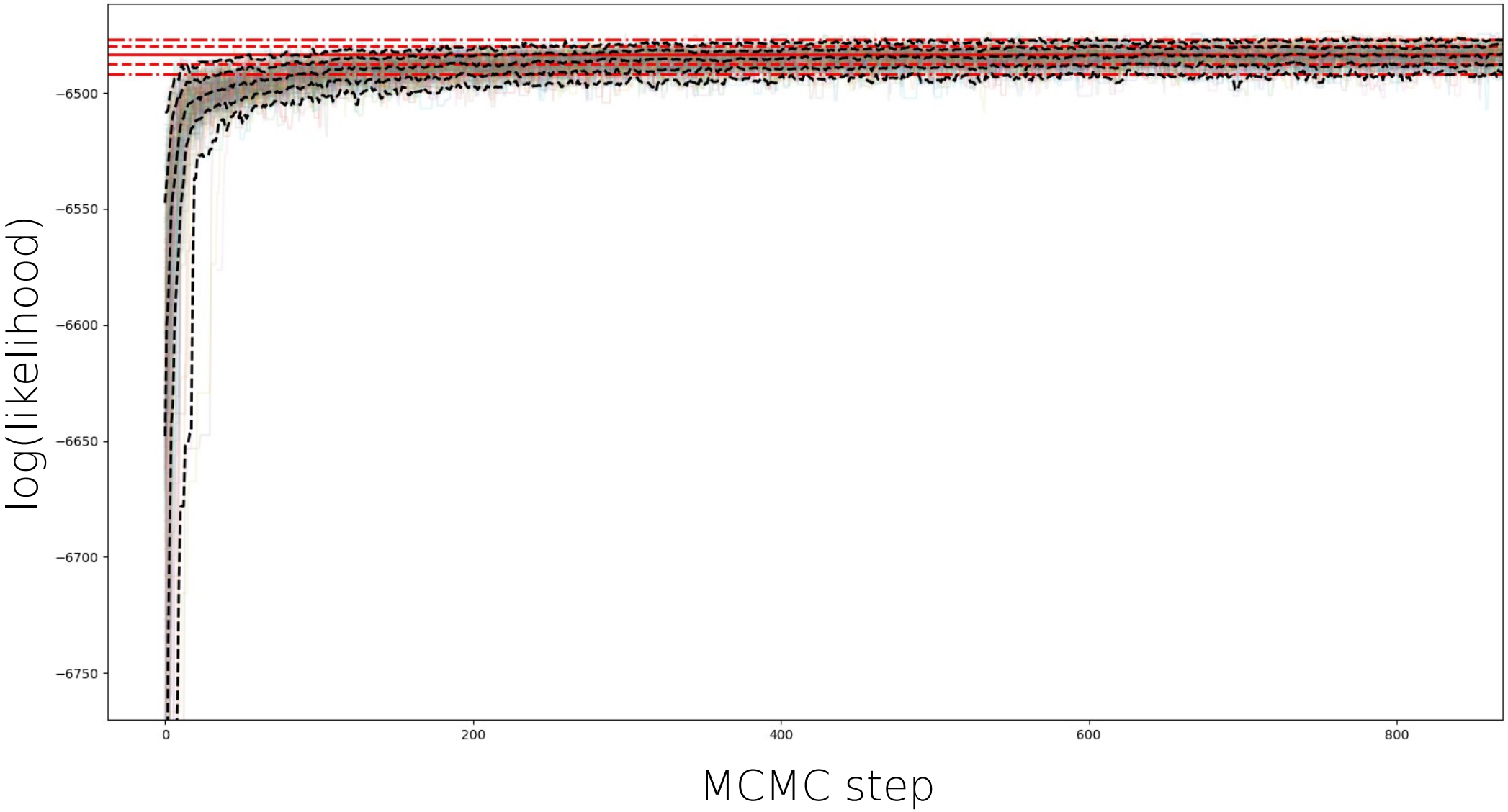
Introducing : ECLAIR

```
-----------------------------------------------------
-- Setting for parallel computing (default "none") --
-------------------------------------------------------------------------------
-- Current choices :                                                         --
-- * "none": no parallelization                                              --
-- * "multiprocessing N": OpenMP parallelization with N threads              --
-- * "MPI": MPI parallelization (requires "schwimmbad" python module)        --
-------------------------------------------------------------------------------
#parallel  none

-----------------------------------------------------
-- Set the number of walkers (default "prop_to 2") --
-------------------------------------------------------------------------------
-- Current choices :                                                         --
-- * "custom  N": number of walkers fixed to X                               --
-- * "prop_to  N": number of walkers is N times the number of free parameters --
-- Note:                                                                     --
-- Number of walkers has to be at least twice the number of free parameters  --
-------------------------------------------------------------------------------
#n_walkers  prop_to  2

--------------------------
-- Number of MCMC steps --
--------------------------
n_steps  10000

------------------------------------------------
-- MCMC thinning factor (default no thinning) --
------------------------------------------------
#thin_by  1

------------------------------------------------
-- MCMC temperature of the MCMC (default 1.) --
------------------------------------------------
#temperature  1.

-------------------------------------------
-- Optional keyword arguments for sampler --
-------------------------------------------------------------------------------
-- Note:                                                                     --
-- Syntax should be "sampler_kwarg  kwarg_name  kwarg"                       --
-------------------------------------------------------------------------------
#sampler_kwarg  moves  MCMCsampler.moves.StretchMove(a=2)
```

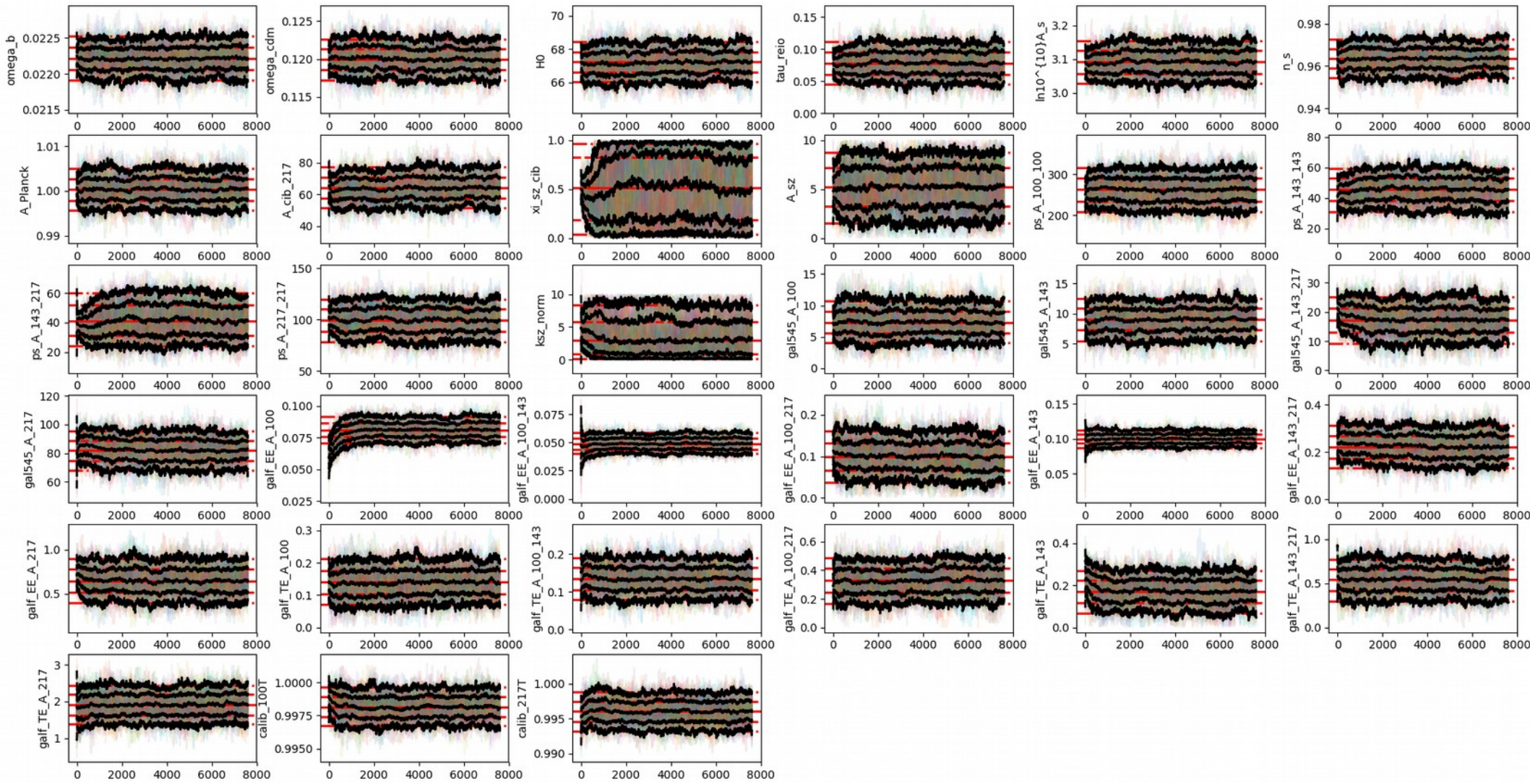
Introducing : ECLAIR

- Two (fairly) short files in Python 2/3 : main (~200) & parser (~500)
- Human-readable/tweakable, well-commented (I hope !)
- Working with any CLASS variant, no modification required
- Growing number of likelihoods/datasets implemented (easy to add new ones)
- MCMC algorithm : Affine-Invariant Ensemble sampling
- Intuitive visualization scripts to assess convergence

Visualization tools



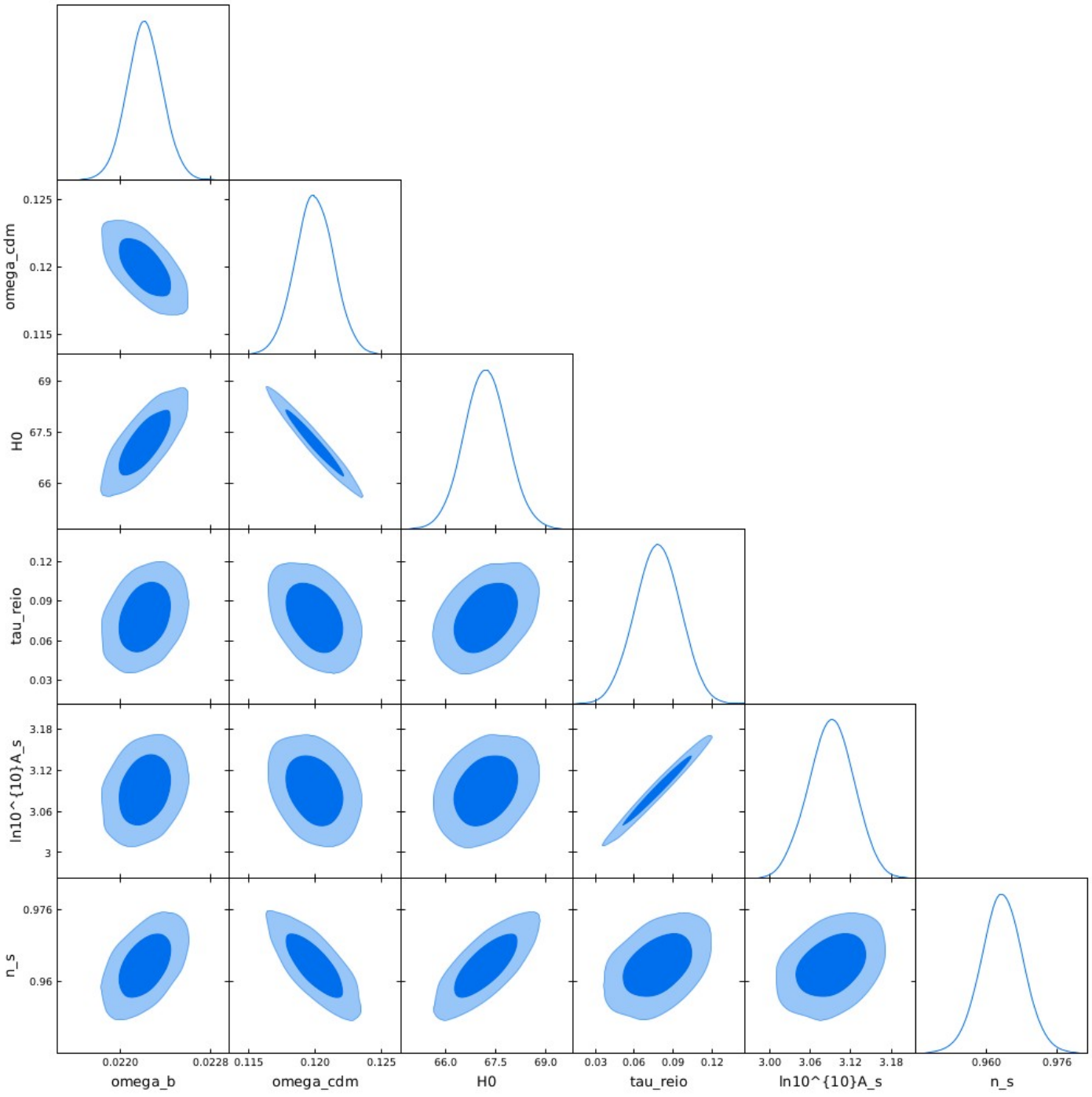
Visualization tools



Introducing : ECLAIR

- Two (fairly) short files in Python 2/3 : main (~200) & parser (~500)
- Human-readable/tweakable, well-commented (I hope !)
- Working with any CLASS variant, no modification required
- Growing number of likelihoods/datasets implemented (easy to add new ones)
- MCMC algorithm : Affine-Invariant Ensemble sampling
- Intuitive visualization scripts to assess convergence
- Contour plot scripts (interfaced with getdist)

Contour plots



Introduction

My “needs”:

- Juggling with many cosmological models (and as many Boltzmann solvers)
- Non trivial exploration of parameter space (priors, constraints…)
- A (relatively) big cluster to exploit

Introducing : ECLAIR

- Two (fairly) short files in Python 2/3 : main (~200) & parser (~500)
- Human-readable/tweakable, well-commented (I hope !)
- Working with any CLASS variant, no modification required
- Growing number of likelihoods/datasets implemented (easy to add new ones)
- MCMC algorithm : Affine-Invariant Ensemble sampling
- Intuitive visualization scripts to assess convergence
- Contour plot scripts (interfaced with getdist)
- Convenient custom parser :“constraint” and “deriv” features

ECLAIR parsing features

```
#################################
## Parameters special settings ##
#################################

----------------------------------
-- Put constraints on parameters --
-------------------------------------------------------------------------------
-- Syntax should be "constraint XXX = YYY", where XXX is the parameter      --
-- forced to be equal to YYY. In XXX and YYY:                               --
-- * YYY can be any fonction of any number of parameters                    --
-- * use syntax class[par_name] if a CLASS parameter involved               --
-- * use syntax likes[par_name] if a likelihood parameter is involved       --
-------------------------------------------------------------------------------
#constraint  class[omega_b]  =  class[omega_cdm] + 5 * class[H0]

-----------------------------------------------
-- Request some derived parameters in output --
-------------------------------------------------------------------------------
-- Syntax should be "deriv  par_name  quantity_requested", where:           --
-- * "par_name": name of derived parameter as will be stored in chain (cannot --
--               contain any space)                                         --
-- * "quantity_requested": any command/function, where:                     --
--     - CLASS wrapper accessible via "class_run" instance                  --
--     - CLASS background quantities accessible via "bg" dictionnary        --
--     - CLASS thermodynamical quantities accessible via "th" dictionnary   --
--     - CLASS parameters accessible via "class_input" dictionnary          --
--     - nuisance parameters accessible via "likes_input" dictionnary       --
-------------------------------------------------------------------------------
#deriv  my_H0      bg['H [1/Mpc]'][-1]*299792.458
#deriv  sigma_8    class_run.sigma8()
```

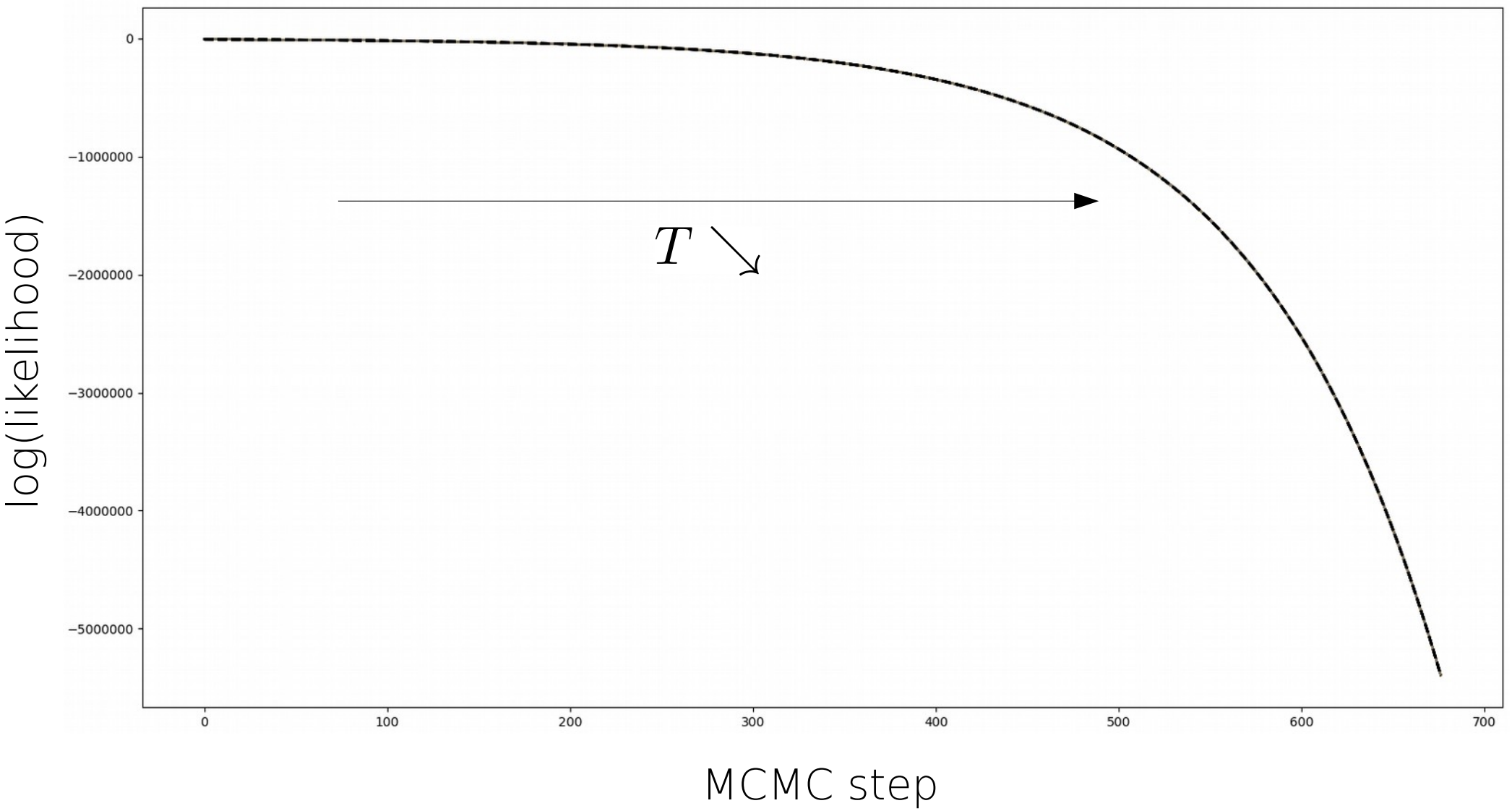
+ can put priors on any derived parameter

Introducing : ECLAIR

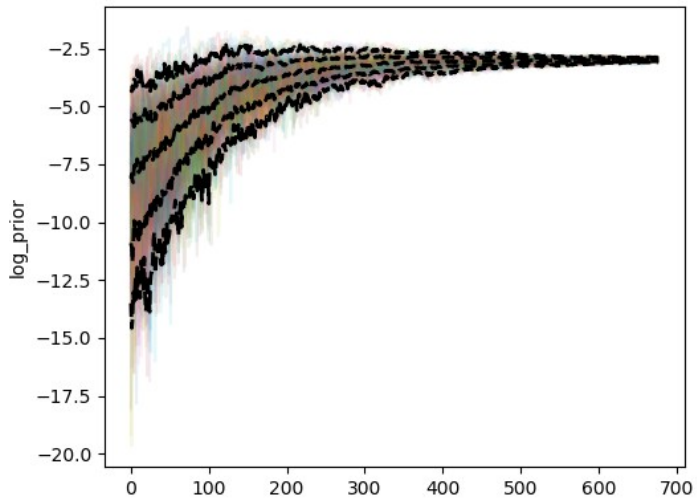
- Two (fairly) short files in Python 2/3 : main (~200) & parser (~500)
- Human-readable/tweakable, well-commented (I hope !)
- Working with any CLASS variant, no modification required
- Growing number of likelihoods/datasets implemented (easy to add new ones)
- MCMC algorithm : Affine-Invariant Ensemble sampling
- Intuitive visualization scripts to assess convergence
- Contour plot scripts (interfaced with getdist)
- Convenient custom parser :“constraint” and “deriv” features
- Robust minimizer combining simulated annealing & ensemble sampling (SAVES ?)

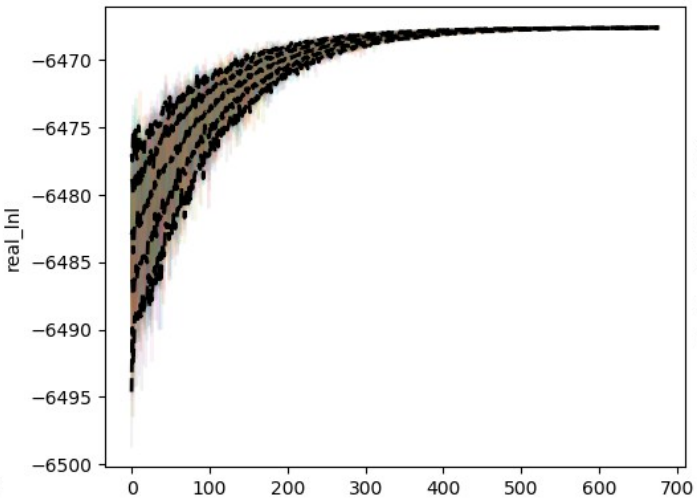
Minimizing with ECLAIR

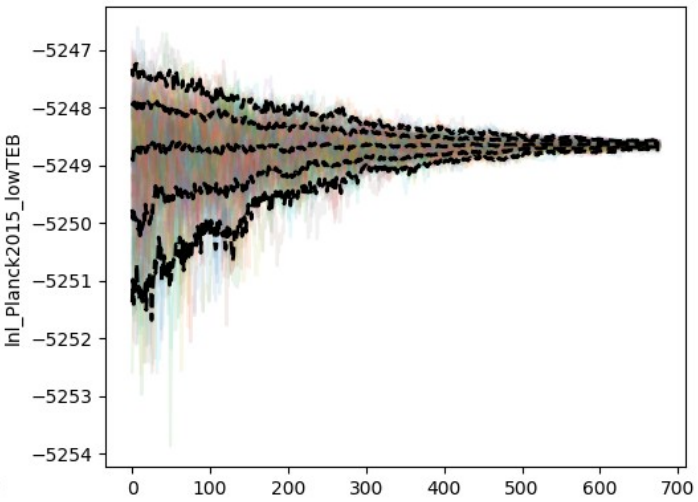
$$\mathcal{L} \longrightarrow \mathcal{L}^{1/T}$$

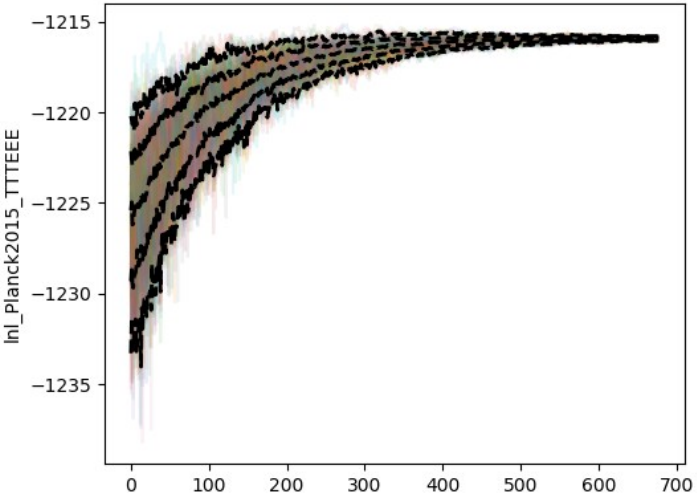


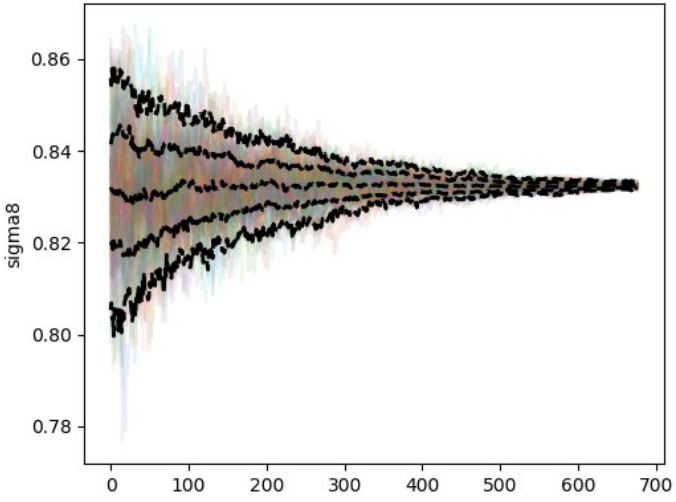
Minimizing with ECLAIR



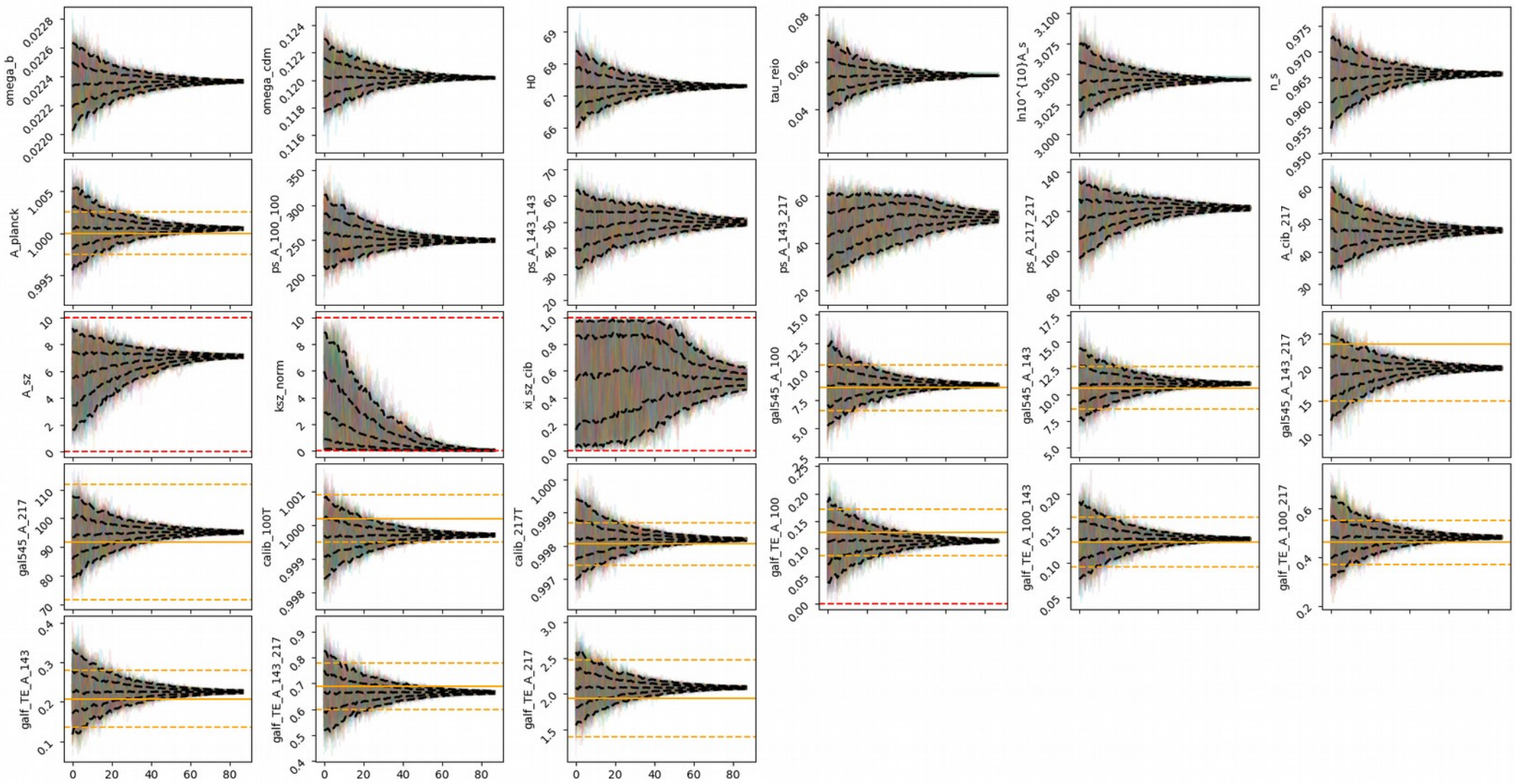








Minimizing with ECLAIR

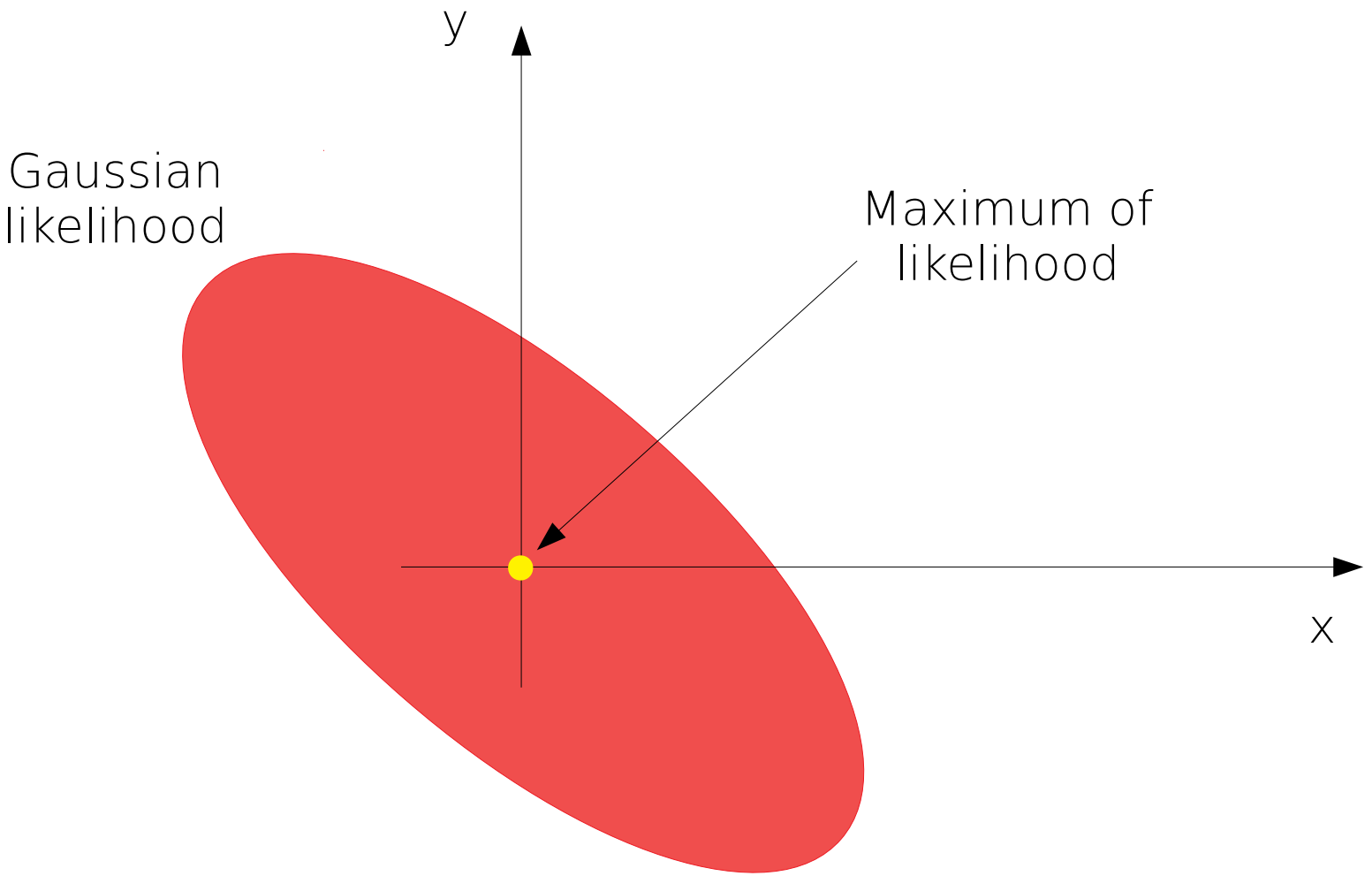


+ Bayesian evidence computing

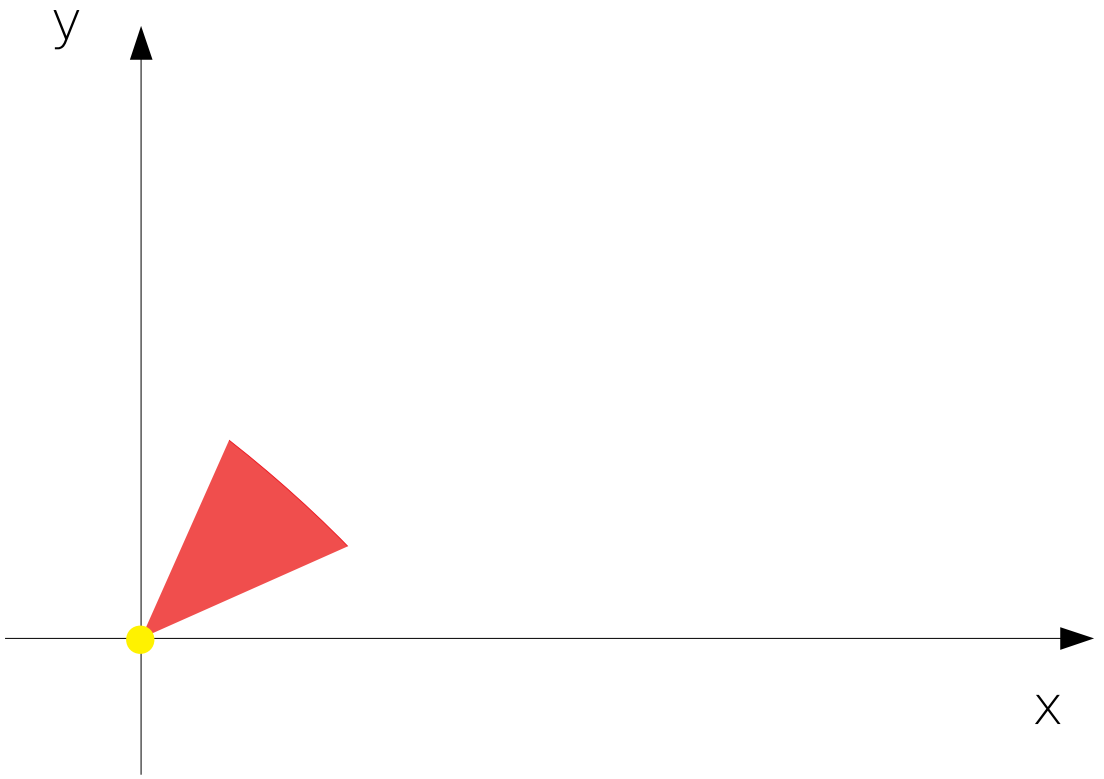
(+ profile likelihood, work in progress)

Identifying prior effects with ECLAIR

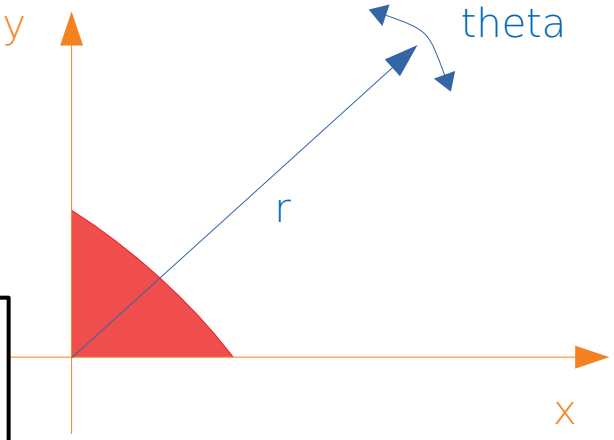
Effects of priors

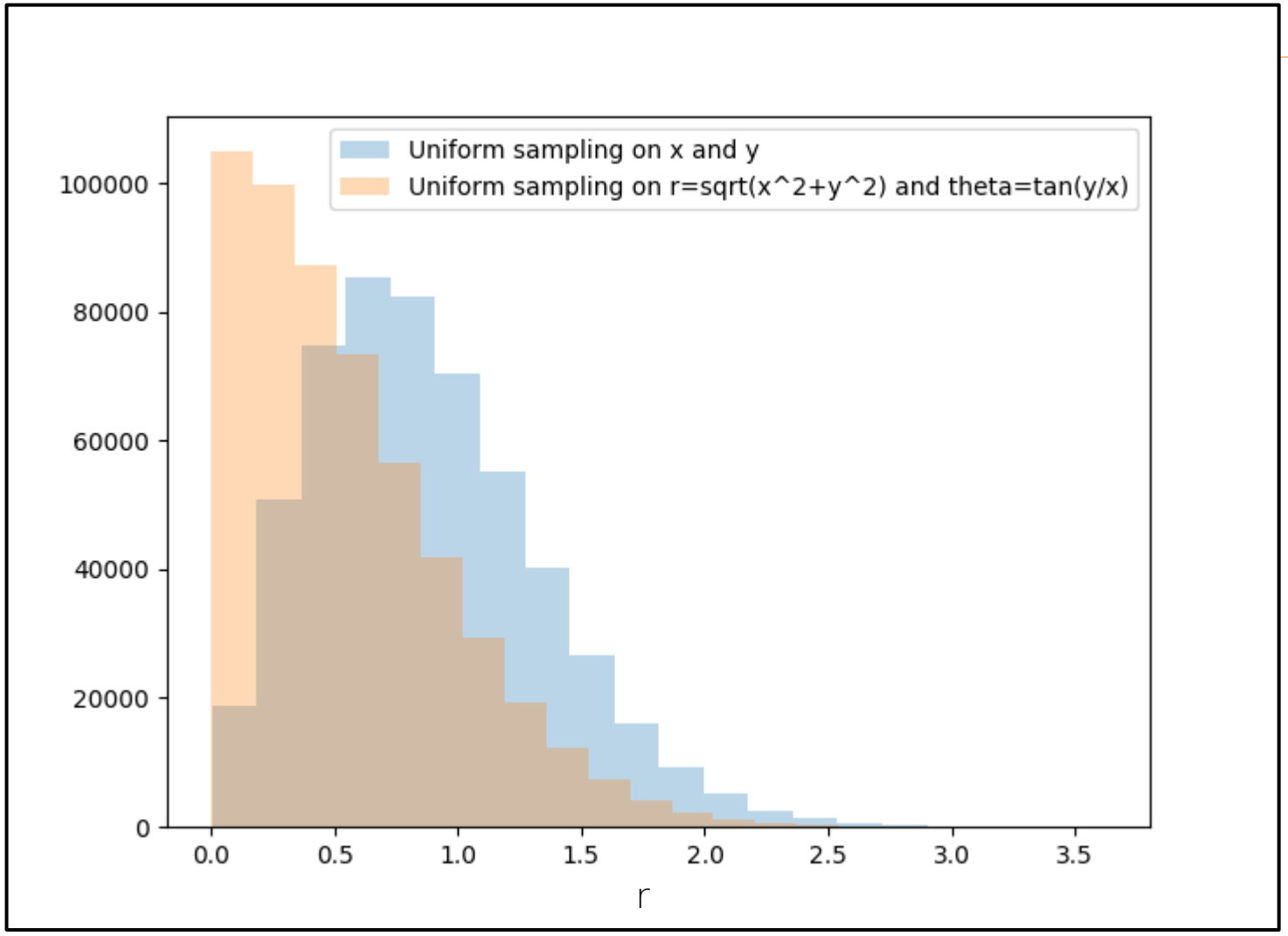


Effects of priors



Effects of priors



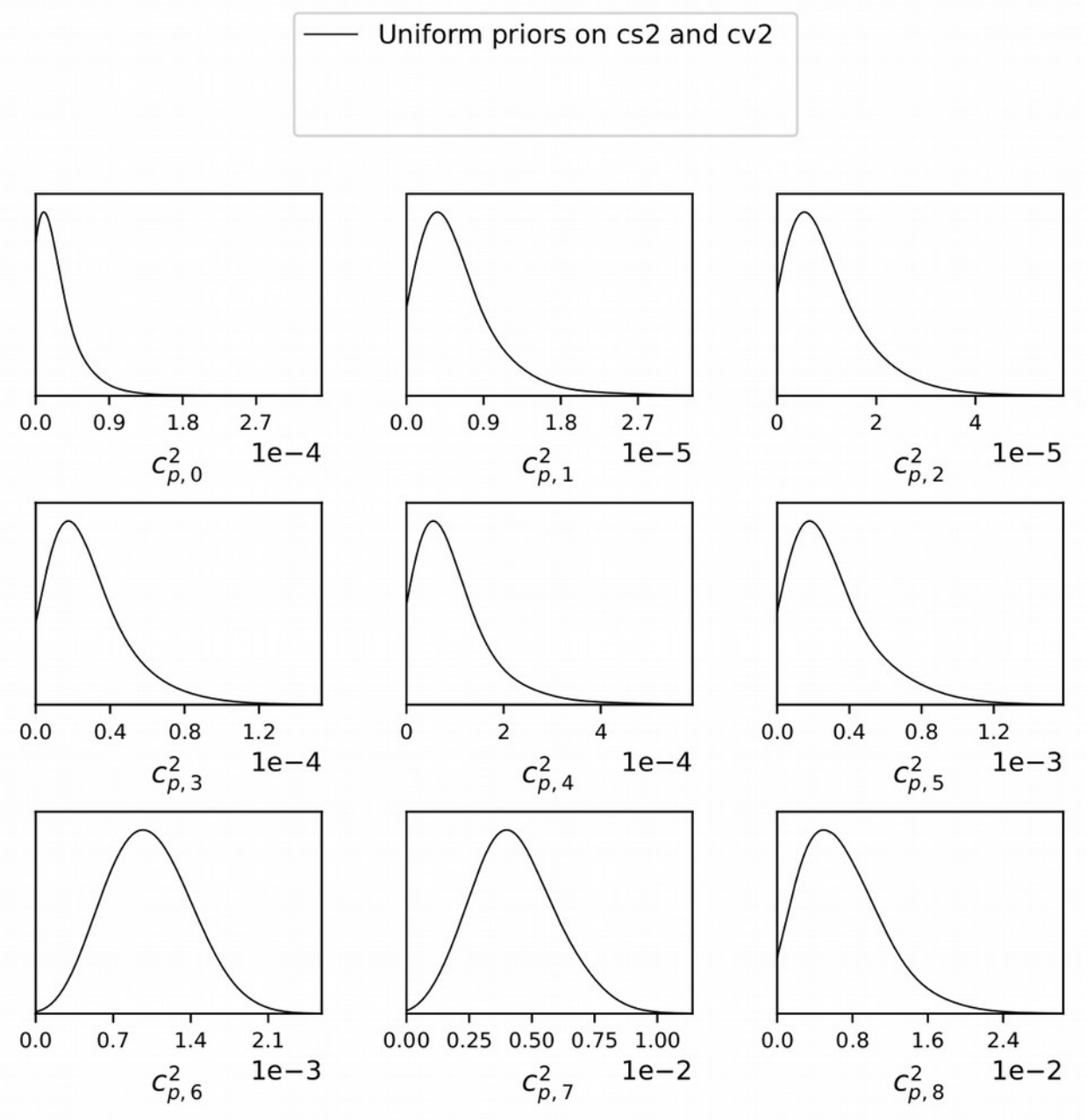


Effects of priors

$$c_p^2 = c_s^2 + \frac{8}{15} c_v^2$$

$$(c_s^2, c_v^2) \quad > 0$$

Ilić et al, 2021



Effects of priors

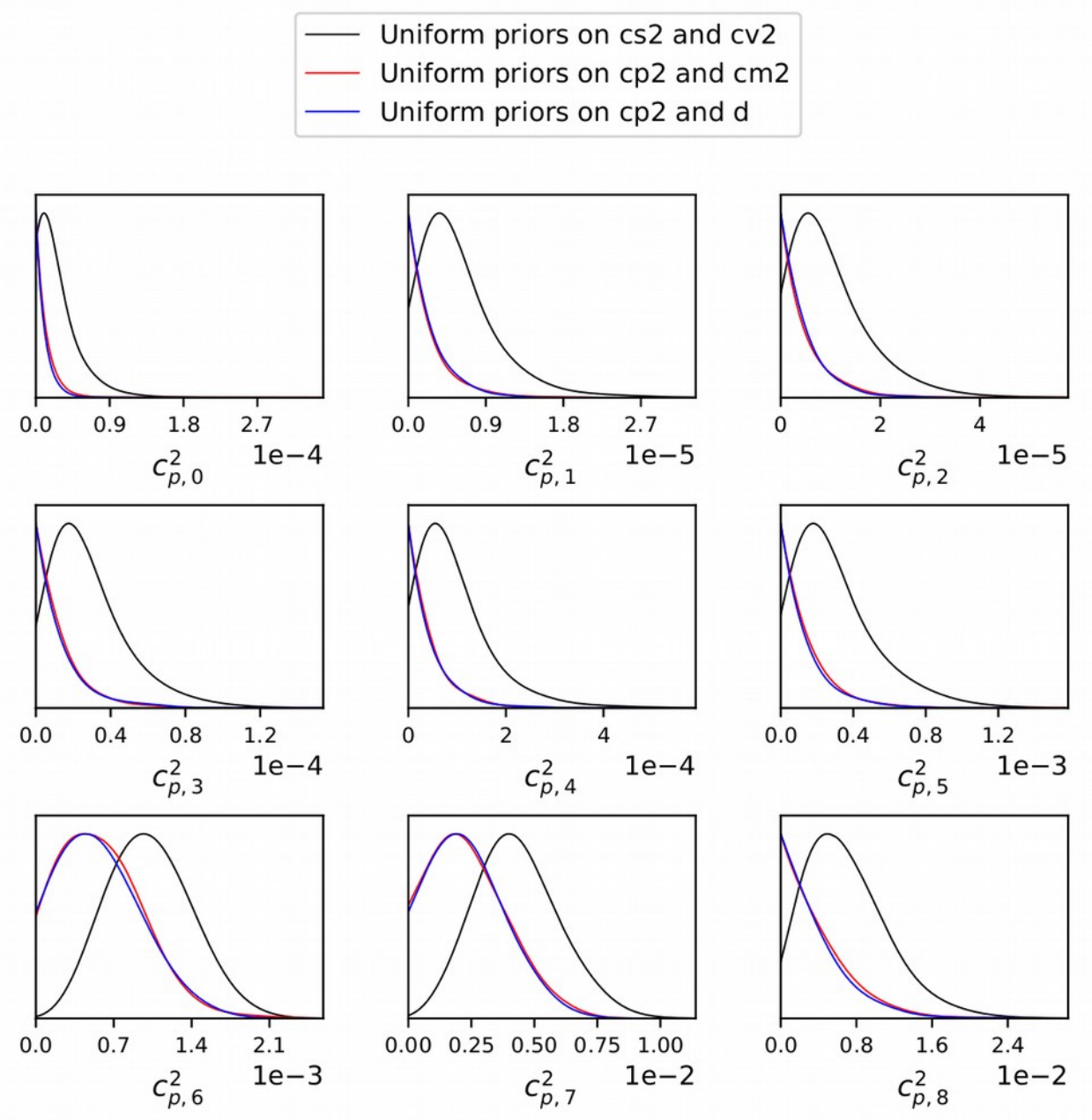
$$c_p^2 = c_s^2 + \frac{8}{15} c_v^2$$

$$(c_s^2, c_v^2) \quad > 0$$

(c_p^2, c_m^2) (c_p^2, d)

$$c_m^2 = \arctan\left(\frac{8}{15}\frac{c_v^2}{c_s^2}\right) \qquad d = \frac{c_s^2}{c_p^2}$$

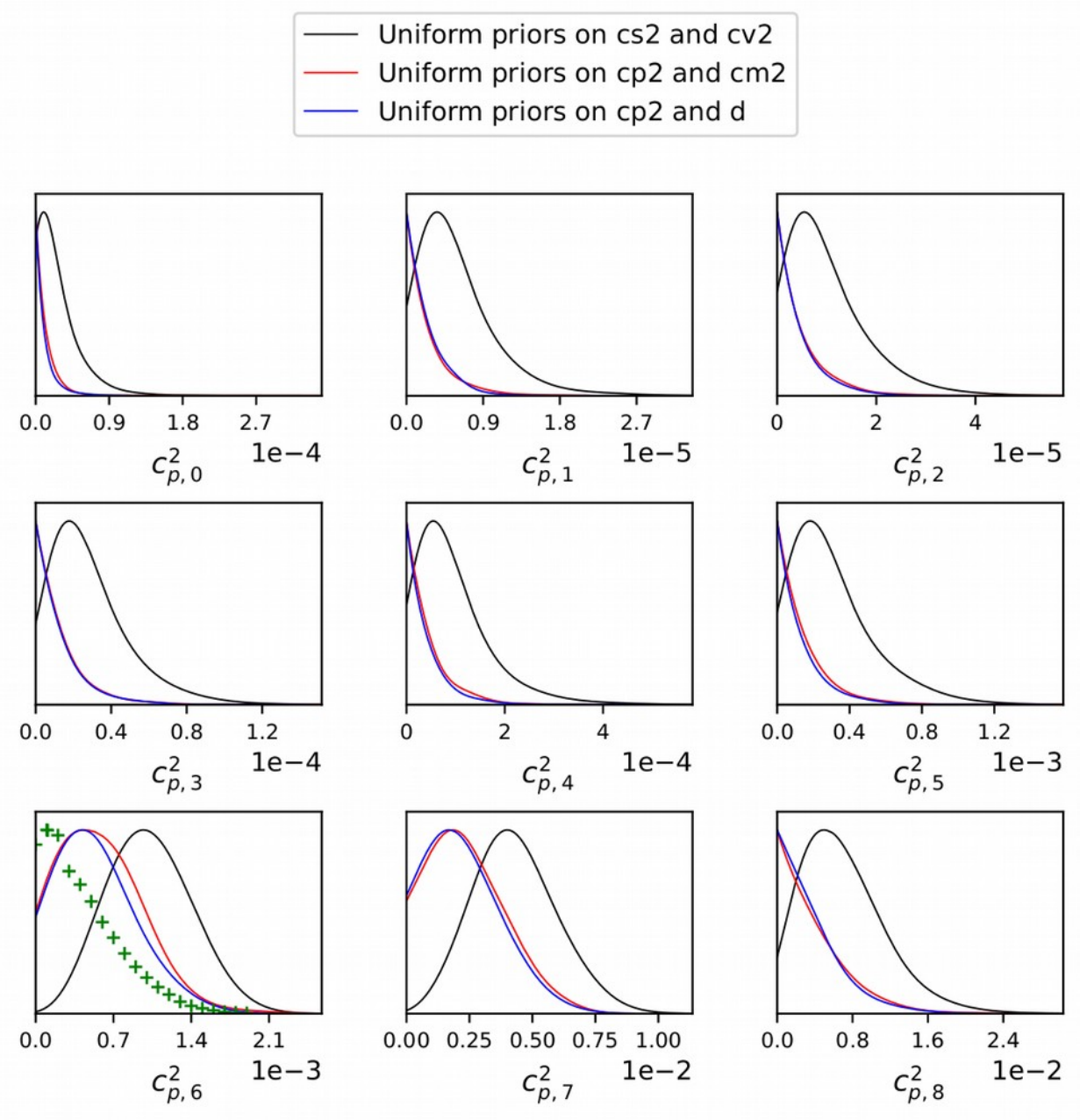
Ilić et al, 2021



Effects of priors

$$c_p^2 = c_s^2 + \frac{8}{15} c_v^2$$

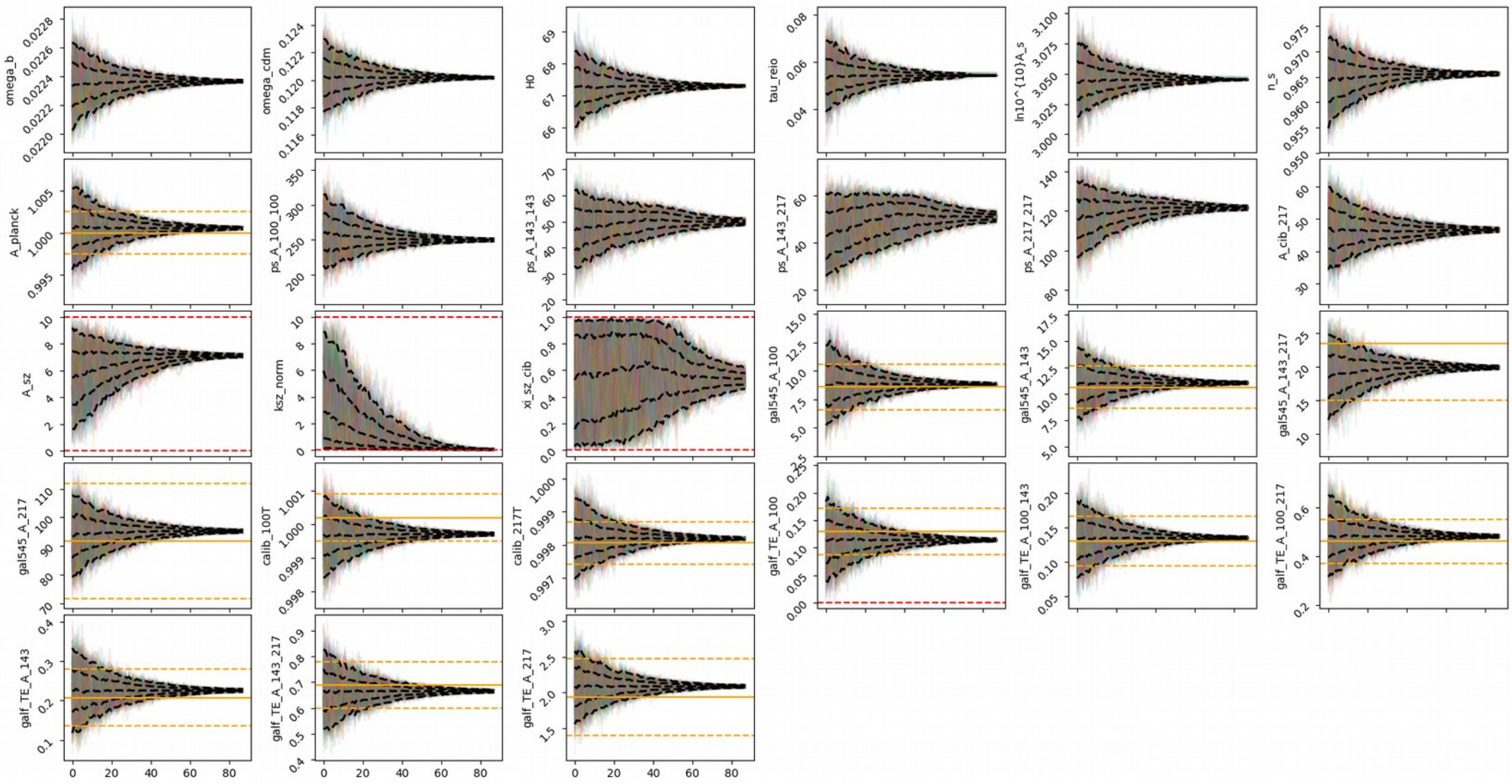
Ilić et al, 2021



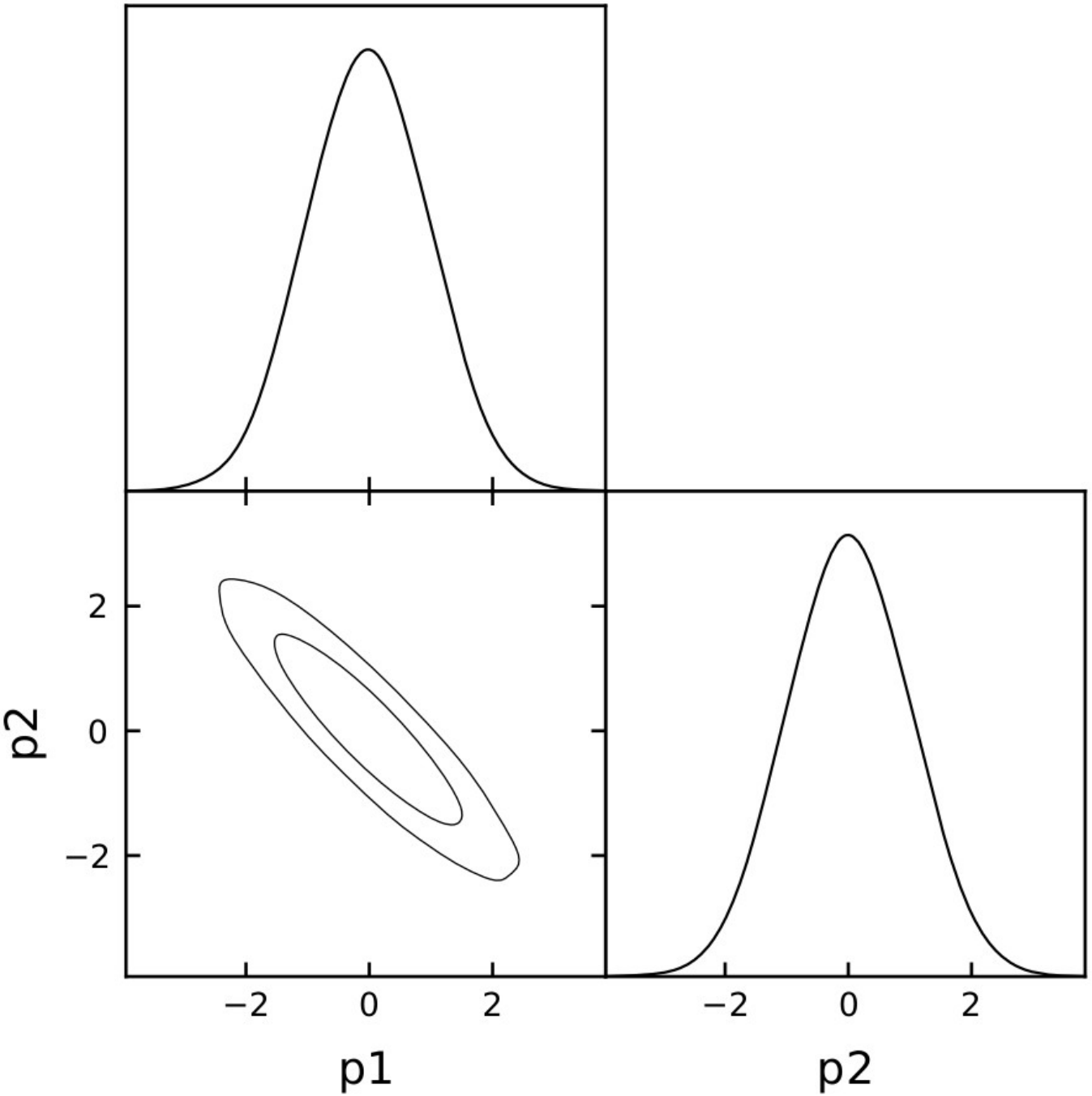
Frequentist approach :
Computation of the
“profile likelihood”
=
1D grid on given parameter,
minimize likelihood wrt
all other parameters

Identifying prior effects with ECLAIR

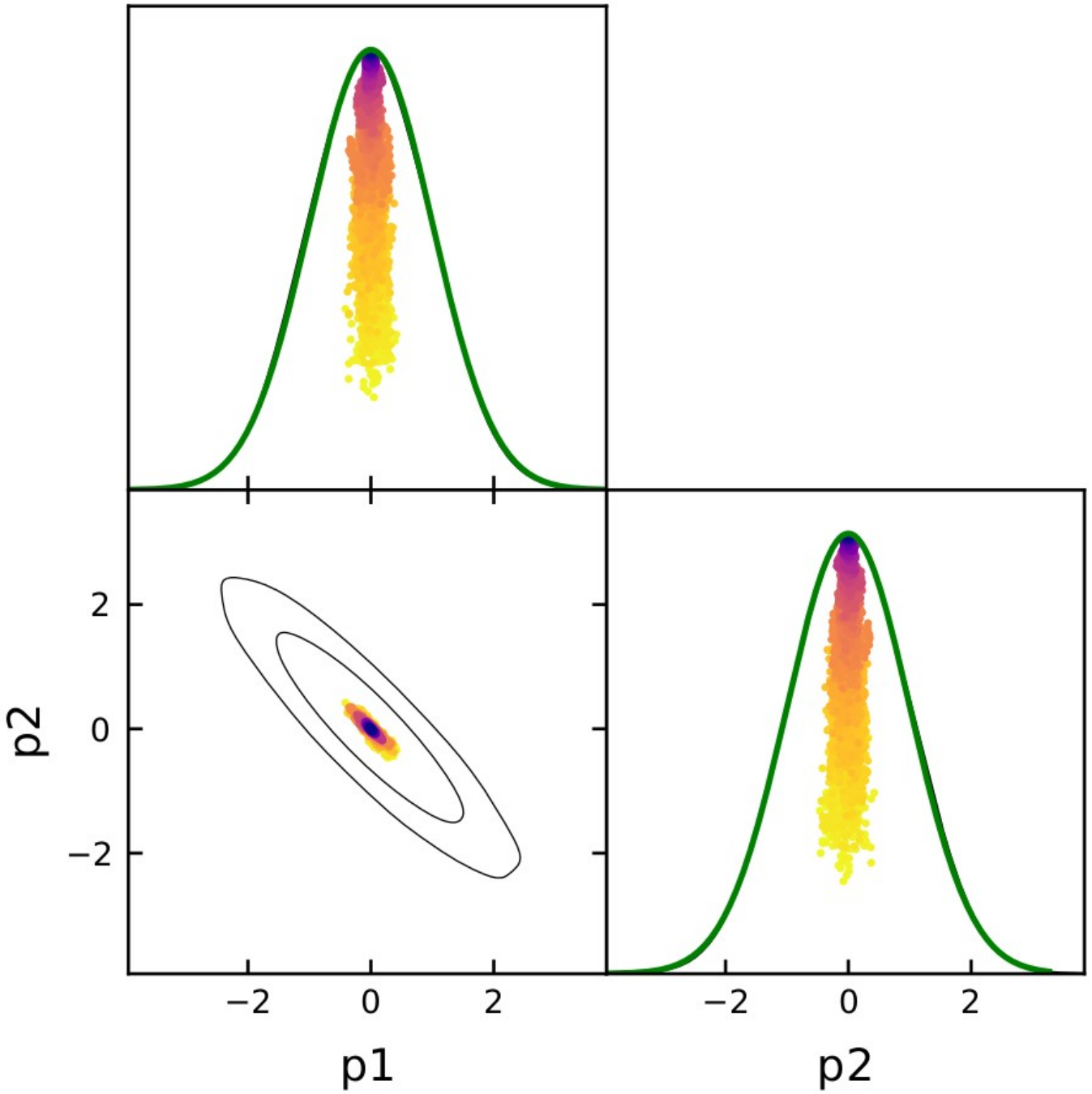
Identifying prior effects with ECLAIR



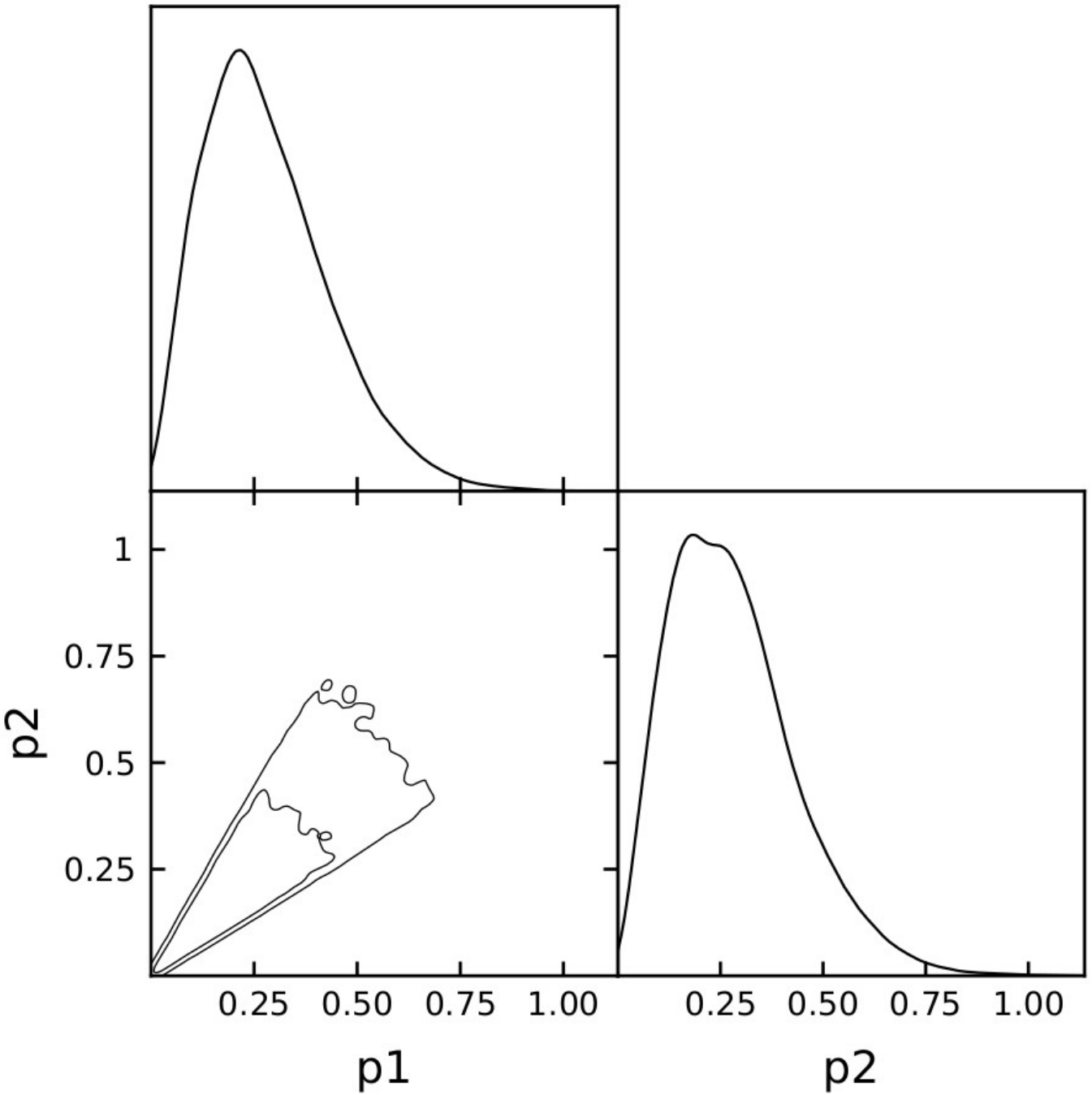
Identifying prior effects with ECLAIR



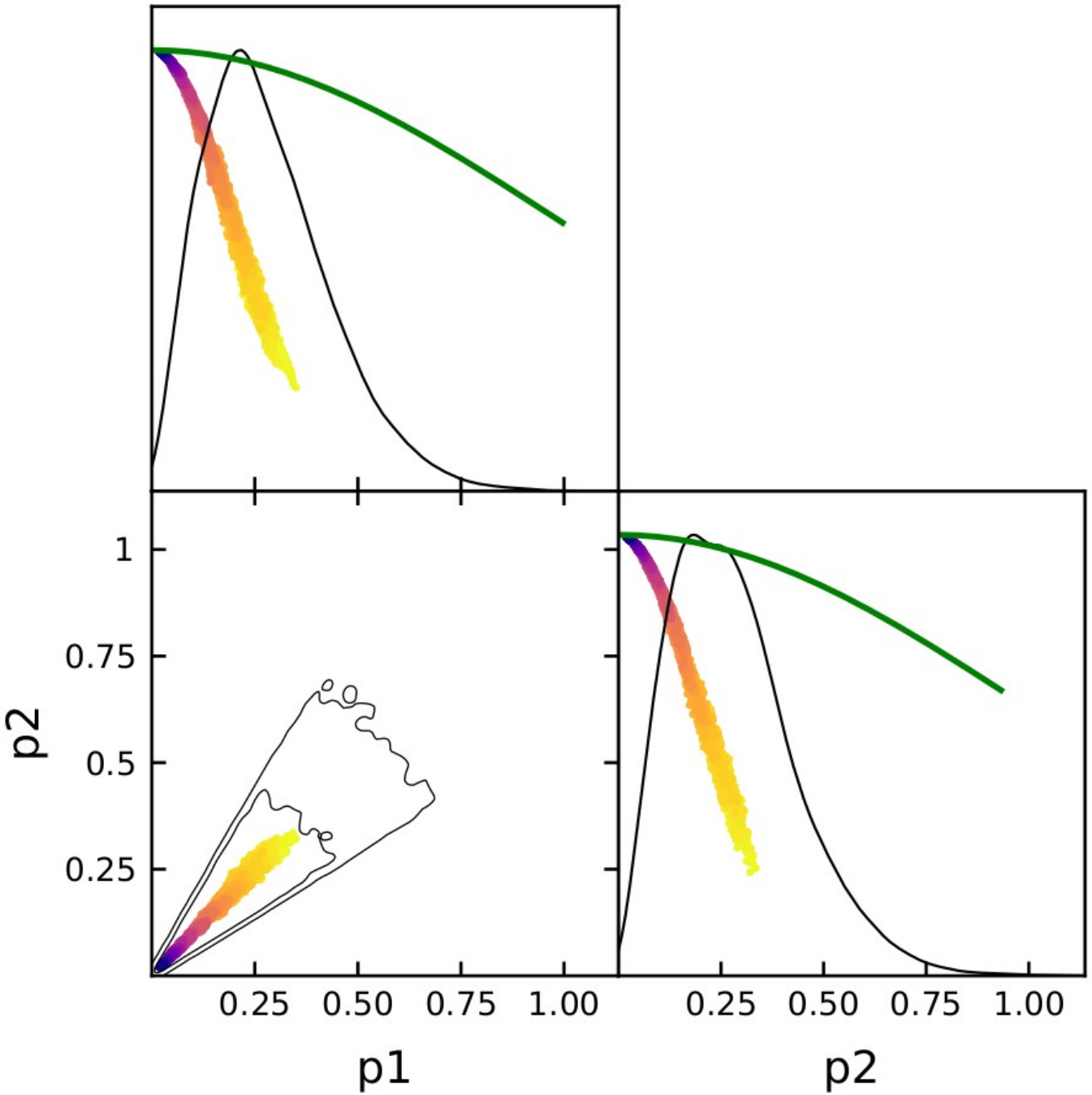
Identifying prior effects with ECLAIR



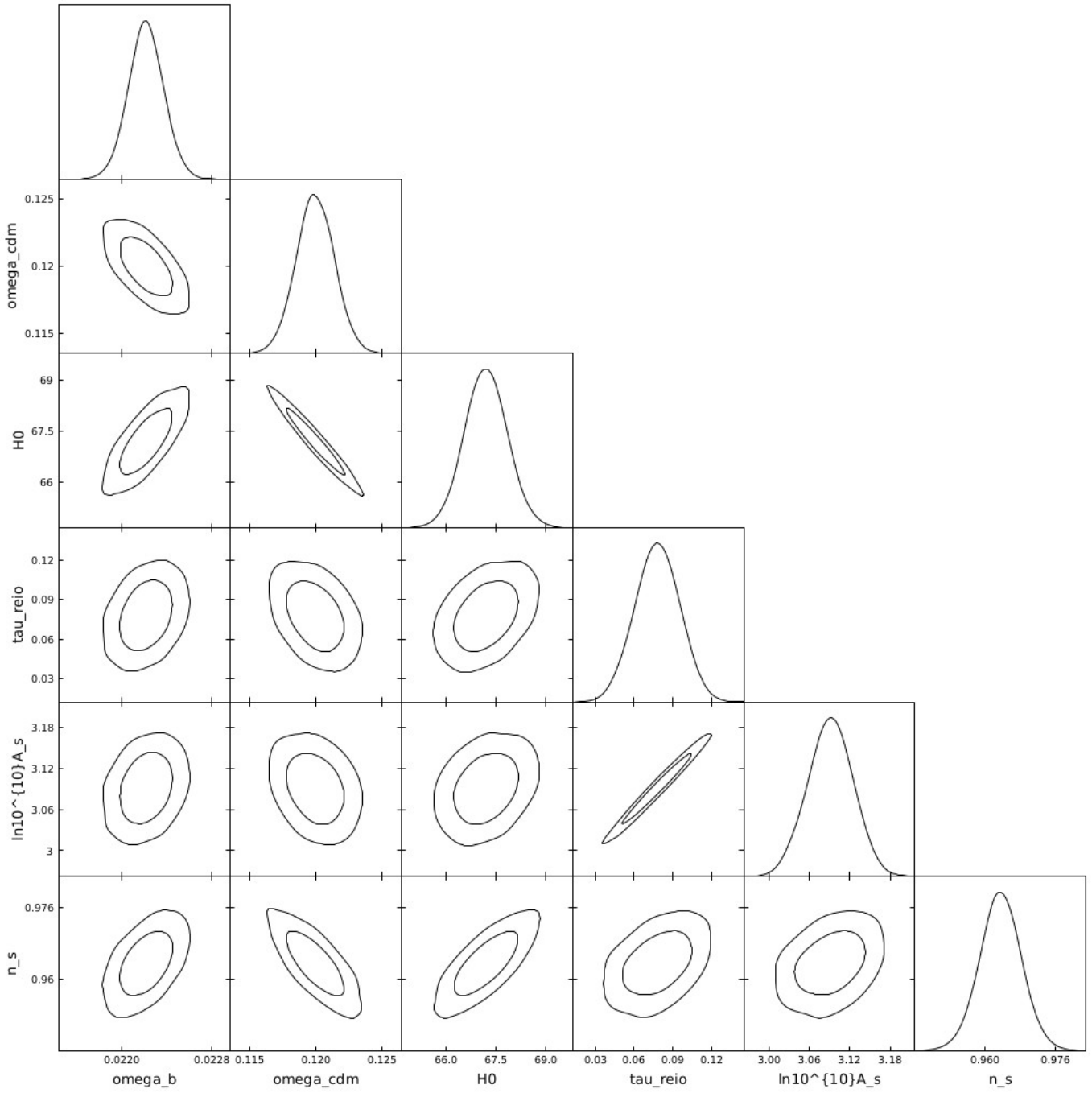
Identifying prior effects with ECLAIR



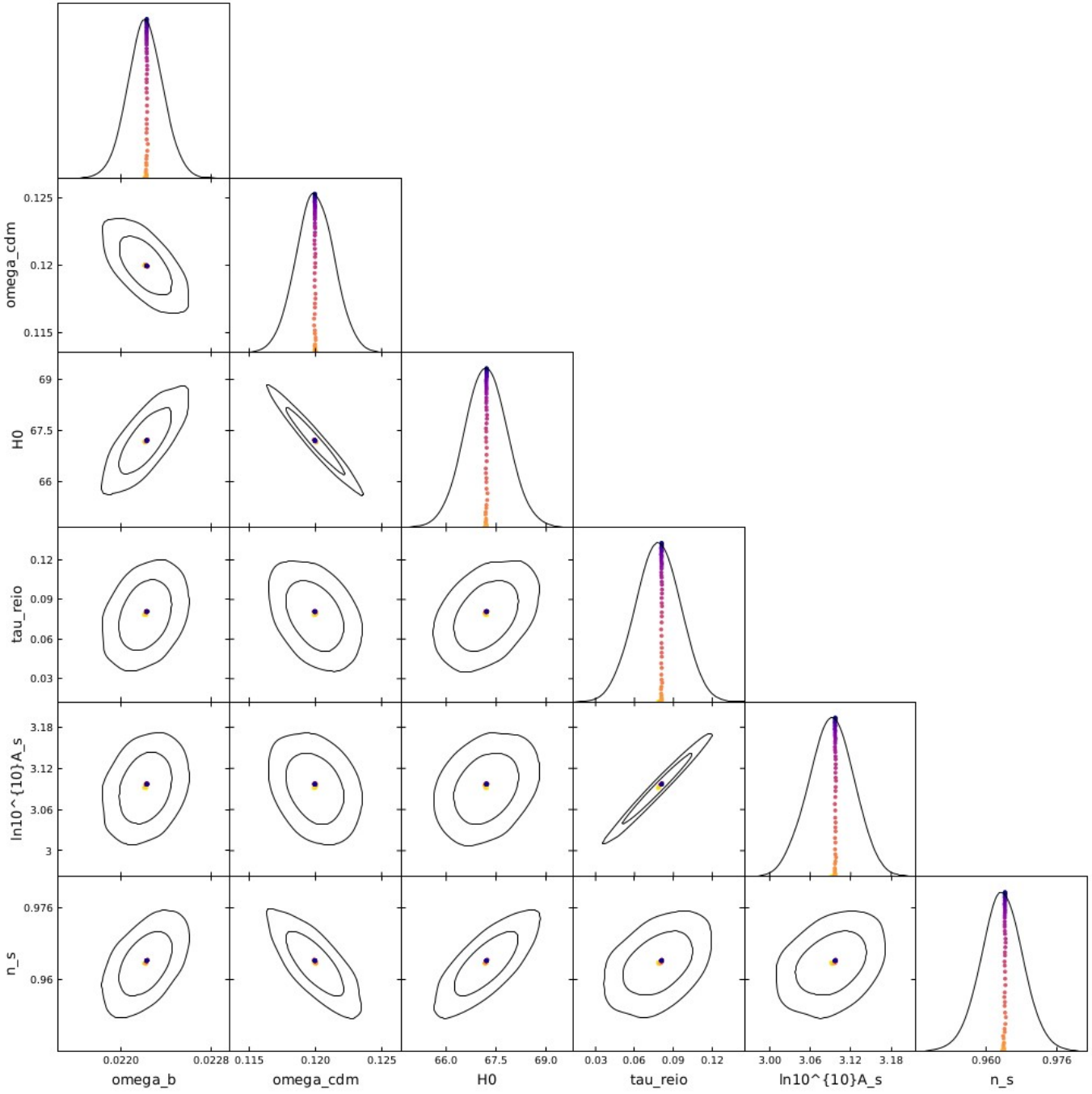
Identifying prior effects with ECLAIR



Identifying prior effects with ECLAIR



Identifying prior effects with ECLAIR



Thank you
for your attention !