CLASS Beyond ΛCDM

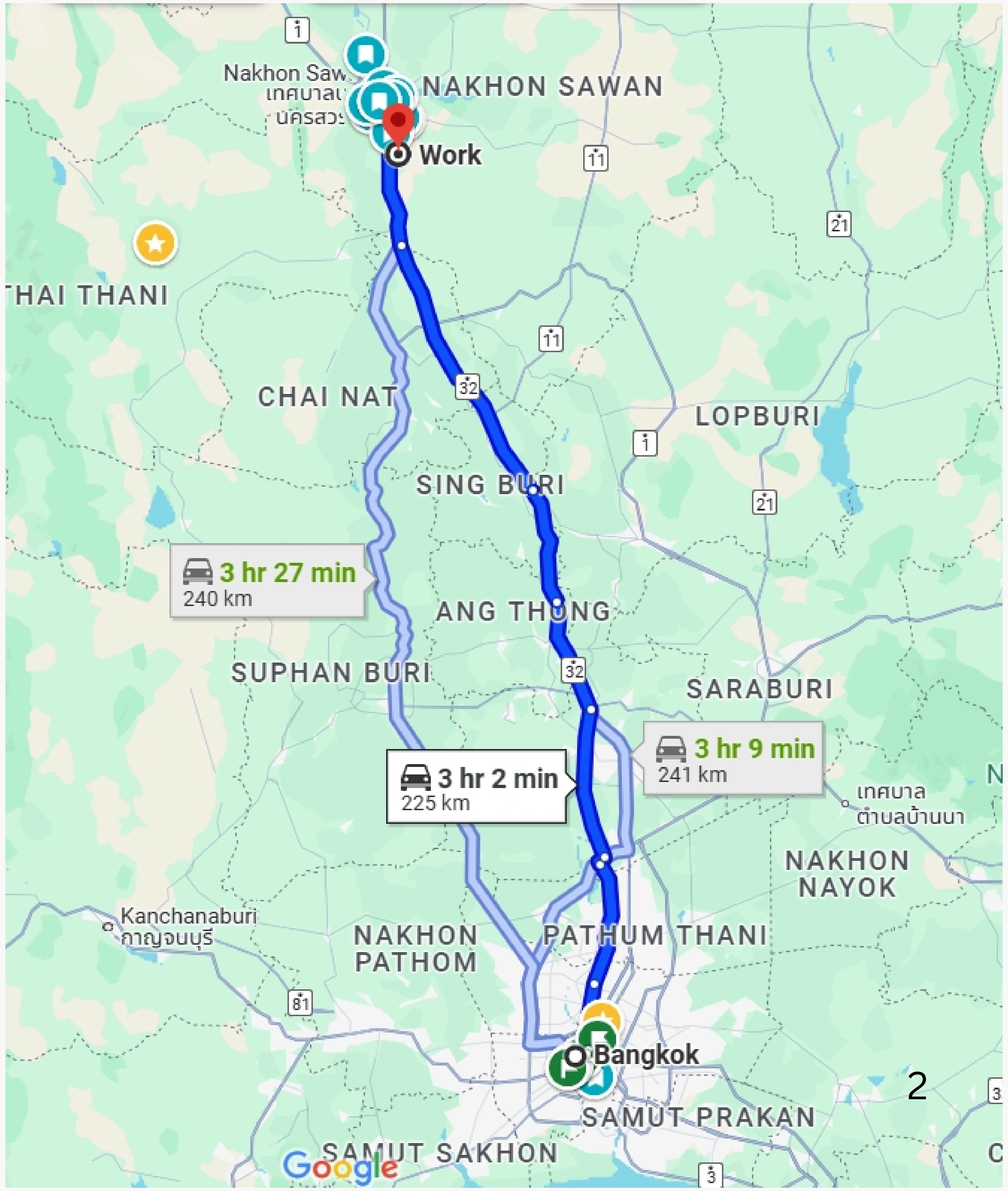
Cosmology with Computations Workshop (CosCOM)

15-21 December 2024

Nandan Roy
Centre for Theoretical Physics and Natural Philosophy
Mahidol University Nakhon Sawan Campus, Thailand

Centre for Theoretical Physics & Natural Philosophy, Mahidol University, Nakhonsawan Campus

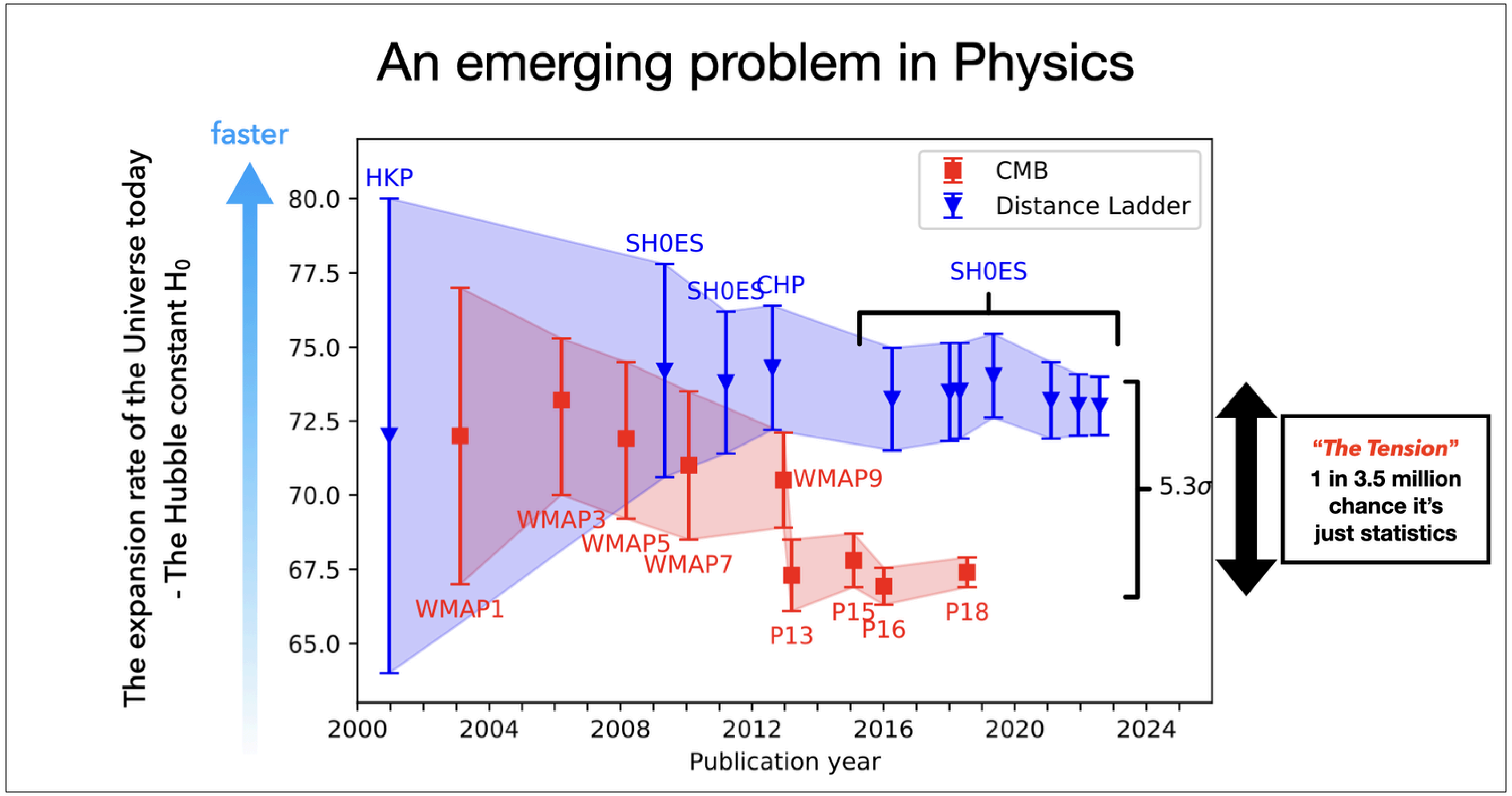
Centre for Theoretical Physics & Natural Philosophy
"Nakhonsawan Studiorum for Advanced Studies" - NAS



State of the Art

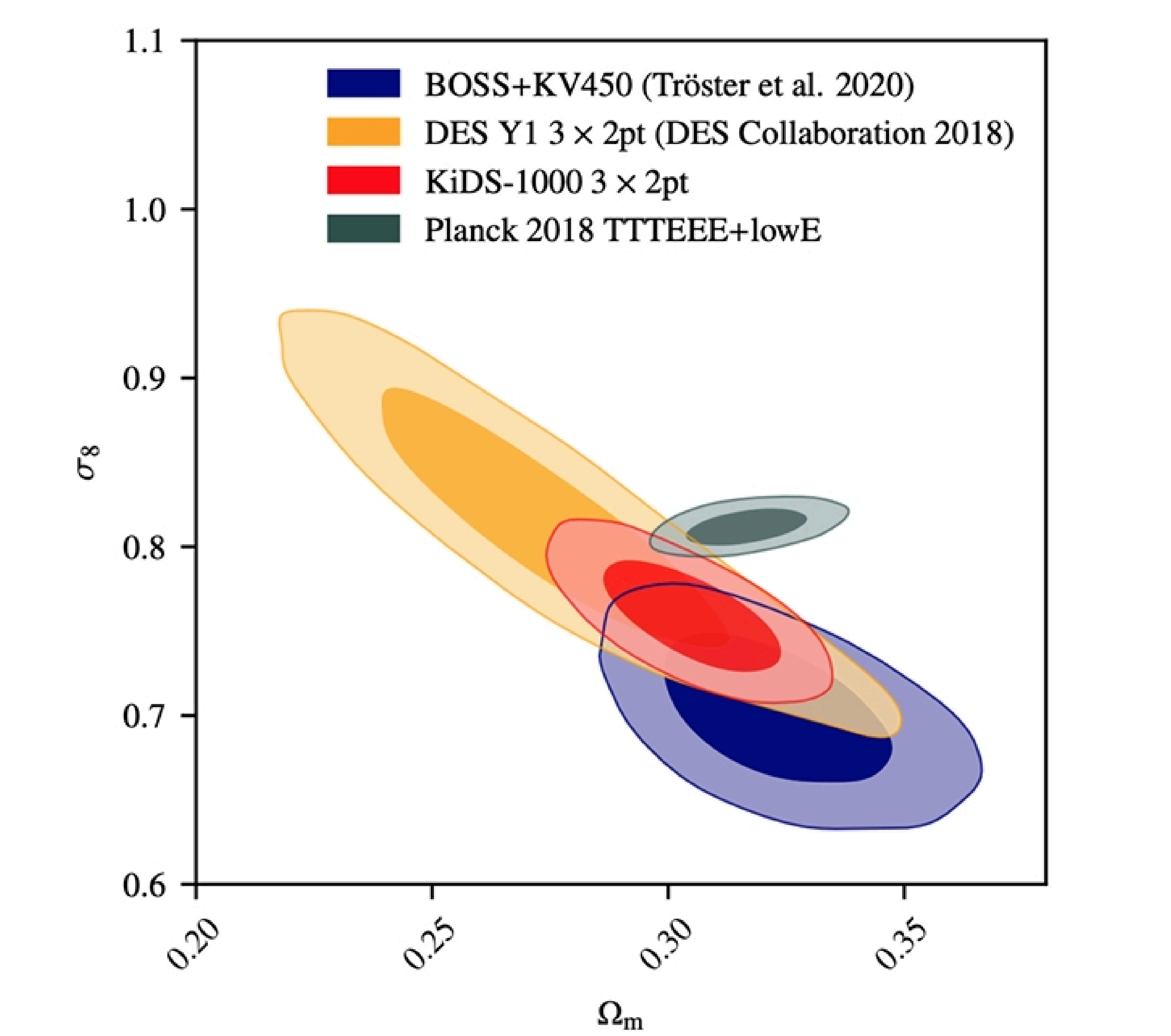
- Our universe is not only expanding but it is also accelerating!!
- ΛCDM model has been constrained with unprecedented accuracy. But it suffers from the challenges coming from both the theoretical and observational sides.
- From theoretical side it suffers from the problems like Cosmological Constant Problem, Coincidence Problem, Fine tuning problem etc.
- With the improvement in our ability to constrain the cosmological parameters, a few statistically significant tensions has emerged.
- It seems that the late time cosmological data and early time cosmological data are in tension.
- We need to extent our imagination beyond standard ΛCDM.

Current Tension in Cosmology

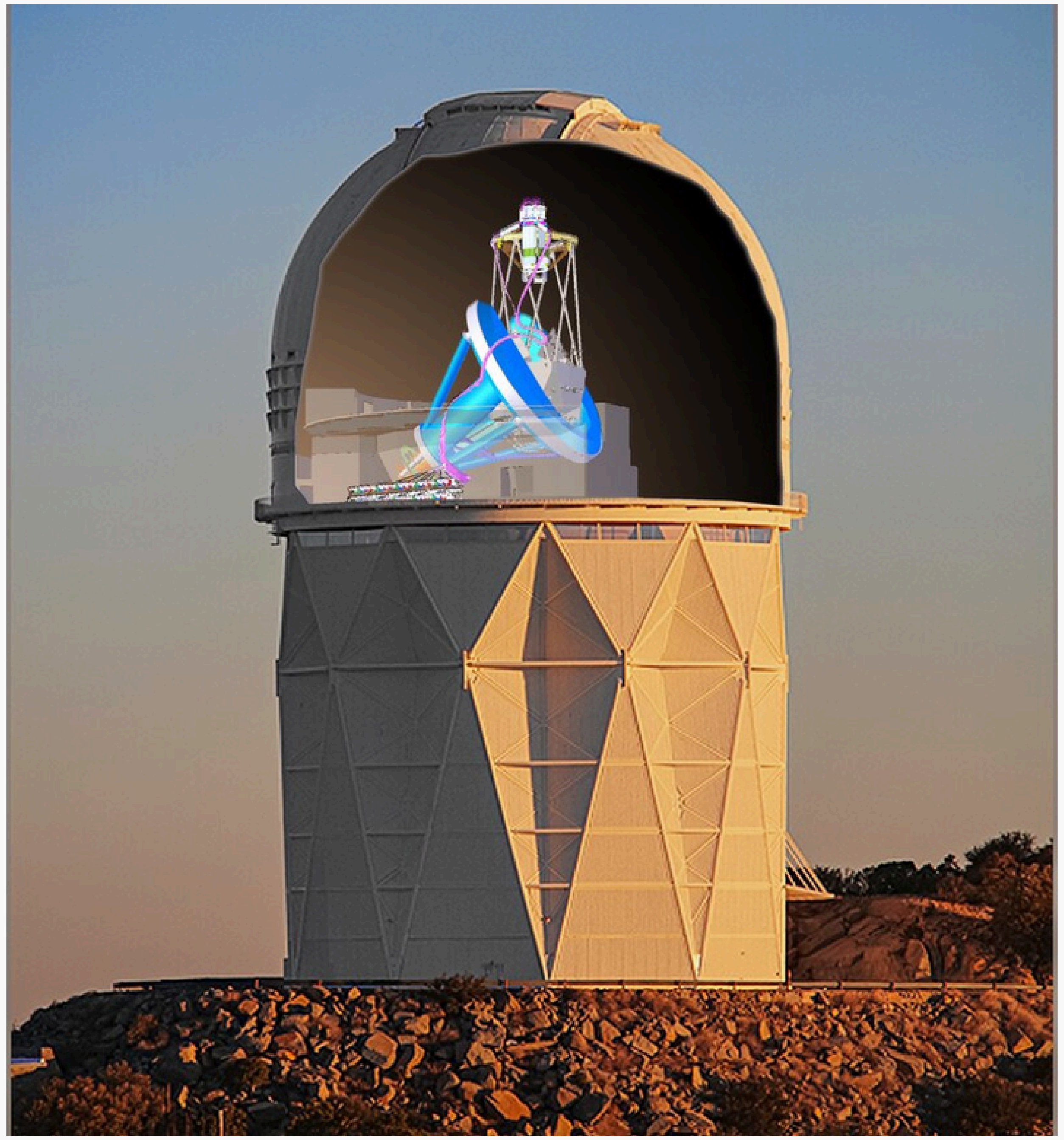


CMB Planck data together with BAO, BBN, and DES have constraint the Hubble parameter to be H0 (67.0 − 68.5)km/s/Mpc. On the other hand, cosmic distance ladder and time delay measurement like those reported by SH0ES and H0LiCOW collaborations have reported H0 = (74.03 ± 1.42) km/s/Mpc and H0 = (73.3 +1.7 −1.8) km/s/Mpc respectively by observing the local Universe.

σ_8 Tension



DESI 2024 Results

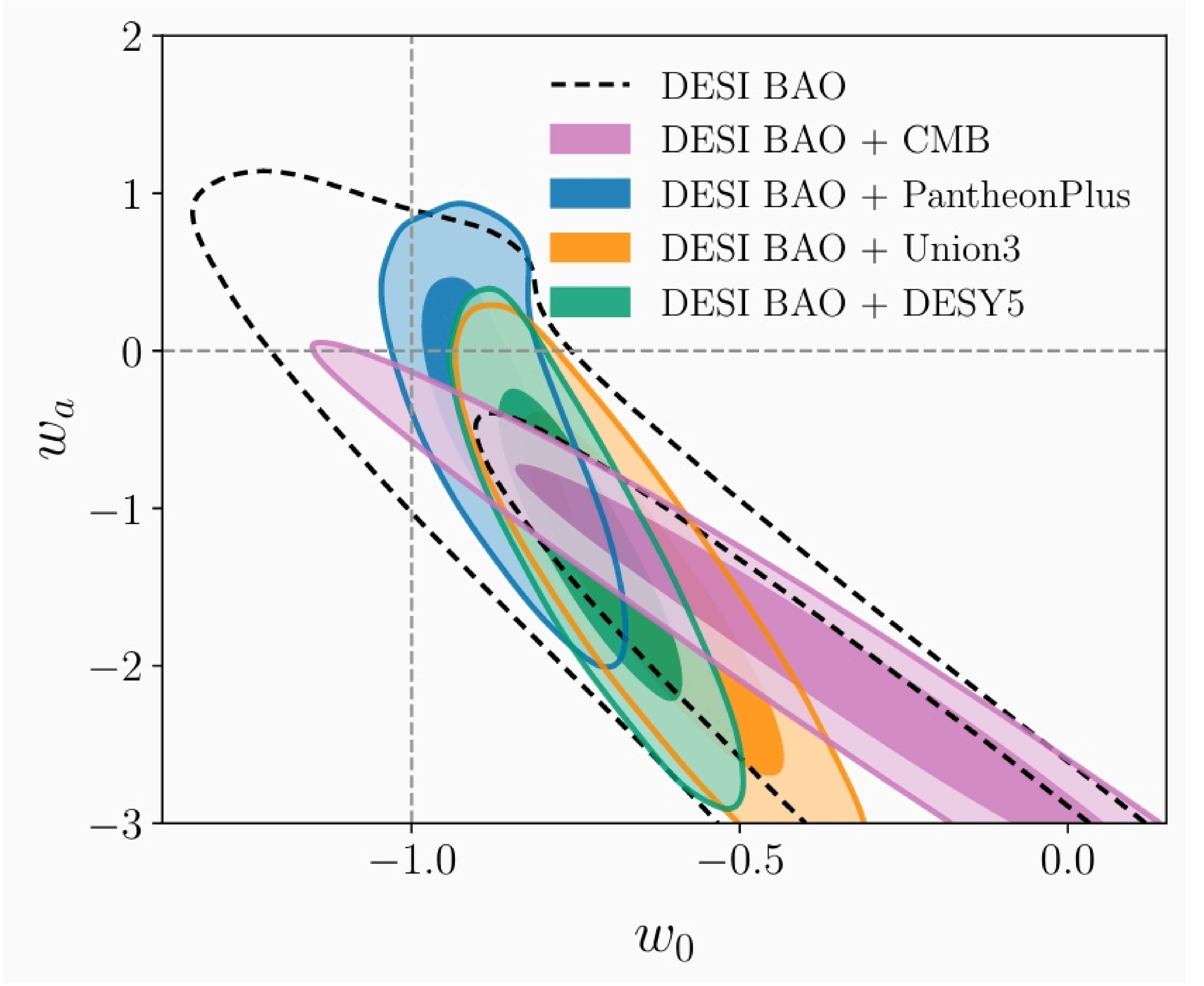


DESI installed on the Nicholas U. Mayall 4-meter Telescope at Kitt Peak National Observatory

Dark Energy Spectroscopic Instrument (DESI) data covers on the redshiftrange of z∈[0.1,4.2], the survey is divided in to seven redshift bins. This data contains five different samples, the Bright Galaxy Sample(BGS), Luminous Red Galaxy Samples(LRG), Emission Line Galaxy Sample(ELG), Quasar Sample(QSO) and Lyman-α Forest Sample(Lyα).

tracer	redshift	$N_{\rm tracer}$	$z_{\rm eff}$	$D_{\rm M}/r_{\rm d}$	$D_{\rm H}/r_{\rm d}$	r or $D_{\rm V}/r_{\rm d}$	$V_{\rm eff}$ (Gpc^3)
BGS	0.1 – 0.4	300,017	0.295	—	—	7.93 ± 0.15	1.7
LRG1	0.4 – 0.6	506,905	0.510	13.62 ± 0.25	20.98 ± 0.61	−0.445	2.6
LRG2	0.6 – 0.8	771,875	0.706	16.85 ± 0.32	20.08 ± 0.60	−0.420	4.0
LRG3+ELG1	0.8 – 1.1	1,876,164	0.930	21.71 ± 0.28	17.88 ± 0.35	−0.389	6.5
ELG2	1.1 – 1.6	1,415,687	1.317	27.79 ± 0.69	13.82 ± 0.42	−0.444	2.7
QSO	0.8 – 2.1	856,652	1.491	—	—	26.07 ± 0.67	1.5
Lya QSO	1.77 – 4.16	709,565	2.330	39.71 ± 0.94	8.52 ± 0.17	−0.477	—

DESI 2024 Results



(w_0, w_a) Parametrization

$$w(a) = w_0 + w_a(1 - a)$$

Suggests dynamical nature of the dark energy .

Cosmologists have compelling evidence to explore dynamical dark energy models.

Modification of *class*

You might be interested in implementing new features in *class*:

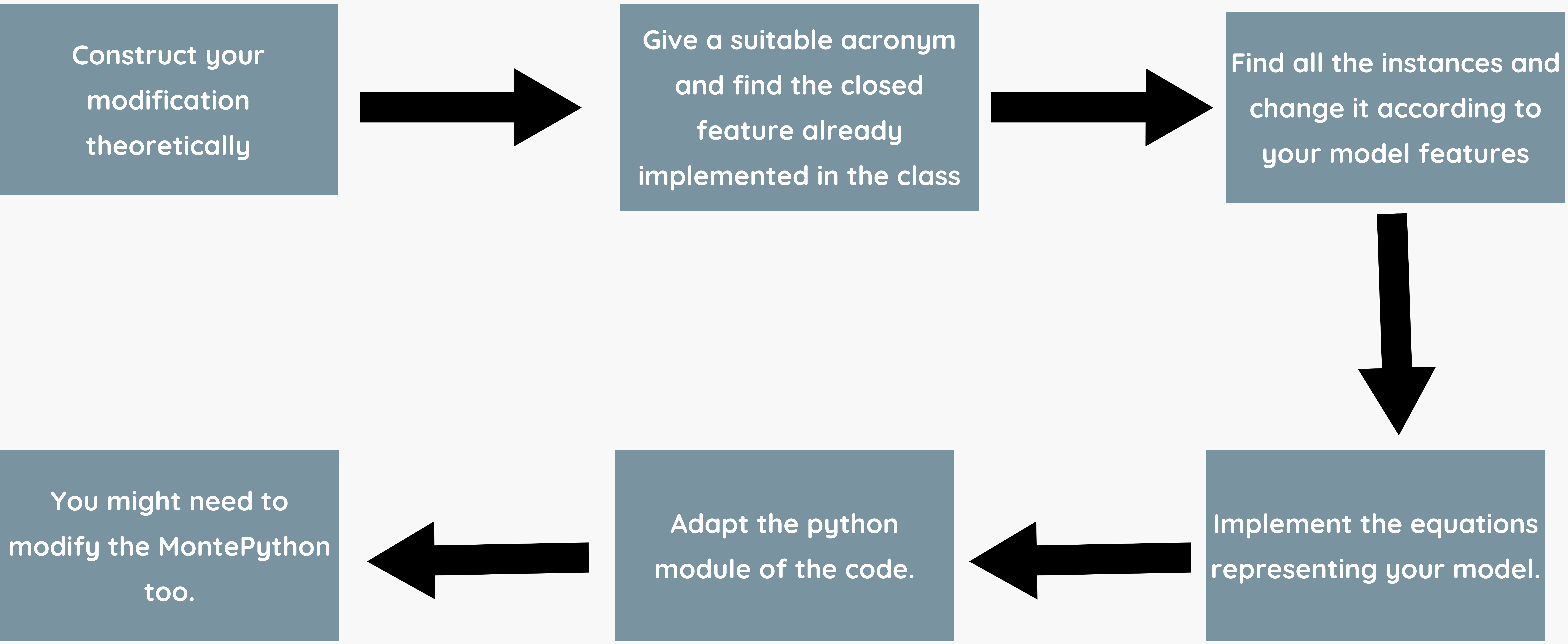
- May be addition of a new species.
- Modification of the Gravity.
- New mathematical description of the existing species.
- New input or output observables.

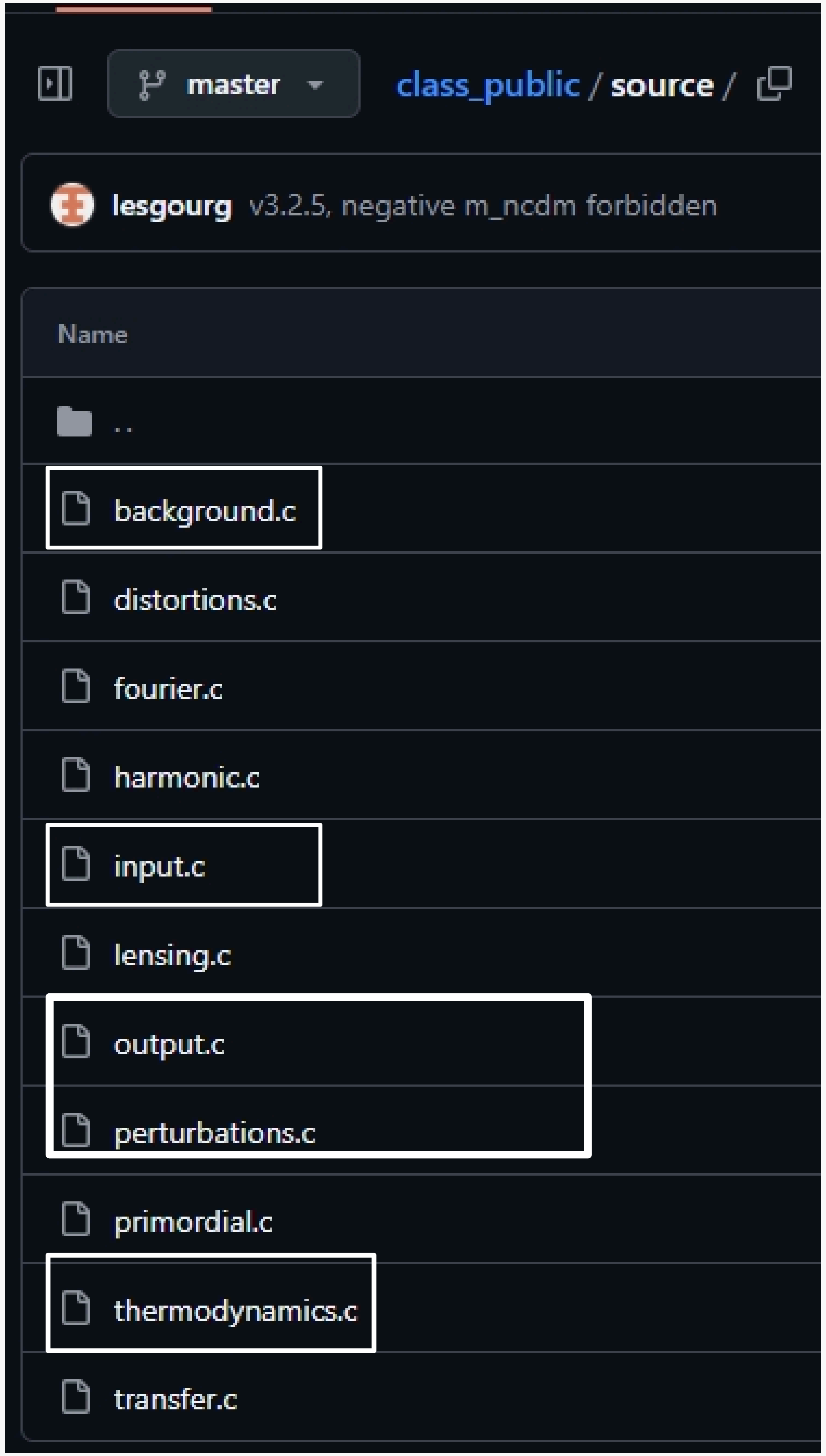
Modification of *class*

THE LOGIC

1. Think about an acronym that is easy to search in the code and distinguish your modification from the other acronyms in the code.
2. Think of the feature of your modification which is closest to the feature or species which is already implemented in class. Find that acronym and search for it. For example, in case you want to implement a new fluid search for the for: fld)
3. Find for all occurrences of fld in include/.h and source/.c (normally they are all within some "if (has_fld) (...)" and you can search directly for occurrences of has_fld)
4. Duplicate these occurrences according to your new model.
5. Change fld into your acronym.
6. Change some equations to describe the specific properties of your model.
7. Adapt the python module of the code.

Modification of *class*





Structure to modify CLASS

class_public/isource/xxxx.c

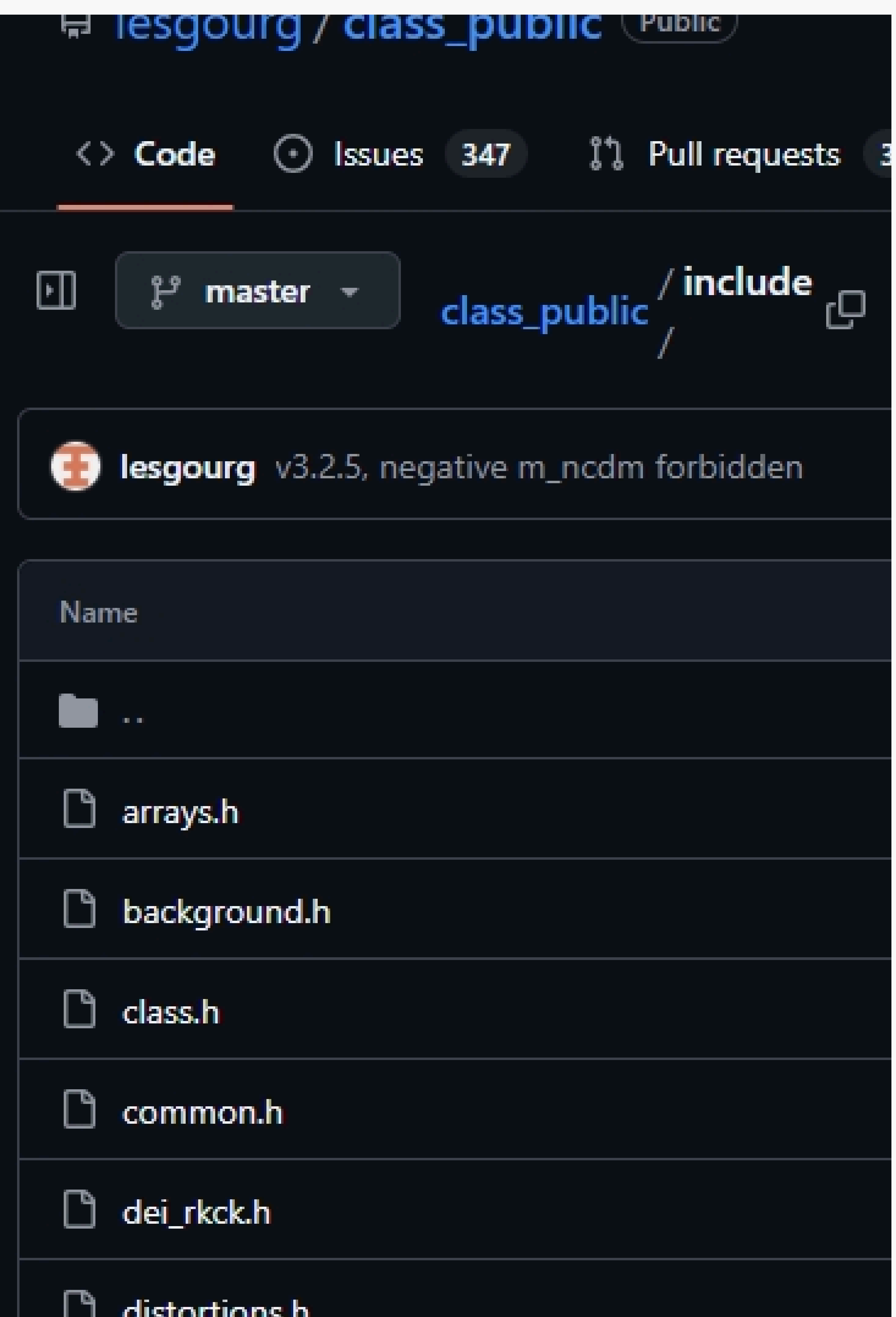
background.c : File to modify the background cosmology

input.c: to include the input parameters of your models

output.c: to include the output parameters of your models

perturbation.c modification of the perturbed equations

thermodynamics.c to modify the thermodynamics



Structure to modify CLASS

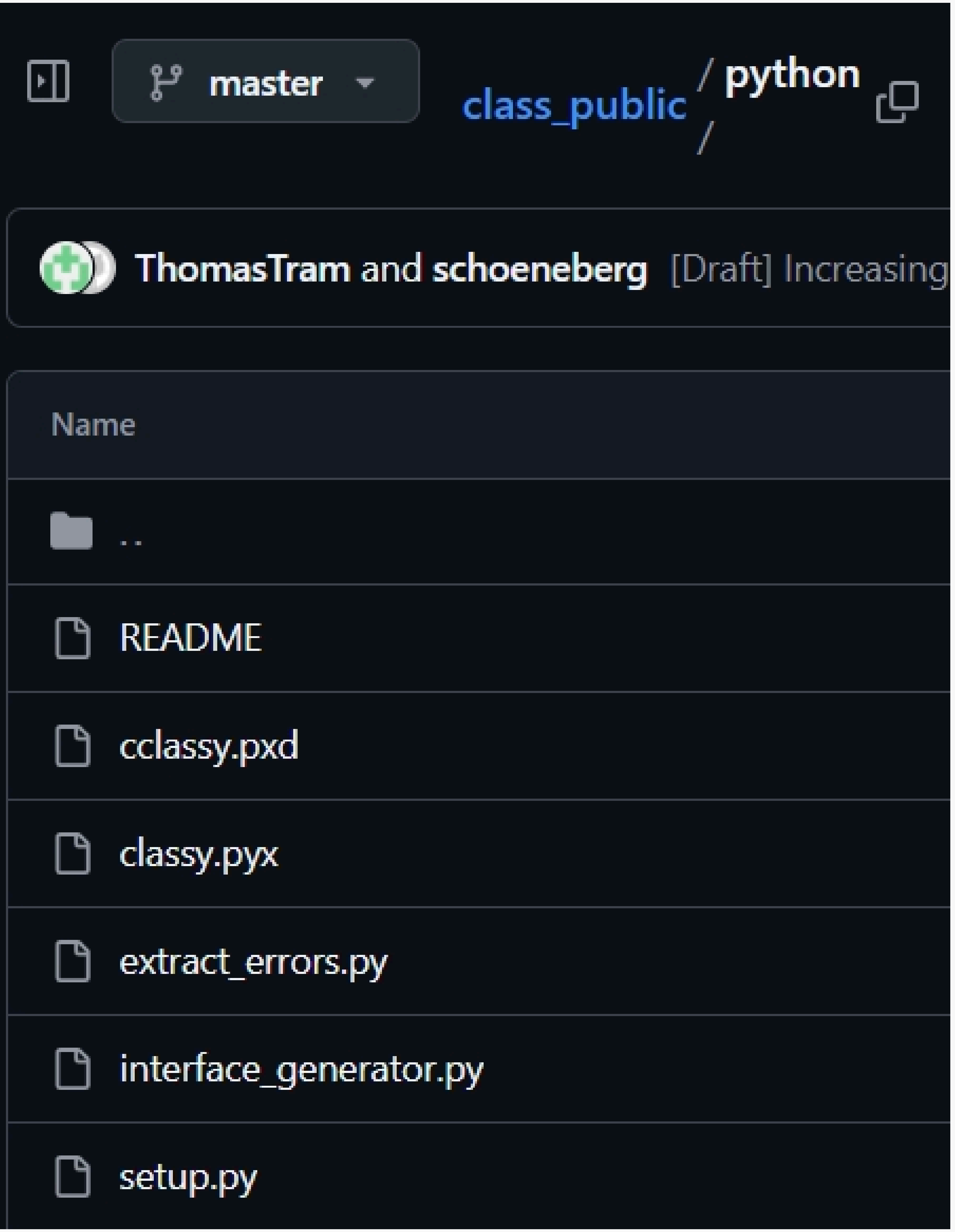
class_public/include/background.h

```
double Omega0_lambda;     /**< \f$ \Omega_{0_\Lambda} \f$:
double Omega0_fld;        /**< \f$ \Omega_{0 de} \f$: fluid
double Omega0_scf;        /**< \f$ \Omega_{0 scf} \f$: scal
short use_ppf; /**< flag switching on PPF perturbation equ
                  perturbation structure, but we leave it
double c_gamma_over_c_fld; /**< ppf parameter defined in e
enum equation_of_state fluid_equation_of_state; /**< param
double w0_fld;   /**< \f$ w0_{DE} \f$: current fluid equat
double wa_fld;   /**< \f$ wa_{DE} \f$: fluid equation of s
double cs2_fld;  /**< \f$ c^2_{s~DE} \f$: sound speed of t
                    not [delta p/delta rho] in the synchro
```

```
int index_bg_rho_g;          /**< photon density */
int index_bg_rho_b;          /**< baryon density */
int index_bg_rho_cdm;        /**< cdm density */
int index_bg_rho_idm;        /**< idm density */
int index_bg_rho_lambda;     /**< cosmological constant density */
int index_bg_rho_fld;        /**< fluid density */
int index_bg_w_fld;          /**< fluid equation of state */
int index_bg_rho_idr;        /**< density of interacting dark radiation */
int index_bg_rho_ur;         /**< relativistic neutrinos/relics density */
int index_bg_rho_dcdm;       /**< dcdm density */
int index_bg_rho_dr;         /**< dr density */
```

Add the parameter and the variables of your model

Structure to modify CLASS



class_public/include/python/

```
double Omega0_fld
double w0_fld
double wa_fld
double cs2_fld
double Omega0_ur
```

cclassy.pxd Add the same variable or parameter which you have added in include.h here so that MontePython can find it.

cclassy.pyx If you want to copute some observables as drive observable define it here

```
def theta_s_100(self):
    self.compute(["thermodynamics"])
    return 100.*self.th.rs_rec/self.th.da_rec/(1.+self.th.z_rec)

def theta_star_100(self):
    self.compute(["thermodynamics"])
    return 100.*self.th.rs_star/self.th.da_star/(1.+self.th.z_star)

def Omega_Lambda(self):
    return self.ba.Omega0_lambda
```

Lets Modify class

Read carefully the preamble of each file you want to modify.

```
/** - compute expansion rate H from Friedmann equation: this is the
    only place where the Friedmann equation is assumed. Remember
    that densities are all expressed in units of \f$ [3c^2/8\pi G] \f$, ie
    \f$ \rho_{class} = [8 \pi G \rho_{physical} / 3 c^2]\f$ */
pvecback[pba->index_bg_H] = sqrt(rho_tot-pba->K/a/a);

/** - compute derivative of H with respect to conformal time */
pvecback[pba->index_bg_H_prime] = - (3./2.) * (rho_tot + p_tot) * a + pba->K/a;
```

Densities are all expressed in units of

$$3c^2/8\pi G$$

$$\rho_{class} = [8\pi G\rho_{ph}/3c^2]$$

```
pvecback[pba->index_bg_H] = sqrt(rho_tot-pba->K/a/a);

/** - compute derivative of H with respect to conformal time */
pvecback[pba->index_bg_H_prime] = - (3./2.) * (rho_tot + p_tot) * a + pba->K/a;
```

Compute the density and pressure for each component and add it to the total density and pressure.

```
/* photons */
pvecback[pba->index_bg_rho_g] = pba->Omega0_g * pow(pba->H0,2) / pow(a,4);
rho_tot += pvecback[pba->index_bg_rho_g];
p_tot += (1./3.) * pvecback[pba->index_bg_rho_g];
dp_dloga += -(4./3.) * pvecback[pba->index_bg_rho_g];
rho_r += pvecback[pba->index_bg_rho_g];

/* baryons */
pvecback[pba->index_bg_rho_b] = pba->Omega0_b * pow(pba->H0,2) / pow(a,3);
rho_tot += pvecback[pba->index_bg_rho_b];
p_tot += 0;
rho_m += pvecback[pba->index_bg_rho_b];

/* cdm */
if (pba->has_cdm == _TRUE_) {
  pvecback[pba->index_bg_rho_cdm] = pba->Omega0_cdm * pow(pba->H0,2) / pow(a,3);
  rho_tot += pvecback[pba->index_bg_rho_cdm];
  p_tot += 0.;
  rho_m += pvecback[pba->index_bg_rho_cdm];
}
```

Dark Energy in class

class_public/explanatory.ini

```
# 8) Dark energy contributions.
#    At least one out of three conditions must be satisfied:
#          - 'Omega_Lambda' unspecified.
#          - 'Omega_fld' unspecified.
#          - 'Omega_scf' set to a negative value. [Will be refered to as
#            unspecified in the following text.]
#      The code will then use the first unspecified component to satisfy the
#      closure equation (sum_i Omega_i) equals (1 + Omega_k)
#      (default: 'Omega_fld' and 'Omega_scf' set to 0 and 'Omega_Lambda'
#      inferred by code)
Omega_fld = 0
Omega_scf = 0
# Omega_Lambda = 0.7
```

You can chose three different description of the dark energy components.

- Cosmological Constant
- Fluid
- Scalar Field

Cosmological Constant

```
/* Lambda */
if (pba->has_lambda == _TRUE_) {
  pvecback[pba->index_bg_rho_lambda] = pba->Omega0_lambda * pow(pba->H0,2);
  rho_tot += pvecback[pba->index_bg_rho_lambda];
  p_tot -= pvecback[pba->index_bg_rho_lambda];
}
```

Scalar Field

```
if (pba->has_fld == _TRUE_) {

  /* get rho_fld from vector of integrated variables */
  pvecback[pba->index_bg_rho_fld] = pvecback_B[pba->index_bi_rho_fld];

  /* get w_fld from dedicated function */
  class_call(background_w_fld(pba,a,&w_fld,&dw_over_da,&integral_fld), pba->error_message, pba->error_message);
  pvecback[pba->index_bg_w_fld] = w_fld;

  // Obsolete: at the beginning, we had here the analytic integral solution corresponding to the case w=w0+w1(1-a/a0):
  // pvecback[pba->index_bg_rho_fld] = pba->Omega0_fld * pow(pba->H0,2) / pow(a,3.*(1.+pba->w0_fld+pba->wa_fld)) * exp(3.*pba->wa_fld*(a-1.));
  // But now everthing is integrated numerically for a given w_fld(a) defined in the function background_w_fld.

  rho_tot += pvecback[pba->index_bg_rho_fld];
  p_tot += w_fld * pvecback[pba->index_bg_rho_fld];
  dp_dloga += (a*dw_over_da-3*(1+w_fld)*w_fld)*pvecback[pba->index_bg_rho_fld];
}
```

```
/* Derivative of total pressure w.r.t. conformal time */
pvecback[pba->index_bg_p_tot_prime] = a*pvecback[pba->index_bg_H]*dp_dloga;
if (pba->has_scf == _TRUE_) {
  /** The contribution of scf was not added to dp_dloga, add p_scf_prime here: */
  pvecback[pba->index_bg_p_prime_scf] = pvecback[pba->index_bg_phi_prime_scf]*
    (-pvecback[pba->index_bg_phi_prime_scf]*pvecback[pba->index_bg_H]/a-2./3.*pvecback[pba->index_bg_dV_scf]);
  pvecback[pba->index_bg_p_tot_prime] += pvecback[pba->index_bg_p_prime_scf];
}
```

Fluid Description

```
  rho_tot += pvecback[pba->index_bg_rho_fld];
  p_tot += w_fld * pvecback[pba->index_bg_rho_fld];
  dp_dloga += (a*dw_over_da-3*(1+w_fld)*w_fld)*pvecback[pba->index_bg_rho_fl
}
```

```
switch (pba->fluid_equation_of_state) {
case CLP:
  *w_fld = pba->w0_fld + pba->wa_fld * (1. - a);
  break;
```

```
/** - then, give the corresponding analytic derivative dw/da
    by perturbation equations; we could compute it numerical
    but with a loss of precision; as long as there is a simp
    analytic expression of the derivative of the previous
    function, let's use it! */
switch (pba->fluid_equation_of_state) {
case CLP:
  *dw_over_da_fld = - pba->wa_fld;
  break;
```

```
/** - finally, give the analytic solution of the following integral:
    \f$ \int_{a}^{a0} da 3(1+w_{fld})/a \f$. This is used in only
    one place, in the initial conditions for the background, and
    with a=a_ini. If your w(a) does not lead to a simple analytic
    solution of this integral, no worry: instead of writing
    something here, the best would then be to leave it equal to
    zero, and then in background_initial_conditions() you should
    implement a numerical calculation of this integral only for
    a=a_ini, using for instance Romberg integration. It should be
    fast, simple, and accurate enough. */
switch (pba->fluid_equation_of_state) {
case CLP:
  *integral_fld = 3.*((1.+pba->w0_fld+pba->wa_fld)*log(1./a) + pba->wa_fld*(a-1.));
  break;
```

Modification of the perturbation

class_public/source/perturbations.c

pba->has_scf

```
*/
if (pba->has_scf == _TRUE_) {

  if (ppt->gauge == synchronous){
    delta_rho_scf =  1./3.*
      (1./a2*ppw->pvecback[pba->index_bg_phi_prime_scf]*y[ppw->pv->index_pt_phi_prime_scf]
       + ppw->pvecback[pba->index_bg_dV_scf]*y[ppw->pv->index_pt_phi_scf]);
    delta_p_scf = 1./3.*
      (1./a2*ppw->pvecback[pba->index_bg_phi_prime_scf]*y[ppw->pv->index_pt_phi_prime_scf]
       - ppw->pvecback[pba->index_bg_dV_scf]*y[ppw->pv->index_pt_phi_scf]);
  }
  else{
    /* equation for psi */
    psi = y[ppw->pv->index_pt_phi] - 4.5 * (a2/k/k) * ppw->rho_plus_p_shear;

    delta_rho_scf =  1./3.*
      (1./a2*ppw->pvecback[pba->index_bg_phi_prime_scf]*y[ppw->pv->index_pt_phi_prime_scf]
       + ppw->pvecback[pba->index_bg_dV_scf]*y[ppw->pv->index_pt_phi_scf]
       - 1./a2*pow(ppw->pvecback[pba->index_bg_phi_prime_scf],2)*psi);
    delta_p_scf =  1./3.*
      (1./a2*ppw->pvecback[pba->index_bg_phi_prime_scf]*y[ppw->pv->index_pt_phi_prime_scf]
       - ppw->pvecback[pba->index_bg_dV_scf]*y[ppw->pv->index_pt_phi_scf]
       - 1./a2*pow(ppw->pvecback[pba->index_bg_phi_prime_scf],2)*psi);
  }
```

pba->has_fld

```
/* fluid contribution */
if (pba->has_fld == _TRUE_) {

  class_call(background_w_fld(pba,a,&w_fld,&dw_ove
  w_prime_fld = dw_over_da_fld * a_prime_over_a * 

  if (pba->use_ppf == _FALSE_) {
    ppw->delta_rho_fld = ppw->pvecback[pba->index_
    ppw->rho_plus_p_theta_fld = (1.+w_fld)*ppw->pv
    ca2_fld = w_fld - w_prime_fld / 3. / (1.+w_fld
    /** We must gauge transform the pressure pertu
    ppw->delta_p_fld = pba->cs2_fld * ppw->delta_r
  }
```

Let us modify the fluid description

Jassal-Bagla-Padmanabhanparametrization(JBP)

$$w(z) = w_0 + \frac{z}{(1+z)^2} w_1$$

- **Jassal, H.K.; Bagla, J.S.; Padmanabhan, T. WMAP constraints on low redshift evolution of dark energy. Mon. Not. Roy. Astron. Soc. 2005, 356, L11.**

$$\rho + 3H\rho(1 + w(a)) = 0$$

$$\rho = \rho_0 Exp[\int_a^{a_0} da\, 3(1 + w(a))/a]$$

Let us modify the fluid description

Jassal-Bagla-Padmanabhanparametrization(JBP)

$$w(z) = w_0 + \frac{z}{(1+z)^2} w_1$$

- **Write the EOS in terms of w(a).**
- **Find the differentiation:**

$$\frac{dw(a)}{da}$$

- **Find the integration:**

$$\int_a^{a0} da \frac{3(1+w_{fld})}{a}$$

Let us modify the fluid description

Jassal-Bagla-Padmanabhanparametrization(JBP)

$$w(a) = w_0 + \frac{\left(\frac{a_0}{a} - 1\right)a^2 w_1}{a_0^2}$$

$$\frac{dw(a)}{da} = \frac{(a_0 - 2a)w_1}{a_0^2}$$

$$\int_a^{a0} da \frac{3(1 + w_{fld})}{a} = -3(1 + w_0)\ln\left(\frac{a}{a_0}\right) + \frac{3}{2} w_1 \left(1 - \frac{a}{a_0}\right)^2$$

Let us modify the fluid description

include/background.h

```c
#ifndef __BACKGROUND__
#define __BACKGROUND__

#include "common.h"
#include "quadrature.h"
#include "growTable.h"
#include "arrays.h"
#include "dei_rkck.h"
#include "parser.h"

/** List of possible parametrisations of the

enum equation_of_state {CLP,EDE,COSCOM};


```

```c
                      perturbation structure, but we lea
  double c_gamma_over_c_fld; /**< ppf parameter define
  enum equation_of_state fluid_equation_of_state; /**<
  double w0_fld;   /**< \f$ w0_{DE} \f$: current fluid
  double wa_fld; /**< \f$ wa_{DE} \f$: fluid equation
  double w0_cos;
  double w1_cos;
  double cs2_fld;  /**< \f$ c^2_{s~DE} \f$: sound spee
this is
```

Let us modify the fluid description

source/background.c

```
  case COSCOM:
    *w_fld = pba->w0_cos + pba->w1_cos * (1. - a)*a;
  break;
}
```

The EOS

```
case COSCOM:
  *dw_over_da_fld = pba->w1_cos*(1 - 2.*a);
  break;
```

The Derivative

```
 case COSCOM:
   *integral_fld = -3.0*(1+pba->w0_cos)*log(a) + 1.5*pba->w1_cos*pow((1 -a),2);
   break;
 }
```

The Integration

Let us modify the fluid description

source/input.c

```c
      }
      else if ((strstr(string1,"COSCOM") != NULL) || (strstr(string1,"COSCOM") != NULL)) {
        pba->fluid_equation_of_state = COSCOM;
      }
      else {
        class_stop(errmsg,"incomprehensible input '%s' for the field 'fluid_equation_of_state'",string1);
      }
    }

    if (pba->fluid_equation_of_state == CLP) {
      /** 8.a.2.2) Equation of state of the fluid in 'CLP' case */
      /* Read */
      class_read_double("w0_fld",pba->w0_fld);
      class_read_double("wa_fld",pba->wa_fld);
      class_read_double("cs2_fld",pba->cs2_fld);
    }
    if (pba->fluid_equation_of_state == EDE) {
      /** 8.a.2.3) Equation of state of the fluid in 'EDE' case */
      /* Read */
      class_read_double("w0_fld",pba->w0_fld);
      class_read_double("Omega_EDE",pba->Omega_EDE);
      class_read_double("cs2_fld",pba->cs2_fld);
    }
    if (pba->fluid_equation_of_state == COSCOM) {
      /** 8.a.2.3) Equation of state of the fluid in 'EDE' case */
      /* Read */
      class_read_double("w0_cos",pba->w0_cos);
      class_read_double("w1_cos",pba->w1_cos);
      class_read_double("cs2_fld",pba->cs2_fld);
    }
  }
```

Let us modify the fluid description

source/input.c

```
/** 9.a.2) Equation of state
pba->fluid_equation_of_state
pba->w0_fld = -1.;
pba->cs2_fld = 1.;
/** 9.a.2.1) 'CLP' case */
pba->wa_fld = 0.;
/** 9.a.2.2) 'EDE' case */
pba->Omega_EDE = 0.;
/** 9.a.2.2) 'COSCOM' case */
pba->w0_cos = -1.;
pba->w1_cos = 0.;
```

```
/** 5) Hubble parameter */
pba->h = 0.67810;
pba->H0 = pba->h*1.e5/_c_;

/** 6) Primordial Helium fraction */
pth->YHe = _YHE_BBN_;

/** 7) Recombination algorithm */
pth->recombination=recfast;
pth->recfast_photoion_mode=recfast_photoion_Tmat;

```

Let us modify the fluid description

python/classy.pxd

```
double Omega0_dcdm
double Omega0_ncdm_tot
double Omega0_lambda
double Omega0_fld
double w0_fld
double wa_fld
double w0_cos
double w1_cos
double cs2_fld
```

explanatory.inii

```

# 8.a.2) Choose your equation of state between
#          - 'CLP' for p/rho = w0_fld + wa_fld (
#            (Chevalier-Linder-Polarski),
#          - 'EDE' for early Dark Energy
#       (default:'fluid_equation_of_state' set t
fluid_equation_of_state = COSCOM
w0_cos = -0.9
w1_cos = 0.
```

Let us modify the scalar field description

The KG Equations for the Scalar Field

$$\ddot{\phi} + 3H\dot{\phi} + \frac{dV(\phi)}{d\phi} = 0$$

In class code the potential

$$V = V_{p_{scf}} * V_{e_{scf}}$$

$$V_e = \exp(-\lambda\phi)$$

$$V_p = (\phi - B)^\alpha + A$$

```
* Scalar field potential and its derivatives with respect to the field _scf
 * For Albrecht & Skordis model: 9908085
 * - \f$ V = V_{p_{scf}}*V_{e_{scf}} \f$
 * - \f$ V_e =  \exp(-\lambda \phi) \f$ (exponential)
 * - \f$ V_p = (\phi - B)^\alpha + A \f$ (polynomial bump)
```

Let us modify the scalar field description

The KG Equations for the Scalar Field

$$\ddot{\phi} + 3H\dot{\phi} + \frac{dV(\phi)}{d\phi} = 0$$

In class code the potential

$$V = V_{p_{scf}} * V_{e_{scf}}$$

$$V_e = \exp(-\lambda\phi)$$

$$V_p = (\phi - B)^{\alpha} + A$$

We need to find out.

$$V(\phi)$$

$$\frac{dV(\phi)}{d\phi}$$

$$\frac{d^2V(\phi)}{d\phi^2}$$

Let us modify the scalar field description

explanatory.ini

```
scf_parameters = 10.0, 0.0, 0.0, 0.0, 100.0, 0.0

# 8.b.2) Scalar field (scf) initial conditions from attractor solution (assuming
#        pure exponential potential). (default: yes)
attractor_ic_scf = yes

# 8.b.3) Scalar field (scf) shooting parameter: If Omega_scf is set (can only be negative),
#        the following index (0,1,2,...) in the list scf_parameters will be used for shooting
#        (See also the section about shooting in input.c)
#        Basically parameter number scf_tuning_index will be adjusted until
#        the correct Omega_scf is found to suffice the budget equation
scf_tuning_index = 0



# 8.b.4) Scalar field (scf) shooting parameter. With this, you can overwrite some parameter
#        of 8.b.1) depending on the index defined in 8.b.3)
scf_shooting_parameter =
```

Let us modify the scalar field description

```
# 8.b.1) Scalar field (scf) potential parameters and initial conditions
#         (scf_parameters = [scf_lambda, scf_alpha, scf_A, scf_B, phi,
#         phi_prime]). V = ((\phi-B)^\alpha + A)exp(-lambda*phi), see
#         http://arxiv.org/abs/astro-ph/9908085. If 'attractor_ic_scf' is set to
#         'no', the last two entries are assumed to be the initial values of phi
#         in units of the reduced planck mass m_Pl and the conformal time
#         derivative of phi in units of [m_Pl/Mpc]. (Note however that CLASS
#         determines the initial scale factor dynamically and the results might
#         not be as expected in some models.)
scf_parameters = 10.0, 0.0, 0.0, 0.0, 100.0, 0.0
```

$$V = V_{p_{scf}} * V_{e_{scf}}$$

explanatory.ini

$$V_e = \exp(-\lambda\phi)$$

$$V_p = (\phi - B)^\alpha + A$$

Thank you