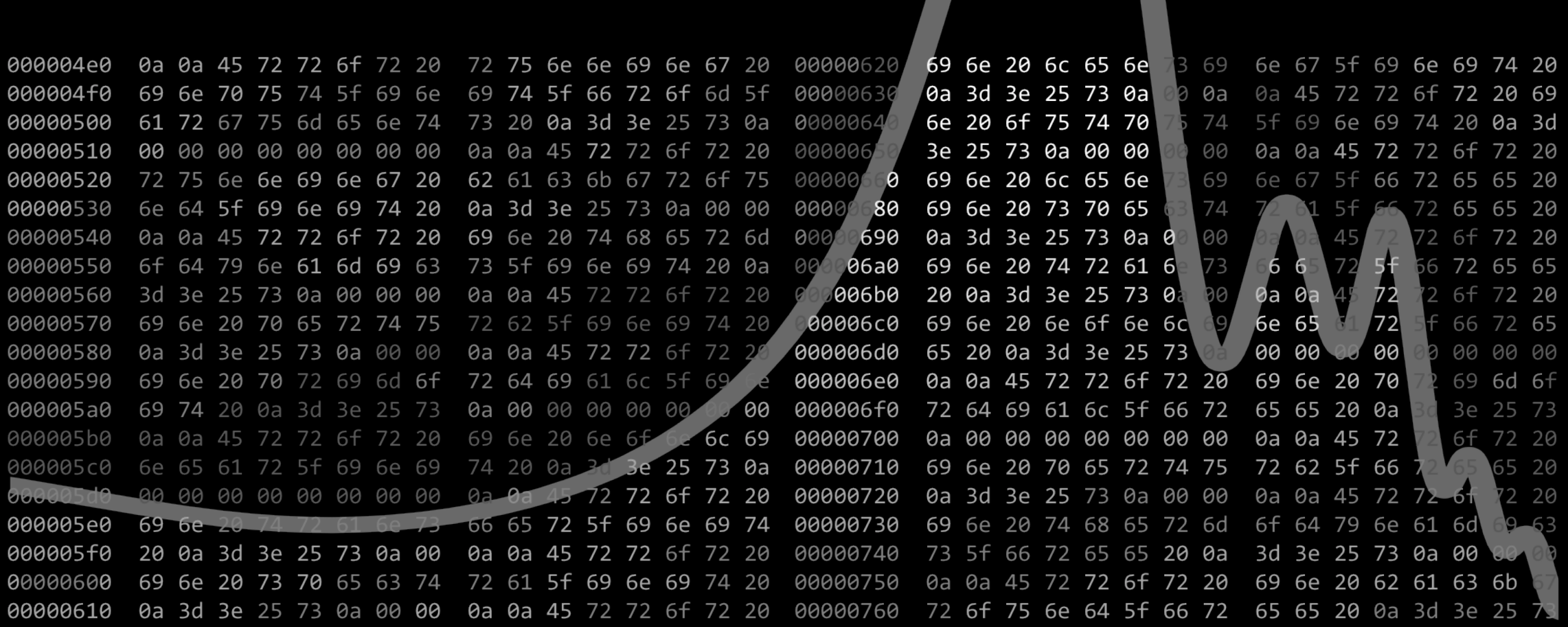
class.c



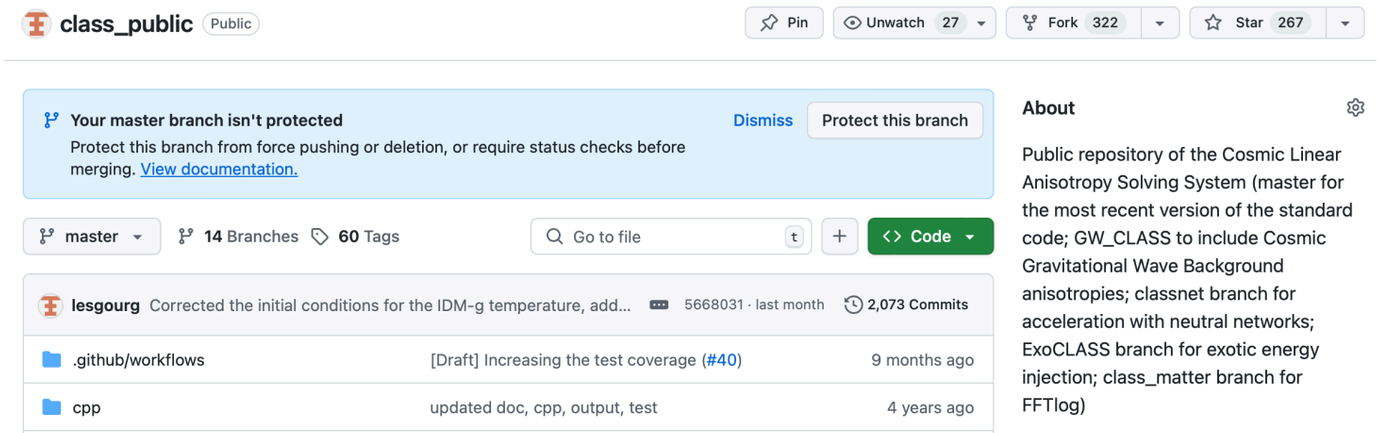
JL, Thomas Tram, Nils Schöneberg (+community)

GGI 2025

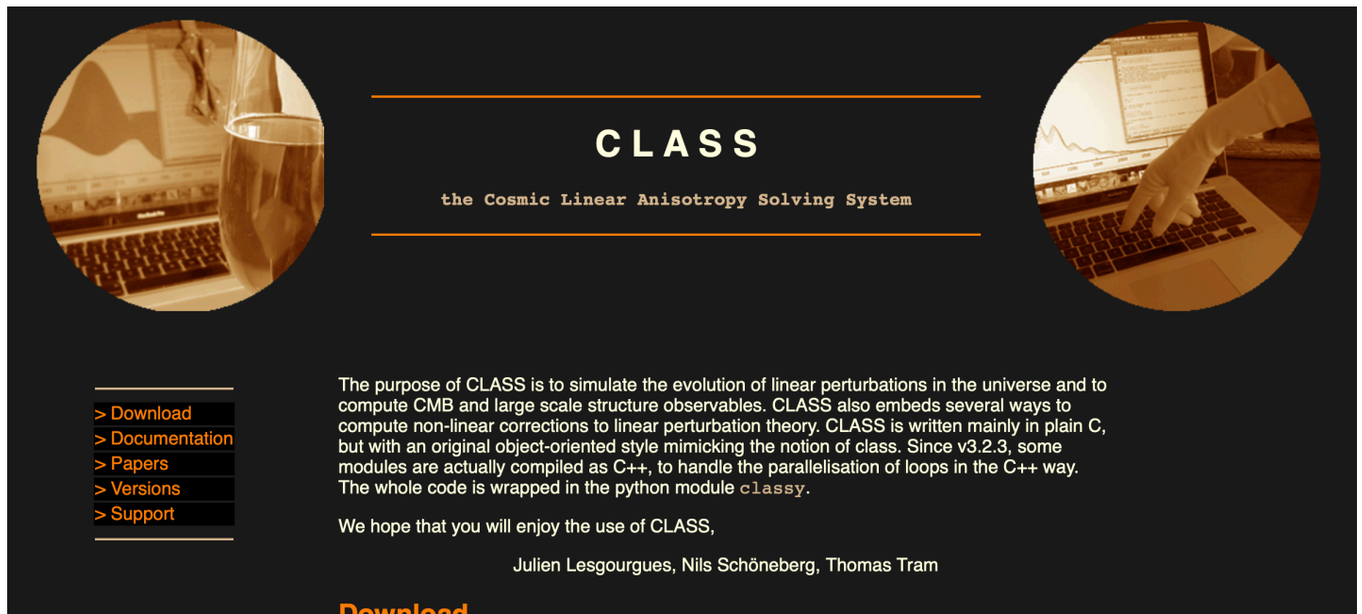
Where they code lives

- CLASS defined as a python module on PiPY repository

- https://github.com/lesgourg/class_public/ contains master branch (and other public branches)



- http://class-code.net/ contains latest class_public-x.y.z.tar.gz + info, links to docs, wiki, articles

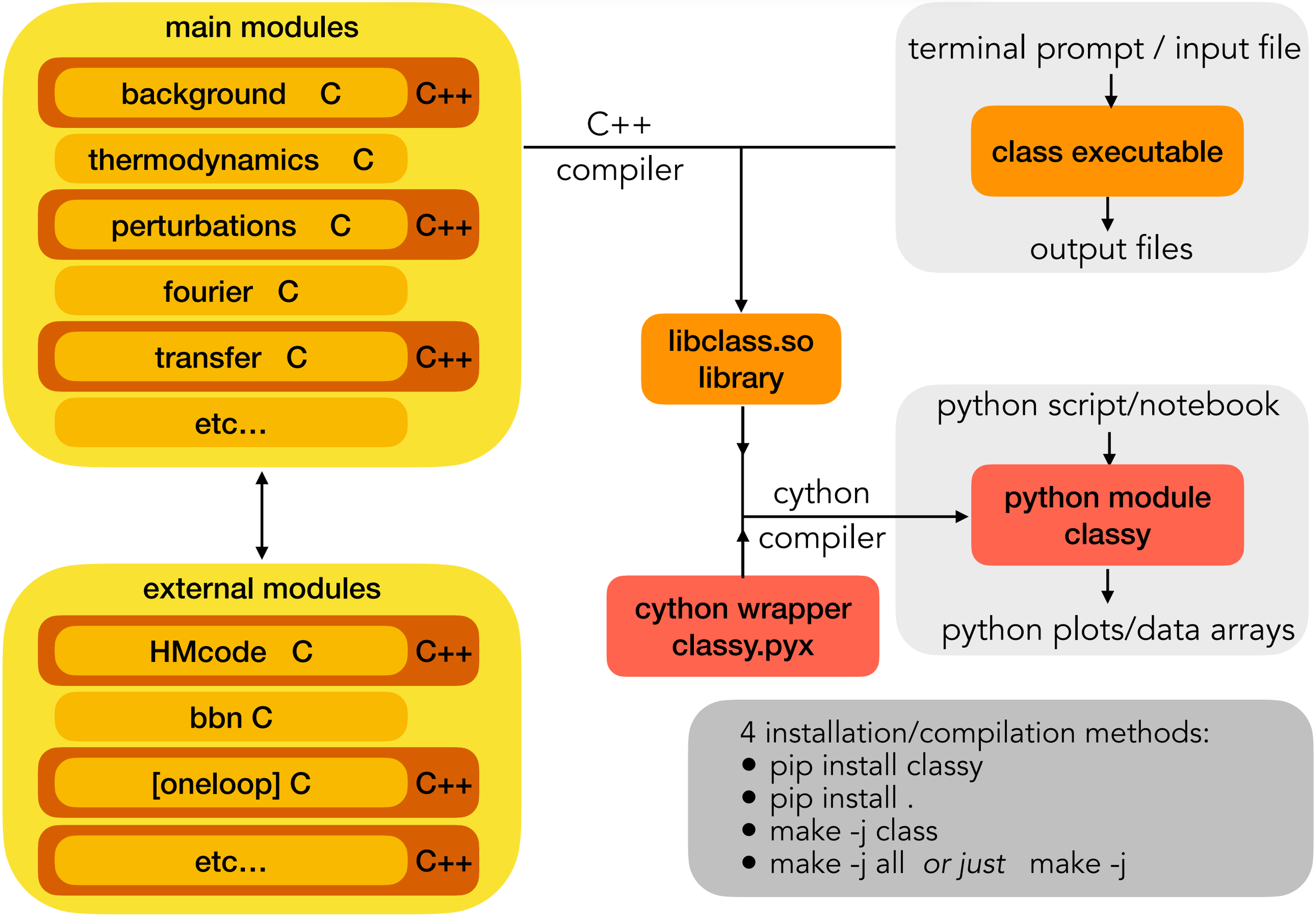


Where they code is documented

`class` is documented in various places:

- The file `explanatory.ini` lists all input parameters. (downloadable from GitHub page if `classy` installed via pip)
- **NEW!** Documentation of the `classy` wrapper at `https://class-code.readthedocs.io`.
- **NEW!** `class` LLM agent at `https://classclapp.streamlit.app/`.
- **Updated!** Python docstrings in the `classy` wrapper.
- `class` manual in repository
- Online documentation at `https://github.com/lesgourg/class_public/wiki`
- Old course notes linked on `http://class-code.net` and `https://schoeneberg.github.io/`

Overall architecture of CLASS



4 installation/compilation methods:

- pip install classy
- pip install .
- make -j class
- make -j all *or just* make -j

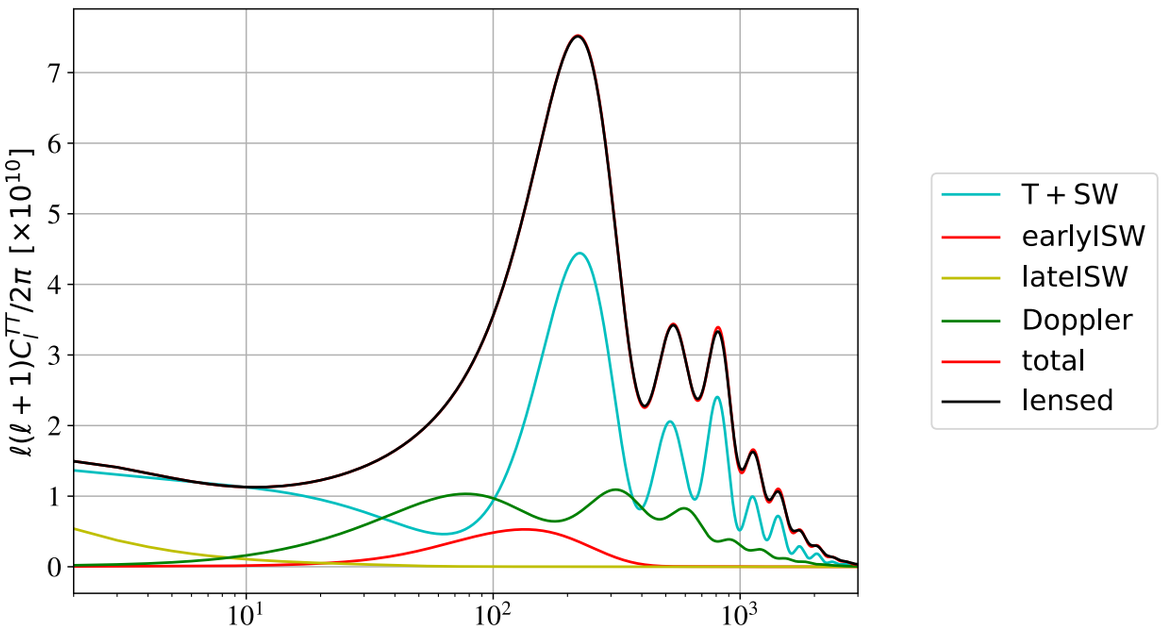
Basic runs

- Run with one of :
  - ./class default.ini
  - python scripts/warmup.py
  - jupyter notebook notebooks/warmup.ipynb
- Syntax for all input parameters detailed in explanatory.ini
- Input file: omega_b = 0.02238

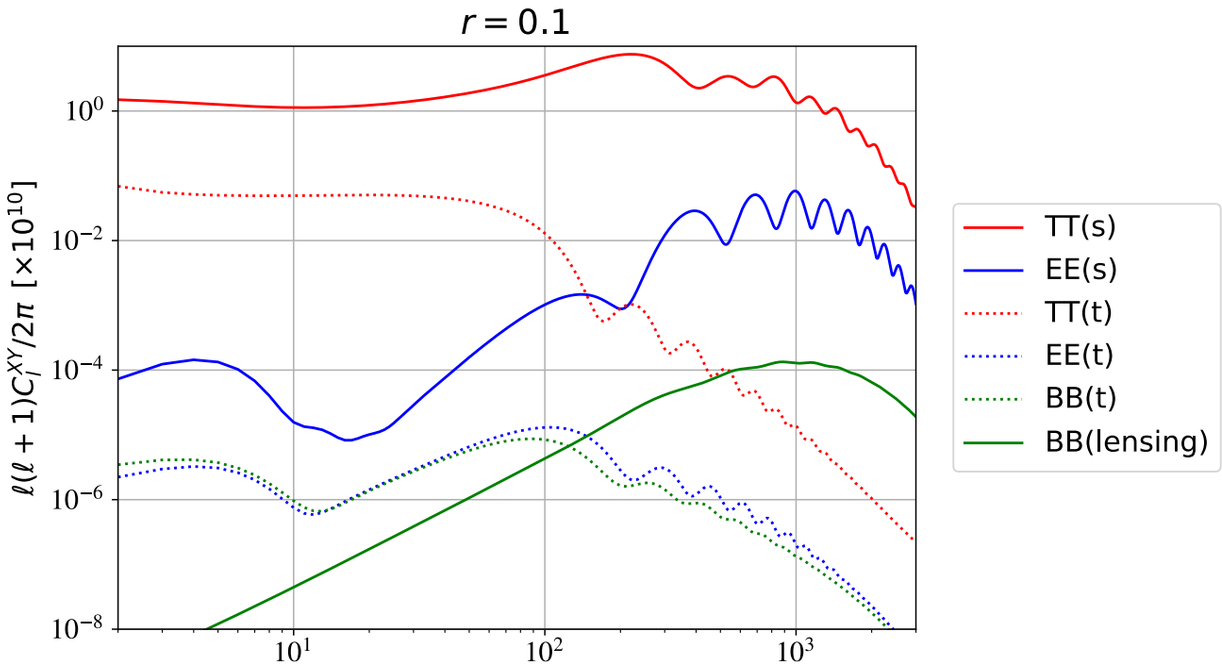
  output = tCl, pCl
- python: xxx.set({'omega_b':'0.02238', 'output':'tCl, pCl'})
- demo with notebooks/warmup.ipynb

Example of output from other notebooks in notebooks/

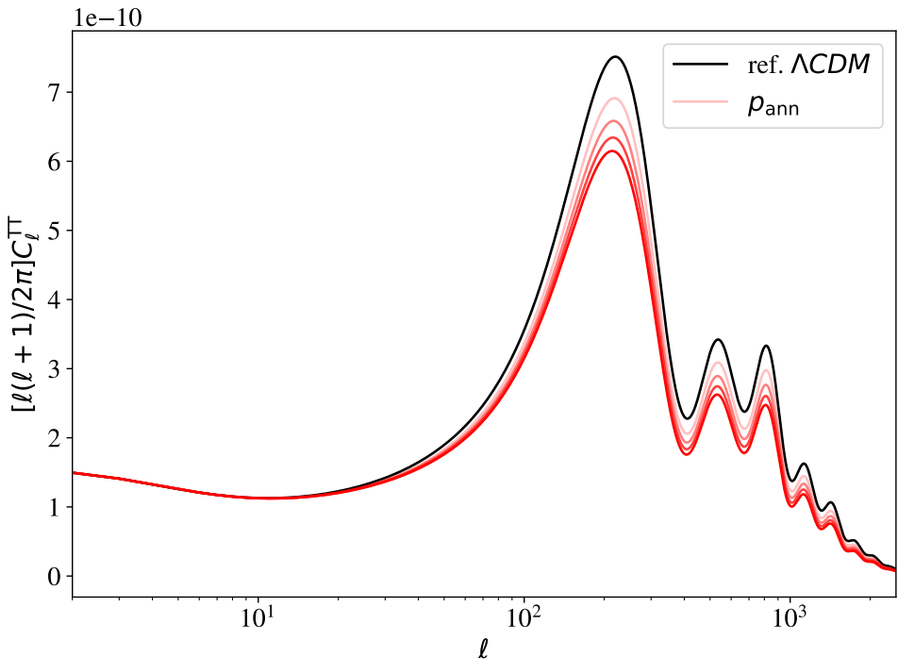
cltt_terms



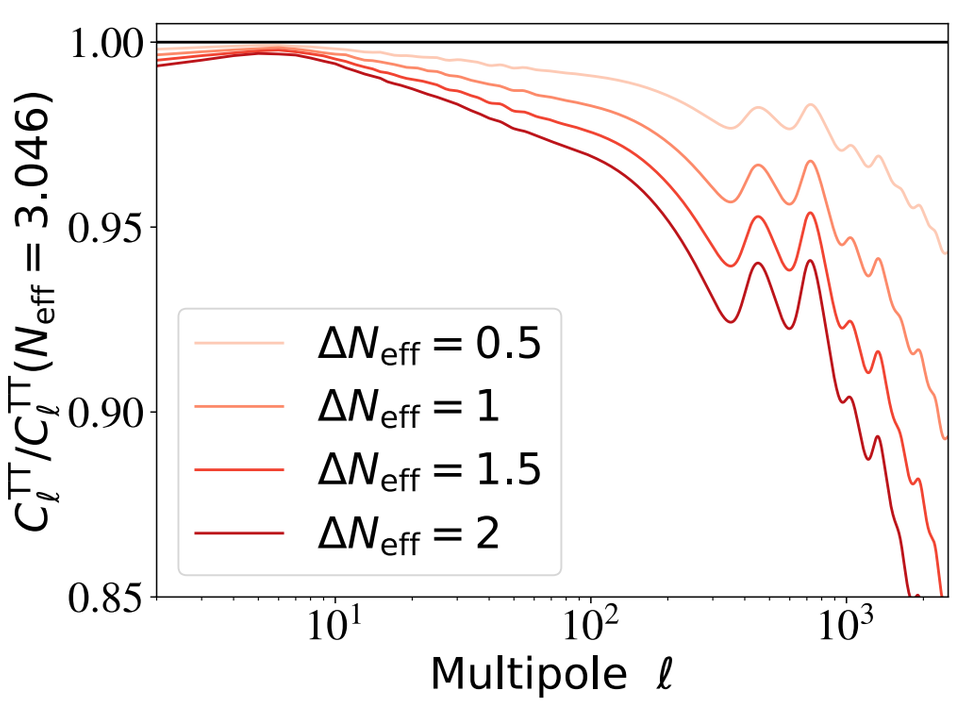
cl_ST



varying_pann

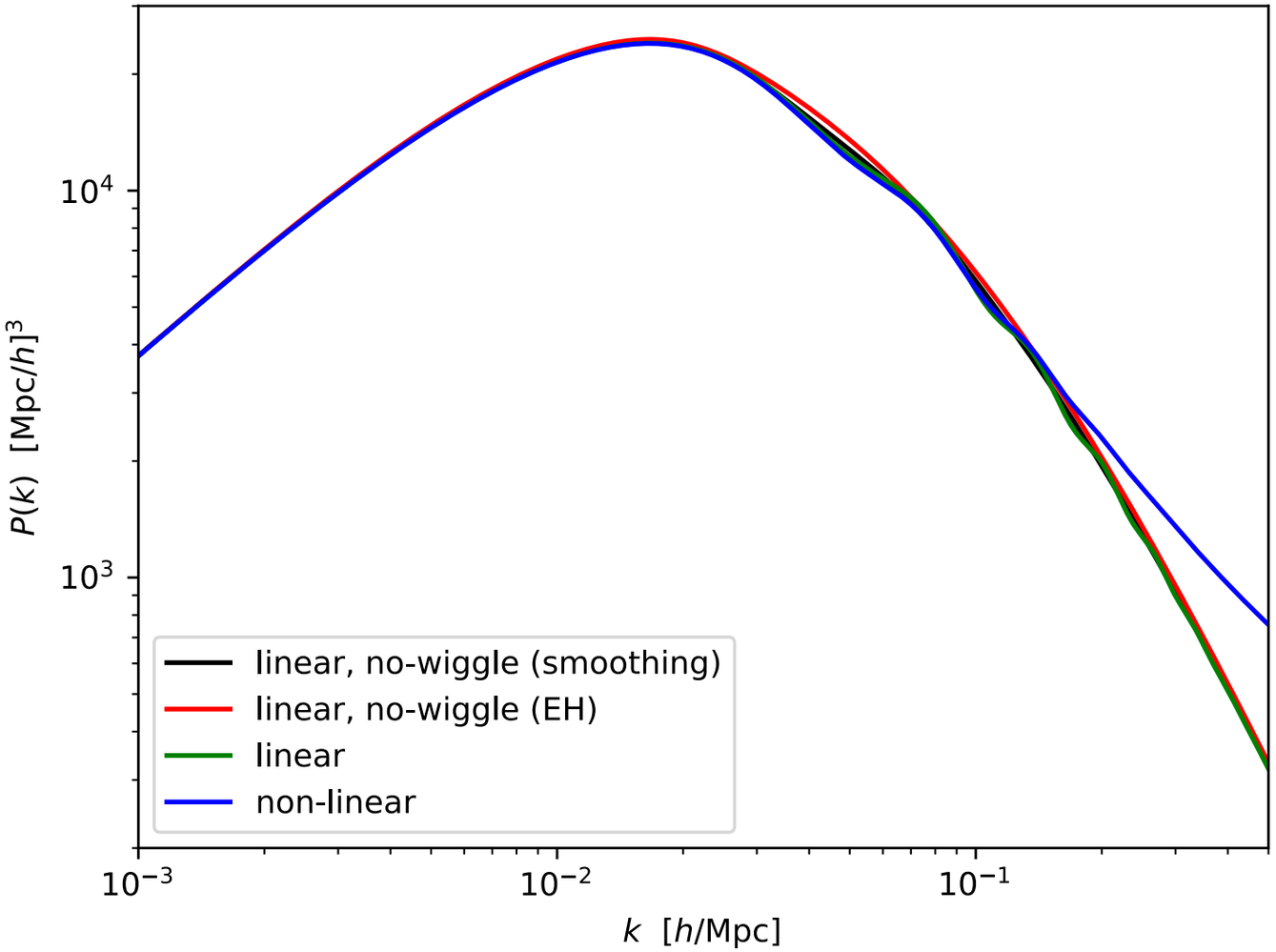


varying_neff

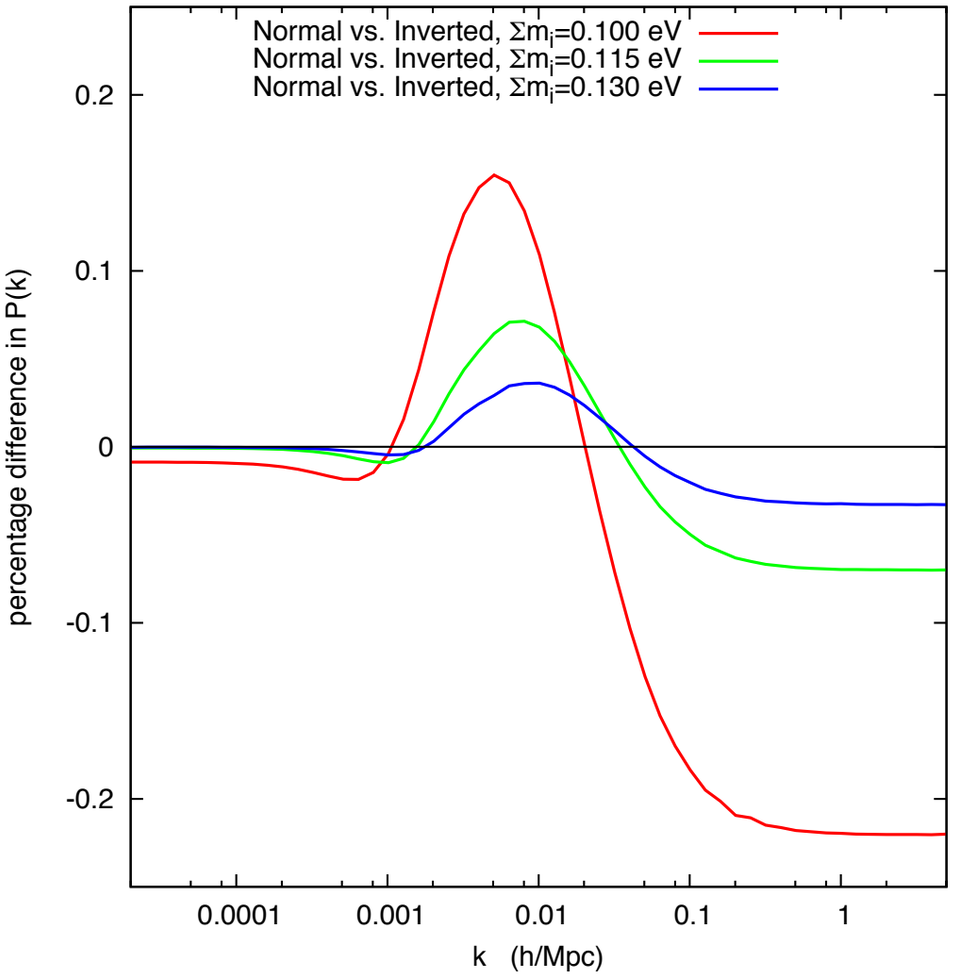


Example of output from other notebooks in notebooks/

test_hmcode

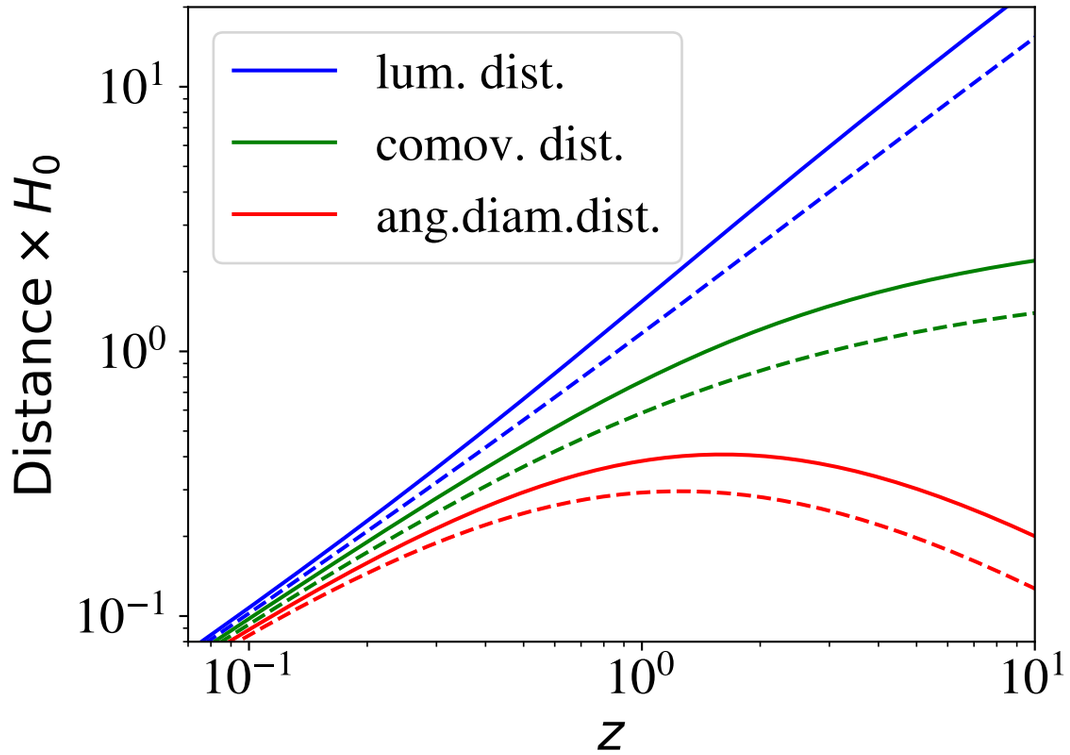


neutrino_hierarchy

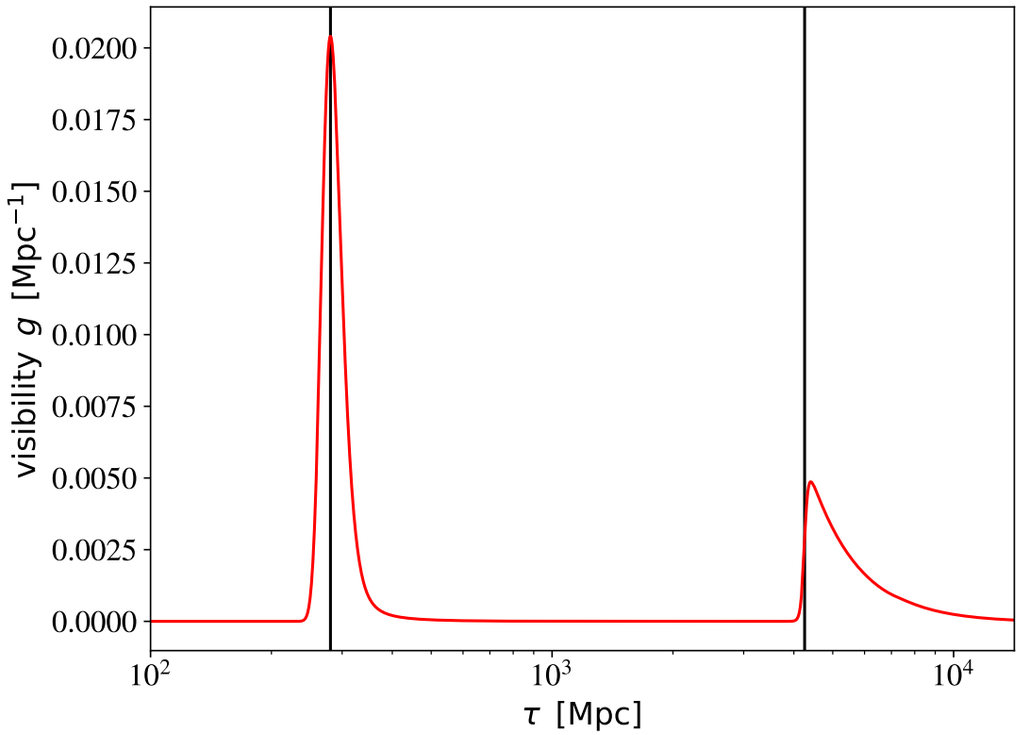


Example of output from other notebooks in notebooks/

distances

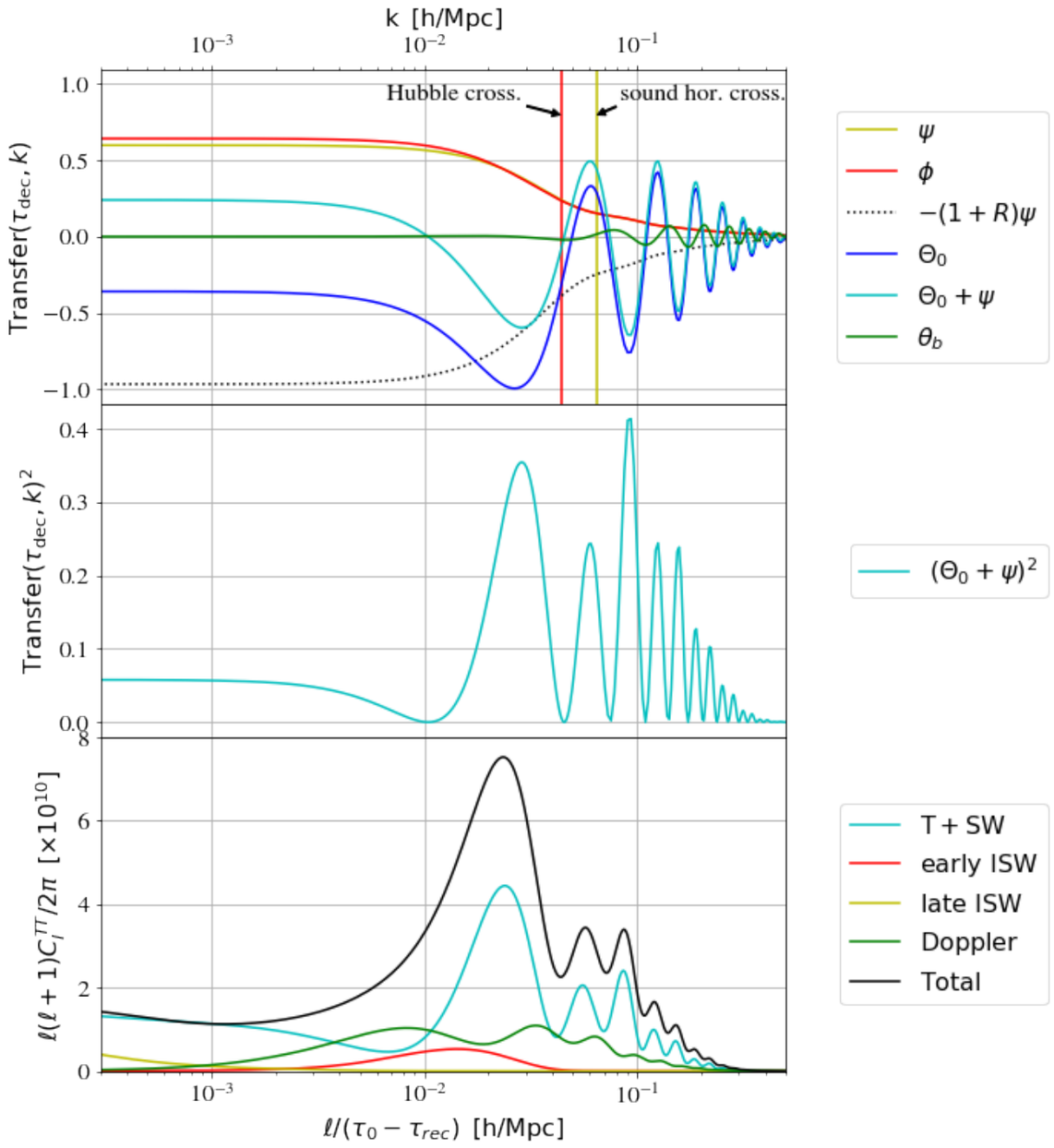


thermo

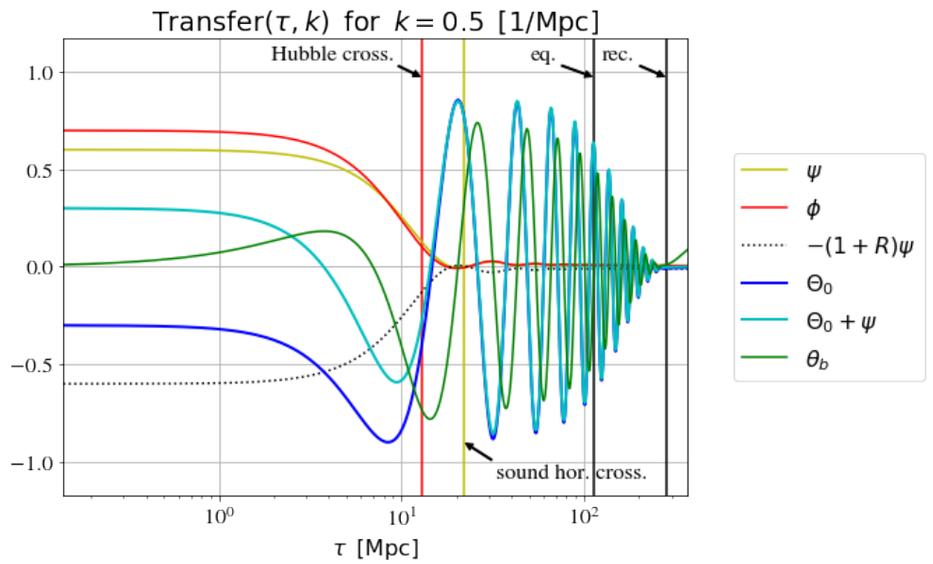


Example of output from other notebooks in notebooks/

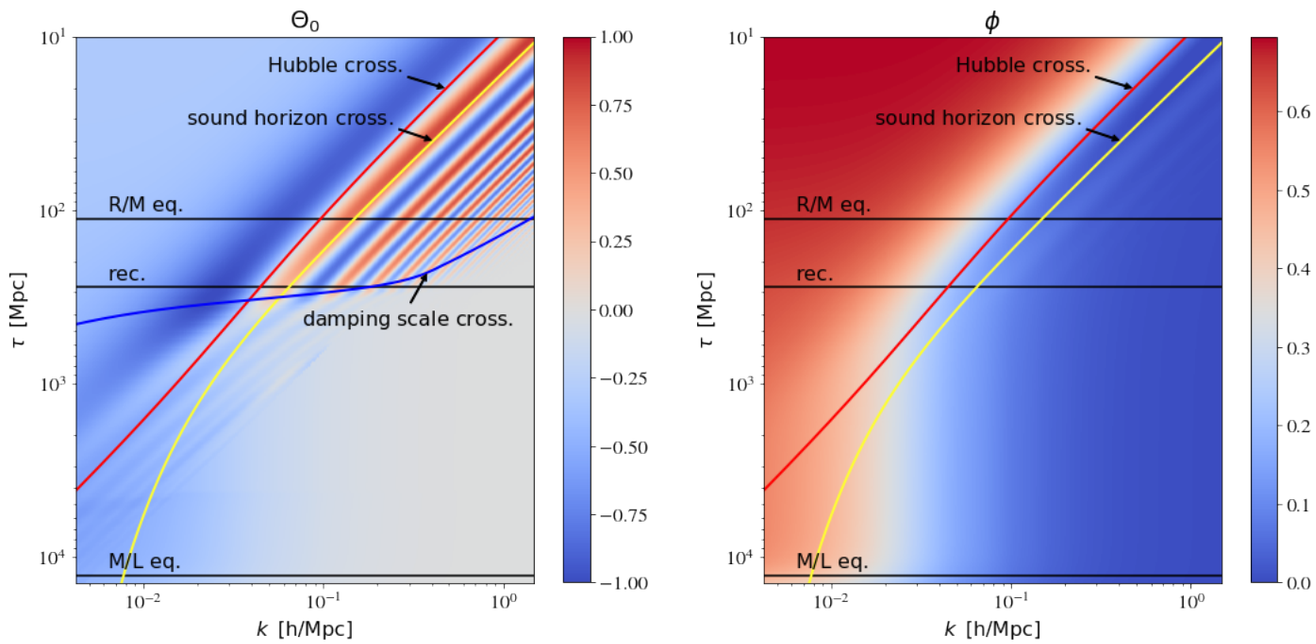
one_time



one_k



many_k



Content of the modules

Main modules

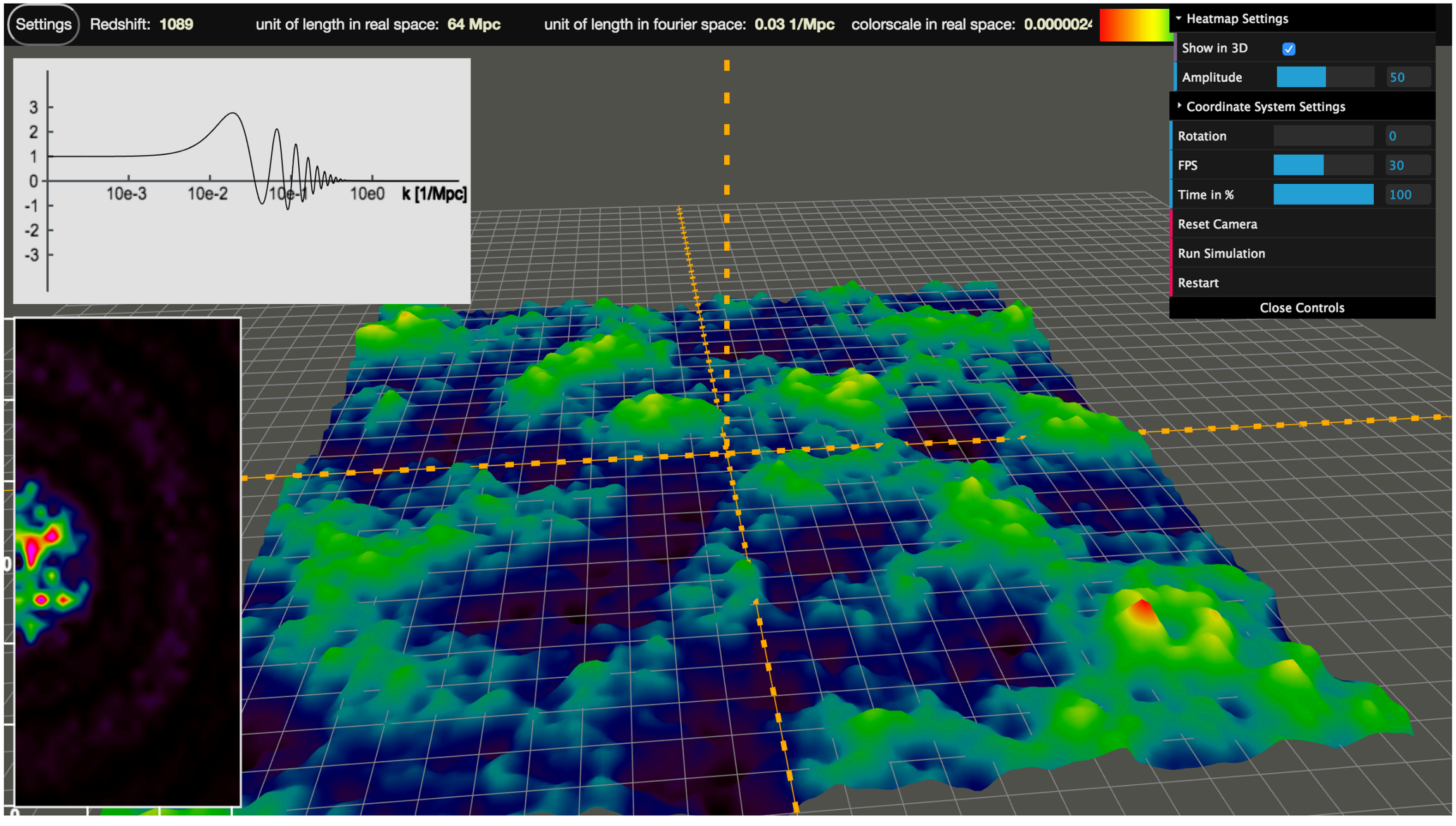
1. input
2. background — e.g. $d_{\rm a}(z)$
3. thermodyamics — e.g. $g(\tau)$
4. perturbations — e.g. $\delta_{\rm c}(\tau, k)$
5. primordial — e.g. $\mathscr{P}_{\mathscr{R}}(k)$
6. fourier — e.g. $P_{\rm m}(z, k)$
7. transfer — e.g. $\Delta_\ell(\tau, k)$
8. harmonic — e.g. $C_\ell^{\rm TT,unlensed}$
9. lensing — e.g. $C_\ell^{\rm TT,lensed}$
10. distortions — e.g. $y, \mu, \ldots$

External modules

(codes/data)

- bbn
- HyRec2020
- RecfastCLASS
- Halofit
- HMcode
- heating
- distorsions
- external_Pk
- RealSpaceInterface

RealSpaceInterface



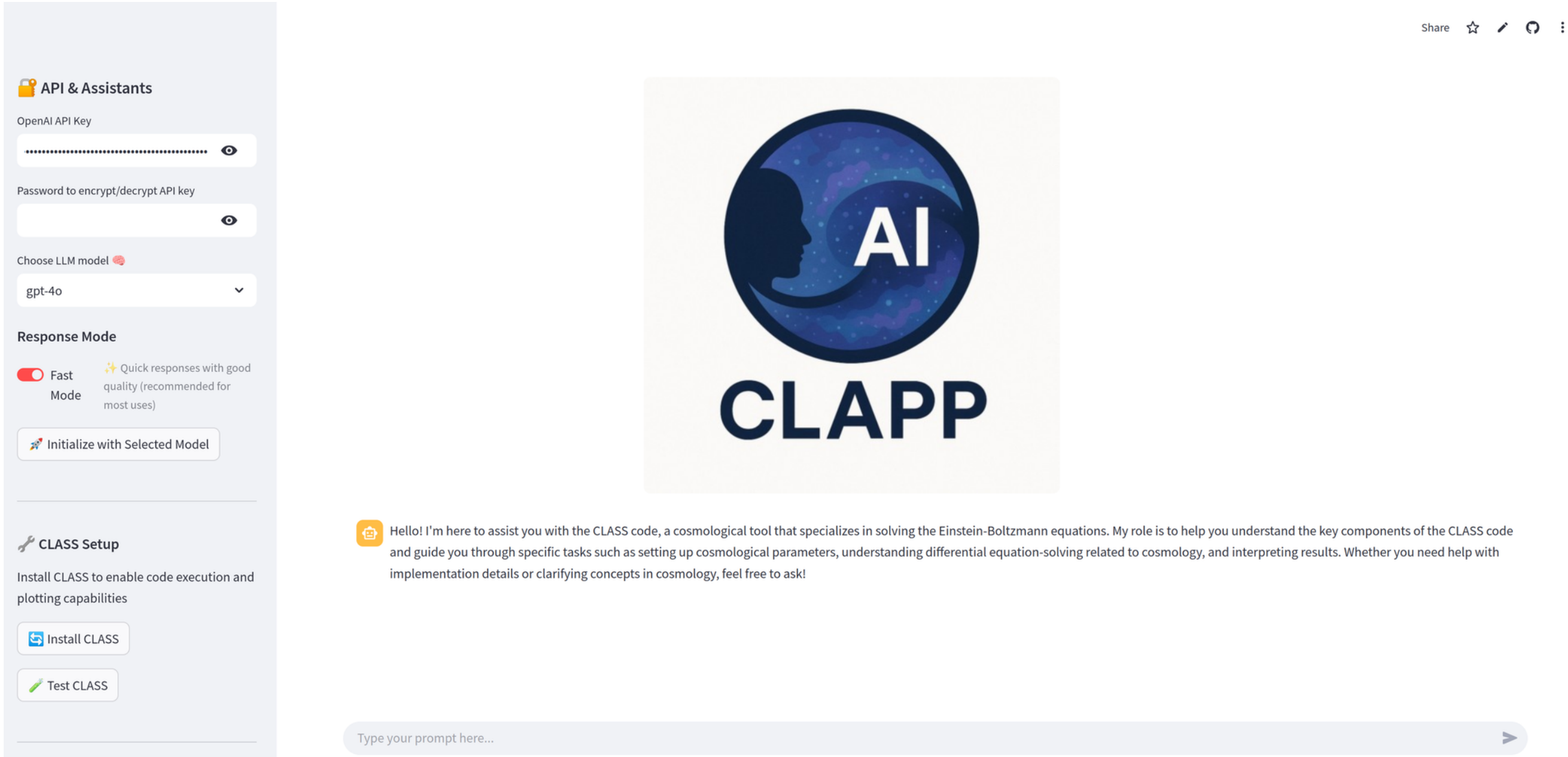
CLAPP

- described in arXiv:2508.05728

CLAPP: The CLASS LLM Agent for Pair Programming

S. Casas,[1,2] C. Fidler,[1] B. Bolliet,[3,4] F. Villaescusa-Navarro,[5,6] and J. Lesgourgues[1]

https://classclapp.streamlit.app/ or https://github.com/santiagocasas/clapp



CLAPP

- described in arXiv:2508.05728

CLAPP: The CLASS LLM Agent for Pair Programming

S. Casas,[1,2] C. Fidler,[1] B. Bolliet,[3,4] F. Villaescusa-Navarro,[5,6] and J. Lesgourgues[1]

https://classclapp.streamlit.app/ or https://github.com/santiagocasas/clapp

Example Questions:

1. Can you show me how to plot in python the temperature C_ells for CMB using classy?
2. Can you show me a python code to plot the ratio between the lensing C_ells when using two nonlinear models: hmcode and halofit?
3. Give me a python code that compares all nonlinear matter power spectrum methods available in classy and shows the ratio against halofit
4. Give me the growth rate as a function of z, between 0 and 3 for a LCDM cosmology
5. Show me the energy density of dark energy as a function of z, for a DESI-like cosmology with w0=-0.7 and wa=-0.8?

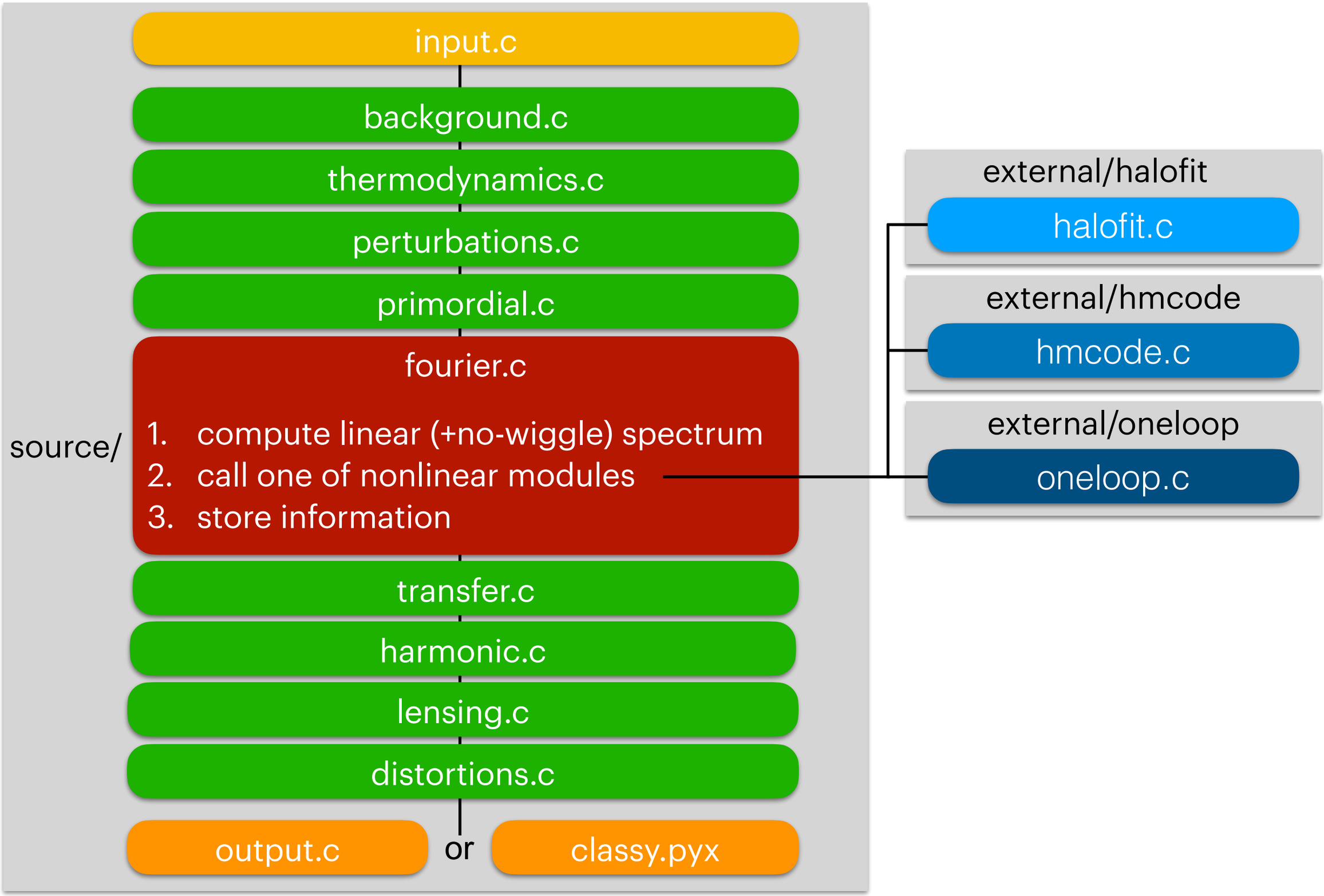
Range of application

- Lots of cosmologies within FLRW (incl. non-standard DM models, neutrinos, DE models)
- Lots of observables (incl. density Cl's, shear Cl's w/o Limber, HMcode 2020)
- Separate public branches: `GW_CLASS` for GW anisotropies, `ExoCLASS` for interface with particle physics codes, `class_matter` for FFTlog instead of line-of-sight…
- CLASS interfaced from MontePython, Cobaya, Cosmosis, CosmoPower, OLÉ https://github.com/svenguenther/OLE (see arXiv:2503.13183, Günther et al.)

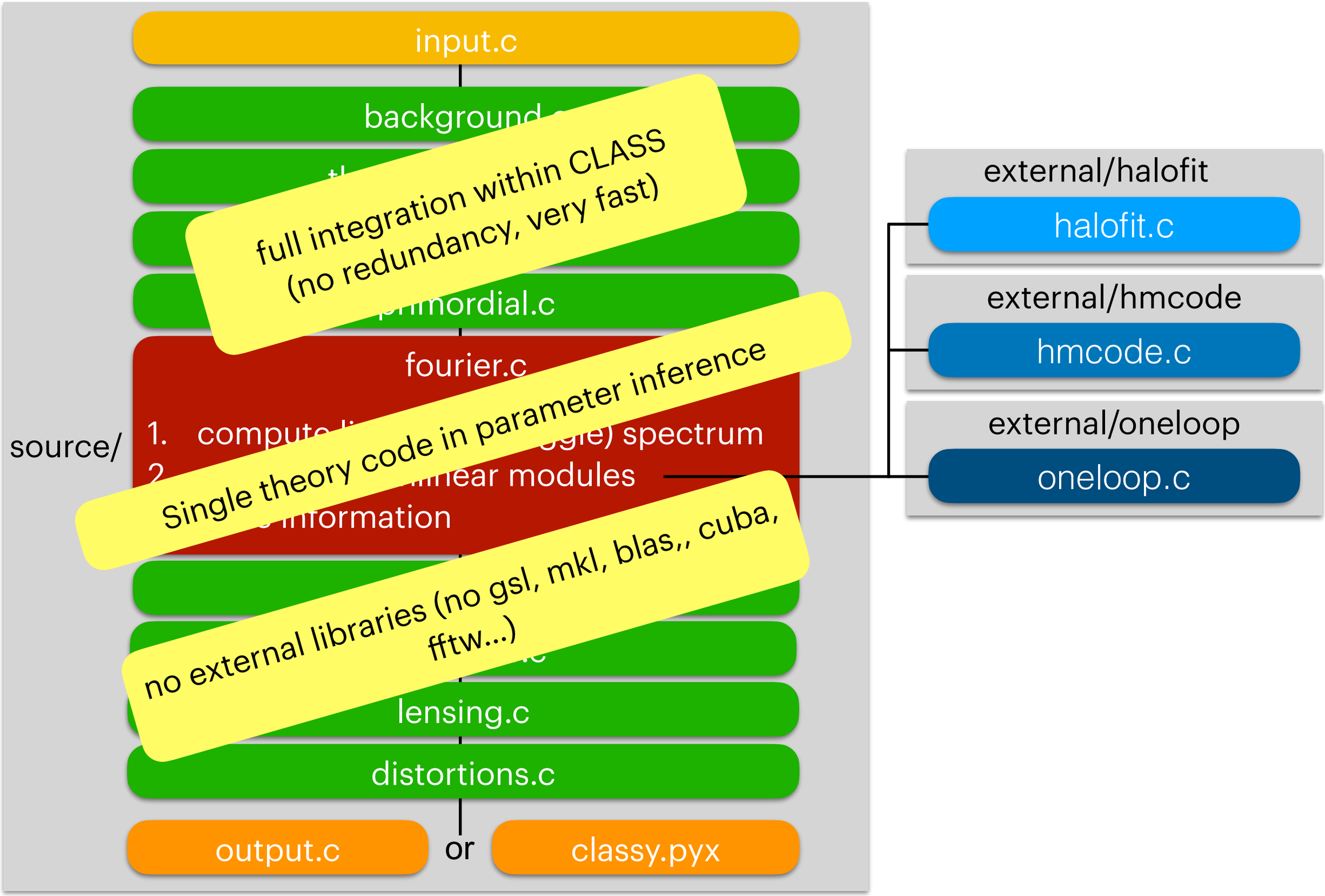
Range of application

- Lots of cosmologies within FLRW (incl. non-standard DM models, neutrinos, DE models)
- Lots of observables (incl. density Cl's, shear Cl's w/o Limber, HMcode 2020)
- Separate public branches: `GW_CLASS` for GW anisotropies, `ExoCLASS` for interface with particle physics codes, `class_matter` for FFTlog instead of line-of-sight…
- CLASS interfaced from MontePython, Cobaya, Cosmosis, CosmoPower, OLÉ https://github.com/svenguenther/OLE (see arXiv:2503.13183, Günther et al.)
- Coming soon:
  - 3.4: New architecture for classy.py, DM interacting with massive neutrinos, vector modes…
  - 3.5: One-loop matter power spectrum and bispectrum of EFTofLSS

Overall structure of CLASS v3.5



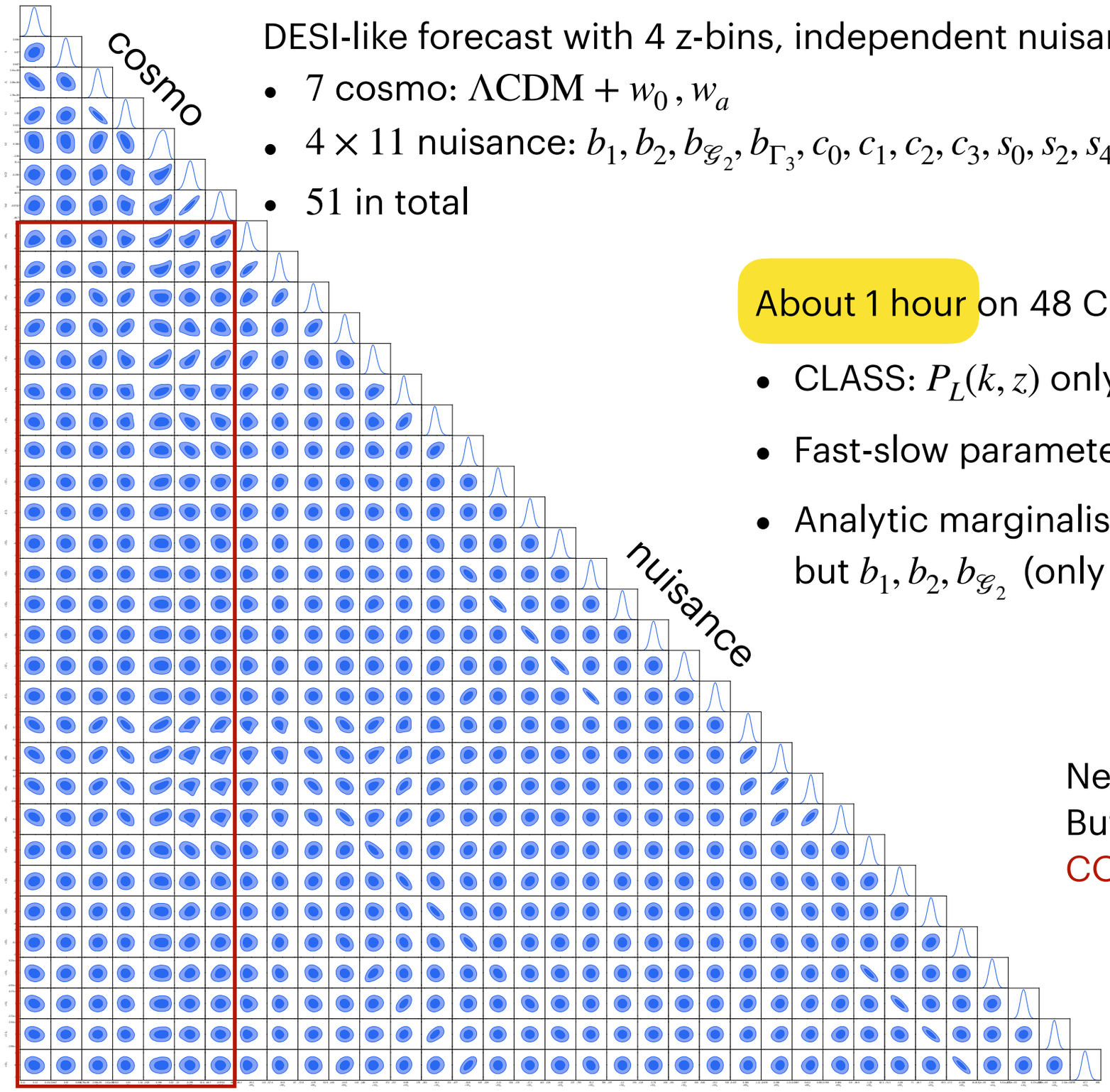
Overall structure of CLASS v3.5



What happens in the oneloop module ?

- Step 1 : compute log-Fourier transform of 42 convolution kernels $K_{ij}^{(n)}$ for 42 loop integrals
  - Cosmology-independent: once per MCMC (then cached)
  - takes **0.6 s**
- Step 2 : compute log-Fourier transform of linear spectrum $P_{\rm lin}$ into coefficients c_i , then compute 42 loops $L^{(n)} = c_i \, K_{ij}^{(n)} \, c_j$ at these z_k
  - Cosmology-dependent, but independent of nuisance parameter: once per CLASS call = every time slow parameters are varied (Cholesky decomposition)
  - takes **0.07s** in rest of CLASS + **0.09 s** in one-loop module
- Step 3 : assemble $P_X^{\rm rsd}(k,z,\mu)$ or $P_{X,\ell=0,2,4}^{\rm rsd}(k,z)$ for $X \in$ {matter, tracers, cross} for requested {bias} {counter-terms} {stochastic terms} $\{z\}$ $\{\mu$ or $\ell\}$
  - Depends on nuisance parameter
    = every time fast parameter are varied (Cholesky decomposition).
    No need to go through perturbation module again
  - takes **0.003 s**

MontePython run



DESI-like forecast with 4 z-bins, independent nuisance parameters across bins

- 7 cosmo: $\Lambda\mathrm{CDM} + w_0\,, w_a$
- 4×11 nuisance: $b_1, b_2, b_{\mathcal{G}_2}, b_{\Gamma_3}, c_0, c_1, c_2, c_3, s_0, s_2, s_4$
- 51 in total

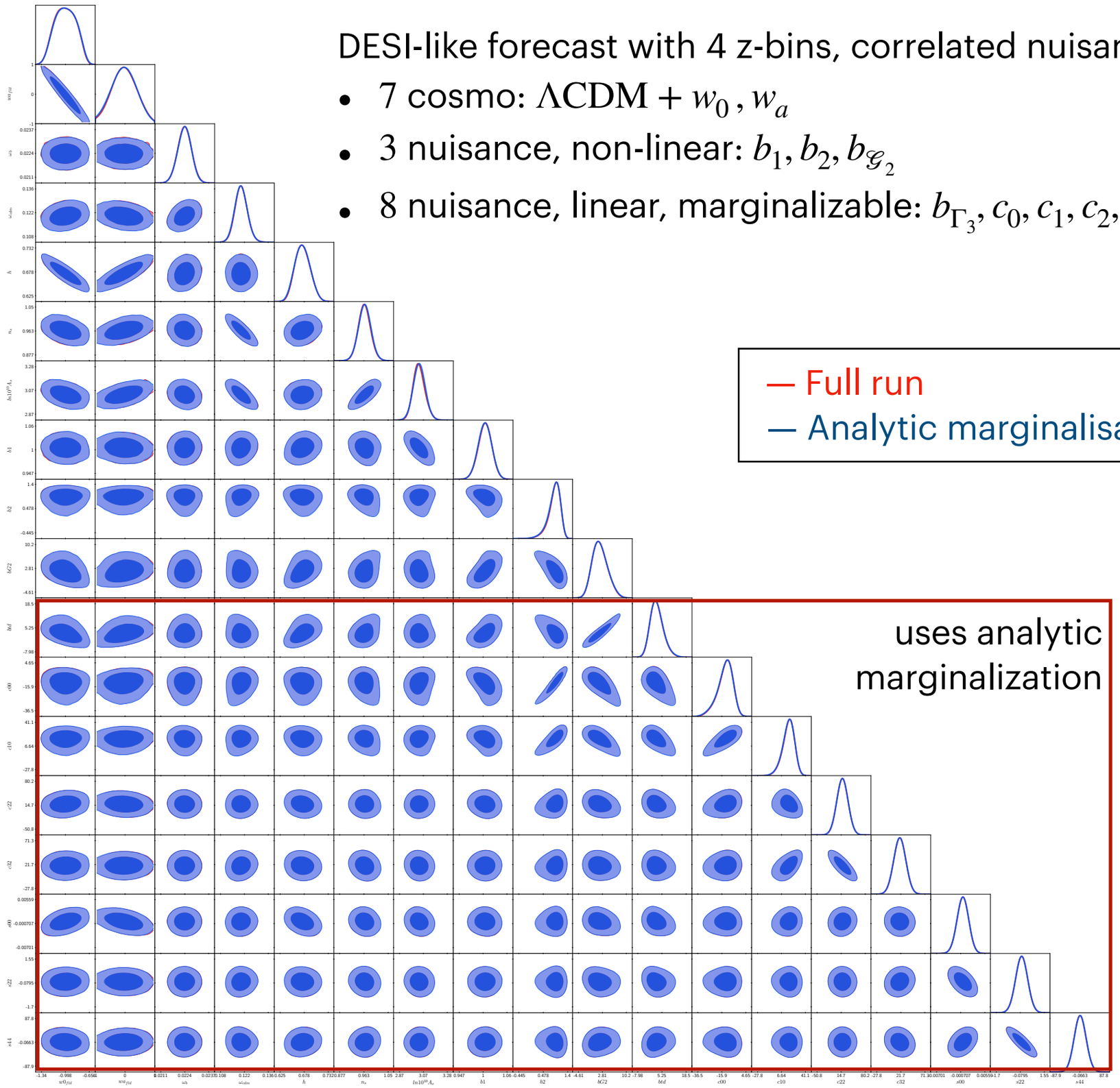
About 1 hour on 48 CPUs (8 chains on 6 cores each)...

- CLASS: $P_L(k,z)$ only, OneLoop module adds 0.09s
- Fast-slow parameters (Cholesky)
- Analytic marginalisation over all nuisance param. but $b_1, b_2, b_{\mathcal{G}_2}$ (only 4×3 fast param.)

Need for emulator not obvious...
But compatible with OLÉ, CONNECT, CosmoPower...

Institute for Theoretical Particle Physics and Cosmology | RWTH AACHEN UNIVERSITY

MontePython run



DESI-like forecast with 4 z-bins, correlated nuisance parameters across bins

- 7 cosmo: $\Lambda\mathrm{CDM} + w_0\,, w_a$
- 3 nuisance, non-linear: $b_1, b_2, b_{\mathcal{G}_2}$
- 8 nuisance, linear, marginalizable: $b_{\Gamma_3}, c_0, c_1, c_2, c_3, s_0, s_2, s_4$

— Full run
— Analytic marginalisation

Institute for Theoretical Particle Physics and Cosmology
RWTH AACHEN UNIVERSITY

MontePython run

```
#------Experiments to test (separated with commas)-----
data.experiments = [‘bbn_shifted' , 'desi_elg_oneloop']
data.over_sampling = [1, 6]

#------ Parameter list -------

# Cosmological parameter list, names need to be understood by class

data.parameters['w0_fld']          = [-1.,       -2.,   0., 1.2093e-01, 1., 'cosmo']
data.parameters['wa_fld']          = [ 0.,       -1.,   1., 3.2132e-01, 1., 'cosmo']
data.parameters['omega_b']      = [ 0.022445, None, None, 3.6756e-04, 1., 'cosmo']
data.parameters['omega_cdm'] = [ 0.1200,   None, None, 3.7164e-03, 1., 'cosmo']
data.parameters['h']                 = [ 0.6736,   None, None, 1.5357e-02, 1., 'cosmo']
data.parameters['n_s']               = [ 0.9649,   None, None, 2.4577e-02, 1., 'cosmo']
data.parameters['ln10^{10}A_s'] = [ 3.044784, None, None, 5.1376e-02, 1., 'cosmo']

# Nuisance parameter list, same call, except the name does not have to be a class name

# - biases
data.parameters['b1']      = [1.,     None, None, 2.5482e-02, 1., 'nuisance']
data.parameters['b2']      = [1.,     None, None, 1.8695e-01, 1., 'nuisance']
data.parameters['bG2']   = [1.,     None, None, 1.6272e+00, 1., 'nuisance']
data.parameters['btd']    = [1.,     None, None, 2.6540e+00, 1., 'nuisance']

# - counterterms
data.parameters['c00']     = [-10.,     None, None, 7.7154e+00, 1., 'nuisance']
data.parameters['c10']     = [ 20.,     None, None, 1.4329e+01, 1., 'nuisance']
data.parameters['c22']     = [ 20.,     None, None, 2.3444e+01, 1., 'nuisance']
data.parameters['c32']     = [ 20.,     None, None, 1.8421e+01, 1., 'nuisance']

# - stochastic terms
data.parameters['s00']     = [0.,       None, None, 1.8107e-03, 1., 'nuisance']
data.parameters['s22']     = [0.,       None, None, 4.7421e-01, 1., 'nuisance']
data.parameters['s44']     = [0.,       None, None, 2.4973e+01, 1., ‘nuisance']

# Fixed class parameters

data.cosmo_arguments[‘non_linear’] = ‘oneloop’
data.cosmo_arguments['Omega_Lambda'] = 0.
```

← Analytic marginalisation: nuisance -> derived_lkl