The CLOE organization

cloelib, cloelike, playground (+ euclidlib)

Andrea Pezzotta (INAF - OAB)

on behalf of

CLOE-org maintainers

A joint effort within KP-JC1









Marco Bonici, Guadalupe Cañas Herrera, Carmelita Carbone, Pedro Carrilho, Santiago Casas, Martin Crocce, Chiara Moretti, Andrea Pezzotta, Isaac Tutusaus

A theory and likelihood code for Euclid

- Development of **Cosmology Likelihood for Observables in Euclid (CLOE)** started in 2020 thanks to the combined effort from

  IST:L (likelihood) and **IST:NL** (non-linear)

- Early 2024: frozen version used to obtain **MCMC-based forecasts** with non-linear modelling:

  - Paper 1: **Theoretical recipe** *(Cardone et al.)*
  - Paper 2: **Code implementation** *(Joudaki et al.)*
  - Paper 3: **Inference and forecasts** *(Cañas Herrera et al.)*
  - Paper 4: **Validation and performances** *(Martinelli et al.)*
  - Paper 5: **Extensions beyond standard models of theoretical probes and systematic effects** *(Goh et al.)*
  - Paper 6: **Impact of systematic uncertainties on the cosmological analysis** *(Blot et al.)*
  - **Other papers currently under development within KPs**

Restructuring of CLOE

New challenges for DR1 motivate key changes to CLOE

- **More flexible architecture** to support evolving SWG needs
- **Improved modularity** for easier debugging, maintenance, and extensions
- **Cleaner, more intuitive interface** for users and developers
- **Decoupling from specific frameworks** (e.g. **`Cobaya`**)
- **Support for external likelihoods** (e.g. SNIa, CMB, …)

The main path toward a more flexible code involves a **separation into a theory and a likelihood code**.

What is the main task of CLOE?

$$-2\log\mathcal{L} \propto [\boldsymbol{d} - \boldsymbol{t}(\boldsymbol{\theta})]^T C^{-1} [\boldsymbol{d} - \boldsymbol{t}(\boldsymbol{\theta})]$$

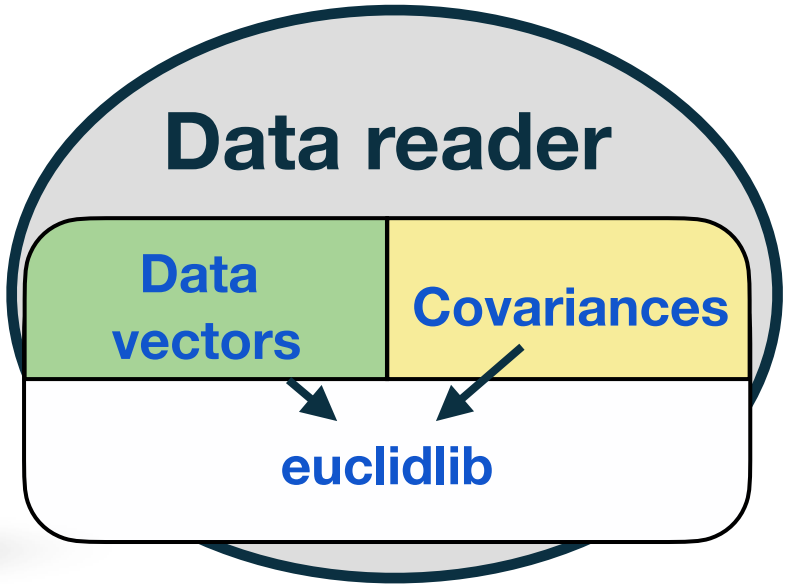
Likelihood

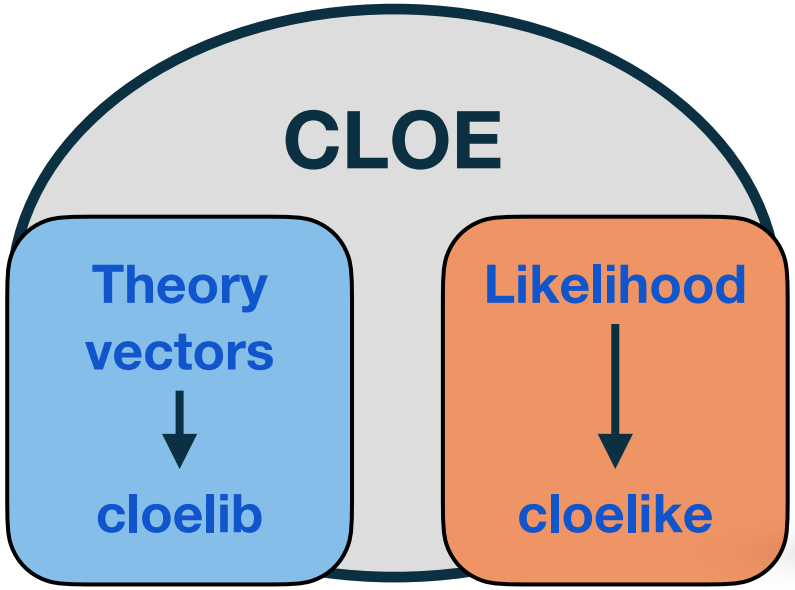
Theory vector

Covariance matrix

Data vector

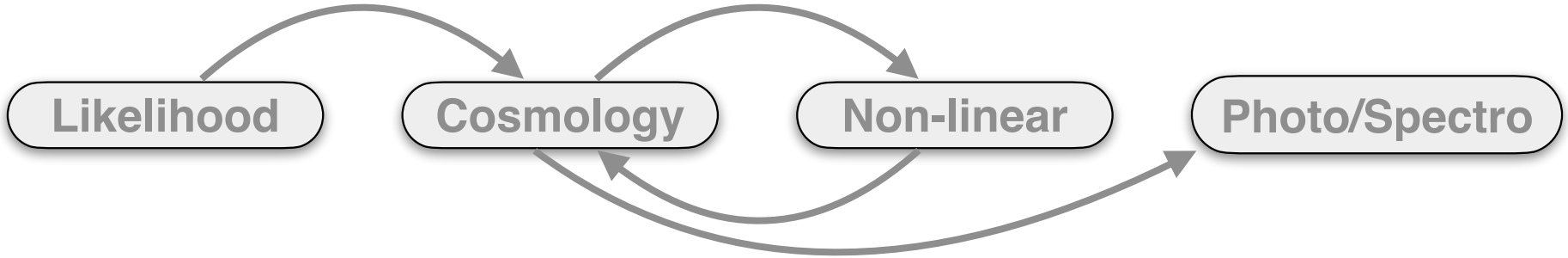
G. Cañas-Herrera
N. Tessore



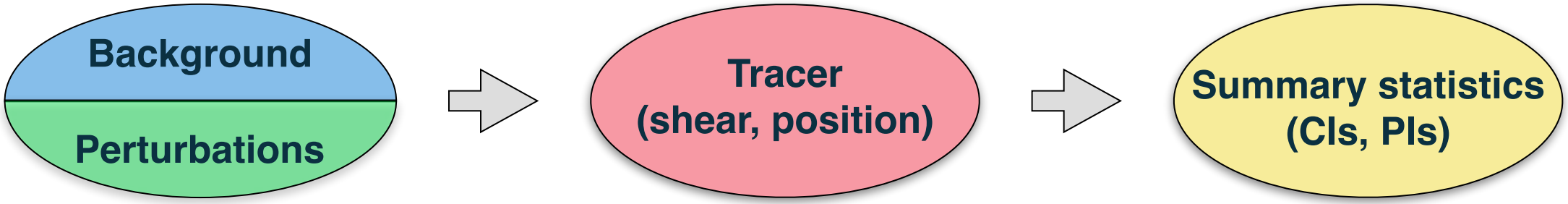


The new structure of cloelib

Old CLOE: Computation of theory model reliant on likelihood evaluation

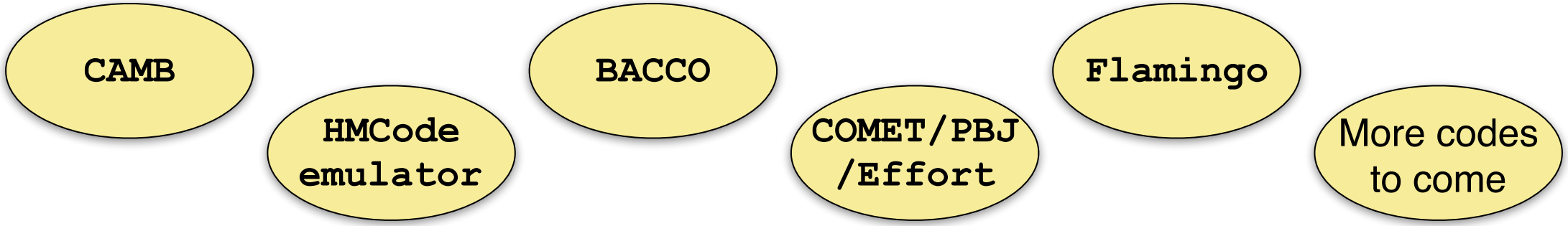


Hierarchical structure, avoiding back and forth for non-linear corrections, and dependency on likelihood



Exploiting validated external codes

Several public (or soon to be) codes available on the market



External codes used to model:

- **Background quantities** (e.g. **CAMB**)
- **Matter power spectra** (e.g. **CAMB**, **HMCode**, **BACCO**)
- **Baryonic feedback** (e.g. **HMCode**, **Flamingo**)
- **Non-linear bias and RSD** (e.g. **COMET**, **PBJ**, **Effort**, **BACCO**)
- …

What does cloelib do?

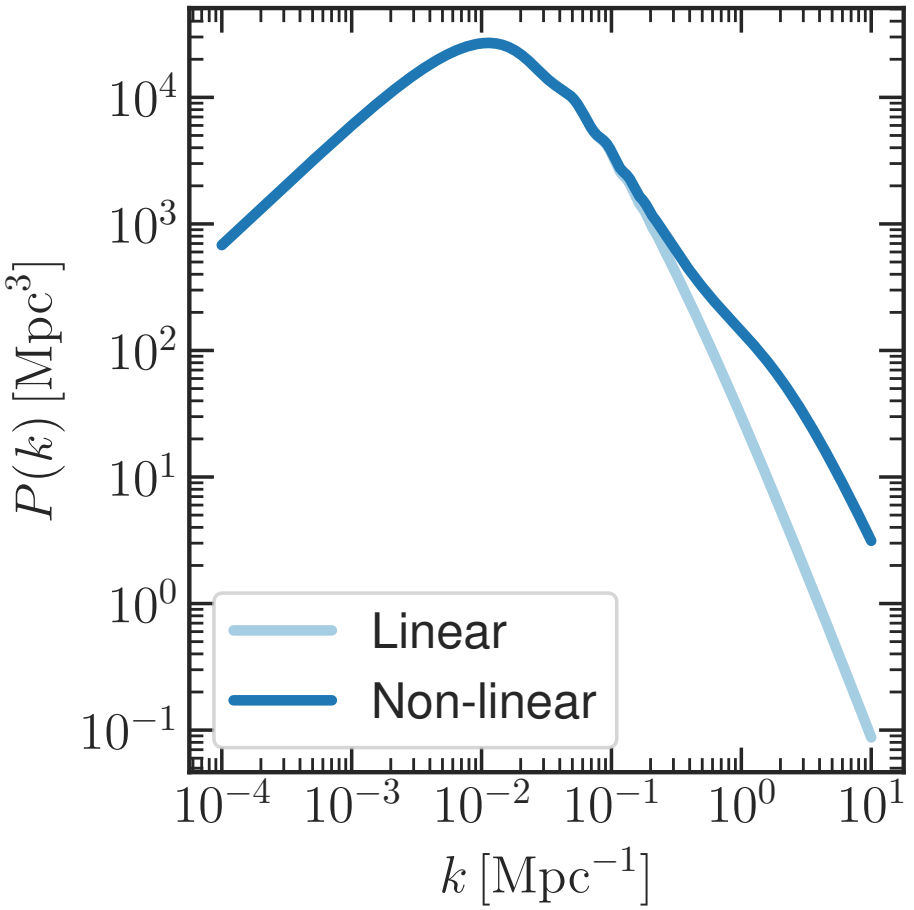
The main calculation of Euclid observables is done within **cloelib**

- **Calculation of observables**
  - **Photometric:** Harmonic space multipoles, 2PCF (through **`HMCode2020`** and **`CAMB`**)
  - **Spectroscopic:** Power spectrum multipoles with EFT and VDG_∞ models (through **`COMET`** and **`PBJ`**), BAO
- **Convolution with mixing matrix**
- **Observational systematics**
  - **Photometric:** intrinsic alignment, galaxy bias, lens magnification, multiplicative shear bias, per-bin Δz shifts
  - **Spectroscopic:** galaxy bias, RSD, redshift errors, purity (constant suppression)

How does it work?

1) Create instance of **Background & Perturbations** (multiple choices available)

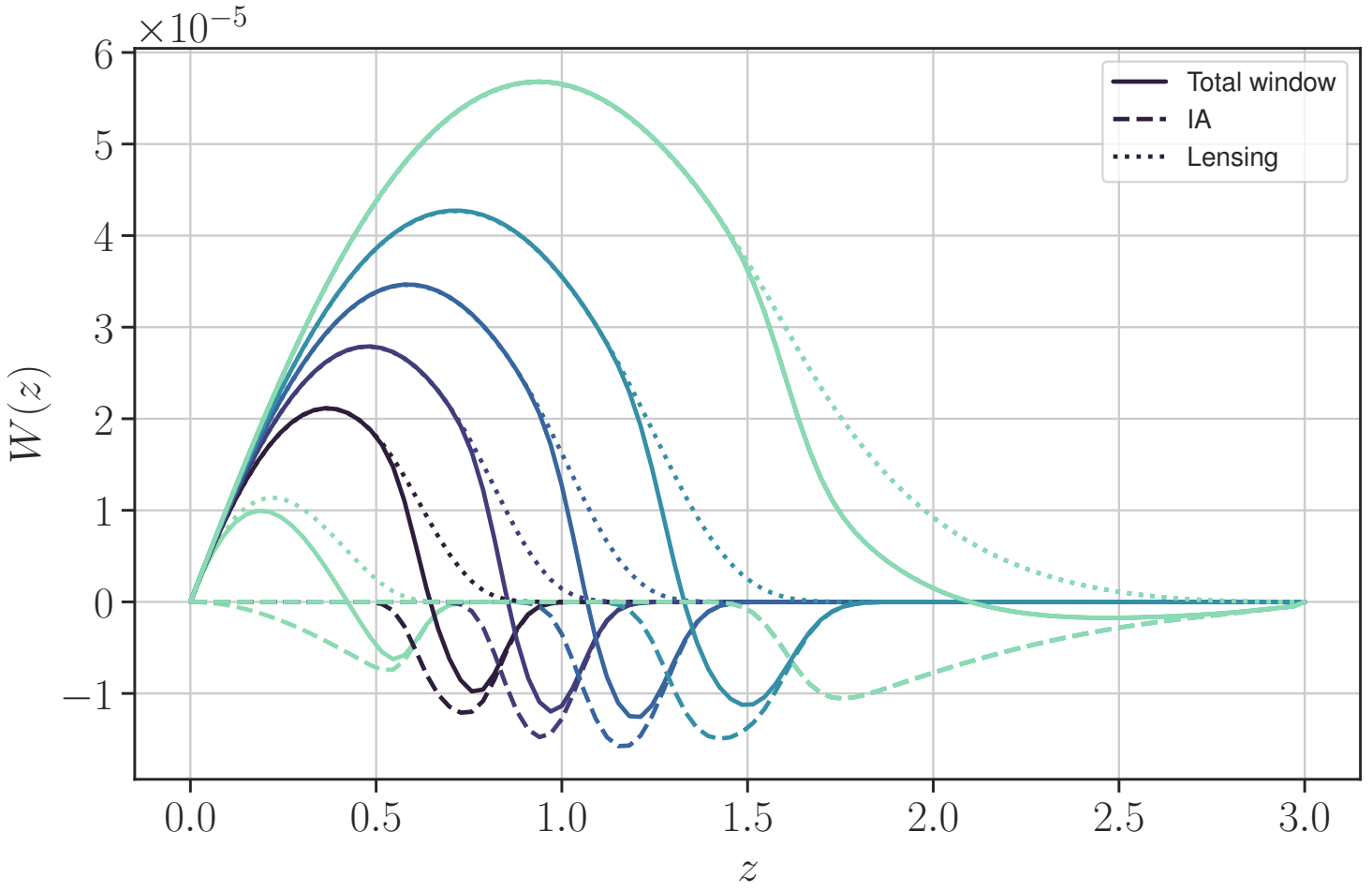
```
background = CAMBBackground(
    H0=70.0, Omega_cdm0=0.25, Omega_b0=0.05,
    w0=-1, wa=0, Omega_k0=0.0, ns=0.96,
    As=2e-9, mnu=0.06, gamma_MG=0.545)

linear_perturbations_camb = \
    CAMBLinearPerturbations(
        background, zs)
nonlinear_perturbations_camb = \
    CAMBNonLinearPerturbations(
        background, zs,
        nonlinear_model = 'mead')

pk_lin = (
    linear_perturbations_camb
    .matter_power_spectrum())
pk_nonlin = (
    nonlinear_perturbations_camb
    .matter_power_spectrum())
```



How does it work?

2) Create instance of **Tracer**

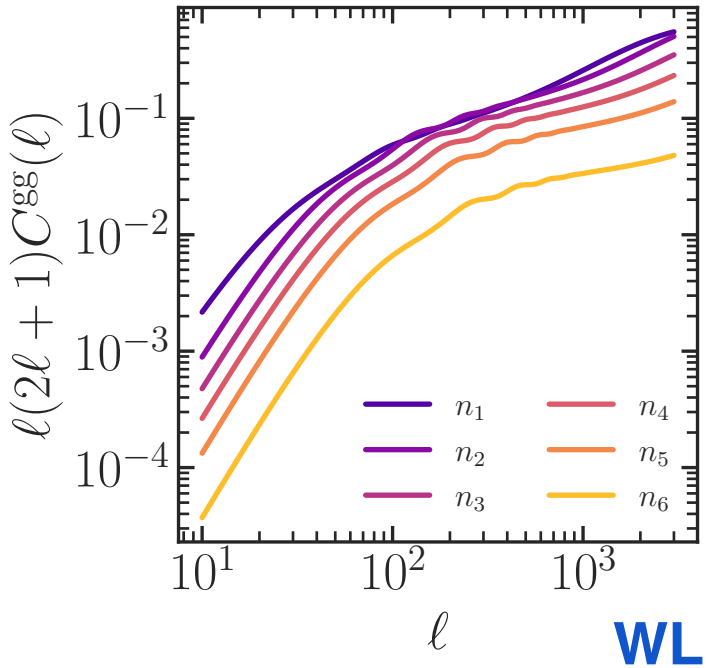
```
tracer_she = ShearTracer(
    perturbations=perturbations,
    dndz=my_dndz_she_norm, z = zs,
    nuisance_params={
        'AIA': 1.72,
        'CIA': 0.0134,
        'EtaIA':-0.41,
        'multiplicative_bias_1': 0.001,
        'multiplicative_bias_2': 0.001,
        'multiplicative_bias_3': 0.001,
        'multiplicative_bias_4': 0.001,
        'multiplicative_bias_5': 0.001,
        'multiplicative_bias_6': 0.001,
        'dz_shear_1': 0.0001,
        'dz_shear_2': 0.0001,
        'dz_shear_3': 0.0001,
        'dz_shear_4': 0.0001,
        'dz_shear_5': 0.0001,
        'dz_shear_6': 0.0001})
```



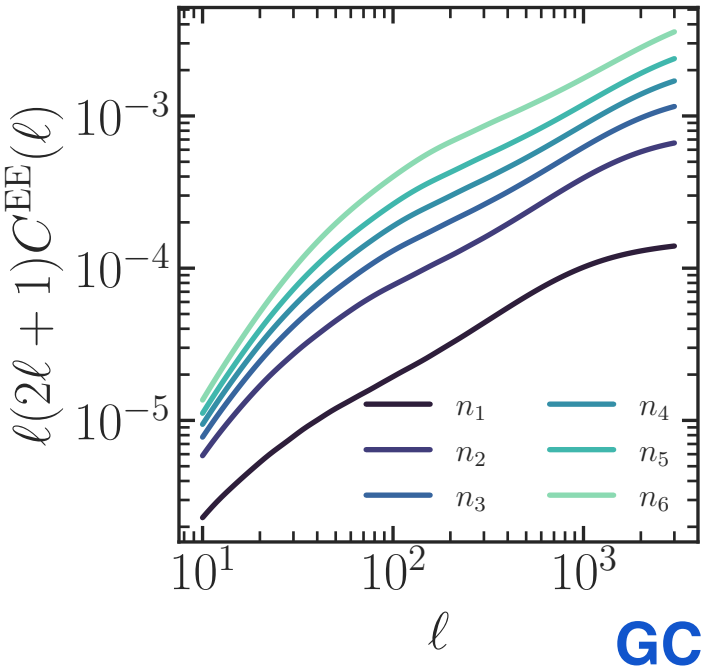
How does it work?

3) Create instance of **AngularTwoPoint**

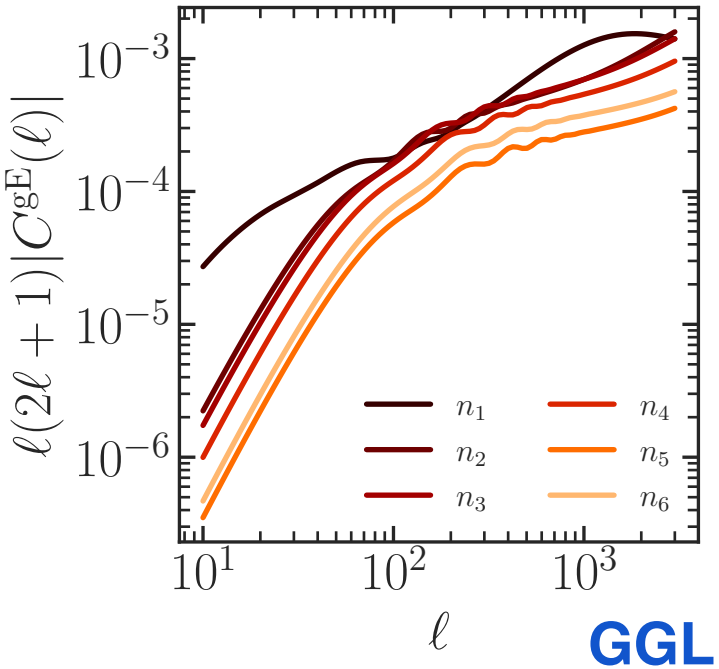
```
twopoint_pospos = AngularTwoPoint(tracer_pos, tracer_pos); cells_GG = twopoint_pospos.get_Cl(ells, 0, perturbations.k)
twopoint_shepos = AngularTwoPoint(tracer_she, tracer_pos); cells_GGL = twopoint_shepos.get_Cl(ells, 0, perturbations.k)
twopoint_sheshe = AngularTwoPoint(tracer_she, tracer_she); cells_EE = twopoint_sheshe.get_Cl(ells, 0, perturbations.k)
```



WL



GC



GGL

How does it work?

2) Create instance of **SpectroPower**

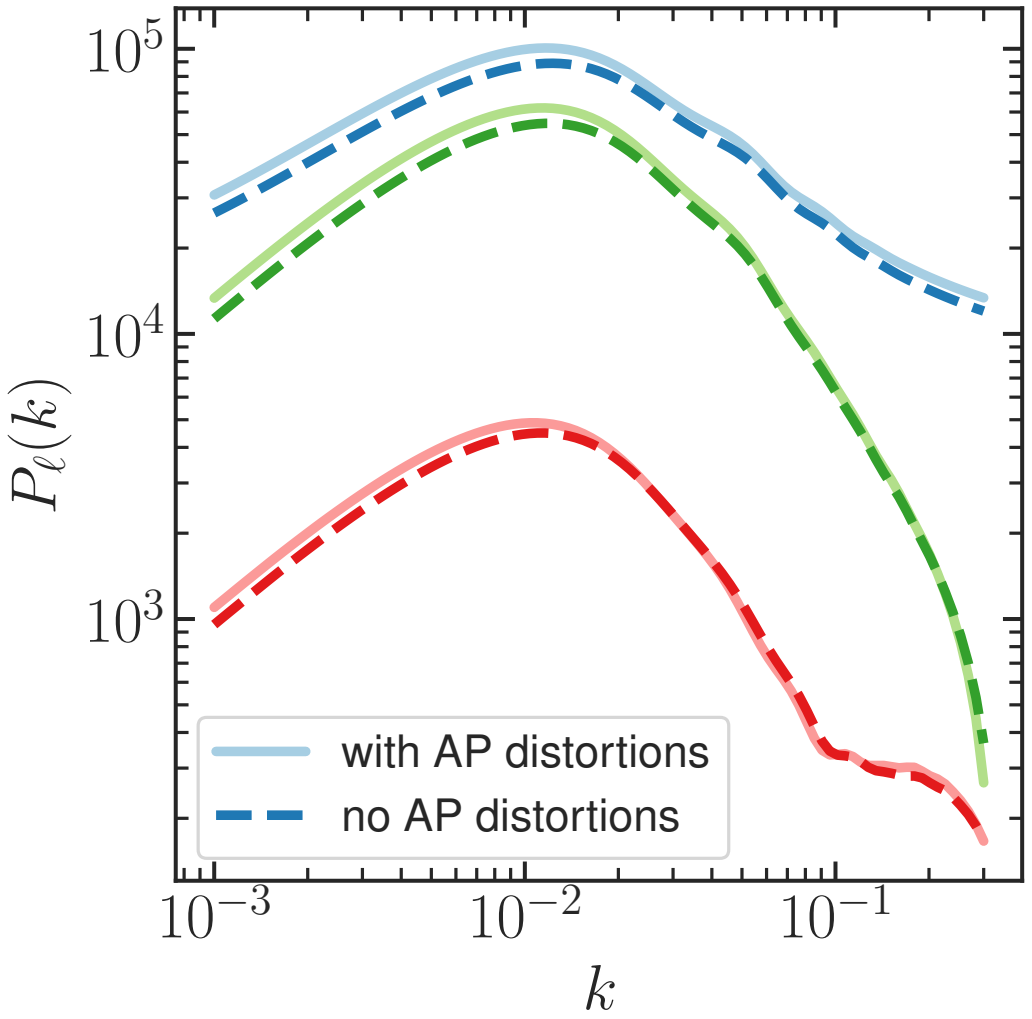
```
RSD_parameters = {
 'b1': 1.412, 'b2': 0.695,
 'bG2': -0.156, 'bGam3': 0.323,
 'c0': 30.948, 'c2': 46.233,
 'c4': 10.057, 'cnlo': 0.0
}
pkmu_comet = CometEFT_SpectroPower(
    background, RSD_parameters, redshift=zs[0])
```

$$P_{\rm gg}(k,\mu,z)$$

3) Create instance of **LegendreMultipoles**

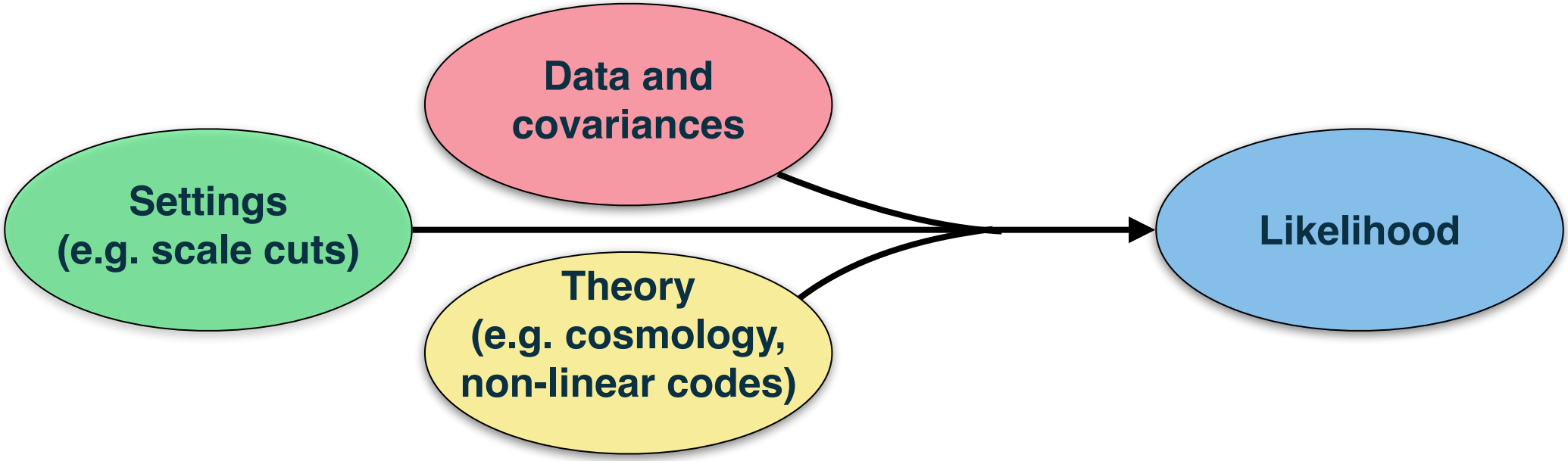
```
noise_syst_parameters = {
 'NP0': 1.056,
 'NP20': 0.0, 'NP22': 0.0,
 'fout': 0.0, 'sigmaz': 0.0
}
mps_comet = LegendreMultipoles(
    pkmu_comet, background_fiducial,
    noise_syst_parameters, nbar=1e-4)

pell_comet = mps_comet.power_multipoles(
    k=k, ells=[0,2,4])
pell_comet_noAP = mps_comet.power_multipoles(
    k=k, ells=[0,2,4], use_AP=False)
```

$$P_\ell(k,z)$$



The new structure of cloelike

Theory (from cloelib) **imported as external module**



- Easier to interface with **multiple samplers** (e.g. `Nautilus`, `Emcee`, `MH`, …)
- Plan to extend it in the future to include **automatic differentiation / gradients**

What does cloelike do?

The likelihood calculation of Euclid observables is done within **cloelike**

- **Integration** of theory code into likelihood modules
- **Different likelihood examples** for various probe combination (enabling *“create your likelihood”*): **WL**, **GCphot**, **2x2pt**, **3x2pt**, **GCspectro (FS, BAO)**, **3x2pt+GCspectro**
- **Example scripts** for **`nautilus`** sampler

Next on schedule:

- Interface with **multiple samplers** (e.g. **`nautilus`**, **`emcee`**, **`MH`**, …) and **frameworks** (e.g. **`Cobaya`**, **`CosmoSIS`**, …)
- Example scripts for **profile likelihood**

How does it work?

1) Setting up data and settings (here through euclidlib)

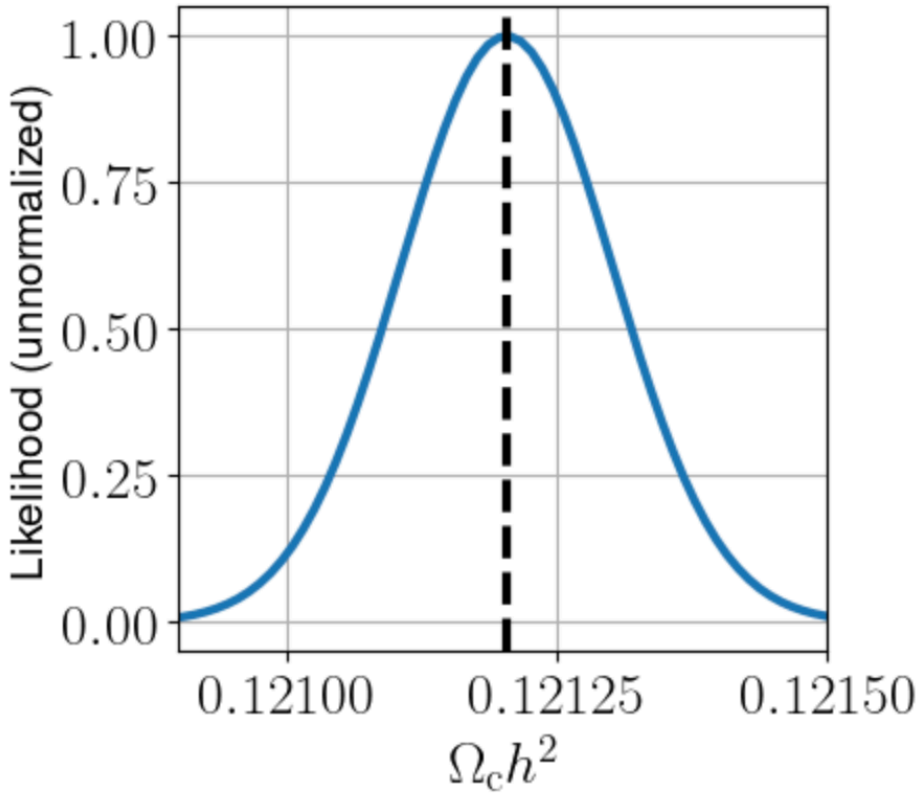
```
cells_data = el.photo.angular_power_spectra('synth_cells_5000_binned.fits')
mixmat = el.photo.mixing_matrices('mixmat_identity_5000_binned.fits')
full_cov=np.load('cov_Gauss_3x2pt_2D_probe_zpair_ell_2500deg2_ellmax5000_Bmode_copy.npy')
```

2) Create instance of desired **EuclidLikelihood**

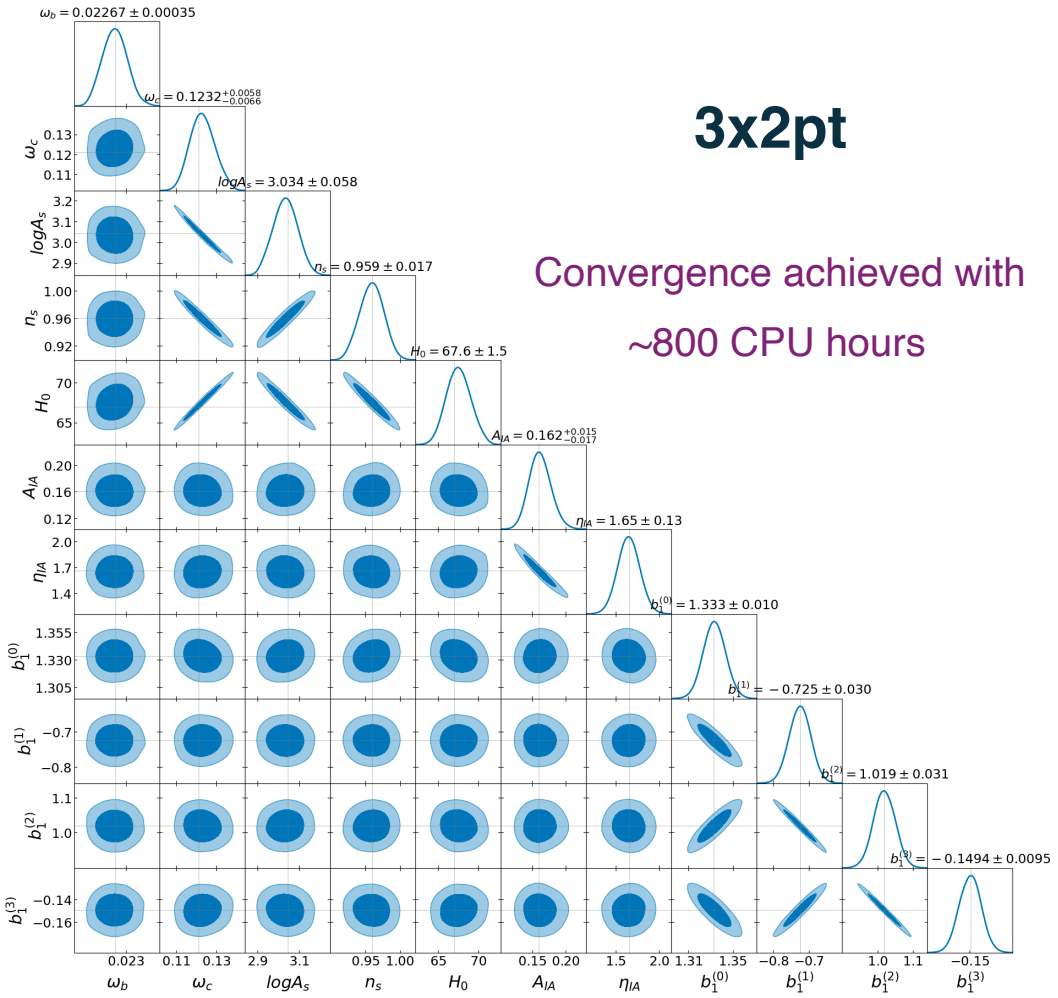
```
like_test = EuclidLikelihood_3x2ptPlusGCspectro_ClsPlusPls(
    data=data,
    settings=settings,
    Background=CAMBBackground,
    LinPerturbations=HMemuLinearPerturbations,
    NonLinPerturbations=HMemuNonLinearPerturbations,
    SpectroPower=CometEFT_SpectroPower
)
```

3) Call likelihood with set of parameters

```
result = like_test.loglike(default_pars)
```



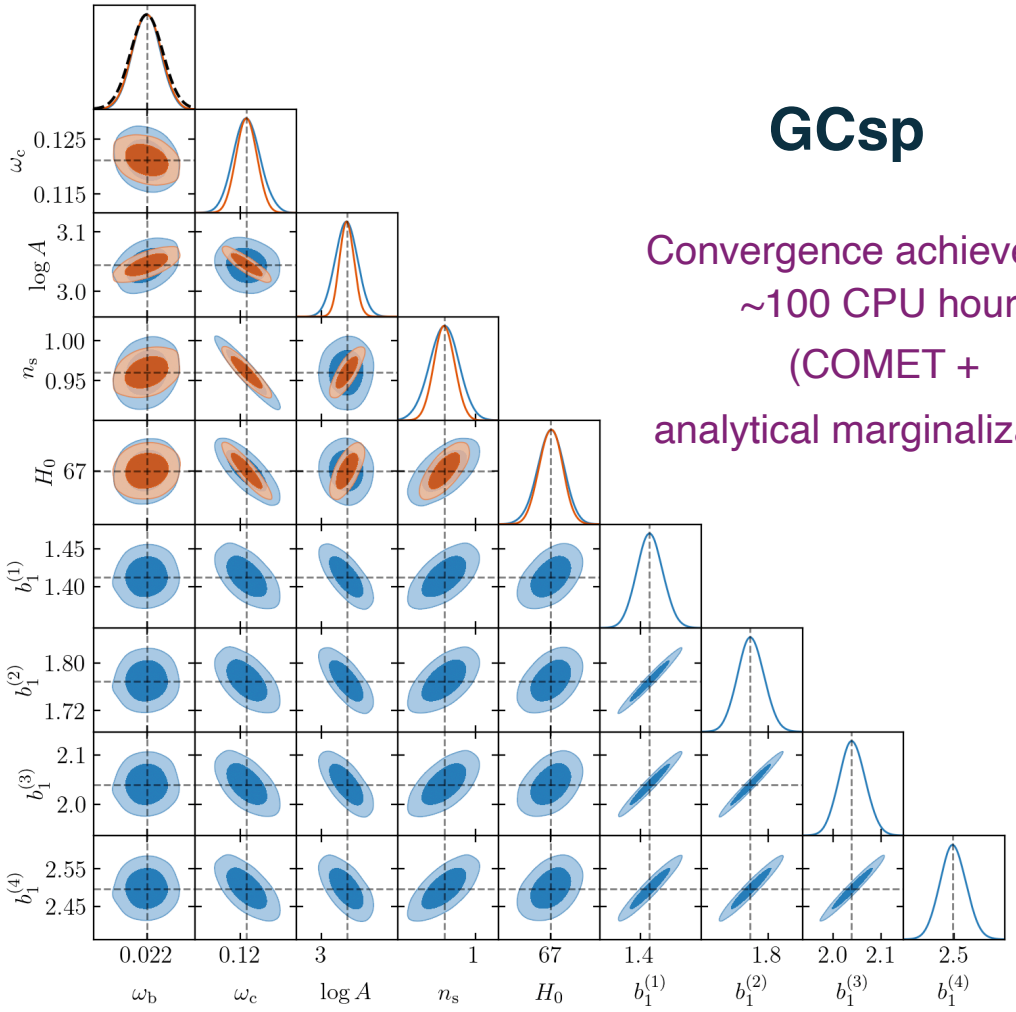
How does it work?



3x2pt

Convergence achieved with

~800 CPU hours



GCsp

Convergence achieved with

~100 CPU hours

(COMET +

analytical marginalization)

How can you test it?

All **tutorials/notebooks/scripts** for the cloe-org ecosystem are hosted in **playground**

- Tutorials for calculations of **background, photo, and spectra observables**
- Tutorials for cloelike with **various likelihood options**
- **Validation notebooks** comparing cloelib to **`CCL`**, **`CosmoSIS`**, **`COMET`**, **`PBJ`**
- Scripts for **sampling with cloelike** (3x2pt and GCspectro available)

Releases (and where to find them)

Clone the repositories and

```
> pip install .
```

- **cloelib v0.8** (see github.com/cloe-org/cloelib)
- **cloelike v0.1** (see github.com/cloe-org/cloelike)
- Include all scientific requirements from previous slides plus
  - **Unit tests** and continuous integration
  - **Contributing guidelines**
- Announced on slack channels (**#dr1-kp-jc1** and **#cloe-org**)
- Feedback from users is essential

Contributing

Contributing to cloelib

Extract from github.com/cloe-org

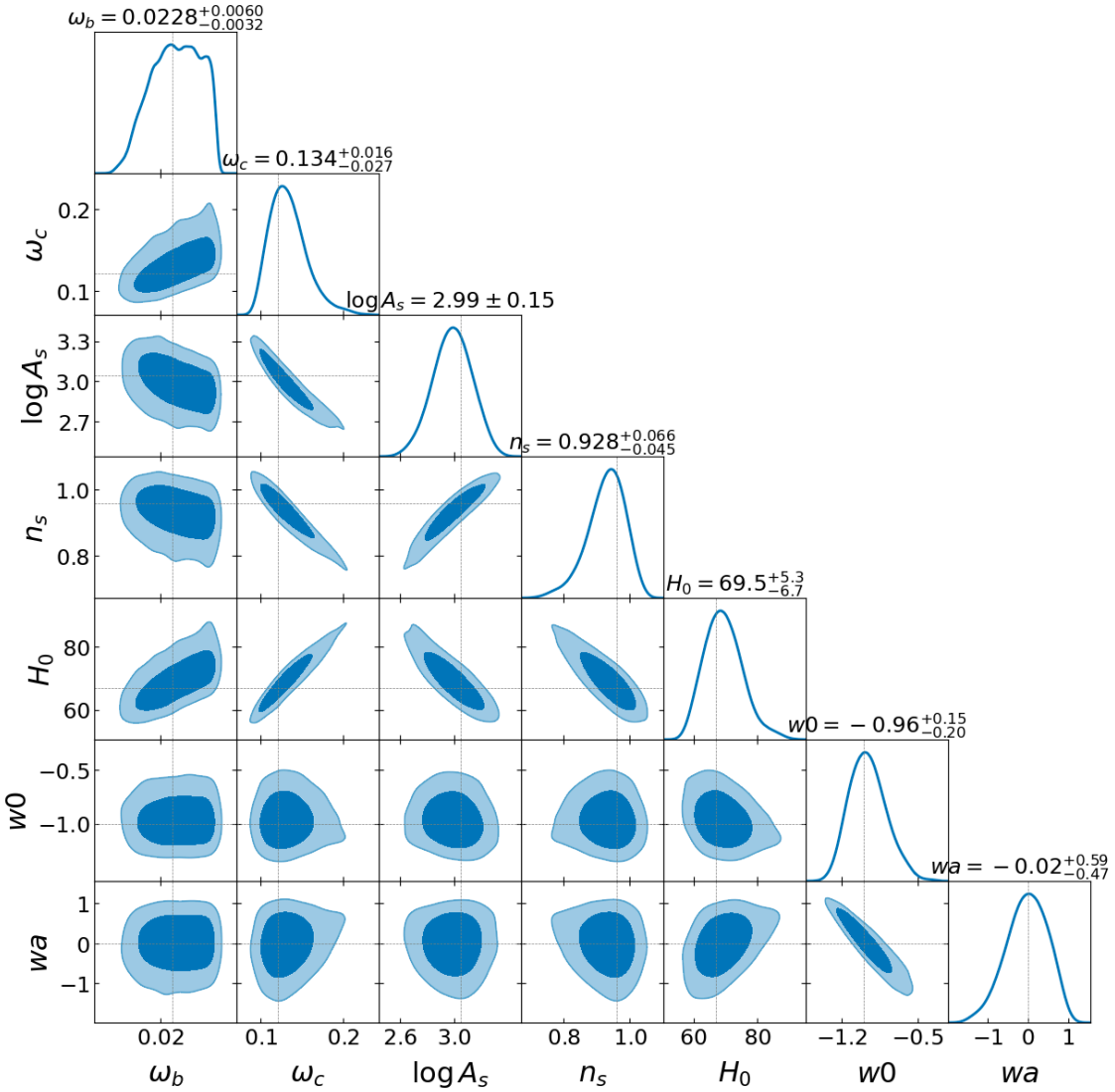
Please read the overall contribution guidelines applicable to cloe-org before proceeding.

Specific Guidelines for cloelib

1. **For Euclid Consortium Members**: Before contributing, discuss your proposal with the KP-JC-1 coordinators to ensure smooth coordination with all involved parties.
2. **Check Ongoing Tasks in the Issue Board**: Verify that your contribution does not overlap with existing tasks in the cloelib repository.
3. **Open an Issue Following the Issue Template**: Tag @cloe-org/cloe-maintainers in your issue to discuss the proposed implementation. This helps ensure a smooth integration with the software architecture and overall repository maintenance.
4. **Open a Pull Request Following the PR Template**: Fill in all the required information in the Pull Request template to facilitate review and approval.

Maintenance of cloelib

@cloe-org/cloe-maintainers reserve the right to contact previous contributors for future updates or further integrations within the software. If a feature becomes unsustainable for unforeseen reasons, @cloe-org/cloe-maintainers will consider it deprecated.


$\omega_b = 0.0228^{+0.0060}_{-0.0032}$
$\omega_c = 0.134^{+0.016}_{-0.027}$
$\log A_s = 2.99 \pm 0.15$
$n_s = 0.928^{+0.066}_{-0.045}$
$H_0 = 69.5^{+5.3}_{-6.7}$
$w0 = -0.96^{+0.15}_{-0.20}$
$wa = -0.02^{+0.59}_{-0.47}$
ω_b
ω_c
$\log A_s$
n_s
H_0
$w0$
wa