JAM : cosmological inference made simple

Stéphane Ilić

CEICO, Prague

IRAP, Toulouse

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013)/ERC Grant Agreement No. 617656 "Theories and Models of the Dark Sector: Dark Matter, Dark Energy and Gravity."

European Research Council
Established by the European Commission

Atelier Outils de l'action Dark Energy
@ IHP, 19/11/2019

Introduction

Several “CosmoBoxes” on the market :

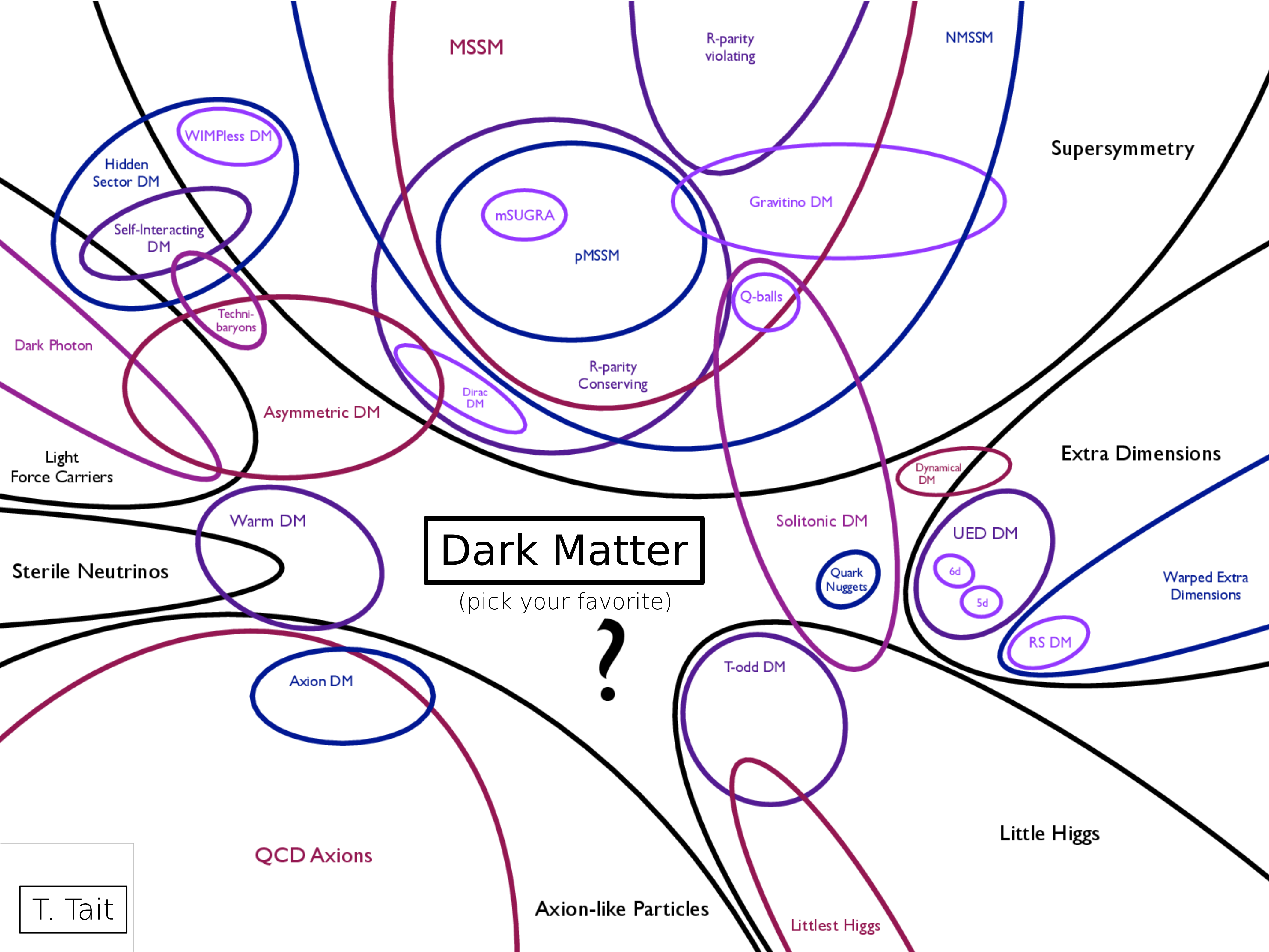
- CosmoMC
- MontePython
- CosmoSIS
- Cobaya
- ...

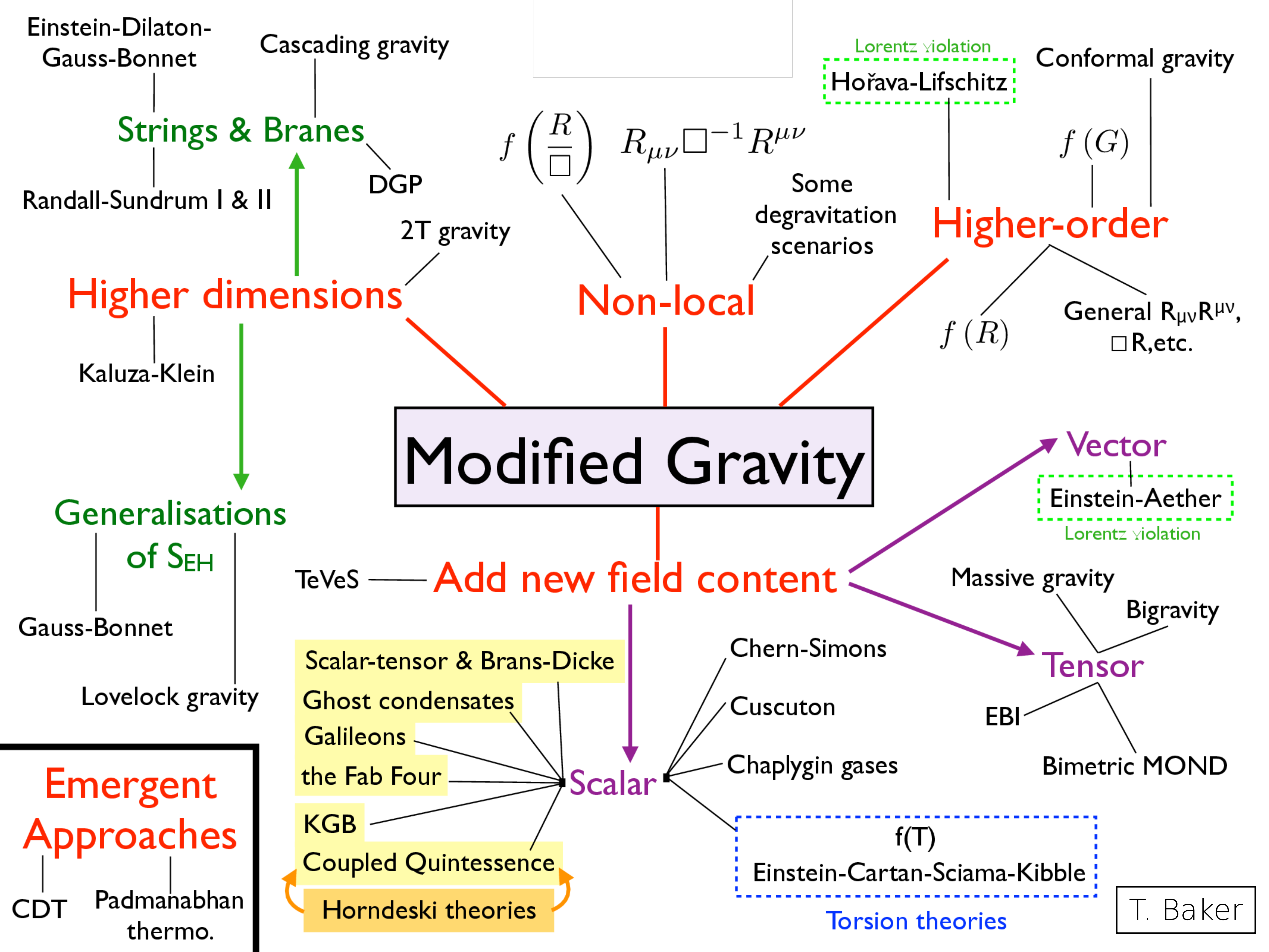
...but none entirely satisfying for my needs

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


MSSM
R-parity violating
NMSSM
Supersymmetry
WIMPless DM
Hidden Sector DM
Self-Interacting DM
Techni-baryons
mSUGRA
pMSSM
Gravitino DM
Q-balls
R-parity Conserving
Dirac DM
Dark Photon
Asymmetric DM
Light Force Carriers
Extra Dimensions
Dynamical DM
Solitonic DM
UED DM
6d
5d
Warped Extra Dimensions
RS DM
Quark Nuggets
Warm DM
Sterile Neutrinos
Dark Matter
(pick your favorite)
?
Axion DM
T-odd DM
Little Higgs
QCD Axions
Axion-like Particles
Littlest Higgs
T. Tait


Einstein-Dilaton-Gauss-Bonnet
Cascading gravity
Strings & Branes
DGP
Randall-Sundrum I & II
2T gravity
Higher dimensions
Kaluza-Klein
Generalisations of S_{EH}
Gauss-Bonnet
Lovelock gravity
$f\left(\frac{R}{\Box}\right)$
$R_{\mu\nu}\Box^{-1}R^{\mu\nu}$
Some degravitation scenarios
Non-local
Lorentz violation
Hořava-Lifschitz
Conformal gravity
$f(G)$
Higher-order
$f(R)$
General $R_{\mu\nu}R^{\mu\nu}$, $\Box R$, etc.
Modified Gravity
Vector
Einstein-Aether
Lorentz violation
TeVeS
Add new field content
Massive gravity
Bigravity
Tensor
EBI
Bimetric MOND
Scalar-tensor & Brans-Dicke
Ghost condensates
Galileons
the Fab Four
KGB
Coupled Quintessence
Horndeski theories
Scalar
Chern-Simons
Cuscuton
Chaplygin gases
f(T)
Einstein-Cartan-Sciama-Kibble
Torsion theories
Emergent Approaches
CDT
Padmanabhan thermo.
T. Baker

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

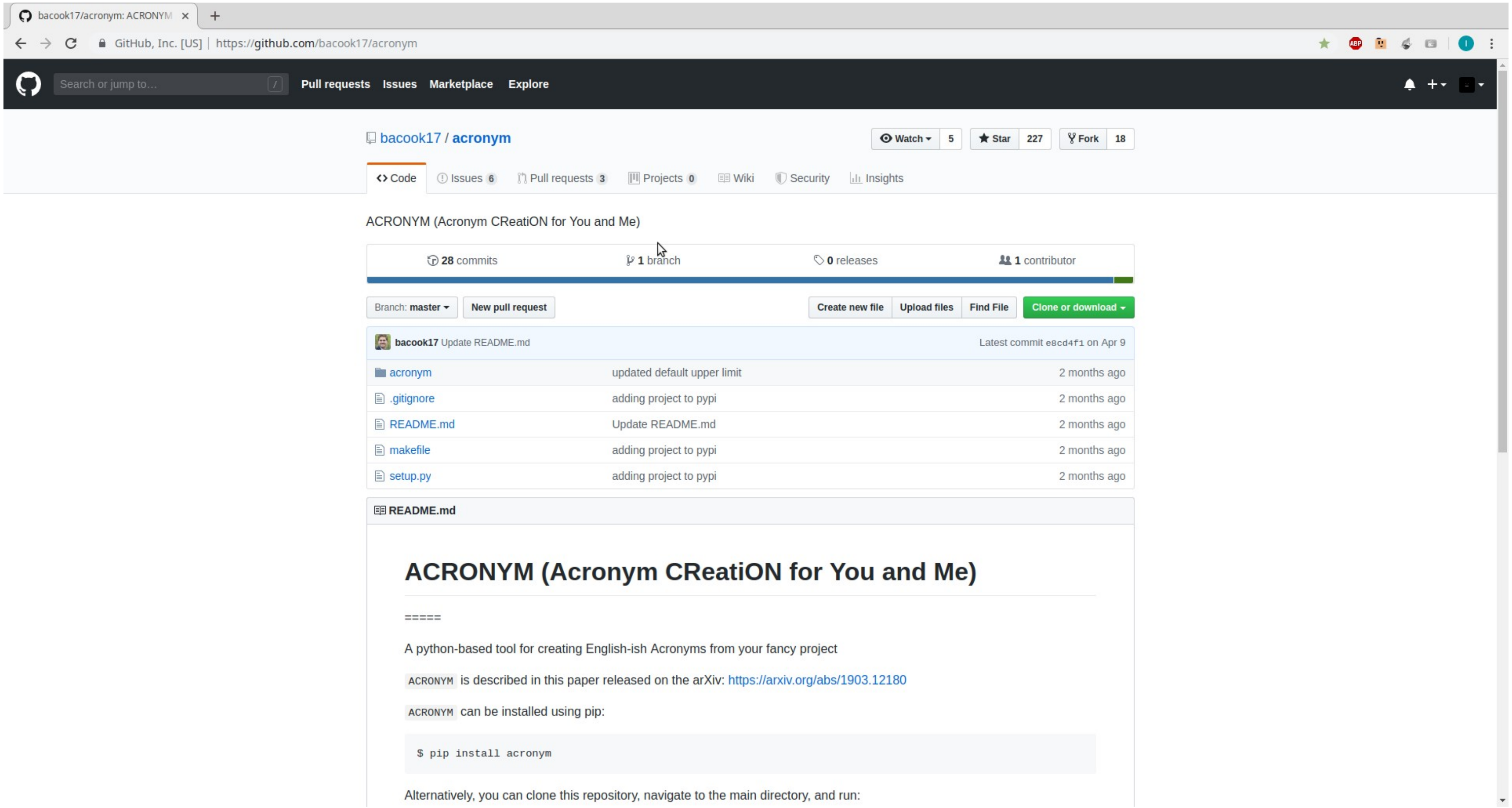
Introducing : NAME/ACRONYM PENDING

Introducing : NAME/ACRONYM PENDING

JAM ? (= Just A (simple) MCMC tool)

JUSTICE ? (= JUst a Simple Toolbox for InferenCE)

...



Introducing : JAM

- Two (fairly) short files in Python 2/3 : main (~200) & parser (~500)
- Human-readable/tweakable, well-commented (I hope !)

Introducing : JAM

```
############################
### Free MCMC parameters ###
############################
#----------------------------------------------------------------#
# Column order :                                                 #
# > type  name  start  min  max  width                           #
# Notes :                                                        #
# > "type" : "var_class" if a class parameter otherwise "var"    #
# > "width" : only used for initializing the walkers positions   #
#----------------------------------------------------------------#

### Class parameters
var_class  omega_b        0.02222   0.005   0.1   0.0001
var_class  omega_cdm      0.1197    0.1     0.13  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

### Calibration parameter common to all Planck 2015/18 likelihoods (incl. lensing)
var  A_planck  1.  0.9  1.1  0.002
```

Introducing : JAM

```
##########################
### Priors on parameters ###
##########################
#--------------------------------#
# Only Gaussian prior implemented #
# Column order :                  #
# > type  name  mean  stddev      #
#--------------------------------#

### Prior on calibration parameter common to all Planck 2015/18 likelihoods
gauss_prior  A_planck  1.  0.0025
```

Introducing : JAM

```
#######################
### Fixed parameters ###
#######################
#-------------------------------------------------------------#
# Column order :                                              #
# > type  name  value                                         #
# Notes :                                                     #
# > "type" : "fix_class" if a class parameter otherwise "fix" #
#-------------------------------------------------------------#

### Class parameters
#---------------------------------------#
# Note :                                #
# > "non_linear" instead of "non linear" #
#---------------------------------------#
fix_class  output         tCl pCl lCl mPk
fix_class  lensing        yes
fix_class  l_max_scalars  2508
fix_class  T_cmb          2.7255
fix_class  non_linear     halofit
fix_class  P_k_max_h/Mpc  1.
fix_class  N_ur           2.0328
fix_class  N_ncdm         1
fix_class  m_ncdm         0.06

### Planck 2015 full TT fixed nuisance parameters
### Additional fixed parameters for Planck 2015 full TTTEEE
fix  cib_index           -1.3
fix  galf_EE_index       -2.4
```

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 : JAM

- 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 : JAM

```
#############
### CLASS ###
#############

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

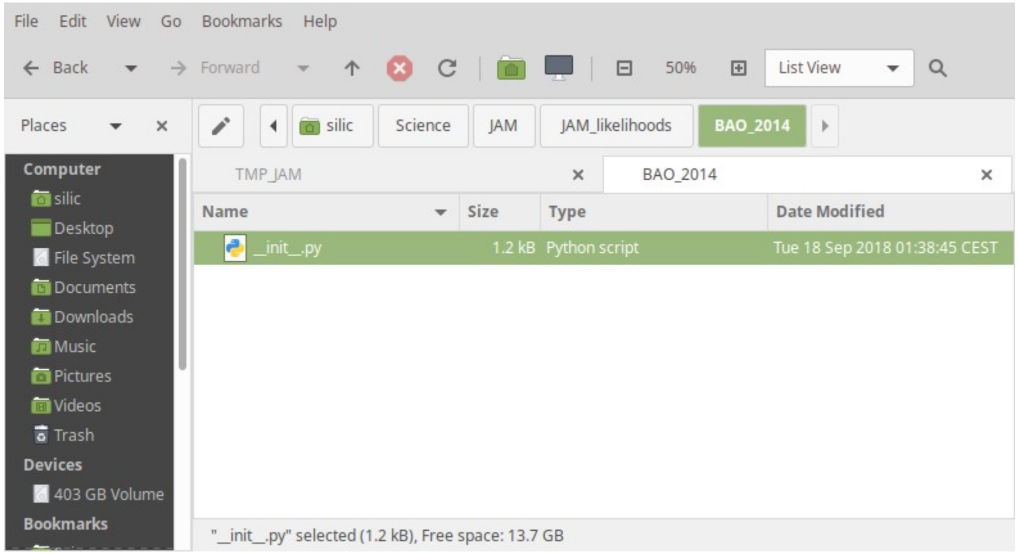
Introducing : JAM

- 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 : JAM

```
##################
### Likelihoods ###
##################

#---------------------------------------#
# Select the likelihoods to be combined #
#---------------------------------------#
# Current choices :                     #
# > BAO_2014                            #
# > H0_HST                              #
# > SN_JLA                              #
# > Planck2015_highTT                   #
# > Planck2015_highTTlite               #
# > Planck2015_highTTTEEE               #
# > Planck2015_highTTTEEElite           #
# > Planck2015_lensT                    #
# > Planck2015_lensTP                   #
# > Planck2015_lowTEB                   #
# > Planck2015_lowTT                    #
# > Planck2018_highTT                   #
# > Planck2018_highTTlite               #
# > Planck2018_highTTTEEE               #
# > Planck2018_highTTTEEElite           #
# > Planck2018_lensCMBdep               #
# > Planck2018_lensCMBmarg              #
# > Planck2018_lowBB                    #
# > Planck2018_lowEE                    #
# > Planck2018_lowEEBB                  #
# > Planck2018_lowTT                    #
#---------------------------------------#
likelihood  Planck2015_lowTEB
likelihood  Planck2015_TTTEEE
```

Introducing : JAM




```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”:

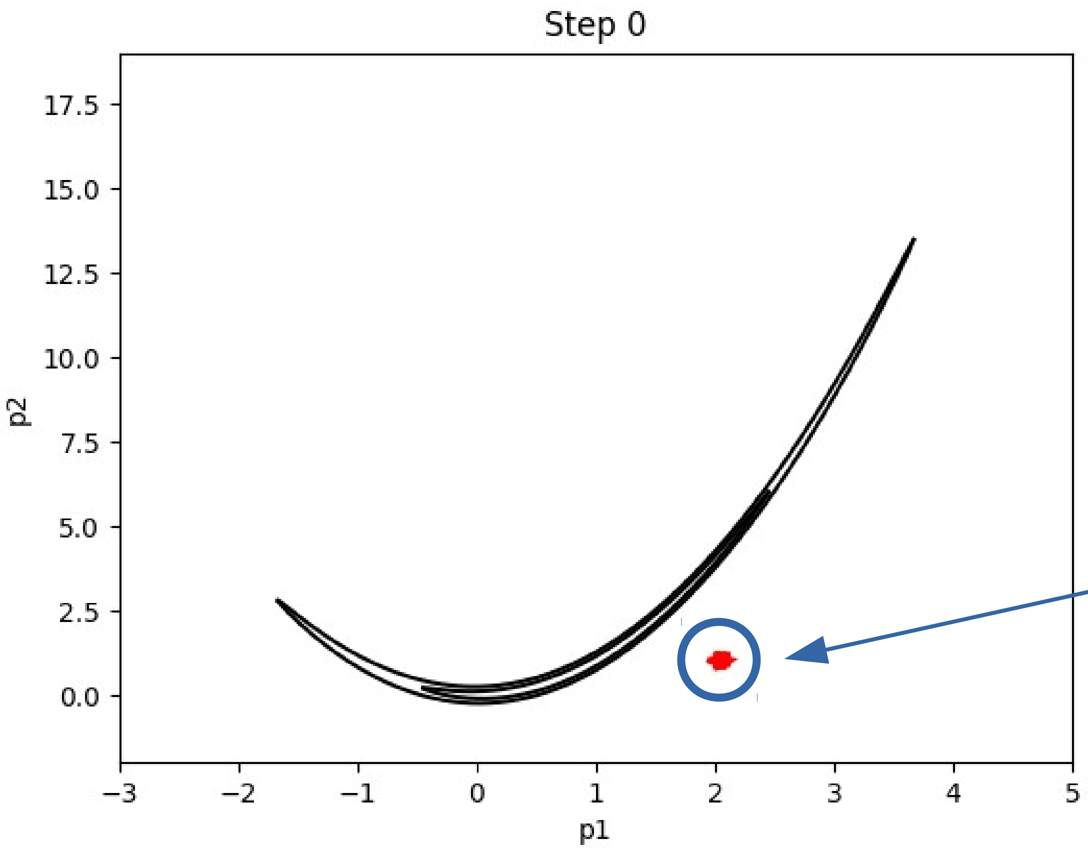
- Juggling with many cosmological models (and as many Boltzmann solvers)
- Non trivial exploration of parameter space (priors, constraints...)
- A (relatively) big cluster to exploit

Introducing : JAM

- 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

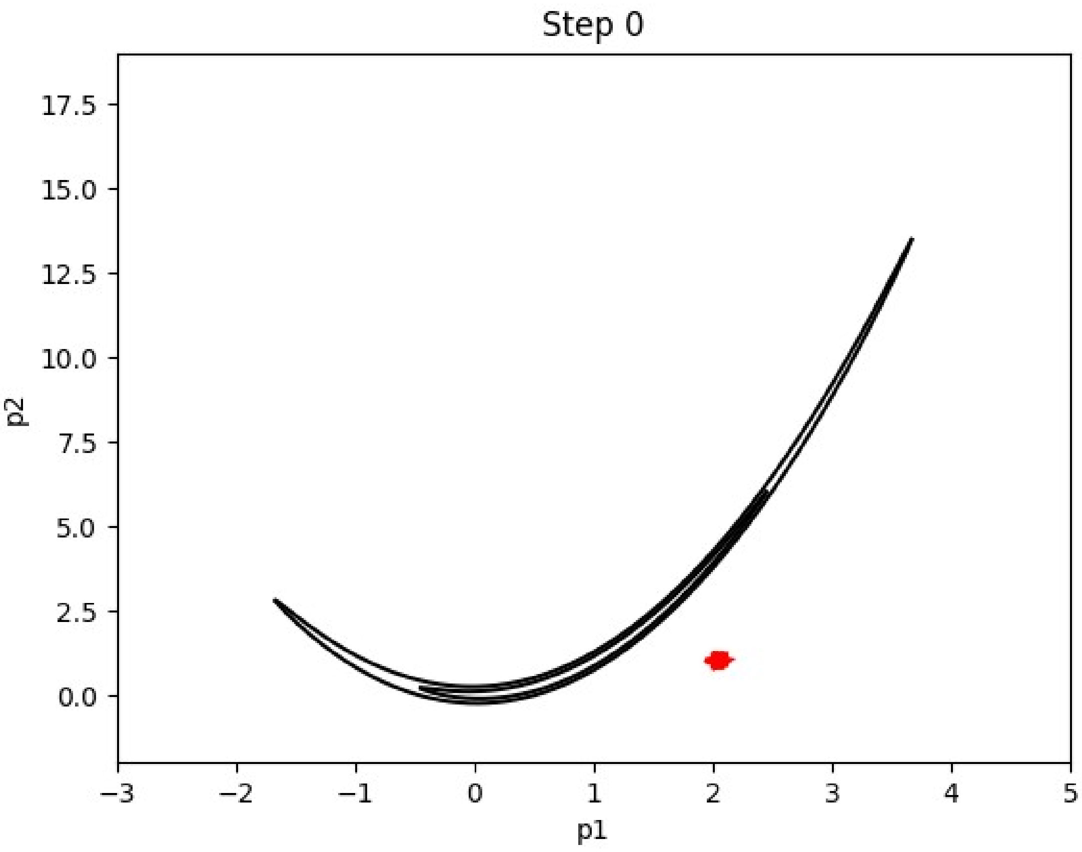
$$\pi(x) \propto \exp\left(\frac{-(x_1 - x_2)^2}{2\epsilon} - \frac{(x_1 + x_2)^2}{2}\right)$$



Collection of "walkers" initialized at random positions

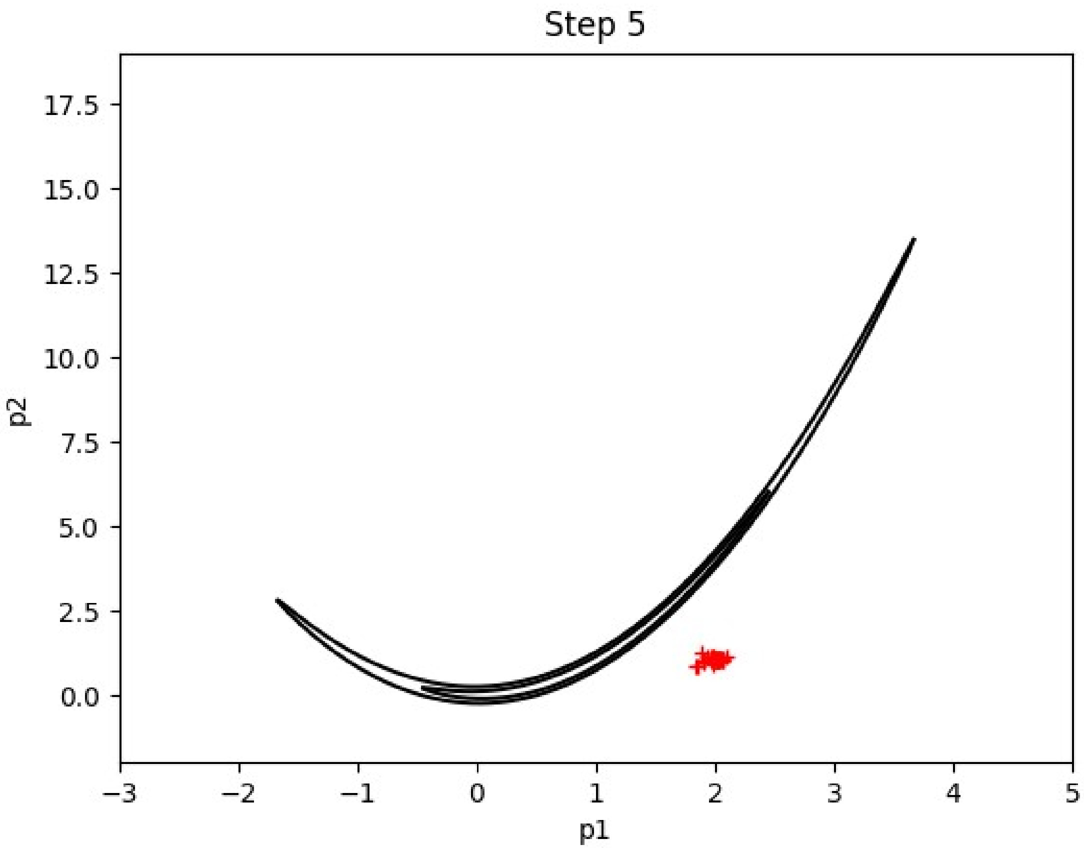
Ensemble sampling

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



Ensemble sampling

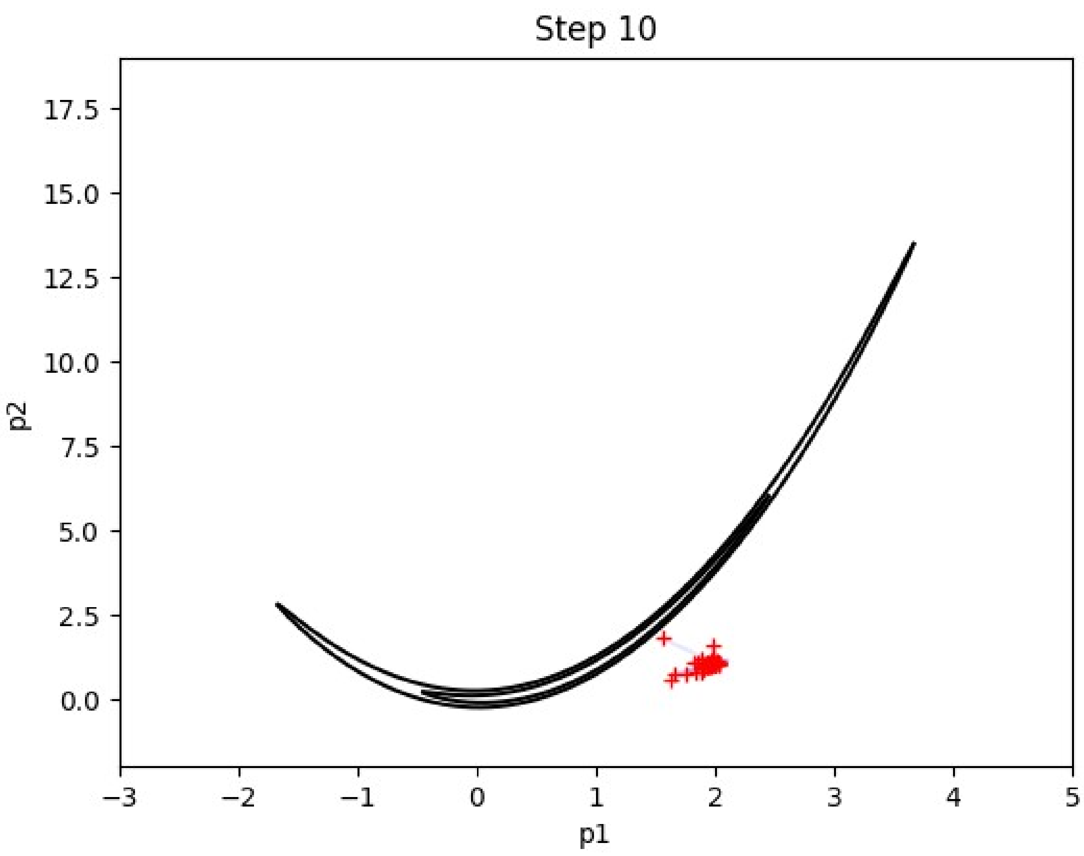
$$\pi(x) \propto \exp\left(\frac{-(x_1 - x_2)^2}{2\epsilon} - \frac{(x_1 + x_2)^2}{2}\right)$$



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

Ensemble sampling

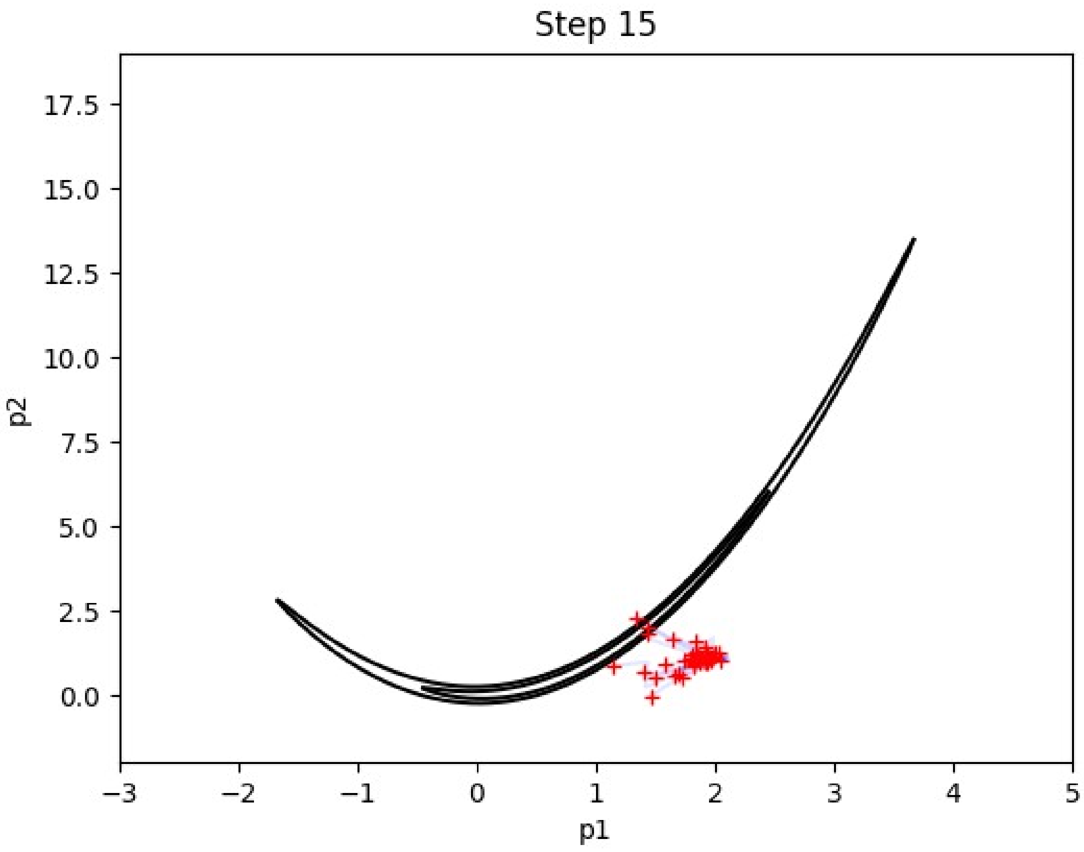
$$\pi(x) \propto \exp\left(\frac{-(x_1 - x_2)^2}{2\epsilon} - \frac{(x_1 + x_2)^2}{2}\right)$$



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

Ensemble sampling

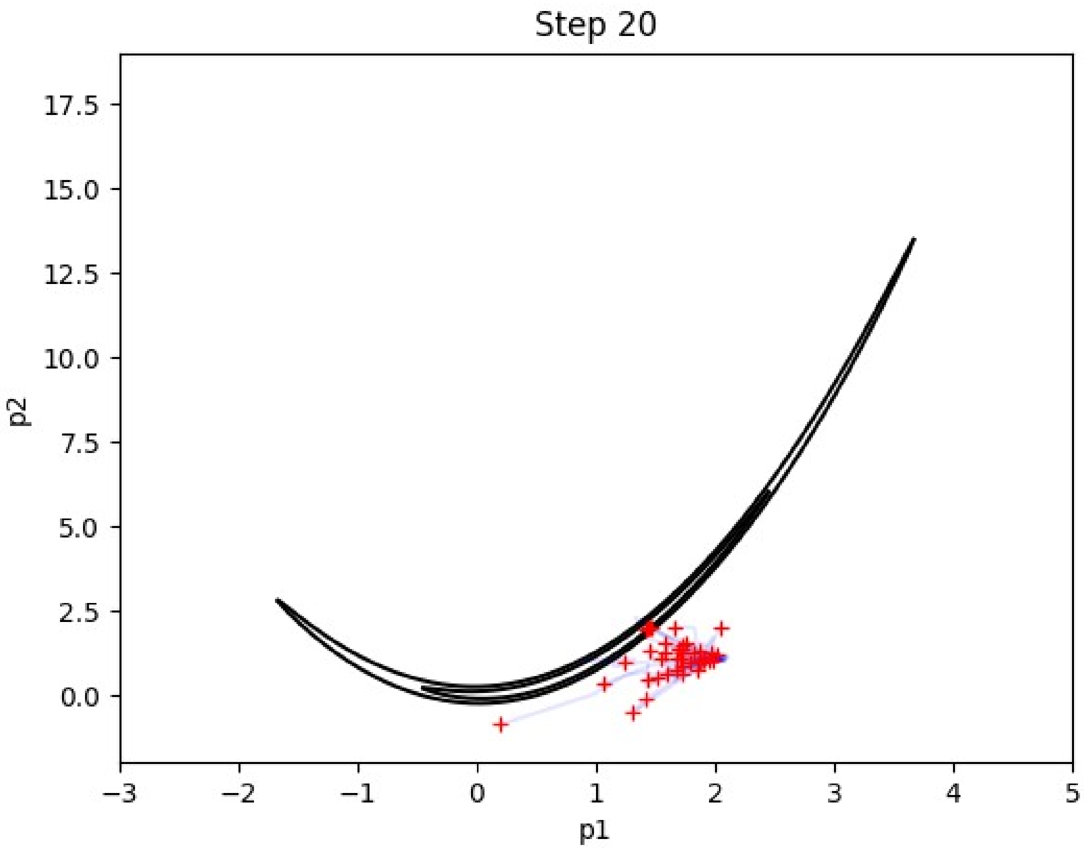
$$\pi(x) \propto \exp\left(\frac{-(x_1 - x_2)^2}{2\epsilon} - \frac{(x_1 + x_2)^2}{2}\right)$$



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

Ensemble sampling

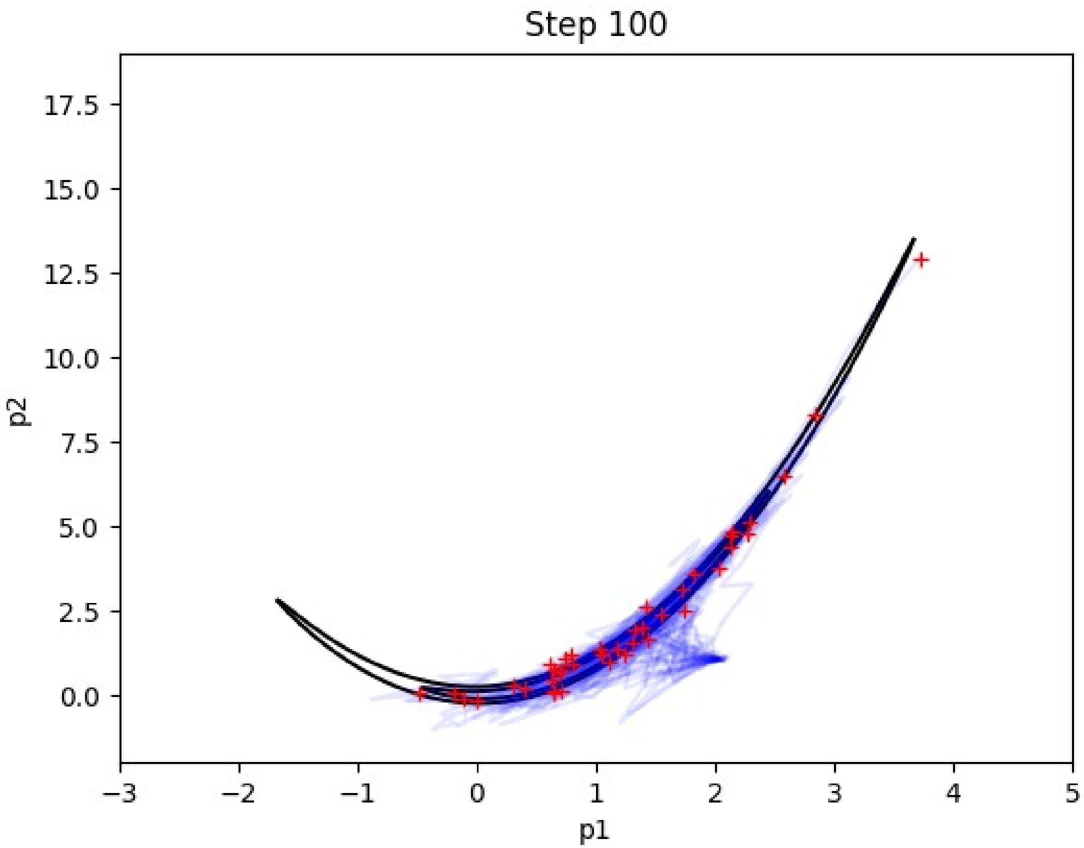
$$\pi(x) \propto \exp\left(\frac{-(x_1 - x_2)^2}{2\epsilon} - \frac{(x_1 + x_2)^2}{2}\right)$$



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

Ensemble sampling

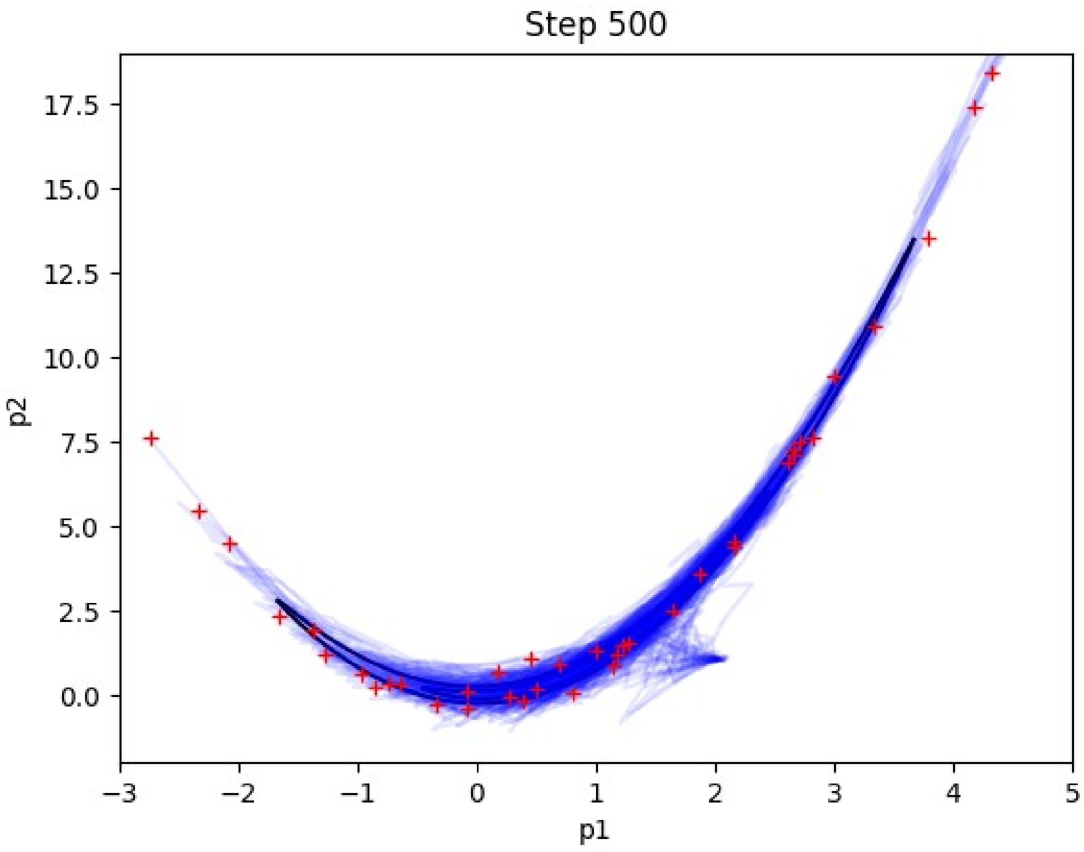
$$\pi(x) \propto \exp\left(\frac{-(x_1 - x_2)^2}{2\epsilon} - \frac{(x_1 + x_2)^2}{2}\right)$$



...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

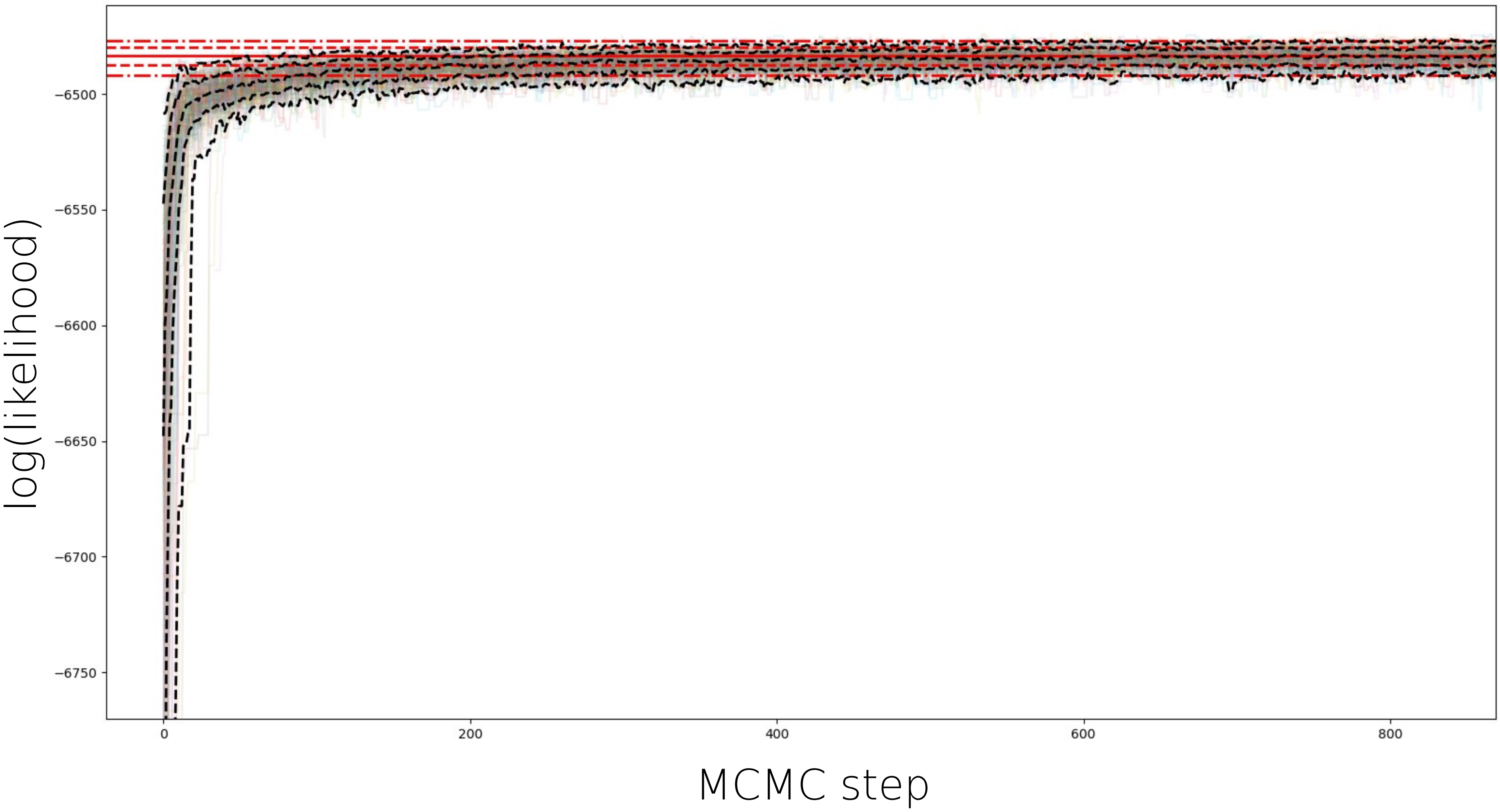
Introducing : JAM

```
############
### MCMC ###
############

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

#-----------------------------------------------------------------------------#
# Number of walkers (has to be at least 2 times the number of free parameters) #
#-----------------------------------------------------------------------------#
# Current choices :                                                            #
# > "custom  X"  => fixed to X                                                 #
# > "prop_to  X" => X times the number of free parameters                      #
#-----------------------------------------------------------------------------#
#n_walkers  custom  1000
n_walkers  prop_to  4

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

#-------------------------------------------------#
# Thinning factor (i.e. keep only every X step) #
#-------------------------------------------------#
thin_by  1

#-------------------------#
# Temperature of the MCMC #
#-------------------------#
temperature  1.

#------------------------------------------------------------------------#
# Parameter for the "stretch move" of the Ensemble sampler (default is 2) #
#------------------------------------------------------------------------#
stretch  2.
```

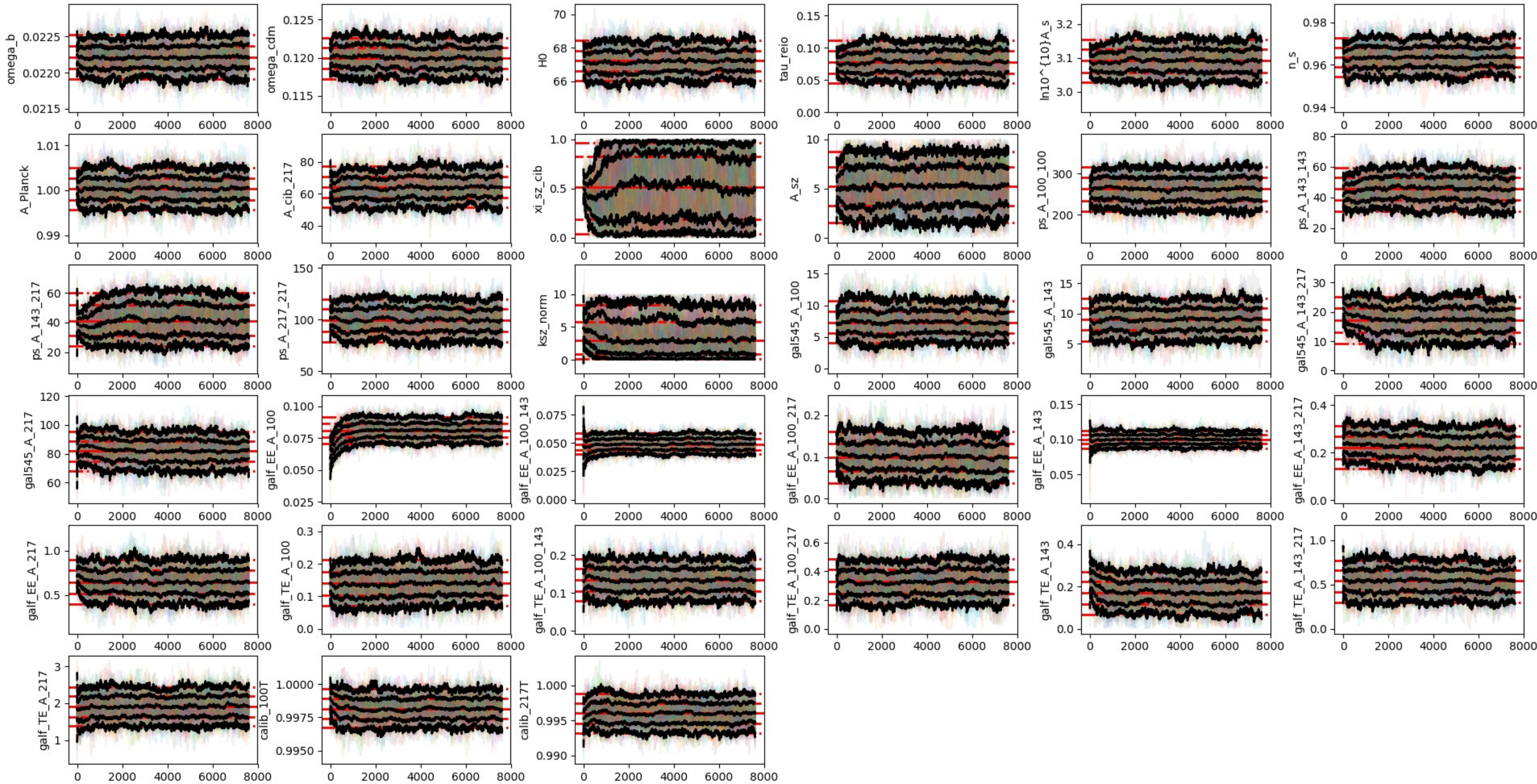
Introducing : JAM

- 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

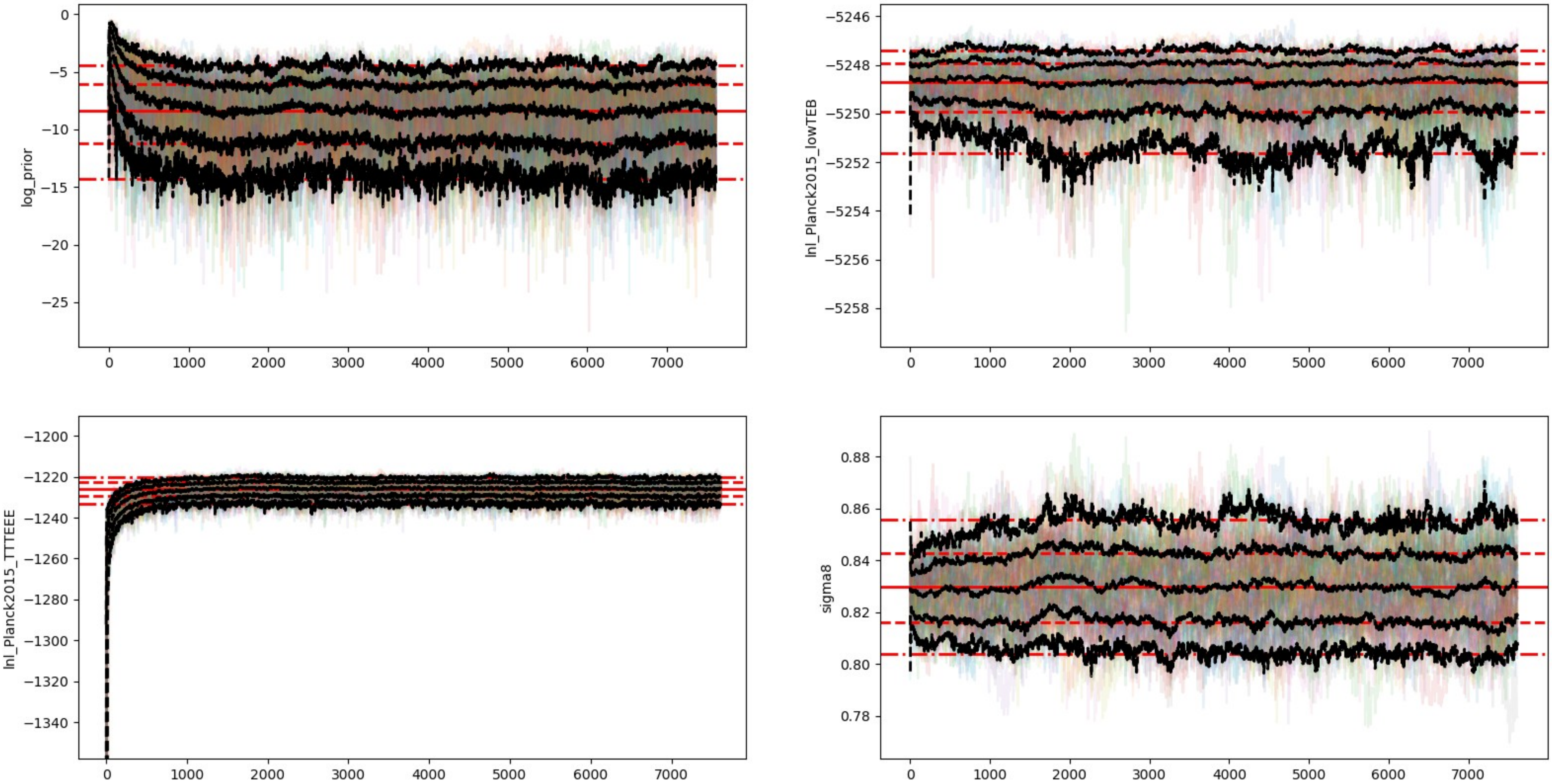
Visualization tools



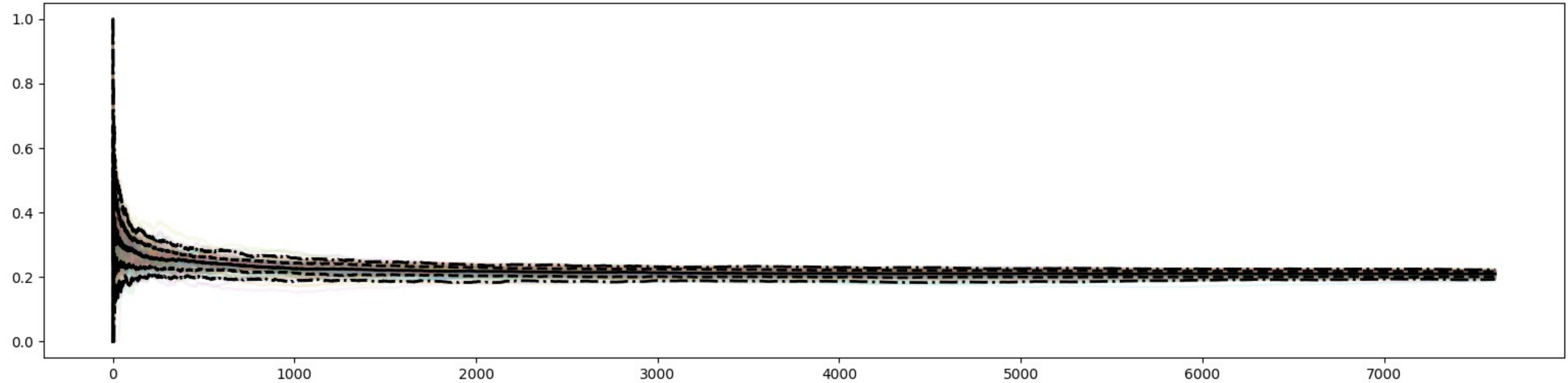
Visualization tools

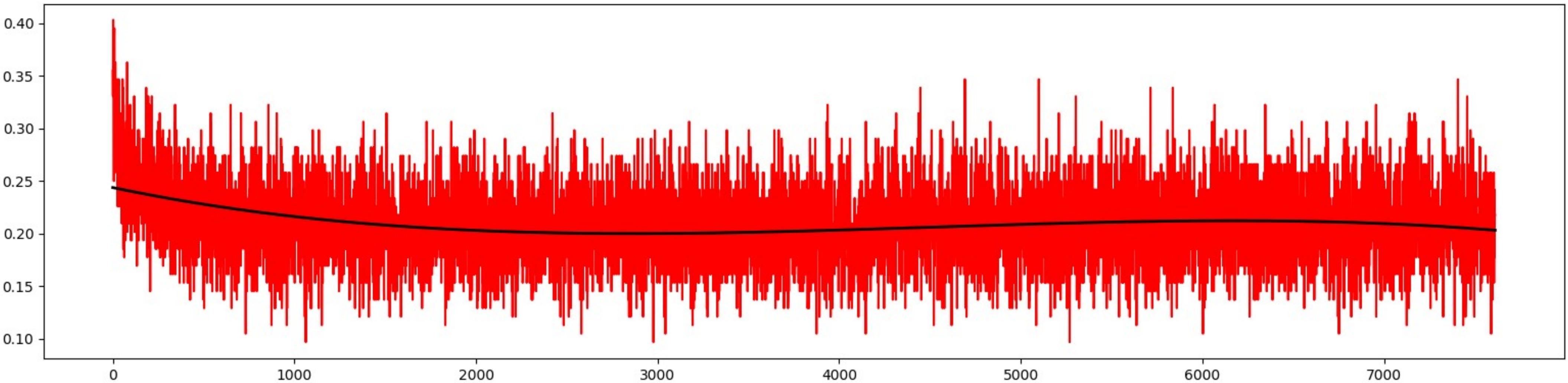


Visualization tools



Visualization tools

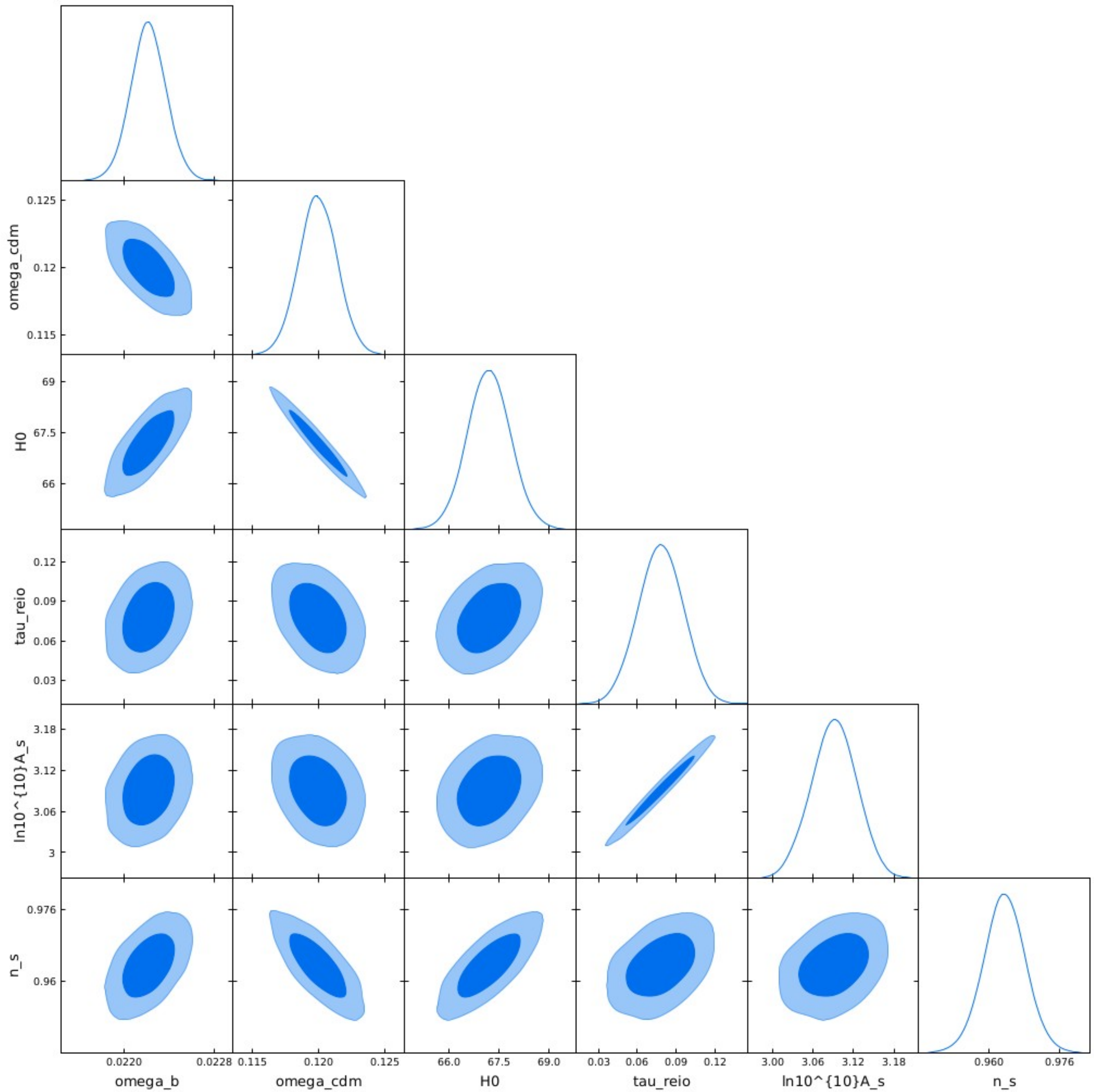




Introducing : JAM

- 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)
- <u>Non trivial exploration of parameter space (priors, constraints…)</u>
- A (relatively) big cluster to exploit

Introducing : JAM

- 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

JAM parsing features

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

#------------------------------#
# Put constraints on parameters #
#------------------------------#----------------------------------#
# Syntax :                                                        #
# > constraint  XXX  =  YYY                                       #
# > where XXX is the parameter forced to be equal to YYY          #
# Notes :                                                         #
# > YYY can be any fonction of any parameter                      #
# > in XXX and YYY, use syntax class[par_name] if class parameter #
# > in XXX and YYY, use syntax likes[par_name] otherwise          #
# Examples :                                                      #
# > class[omega_b]  =  class[omega_cdm]                           #
#-----------------------------------------------------------------#
#constraint  class[par_1] = class[par_2]+class[par_3]

#-------------------------------------------#
# Request some derived parameters in output #
#-------------------------------------------#----------------------------#
# Syntax :                                                               #
# > deriv  name  quantity_requested                                      #
# Notes :                                                                #
# > "name" == name of derived parameter in chain (should contain no space) #
# > "quantity_requested" can be any command one wants                    #
# > class wrapper accessible via "class_run" instance                    #
# > class background quantities accessible via "bg" dictionnary          #
# > class parameters accessible via "class_input" dictionnary            #
# > nuisance parameters accessible via "likes_input" dictionnary         #
# Examples :                                                             #
# > for H0 :      deriv  H0      bg['H [1/Mpc]'][-1]*299792.458          #
# > for sigma_8 :  deriv  sigma8  class_run.sigma8()                     #
# > for sum_nu  :  deriv  sum_nu  class_input['m_ncdm_val_0']+...        #
#------------------------------------------------------------------------#
# deriv  H0     bg['H [1/Mpc]'][-1]*299792.458
```

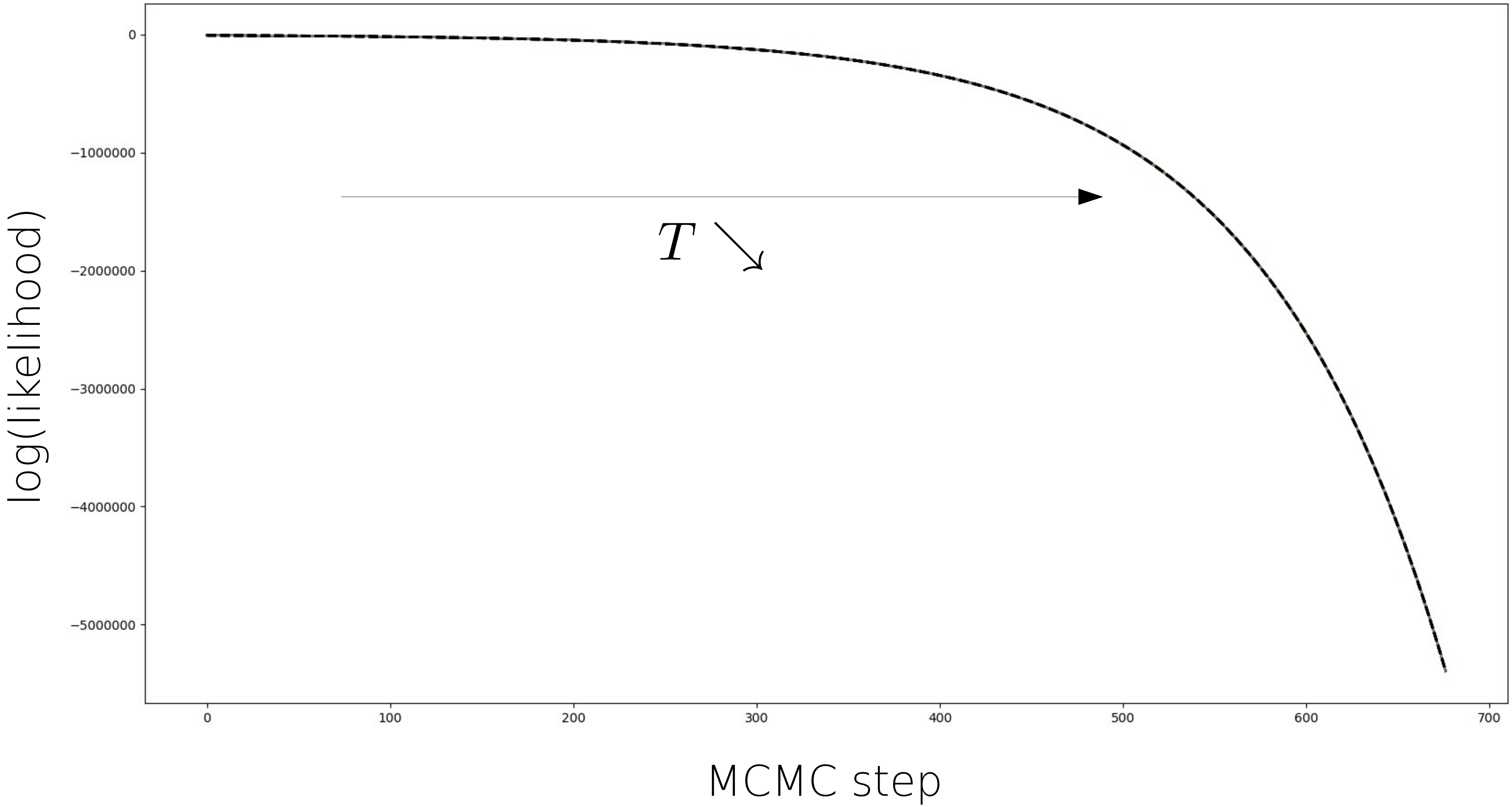
+ can put priors on any derived parameter

Introducing : JAM

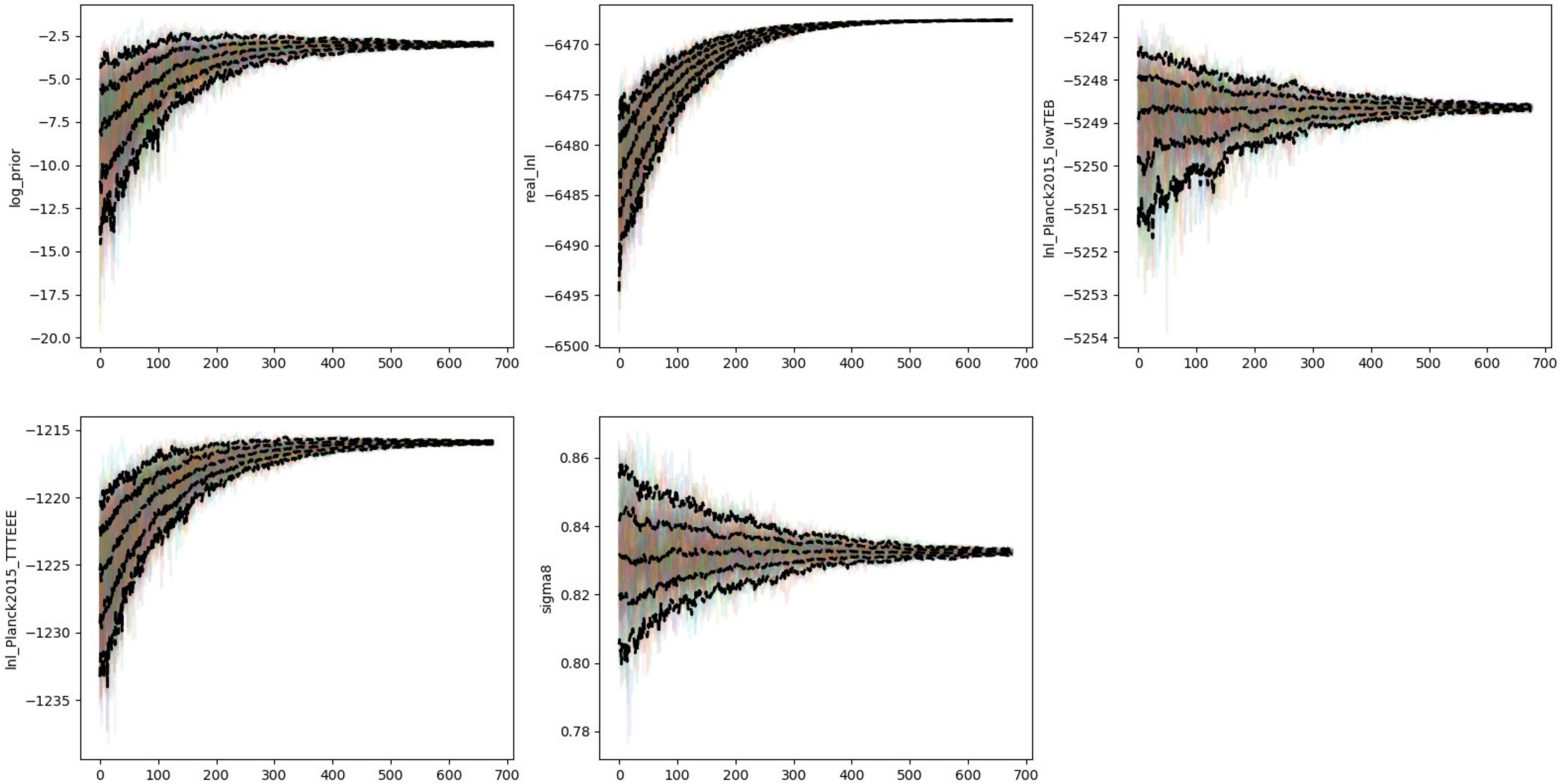
- Two (fairly) short files in Python 2/3 : main (~200) & parser (~500)
- Human-readable/tweakable, well-commented (I hope !)
- Working with any CLASS variant, no modification required
- Growing number of likelihoods/datasets implemented (easy to add new ones)
- MCMC algorithm : Affine-Invariant Ensemble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

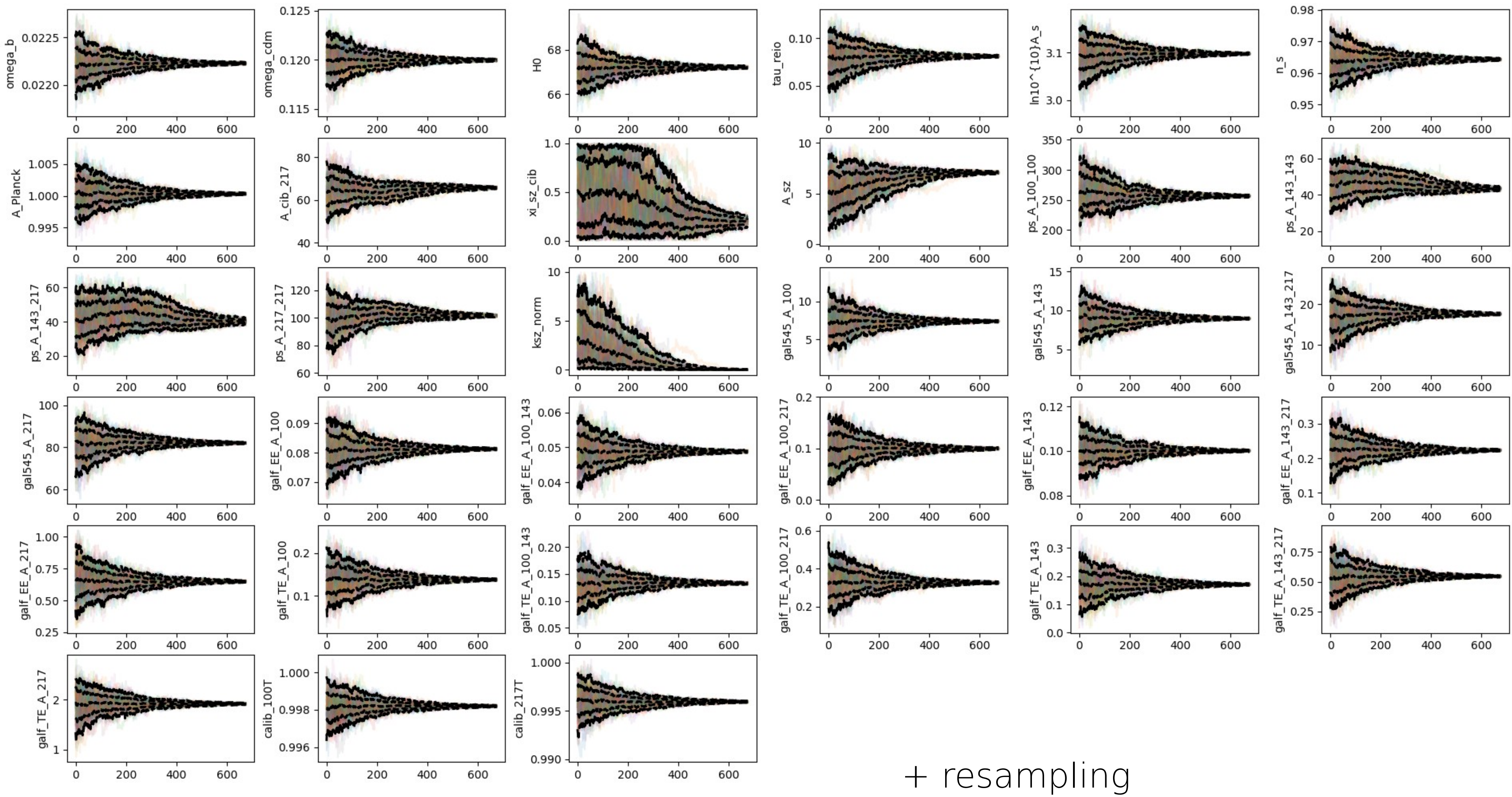
$$\mathcal{L} \longrightarrow \mathcal{L}^{1/T}$$



Minimizing with JAM



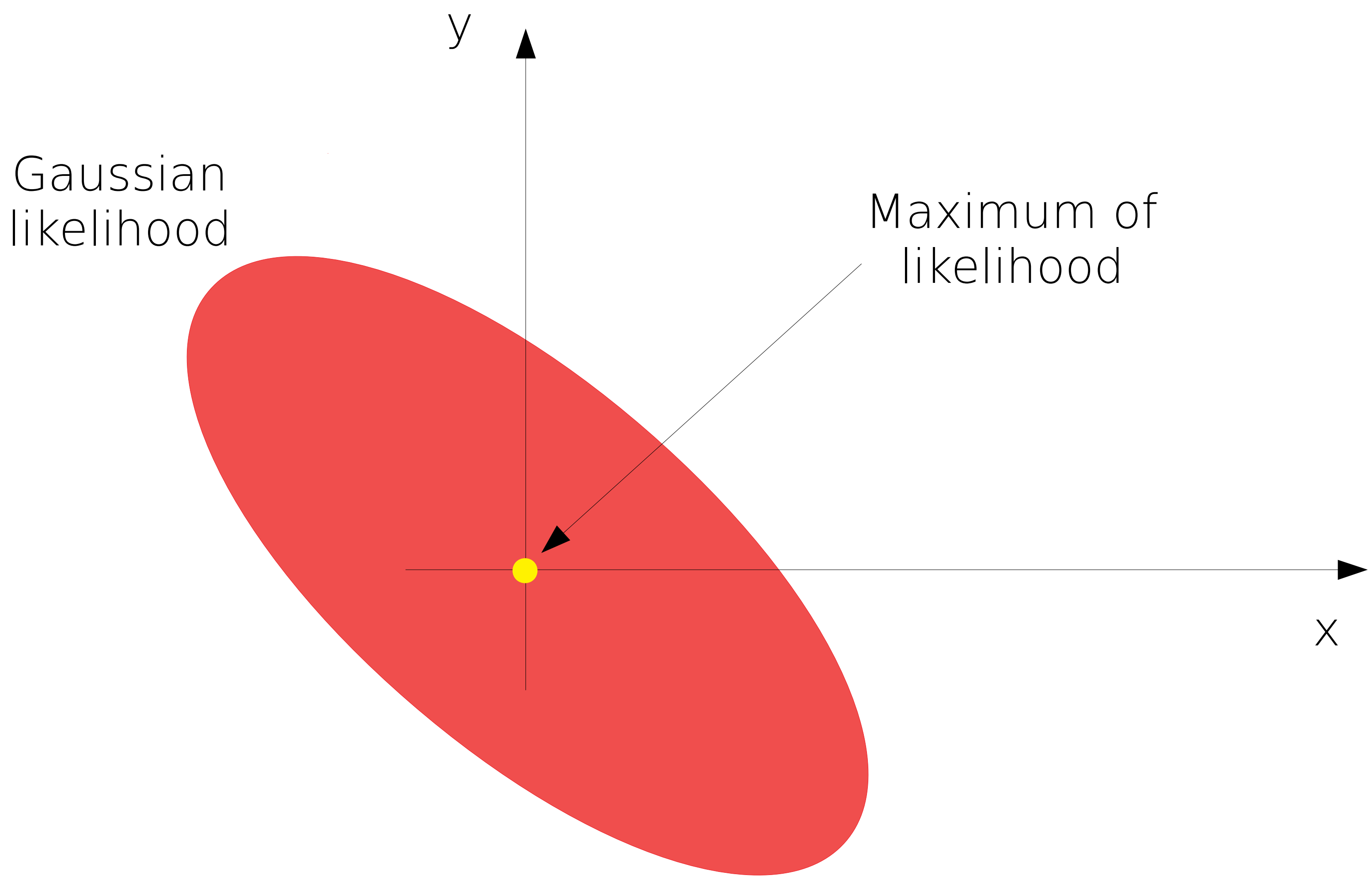
Minimizing with JAM



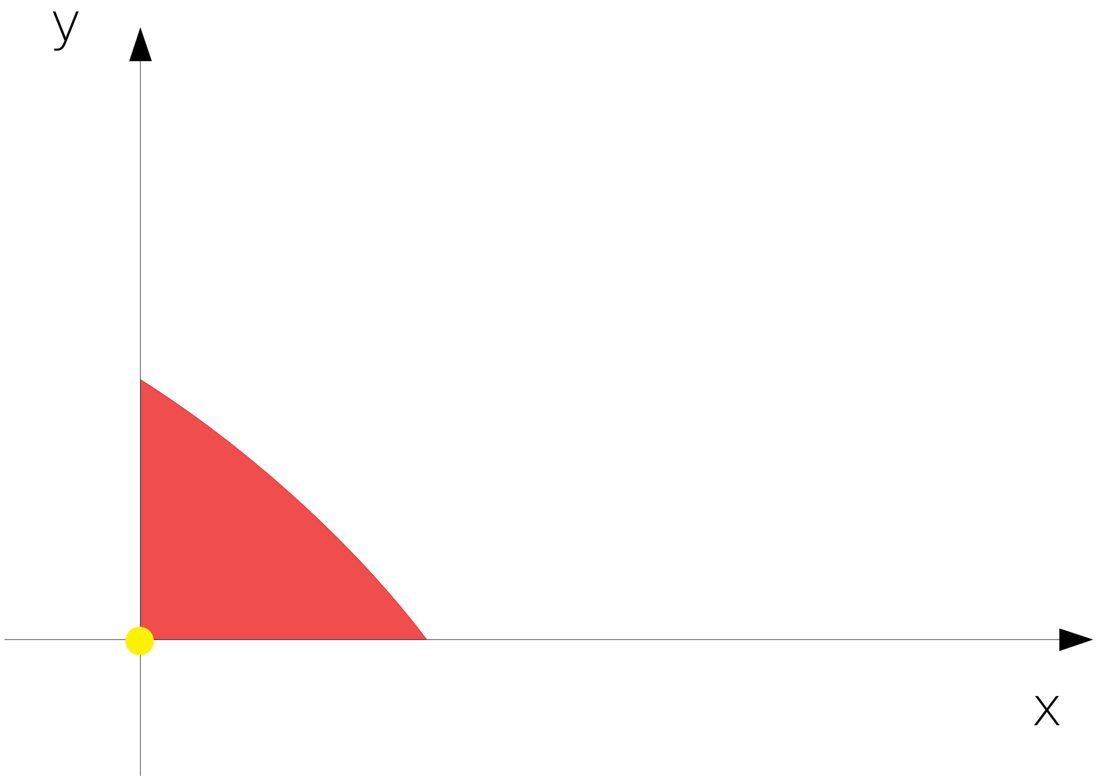
+ resampling
+ evidence computing

Identifying prior effects with JAM

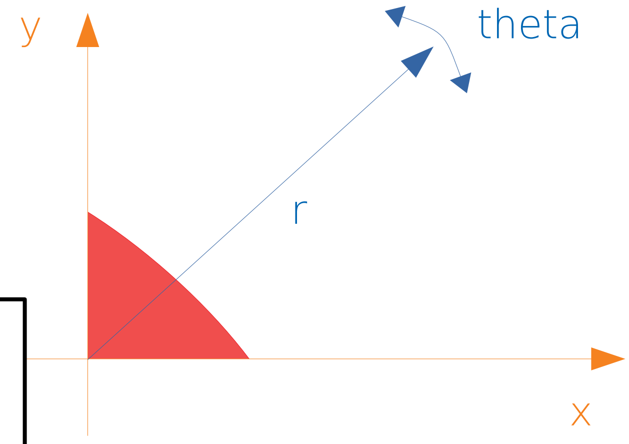
Effects of priors

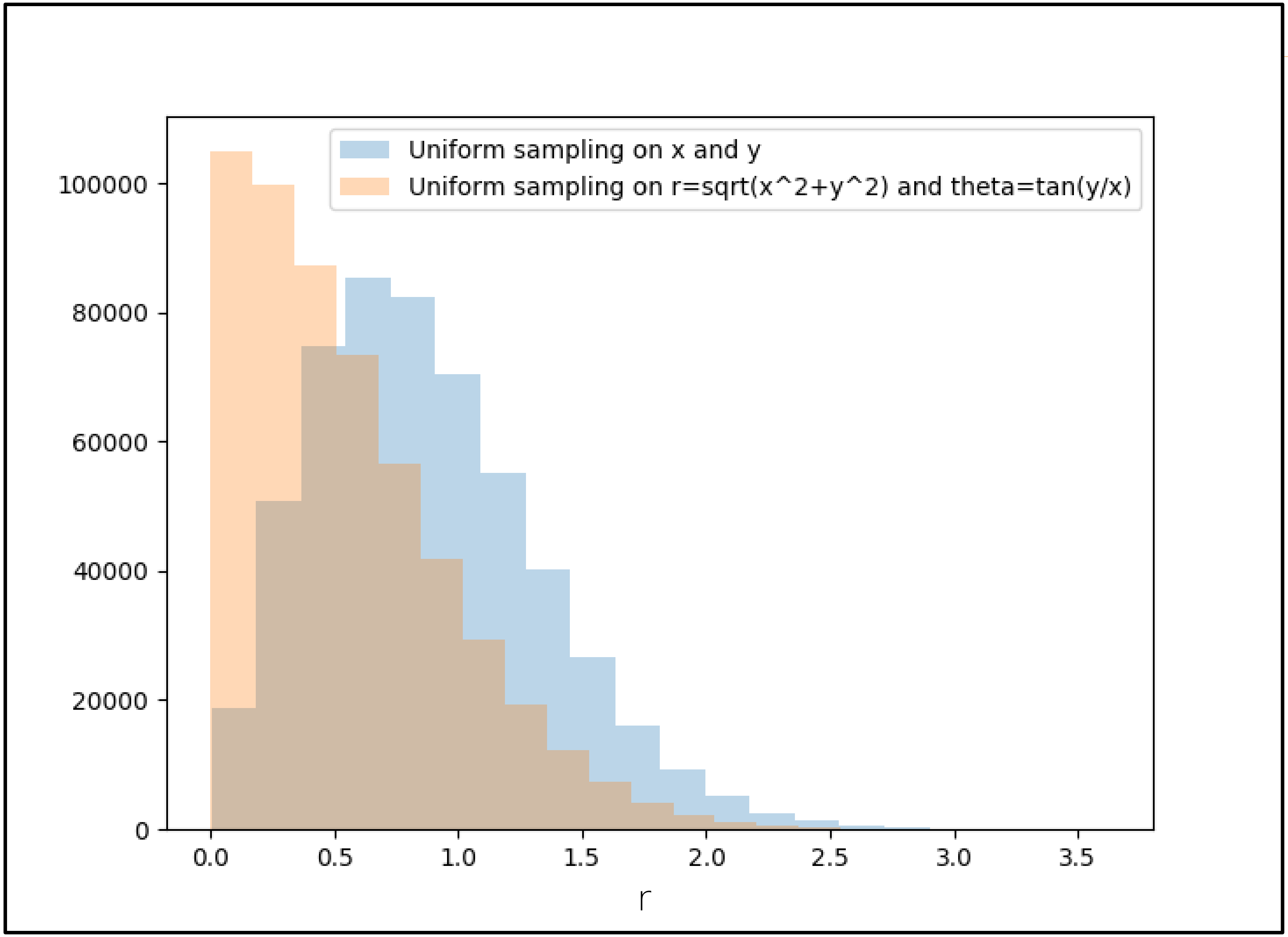


Effects of priors



Effects of priors



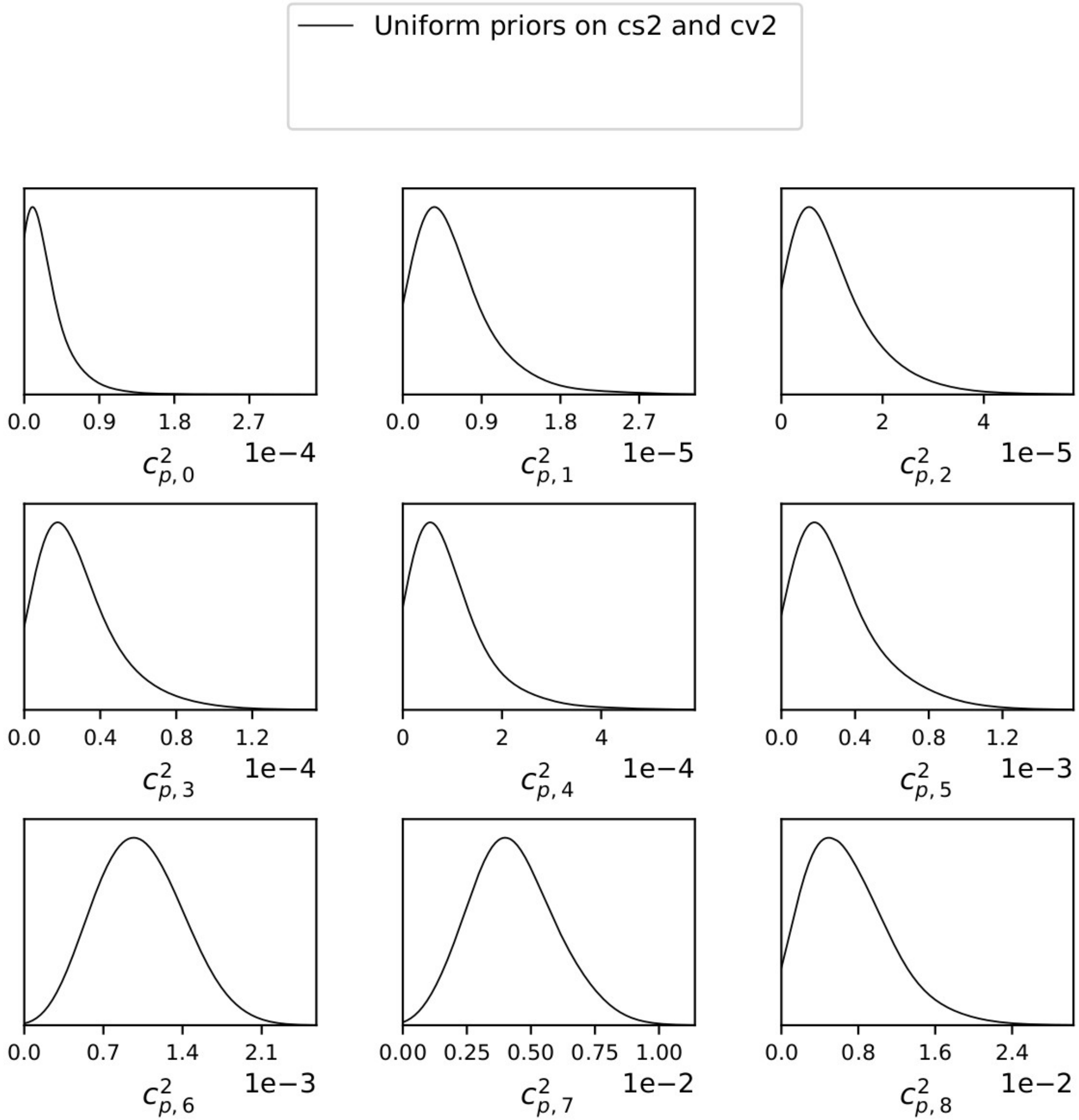


Effects of priors

Ilić et al, in prep

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

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



Effects of priors

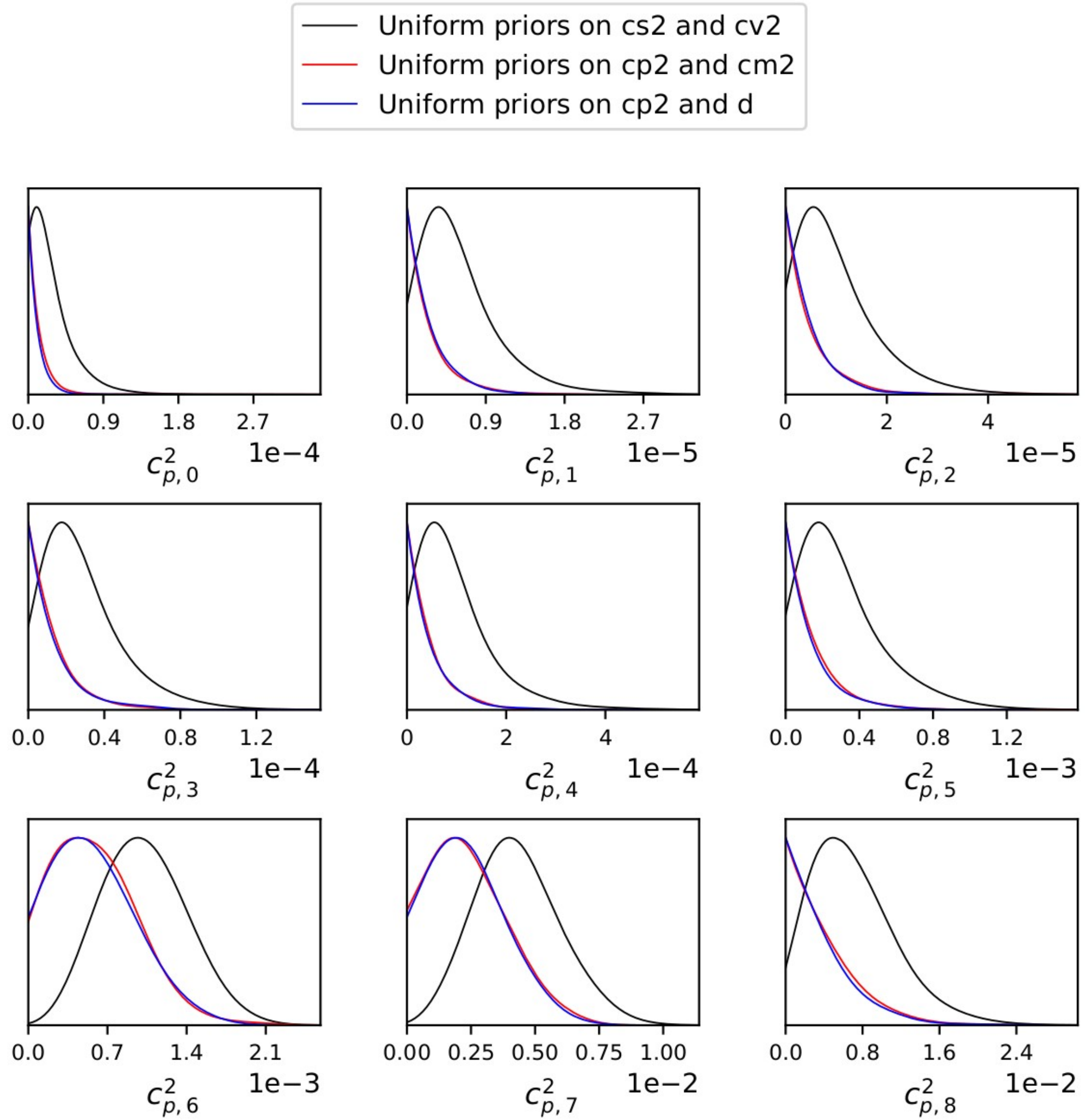
Ilić et al, in prep

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

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

$$(c_p^2, c_m^2) \qquad\qquad (c_p^2, d)$$

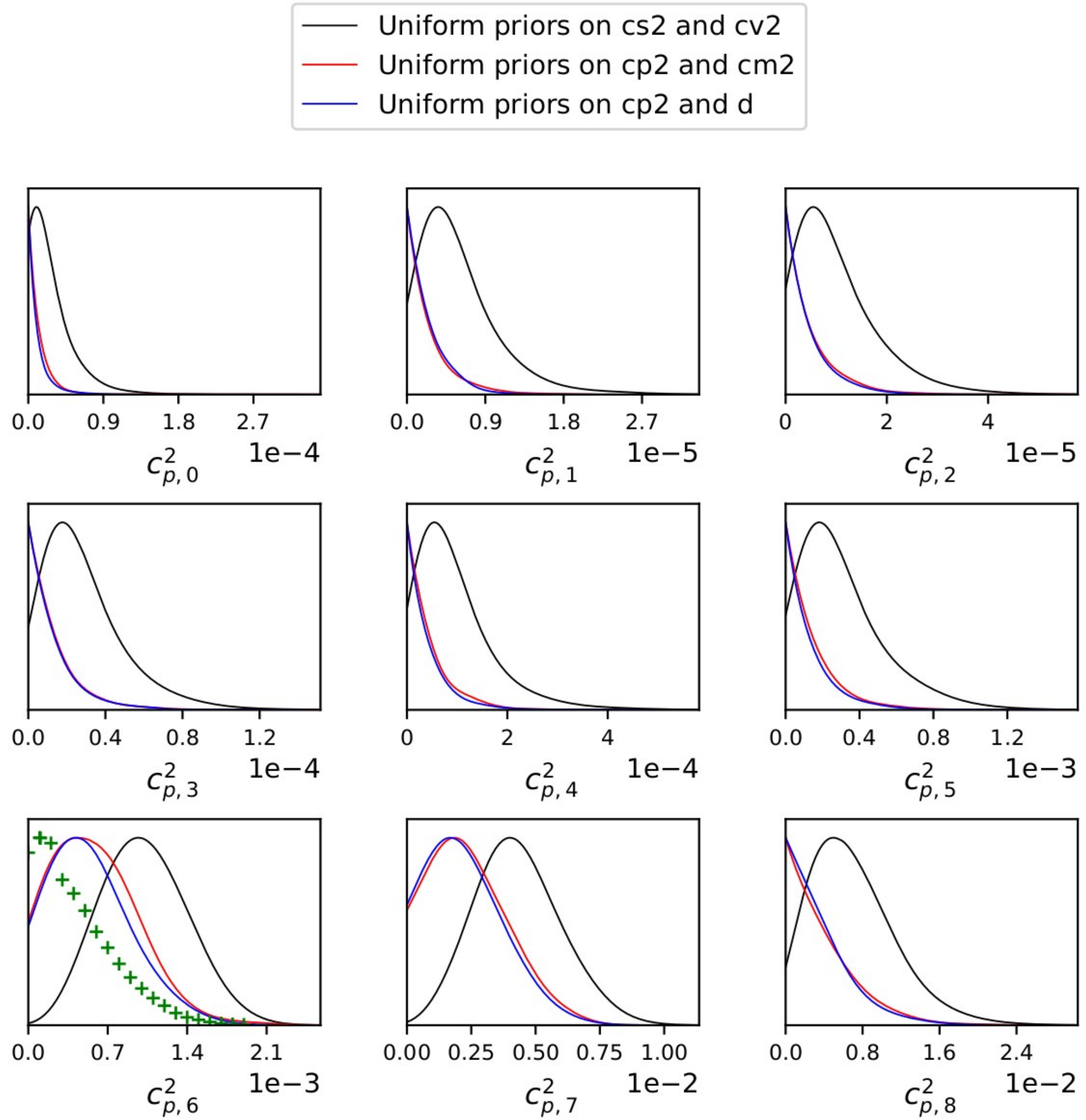
$$c_m^2 = \arctan\left(\frac{8}{15}\frac{c_v^2}{c_s^2}\right) \qquad\qquad d = \frac{c_s^2}{c_p^2}$$



Effects of priors

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

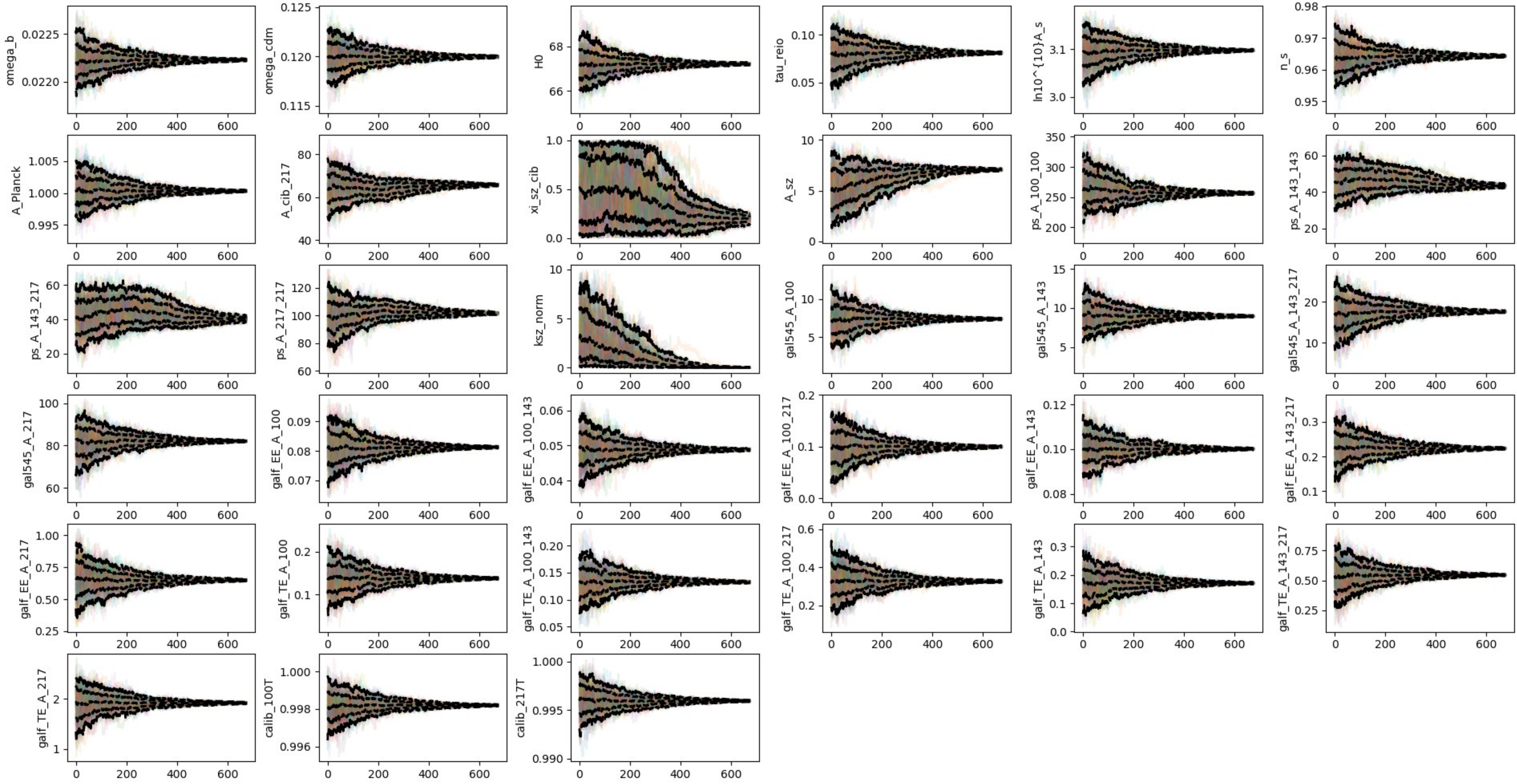
Ilić et al, in prep



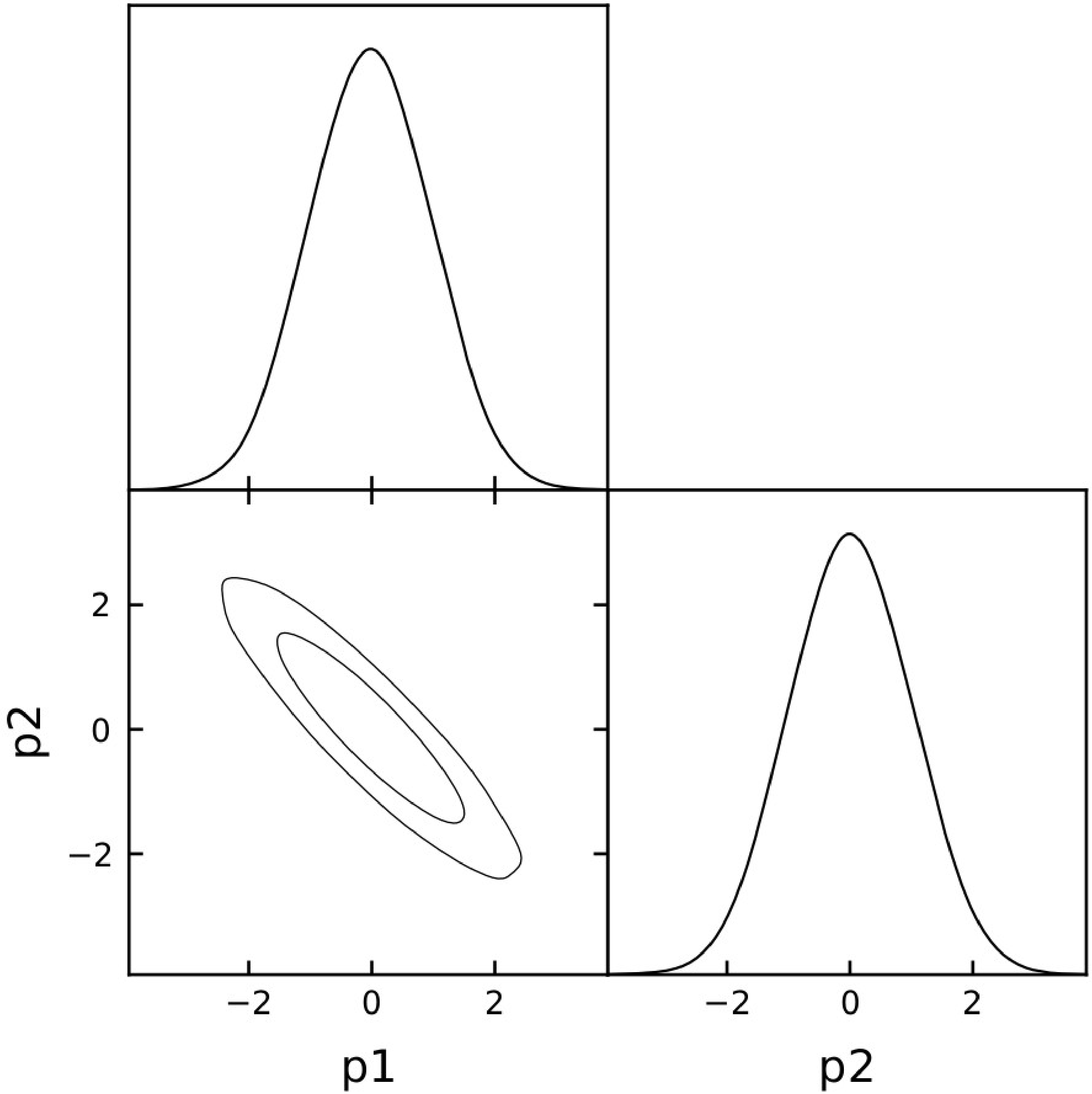
<u>Frequentist approach :</u>
Computation of the
"profile likelihood"
=
1D grid on given parameter,
minimize likelihood wrt
all other parameters

Identifying prior effects with JAM

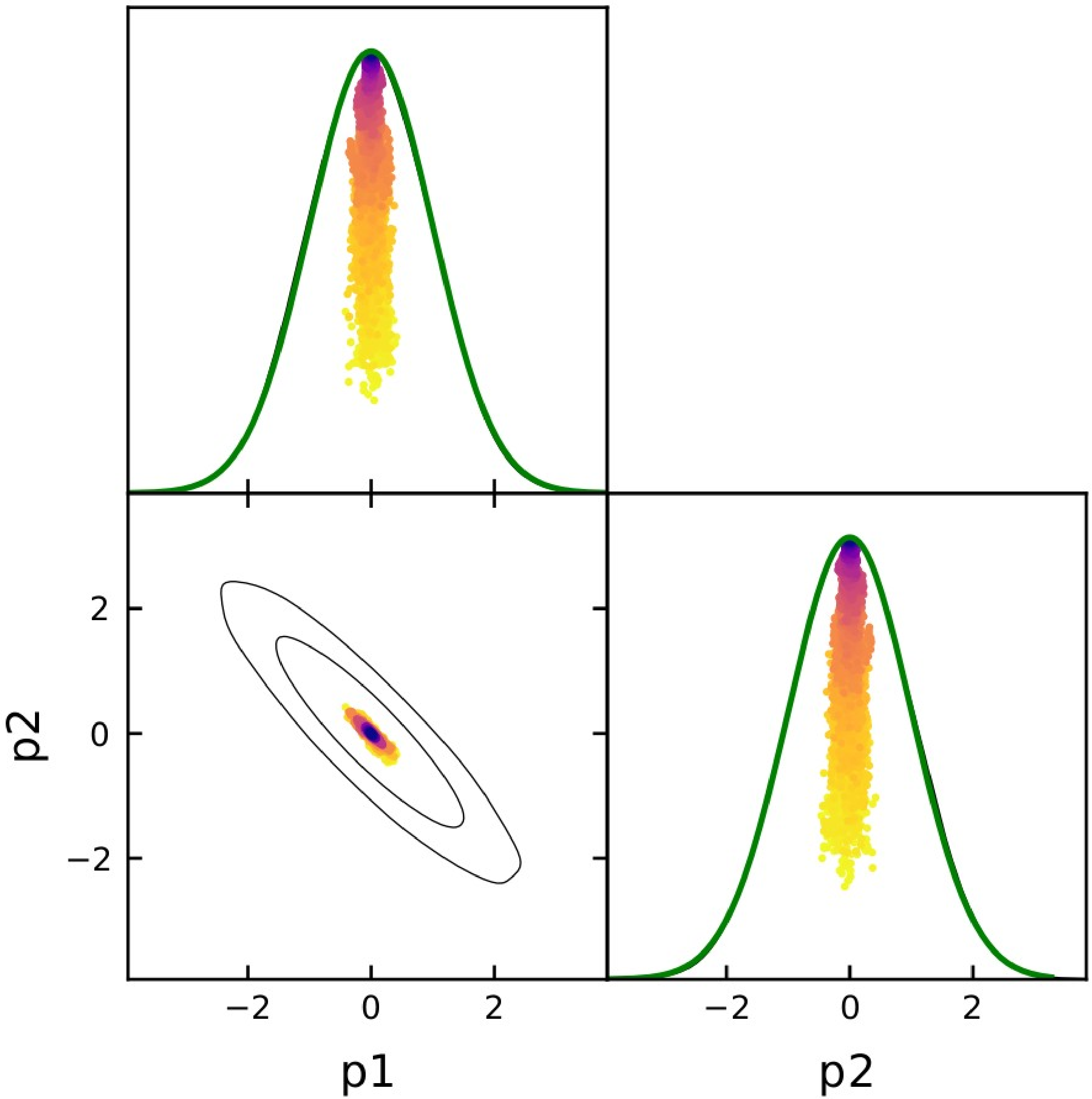
Identifying prior effects with JAM



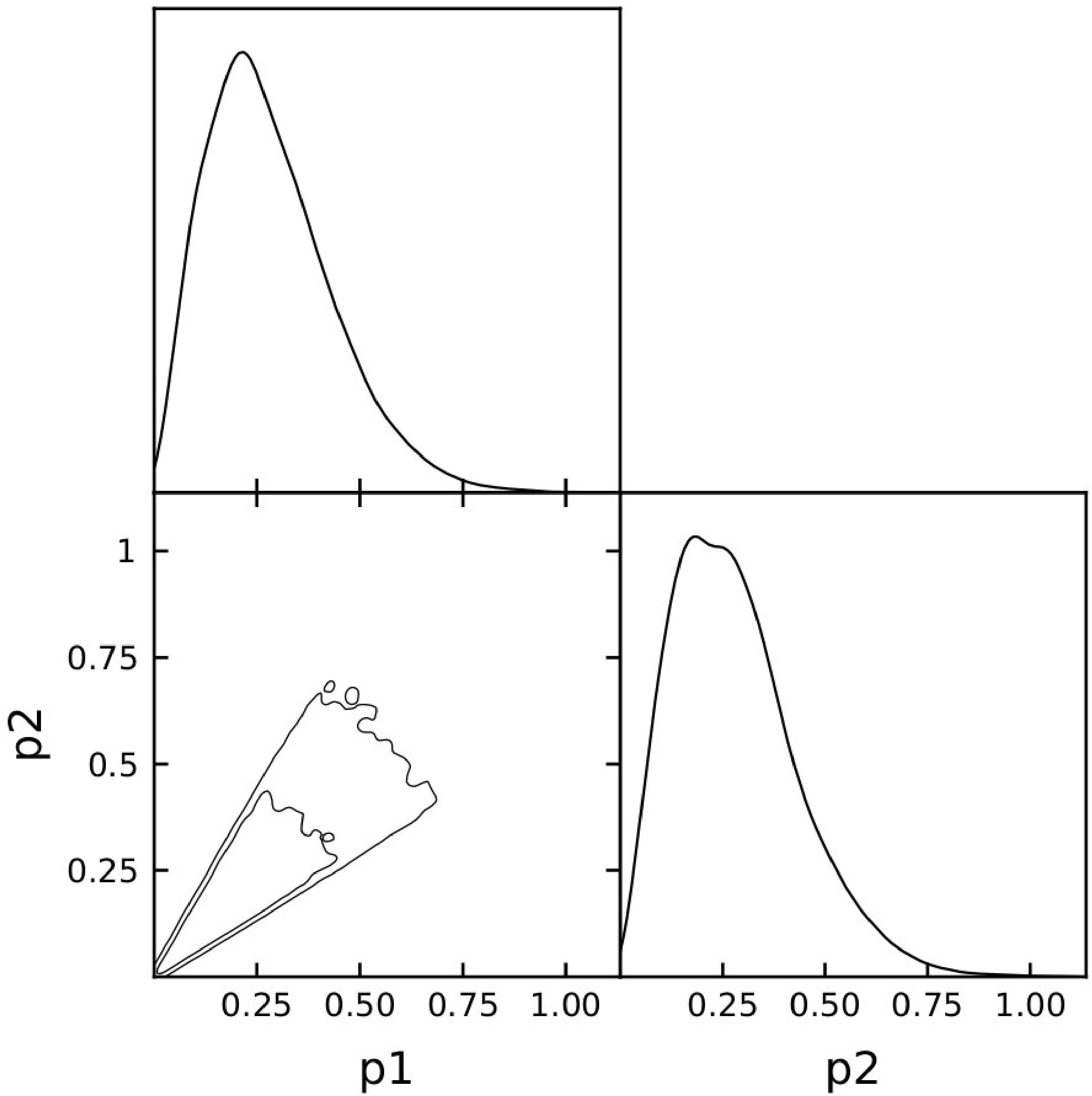
Identifying prior effects with JAM



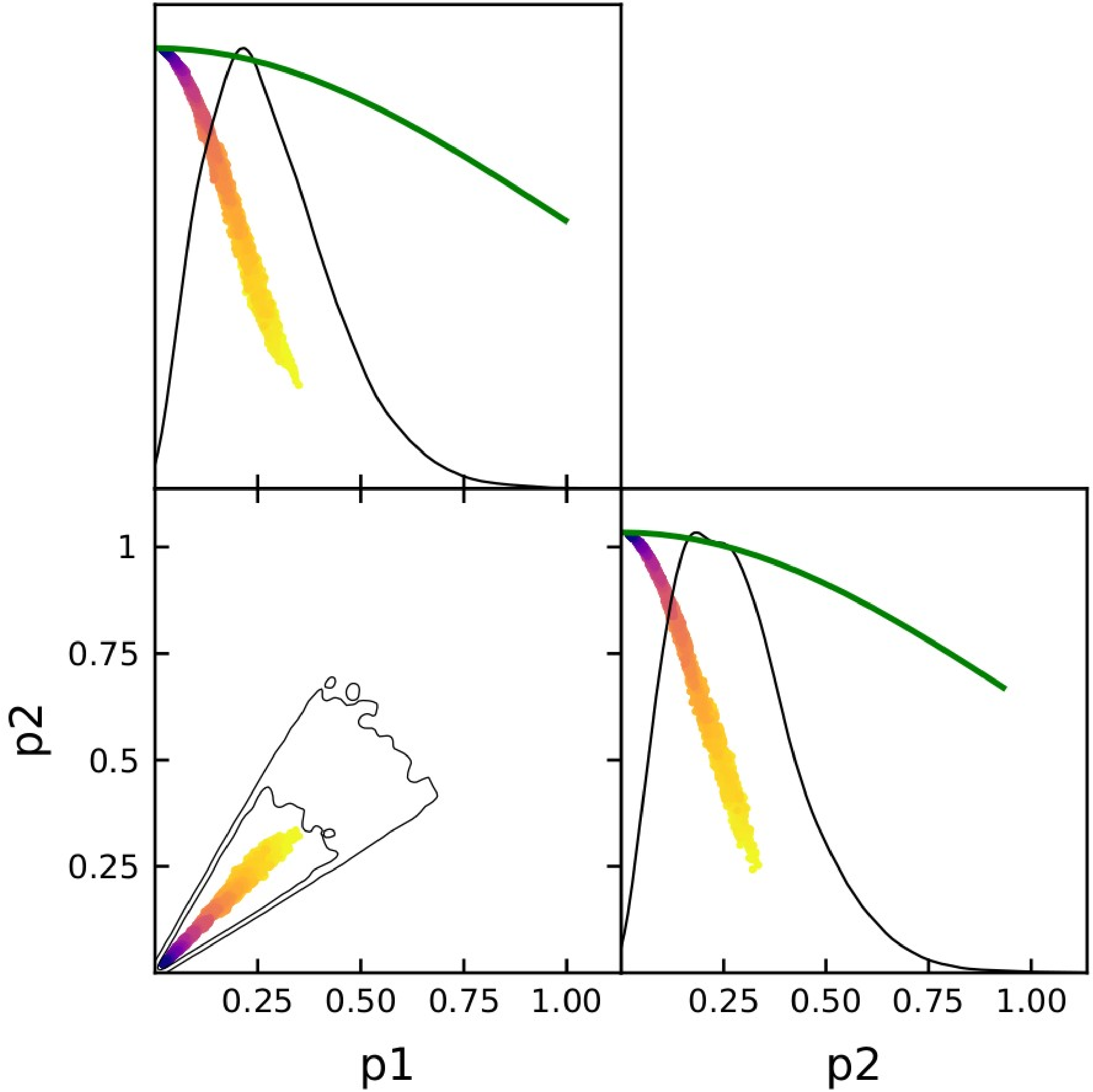
Identifying prior effects with JAM



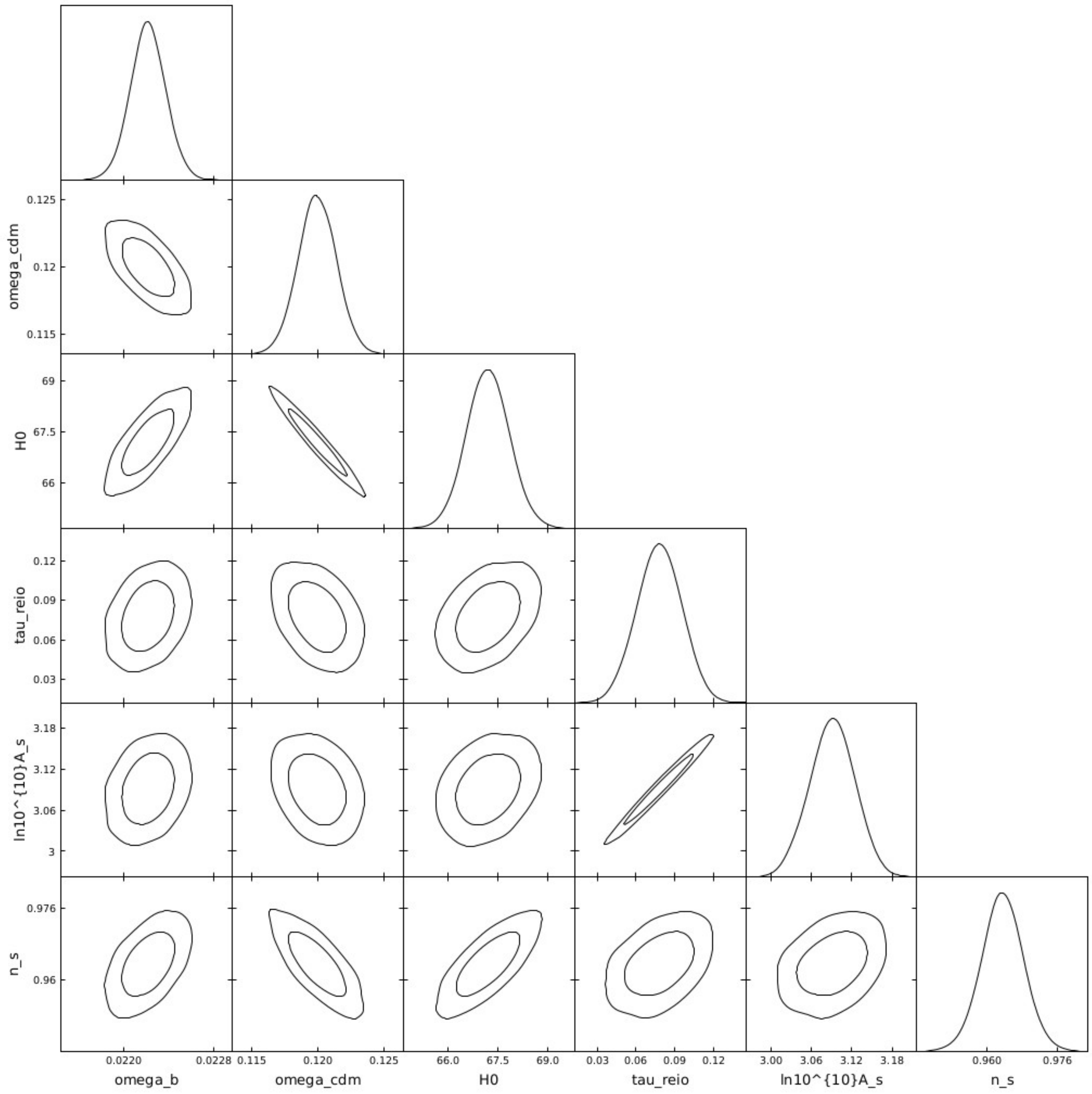
Identifying prior effects with JAM



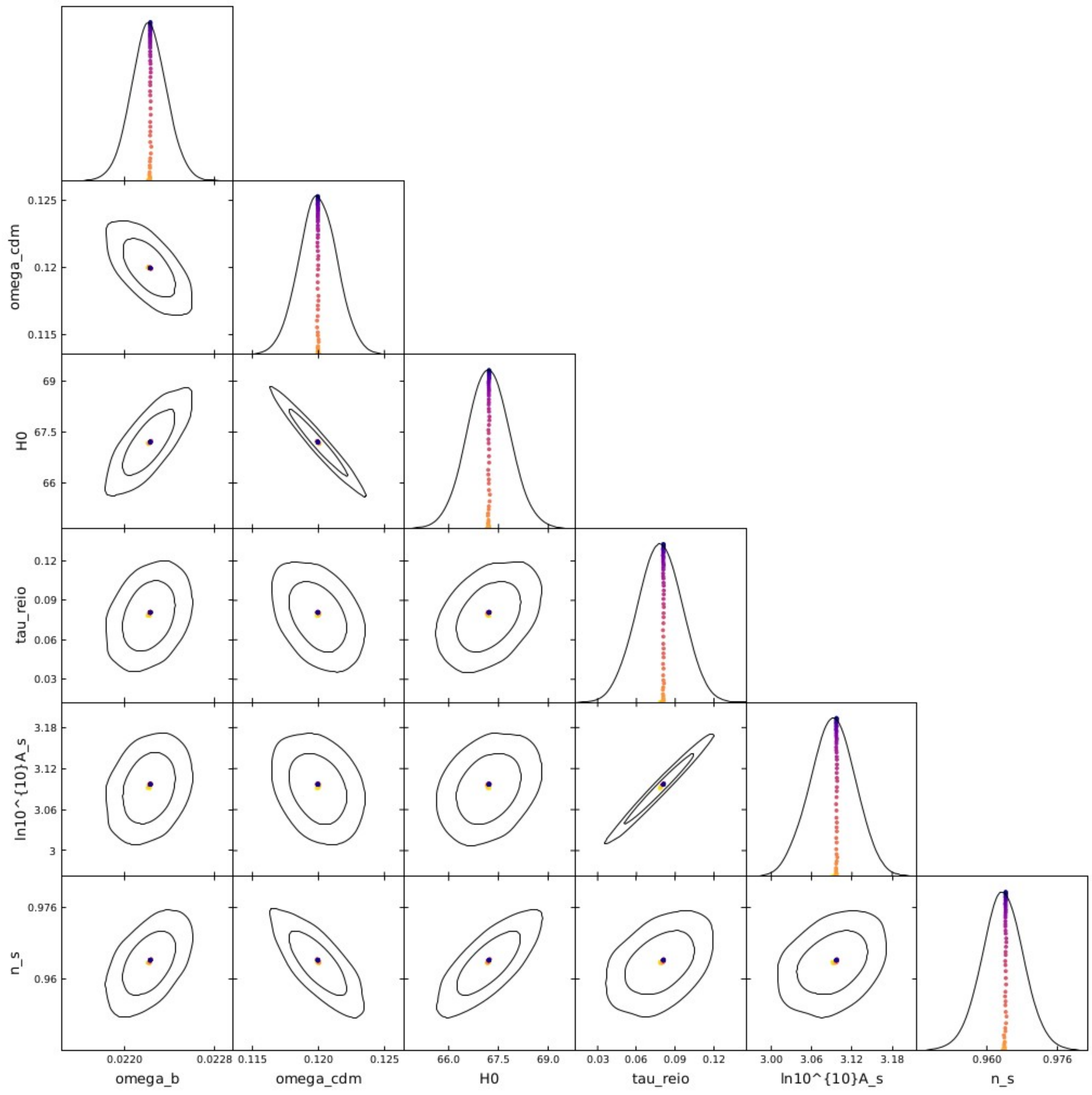
Identifying prior effects with JAM



Identifying prior effects with JAM



Identifying prior effects with JAM



Thank you
for your attention !