Title

ABCMB: A Python+JAX Package for the Cosmic Microwave Background Power Spectrum

Journal

Journal of Cosmology and Astroparticle Physics, 2026(08)

ISSN

1475-7516

Authors

Zhou, Zilu

Giovanetti, Cara

Liu, Hongwan

Publication Date

2026-08-01

DOI

10.1088/1475-7516/2026/08/078

Prepared for submission to JCAP

ABCMB: A Python+JAX Package for the Cosmic Microwave Background Power Spectrum

Zilu Zhou [a], **Cara Giovanetti** [b,c] **and Hongwan Liu** [d]

[a] Center for Cosmology and Particle Physics, Department of Physics, New York University, New York, NY 10003, USA
[b] Theory Group, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
[c] Leinweber Institute for Theoretical Physics, University of California, Berkeley, CA 94720, USA
[d] Physics Department, Boston University, Boston, MA 02215, USA

E-mail: zz1994@nyu.edu, cgiovanetti@lbl.gov, hongwan@bu.edu

Abstract. We present ABCMB, a differentiable Einstein-Boltzmann solver for the cosmic microwave background (CMB). ABCMB is an analysis-ready code package capturing important effects to linear order in ΛCDM cosmology. It computes the CMB power spectrum and includes effects like lensing, polarization, massive neutrinos, and a state-of-the-art treatment of Big Bang Nucleosynthesis and recombination. ABCMB has sub-percent-level agreement with CLASS and can be run on a GPU with competitive, and sometimes even faster, run times, owing to ABCMB's favorable run time scaling with maximum multipole moment. It is refactored compared to previous codes and takes advantage of object-oriented programming to improve extensibility, meaning new physics can be added to it without the need for modifying source files. ABCMB provides accurate and stable gradients to the user, making Fisher analyses straightforward, and enabling the use of efficient gradient-based sampling methods.

Contents

1 Introduction **2**

2 Using ABCMB **3**
2.1 JAX 3
2.2 Architecture and philosophy 4
2.2.1 Capabilities 5
2.3 Basic tutorial 6
2.3.1 Adding new species 7
2.4 Detailed functionality 10
2.4.1 Initialization and fluid species 10
2.4.2 Parameters and computation 15
2.4.3 Neutrino parameters 17

3 Gradients of ABCMB outputs **19**

4 Performance and validation **21**
4.1 HyRex and HYREC-2 22
4.2 The ABCMB Einstein-Boltzmann solver and CLASS 23
4.3 Performance 24

5 Discussion **28**

A Other options **30**
A.1 BBN 30
A.2 Parameters and run options 31
A.2.1 Initialization options 31
A.2.2 Cosmological parameters 35

B Physics implementation in ABCMB **37**
B.1 Recombination 37
B.1.1 Helium recombination 38
B.1.2 Hydrogen recombination 38
B.2 Reionization 41
B.3 Perturbations 41
B.3.1 Linear perturbations 41
B.3.2 Massive neutrinos 43
B.4 Transfer functions and power spectra 44
B.5 Lensing 46
B.6 Approximation schemes 47

C Currently implemented species **47**
C.1 Dark energy 47
C.2 Cold dark matter 48
C.3 Baryons 48
C.4 Photons 48
C.5 Massless neutrinos 49
C.6 Massive neutrinos 49

D Gradients with respect to cosmological parameters **50**

1 Introduction

The Cosmic Microwave Background (CMB) is a pillar of precision cosmology, with extraordinarily detailed measurements making it a rich source of information about the early universe. The success of *Planck* [1], the Atacama Cosmology Telescope (ACT) [2], and the South Pole Telescope (SPT) [3]/SPT-3G [4] in measuring CMB anisotropies to high precision has led to world-leading inference of cosmological parameters. Near-future results from the Simons Observatory [5] and SPT-3G can reduce uncertainties on these parameters even further.

As such, consistency with the CMB is a high barrier that any cosmological model must clear, be it the Lambda Cold Dark Matter (ΛCDM) paradigm or models of cosmology involving new physics. Indeed, an analysis of effects on the CMB power spectrum is central to investigating many models of dark matter or other physics beyond the Standard Model (BSM) (see e.g. refs. [6–14]). Performing such analyses often involves modifying and running public Einstein-Boltzmann solver codes, such as CLASS [15–18] or CAMB [19, 20], that compute the CMB anisotropy power spectrum in user-defined cosmologies.

Although existing Boltzmann codes are sufficiently fast for many applications, it is also easy to find examples of analyses involving complicated posteriors (e.g. bimodal posteriors), the comparison of a large number of different models, or high-dimensional parameter spaces, in which full parameter estimation with one of these codes becomes computationally expensive, requiring millions of calls to the Boltzmann code in traditional Markov-Chain Monte Carlo (MCMC) methods. This is especially true for analyses involving a large number of extra model or nuisance parameters, e.g. those involving the CMB in combination with Big Bang Nucleosynthesis (BBN) [21], large-scale structure [22], ultraviolet luminosity functions [23] and others.

Solutions to this problem fall into two primary categories. The first is emulation, in which the cost of an individual code call is reduced by orders of magnitude using machine learning to mimic output of a more expensive Einstein-Boltzmann or likelihood calculation [24–31]. While traditional emulators are limited by the need to retrain them for analyses involving new physics, recently released packages [29] have used online learning strategies to dramatically reduce this additional training time. The second strategy is differentiability, in which the Einstein-Boltzmann solver returns gradients of its output so that an investigator can use efficient, gradient-based sampling algorithms that require fewer code calls than methods that do not use gradients for sufficiently complex posteriors [32–35]. This approach maintains the reliability of first-principles computation while still offering performance improvements over non-differentiable codes. The SPT collaboration, for example, has released a differentiable likelihood package to be used in conjunction with these differentiable codes [36], and used it in analyses of the first SPT-3G data release [4]. A differentiable version of the likelihoods used in the *Planck* analysis is also available [37], opening the door to differentiable analysis pipelines for *Planck*. Having a differentiable Einstein-Boltzmann solver is essential to take full advantage of these tools. These two strategies are not mutually exclusive, as indeed many emulators are also differentiable.

Modifying existing codes to include new particle interactions or entirely new fluid species can also be challenging. While many public codes employ modular organization and are easy to browse, adding new parameters or functions may require declaring and defining new variables in separate source files, parsing large data structures (which may also be defined in source files separate from the ones the user wishes to modify), and generally being very familiar with the interior organization of the codes. Einstein-Boltzmann solvers tend to have large source files, and so the need to understand their interior design can be a hindrance to including this new physics.

In this paper, we introduce Autodifferentiable Boltzmann solver for the CMB (ABCMB), a new differentiable Einstein-Boltzmann solver. ABCMB computes the matter power spectrum and the CMB temperature and E-mode power spectra, achieving comparable accuracy to CLASS over a wide range of ΛCDM parameters, including the effects of massive neutrinos and lensing. It has competitive, and indeed sometimes faster, run times as compared to CLASS when run on a GPU for calculations of equivalent precision.

Primary breakthroughs of ABCMB include:

1. Ease of use and extensibility. New physics can be added to ABCMB without ever needing to open a source file, and the code is written in Python so that it can be easily learned and used by physicists.

2. A high precision recombination calculation required for state-of-the-art analyses. This is achieved through the companion code HyRex, a differentiable version of HYREC-2 [38], which is not currently implemented in other differentiable Einstein-Boltzmann solvers.

3. A fast, state-of-the-art BBN calculation. ABCMB ships with LINX [39] and can compute BBN abundances given user input, rather than interpolating pre-tabulated results, enabling precise BBN theoretical predictions.

ABCMB and HyRex use JAX [40, 41], making them faster than pure Python and backend-agnostic so they can be run on CPU or GPU, and easily interpretable to Python users, keeping barriers to entry low. These codes are fully differentiable, meaning functional gradients can be computed using autodifferentiation (AD), making the code particularly powerful for Fisher forecasting and when used in combination with efficient, gradient-based parameter estimation methods. ABCMB is available publicly via PyPI[1] and GitHub[2]. HyRex, included with ABCMB, is also available as a standalone package.[3]

In addition to introducing ABCMB, in this paper we take advantage of the code's differentiability to explore the relationships between the CMB power spectra and various cosmological parameters. In particular, because LINX is included with ABCMB, we are able to explore the interdependence of BBN abundances, the baryon density, the effective number of neutrinos, and the CMB power spectra.

The rest of the paper is organized as follows. The next section is dedicated to practical use of the code; after a brief discussion of JAX and its utility for physics codes, we provide a high-level overview of the philosophy of ABCMB. We proceed through a basic tutorial, followed by more detailed instructions for using the code. In section 3, we explore the advantages of the differentiability of ABCMB. We show how ABCMB can be used to better understand how the CMB and matter power spectra behave when cosmological parameters are varied, and can be combined with other differentiable tools to reveal parameter dependencies that have not previously been explored. We validate the code against CLASS and HYREC-2 [38] and benchmark the code performance in section 4, and conclude and discuss directions for future work in section 5.

The paper also includes four appendices. The first provides a complete index of run options and input parameters, beginning with an overview of options for BBN in appendix A.1 and detailing all other options in appendix A.2. Appendix B details the physics implementation in ABCMB, including computation strategies for recombination (appendix B.1) and reionization (appendix B.2), perturbations (appendix B.3), transfer and power spectra (appendix B.4), and lensing (appendix B.5). Appendix B.6 indexes approximation schemes used in the code. Appendix C provides further detail about the equations describing each fluid species in ABCMB, and appendix D includes reference figures of gradients of the ABCMB output computed with respect to a number of cosmological parameters.

2 Using ABCMB

2.1 JAX

ABCMB uses JAX, a library that can be imported to any Python script and used to write fast and differentiable code. The benefits of JAX for open-source physics codes are discussed in more detail in ref. [39] (section IIIA) in the context of the BBN code LINX. In short, Python code written with JAX can be Just-In-Time (JIT) compiled for faster run times, is backend-agnostic (meaning code that runs on standard CPUs can also be trivially GPU-accelerated), and is automatically parallelized over all available devices with JIT automatic parallelism. The JAX `vmap` transformation can be applied to any function to vectorize over its JAX array inputs. All of these features lead to superior performance

[1] https://pypi.org/project/ABCMB/
[2] https://github.com/TonyZhou729/ABCMB
[3] https://github.com/TonyZhou729/HyRex

over pure Python. Finally, JAX code is autodifferentiable, meaning gradients of all functions are computed by applying the chain rule to operations close to compiler-level (as opposed to numerical derivatives computed with finite differences), unlocking gradient-based sampling algorithms including Hamiltonian Monte Carlo (HMC) (see e.g. refs. [42, 43]) for efficient parameter estimation.

2.2 Architecture and philosophy

The ABCMB architecture is a significant departure from predecessor codes. We have refactored subroutines to improve extensibility, hardcoding as little as possible about the species in the ABCMB cosmology or calculation of various thermodynamic quantities.

The basic inputs required to perform a computation consists of a dictionary of parameters, and a collection of fluids, each defined as an independent fluid module. ΛCDM fluids—such as baryons, cold dark matter (CDM), and dark energy—already come predefined in ABCMB. Each fluid module contains methods for background quantities like energy density or equation of state, as well as for computing fluid perturbations according to the fluids' Boltzmann hierarchies. The dictionary of parameters can be used to either set properties within these fluids, e.g. the CDM energy density $\omega_{\rm cdm}$ and neutrino mass m_ν, or to set other global values required for the computation, such as the scalar amplitude A_s, the spectral index n_s, and the reionization optical depth $\tau_{\rm reion}$. New fluids can be defined by creating a new fluid module, which contains required methods for the fluid's background and perturbation quantities, and passing that module into ABCMB at initialization. Parameters of the new fluid can be included in the parameter dictionary as necessary; all new fluid parameters must be assigned a value in this dictionary for a successful computation.

While each fluid module is independent, methods within each module are written to accept both the parameter dictionary and a list of all perturbations, so that fluids can be easily coupled to one another. The computation of background quantities, perturbations, and spectra are each performed within their own centralized modules, which are aware of the full collection of fluids defined in the calculation. These modules can loop through all of the fluids and call methods within them, allowing them to construct e.g. the full set of coupled ordinary differential equations needed to solve for the evolution of perturbations, or integrate the Hubble parameter as a function of time. As an example, the module that computes perturbations constructs a single vector including density, velocity, shear, etc. perturbations for every fluid in a cosmology. When this module calls methods related to a given fluid, it passes a list of all fluids, and this entire vector of perturbations y, to the method. A method need only reference the indices of y related to another fluid to access the current values of the latter's Boltzmann hierarchy, and therefore couple to it. Fluids contain attributes that make finding those indices trivial, attributes which can be accessed through the list of fluids that is also passed to all methods.

We stress that these centralized background, perturbation and spectrum modules *are designed to minimize the need for users to modify them*: users should only need to modify the methods of individual fluids within each fluid module, or define new fluids as necessary. This philosophy makes extensions to ABCMB easy: the user does not need to modify any source files to include new fluids! Instead, the user need only define a new fluid class, which contains all of the background and perturbation quantities relevant to itself and pass in the new fluid at initialization. The background, perturbations, and spectrum modules will ask the new fluid for e.g. its energy density, or its perturbation equations, or for its attribute declaring whether it behaves as matter at late times, as they are needed. This behavior will occur without ABCMB being explicitly instructed to do so, so long as a new fluid module is defined with the attributes and methods the code expects. ABCMB provides template base classes, discussed in more detail below, to make the process of defining new fluids as streamlined as possible.

Fig. 1 shows the code block diagram for an ABCMB computation, organized by the computation modules (with methods of the fluid modules called by each of them as necessary). Control is managed by `abcmb.main.Model`, with its `__call__` the primary function used for computation.

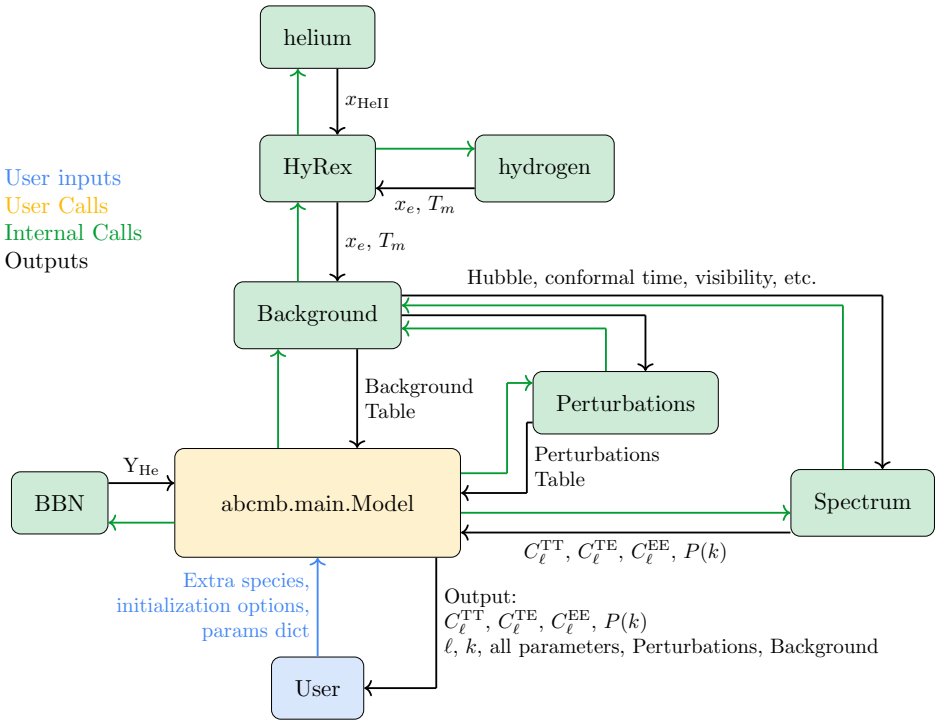

Figure 1: A code block diagram for ABCMB. Control is managed by `abcmb.main.Model`, which dispatches a BBN calculation, the computation of background quantities (including recombination), the evolution of perturbations, and the integration of those perturbations to compute a CMB power spectrum. The user will rarely have cause to call these submodules directly, apart from perhaps the HyRex module which can be run standalone. Instead these submodules return desired outputs through `abcmb.main.Model` to the user.

2.2.1 Capabilities

ABCMB is intended for use as a state-of-the-art Einstein-Boltzmann solver suitable for cosmological analyses, and currently includes a wide range of features needed to perform many of the typical analyses of existing CMB data with no loss of precision compared to other standard Boltzmann codes. In section 4, we will verify that our treatment of lensing, E-mode polarization, and massive neutrinos, in addition to our base Einstein-Boltzmann solver in ΛCDM, agrees with state-of-the-art codes at a level of roughly $2-4$ permille.

We include detailed treatment of recombination on par with HYREC-2, and push beyond the capabilities of state-of-the-art codes with a more detailed treatment of BBN. The inclusion of a BBN calculation is especially relevant in light of new measurements of the primordial helium-4 abundance [44]; BBN currently provides tighter constraints on $N_{\rm eff}$ than CMB experiments, even in combination [45], making it crucial to include BBN information in any analysis involving this parameter.

All of this is notwithstanding the technical capabilities of ABCMB—it is fully differentiable and has competitive run times as compared to CLASS. Indeed, since ABCMB is optimized to run on GPU, it scales more favorably with the number of modes ℓ considered than codes like CLASS do. When considering modes up to $\ell = 4000$, as is required to analyze data from ACT DR6 [2], ABCMB can even outperform the CLASS run time (see section 4).

ABCMB is easy to modify, and was written with new physics in mind. BSM approaches to the Hubble tension [8, 12, 13], neutrino mass hierarchies [14], interacting dark matter [6, 7], and more can

be added to ABCMB without needing to hack source files. Indeed, a few examples of modifications like these are demonstrated on the ABCMB GitHub.

Nevertheless, there are many desirable features that are not as yet included in ABCMB. To name a few examples, it currently does not allow the user to explore non-flat cosmologies (e.g. the curvature parameter is fixed to 0). In addition, ABCMB only allows for computation in the synchronous gauge, though this does not hinder the accuracy or capabilities of the code. Not all approximation schemes that improve the performance of other CMB codes are included in ABCMB (see section 4 and appendix B.6). ABCMB only computes E-mode polarization, and not B-modes. These enhancements and more are left to future work.

2.3 Basic tutorial

To compute the CMB power spectrum with ABCMB, the user should begin by initializing an instance of `abcmb.main.Model`. This wrapper class includes a number of options that can be specified at initialization, which will be discussed below. After the model is initialized, cosmological parameters can be specified in a dictionary and passed into the initialized `Model`. To run with default parameters and options (see appendix A.2), we can skip inputs for both initialization and call:

```
from abcmb.main import Model

# initialize the Model with default initialization options
model = Model()

# pass nothing into model to use default parameters
output = model()

# extract Cl, Pk, ells, k, and params as JAX arrays
ClTT = output.ClTT
ClTE = output.ClTE
ClEE = output.ClEE
Pk   = output.Pk

l          = output.l
k          = output.k
out_params = output.params
```

On call, `model()` outputs an object `output`. This object contains the CMB spectra C_ℓ^{TT}, C_ℓ^{TE}, C_ℓ^{EE},[4] as well as the linear matter power spectrum $P(k)$. Also included are a vector of multipoles ℓ for the CMB spectra, a vector of wavenumbers k for the matter power spectrum, as well as a full list of parameters used for this particular computation. Additional outputs related to the background and perturbation computation are discussed below.

To specify non-default cosmological parameters, use a dictionary containing the keys ABCMB expects (see appendix A.2 for a complete list):

```
from abcmb.main import Model
# we will need some functions from JAX's numpy; we cannot use regular numpy and still
# have fast code
import jax.numpy as jnp

# default initialization options
model = Model()

# pass new values for LCDM parameters, and Neff
params = {"omega_b"   : 0.0224, # Omega_b h^2
          "omega_cdm" : 0.13, # Omega_cdm h^2
```

[4] Not to be confused with $D_\ell \equiv (\ell(\ell+1)/2\pi)C_\ell$; ABCMB outputs C_ℓ.

```
          "h"         : 0.68,
          "A_s"       : 2.1e-9,
          "n_s"       : 0.98,
          "tau_reion" : 0.056,
          "Neff"      : 3.1}

# all other params use default values
output = model(params)
```

New C_ℓ's extracted from this `output` will reflect the new parameter values selected.

As this line is run, `model` is just-in-time (JIT) compiled—future calls of this function that use inputs of the same sizes and datatypes as above will use a cached compiled version of the code and will evaluate quickly (see section 4).

ABCMB organizes the background quantities (including recombination) and fluid perturbations into separate `Background` and `Perturbations` modules. Relevant quantities stored in these modules include values of fluid perturbations, the free electron fraction during recombination, the value of the Hubble parameter at all times, the redshift and sound horizon at baryon decoupling, etc.

```
from abcmb.main import Model
import jax.numpy as jnp

# initialize and call as before
model = Model()
output = model()

# After computation, we can access perturbations and background objects
PT = output.PT
BG = output.BG

# with BG exposed, access the redshift and sound horizon at decoupling in Mpc
z_d  = BG.z_d()
rs_d = BG.rs_d()

# with PT exposed, we can access the log scale factor during computation
perturbations_scale_factor = PT.lna

# get one value of wavenumber k at a corresponding index
k_index = 100 # can be anything from 0 to len(PT.k) (600 by default)
k_val = PT.k[k_index]

# get the photon density perturbation for all scale factors at given k index.  The values of
# the scale factor are already exposed in perturbations_scale_factor
photon_perturbations = PT.species_perturbations["Photon"]
delta_g = photon_perturbations["delta"][:, k_index]
```

See ABCMB documentation for a complete list of methods and attributes of these objects.

2.3.1 Adding new species

Finally, it is possible to input new fluids to `abcmb.main.Model` to create user-defined cosmologies. Massive neutrinos are not included in the default ABCMB cosmology but are predefined and can be added easily. For example, CMB analyses with massive neutrinos frequently set massive neutrino parameters in the following way:

```
from abcmb.main import Model
from abcmb import species # we will need the species module
```

```
# define a tuple of species to add.  MassiveNeutrino is already defined in species.py
user_species = (
    species.MassiveNeutrino,
)

# add the new species to ABCMB via input to Model at initialization
LCDM_mnu = Model(user_species = user_species)

# MassiveNeutrinos have some extra parameters you can specify
params_LCDM_mnu = {
    "N_nu_massless" : 2.,      # Scaling of massless neutrino energy density
                                  # (literal number of species if there are no massive
                                  # neutrinos)
    "N_nu_massive"  : 1.,      # Number of species of massive neutrinos
    "T_nu_massive"  : 0.71611, # Massive neutrino effective temperature ratio.
    "m_nu_massive"  : 0.06,    # Neutrino mass, in eV
}

output_mnu = LCDM_mnu(params_LCDM_mnu)
```

ABCMB also ships with a variety of base classes whose attributes and methods can be inherited to quickly define new fluids, as will be discussed in more detail in section 2.4. As an example to illustrate the relative ease with which user-defined fluids can be added to ABCMB, below we sketch how to add a new perturbed fluid to the code (more concrete examples are explored in depth on the ABCMB GitHub):

```
from abcmb.main import Model
from abcmb.species import StandardFluid
import abcmb.constants as cnst

import jax.numpy as jnp

# Define a new class that describes the relevant background and perturbation functions
# for this new species
class myFluid(StandardFluid):
    """
    My fluid.
    Required input parameters: params['myParam'].
    """

    # Number of moments in the Boltzmann Hierarchy
    num_equations = 2 # Pick a non free-streaming species, which requires density
                      # and velocity perturbations

    name = "myFluid"

    def __init__(self, first_idx, specs):
        super().__init__(first_idx, specs)

    def rho(self, lna, params):
        """
        Energy density at log scale factor lna.
        Should be in units of eV/cm^3.
        """
        ...

    def P(self, lna, params):
```

```
        """
        Pressure at log scale factor lna.
        """
        ...

    def y_ini(self, k, tau_ini, args):
        """
        Adiabatic superhorizon initial conditions for the new fluid.
        """
        params = args
        myNewParam = params["myParam"] # we will give this a value in the params dict later
        ...

    def y_prime(self, k, lna, metric_h_prime, metric_eta_prime, y, args):
        """
        Derivatives of the fluid's perturbations w.r.t. lna.
        """
        ...

    def output_perturbations(self, lna, modes, args):
        """
        Instructions for which perturbation equations to output
        """
        ...
```

Note that all calculations in ABCMB take place in the synchronous gauge when defining methods related to perturbations.

Once these methods are defined, we pass the fluid into `Model` at initialization through the `user_species` argument. Then, when specifying the input parameters, the user need only provide a value for the key they used to define their model parameter—this new parameter does not need to be declared elsewhere.

```
user_species = (
    myFluid,
)

myModel = Model(user_species = user_species)

# the methods we defined above are expecting an entry in the params dict for "myParam", which
# has no default value and therefore must be specified here:
params = {
    "myParam" : .3,
}

output_myFluid = myModel(params)
```

This will compute the CMB power spectrum in a cosmology with all ΛCDM fluids and this new fluid. We emphasize that *ABCMB allows the user to add this new fluid without ever opening a single source file*, avoiding the need to hack multiple source files in languages other than Python to add new species, a procedure that is typical of other Boltzmann solvers.

Below, we provide more details about advanced functionality, other initialization options, and other abstract base classes that can be used to define new fluids. Additional examples are also available on GitHub.

2.4 Detailed functionality

2.4.1 Initialization and fluid species

The user primarily interfaces with `abcmb.main.Model` in order to run the entire code. `Model` includes a number of initialization options which the user can specify as keyword arguments; see appendix A.2 for a complete list.

`Model` also contains a list of fluid species. By default, an ABCMB cosmology includes CDM, dark energy, baryons, photons, and massless neutrinos. The user can choose to add additional fluids to `Model`, which are appended to the end of the list. These additional fluids can be other species that ship with ABCMB (e.g. massive neutrinos), or user-defined by inheriting from the ABCMB `Fluid` object or its child classes.

We utilize the equinox [46] package to manage object oriented programming, subclassing and inheritance. A parent object from the ABCMB `species` module specifies all attributes and methods a fluid species must contain. We have included several parent fluids that the user may choose to define a custom fluid.

The most fundamental of these parent classes is the `Fluid` class. To specify a `Fluid`, the user must specify all of the following methods:

- `rho`: The energy density ρ of a fluid with respect to log scale factor $\ln a$, in $\mathrm{eV\,cm^{-3}}$.
- `P`: The pressure p of a fluid with respect to log scale factor, in $\mathrm{eV\,cm^{-3}}$.
- `y_ini`: A vector of initial conditions for all perturbation modes of the fluid, as a function of wavenumber, and initial conformal time for the fluid.

  The evolution of perturbed fluids in ABCMB is described by a differential equation for the Boltzmann hierarchy F of the fluid in the synchronous gauge. For each moment F_l, the user should specify $\frac{dF_l}{d\ln a}$ for the fluid, which may be a function of other moments of the fluid, moments of other fluids, or other parameters. The values for the perturbations F_l for all fluids are stored in a vector `y`.

  `y_ini` is the initial condition for all of a fluid's F_l at an initial conformal time τ_0. It should have a length equal to the fluid's number of l moments (described below).
- `y_prime`: A vector of derivatives with respect to $\ln a$ for the perturbed moments of the fluid (e.g. the functions $\frac{dF_l}{d\ln a}$ for all l moments), given wavenumber, log scale factor, metric perturbations, and current fluid perturbation values. This vector should also have a length equal to the fluid's number of l moments.
- `rho_delta`: To define this method and those that follow, we use the formalism of ref. [47], in the synchronous gauge. See appendix B for more details. The perturbed line element in the Friedmann-Robertson-Walker metric has the form

$$ds^2 = a^2(\tau)\left(-d\tau^2 + (\delta_{ij} + h_{ij})dx_i dx_j\right).$$

  For scalar perturbations h_{ij} can be written in Fourier space as

$$h_{ij}(\vec{x},\tau) = \int d^3\vec{k}\, e^{i\vec{k}\cdot\vec{x}}\left[\hat{k}_i\hat{k}_j h_m(\vec{k},\tau) + \left(\hat{k}_i\hat{k}_j - \frac{1}{3}\delta_{ij}\right)6\eta(\vec{k},\tau)\right], \tag{2.1}$$

  where, throughout this paper, we write the first metric perturbation as h_m so that it may be disambiguated from the dimensionless Hubble constant h. One can solve for the metric perturbation evolution and find

$$h_m' = \frac{2k^2\eta}{\mathcal{H}^2} + \frac{8\pi G a^2}{\mathcal{H}^2}\sum_i \bar{\rho}_i\delta_i \tag{2.2}$$

$$\eta' = \frac{4\pi G a^2}{\mathcal{H}k^2}\sum_i (\bar{\rho}_i + \bar{p}_i)\,\theta_i\,, \tag{2.3}$$

where primes denote derivatives with respect to $\ln a$. The density perturbation for a fluid i, δ_i, is defined as $(\rho_i - \bar{\rho}_i)/\bar{\rho}_i$, where $\bar{\rho}_i$ the homogeneous background density and ρ_i the actual energy density, including perturbations. Similarly the velocity perturbation $\theta_i \equiv \nabla \cdot \vec{v}_i$, or in Fourier space $ik^j v_j$. $\mathcal{H} = aH$ is the conformal Hubble parameter, G is Newton's constant, and $\bar{\rho}_i$ and $\bar{p}_i$ denote the background energy density and pressure. The sums in eqs. (2.2) and (2.3) run over all fluid species in a cosmology.

`rho_delta`, therefore, is a method that computes the fluid's contribution to this sum. It returns $\bar{\rho}_i \delta_i$ for the given species, to be included in the sum in eq. (2.2) when computing h'_m. This method takes a log scale factor and a vector of magnitudes of perturbations of all fluids `y`, and returns the fluid's $\bar{\rho}_i \delta_i$ in units $\mathrm{eV\,cm^{-3}}$.

- `rho_plus_P_theta`: A method to compute a fluid's $(\bar{\rho}_i + \bar{p}_i)\,\theta_i$, to be included in the sum in eq. (2.3) when computing η'. This method takes a log scale factor and a vector of magnitudes of perturbations of all fluids `y`, and returns the fluid's $(\bar{\rho}_i + \bar{p}_i)\,\theta_i$ in units $\mathrm{eV\,cm^{-3}\,Mpc^{-1}}$. Note the difference in units from `rho_delta` and `rho_plus_P_sigma` (below), as the velocity perturbation θ_i carries units of Mpc^{-1}.

- `rho_plus_P_sigma`: The CMB source functions (see eqs. (B.21)) depend on higher derivatives of the metric perturbations, and involve a sum over $(\bar{\rho}_i + \bar{p}_i)\,\sigma_i$ for each fluid, where σ_i is the fluid's shear perturbation. `rho_plus_P_sigma` is the fluid's contribution to this sum, in units $\mathrm{eV\,cm^{-3}}$, given the log scale factor and perturbation magnitudes.

- `output_perturbations`: A book-keeping method to instruct which perturbation equations of the fluid should be saved in a perturbation dictionary in `PerturbationTable`. This is an opportunity for the user to view the perturbation evolution of their new species. For instance, `Baryon.output_perturbations` saves both the density and velocity perturbations so that δ_b and θ_b are both visible after evaluation of a model.

`Fluid` also includes a method `w` (the equation of state), which is automatically defined to be the fluid's `P`/`rho` and does not need to be implemented by the user.

In addition, a `Fluid` includes the following fields (static attributes):

- `first_idx`: an integer index that tracks the position of the `Fluid`'s density perturbation in ABCMB's internal array of perturbations `y`. This parameter defaults to 0 but is automatically set internally when `Model` is initialized, and so it does not need to be set by the user.

- `num_equations`: the number of moments in the `Fluid`'s Boltzmann hierarchy, which defaults to 0. In the array of perturbations `y`, all perturbations for fluid are stored in `y[first_idx:first_idx + num_equations]` For all fluids, the density perturbation δ_i is stored at position `first_idx`, θ_i at `first_idx+1`, σ_i at `first_idx+2`, and so on, until the highest mode in the hierarchy is stored at index `first_idx+num_equations-1`.

- `name`: a string naming the fluid, which defaults to an empty string.

- `is_matter`: a boolean that indicates whether the fluid contributes to the matter overdensity today. Defaults to `False`.

These four fields should be treated as static—once a `Fluid` is initialized, the user should not attempt to update these attributes.

While any new fluid can be defined as a child class of `Fluid`, ABCMB also includes classes that inherit from `Fluid` and have already defined some of these methods to expedite the process of defining new species in some broad, general cases.

The first is `BackgroundFluid`, which is used for fluids that do not have perturbations. A fluid that inherits from `BackgroundFluid` has its perturbations methods set to empty arrays or 0. The user need only define a `BackgroundFluid`'s `rho` and `P`, and optionally `name` and `is_matter` (though we expect the default `is_matter=False` to be appropriate in most cases). `num_equations` is fixed to

0 in this parent class. For example, one can define a dynamical dark energy scenario by inheriting from `BackgroundFluid`:

```
from abcmb.species import BackgroundFluid
import jax.numpy as jnp

class DynamicalDarkEnergy(BackgroundFluid): # inherits from BackgroundFluid!
    """
    Dynamical dark energy implementation using the CPL parameterization.

    Required input parameters (passed in at runtime): params['w0DE'], params['waDE']

    Required derived parameters (computed given other parameters): params['omega_Lambda']
    """

    name = "DynamicalDarkEnergy"

    def __init__(self, first_idx, specs):
        super().__init__(first_idx, specs) # super init as usual in Python

    def rho(self, lna, params):
        """
        Energy density at log scale factor lna,
        in units of eV/cm^3.
        """
        rho_i = params["omega_Lambda"] * (3.*cnst.H0_over_h**2/8./jnp.pi/cnst.G)
        a = jnp.exp(lna)

        # Below is obtained by solving continuity equation \dot{\rho} = -3H\rho*(1+w)
        return rho_i * a**(-3.*(1.+params["w0DE"]+params["waDE"])) * \
             jnp.exp(3*params["waDE"]*(a-1))

    def P(self, lna, params):
        """
        Pressure at log scale factor lna.
        """
        a = jnp.exp(lna)
        w = params["w0DE"] + (1-a)*params["waDE"]
        return w*self.rho(lna, params)
```

This new fluid can be passed to the `user_species` argument of `Model` at initialization and will be treated appropriately so long as values of `w0DE` and `waDE` are specified in the `params` dict; we will demonstrate how to do this in the next subsection. Note that we specify which parameters we need to define in the fluid's doc string; this is done as a convenience for the user in all fluids that ship with ABCMB.

Before specifying values of these parameters, though, we must initialize a model with this new fluid. This fluid can even replace the default dark energy fluid in ABCMB by instructing ABCMB not to use the default ΛCDM species:

```
from abcmb.main import Model

# Since we're not using the default species set, we need to add all the other
# fluids too.  They are already defined for us in species.
user_species = {
    species.Baryon,
    species.Photon,
    species.ColdDarkMatter,
```

```
    species.MasslessNeutrino,
    DynamicalDarkEnergy # our new species
}

# be sure to instruct Model not to populate the LCDM species, since we are replacing one
DDEModel = Model(user_species=user_species, use_LCDM_species=False)
```

The above is appropriate for fluids without perturbations. Our other parent class, `StandardFluid`, is used for fluids with perturbations. A `StandardFluid` still requires the user to specify `rho`, `P`, `y_ini`, and `y_prime`, as well as all of the attributes of a `Fluid`. However, the required `rho_delta`, `rho_plus_P_theta`, and `rho_plus_P_sigma` are automatically defined for a `StandardFluid` based on the fluid's `self.rho` and `self.P`:

- `rho_delta`: `y[self.first_idx]*self.rho`, where `y[self.first_idx]` is the full vector of perturbations `y` sliced at the index of the density perturbation for this fluid `self.first_idx`.
- `rho_plus_P_theta`: `y[self.first_idx+1](self.rho+self.P)`, where `y[self.first_idx+1]` is the velocity perturbation for this fluid.
- `rho_plus_P_sigma`: `y[self.first_idx+2](self.rho+self.P)`, where `y[self.first_idx+2]` is the shear perturbation for this fluid.

This streamlines the process of defining new perturbed fluids; such a fluid can inherit from this parent class to avoid needing to specify these three methods.

The user is free to specify additional methods in a new species. This is done routinely in ABCMB for `StandardFluid`s; ABCMB's `species.Baryon` is an example of a child class of `StandardFluid` that defines these necessary methods as well as auxiliary methods unique to this fluid. `species.Baryon` also illustrates the ease with which fluids can be coupled to one another, without needing to hardcode couplings into source files:

```
from abcmb.species import StandardFluid
import abcmb.constants as cnst
import jax.numpy as jnp

class Baryon(StandardFluid):
    """
    Baryon fluid species implementation (abbreviated)
    """

    name = "Baryon"
    num_equations = 2
    is_matter = True

    def __init__(self, first_idx, specs):
        super().__init__(first_idx, specs)

    def rho(self, lna, args):
        ...

    def P(self, lna, args):
        ...

    # An example of an auxiliary method not inherited from Standard Fluid.
    # See also how we can easily access aspects of the photon fluid!
    def cs2(self, lna, args):
        """
        Baryon sound speed squared.
        """
```

```
        BG, params, species_list, species_dict = args
        # Get photon class from list
        i = species_dict["Photon"]
        photon = species_list[i]

        Tm = BG.Tm(lna, params) # Baryon temp
        Tg = BG.TCMB(lna, params) # Photon temp
        # we'll need another function, mean_mass, to compute this sound speed
        mu = self.mean_mass(lna, (BG,params))
        R = 4.*photon.rho(lna, params)/3./self.rho(lna, params)

        return Tm/mu * (5./3. - 2./3.*mu*R/cnst.me/BG.aH(lna, params)/BG.tau_c(lna, params) \
               * (Tg/Tm - 1.))

    # Another auxiliary function
    def mean_mass(self, lna, args):
        ...

    def y_ini(self, k, tau_ini, args):
        ...

    def y_prime(self, k, lna, metric_h_prime, metric_eta_prime, y, args):
        # we include y_prime to highlight how fluids are coupled to one another--nothing
        # is hardcoded here!

        BG, params, species_list, species_dict = args
        # species dict tells us at what index the photon is stored
        i = species_dict["Photon"]

        # now we access the actual photon object with species_list
        photon = species_list[i]

        aH = BG.aH(lna, params)
        cs2 = self.cs2(lna, args) # use our custom method
        R = 4.*photon.rho(lna, params)/3./self.rho(lna, params)
        tau_c = BG.tau_c(lna, params)

        delta = y[self.first_idx]
        theta = y[self.first_idx+1]

        # get the photon's velocity perturbation with photon.first_idx.  This index is the
        # location of the first mode of the Boltzmann hierarchy, so add one to get to the
        # velocity perturbation
        theta_g = y[photon.first_idx+1]
        delta_prime = -theta/aH-metric_h_prime/2.

        # compute baryon theta prime including photon slip
        theta_prime = -theta + cs2*k**2*delta/aH + R/tau_c/aH*(theta_g-theta)

        return jnp.array([delta_prime, theta_prime])

    def output_perturbations(self, lna, modes, args):
        return {
            "delta": modes[self.first_idx],
            "theta": modes[self.first_idx + 1],
        }
```

While we do not demonstrate it here, even the child classes defined in ABCMB can be used as parent

classes for other fluids. The ABCMB GitHub includes an example of a child class that inherits from ABCMB's `species.ColdDarkMatter`.

Keeping a consistent structure across all fluids allows the rest of the program to operate while remaining agnostic to the specifics of any individual fluid. All new physics the user may wish to introduce to our program can be specified in one place, while leaving other modules unmodified.

Note that the user does not (and should not) specify the cosmological parameters during initialization. In other words, the user may specify that there is a new fluid in their model, and can indicate where a new parameter will be used with a new key in the parameter dictionary, but no value for that parameter should be specified at this stage. Only the fields—the immutable attributes of an object—are specified during this step. Attempting to change a field mid-way through an analysis will trigger recompilation, and so model parameters whose values may change as the model is called repeatedly should not be assigned values until the next step.

2.4.2 Parameters and computation

After an instance of `Model` has been initialized with the desired initialization options and fluids, the user can specify cosmological parameters through dictionary inputs to the initialized `Model`'s `params` parameter.

ABCMB natively allows a number of different parameters to define a cosmology—see appendix A.2 for a complete list and default values. If in defining new fluids the user needs additional parameters, these parameters need only be passed in with the expected key. For example, the `DDEModel` that was defined in the previous subsection required two new parameters, `params['w0DE']` and `params['waDE']`:

```
# continuing from dynamical dark energy example above, where we initialized DDEModel

# omega_b and Neff are ABCMB parameters, which can always be specified.  w0DE and waDE
# were used to define DynamicalDarkEnergy methods, and now we need to give them values:
params = {
    "omega_b" : 0.0224,
    "Neff"    : 3.1,
    "w0DE"    : -0.8,
    "waDE"    : -0.6,
}

# all other params use default values; call the code and store the output
output_DDE = DDEModel(params)
```

These values in the `params` dictionary, as well as default values for all other parameters, are automatically passed to all methods of all fluids. Since `DDEModel` was already initialized with the `DynamicalDarkEnergy` fluid, it will automatically be able to access these values for `params['w0DE']` and `params['waDE']`.

When called, ABCMB will first compute derived parameters from the user's input parameters, such as total radiation energy density (which is fully specified given other fluid parameters) or the Hubble constant H_0 (which is specified by the user's input dimensionless Hubble constant h). A full list of derived parameters is included in appendix A.2.

Depending on the selected initialization options (see appendix A for more information), the calculation of derived parameters may also include a BBN calculation. ABCMB can dispatch LINX to compute the helium-4 mass fraction Y_{He} given other cosmological parameters. LINX allows the user to modify BBN nuisance parameters not typically included in interpolation tables, such as the neutron lifetime or uncertainties on nuclear reaction rates, in order to demonstrate their affect on the CMB power spectrum, . Alternatively, ABCMB can interpolate over a table of pre-tabulated values of Y_{He}, but this will not include the effects of BBN nuisance parameters. The user may also input Y_{He} directly. We note that LINX does not (currently) share the same extensibility strategy as ABCMB, and may need to be modified separately for new physics. For generic new physics scenarios (other

than nonstandard $N_{\rm eff}$), there are no BBN interpolation tables available for Boltzmann codes, and so if BBN is significantly impacted by new physics LINX will need to be modified in these scenarios. See appendix A.1 for more information about ABCMB and BBN.

After all parameter values are set, `Model` moves on to dispatch the various modules for computation of the CMB and matter power spectra.

The `Background` module is called next, which is responsible for computing various background quantities. These include the total energy density, Hubble, conformal time, and other quantities of interest such as the redshift of baryon decoupling and the sound horizon at decoupling. `Background` is also responsible for managing recombination, and the companion code HyRex has been integrated into this module for this purpose (see appendix B.1). `Background` stores critical background functions which appear frequently in the calculation of recombination, perturbations, and power spectra, including Hubble, the visibility function, and conformal time. These functions are also directly accessible to the user through the `BG` attribute of the object returned by `Model`.

Once the background cosmology is computed, ABCMB uses the output from `Background` to compute cosmological perturbations in the `Perturbations` module. This is the most computationally expensive aspect of the calculation, requiring integration of Boltzmann hierarchies for all species in the user-specified cosmology, tracking the evolution of all of these moments. The physics and equations implemented in this module are described in appendix B.3; these quantities, including density, velocity, and shear perturbations (or higher moments) are also accessible to the user through the `PT` attribute of the object returned by `Model`, with the option to append the perturbations of any user defined species as well.

With the output from the perturbations module, `Model` finally dispatches the computation of the power spectra in the `Spectrum` module. This module integrates transfer functions against tabulated Bessel functions to compute the matter power spectrum and C_ℓ's, optionally including effects from lensing. This calculation is described in more detail in appendices B.4 and B.5.

$N_{\rm eff}$. $N_{\rm eff}$ is an optional input in the `params` dictionary in ABCMB (it can also be treated as a derived parameter, described below). If the user chooses to specify `params['Neff']`, ABCMB will automatically adjust the massless neutrino energy density to provide a cosmology with the desired $N_{\rm eff}$, with this parameter defined as

$$N_{\rm eff} = \left(\frac{\rho_R - \rho_\gamma}{\rho_\nu^{\rm inst}}\right)_0 . \tag{2.4}$$

ρ_R is the total energy density in radiation,[5] ρ_γ is the energy density in photons and $\rho_\nu^{\rm inst}$ is the energy density in one neutrino species assuming that species decoupled instantaneously (so its temperature is $T_\nu^{\rm inst} = (4/11)^{1/3} T_\gamma$). The subscript "0" indicates that this ratio should be computed well after electron/positron annihilation (or any other transitions in a user-defined cosmology that would change the number of total number of effectively massless degrees of freedom in the early universe). We note that in Boltzmann codes, $N_{\rm eff}$ is typically treated as a constant anyway, since modes around the time of electron-positron annihilation freeze-out lie well beyond experimental reach. Nevertheless it can be useful to insist on this "post-annihilation" definition of $N_{\rm eff}$ in case ABCMB is used in conjunction with LINX for BBN, since these energy densities do evolve during BBN.

This prescription decouples user input radiative species from the calculation of $N_{\rm eff}$. In other words, the user is free to specify a new fluid whose energy density contributes to ρ_R, and yet can still input $N_{\rm eff}$ independently of the parameters of the fluid. For example, consider a cosmology with user-defined dark radiation, whose energy density ρ_d contributes meaningfully to ρ_R. If the user inputs $N_{\rm eff} = 4.5$ and yet $\left(\rho_d/\rho_\nu^{\rm inst}\right)_0 = 0.5$ given other input fluid parameters, ABCMB will still realize a cosmology with $N_{\rm eff} = 4.5$ by adjusting the energy density in massless neutrinos. More specifically, ABCMB adjusts the number of massless neutrino species; this procedure is discussed in more detail in the next section.

[5] ABCMB actually tallies the energy density in all species $\rho_{\rm tot}$ to compute $N_{\rm eff}$, but since it performs this calculation deep in radiation domination, $\rho_{\rm tot} \approx \rho_R$ to precision far beyond what can be measured today.

This prescription works for cosmologies with massless neutrinos, cosmologies with massive neutrinos, and cosmologies with user-defined radiative fluids, so long as the user's requested $N_{\rm eff}$ is not so small that it would require a negative number of massless neutrinos to realize (in which case ABCMB will throw an error and exit). This is also demonstrated in more detail in the next section.

ABCMB also allows the user to treat `params['Neff']` as a derived parameter, meaning the user can specify all other physical parameters contributing to $N_{\rm eff}$, and allow ABCMB to compute and store the resulting $N_{\rm eff}$. In this case, the user is required to specify a number of massless neutrino species. All other parameters that contribute to $N_{\rm eff}$—the massless neutrino temperature, the number and temperature of massive neutrino species, parameters controlling the energy densities of user-defined fluids—must also be specified, or left at their default values in the case of other neutrino parameters. This option may be more desirable if the user wishes to explore cosmologies where changes to $N_{\rm eff}$ occur due to the inclusion of a specific new fluid, or if the scenario of interest is particularly sensitive to the neutrino temperature.

The user switches between these options based on the keys they choose to specify in the `params` dictionary. If the user specifies `params['Neff']`, the first approach will be used and the number of massless neutrinos will be adjusted to account for all other input parameters. If instead the user specifies the number of massless neutrino species `params['N_nu_massless']`, ABCMB will compute $N_{\rm eff}$ given this and all other inputs. If the user attempts to specify both `params['Neff']` and `params['N_nu_massless']`, the code will print a warning and exit without running any computations. If neither parameter is specified, the code defaults to $N_{\nu,\rm massless} = 3$ and adjusts $N_{\rm eff}$ accordingly (properly accounting for any user-defined fluid contributions to $N_{\rm eff}$). Default values are used for the massless neutrino temperature, the number of massive neutrinos, and the massive neutrino temperature in all scenarios, unless the user provides other values for these parameters (see appendix A.2).

2.4.3 Neutrino parameters

We include detailed descriptions of the implementations of all fluids in appendices B and C. However, since the details of the neutrino implementation are especially relevant for specifying $N_{\rm eff}$ and understanding user input parameters like the massive neutrino temperature, we define parameters related to these fluids here.

Massless neutrinos. The massless neutrino energy density is parametrized according to

$$\rho_{\nu,\rm massless}(a) = N_{\nu,\rm massless}\frac{7\pi^2}{120}T^4_{\nu,\rm massless}a^{-4}, \tag{2.5}$$

where $T_{\nu,\rm massless}$ is the temperature of massless neutrinos today and $N_{\nu,\rm massless}$ is the number of massless neutrinos. a is the scale factor at the given temperature. In analyses where all neutrinos are massless, $N_{\nu,\rm massless}$ is 3 by default and $T_{\nu,\rm massless}$ is $0.71636856 \times T_\gamma$, T_γ the photon temperature, to recover a default $N_{\rm eff}$ of 3.044 [48–50]. These parameters can be adjusted by specifying `params['N_nu_massless']` ($N_{\nu,\rm massless}$) and `params['T_nu_massless']` ($T_{\nu,\rm massless}/T_\gamma$ after decoupling).

Above, we discussed that the number of massless neutrino species $N_{\nu,\rm massless}$ may adjust in response to a user input $N_{\rm eff}$. In particular, after the user inputs a value for $N_{\rm eff}$, ABCMB loops through all fluids present in the user-specified cosmology—with the exception of massless neutrinos—and computes each fluid's contribution to $N_{\rm eff}$. If the calculated $N_{\rm eff}$ does not match the user specified $N_{\rm eff}$, ABCMB will make up the difference by adjusting $N_{\nu,\rm massless}$. In other words, ABCMB solves eq. (2.4) for $N_{\nu,\rm massless}$ given all fluids and $N_{\rm eff}$, and fixes this parameter to the value required to provide the requested $N_{\rm eff}$ in a user's specified cosmology.

Massive neutrinos. We follow the CLASS prescription for treating massive neutrinos. Massive neutrinos have three new parameters: $N_{\nu,\rm massive}$, the number of massive neutrinos, $T_{\nu,\rm massive}$, a temperature associated with massive neutrinos, and m_ν, the neutrino masses, which are assumed to be

degenerate. These parameters are derived based on the results obtained in ref. [51], which carefully studies the phase-space density of each neutrino flavor including distortions to the Fermi-Dirac distribution due to non-instantaneous decoupling.

Let $f_{\rm exact}(p,a)$ be the exact phase space density of one massive neutrino (as computed in ref. [51]) as a function of momentum p and scale factor a.[6] Then $T_{\nu,\rm massive}$ is set by ensuring that the number density $n_{\nu,\rm massive}(a=1)$—and likewise also $\rho_{\nu,\rm massive}(a=1) = m_\nu n_{\nu,\rm massive}(a=1)$—is equal to the number density of a free-streaming fermion that decoupled relativistically and instantaneously, with present-day temperature $T_{\nu,\rm massive}$, i.e.

$$\int \frac{d^3\vec{p}}{(2\pi)^3} f_{\rm exact}(p,a=1) = \int \frac{d^3\vec{p}}{(2\pi)^3} \frac{1}{e^{p/T_{\nu,\rm massive}}+1}\,, \tag{2.6}$$

where the left-hand side can be evaluated using the results of ref. [51], leading to the ΛCDM value for three degenerate neutrinos of[7]

$$\left(\frac{T_{\nu,\rm massive}}{T_\gamma}\right)_0 \approx 0.71611\,. \tag{2.7}$$

At all $a < 1$, we make the further assumption that the phase space density can be approximated as

$$f(p,a) \approx \left(e^{pa/T_{\nu,\rm massive}}+1\right)^{-1}\,, \tag{2.8}$$

i.e. it can be treated as free-streaming, with $T = a^{-1}T_{\nu,\rm massive}$. The massive neutrino energy density is then computed at all redshifts as

$$\rho_{\nu,\rm massive}(a) = 2N_{\nu,\rm massive}\int \frac{d^3\vec{p}}{(2\pi)^3} f(p,a)\sqrt{p^2+m_\nu^2}\,, \tag{2.9}$$

with the approximate $f(p,a)$ defined in eq. (2.8). This simplification, however, results in a $\rho_{\nu,\rm massive}$ that has fractional error from the true value $\rho_{\nu,\rm massive,exact}$ of

$$\frac{\rho_{\nu,\rm massive,exact}(a)}{\rho_{\nu,\rm massive}(a)} - 1 = \frac{\int \frac{d^3\vec{p}}{(2\pi)^3}\delta f(p)\sqrt{p^2+m_\nu^2a^2}}{\int \frac{d^3\vec{p}}{(2\pi)^3}[e^{p/T_{\nu,\rm massive}}+1]^{-1}\sqrt{p^2+m_\nu^2a^2}} \overset{a\to 0}{\to} \frac{\int \frac{d^3\vec{p}}{(2\pi)^3} p\,\delta f(p)}{\int \frac{d^3\vec{p}}{(2\pi)^3} p[e^{p/T_{\nu,\rm massive}}+1]^{-1}}\,, \tag{2.10}$$

where

$$\delta f(p) \equiv f(p,a=1) - \frac{1}{e^{p/T_{\nu,\rm massive}}+1}\,, \tag{2.11}$$

i.e. the spectral distortion relative to the assumed distribution in eq. (2.8) at $a = 1$. At sufficiently early times when $T_{\nu,\rm massive} \gg m_\nu a$, the ratio goes to a constant that is independent of m_ν and a.

To compensate for this error, as well as the error associated with assuming that all massive neutrinos have the same phase space, we simply set $N_{\nu,\rm massless}$ such that the total $N_{\rm eff}$ is equal to the user-specified value, which in ΛCDM is 3.044. Specifically, ABCMB performs the same procedure as in the massless case: it loops through all species, except massless neutrinos, and computes the contribution of each to $N_{\rm eff}$. When it comes time to add the massive neutrino contribution, it computes the approximate energy density $\rho_{\nu,\rm massive}$ with temperature $T_{\nu,\rm massive}$, knowing this is not the correct energy density in massive neutrinos at early times. Instead, ABCMB adjusts the energy density in massless neutrinos by adjusting $N_{\nu,\rm massless}$—exactly as it did in scenarios that do not involve

[6] In fact, we assume that all massive neutrinos have the same phase distribution by taking average of the phase space of each neutrino flavor.

[7] We note that this result is commonly extrapolated to cases where only one neutrino is massive, or where $N_{\rm eff}$ takes on a different value. We caution that extrapolation of the results in ref. [51] to this regime has not been validated, to our knowledge.

massive neutrinos. This means that $N_{\nu,\text{massless}}$ is not truly the "number" of massless neutrino species anymore in massive neutrino cosmologies. Instead, $N_{\nu,\text{massless}}$ absorbs a correction from the massive neutrino energy density being computed from $T_{\nu,\text{massive}}$.[8]

With the temperature set as described above, ABCMB computes the energy density and momentum of massive neutrinos by integrating their phase space with Gaussian quadrature—see appendix B.3.2 for more detail.

3 Gradients of ABCMB outputs

A major advantage of code written in JAX is the effortless calculation of gradients with autodifferentiation (AD). We emphasize that gradients computed with JAX, and therefore ABCMB, are not a result of numerical differentiation, but instead use AD (i.e. decomposing all functions into low-level operations like addition, multiplication, etc. whose functional derivatives are known, and applying the chain rule). This makes the resulting gradients more stable than traditional numerical derivatives and more reliable for techniques like Fisher forecasting (as recently explored in ref. [33]), and is more efficient than traditional methods like finite differences. As discussed in section 2.1, this feature can also be used with more sophisticated sampling techniques than traditional Markov Chain Monte Carlo (MCMC), leading to fewer function calls and more efficient parameter estimation for high-dimensional or complex posteriors.

ABCMB uses forward AD (or forward accumulation) by default. For a function

$$y(x) = f(g(h(x))), \tag{3.1}$$

forward AD computes $\partial y/\partial x$ by first computing $\partial h/\partial x$, followed by $\partial g/\partial h$, and finally followed by $\partial f/\partial g$, computing partial derivatives from inside to outside. By contrast, reverse AD computes derivatives in reverse order, beginning with $\partial f/\partial g$ and ending with $\partial h/\partial x$. In either case, the value of the partial derivative is propagated either forward or backward through these chains as subsequent derivatives are computed.

While these two operations are mathematically equivalent, their performance can vary depending on the scenario. If $y : \mathbb{R}^n \to \mathbb{R}^m$, forward AD is more efficient than reverse AD for $m > n$, while reverse AD is more efficient for $n > m$. This is because forward AD requires a new pass through the chain for each independent input variable (but only one pass to propagate each input variable through *all* outputs), while reverse AD instead needs a separate pass through the chain for all output variables (but only one pass to propagate to *all* inputs).

ABCMB has a large number of outputs—it is expected that several thousand C_ℓ and $P(k)$ values are computed in a single evaluation, and so forward AD is more appropriate. The same is true for HyRex. We include illustrations of gradients with respect to a number of cosmological parameters in appendix D. Each of these is computed using forward AD with the built-in JAX function `jax.jacfwd`.[9]

When ABCMB runs with LINX, the entire pipeline from BBN through recombination and the CMB power spectrum is fully differentiable. As such we are able to take derivatives of these quantities with respect to both the baryon density and N_{eff} through all of these epochs. LINX also allows the user to choose different reaction networks—different sets of experimentally-determined rates for the reactions active during BBN—and so here we compare the results from differentiating through ABCMB only, through its tabulated results using the reaction network of ref. [52], versus differentiating through both ABCMB and LINX, with the LINX default reaction network from ref. [53]. We show these results in figures 2 and 3. The effects of using a different reaction network with LINX when computing gradients with respect to these parameters is as large as percent-level in some spectra.

In new physics analyses, the ability to take gradients across epochs becomes especially important, in particular if scenarios of interest have model parameters that have a strong impact on BBN, or additional sensitivity to BBN parameters like the neutron lifetime or individual nuclear reaction rates.

[8] This is similar to the treatment in CLASS, where even in cosmologies where all three neutrinos are massive, the user still must include some small nonzero "`N_ur`" to achieve an N_{eff} of 3.044.

[9] For applications involving likelihoods, where the ABCMB output is collapsed to a single float, one might expect reverse AD to be more efficient. While ABCMB supports reverse AD, these efficiency gains do not materialize due to high memory requirements. Further optimization along this path is left to future work.

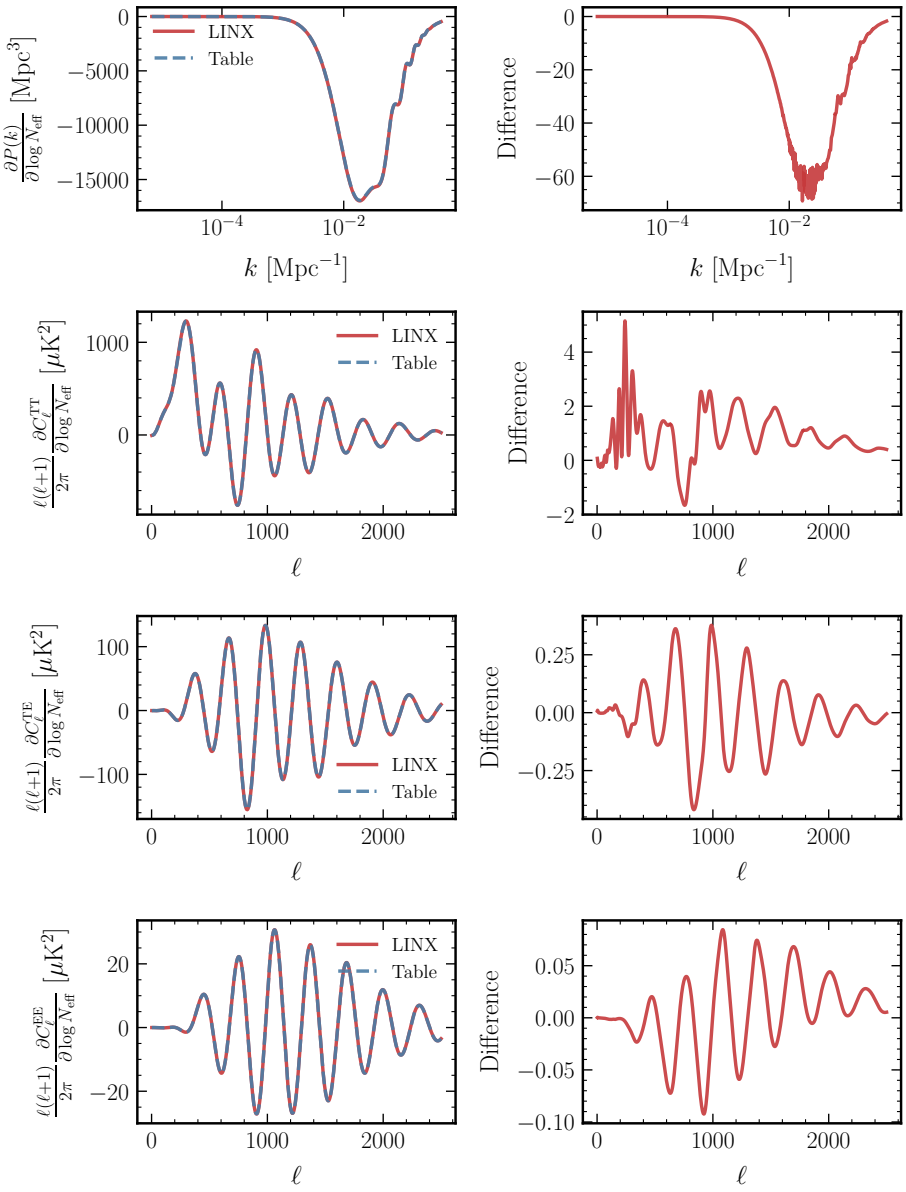

Figure 2: Gradients of matter and CMB power spectra with respect to $N_{\rm eff}$. The blue dashed curves were computed with helium abundance interpolated from the CLASS default BBN table, and the red solid curves used the LINX prediction for the abundance given $N_{\rm eff}$ with a different reaction network. The right panels of each spectrum gradient shows the absolute difference between the two approaches, with identical units and normalization as the left panels where applicable.

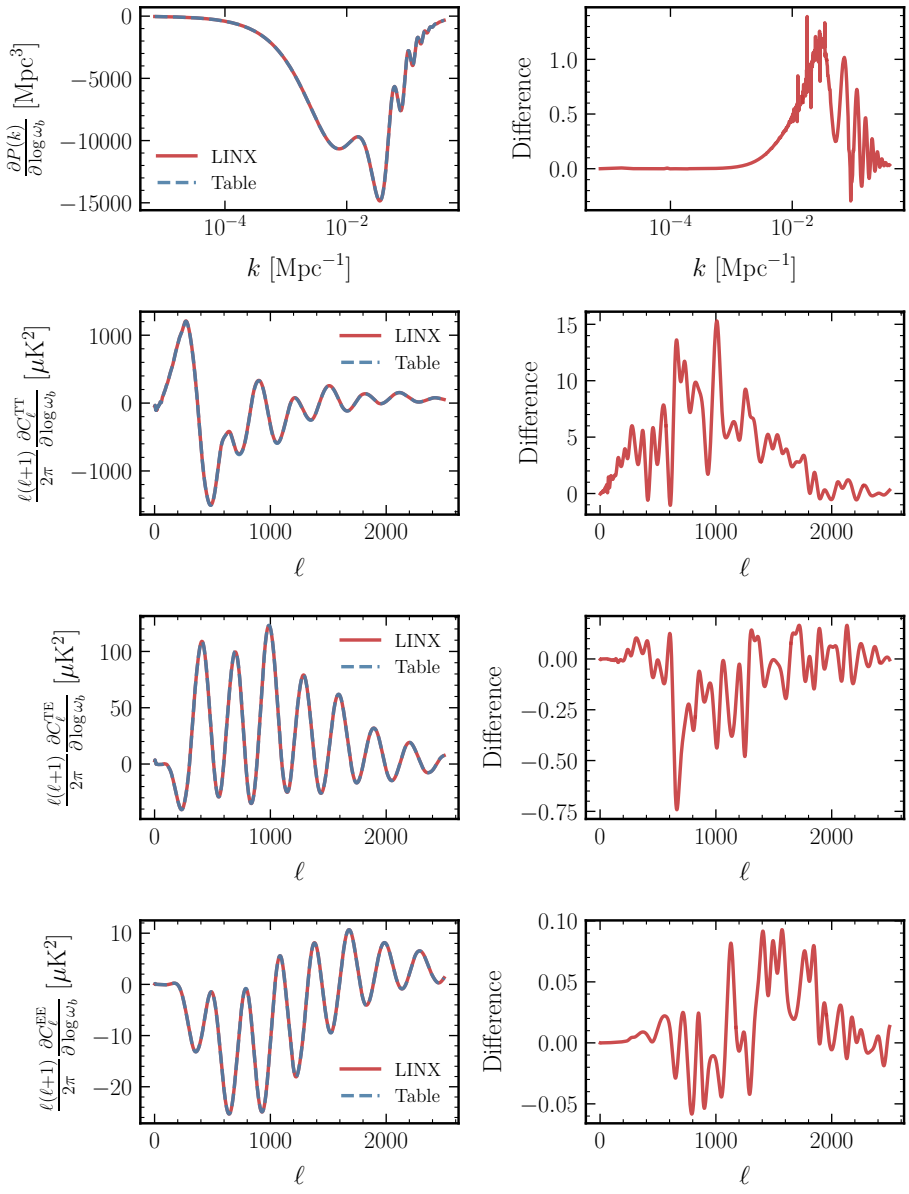

Figure 3: Gradients of matter and CMB power spectra with respect to the baryon density ω_b. The blue dashed curves were computed with helium abundance interpolated from the CLASS default BBN table, and the red solid curves used the LINX prediction for the abundance given ω_b with a different reaction network. As in fig. 2, the right panels of each spectrum gradient shows the absolute difference between the two approaches, with identical units and normalization as the left panels where applicable.

4 Performance and validation

We validate the output of HyRex and ABCMB against that of HYREC-2 and CLASS. We separately validate each of these components, and we further separate the matter power spectrum, the CMB

power spectrum, lensing, massive neutrinos, and polarization when validating our Einstein-Boltzmann solver. We also report the performance of the code on different hardware. We note that there is some variation of these reported run times with the version of JAX used, with version 0.4.28 and earlier running faster on CPU and versions 0.6.1 and later running faster on GPU. We report run times below only for versions $\geq$ 0.6.1, as there is no support for earlier versions of JAX on the newest versions of Python.

To ensure our code has good agreement with the benchmark codes HYREC-2 and CLASS across a wide range parameter values, we randomly generate 50 samples of the ΛCDM parameters. These are $\{h, \omega_b, \omega_{\rm cdm}\}$ (the Hubble parameter in units of $100\,{\rm km\,s^{-1}\,Mpc^{-1}}$, the present-day ratio of the baryon energy density to the critical energy density times h^2, and similarly for CDM) for recombination and $\{h, \omega_b, \omega_{\rm cdm}, A_s, n_s, \tau_{\rm reion}\}$ (additionally including the scale of the primordial power spectrum A_s, the scalar tilt n_s, and the optical depth to reionization $\tau_{\rm reion}$) for the CMB power spectra. These parameters are uniformly sampled over the $\pm 5\sigma$ range reported by Planck 2018, with the exception of the Hubble parameter, which is sampled over the wider interval $h \in [0.6, 0.8]$ in light of the Hubble tension. We summarize these parameter ranges in table 1 along with other fixed cosmological parameters.

Varied	**Range**
h	[0.6, 0.8]
ω_b	[0.02162, 0.02312]
$\omega_{\rm cdm}$	[0.114, 0.126]
$10^9 A_s$	[1.931, 2.256]
n_s	[0.9439, 0.9859]
$\tau_{\rm reion}$	[0.0179, 0.0909]
Fixed	**Value**
$N_{\rm eff}$	3.044
m_ν	0.06 eV
$T_{\nu,\rm massive}$	0.71611
$N_{\nu,\rm massive}$	1
$\Delta z_{\rm reion}$	0.5
$z_{\rm HeII\,reion}$	3.5
$\Delta z_{\rm HeII\,reion}$	0.5
$\beta_{\rm reion}$	1.5

Table 1: Parameters used to perform x_e comparison against HYREC-2, and C_ℓ, $P(k)$ comparisons against CLASS. The six ΛCDM parameters are uniformly sampled 50 times in the indicated range. These ranges correspond to the 5σ range reported by Planck 2018, and for h it is further widened in light of the Hubble tension. $N_{\rm eff}$ is fixed in these tests, and $\rm Y_{He}$ is inferred using the CLASS BBN table. In tests including one massive neutrino, we use the massive neutrino related parameters listed in the table, which are otherwise excluded. The four reionization parameters $\Delta z_{\rm reion}$, $z_{\rm HeII\,reion}$, $\Delta z_{\rm HeII\,reion}$, and $\beta_{\rm reion}$ are further discussed in appendix B.2.

For each random set of parameters, we compare the output of our code to the benchmark, and at each redshift we record the maximum error incurred across all 50 evaluations. The residuals are defined throughout as |benchmark − ABCMB|/benchmark.

4.1 HyRex and HYREC-2

HYREC-2 is a state-of-the-art recombination code whose output HyRex must match in order to deliver the precision demanded by e.g. *Planck*. We therefore follow the procedure described above, comparing the output of HyRex for each sample to that of HYREC-2. In particular we compare the x_e output of the two codes in the redshift range $1 \leq z \leq 8000$. We show the maximum error as a function of redshift in figure 4.

The agreement is excellent across the entire solution region. The outputs never differ by more than 0.1%, and the largest errors occur at late times, where recombination is only approximated with

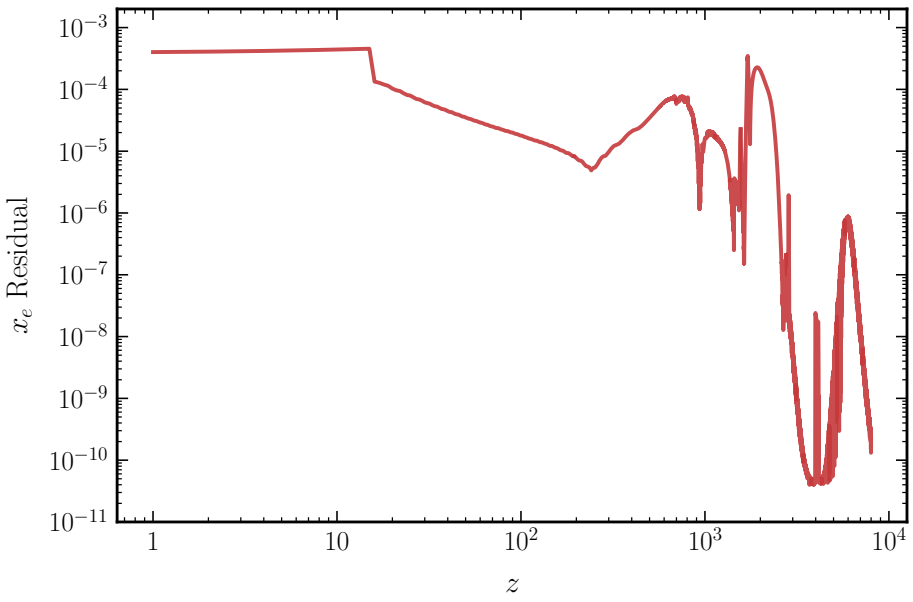

Figure 4: Maximum residual comparison between HyRex and HYREC-2 for x_e, the free electron fraction across redshift. We make 50 comparisons using randomly generated values for the ΛCDM parameters $\{h, \omega_b, \omega_{\rm cdm}\}$ in the range specified in table 1, and report the maximum error occurred at each redshift across all runs. The two codes agree to $< 0.1\%$, with better agreement at early times more relevant for the CMB.

the three-level atom (TLA) and reionization is more relevant. This is despite the use of different ODE solvers by the two codes.

4.2 The ABCMB Einstein-Boltzmann solver and CLASS

CLASS is widely used to compute the CMB power spectrum, and so we compare the output of the ABCMB Einstein-Boltzmann solver to that of CLASS. We check the linear matter power spectrum and the CMB power spectra (temperature autocorrelation and E-mode polarization autocorrelation), in ΛCDM (with massless neutrinos), including massive neutrinos, and with and without lensing. For the matter spectrum we compute to $k_{\rm max} = 0.5$ Mpc^{-1}, a small scale where non-linear corrections are known to be significant. We test the CMB spectra in the interval $2 \leq \ell \leq 4000$, covering the full measurement extent of both Planck 2018 [1] and ACT DR6 [2]. We follow the procedure described above, using CLASS as the benchmark, and record the largest residual difference compared to CLASS across all 50 runs at each ℓ and k. For the CLASS precision settings, we use `'accurate_lensing'=1`, which is important as the alternative approximation introduces percent level errors for $\ell > 3000$ (see appendix B.5 for further discussion). Other precision settings, such as the perturbation Boltzmann hierarchy cutoffs, and settings related to the fluid approximations, are set as the CLASS default values. We use the ABCMB default settings in these comparisons (see appendix A.2).

We show a comparison of the linear matter power spectrum in figure 5, both in ΛCDM and with massive neutrinos. In both cases we find good agreement with CLASS, to better than 2×10^{-3} on all scales. This indicates ABCMB accurately captures the percent-level suppression induced by massive neutrinos at large k.

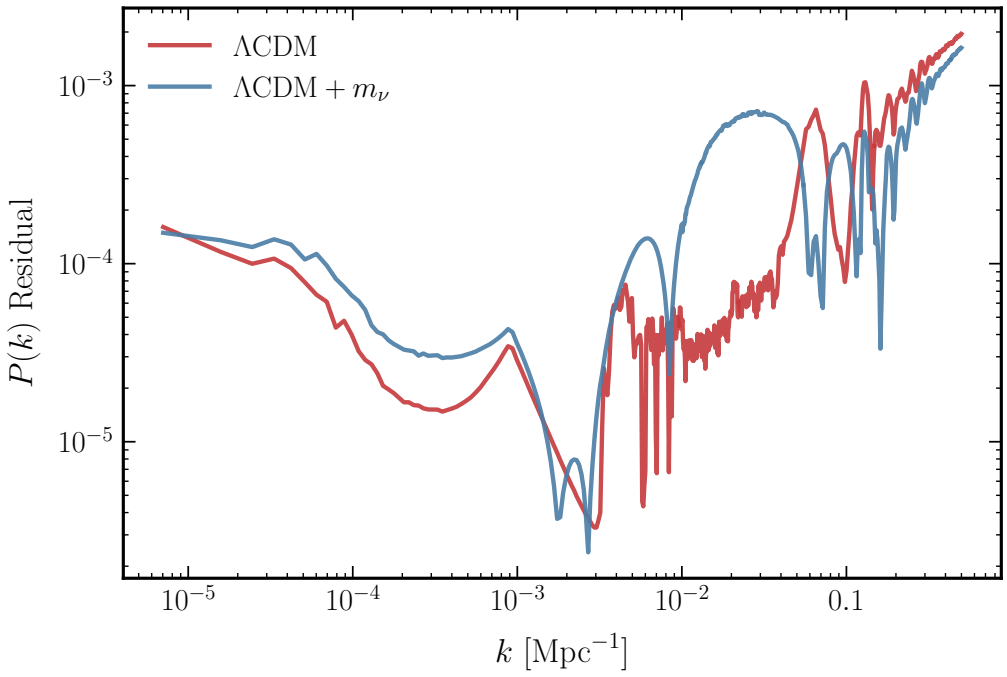

Figure 5: Residuals in the linear matter power spectrum computed as |CLASS − ABCMB|/CLASS. We make 50 comparisons using randomly generated values for the 6 ΛCDM parameters, and report the maximum error at each wavenumber k across all runs. The range of the parameters sampled can be found in table 1. The red curve denotes ΛCDM with massless neutrinos only, while the blue curve contains one massive neutrino. The errors are comparable in the two models, with accuracy at the 2 permille level at $k = 0.5$ Mpc^{-1}, where non-linear corrections are already important. The level of accuracy on these small scales indicate that the percent level suppression by neutrino mass is well captured in ABCMB.

Next we turn to the CMB power spectra. Here we only compare the TT and EE spectra to CLASS and do not compare the TE spectrum, since the cross correlation frequently crosses zero and makes the residual comparison difficult to interpret. Since the TE spectrum uses the same transfer functions as the TT and EE spectra, their accuracy should be representative of the cross correlation as well.

We show the TT spectra residual in figure 6 and the EE spectra residual in figure 7. In each figure we check the accuracy of the unlensed ΛCDM, lensed ΛCDM, and lensed ΛCDM + m_ν combinations. We find sub-percent accuracy in all cases, and in most cases we find the error hovers around 2 permille, with exceptions only at very low ℓ where CMB measurements are cosmic variance dominated.

4.3 Performance

As ABCMB is written in JAX, it is backend-agnostic and returns equivalent output on both CPU and GPU.

HyRex. On an M1 Macbook Pro with 8 cores, HyRex computes x_e, T_m and $\ln a$ during recombination and reionization in about 50 ms, after a ~ 30 s compile time. On a single Intel Xeon Platinum 8592+ 64C CPU core, the equivalent computation takes 37 ms, after a ~ 28 s compile time. Increasing the number of CPU cores available reduces compile time, but does not improve run time.

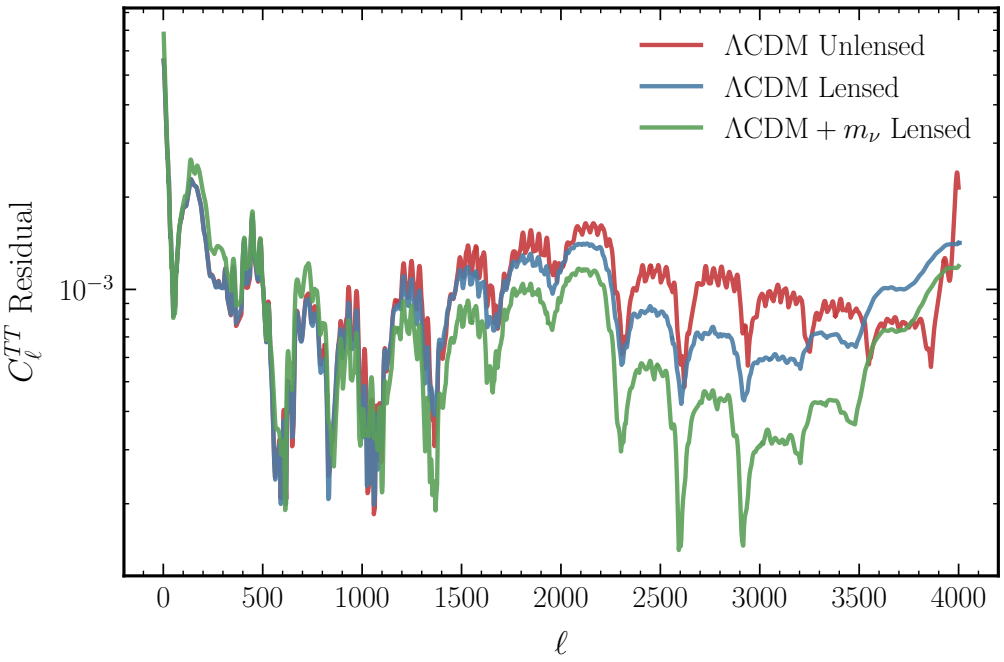

Figure 6: Residuals in the CMB TT power spectrum computed as |CLASS − ABCMB|/CLASS. We make 50 comparisons using randomly generated values for the 6 ΛCDM parameters, and report the maximum error at each angular scale ℓ across all runs. The range of the parameters sampled can be found in table 1. The red , blue and green curves correspond to the unlensed ΛCDM, lensed ΛCDM, and lensed ΛCDM with 1 massive neutrino, respectively. The agreement is better than 7 permille across all ℓ (and typically closer to 1 permille), and the presence of lensing and massive neutrino do not induce additional error above that of the unlensed spectrum.

On an NVIDIA L40S GPU run time is 1.5 s, and on an H200 NVIDIA GPU it is 1.7 s, with compile times around 30 s on both devices. This slowdown is likely from use of `lax.while_loop` and implicit use of the same function in the differential equation solvers, which relies on (slow) information exchange between the host CPU and the GPU.

ABCMB. These benchmarks include the HyRex run time, when HyRex is restricted to run on CPU (its faster backend). We conduct the tests using the CPU and GPU devices on the NYU Torch High Performance Computing cluster. All CPU tests are run on the Intel Xeon Platinum 8592+ 64C processors, while the GPU runs were tested on the NVIDIA L40S, A100, H100, and H200 cards. We compute the lensed ΛCDM and ΛCDM + m_ν CMB power spectra (including TT, TE and EE) up to $\ell_{\rm max} = 4000$, and the matter power spectrum up to $k_{\rm max} = 0.5$ Mpc^{-1}. The run times are shown in tables 2 (ΛCDM) and 3 (ΛCDM + m_ν). All runs are performed with 16GB memory. After an initial evaluation that compiles the code (compile time reported in fourth column), we report the average run time and standard deviation across 10 calls. We find that ABCMB performs most optimally on GPUs, outperforming its single CPU core runtime by about 10 times. We also run a test using 10 CPUs with JAX's built-in autoparallelization. We see that the compile time is considerably shorter, and the run time also sees a modest reduction.

The ABCMB run time can be compared against existing state-of-the-art Boltzmann solvers such as CLASS. However, given that ABCMB is GPU-optimized, while codes like CLASS and CAMB can are CPU-optimized, an apples-to-apples comparison between ABCMB and existing packages is

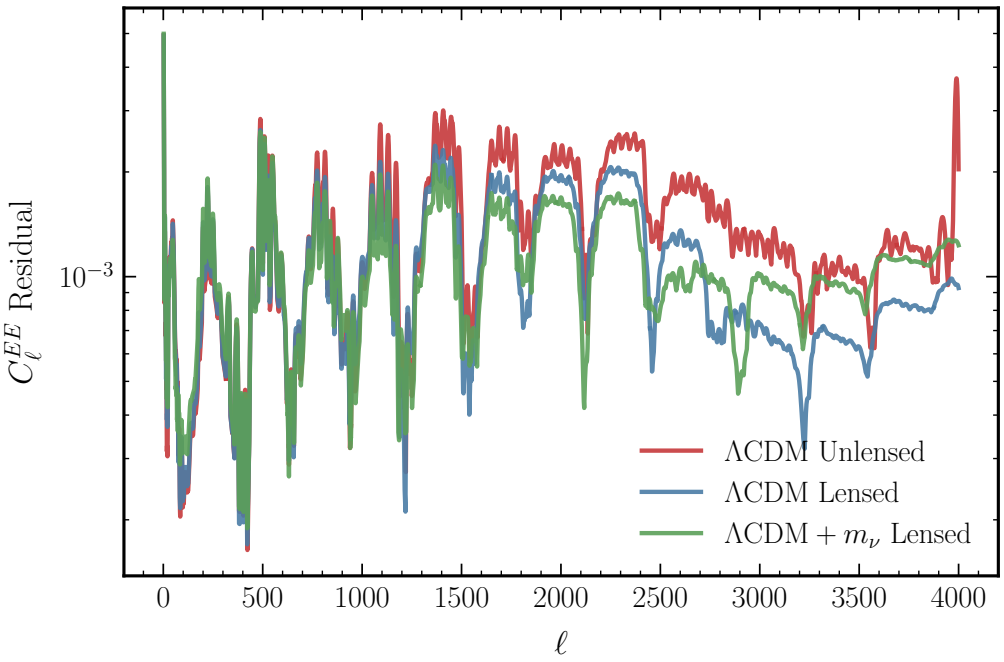

Figure 7: Residuals in the CMB EE power spectrum computed as |CLASS − ABCMB|/CLASS. We make 50 comparisons using randomly generated values for the 6 ΛCDM parameters, and report the maximum error at each angular scale ℓ across all runs. The range of the parameters sampled can be found in table 1. The red, blue and green curves correspond to the unlensed ΛCDM, lensed ΛCDM, and lensed ΛCDM with 1 massive neutrino, respectively. The agreement is comparable to the levels seen in the TT spectra, and again we find that lensing and massive neutrino do not induce more error.

Device	Mean (s)	Variance (s)	Compile Time (s)
CPU Single Core	110.360	0.382	202
CPU 10 Cores	60.001	1.429	108
NVIDIA L40S	13.737	0.032	125
NVIDIA A100	8.964	0.031	165
NVIDIA H100	6.705	0.018	120
NVIDIA H200	6.402	0.017	112

Table 2: ABCMB run times for ΛCDM with massless neutrinos only. All runs compute lensed TT, TE, and EE spectra to $\ell_{\rm max} = 4000$ as well as $P(k)$ to $k_{\rm max} = 0.5$ Mpc^{-1}. The rows for correspond to the devices used to conduct the tests. All run times are averaged over 10 identical runs with fluctuations characterized by the variance. We also include the compile time incurred on the initial call.

difficult. Nevertheless, a comparison between GPU ABCMB and CPU CLASS in a realistic scenario is still informative.

An analysis using Planck, ACT and SPT for CMB, in combination with BAO, requires computing the lensed temperature and polarization spectra out to $\ell_{\rm max} \sim 4000$, and the matter power spectrum to non-linear scales; we therefore compare at the same $\ell_{\rm max} = 4000$ and $k_{\rm max} = 0.5\,{\rm Mpc}^{-1}$ used in the standalone ABCMB tests above. We additionally require the accurate lensing setting for CLASS

Device	Mean (s)	Variance (s)	Compile Time (s)
CPU Single Core	256.475	4.940	346
CPU 10 Cores	213.445	4.263	270
NVIDIA L40S	26.980	0.092	181
NVIDIA A100	17.931	0.079	211
NVIDIA H100	12.813	0.059	141
NVIDIA H200	11.699	0.047	129

Table 3: ABCMB run times for ΛCDM with one massive neutrino. The runs use the same precision parameter and compute identical outputs as table 2.

which is relevant at such high ℓ's, and slightly impacts performance (see Appendix B.5). Modern analyses typically include one massive neutrino, which also noticeably impacts performance.

ABCMB run times under these conditions are reported in table 3, with the fastest time around 11.7s on the NVIDIA H200 graphics card. In contrast, on an Intel Xeon Platinum 8592+ 64C CPU, the analogous CLASS computation runs in about 13 seconds on a single core, or about 2 seconds threaded across 10 cores.

The CLASS performance is enabled by several perturbation approximation schemes that are not presently implemented in ABCMB. These include the tight-coupling approximation (TCA), the ultra-relativistic fluid approximation (UFA), and the radiation streaming approximation (RSA). TCA is used to alleviate the stiffness of the baryon-photon coupling equations at early times, and to remove the redundant higher moments (i.e. $l \geq 3$) in the photon Boltzmann hierarchy. UFA and RSA are used to drastically simplify the subhorizon modes of relativistic species, and a similar fluid approximation can be used in place of the massive neutrino Boltzmann hierarchy at late times to reduce complexity. Without the three fluid approximations, the CLASS run time can increase by a factor of about 5.

Nevertheless, there is not (as of yet) a backend on which ABCMB runs faster than multithreaded CLASS with fluid approximations. However, we note that since there are applications in which a solver may be restricted to a single device or core (e.g. nested sampling, or even MCMC with many chains), the single-core comparison is still informative. Since ABCMB can also use gradient-based sampling methods it is difficult to draw further comparisons between total analysis times of these two codes for applications like parameter estimation—the fact that the ABCMB runtime is across the board comparable to that of CLASS should be taken as an encouraging sign.

The ABCMB run time scales favorably with $\ell_{\rm max}$. This is because the code is GPU-optimized, and so evaluation can occur in parallel on a single device, until available memory is saturated. This means that in general one may expect the ABCMB run time to become even faster than the single-core CLASS run time as $\ell_{\rm max}$ increases, and we illustrate these scalings in the case of a massive neutrino in figure 8.

The ABCMB default parameters are selected to produce ~ 1 permille agreement with CLASS. However, this level of precision may not always be necessary, especially in more exploratory applications. In these scenarios, the user may enjoy $\sim 25\%$ faster run times across devices by setting the option `rtol_large_k_PE=1e-3` when initializing `Model` (see appendix A.2). This parameter controls the tolerances of the differential equation solver used for perturbations at large k, and the default 10^{-4} is only needed if ~ 1 permille agreement is required at large ℓ. We find that a larger tolerance of 10^{-3} is still sufficient to produce agreement at the level of ~ 8 permille with CLASS for both the TT and EE spectra. Adjusting this parameter has little effect on the agreement of $P(k)$ with CLASS, and so we certainly recommend increasing this tolerance if the user only requires the matter power spectrum for their analysis.

Gradients. We repeat the timing test discussed above for computing the gradients of ABCMB output with respect to cosmological parameters, using JAX auto-differentiation. We apply forward mode auto-differentiation to the function over the 6 ΛCDM parameters to obtain a gradient vector $\vec{\nabla}_\theta C_\ell$ for each of the TT, TE, and EE spectra, as well as $\vec{\nabla}_\theta P(k)$. We report the performance in

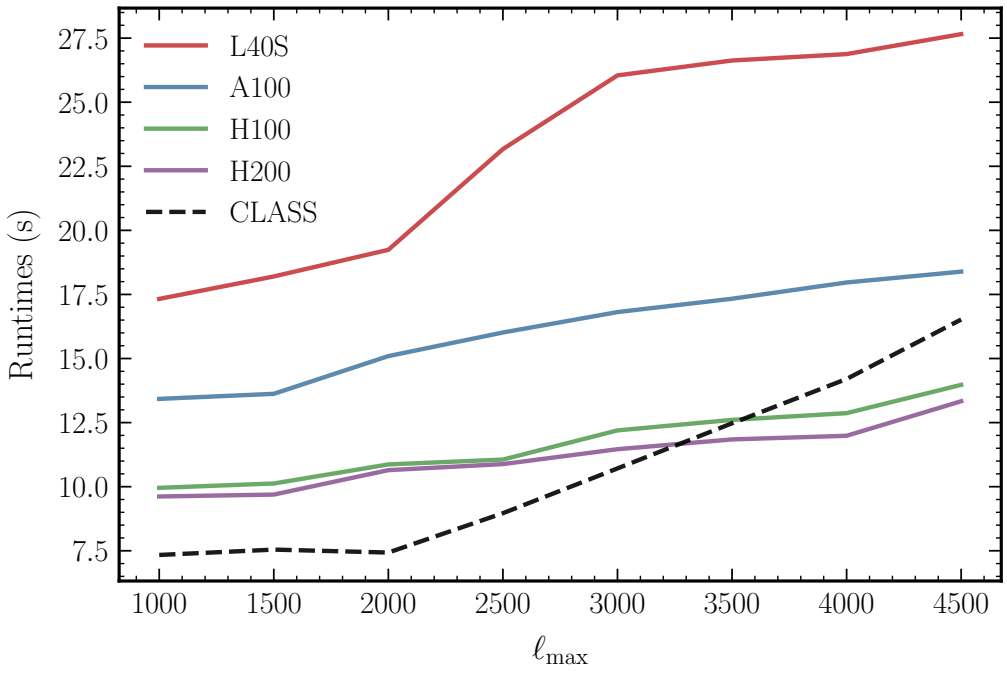

Figure 8: Single evaluation run times, after compilation, of ABCMB, on different hardware and as a function of ℓ_{max}, using default precision parameters and including one massive neutrino. The scaling is nearly flat, providing a performance advantage as ℓ_{max} grows. We also show the CLASS run time on a single Intel Xeon Platinum 8592+ 64C processor for comparison; the scaling is closer to linear and therefore the run time can become longer than the ABCMB run time for large ℓ_{max}.

table 4 for ΛCDM and in table 5 with one massive neutrino. CPU run times are not reported as they are too slow to be practical.

Device	Mean (s)	Variance (s)	Compile Time (s)
NVIDIA L40S	46.924	0.245	337
NVIDIA A100	25.051	0.311	370
NVIDIA H100	14.920	0.075	242
NVIDIA H200	13.336	0.064	260

Table 4: ABCMB run times for forward mode auto-differentiation on ΛCDM outputs (with massless neutrinos). As in the timing tests for the outputs themselves, each device test consists of one initial compiling run, and 10 subsequent runs with varying cosmological parameters.

5 Discussion

ABCMB is an accurate, differentiable, backend-agnostic Einstein-Boltzmann solver for the CMB, offering many precision features and written with extensibility in mind. We have validated the code against HYREC-2 and CLASS to deliver the same level of precision in the matter power and CMB power spectra, including effects from massive neutrinos, lensing, and polarization. This tool is useful for new physics analyses, as it provides a simple, intuitive, robust code architecture that can be

Device	Mean (s)	Variance (s)	Compile Time (s)
NVIDIA L40S	102.697	0.510	387
NVIDIA A100	50.778	0.204	380
NVIDIA H100	32.345	0.151	334
NVIDIA H200	28.229	0.114	281

Table 5: ABCMB run times for forward mode auto-differentiation on outputs from ΛCDM with one massive neutrino.

modified to accommodate a number of scenarios without extensive knowledge of the inner workings of the code.

Both ΛCDM and new physics analyses stand to benefit from the differentiability of ABCMB, as efficient sampling algorithms capitalizing on gradient information become available for complex posteriors. The availability of stable functional gradients means ABCMB can also be used for Fisher forecasting. ABCMB's speed comes from JAX, not emulation, and so it can also be used in frequentist analyses that rely on dependable likelihood estimates at high resolution.

While ABCMB includes a number of desirable features for high-precision computation of the CMB power spectrum, there are many additional features that we leave to future work, some of which have already been mentioned. All calculations in ABCMB are currently performed in flat spacetime geometries, and in the synchronous gauge. No tensor perturbations or CMB B-modes are evaluated. The HyRex module enforces a strict assumption that the recombination of helium and hydrogen do not overlap—while this is a good assumption in most cosmologies, future work may include relaxing this assumption. HYREC-2 also natively supports time-varying electron mass and fine structure constant, which are eschewed in ABCMB for the time being. Finally, ABCMB is complete only to linear order—there is currently no implementation of HALOFIT [54] or other nonlinear corrections to the matter power spectrum. Curved spacetimes, the conformal Newtonian gauge, tensor perturbations, B-modes, extreme recombination histories, time-varying fundamental constants, and nonlinear corrections are left for future development. As it stands, however, we believe that ABCMB is already capable of performing many interesting cosmological analyses.

Future work will also include enhancements to the ABCMB performance. Better single-GPU run time can potentially be achieved by implementing batched solvers like those used in DISCO-DJ [33]. There are also a number of approximations used in CLASS and CAMB in the linear perturbations equations, such as the various fluid approximations for massless and massive neutrinos, that could help make ABCMB faster. In particular, the UFA and RSA discussed in section 4 are even known to reduce error in relativistic fluid perturbations, by decoupling the photon and neutrino Boltzmann hierarchies to avoid error incurred by introducing a cutoff [16]. For these reasons, we seek to implement UFA and RSA for improvements to both speed and accuracy.

ABCMB is already combined with LINX, so that it is trivial to perform fully differentiable joint CMB+BBN analyses including all BBN and Planck nuisance parameters, in both ΛCDM and new physics scenarios. Particularly exciting is the prospect of further combining ABCMB with DISCO-DJ II [35]; in this setup, the linear and nonlinear matter power spectra can be computed using the DISCO-DJ II N-body simulations, which can then be used to compute the CMB power spectrum using a ABCMB in a fully JAX, fully differentiable pipeline. ABCMB is written with new physics in mind, and can be used to perform self-consistent, state-of-the-art analyses of BSM particle physics scenarios, potentially combining with these other probes that have been developed in JAX. The existence of online learning emulator frameworks such as that in Ref. [29] also opens the possibility for fast analyses with ABCMB using ordinary, non-differentiable MCMC. These and other analyses are left to future work.

Acknowledgments

We would like to thank Yacine Ali-Haïmoud, I-Kai Chen, Colin Hill, Nanoom Lee, Florian List, Siddharth Mishra-Sharma, Clark Miyamoto, Caio Nascimento, Roman Scoccimarro, Neal Weiner,

Zachary Weiner, and Ian Williams for helpful discussions. We thank Skye Wanderman-Milne and Dan Foreman-Mackey for their assistance with JAX, especially with patching together diffrax solutions in different regimes, as well as other participants of the October 2024 JAXtronomy meeting. We thank Yacine Ali-Haïmoud, Julien Lesgourgues, Florian List, Marlon Josue Rivera Valladares, and Zachary Weiner for helpful feedback on a draft version of this manuscript and on our code. We thank Katelin Schutz and Marlon Josue Rivera Valladares for beta testing our code, and we thank Catherine Welch for verifying our installation instructions. We benefited tremendously from Hans A. Winther's cosmology lecture notes.[10] We use code packages numpy [55], scipy [56], diffrax [57], equinox [46], JAX [40, 41], matplotlib [58], interpax [59], and sphinx [60]. We use a custom interpolator developed by Yunfan Zhang.[11] The authors acknowledge the use of Claude Sonnet (Anthropic) to generate most docstrings in both ABCMB and HyRex.

C.G. is supported by the Office of High Energy Physics of the U.S. Department of Energy under contract DE-AC02-05CH11231. H.L. is supported by the U.S. Department of Energy under grant DE-SC0026297. In addition, H.L. is supported by the Cecile K. Dalton Career Development Professorship, endowed by Boston University trustee Nathaniel Dalton and Amy Gottleib Dalton. This work was supported in part through the NYU IT High Performance Computing resources, services, and staff expertise. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a Department of Energy User Facility (project m3166). This research used the Lawrencium computational cluster resource provided by the IT Division at the Lawrence Berkeley National Laboratory (Supported by the Director, Office of Science, Office of Basic Energy Sciences, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231).

A Other options

A.1 BBN

ABCMB natively includes three options for determining the helium-4 mass fraction,[12]

$$\mathrm{Y_{He}} = \frac{\rho_{\mathrm{He}}}{\rho_{\mathrm{H}} + \rho_{\mathrm{He}}}, \tag{A.1}$$

from user input. The first and simplest option is for the user to input their own float via `params['YHe']` at runtime.

The second option is analogous to the CLASS `'BBN'` option, where user input `params['omega_b']` and `params['Neff']` are used to look up a precomputed value of the helium-4 mass fraction from the public BBN code PArthENoPE [52, 61, 62]. We use an identical table for this option, reproducing the CLASS functionality. This option is triggered by the user setting `bbn_type='Table'` (any capitalization) when initializing an instance of `abcmb.main.Model`.

Finally, ABCMB also ships with LINX, a fast and differentiable JAX BBN code [39]. Combination with LINX allows the user to sample BBN nuisance parameters such as the neutron lifetime or individual reaction rates as a part of a joint CMB+BBN analysis. To run LINX to obtain the helium-4 abundance including effects from varying these nuisance parameters alongside cosmological parameters, the user must do the following:

1. Initialize an instance of `abcmb.main.Model` with `bbn_type='LINX'` (any capitalization). Optionally include a choice of nuclear reaction network by also initializing with `linx_reaction_net`, which can be set to any of the (case-sensitive) LINX reaction network options (`'np_only'`, `'key_PRIMAT_2023'` (default), `'full_PRIMAT_2023'`, `'key_PRIMAT_2018'`, `'key_YOF'`, or `'key_PArthENoPE'`). See ref. [39] for more information about each choice of reaction network.

[10] https://cmb.wintherscoming.no/index.php

[11] https://github.com/jax-ml/jax/issues/16182

[12] This is as opposed to the "helium-4 mass fraction" sometimes used in BBN analyses, defined as $\mathrm{Y_P} = \frac{4n_{\mathrm{He}}}{n_{\mathrm{B}}}$, n_{B} the total number of baryons.

2. Specify `params['Delta_Neff_init']`, and do not specify any `params['Neff']`, `params['N_nu_massless']`, or `params['T_nu_massless']`. $N_{\rm eff}$ in LINX is computed by including an inert relativistic species with energy density $\rho_{\rm ext}$. The user specifies the parameter

$$\Delta N_{{\rm eff},i} \equiv \frac{8}{7}\left(\frac{11}{4}\right)^{4/3}\frac{\rho_{{\rm ext},i}}{\rho_{\gamma,i}}, \tag{A.2}$$

where the subscript i indicates initial quantities in the BBN calculation. Since LINX starts its calculation well before electron/positron annihilation, this input parameter is *not* the same as $\Delta N_{\rm eff} = \frac{8}{7}\left(\frac{11}{4}\right)^{4/3}\left(\frac{\rho_{\rm ext}}{\rho_\gamma}\right)_0$, which instead is now a derived parameter.

3. Optionally specify BBN nuisance parameters. These can be accessed via `params['tau_n_fac']` (scaling of the input neutron lifetime from 879.4 s) and `params['nuclear_rates_q']` (a vector of scalings for each of the rates in the nuclear network).

See ref. [39] for more details about using LINX. These latter two BBN options will override any user input `params['YHe']`, and we do not recommend specifying a value for it if these options are used.

A.2 Parameters and run options

Below is a complete list of initialization options, input parameters, and derived parameters that ship with ABCMB as of `v0.3.1`. The can be specified individually, or packed into key/value pairs in a dictionary and unpacked during initialization of `Model`.

A.2.1 Initialization options

These options should be specified once as keyword arguments to `abcmb.main.Model` at initialization and used consistently for an entire analysis. Changing these options will trigger recompilation.

The following keyword arguments control input options:

- `use_LCDM_species`. Boolean; whether ABCMB should use the default baryon, photon, massless neutrinos, CDM, and dark energy fluids in computation. Setting this parameter to `True` instructs ABCMB to use all of these species, while setting it to `False` uses none of them. Defaults to `True`. This option should be used if the user wishes to replace any of the standard species with a modified species, e.g. interacting dark matter over CDM. Note that ABCMB expects each cosmology to contain at least some version of the five ΛCDM fluids, and will fail if any are missing.
- `input_tau_reion`. Boolean; whether the optical depth to reionization $\tau_{\rm reion}$ is an input parameter, or if it is computed given the redshift to reionization $z_{\rm reion}$. If set to `True`, the initialized `Model` should always be called with $\tau_{\rm reion}$ rather than $z_{\rm reion}$, or else a default $\tau_{\rm reion}$ value is used over the user input. Defaults to `True`.

The following keyword arguments control output options:

- `l_min`. The minimum ℓ at which to output CMB power spectra. Defaults to 2.
- `l_max`. The maximum ℓ at which to output CMB power spectra. Defaults to 2500.
- `k_max`. The maximum k at which to output the matter power spectrum. Defaults to $0.5\,{\rm Mpc}^{-1}$.
- `lensing`. Boolean; whether to compute corrections from lensing. Defaults to `False`.

These keyword arguments control how ABCMB computes (or does not compute) $Y_{\rm He}$:

- `bbn_type`. Specifies how to compute the primordial helium mass fraction. Options are (any capitalization) `'Table'` to use precomputed yields, `'LINX'` to compute abundances given other input parameters using LINX, or `None` to use user input $Y_{\rm He}$. Defaults to `None`.

- `linx_reaction_net`. If `bbn_type` is set to use LINX, specify a BBN reaction network. Options are `'np_only'`, `'key_PRIMAT_2023'`, `'full_PRIMAT_2023'`, `'key_PRIMAT_2018'`, `'key_YOF'`, or `'key_PArthENoPE'`. See ref. [39] for more information about each choice of reaction network. Defaults to `'key_PRIMAT_2023'`.

These keyword arguments control cutoffs for Boltzmann hierarchies:

- `l_max_g`. The number of moments l to use in the Boltzmann hierarchy for photons for temperature perturbations. Defaults to 12.
- `l_max_pol_g`. The number of moments l to use in the Boltzmann hierarchy for photons for polarization. Defaults to 10.
- `l_max_massless_nu`. The number of moments l to use in the Boltzmann hierarchy for massless neutrinos. Defaults to 17.
- `l_max_massive_nu`. The number of moments l to use in the Boltzmann hierarchy for massive neutrinos. Defaults to 17.

The wavenumbers k used by ABCMB to compute perturbations are unevenly spaced, in order to provide adequate resolution at the most critical regimes without becoming too memory intensive. We employ the same scheme as implemented in CLASS, reproducing similar formulae here so that we can indicate which parameters are controlled by user input. The k-grid is computed according to

$$\begin{aligned} k_{n+1} &= k_n + \Delta k(k_n) \cdot s(k_n) \\ \Delta k(k_n) &= \left(k_{\rm step}^{\rm super} + \frac{1}{2} \tanh \left(\frac{k_n - k_{\rm rec}^{\rm fid}}{k_{\rm rec}^{\rm fid} k_{\rm step}^{\rm transition}} + 1 \right) \left(k_{\rm step}^{\rm sub} - k_{\rm step}^{\rm super} \right) \right) k_{\rm rec}^{\rm fid} \end{aligned} \tag{A.3}$$

$$s(k_n) = \frac{\frac{k_n^2}{\left(H_0^{\rm fid}\right)^2} + 1}{\left(\frac{k_n^2}{\left(H_0^{\rm fid}\right)^2} + \frac{1}{k_{\rm step}^{\rm super,reduction}} \right)}, \tag{A.4}$$

where

$$k_{\rm rec}^{\rm fid} = \frac{2\pi}{r_s^{\rm rec,fid}}$$

and $k_{\rm step}^{\rm sub}$, $k_{\rm step}^{\rm super}$, $k_{\rm step}^{\rm transition}$, $k_{\rm step}^{\rm super,reduction}$, $r_s^{\rm rec,fid}$, and $H_0^{\rm fid}$ are user input. In order to prevent recompilation each time the user provides new input parameters, the same k grid must be used for each call of an initialized model. Therefore the grid specification requires fiducial values for the wavenumber corresponding to the scale of the sound horizon at recombination $k_{\rm rec}^{\rm fid}$, the sound horizon itself at recombination $r_s^{\rm rec,fid}$, and the Hubble constant $H_0^{\rm fid}$, rather than updating the grid each time the user inputs a new value of e.g. H_0.

The user also defines minimum and maximum k via

$$k_{\rm min} = \frac{k'_{\rm min} \tau_0}{\tau_0^{\rm fid}} \tag{A.5}$$

$$k_{\rm max} = \frac{k'_{\rm max} \tau_0}{\ell_{\rm max} \tau_0^{\rm fid}} \tag{A.6}$$

where the combinations $k'_{\rm min}\tau_0$ and $\frac{k'_{\rm max}\tau_0}{\ell_{\rm max}}$ are input by the user. The fiducial conformal time $\tau_0^{\rm fid}$ again takes on some fiducial value to prevent recompilation, which can be specified by the user. Input arguments are defined as:

- `k_step_sub`. $k_{\rm step}^{\rm sub}$ in eq. (A.3). Dimensionless parameter describing the number of periods of acoustic oscillation at decoupling (given by $\frac{2\pi}{r_s^{\rm rec}}$) by which to step in k for modes inside the horizon at decoupling. Defaults to 5×10^{-2}.

- `k_step_super`. $k_{\rm step}^{\rm super}$ in eq. (A.3). Dimensionless parameter describing the number of periods of acoustic oscillation at decoupling by which to step in k for modes outside the horizon at decoupling. Defaults to 2×10^{-3}.
- `k_step_transition`. $k_{\rm step}^{\rm transition}$ in eq. (A.3). Modulates a smooth transition between `k_step_sub` and `k_step_super`. Defaults to 2×10^{-1}.
- `k_step_super_reduction`. $k_{\rm step}^{\rm super,reduction}$ in eq. (A.4). For the very largest scales, `k_step_super` is reduced by this amount (as can be seen by taking a limit of eq. (A.4)). Defaults to 1×10^{-1}.
- `k_min_tau0`. $k'_{\rm min} * \tau_0$ in eq. (A.5); minimum k scaled by an initial conformal time parameter τ_0. Defaults to 1×10^{-1}. This sets the internal parameter `k_min` to the right-hand side of eq. (A.5).
- `k_max_tau0_over_l_max`. $\frac{k'_{\rm max}\tau_0}{\ell_{\rm max}}$ in eq. (A.6); maximum k scaled by an initial conformal time parameter τ_0 per maximum ℓ. Defaults to 1.8. Sets the internal parameter `k_max_cmb` to the right-hand side of eq. (A.6).
- `H0_fid`. A fiducial value for the Hubble constant used to compute the k grid, as shown in eq. (A.4). Defaults to $2.255560 \times 10^{-4}{\rm Mpc}^{-1}$.
- `rs_rec_fid`. A fiducial value for the sound horizon at recombination, used to specify $k_{\rm rec}^{\rm fid}$. Defaults to 144.6279 Mpc.
- `tau0_fid`. A fiducial value for conformal time today, as in eqs. (A.5) and (A.6). Defaults to 1.418668×10^4Mpc.

A separate k-grid, indicated below by $\tilde{k}$, is used for integrating transfer functions, in particular

$$\tilde{k}_{n+1} = \tilde{k}_n + \frac{\tilde{k}_{\rm period}\tilde{k}_{\rm transfer}^{\rm linstep}\tilde{k}_n}{\tilde{k}_n + \frac{\tilde{k}_{\rm transfer}^{\rm linstep}}{\tilde{k}_{\rm transfer}^{\rm logstep}}\left(\frac{1}{\rm Mpc}\right)}, \tag{A.7}$$

where

$$\tilde{k}_{\rm period} = \frac{2\pi}{\tau_0^{\rm fid} - \tau_{\rm rec}^{\rm fid}}. \tag{A.8}$$

$\tau_0^{\rm fid} - \tau_{\rm rec}^{\rm fid}$ is the lookback time to recombination. $\tau_0^{\rm fid}$ is the same parameter that appears in eqs. (A.5) and (A.6). $\tau_{\rm rec}^{\rm fid}$, $\tilde{k}_{\rm transfer}^{\rm linstep}$, and $\tilde{k}_{\rm transfer}^{\rm logstep}$ are user-defined. This parametrization ensures logarithmic growth for small k and linear growth for large k. The grid is computed between `k_min` and `k_max_cmb`. The input arguments are

- `k_transfer_linstep`. $\tilde{k}_{\rm transfer}^{\rm linstep}$ in eq. (A.7); the size of the step in $\tilde{k}$ deep in the linear regime, for large $\tilde{k}$. Defaults to 4.5×10^{-1}.
- `k_transfer_logstep`. $\tilde{k}_{\rm transfer}^{\rm logstep}$ in eq. (A.7); $\log\left(\tilde{k}_{\rm transfer}^{\rm logstep}\right)$ is the size of the step in $\log\tilde{k}$ deep in the logarithmic regime, for small $\tilde{k}$. Defaults to 170.
- `tau_rec_fid`. $\tau_{\rm rec}^{\rm fid}$ in eq. (A.8). Defaults to 281.040 565 Mpc.

As discussed above, our construction of these grids differs from the approach of CLASS by the inclusion of fiducial cosmological parameters. This is to ensure fixed array sizes at compile time and prevent recompilation. These static fiducial parameters are sufficient to reproduce the CLASS result within $\pm 5\sigma$ of the *Planck* best-fit cosmological parameters (see section 4). If the user is interested in parameter values that differ from these fiducial values by $\mathcal{O}(1)$, then new $r_s^{\rm rec,fid}$, $H_0^{\rm fid}$, $\tau_0^{\rm fid}$, or $\tau_{\rm rec}^{\rm fid}$ may need to be computed and input by the user at initialization.

Otherwise, the keyword arguments listed above that control the perturbations and transfer k grids have no impact on the underlying physics. They are merely defined such that various integrals over k can be done efficiently while still retaining as much precision as possible. The scheme in

eqs. (A.3)-(A.8) that we adapt from CLASS is optimized for precisely this purpose, and we encourage the user not to modify them without a clear reason.

The following keyword argument sets the characteristics of the primordial power spectrum:

- `k_pivot`. The pivot scale; see eq. (B.19). Defaults to $0.05\,\mathrm{Mpc}^{-1}$.

We adopt the CLASS strategy for determining the initial scale factor $a_i(k)$ at which to begin integrating perturbations with wavenumber k. $a_i(k)$ has to satisfy two conditions. First, it must be early enough such that the baryons and photons are tightly coupled, i.e. a_i has to be sufficiently small such that the ratio

$$R_{\mathrm{tc}} = \mathcal{H}(a_i)\tau_c(a_i) \ll 1, \tag{A.9}$$

where $\tau_c = (an_e\sigma_T)^{-1}$ is the Compton scattering time, with σ_T the Thomson cross section. This relationship is k-independent, and so sets a maximum value for a_i for all modes.

Second, for perturbations on large scales, we need to ensure that integration begins while the mode is superhorizon, i.e. for each k, we must choose a_i such that the ratio

$$R_{\mathrm{large}} = \frac{k}{\mathcal{H}(a_i)} \ll 1, \tag{A.10}$$

which sets a value of a_i for each k-mode.

Both R_{tc} and R_{large} are constants much smaller than 1 that can be specified as user inputs in order to guarantee an appropriate choice of a_i. Given these two variables, ABCMB solves for the appropriate a_i for each k, i.e. a_i such that $R_{\mathrm{tc}} = \mathcal{H}(a_i)\tau_c(a_i)$ and $R_{\mathrm{large}} = k/\mathcal{H}(a_i)$, and chooses the smaller value of a_i as the initial scale factor to start the integration at.

In the code, these initial conditions are set by the following two keyword arguments:

- `R_tc`. R_{tc} in eq. (A.9). For small k modes this condition will be satisfied first and will provide the initial condition in $\ln a$ for integration. Defaults to 1.5×10^{-3}.
- `R_large`. R_{large} in eq.(A.10). For large k modes this condition will be satisfied first and will provide the initial condition in $\ln a$ for integration. Defaults to 7×10^{-2}.

These inputs determine settings for the differential equation solver used for perturbations:

- `max_steps_PE`. Maximum number of steps used to solve the differential equation describing perturbations for a single k mode. Defaults to 2048.
- `k_split_PE`. Value of k at which to shift tolerances in solving the differential equation differential equation describing perturbations for a single k mode (see below). Defaults to 1×10^{-2} Mpc^{-1}.
- `rtol_small_k_PE`. The relative tolerance used by the perturbations differential equation solver below `k_split_PE`. Defaults to 1×10^{-5}.
- `rtol_large_k_PE`. The relative tolerance used by the perturbations differential equation solver above `k_split_PE`. Defaults to 1×10^{-4}.
- `atol_small_k_PE`. The absolute tolerance used by the perturbations differential equation solver below `k_split_PE`. Defaults to 1×10^{-10}.
- `atol_large_k_PE`. The absolute tolerance used by the perturbations differential equation solver above `k_split_PE`. Defaults to 1×10^{-6}.
- `pcoeff_PE`. The `pcoeff` value used in the adaptive stepsize controller for the differential equation solved for perturbations; see diffrax [57] documentation for more information. Defaults to 0.25.
- `icoeff_PE`. The `icoeff` value used in the adaptive stepsize controller for the differential equation solved for perturbations; see diffrax documentation for more information. Defaults to 0.8.

- `dcoeff_PE`. The `dcoeff` value used in the adaptive stepsize controller for the differential equation solved for perturbations; see diffrax documentation for more information. Defaults to 0.

These input parameters control the overall scaling of the different physical contributions to the CMB temperature transfer function; they can be adjusted to highlight the various contributions for pedagogical purposes:

- `scale_sw`. Scaling of the Sachs-Wolfe contribution to the CMB temperature transfer function; 0 shuts this contribution off. Defaults to 1.
- `scale_isw`. Scaling of the integrated Sachs-Wolfe contribution to the CMB temperature transfer function; 0 shuts this contribution off. Defaults to 1.
- `scale_dop`. Scaling of the Doppler contribution to the CMB temperature transfer function; 0 shuts this contribution off. Defaults to 1.
- `scale_pol`. Scaling of the polarization contribution to the CMB temperature transfer function; 0 shuts this contribution off. Defaults to 1.

Finally, this input argument controls the method used for autodifferentiation, should the user request a gradient:

- `adjoint`. The method to use for computing adjoints through `diffrax` differential equations. Defaults to `diffrax.ForwardMode`, and `diffrax.RecursiveCheckpointAdjoint` is recommended for reverse AD.

A.2.2 Cosmological parameters

The following input parameters can be specified by the user after initialization, at runtime, through a dictionary passed to the initialized `Model`. If unspecified, default values will be used.

- `params['omega_b']`. Total energy density in baryons, compared to the critical matter density and multiplied by h^2. Defaults to 0.02237.
- `params['omega_cdm']`. Total energy density in CDM, compared to the critical matter density and multiplied by h^2. Defaults to 0.120.
- `params['Neff']`. As defined in eq. (2.4). This parameter cannot be specified simultaneously with `params['N_nu_massless']`; see section 2.4.3. This parameter cannot be specified if LINX options are used for BBN. Defaults to 3.044.
- `params['h']`. Hubble parameter. Defaults to 0.6736.
- `params['A_s']`. Amplitude of fluctuations. Defaults to 2.1×10^{-9}.
- `params['n_s']`. Scalar spectrum power law index. Defaults to 0.9649.
- `params['YHe']`. Primordial helium mass fraction, $\frac{\rho_{\mathrm{He}}}{\rho_{\mathrm{H}}+\rho_{\mathrm{He}}}$. Defaults to 0.245 if `Model` is not initialized with `bbn_type`.
- `params['TCMB0']`. CMB temperature today, in eV. Defaults to $2.34865418 \times 10^{-4}$.
- `params['N_nu_massless']`. Massless neutrino energy density as given by eq. (2.5); `params['N_nu_massless']` is $N_{\nu,\mathrm{massless}}$. This parameter cannot be specified simultaneously with `params['Neff']`, and is adjusted internally according to section 2.4.3 depending on input value of `params['Neff']`, if `params['Neff']` is specified.
- `params['T_nu_massless']`. The *ratio* of the massless neutrino temperature ($T_{\nu,\mathrm{massless}}$ in eq. (2.5)) to the photon temperature, at late times. Defaults to 0.71636856 to capture effects from non-instantaneous neutrino decoupling.

- `params['N_nu_massive']`. Number of massive neutrino species. Defaults to 0.
- `params['m_nu_massive']`. Mass of massive neutrino species, in eV. Defaults to 0.06 (but not used if `params['N_nu_massive']` is 0). All massive neutrinos have the same mass.
- `params['T_nu_massive']`. The temperature *ratio* of massive neutrinos to the CMB temperature, at late times. Defaults to 0.71611, as per ref. [51](see section 2.4.3).
- `params['tau_reion']`. Optical depth to reionization, defined as

$$\tau_{\rm reion} = \int d\ln a \frac{a\sigma_T n_H x_e^{\rm reion}}{\mathcal{H}}\,, \tag{A.11}$$

  where $x_e^{\rm reion}$ denotes the free electron fraction coming purely from reionization[13], as given by equations B.6 and B.8. Defaults to 0.05430842. This integrated quantity has a one-to-one correspondence with the parameter $z_{\rm reion}$, the redshift of hydrogen and HeI reionization, once the remaining four reionization parameters are fixed. If $\tau_{\rm reion}$ is passed in, the appropriate value for $z_{\rm reion}$ is found with eq. (A.11) using a root finder. For this reason in ABCMB only one of $\tau_{\rm reion}$ or $z_{\rm reion}$ can be specified, and to switch between these options one should set the keyword argument `input_tau_reion` to `True` or `False` appropriately.
- `params['z_reion']`. Redshift of hydrogen and HeI reionization (see eq. (B.6)). Defaults to 7.67. Set to 0 to turn off this reionization. If specified, ABCMB will compute the integral (A.11) to find the optical depth to reionization automatically. As discussed above, only one of $\tau_{\rm reion}$ or $z_{\rm reion}$ can be specified, and to switch between these options one should set the keyword argument `input_tau_reion` to `True` or `False` appropriately.
- `params['Delta_z_reion']`. Width of the tanh step for hydrogen and HeI reionization (see eq. (B.6)). Defaults to 0.5. Set to 0 to turn off this reionization.
- `params['z_reion_He']`. Redshift of HeII reionization (see eq. (B.8)). Defaults to 3.5. Set to 0 to turn off this reionization.
- `params['Delta_z_reion_He']`. Width of the tanh step for HeII reionization (see eq. (B.8)). Defaults to 0.5. Set to 0 to turn off this reionization.
- `params['exp_reion']`. Exponent $\beta_{\rm reion}$ that appears in the tanh parametrization for reionization (see eq. (B.7)). Defaults to 1.5.
- `params['nuclear_rates_q']`. Optional; used only if `bbn_type='LINX'`. An array of scalings of nuclear rates in the LINX BBN reaction network. This array should include one value q_i for each reaction in the network (12 reactions if a "key" network is used). Each q_i which scales the median values for its corresponding reaction rates $\overline{r}_i$ according to $\log r_i(T) = \log \overline{r}_i(T) + q_i\sigma_i(T)$, where $r_i(T)$ is the rate that LINX will use for the specified reaction. q_i is a unit Gaussian random variable.
- `params['tau_n_fac']`. Optional; used only if `bbn_type='LINX'`. This input parameter is a multiplicative scaling of the fiducial neutron lifetime 879.4 s.

Finally, given the set of user input parameters, the following parameters are derived at runtime. User inputs will be overwritten, and therefore should not be input.

- `params['omega_m']`. The total matter density compared to the critical matter density and multiplied by h^2. This is computed via summing over all fluid energy densities whose flag `is_matter=True`.

[13]The tiny amount contributed by the free electrons after recombination is neglected in this integral. This is a good assumption in ΛCDM, but new physics such as decaying dark matter can add an early reionization component. We leave the generalization to this class of models to future work.

- `params['R_b']`. The baryon fraction. Given by `params['omega_b']` / `params['omega_m']`.
- `params['H0']`. Value of the Hubble constant today in km/s/Mpc. Given by `params['h']`$\times 100\,$km/s/Mpc.
- `params['omega_r']`. Total energy density in radiation, compared to the critical matter density and multiplied by h^2. This is calculated as $\frac{8\pi G\rho_R a_i^4 h^2}{3H_0^2}$, where the total radiation energy density ρ_R is computed by ABCMB given all input fluids and their energy densities at some very small scale factor a_i, and G is Newton's constant. This assumes radiation domination at a_i.
- `params['R_nu']`. The fraction of radiation in neutrinos. Given by ρ_ν/ρ_R, where ρ_R is computed as described above and ρ_ν is similarly computed at some very small scale factor a_i. This parameter determines adiabatic initial conditions for perturbations; while modes inside the horizon during e^+e^- annihilation should in principle be described by a different `R_nu`, the effects of neglecting this correction do not affect modes we will measure in the foreseeable future.
- `params['om']`. Defined as $\omega \equiv \frac{\Omega_m}{\sqrt{\Omega_r}} H_0$, important in determining the conformal time and the scale factor at early times, and used in setting adiabatic initial conditions for various fluids. See CAMB notes[14] for more details.
- `params['omega_Lambda']`. Total energy density in dark energy, compared to the critical matter density and multiplied by h^2. Inferred from other densities such that it is given by `params['h']`^{2} $-$ `params['omega_r']` $-$ `params['omega_m']`.

B Physics implementation in ABCMB

In this appendix, we detail the physics implemented in the various ABCMB modules. We begin with a detailed overview of the companion code HyRex, used for computing the free electron fraction and matter temperature during recombination. We then move on to the calculation of cosmological perturbations and spectra.

B.1 Recombination

When the user calls ABCMB, the code first computes the free electron fraction x_e and the matter temperature T_m as a function of scale factor a in the HyRex module. These are used as background quantities in the evolution of the perturbation equations used to compute the CMB power spectrum.

We follow the implementation in HYREC-2, whose accuracy exceeds that of RECFAST [63] and rivals that of the earlier code HYREC [64], while running in a fraction of HYREC's run time [38]. HYREC-2 reduces computational overhead by revising the Effective Multi-Level Atom (EMLA) [65] down to an effective 4-level atom, where corrections to the Lyman-α escape rate are precomputed from HYREC. These corrections have predictable scalings with cosmological parameters and therefore do not need to be recomputed for new sets of parameters. We follow this procedure as well, rewriting HYREC-2 in a differentiable and user-friendly JAX implementation, and also including a simple reionization model. This calculation separates helium recombination from hydrogen recombination (as is done in HYREC-2), which are further separated into various thermodynamic phases.

Throughout this section, fractional abundances x_i are always the number density of the subscripted state over the number density of all hydrogen states, $x_i \equiv n_i/n_H$. Calculations in HyRex use units eV for energies and temperatures, eV/cm^3 for energy densities, and s for time. Hubble is computed in s^{-1}.

[14] https://cosmologist.info/notes/CAMB.pdf

B.1.1 Helium recombination

We follow the treatment of helium recombination in HYREC [64], which occurs in two phases: first, doubly-ionized helium HeIII recombines to singly-ionized helium HeII, and then, at a later time, that singly-ionized helium recombines to neutral helium HeI.

HeII recombination is fast compared to the expansion rate, so we begin by tracking the fractions of doubly- and singly-ionized helium in Saha equilibrium, x_{HeIII} and x_{HeII}, respectively:

$$\begin{aligned}\frac{x_{\mathrm{HeIII}}\,(1+f_{\mathrm{He}}+x_{\mathrm{HeIII}})}{f_{\mathrm{He}}-x_{\mathrm{HeIII}}} &= \frac{(2\pi\mu_{e/\mathrm{HeIII}}T)^{3/2}}{h_P^3 n_{\mathrm{H}}}e^{-E_{I_2}/T},\\ x_{\mathrm{HeII}} &= f_{\mathrm{He}}-x_{\mathrm{HeIII}}\\ x_e &= 1+f_{\mathrm{He}}+x_{\mathrm{HeIII}}\end{aligned}$$

where E_{I_2} is the second ionization energy of helium, T is the radiation temperature, $\mu_{e/\mathrm{HeIII}}$ is the reduced mass of the electron-HeIII system, h_P is Planck's constant (to be disambiguated from the dimensionless equivalent of the Hubble constant h), and n_{H} is the hydrogen number density. f_{He} is the number abundance of helium nuclei relative to hydrogen. Solving the first of these equations for x_{HeIII} provides all of the information needed to track x_e in this epoch, and so each of these fractions is tracked from when integration starts ($z = 8000$ by default) until most HeIII has recombined to HeII ($x_{\mathrm{HeIII}} < 10^{-9}$ by default).

After crossing this threshold, we switch to a post-Saha equilibrium expansion to describe fast recombination to HeI, given by

$$\begin{aligned}\frac{x_{\mathrm{HeII}}(1+x_{\mathrm{HeII}})}{f_{\mathrm{He}}-x_{\mathrm{HeII}}} &= 4\frac{(2\pi\mu_{e/\mathrm{HeII}}T)^{3/2}}{h_P^3 n_{\mathrm{H}}}e^{-E_{I_1}/T}\\ x_e^{\mathrm{Saha}} &= 1+x_{\mathrm{HeII}}\\ x_e &= x_e^{\mathrm{Saha}}+\Delta x_e,\end{aligned} \tag{B.1}$$

where E_{I_1} is the first ionization energy of helium and $\mu_{e/\mathrm{HeII}}$ is the reduced mass of the electron-HeII system. x_e^{Saha} is the Saha equilibrium value for x_e, which differs from the true value of x_e by a small amount $\Delta x_e = \frac{dx_e^{\mathrm{Saha}}}{dt}/\frac{\partial \dot{x}_e}{\partial x_e}\big|_{x_e^{\mathrm{Saha}}}$.

A large Δx_e (10^{-5} by default, which is more precise than the default HYREC-2 threshold 5×10^{-4}) indicates a significant departure from Saha equilibrium. This suggests slower recombination of helium and requires solving for recombination of x_{HeII} to neutral x_{HeI} in full detail, including effects from radiative transfer to hydrogen. This involves the construction and solution of an ODE, which is provided in ref. [64]. We solve this ODE with diffrax [57], terminating when x_{HeII} drops below 10^{-4}.

This calculation describes the evolution of x_e down to a redshift of about 1700 (with the exact value depending on the user's specified cosmology). This history is returned to the main HyRex module, and its stopping redshift is used as an initial condition for hydrogen recombination. We assume these epochs do not overlap—this is a reasonable assumption, given that hydrogen recombination tends not to begin in earnest until redshift $z \sim 1600$, as verified in ref. [64].

This computation is summarized in figure 9, where we delineate the three regimes described above.

B.1.2 Hydrogen recombination

The calculation of hydrogen recombination requires a good deal more detail and care to obtain the precision required by the CMB, as the CMB probes hydrogen recombination directly (helium recombination primarily affects the Silk damping scale [66]). Hydrogen that recombines directly to the ground state emits high-energy photons that quickly lead to another ionization event, leading to no net change in ionization. Therefore, recombination to excited states dominates ("case B" recombination). The escape rate of Lyman emissions from hydrogen recombining to excited states and de-exciting therefore must be modeled carefully, and transition rates from other excited states must also be taken into account. Meanwhile, the hydrogen $2s$ state is metastable, requiring emission of two

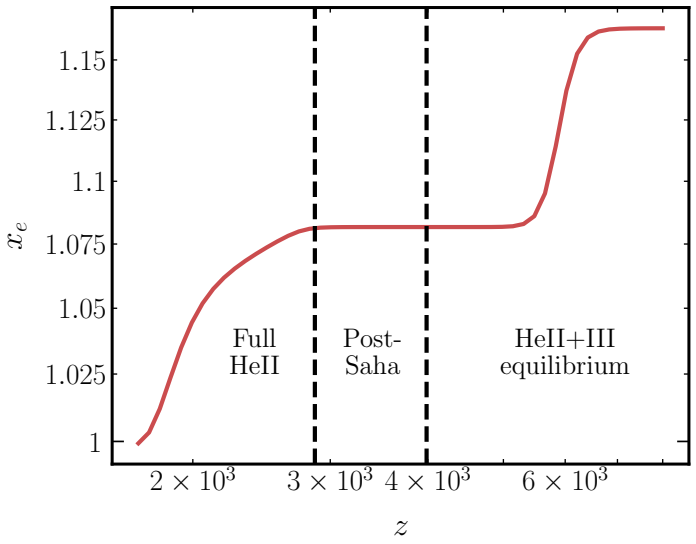

Figure 9: The free electron fraction x_e as a function of redshift z during helium recombination only, as computed by HyRex. The relevant cosmological parameters chosen are $Y_{He} = 0.245$, $N_{eff} = 3.044$, $h = 0.7$, $\Omega_b h^2 = 0.0224$, and $\Omega_{CDM} h^2 = 0.12$, and the dashed lines separate different solving regimes as described in the main text.

photons to decay to the ground state, further complicating the set of equations that must be solved. Detailed discussions of these phenomena are available in e.g. Refs. [67–71].

As in HyRec [64], the hydrogen recombination is broken into four parts: a post-Saha equilibrium phase, an EMLA + two-photon processes phase, an EMLA phase where two-photon $2s \to 1s$ decays are no longer active, and a phase described by the simplified three-level atom (TLA) model.

At early times, during helium recombination, hydrogen remains in Saha equilibrium. Though hydrogen departs from Saha equilibrium while helium is still recombining, the Saha-equilibrium free electron fraction still makes for a reasonable initial condition to begin integrating the post-Saha equilibrium phase, incurring errors only as large as about one part in 10^4, as verified by ref. [64]. Our initial condition is therefore

$$x_e^{\rm Saha} = \frac{\sqrt{s^2 + 4s} - s}{2}, \tag{B.2}$$

where

$$s = \frac{(2\pi\mu_{e/\rm H} T)^{3/2}}{h_P^3 n_{\rm H}} e^{-E_R/T},$$

E_R is the Rydberg constant, and $\mu_{e/\rm H}$ is the reduced mass for the hydrogen/electron system. We then compute x_e as given by eq. (B.1) (with $x_e^{\rm Saha}$ given by eq. (B.2)). Note two-photon processes are already relevant at this stage, and so the derivatives defining Δx_e now depend on the effective recombination coefficients, with $\dot{x}_e$ given by [38]

$$\dot{x}_e = -\sum_{\ell=s,p} C_{2\ell} \left(n_{\rm H} x_e^2 \mathcal{A}_{2\ell} - g_{2\ell} x_{1s} e^{-E_{21}/T_\gamma} \mathcal{B}_{2\ell} \right), \tag{B.3}$$

where $\mathcal{A}_{2\ell}$, $\mathcal{B}_{2\ell}$ are the effective recombination and effective photo-ionization coefficients, respectively, $C_{2\ell}$ is the generalized Peebles-C factor, and $g_{2\ell}$ counts the degrees of freedom of the $2s$ and $2p$ hydrogen states, which are respectively 1 and 3. In HYREC-2, effects of two-photon processes are fully captured in $C_{2\ell}$.

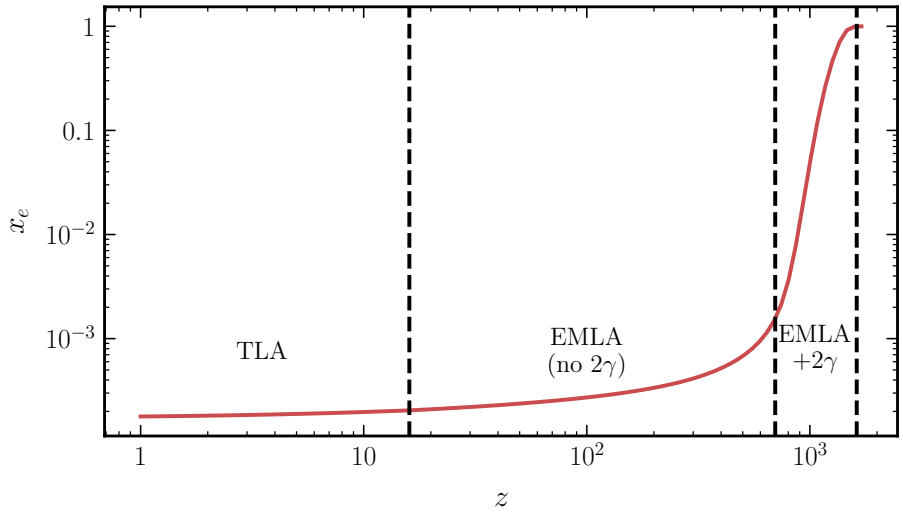

Figure 10: The free electron fraction x_e as a function of redshift z after helium recombination and before reionization, as computed by HyRex. The relevant cosmological parameters chosen are $Y_{He} = 0.245$, $N_{\rm eff} = 3.044$, $h = 0.7$, $\Omega_b h^2 = 0.0224$, and $\Omega_{\rm CDM} h^2 = 0.12$. Dashed lines separate Post-Saha (highest z), the EMLA + two-photon phase, the EMLA phase after two-photon processes shut off, and the final TLA phase at low redshifts.

The post-Saha phase is short, as shown by the rightmost segment of figure 10. However, it is critical for numerical stability; the EMLA phases are stiff, and so a brief post-Saha phase is required to move into a more stable region.

We move to the EMLA phases once Δx_e exceeds 10^{-5} by default. EMLA is again described by an ODE provided in ref. [65], though its form is dramatically simplified in HYREC-2 due to the use of precomputed effective recombination coefficients. We solve eq. (B.3), the EMLA including two-photon processes down to redshift $z = 700$, after which two-photon processes no longer contribute [64]. During this epoch the matter temperature T_m departs from the CMB temperature $T_{\rm CMB}$ as the baryons and photons decouple, and the matter temperature is tracked via

$$T_m = T_{\rm CMB}\left(1 - \frac{H}{\Gamma_C}\right), \tag{B.4}$$

where Γ_C is the Compton scattering rate. With σ the Stefan-Boltzmann constant, $\sigma_{\rm Thomson}$ the Thomson scattering cross section, m_e the electron mass, and $f_{\rm He}$ the helium number fraction, Γ_C is given by $\frac{x_e}{1+f_{\rm He}+x_e}\frac{32\sigma T_{\rm CMB}^4 \sigma_{\rm Thomson}}{3m_e}$.

After a redshift of 700 we finally switch a simplified ODE for the EMLA where two-photon rates are set to zero. The matter temperature in this regime evolves according to

$$\frac{dT_m}{d\ln a} = \frac{-2HT_m + \Gamma_C(T_{\rm CMB} - T_m)}{H}. \tag{B.5}$$

The effective recombination coefficients in HYREC-2 are only tabulated down to a redshift of about 15. Since x_e is much less than 1 in this regime and reionization processes dominate, we follow the procedure in HYREC-2 and describe recombination with a hydrogen TLA [67, 68].

This computation is summarized in figure 10, with each of the epochs described above separated by dashed lines.

After each of these elements is computed, the resulting x_e, T_m and log scale factor $\ln a$ are returned back up to the main HyRex module. They can be used as standalone outputs for analyses

sensitive to recombination, or passed up to the ABCMB Einstein-Boltzmann solver and used to compute the matter and CMB power spectra.

B.2 Reionization

Once HyRex computes the recombination history, ABCMB patches a reionization correction on to the resulting x_e. As in ref. [72], and as implemented in CAMB and as an option in CLASS, we use a simple tanh model for hydrogen and HeI reionization,

$$x_e^{\text{reion}} = \frac{1 + f_{\text{He}}}{2}\left(1 + \tanh\left(\frac{y_{\text{reion}} - y}{\Delta y_{\text{reion}}}\right)\right), \tag{B.6}$$

where

$$\begin{aligned} y &= (1+z)^{\beta_{\text{reion}}} \\ y_{\text{reion}} &= (1+z_{\text{reion}})^{\beta_{\text{reion}}} \\ \Delta y_{\text{reion}} &= \beta_{\text{reion}} \Delta z_{\text{reion}} (1+z_{\text{reion}})^{\beta_{\text{reion}}-1}, \end{aligned} \tag{B.7}$$

to the end of the x_e history. The redshift of reionization z_{reion}, reionization width Δz_{reion} and the exponent β_{reion} (usually 3/2) can be specified by the user.

We include a second reionization step for reionization of HeII to HeIII:

$$x_e^{\text{HeII reion}} = \frac{f_{\text{He}}}{2}\left(1 + \tanh\left(\frac{z_{\text{HeII reion}} - z}{\Delta z_{\text{HeII reion}}}\right)\right), \tag{B.8}$$

where the redshift $z_{\text{HeII reion}}$ and width $\Delta z_{\text{HeII reion}}$ are user input; they are independent from the hydrogen reionization parameters.

We do not track the matter temperature during reionization. While we might track the small effect on the matter temperature from Compton scattering of electrons with CMB photons, detailed simulations are required to model the effects of photoheating from high-energy free electrons, which is the dominant effect (see e.g. refs. [73, 74]). Since these effects are not relevant for accurate CMB calculations, we eschew them entirely.

The full x_e output of HyRex, including the reionization model appended to the solution by ABCMB, is shown in figure 11.

B.3 Perturbations

After obtaining the electron fraction and matter temperature histories described in the previous section during the initialization of a `Background` object, ABCMB turns to the `Perturbations` module. Its purpose is to concurrently evolve the linear perturbations for each fluid, starting from the initial conditions specified by their respective fluid modules. At the end of its computation, the evolution histories of every fluid is packaged into a table over a grid of time and wavenumbers, and forwarded to the spectrum module for computing power spectra.

This section describes the physical and computational approach in general; specific perturbations equations and other fluid-specific quantities are provided in more detail in appendix C. Units used by ABCMB to implement what follows are eV for energy, eV/cm^3 for energy density, Mpc for time and length, and Mpc^{-1}for conformal Hubble.

B.3.1 Linear perturbations

ABCMB works entirely in the synchronous gauge and begins with adiabatic initial conditions for all fluids. The modularity of fluid species makes it simple to replace the perturbation equations with the Newtonian or other gauges, as well as isocurvature perturbations, though these are not currently implemented in ABCMB.

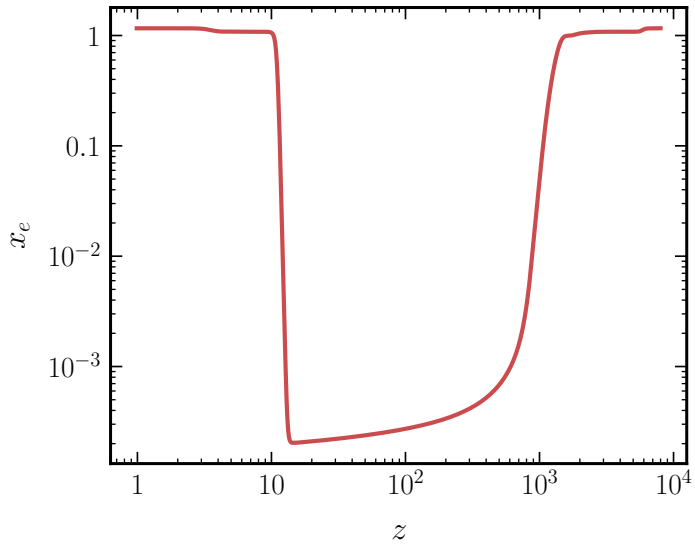

Figure 11: The full x_e history as computed by HyRex, including helium recombination, hydrogen recombination, and reionization. The relevant input parameters chosen to produce this result are $Y_{He} = 0.245$, $N_{eff} = 3.044$, $h = 0.7$, $\Omega_b h^2 = 0.0224$, $\Omega_{CDM} h^2 = 0.12$, $z_{reio} = 11$, and $\Delta z_{reio} = 0.5$.

For the perturbation equations currently implemented, we follow the formalism of ref. [47]. In the synchronous gauge and Fourier space, all collisionless fluid species obey

$$\delta' = -(1+w)\left(\frac{\theta}{\mathcal{H}} + \frac{h'_m}{2}\right) - 3\left(c_s^2 - w\right)\delta \tag{B.9}$$

$$\theta' = -(1+3w)\theta - \frac{w'}{1+w}\theta + \frac{c_s^2}{1+w}\frac{k^2}{\mathcal{H}}\delta - \frac{k^2}{\mathcal{H}}\sigma\,, \tag{B.10}$$

where δ, θ σ are the density, velocity and shear perturbations respectively, w is the equation of state, c_s^2 is the sound speed squared, $\mathcal{H} = aH$ is the conformal Hubble parameter, and h_m is the first of two metric perturbations in the synchronous gauge (to be disambiguated from the dimensionless Hubble constant h). The derivatives are taken with respect to $x \equiv \ln a$, thus the reader may notice a factor of $\mathcal{H}$ difference from definitions encountered elsewhere in the literature where derivatives are written with respect to conformal time. Accompanying these are the scalar metric perturbation equations

$$h'_m = \frac{2k^2\eta}{\mathcal{H}^2} + \frac{8\pi G a^2}{\mathcal{H}^2}\sum_i \bar{\rho}_i \delta_i \tag{B.11}$$

$$\eta' = \frac{4\pi G a^2}{\mathcal{H}k^2}\sum_i (\bar{\rho}_i + \bar{p}_i)\theta_i\,, \tag{B.12}$$

where $\bar{\rho}$ and $\bar{p}$ denote the background energy density and pressure, and the sum is over all fluid species.

These equations are promoted to a Boltzmann hierarchy (so that shear and higher moments are also included) when the fluid is relativistic and free streaming. In ΛCDM this is relevant for photons, massless neutrinos, and, at early times, massive neutrinos. In addition, a collision term must be included in the velocity perturbation for inter-fluid scattering, such as that of the baryon-photon scattering. We include a full list of perturbation equations for all currently implemented species in appendix C.

B.3.2 Massive neutrinos

Unlike other ΛCDM species, massive neutrinos transition from relativistic at early times to non-relativistic at late times. Consequently, its mass enters the Fermi-Dirac distribution in a non-trivial manner, and any integrated quantities involving the distribution have no analytical solutions. Numerical solutions to the integrals over momenta require discretized momentum bins, each of which requires a separate Boltzmann hierarchy, rendering the calculations expensive.

To this end, Gaussian quadrature has proven to be an efficient approximation to the integral, requiring as few as three momentum bins to accurately compute these integrals [20]. This is the method we adopt in ABCMB. We work with the temperature normalized variables

$$x \equiv \frac{am}{T_0}, \quad q \equiv \frac{ap}{T_0}, \quad \epsilon \equiv \sqrt{x^2 + q^2} \tag{B.13}$$

in place of the mass m, physical momentum p, and energy respectively at scale factor a and present neutrino temperature T_0. The massive neutrino contribution to the perturbed stress energy can be written as

$$\begin{aligned}
\delta\rho &= 4\pi \frac{T_0^4}{a^4} \int_0^\infty dq\ q^2 \epsilon f_0(q) \Psi_0(q) \\
(\rho + p)\theta &= 4\pi \frac{T_0^4}{a^4} \int_0^\infty dq\ q^3 f_0(q) k \Psi_1(q) \\
(\rho + p)\sigma &= \frac{8\pi T_0^4}{3a^4} \int_0^\infty dq\ \frac{q^4}{\epsilon} f_0(q) \Psi_2(q)\,,
\end{aligned} \tag{B.14}$$

where Ψ_l's form the Boltzmann hierarchy for momentum bin q and wavenumber k. f_0 is the Fermi-Dirac distribution

$$f_0(q) = \frac{g_s}{(2\pi)^3} \frac{1}{e^\epsilon + 1}\,, \tag{B.15}$$

with $g_s = 2$. Ref. [20] found that integrals of the following form can be approximated with the discrete sum

$$\int_0^\infty dq \frac{q^4 e^q}{(1+e^q)^2} \left(\frac{q}{\epsilon}\right)^w \Psi_l \approx 4 \sum_i K_i \left(\frac{q_i}{\epsilon_i}\right)^w \Psi_l\,, \tag{B.16}$$

which achieves accuracy to better than 2×10^{-4} using a three point quadrature sampling strategy with weights

$$\begin{aligned}
K_i &= \{0.0687359,\ 3.31435,\ 2.29911\} \\
q_i &= \{0.913201,\ 3.37517,\ 7.79184\}\,.
\end{aligned}$$

With these weights and roots we can rewrite eq. (B.14) as

$$\begin{aligned}
\delta\rho &= \frac{4}{\pi^2} \frac{T_0^4}{a^4} \sum_i K_i \frac{(1 + e^{-q_i})\epsilon_i}{q_i^2} \Psi_0(q_i) \\
(\rho + p)\theta &= \frac{4}{\pi^2} \frac{T_0^4}{a^4} \sum_i K_i \frac{(1 + e^{-q_i})}{q_i} k \Psi_1(q_i) \\
(\rho + p)\sigma &= \frac{8}{3\pi^2} \frac{T_0^4}{a^4} \sum_i K_i \frac{(1 + e^{-q_i})}{\epsilon_i} \Psi_2(q_i)\,.
\end{aligned} \tag{B.17}$$

These expressions allow us to limit the massive neutrino Boltzmann hierarchies to only 3 systems of $\ell_{\rm max}$ equations, minimizing their impact on performance. In fact, we may go a bit further and express

the background quantities with Gaussian quadrature, so that a numerical integral may be replaced by a discrete sum. We write the energy density and pressure as

$$\rho = 4\pi \frac{T_0^4}{a^4} \int_0^\infty dq\ q^2 \epsilon f_0(q) = \frac{4}{\pi^2} \frac{T_0^4}{a^4} \sum_i K_i \frac{(1+e^{-q_i})\epsilon_i}{q_i^2}$$
$$p = \frac{4\pi}{3} \frac{T_0^4}{a^4} \int_0^\infty dq\ \frac{q^4}{\epsilon} f_0(q) = \frac{4}{3\pi^2} \frac{T_0^4}{a^4} \sum_i K_i \frac{(1+e^{-q_i})}{\epsilon_i}\,. \tag{B.18}$$

Since these background quantities do not depend on the Boltzmann hierarchies, we can use the more accurate 5-point Gauss-Laguerre sampling with the weights and roots

$$K_i = \{0.0081201,\ 0.689407,\ 2.8063,\ 2.05156,\ 0.12681\}$$
$$q_i = \{0.583165,\ 2.0,\ 4.0,\ 7.26582,\ 13.0\}\,,$$

while minimally impacting performance. As with other species, we include our handling of the Boltzmann hierarchies themselves in appendix C.

B.4 Transfer functions and power spectra

Currently ABCMB implements the standard scalar primordial spectrum, characterized by the amplitude A_s and scalar tilt n_s as

$$\mathcal{P}(k) = A_s \left(\frac{k}{k_{\rm pivot}}\right)^{n_s - 1}\,, \tag{B.19}$$

where we take $k_{\rm pivot} = 0.05\ {\rm Mpc}^{-1}$ by default. The matter power spectrum is computed as

$$P(k,z) = \frac{2\pi^2}{k^3} \mathcal{P}(k) \left[\frac{\sum_i \rho_i(z)\delta_i(k,z)}{\sum_i \rho_i(z)}\right]^2\,, \tag{B.20}$$

where the sums are over all species that are non-relativistic.

The calculation of the CMB power spectra consists of a time integral over the photon streaming history, as well as a line of sight integral to project the spectrum onto the spherical sky. To perform these steps, we follow the formalism of ref. [75] for computing the CMB power spectra in non-flat geometries, where for now we take the zero curvature limit to recover the flat space solutions.

This calculation begins with the CMB source functions, separated into the temperature and polarization contributions. In synchronous gauge they read

$$\begin{aligned}
S_{T0} &= g\left(\frac{\delta_\gamma}{4} + \mathcal{H}\alpha'\right) + g(\eta - \mathcal{H}\alpha' - 2\mathcal{H}\alpha) + 2\mathcal{H}e^{-\kappa}(\eta' - \mathcal{H}'\alpha - \mathcal{H}\alpha') \\
&\qquad + \mathcal{H}\left[g\left(\frac{\theta_b'}{k^2} + \alpha'\right) + g'\left(\frac{\theta_b}{k^2} + \alpha\right)\right] \\
S_{T1} &= e^{-\kappa} k(\mathcal{H}\alpha' + 2\mathcal{H}\alpha - \eta) \\
S_{T2} &= \frac{g}{8}(2\sigma_\gamma + G_{\gamma 0} + G_{\gamma 2}) \\
S_E &= \sqrt{6}\ S_{T2}\,,
\end{aligned} \tag{B.21}$$

where

$$\kappa(\ln a) = \int_{\ln a}^0 \frac{d\ln a'}{\mathcal{H}\tau_c} = \int_{\ln a}^0 d\ln a' \frac{a n_e \sigma_T}{\mathcal{H}}$$
$$g = \frac{1}{\tau_c} e^{-\kappa} = a n_e \sigma_T e^{-\kappa}\,,$$

$G_{\gamma l}$'s are the CMB polarization Boltzmann hierarchy, and $\alpha \equiv \mathcal{H}\frac{h'_m+6\eta'}{2k^2}$. Here $\tau_c = (an_e\sigma_T)^{-1}$ is the Compton scattering time, with σ_T the Thomson cross section. These source functions are the same as those used in CLASS, and the separated temperature source functions differ from the single source function implemented in CAMB by integrating the transfer function by parts. This is both to disentangle the physical contributions to the CMB power spectrum (SW, ISW, Doppler and polarization), and to avoid computing second or third time derivatives of various quantities. With these, transfer functions Δ can be obtained after integrating over the CMB streaming history:

$$\begin{aligned}
\Delta_{T0}(k,\ell) &= \int d\tau\ S_{T0}(k,\tau) j_\ell(x) \\
\Delta_{T1}(k,\ell) &= \int d\tau\ S_{T1}(k,\tau) \frac{d}{dx} j_\ell(x) \\
\Delta_{T2}(k,\ell) &= \int d\tau\ S_{T2}(k,\tau) \frac{1}{2}\left[3\frac{d^2}{dx^2} j_\ell(x) + j_\ell(x)\right] \\
\Delta_{E}(k,\ell) &= \int d\tau\ S_{E}(k,\tau) \sqrt{\frac{3(\ell+2)!}{8(\ell-2)!}} \frac{1}{x^2} j_\ell(x)\,,
\end{aligned}$$

where $x \equiv k(\tau_0 - \tau)$, τ_0 the conformal time today, and j_ℓ is the spherical Bessel function of the first kind.

The spherical Bessel functions of arbitrary ℓ are not currently implemented in JAX. Furthermore, existing numerical algorithms for computing Bessel functions are costly, especially for the CMB which requires repeated evaluations. We find the following strategy for computing $j_\ell(x)$ to be efficient, which involves splitting the function over three intervals:

- The Bessel function is set to 0 in the interval $0 < x \leq x_{\rm start}$. Here $x_{\rm start}$ is defined such that $j_\ell(x_{\rm start})$ exceeds a threshold value of 10^{-10} for the first time. Below this value the bessel function is exponentially suppressed.

- We use linear interpolation in the interval $x_{\rm start} < x \leq x_5$, where x_5 is the location of the fifth local maximum, i.e. "peak" of the Bessel function. For each ℓ we numerically find the two boundaries of this interval, and store j_ℓ computed with `scipy.special.spherical_jn` over 5000 points, corresponding to a fine resolution of ~ 1000 points per oscillation.

- For $x > x_5$ we use an asymptotic expansion given by [76]

$$\begin{aligned}
j_\ell(x > x_5) &\approx \sqrt{\frac{1}{x\sqrt{x^2 - (\ell+\frac{1}{2})^2}}} \cos\left(Q_{\ell+\frac{1}{2}}(x) - \frac{\pi}{4}\right) \\
Q_\ell(x) &\equiv \sqrt{x^2-\ell^2} - \frac{\ell\pi}{2} + \ell \arcsin\left(\frac{\ell}{x}\right),
\end{aligned}$$

  which we find to be accurate and inexpensive to compute for sufficiently large x.

The total temperature transfer function is the sum of the components $\Delta_T \equiv \Delta_{T0} + \Delta_{T1} + \Delta_{T2}$, which is obtained after their respective source functions are integrated. This then finally enters the angular power spectrum via

$$C_\ell^{XY} = 4\pi \int \frac{dk}{k} \mathcal{P}(k) \Delta_X(k,\ell) \Delta_Y(k,\ell)\,, \tag{B.22}$$

where $X, Y \in \{T, E\}$. With these equations ABCMB is able to compute the TT, TE, and EE spectra using the results from perturbations module.

B.5 Lensing

The matter field between the last scattering surface and the observer gravitationally lenses the CMB signal. The effect is most significant at small angular scales, modifying the power spectrum by as much as $\mathcal{O}(0.1)$ at $\ell \gtrsim 2000$. As was done in CAMB and CLASS, we implement the effect of lensing using the spherical-sky correlation function formalism of ref. [77], and we outline the important steps in this section.

Lensing is sourced by the gravitational lensing potential ψ, such that an unlensed signal of the CMB $\Theta(\hat{n})$ is deflected to $\tilde{\Theta}(\hat{n}')$, where $\hat{n}' = \hat{n} + \vec{\nabla}\psi$. The lensing potential obeys the Poisson equation, which in Fourier space reads

$$k^2\psi = 4\pi G a^2 \rho\delta\,. \tag{B.23}$$

Lensing is most efficient after radiation domination ends. For this reason, we have omitted the small contribution from anisotropic stress, and contributions to $\rho\delta$ include only non-relativistic species to good approximation. We can thus write the power spectrum of the lensing potential as

$$P_\psi(k,a) = \frac{9\Omega_m(a)^2\mathcal{H}^4}{8\pi^2 k} P(k,a)\,, \tag{B.24}$$

where $P(k,a)$ is the matter power spectrum in eq. (B.20), and $\Omega_m(a) = \rho_m(a)/\rho_{\rm tot}(a)$ (we use this definition since its evolution with a is more apparent, as $\rho_{\rm crit}$ is only defined today).

For the CMB on the spherical sky, we need the angular lensing power spectrum. This can be obtained with the power spectrum defined above and the transfer function formalism of the previous section. However, the lensing power spectrum varies slowly as a function of k compared to the highly oscillatory Bessel functions at high ℓ. Thus the lensing spectrum can be efficiently computed under the Limber approximation as

$$C_\ell^{\psi\psi} = \frac{8\pi^2}{\left(\ell+\frac{1}{2}\right)^3}\int_0^{\chi_*}\frac{d\chi}{\chi}P_\psi\left(k=\frac{\ell+\frac{1}{2}}{\chi},\tau_0-\chi\right)\left(\frac{\chi_*-\chi}{\chi_*}\right)^2\,, \tag{B.25}$$

where $\chi \equiv \tau_0 - \tau$ is the look back time, and χ_* is defined at recombination. The Limber approximation is highly accurate for $\ell > 10$, and leads to an order 10% error below $\ell = 10$. However, the lensing signal is extremely weak at very large angular scales, and the error virtually unobservable. In figures 6 and 7 we show the residual error in the lensed TT and EE spectra compared to CLASS. Here the Limber approximation is used by ABCMB and not used by CLASS for $\ell \leq 10$, yet the errors remain comparable at all ℓ's to the unlensed comparisons in the same figures. This indicates that the use of Limber approximation at low-ℓ does not dominate the error.

With the lensing power spectrum and angular spectrum defined this way, the lensed CMB power spectra are obtained by implementing Eqs.(35-66) in ref. [77]. In the very last step, the lensed power spectrum $\tilde{C}$ is obtained from the lensed correlation function $\tilde{\xi}$ via an angular Fourier transform, involving an integral of the form

$$\tilde{C}_\ell = \int_{-1}^{1} d\mu\; \tilde{\xi}(\mu) d_\ell(\mu)\,,$$

for each TT, TE, and EE. Here tilde denotes lensed quantities, and d_ℓ is a Wigner matrix element depending on the source (temperature or polarization). Ref. [77] points out that instead of performing the full integration over $[-1,1]$, the lensed power spectrum can be more easily obtained by solving the difference $\tilde{C}_\ell - C_\ell$. The analogous integrand for this difference rapidly vanishes away from $\mu = 1$, allowing one to speed up the integration by restricting the domain down to a smaller interval, typically over $\sim [0.98, 1]$. In CLASS, this approximation is implemented under the "fast lensing" setting, as opposed to the "accurate lensing" setting where the full integral is solved with a high order Gauss-Legendre quadrature. In practice, we find that this approximation introduces error of several percent for $\ell > 3000$. As modern precision measurements easily surpass this angular scale, we conclude that the fast lensing approximation should never be used in ABCMB and thus not implemented.

B.6 Approximation schemes

By default, CLASS uses several approximations that considerably accelerate the power spectra computations. Most notably, the costly Boltzmann hierarchies in the perturbation evolution are aided by the tight-coupling approximation (TCA), the ultra-relativistic fluid approximation (UFA), and the radiation streaming approximation (RSA). While they stem from different physical arguments, all three approximations effective decouple the first three moments in the hierarchy from the remaining moments, applicable to photons and both massive and massless neutrinos. After this decoupling, the higher moments are either replaced by some analytical approximation, or trivially set to zero. In either case, the reduction in the number of equations leads to a notable speedup. In addition, these approximations can help in stabilizing the differential equation solver. TCA in particular has been vital in reducing the stiffness of the baryon-photon coupling equations at early times, introduced by the largeness of the Thomson scattering rate and the smallness of the baryon-photon velocity difference.

At the moment, ABCMB does not utilize any of these three approximations in its perturbation module. Most notably, the diffrax adaptive step integrator used in our differential equation solutions is able to solve the baryon-photon equations before and during recombination without transforming the equations with TCA. The resulting baryon and photon perturbations are highly accurate compared to the CLASS output where TCA is used during recombination. In fact, foregoing TCA in ABCMB is computationally advantageous. This is because in using TCA, we must break the solution into two eras, where the derivative equations and the number of moments being evolved are distinct, and a patching is needed at the break point. In practice, this separate-then-combine approach introduces considerable overhead for the diffrax solver, lengthening both the compile and the computation times, while providing no improvements to accuracy.

RSA and UFA, though not currently included, may still be beneficial to ABCMB's performance. These approximations turn on at late times (after recombination), where the higher l terms in the Boltzmann hierarchies of the radiation species are negligible deep in matter domination. In fact, terminating the solver for the higher l moments here help reduce the error incurred by the Boltzmann hierarchy cutoff, which is known to cause a periodic error for the perturbation modes at a frequency of $(k\tau) \sim l_{\max}$. In addition, a combination of RSA and UFA can accelerate CLASS computations by as much as 70-80%. We leave implementation of RSA and UFA in ABCMB to future work.

Lastly, CLASS uses the time cut and multipole cut approximations to reduce the limits of integration for the CMB transfer functions. In ABCMB, we use simple trapezoid rules over the full range of k and $\ln a$. Even still, we find that the transfer integral run time is vastly subdominant to the perturbations module on the GPU. We thus conclude that these approximations are not necessary and exclude them in the current version.

C Currently implemented species

As discussed in the main text, fluid-specific physics are packaged within individual objects responsible for all features of a given fluid. Here we present a list of all background and perturbation equations for all species currently implemented in ABCMB.

C.1 Dark energy

The `DarkEnergy` fluid is responsible only for energy density and pressure, given that standard cosmology does not introduce any spatial fluctuations to the dark energy field. The equations are then simply

$$\begin{aligned} \rho_\Lambda &= \rho_{\Lambda,0} = \Omega_\Lambda \rho_{\rm crit} \\ P_\Lambda &= -\rho_\Lambda \,, \end{aligned} \tag{C.1}$$

where we used the standard $w = -1$ equation of state for the dark energy, and $\rho_{\rm crit} = \frac{3H_0^2}{8\pi G}$.

C.2 Cold dark matter

CDM behaves like a non-relativistic fluid through all of cosmology, with vanishing sound speed and equation of state and energy density that redshifts simply. For the background we have

$$\begin{aligned} \rho_{\rm cdm} &= \Omega_{\rm cdm}\rho_{\rm crit}a^{-3} \\ P_{\rm cdm} &= 0\,. \end{aligned} \tag{C.2}$$

The CDM density perturbation follows only the gravitational metric perturbation. Its velocity perturbation vanishes in the synchronous gauge as the rest frame of CDM is used to define the coordinate system of the gauge. The only perturbation equation is therefore

$$\delta'_{\rm cdm} = -\frac{1}{2}h'_m\,, \tag{C.3}$$

where we emphasize again that $'$ in this work denotes derivative with respect to $\ln a$ and h_m is the usual metric perturbation, disambiguated from the dimensionless Hubble constant h.

C.3 Baryons

Like CDM, baryons are also non-relativistic throughout the period relevant for the CMB. However, interactions with photons via Thomson scattering induces a non-trivial velocity perturbation. Although pressure can still be ignored, we must account for the baryon sound speed contribution to the velocity perturbation despite its small value. The relevant background quantities are

$$\begin{aligned} \rho_b &= \Omega_b\rho_{\rm crit}a^{-3} \\ P_b &= 0 \\ c_s^2 &= \frac{T_b}{\mu_b}\left(1 - \frac{1}{3}\frac{d\ln T_b}{d\ln a}\right) \\ \mu_b &\equiv \frac{\rho_b}{n_b} = \frac{m_{\rm H}}{(1+x_e)(1-{\rm Y_{He}}) + \frac{m_{\rm H}}{m_{\rm He}}{\rm Y_{He}}}\,, \end{aligned} \tag{C.4}$$

where μ_b is the mean baryon mass, receiving contributions from hydrogen and helium atoms/ions as well as free electrons at a given time. The density and velocity perturbations are

$$\begin{aligned} \delta'_b &= -\frac{\theta_b}{\mathcal{H}} - \frac{1}{2}h'_m \\ \theta'_b &= -\theta_b + \frac{c_s^2k^2\delta_b}{\mathcal{H}} + \frac{4\rho_\gamma}{3\rho_b}\frac{1}{\mathcal{H}\tau_c}(\theta_\gamma - \theta_b)\,, \end{aligned} \tag{C.5}$$

where the subscript γ denotes photon related quantities that we define next.

C.4 Photons

The photon fluid is not parameterized by a density parameter, but instead by its mean temperature today. The energy density and pressure are expressed in terms of the temperature assuming a Bose-Einstein distribution

$$\begin{aligned} \rho_\gamma &= \frac{\pi^2}{15}T_{\gamma,0}^4a^{-4} \\ P_\gamma &= \frac{1}{3}\rho_\gamma\,. \end{aligned} \tag{C.6}$$

Photons require perturbations beyond the density and velocity moments due to their free-streaming nature at late times. The photon Boltzmann hierarchy is divided into the temperature

moments $F_{\gamma l}$ and polarization moments $G_{\gamma l}$. The moments l label the expansion coefficients in the Legendre polynomial basis, and the resulting Boltzmann equations for each coefficient are

$$\begin{aligned}
\delta'_\gamma &= -\frac{4}{3\mathcal{H}}\theta_\gamma - \frac{2}{3}h'_m \\
\theta'_\gamma &= \frac{k^2}{\mathcal{H}}\left(\frac{1}{4}\delta_\gamma - \sigma_\gamma\right) + \frac{1}{\mathcal{H}\tau_c}(\theta_b - \theta_\gamma) \\
\sigma'_\gamma &= \frac{4}{15\mathcal{H}}\theta_\gamma - \frac{3}{10}\frac{k}{\mathcal{H}}F_{\gamma 3} + \frac{2}{15}h'_m + \frac{4}{5}\eta' - \frac{9}{10}\frac{1}{\mathcal{H}\tau_c} + \frac{1}{20}\frac{1}{\mathcal{H}\tau_c}(G_{\gamma 0} + G_{\gamma 2}) \\
F'_{\gamma l} &= \frac{1}{2l+1}\frac{k}{\mathcal{H}}(lF_{\gamma l-1} - (l+1)F_{\gamma l+1}) - \frac{1}{\mathcal{H}\tau_c}F_{\gamma l} && 3 \le l < l_{\max} \\
F'_{\gamma l_{\max}} &= \frac{k}{\mathcal{H}}F_{\gamma l_{\max}-1} - \frac{(l_{\max}+1)}{\mathcal{H}\tau}F_{\gamma l_{\max}} - \frac{1}{\mathcal{H}\tau_c}F_{\gamma l_{\max}} \\
G'_{\gamma l} &= \frac{1}{2l+1}\frac{k}{\mathcal{H}}(lG_{\gamma l-1} - (l+1)G_{\gamma l+1}) - \frac{1}{\mathcal{H}\tau_c}G_{\gamma l} \\
&\qquad + \frac{1}{2}\frac{1}{\mathcal{H}\tau_c}(2\sigma_\gamma + G_{\gamma 0} + G_{\gamma 2})(\delta_{l,0} + \frac{1}{2}\delta_{l,2}) && l < l_{\max} \\
G'_{\gamma l_{\max}} &= \frac{k}{\mathcal{H}}G_{\gamma l_{\max}-1} - \frac{(l_{\max}+1)}{\mathcal{H}\tau}G_{\gamma l_{\max}} - \frac{1}{\mathcal{H}\tau_c}G_{\gamma l_{\max}}\,, && \text{(C.7)}
\end{aligned}$$

where for consistency with ref. [47] we used $\delta_\gamma \equiv F_{\gamma 0}$, $\theta_\gamma \equiv \frac{3}{4}kF_{\gamma 1}$, and $\sigma_\gamma \equiv F_{\gamma 2}/2$.

C.5 Massless neutrinos

Massless neutrinos require two parameters, $N_{\rm massless}$ and $T_{\rm massless}$. These define the background quantities

$$\begin{aligned}
\rho_{\nu,\rm massless} &= N_{\rm massless}\frac{7\pi^2}{120}T^4_{\rm massless}a^{-4} \\
P_{\nu,\rm massless} &= \frac{1}{3}\rho_{\nu,\rm massless}\,. && \text{(C.8)}
\end{aligned}$$

In the absence of massive neutrinos, $N_{\rm massless}$ is the literal number of massless neutrino species. This means $N_{\rm massless} = 3$ in standard cosmology, and the additional correction in $N_{\rm eff} > 3$ is encoded in the late-time temperature $T_{\rm massless}$. These parameters are discussed in more detail in section 2.4.3.

Neutrinos are free-streaming and relativistic at all times, thus generating shear and higher moments in the Boltzmann hierarchy. They are given by

$$\begin{aligned}
\delta'_\nu &= -\frac{4}{3\mathcal{H}}\theta_\nu - \frac{2}{3}h'_m \\
\theta'_\nu &= \frac{k^2}{\mathcal{H}}\left(\frac{1}{4}\delta_\nu - \sigma_\nu\right) \\
\sigma'_\nu &= \frac{4}{15\mathcal{H}}\theta_\nu - \frac{3}{10}\frac{k}{\mathcal{H}}F_{\nu 3} + \frac{2}{15}h'_m + \frac{4}{5}\eta' \\
F'_{\nu l} &= \frac{1}{2l+1}\frac{k}{\mathcal{H}}(lF_{\nu l-1} - (l+1)F_{\nu l+1}) && 3 \le l < l_{\max} \\
F'_{\nu l_{\max}} &= \frac{k}{\mathcal{H}}F_{\nu l_{\max}-1} - \frac{(l_{\max}+1)}{\mathcal{H}\tau}F_{\nu l_{\max}}\,. && \text{(C.9)}
\end{aligned}$$

C.6 Massive neutrinos

We provide an overview of integrated massive neutrino quantities in Sec. B.3.2, including the energy density and pressure. These integrals rely on the Boltzmann hierarchy moments, which for massive neutrinos depend on both the wavenumber k and the momentum q. The momentum dependence is

separable, and each value q labels a separate hierarchy which obeys

$$\begin{aligned}
\Psi_0' &= -\frac{qk}{\epsilon\mathcal{H}}\Psi_1 + \frac{1}{6}h_m'\frac{d\ln f_0}{d\ln q} \\
\Psi_1' &= \frac{1}{3}\frac{qk}{\epsilon\mathcal{H}}(\Psi_0 - 2\Psi_2) \\
\Psi_2' &= \frac{1}{5}\frac{qk}{\epsilon\mathcal{H}}(2\Psi_1 - 3\Psi_3) - \left(\frac{1}{15}h_m' + \frac{2}{5}\eta'\right)\frac{d\ln f_0}{d\ln q} \\
\Psi_l' &= \frac{1}{2l+1}\frac{qk}{\epsilon\mathcal{H}}(l\Psi_{l-1} - (l+1)\Psi_{l+1}) \qquad\qquad 3 \le l < l_{\max} \\
\Psi_{l_{\max}}' &= \frac{qk}{\epsilon\mathcal{H}}\Psi_{l_{\max}-1} - \frac{l_{\max}+1}{\mathcal{H}\tau}\Psi_{l_{\max}}\,,
\end{aligned} \tag{C.10}$$

where $\frac{d\ln f_0}{d\ln q} = -\frac{q}{1+e^{-q}}$, with f_0 the Fermi-Dirac distribution given by eq. (B.15). As a reminder $q = ap/T_0$ is the temperature normalized momentum, T_0 is the massive neutrino temperature today, and $\epsilon \equiv \sqrt{q^2 + (am/T_0)^2}$.

D Gradients with respect to cosmological parameters

For reference, we include additional plots of gradients with respect to cosmological parameters in this appendix. Each of these can be computed by the user with ABCMB using the built-in `jax.jacfwd`. In particular, we compute the gradients of the output $P(k)$ (figure 12), $C_\ell^{\rm TT}$ (figure 13), $C_\ell^{\rm TE}$ (figure 14), and $C_\ell^{\rm EE}$ (figure 15) with respect to all input cosmological parameters using forward AD.

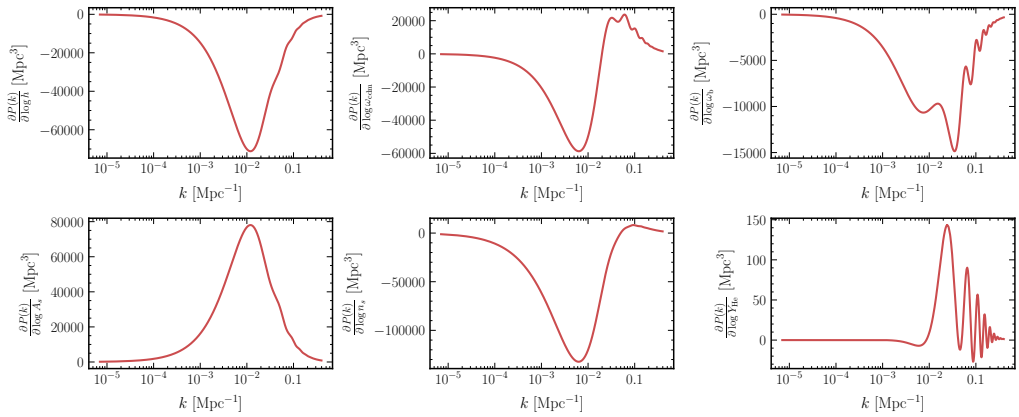

Figure 12: The ΛCDM matter power spectrum gradients with respect to six cosmological parameters.

References

[1] Planck collaboration, *Planck2018 results: VI. Cosmological parameters*, *Astronomy & Astrophysics* **641** (2020) A6.

[2] Atacama Cosmology Telescope collaboration, *The Atacama Cosmology Telescope: DR6 power spectra, likelihoods and ΛCDM parameters*, *Journal of Cosmology and Astroparticle Physics* **2025** (2025) 062.

[3] SPT collaboration, *Constraints on Cosmological Parameters from the 500 deg^2 SPTPOL Lensing Power Spectrum*, *The Astrophysical Journal* **888** (2020) 119.

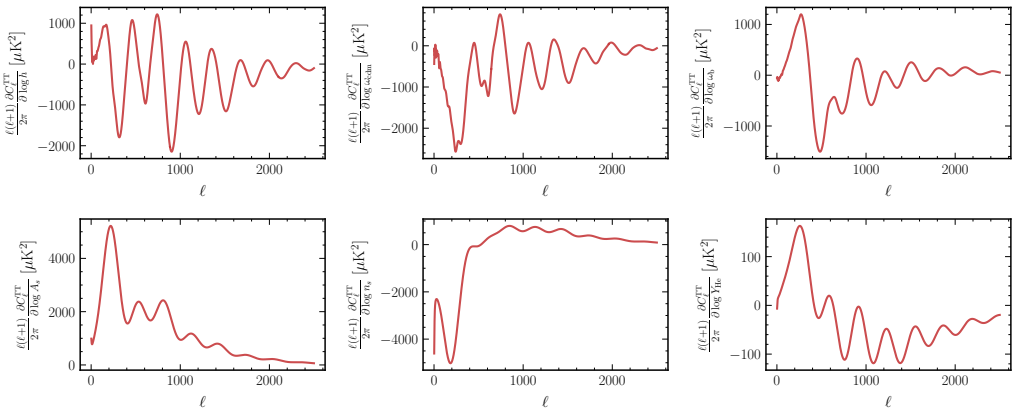

Figure 13: The ΛCDM TT spectrum gradients with respect to six cosmological parameters.

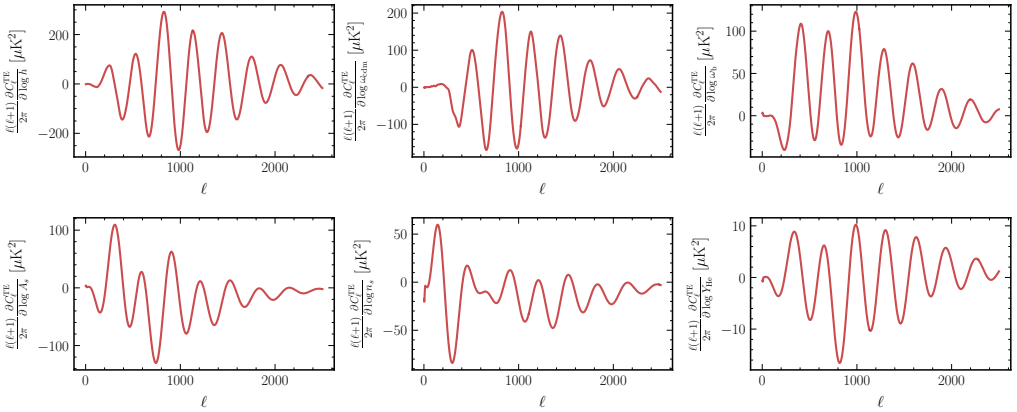

Figure 14: The ΛCDM TE spectrum gradients with respect to six cosmological parameters.

[4] SPT-3G collaboration, E. Camphuis et al., *SPT-3G D1: CMB temperature and polarization power spectra and cosmology from 2019 and 2020 observations of the SPT-3G Main field*, 2025.

[5] Simons Observatory collaboration, *The Simons Observatory: science goals and forecasts*, *Journal of Cosmology and Astroparticle Physics* **2019** (2019) 056–056.

[6] J. Lesgourgues, G. Marques-Tavares and M. Schmaltz, *Evidence for dark matter interactions in cosmological precision data?*, *JCAP* **02** (2016) 037 [1507.04351].

[7] M.A. Buen-Abad, M. Schmaltz, J. Lesgourgues and T. Brinckmann, *Interacting Dark Sector and Precision Cosmology*, *JCAP* **01** (2018) 008 [1708.09406].

[8] E. Di Valentino, C. Bøehm, E. Hivon and F.R. Bouchet, *Reducing the H_0 and σ_8 tensions with Dark Matter-neutrino interactions*, *Phys. Rev. D* **97** (2018) 043513 [1710.02559].

[9] V. Poulin, T.L. Smith, T. Karwal and M. Kamionkowski, *Early Dark Energy Can Resolve The Hubble Tension*, *Phys. Rev. Lett.* **122** (2019) 221301 [1811.04083].

[10] K.K. Boddy, V. Gluscevic, V. Poulin, E.D. Kovetz, M. Kamionkowski and R. Barkana, *Critical assessment of CMB limits on dark matter-baryon scattering: New treatment of the relative bulk velocity*, *Phys. Rev. D* **98** (2018) 123506 [1808.00001].

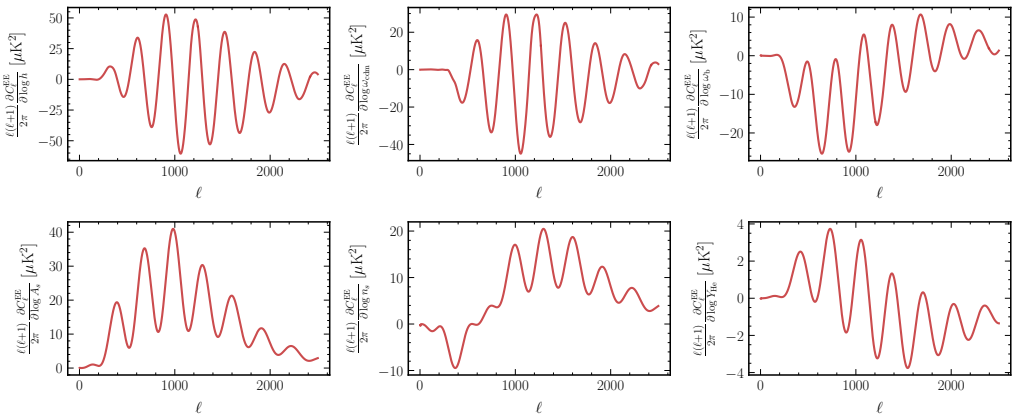

Figure 15: The ΛCDM EE spectrum gradients with respect to six cosmological parameters.

[11] V. Poulin, T.L. Smith, D. Grin, T. Karwal and M. Kamionkowski, *Cosmological implications of ultralight axionlike fields*, *Phys. Rev. D* **98** (2018) 083525 [1806.10608].

[12] D. Aloni, A. Berlin, M. Joseph, M. Schmaltz and N. Weiner, *A step in understanding the hubble tension*, *Physical Review D* **105** (2022) .

[13] Z. Zhou and N. Weiner, *Searching for Dark Matter Interactions with ACT, SPT and DES*, 2024.

[14] L. Herold and M. Kamionkowski, *Revisiting the impact of neutrino mass hierarchies on neutrino mass constraints in light of recent DESI data*, *Physical Review D* **111** (2025) .

[15] J. Lesgourgues, *The Cosmic Linear Anisotropy Solving System (CLASS) I: Overview*, 2011.

[16] D. Blas, J. Lesgourgues and T. Tram, *The Cosmic Linear Anisotropy Solving System (CLASS). part II: Approximation schemes*, *Journal of Cosmology and Astroparticle Physics* **2011** (2011) 034–034.

[17] J. Lesgourgues, *The Cosmic Linear Anisotropy Solving System (CLASS) III: Comparision with CAMB for LambdaCDM*, 4, 2011.

[18] J. Lesgourgues and T. Tram, *The Cosmic Linear Anisotropy Solving System (CLASS) IV: efficient implementation of non-cold relics*, *Journal of Cosmology and Astroparticle Physics* **2011** (2011) 032–032.

[19] A. Lewis, A. Challinor and A. Lasenby, *Efficient computation of CMB anisotropies in closed FRW models*, ApJ **538** (2000) 473 [astro-ph/9911177].

[20] C. Howlett, A. Lewis, A. Hall and A. Challinor, *CMB power spectrum parameter degeneracies in the era of precision cosmology*, J. Cosmology Astropart. Phys. **1204** (2012) 027 [1201.3654].

[21] C. Giovanetti, M. Lisanti, H. Liu, S. Mishra-Sharma and J.T. Ruderman, *Cosmological parameter estimation with a joint-likelihood analysis of the cosmic microwave background and big bang nucleosynthesis*, *Phys. Rev. D* **112** (2025) 063530.

[22] C. Doux, M. Penna-Lima, S.D.P. Vitenti, J. Tréguer, E. Aubourg and K. Ganga, *Cosmological constraints from a joint analysis of cosmic microwave background and spectroscopic tracers of the large-scale structure*, *Monthly Notices of the Royal Astronomical Society* **480** (2018) 5386 [https://academic.oup.com/mnras/article-pdf/480/4/5386/25985754/sty2160.pdf].

[23] J. Barron, D. Curtin, H. Liu, J. Munoz and S. Roy, *Constraining Dark Acoustic Oscillations with the High-Redshift UV Luminosity Function*, 2512.01998.

[24] A. Spurio Mancini, D. Piras, J. Alsing, B. Joachimi and M.P. Hobson, *¡scp¿cosmopower¡/scp¿: emulating cosmological power spectra for accelerated bayesian inference from next-generation surveys*, *Monthly Notices of the Royal Astronomical Society* **511** (2022) 1771–1788.

[25] D. Piras and A. Spurio Mancini, *CosmoPower-JAX: high-dimensional Bayesian inference with differentiable cosmological emulators*, *The Open Journal of Astrophysics* **6** (2023) .

[26] J. El Gammal, N. Schöneberg, J. Torrado and C. Fidler, *Fast and robust Bayesian inference using Gaussian processes with GPry*, *Journal of Cosmology and Astroparticle Physics* **2023** (2023) 021.

[27] A. Nygaard, E.B. Holm, S. Hannestad and T. Tram, *CONNECT: a neural network based framework for emulating cosmological observables and cosmological parameter inference*, *Journal of Cosmology and Astroparticle Physics* **2023** (2023) 025.

[28] M. Bonici, E. Baxter, F. Bianchini and J. Ruiz-Zapatero, *Capse.jl: efficient and auto-differentiable CMB power spectra emulation*, *The Open Journal of Astrophysics* **7** (2024) .

[29] S. Günther, L. Balkenhol, C. Fidler, A.R. Khalife, J. Lesgourgues, M.R. Mosbech et al., *OLÉ – Online Learning Emulation in Cosmology*, 2025.

[30] L. Janken, S. Hannestad, T. Tram and A. Nygaard, *CLiENT: A new tool for emulating cosmological likelihoods using deep neural networks*, 2025.

[31] R.T. Hough, R. Rugg, S. Sahlu and A. Abebe, *Kosmulator: A Python framework for cosmological inference with MCMC*, 2026.

[32] Z. Li, J. Sullivan and M. Millea, *Bolt.jl*, Nov., 2023. 10.5281/zenodo.10065126.

[33] O. Hahn, F. List and N. Porqueres, *Disco-dj i: a differentiable einstein-boltzmann solver for cosmology*, *Journal of Cosmology and Astroparticle Physics* **2024** (2024) 063.

[34] H. Sletmoen, *SymBoltz.jl: a symbolic-numeric, approximation-free and differentiable linear Einstein-Boltzmann solver*, 2025.

[35] F. List, O. Hahn, T. Flöss and L. Winkler, *DISCO-DJ II: a differentiable particle-mesh code for cosmology*, 2025.

[36] L. Balkenhol, C. Trendafilova, K. Benabed and S. Galli, *candl: cosmic microwave background analysis with a differentiable likelihood*, *Astronomy & Astrophysics* **686** (2024) A10.

[37] K. Benabed, "clipy: Pure Python `clik` implementation with JAX support." `https://github.com/benabed/clipy`, 2025.

[38] N. Lee and Y. Ali-Haïmoud, *hyrec-2: A highly accurate sub-millisecond recombination code*, *Physical Review D* **102** (2020) .

[39] C. Giovanetti, M. Lisanti, H. Liu, S. Mishra-Sharma and J.T. Ruderman, *Fast, differentiable, and extensible big bang nucleosynthesis package*, *Phys. Rev. D* **112** (2025) 063531.

[40] J. Bradbury, R. Frostig, P. Hawkins, M.J. Johnson, C. Leary, D. Maclaurin et al., *JAX: composable transformations of Python+NumPy programs*, 2018.

[41] DeepMind, *The DeepMind JAX Ecosystem*, 2020.

[42] R. Neal, *MCMC Using Hamiltonian Dynamics*, in *Handbook of Markov Chain Monte Carlo*, pp. 113–162 (2011), DOI.

[43] M. Betancourt and M. Girolami, *Hamiltonian Monte Carlo for hierarchical models*, *Current trends in Bayesian methodology with applications* **79** (2015) 2.

[44] E. Aver, E.D. Skillman, R.W. Pogge, N.S.J. Rogers, M.K. Weller, K.A. Olive et al., *The LBT $Y_{\rm p}$ Project IV: A New Value of the Primordial Helium Abundance*, 2026.

[45] T.-H. Yeh, K.A. Olive, B.D. Fields, E. Aver, R.W. Pogge, N.S.J. Rogers et al., *The LBT $Y_{\rm p}$ Project V: Cosmological Implications of a New Determination of Primordial ^4He*, 2026.

[46] P. Kidger and C. Garcia, *Equinox: neural networks in JAX via callable PyTrees and filtered transformations*, *Differentiable Programming workshop at Neural Information Processing Systems 2021* (2021) .

[47] C.-P. Ma and E. Bertschinger, *Cosmological perturbation theory in the synchronous and conformal Newtonian gauges*, *Astrophys. J.* **455** (1995) 7 [astro-ph/9506072].

[48] K. Akita and M. Yamaguchi, *A precision calculation of relic neutrino decoupling*, *JCAP* **08** (2020) 012 [2005.07047].

[49] J. Froustey, C. Pitrou and M.C. Volpe, *Neutrino decoupling including flavour oscillations and primordial nucleosynthesis*, *JCAP* **12** (2020) 015 [2008.01074].

[50] J.J. Bennett, G. Buldgen, P.F. De Salas, M. Drewes, S. Gariazzo, S. Pastor et al., *Towards a precision calculation of $N_{\rm eff}$ in the Standard Model II: Neutrino decoupling in the presence of flavour oscillations and finite-temperature QED*, *JCAP* **04** (2021) 073 [2012.02726].

[51] G. Mangano, G. Miele, S. Pastor, T. Pinto, O. Pisanti and P.D. Serpico, *Relic neutrino decoupling including flavour oscillations*, *Nuclear Physics B* **729** (2005) 221–234.

[52] R. Consiglio, P. de Salas, G. Mangano, G. Miele, S. Pastor and O. Pisanti, *PArthENoPE reloaded*, *Computer Physics Communications* **233** (2018) 237–242.

[53] C. Pitrou, A. Coc, J.-P. Uzan and E. Vangioni, *Precision big bang nucleosynthesis with improved helium-4 predictions*, *Physics Reports* **754** (2018) 1–66.

[54] J.A. Peacock and R.E. Smith, "HALOFIT: Nonlinear distribution of cosmological mass and galaxies." Astrophysics Source Code Library, record ascl:1402.032, Feb., 2014.

[55] C.R. Harris et al., *Array programming with NumPy*, *Nature* **585** (2020) 357.

[56] SciPy 1.0 Contributors, *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*, *Nature Methods* **17** (2020) 261.

[57] P. Kidger, *On Neural Differential Equations*, Ph.D. thesis, University of Oxford, 2021.

[58] J.D. Hunter, *Matplotlib: A 2D graphics environment*, *Computing in Science & Engineering* **9** (2007) 90.

[59] R. Conlin, *interpax*, Oct., 2025. 10.5281/zenodo.17378822.

[60] G. Brandl et al., *Sphinx Documentation Generator*, 2023.

[61] O. Pisanti, A. Cirillo, S. Esposito, F. Iocco, G. Mangano, G. Miele et al., *PArthENoPE: Public Algorithm Evaluating the Nucleosynthesis of Primordial Elements*, *Comput. Phys. Commun.* **178** (2008) 956 [0705.0290].

[62] S. Gariazzo, P. F. de Salas, O. Pisanti and R. Consiglio, *PArthENoPE revolutions*, *Comput. Phys. Commun.* **271** (2022) 108205 [2103.05027].

[63] S. Seager, D.D. Sasselov and D. Scott, "RECFAST: Calculate the Recombination History of the Universe." Astrophysics Source Code Library, record ascl:1106.026, June, 2011.

[64] Y. Ali-Haïmoud and C.M. Hirata, *Hyrec: A fast and highly accurate primordial hydrogen and helium recombination code*, *Physical Review D* **83** (2011) .

[65] Y. Ali-Haïmoud and C.M. Hirata, *Ultrafast effective multilevel atom method for primordial hydrogen recombination*, *Physical Review D* **82** (2010) .

[66] W. Hu, D. Scott, N. Sugiyama and M. White, *Effect of physical assumptions on the calculation of microwave background anisotropies*, *Physical Review D* **52** (1995) 5498–5515.

[67] P.J.E. Peebles, *Recombination of the Primeval Plasma*, ApJ **153** (1968) 1.

[68] Y.B. Zeldovich, V.G. Kurt and R.A. Syunyaev, *Recombination of Hydrogen in the Hot Model of the Universe*, *Zhurnal Eksperimentalnoi i Teoreticheskoi Fiziki* **55** (1968) 278.

[69] C.M. Hirata and J. Forbes, *Lyman-α transfer in primordial hydrogen recombination*, *Physical Review D* **80** (2009) .

[70] Y. Ali-Haimoud, D. Grin and C.M. Hirata, *Radiative transfer effects in primordial hydrogen recombination*, *Phys. Rev. D* **82** (2010) 123502 [1009.4697].

[71] D. Grin and C.M. Hirata, *Cosmological hydrogen recombination: The effect of extremely high-n states*, *Physical Review D* **81** (2010) .

[72] A. Lewis, *Cosmological parameters from WMAP 5-year temperature maps*, *Physical Review D* **78** (2008) .

[73] X. Wu, R. Kannan, F. Marinacci, M. Vogelsberger and L. Hernquist, *Simulating the effect of photoheating feedback during reionization*, *Monthly Notices of the Royal Astronomical Society* **488** (2019) 419–437.

[74] B. Villasenor, B. Robertson, P. Madau and E. Schneider, *Effects of Photoionization and Photoheating on Ly-alpha Forest Properties from Cholla Cosmological Simulations*, *The Astrophysical Journal* **912** (2021) 138.

[75] J. Lesgourgues and T. Tram, *Fast and accurate CMB computations in non-flat FLRW universes*, *JCAP* **09** (2014) 032 [1312.2697].

[76] G.N. Watson, *A Treatise on the Theory of Bessel Functions*, Cambridge University Press, Cambridge, England, 2 ed. (1944).

[77] A. Challinor and A. Lewis, *Lensed CMB power spectra from all-sky correlation functions*, *Physical Review D* **71** (2005) .