LSST DESC Notes

Core Cosmology Library: Precision Cosmological Predictions for LSST

Husni Almoubayyed,[1] David Alonso,[2] Jonathan Blazek,[3,4] Philip Bull,[5,6] Jean-Éric Campagne,[7] N. Elisa Chisari,[2] Alex Drlica-Wagner,[8] Zilong Du,[9] Tim Eifler,[10,11] John Ellison,[9] Renée Hlozek,[12] Mustapha Ishak,[13] Shahab Joudaki,[2] Matthew Kirby,[14] David Kirkby,[15] Elisabeth Krause,[16] C. Danielle Leonard,[1] Christiane S. Lorenz,[2] Phil Marshall,[17] Thomas McClintock,[14] Sean McLaughlin,[18] Jérémy Neveu,[7] Stéphane Plaszczynski,[7] Javier Sanchez,[15] Sukhdeep Singh,[1] Anže Slosar,[19] Tilman Tröster,[20] Antonio Villarreal,[21] Michal Vrastil,[22] and Joe Zuntz[20] (LSST Dark Energy Science Collaboration)

[1] *McWilliams Center for Cosmology, Department of Physics, Carnegie Mellon University, Pittsburgh, PA 15213, USA*
[2] *Department of Physics, University of Oxford, Denys Wilkinson Building, Keble Road, Oxford OX1 3RH, United Kingdom*
[3] *Center for Cosmology and Astroparticle Physics, Ohio State, Columbus, OH 43210, USA*
[4] *Laboratory of Astrophysics, École Polytechnique Fédérale de Lausanne (EPFL), Observatoire de Sauverny, 1290 Versoix, Switzerland*
[5] *California Institute of Technology, Pasadena, CA 91125, USA*
[6] *Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, California, USA*
[7] *Laboratoire de l'Accélérateur Linéaire, Université Paris-Sud, CNRS/IN2P3, Université Paris-Saclay, Orsay, France*
[8] *Fermi National Accelerator Laboratory, P. O. Box 500, Batavia, IL 60510, USA*
[9] *Department of Physics and Astronomy, University of California, Riverside, CA 92521, USA*
[10] *Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109, USA*
[11] *Department of Physics, California Institute of Technology, Pasadena, CA 91125, USA*
[12] *Dunlap Institute for Astronomy and Astrophysics & Department for Astronomy and Astrophysics, University of Toronto, ON M5S 3H4*
[13] *Department of Physics, The University of Texas at Dallas, Richardson, TX 75083, USA*
[14] *Univeristy of Arizona, Tucson, AZ 85721, USA*
[15] *Department of Physics and Astronomy, University of California, Irvine, CA 92697, USA*
[16] *Kavli Institute for Particle Astrophysics and Cosmology, Stanford, CA 94305-4085, USA*
[17] *SLAC National Accelerator Laboratory, Menlo Park, CA 94025, USA*
[18] *Stanford University, Stanford, CA, 94305, USA*
[19] *Brookhaven National Laboratory, Physics Department, Upton, NY 11973, USA*
[20] *Institute for Astronomy, Royal Observatory Edinburgh, Edinburgh EH9 3HJ, UK*
[21] *Department of Physics and Astronomy, University of Pittsburgh, Pittsburgh PA 15260*
[22] *Institute of Physics CAS, Prague, 182 21, CZ*

The Core Cosmology Library (CCL) provides routines to compute basic cosmological observables with validated numerical accuracy. These routines have been validated to an accuracy level, documented here, against the results of independent implementations. In the current version, predictions are provided for distances and background quantities, angular auto- and cross-spectra of cosmic shear and clustering, halo bias and the halo mass function. CCL uses different schemes to obtain the matter power spectrum, including analytical, phenomenological and other schemes calibrated through simulations. Fiducial specifications for the expected LSST galaxy distributions and clustering bias are also included, together with the capability of computing redshift distributions for a user-defined photometric red-

shift model. CCL is written in C, with a Python interface. In this note, we explain the functionality of the publicly released (CCL v0.99) library.

Contents

1. Introduction

In preparation for constraining cosmology with the Large Synoptic Survey Telescope (LSST), it is necessary to be able to produce rigorous theoretical predictions for the cosmological quantities that will be measured. The Core Cosmology Library[1] (CCL) aims to provide, in one library, a way of making predictions that are validated to a well-documented numerical accuracy, for the purpose of constraining cosmology with LSST. By constructing a cosmology library specifically with LSST in mind, it is possible to ensure that it is flexible, adaptable, and validated for all cases of interest, as well as user-friendly and appropriate for the needs of all working groups.

The Core Cosmology Library is written in C and incorporates the CLASS code (Blas et al. 2011) to provide predictions for the matter power spectrum. It also incorporates emulated power spectra from the cosmic emulator of Lawrence et al. (2017).[2] A Python wrapper is also provided for ease of use.

This note describes how to install CCL (Section 2), how CCL is documented (Section 3), its functionality (Section 4), the relevant unit tests (Section 5), directions for finding a CCL example (Section 6), the Python wrapper (Section 7), future plans (Section 8), means to contact the developers (Section 9), instructions on how to cite CCL (and CLASS, Section 10), and the license under which CCL is released (Section 11).

2. Installation

2.1. General installation

CCL is available as a Python package through PyPi. To install, simply run:

```
$ pip install pyccl
```

This should work as long as CMake is installed on your system (if it isn't, follow the detailed instructions below).

[1] https://github.com/LSSTDESC/CCL
[2] Future versions of the library will incorporate other power-spectrum libraries and methods.

CCL comes in two forms: a C library and a Python module. These components can be installed independently of each other, instructions are provided below.

For developers: Note that the installation procedure for development workflow is different than stated here. Detailed instructions are provided in the README.md **file in the main** CCL **directory.**

2.2. Dependencies

CCL requires the following software and libraries

- GNU Scientific Library GSL,[3] GSL-2.1 or higher.
- FFTW3[4] version 3.1 or above is required for computation of correlation functions.
- FFTlog[5] is provided within CCL, with minor modifications.
- The C library associated to the CLASS code. (Note that, if you want to use your own version of CLASS, you should follow the steps described in Section 2.7 below.)
- Angpow for computation of angular power spectra[6].

In addition, the build system for CCL requires

- The SWIG[7] Python wrapper.
- CMake[8] version 3.2 or above. This is the only requirement that needs to be manually installed (sew below).

2.3. Installing the dependencies

It is preferable to install GSL and FFTW on your system before building CCL, but only necessary if you want to properly install the C library, otherwise CMake will

[3] https://www.gnu.org/software/gsl/
[4] http://www.fftw.org
[5] http://casa.colorado.edu/~ajsh/FFTLog/ and https://github.com/slosar/FFTLog
[6] https://gitlab.in2p3.fr/campagne/AngPow
[7] http://www.swig.org/
[8] https://cmake.org/

automatically download and build the missing requirements in order to compile CCL.

For CMake, you can follow this set of instructions. On Ubuntu:

```
$ sudo apt-get install cmake
```

On MacOs X:

- Install using a DMG package from this download page: https://cmake.org/download/.
- Or install using a package manager (e.g., brew[9]), MacPorts[10], Fink[11]). For instance, with brew:

  ```
  $ brew install cmake
  ```

To install all the dependencies at once, and avoid having CMake recompiling them, for instance on Ubuntu:

```
$ sudo apt-get install cmake swig libgsl-dev libfftw3-dev
```

2.4. **Compile and install the C library**

To download the latest version of CCL:

```
$ git clone https://github.com/LSSTDESC/CCL.git
$ cd CCL
```

or download and extract the latest stable release from [here](https://github.com/LSSTDESC/CCL/releases). Then, from the base CCL directory run:

```
$ mkdir build && cd build
$ cmake ..
```

[9] https://brew.sh/
[10] https://www.macports.org/
[11] http://www.finkproject.org/

This will run the configuration script, try to detect the required dependencies on your machine and generate a Makefile. Once CMake has been configured, to build and install the library simply run for the `build` directory:

```
$ make
$ make install
```

Often admin privileges will be needed to install the library. If you have those just type:

```
$ sudo make install
```

Note: This is the default install procedure, but depending on your system you might want to customize the intall process. Here are a few common configuration options:

- **C compiler**: In case you have several C compilers on your machine, you will probably need to specify which one to use to `CMake` by setting the environment `CC` like so, **before** running `CMake`:

  ```
  $ export CC=gcc
  ```

- **Install directory**: By default, `CMake` will try to install `CCL` in `/usr/local`, if you would like to instead install `CCL` in a user-defined directory (for instance if you don't have admin privileges), you can specify it to `CMake` by running instead the following command:

  ```
  $ cmake -DCMAKE_INSTALL_PREFIX=/path/to/install ..
  ```

  This will instruct `CMake` to install `CCL` in the following folders: `/path/to/install/include`,`/path/to/install/share`,`/path/to/install/lib`.

Depending on where you install `CCL` your might need to add the installation path to your to your `PATH` and `LD_LIBRARY_PATH` environment variables. In the default case, it will look like:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/bin
```

To make sure that everything is working properly, you can run all unit tests after installation by running from the root CCL directory:

```
$ check_ccl
```

Assuming that the tests pass, you have successfully installed CCL!

If you ever need to uninstall CCL, run the following from the build directory:

```
$ make uninstall
```

You may need to prepend a sudo if you installed CCL in a protected folder.

2.5. Install the pyccl Python module

CCL also comes with a Python wrapper, called pyccl, which can be built and installed regardless of whether you install the C library. For convenience, we provide a PyPi hosted package which can be installed simply by running:

```
$ pip install pyccl
```

(Append --user for single user install.) This only assumes that CMake is available on your system, you don't need to download the source yourself.

You can also build and install pyccl from the CCL source, again **without necessarily installing the C library**. Download the latest version of CCL:

```
$ git clone https://github.com/LSSTDESC/CCL.git
$ cd CCL
```

And from the root CCL folder, simply run:

```
$ python setup.py install # append --user for single user install
```

The pyccl module will be installed into a sensible location in your PYTHONPATH, and so should be picked up automatically by your Python interpreter. You can then simply import the module using import pyccl.

You can quickly check whether `pyccl` has been installed correctly by running `python -c "import pyccl"` and checking that no errors are returned.

For a more in-depth test to make sure everything is working, run from the root `CCL` directory:

```
$ python setup.py test
```

This will run the embedded unit tests (may take a few minutes).

Whatever the install method, you can always uninstall the pyton wrapper by running:

```
$ pip uninstall pyccl
```

For quick introduction to `CCL` in Python, take a look at notebooks in `tests/`.

2.6. **Known installation issues**

In case you have several C compilers on your system, `CMake` may not default to the one you want to use. You can specify which C compiler will be used to compile `CCL` by setting the `CC` environment variable **before** calling any `cmake` or `python setup.py` commands:

```
$ export CC=gcc
```

2.7. **Compiling against an external version of CLASS**

The default installation procedure for `CCL` implies automatically downloading and installing a tagged version of `CLASS`. Optionally, you can also link `CCL` against a different version of `CLASS`. This is useful if you want to use a modified version of `CLASS`, or a different or more up-to-date version of the standard `CLASS`.

To compile `CCL` with an external version of `CLASS`, just run the following `CMake` command at the first configuration step of the install (from the `build` directory, make sure it is empty to get a clean configuration):

```
$ cmake -DEXTERNAL_CLASS_PATH=/path/to/class ..
```

the rest of the build process should be the same.

2.8. Creating a Docker Image

The Dockerfile to generate a Docker image is included in the CCL repository as Dockerfile. This can be used to create an image that Docker can spool up as a virtual machine, allowing you to utilize CCL on any infrastructure with minimal hassle. The details of Docker and the installation process can be found at https://www.docker.com/. Once Docker is installed, it is a simple process to create an image. In a terminal of your choosing (with Docker running), type the command `docker build -t ccl .` in the CCL directory.

The resulting Docker image has two primary functionalities. The first is a CMD that will open Jupyter notebook tied to a port on your local machine. This can be used with the following run command: `docker run -p 8888:8888 ccl`. You can then access the notebook in the browser of your choice at `localhost:8888`. The second is to access the bash itself, which can be done using `docker run -it ccl bash`.

This Dockerfile currently contains all installed C libraries and the Python wrapper. It currently uses continuumio/anaconda as the base image and supports ipython and Jupyter notebook. There should be minimal slowdown due to the virtualization.

3. Documentation

CCL has basic doxygen[12] documentation for its C routines. This can be found in the directory `doc/html` within the CCL repository by opening the `index.html` file in your browser. The `python` routines are documented in situ; you can view the documentation for a function by calling `help(function_name)` from within `python`. A Readthedocs interface is also available, where you can browse the documentation, at: https://readthedocs.org/projects/ccl/.

4. Functionality

[12] http://www.stack.nl/~dimitri/doxygen/

Table 1. Physical constants in CCL.

	G_{Newt}	k_b	σ_{SB}	h	c	eV	$M_\odot$	pc
NIST	-3.0e-05	-1.4e-06	-5.9e-06	1.6e-07	0.0e+00	8.4e-08	–	–
PDG 2013	-6.0e-05	-1.1e-06	-4.8e-06	9.2e-08	0.0e+00	4.9e-08	-2.0e-04	1.1e-09
GSL	-1.9e-04	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	-4.5e-11
CLASS	0.0e+00	0.0e+00	–	0.0e+00	0.0e+00	0.0e+00	–	-6.7e-11

4.1. Physical constants

We have performed a comparison of the physical constants used in CCL and included dependencies and external sources. See Table 1 for percent differences of the constants between these sources.

4.2. Supported cosmological models

Ultimately, CCL plans to incorporate theoretical predictions for all cosmological models of interest to LSST. Currently, the following families of models are supported:

- Flat ΛCDM
- wCDM and the CPL model ($w_0 + w_a$, Chevallier & Polarski 2001 and Linder 2003)
- Non-zero curvature (K)
- All of the above, plus an arbitrary, user-defined modified growth function (see description in Section 4.7)
- The μ, Σ parameterisation of modifications to General Relativity in the quasistatic regime (**??**)
- Massive neutrinos, in combination with any of the above except the user-defined modified growth function and the μ, Σ modified gravity parameterization.

Not all features of CCL are available for all models. For a guide to which predictions are available for each model, see Table 2. Note that if users install their own version of CLASS, CCL can then make predictions for a more extended set of cosmologies.

Table 2. Cosmologies implemented in CCL.

Observable/Model	flat ΛCDM	ΛCDM+K	ΛCDM + m_ν	wCDM	$w_0 + w_a$	MG growth	μ, Σ
Distances	✓	✓	✓	✓	✓	X	✓
Growth	✓	✓	X	✓	✓	✓	✓
$P_m(k,z)$	✓	✓	✓	✓	✓	X	✓
Halo Mass Function	✓	✓	X	✓	✓	X	X
C_l, number counts	✓	✓	X	✓	✓	X	✓
C_l, weak lensing only	✓	✓	✓	✓	✓	X	✓

Users should take care to understand the validity of the CCL assumptions for their own models.

In its default configuration, CCL adopts the nonlinear matter power spectrum from CLASS through the Halofit implementation and the Tinker mass function for cluster number counts.

4.3. **Creating a cosmology**

To use CCL, the first step is to create a `ccl_cosmology` structure, containing all of the information required to compute cosmological observables. A `ccl_cosmology` structure can be generated using the information from a `ccl_parameters` object and a `ccl_configuration` object.

`ccl_parameters` objects contain information about the cosmological parameters of a given model, and are initialized using one of the following routines (the full syntax for each function can be found in the header file `ccl_core.h`):

- `ccl_parameters_create(double Omega_c, double Omega_b, double Omega_k, double N_eff, double *mnu, ccl_mnu_type_label mnu_type, double w0, double wa, double h, double norm_pk, double n_s, double mu_0, double sigma_0, int nz_mgrowth,double *zarr_mgrowth,double *dfarr_mgrowth, int *status)`: general `ccl_parameters` constructor supporting all the models described above.
- `ccl_parameters_create_flat_lcdm(...)`: particular constructor for flat ΛCDM cosmologies.
- `ccl_parameters_create_flat_wcdm(...)`: constant w cosmologies.

- `ccl_parameters_create_flat_wacdm(...)`: $w_0 + w_a$.
- `ccl_parameters_create_lcdm(...)`: curved ΛCDM cosmologies.

The argument `norm_pk` can be used to specify the power spectrum normalization in terms of either σ_8 or $A_{\rm s}$. The `ccl_parameters_create` functions assume σ_8 normalization if `norm_pk` $>$ `1.0e` $-$ `5` and $A_{\rm s}$ normalization otherwise.

`ccl_configuration` objects contain information about the prescriptions to be used to compute transfer functions, power spectra, mass functions, etc. A default `ccl_configuration` object is made available as `default_config`, which specifies that CLASS will be used to compute transfer functions, HaloFit will be used to calculate the matter power spectrum, there will be no impact of baryons included and the Tinker 2010 prescription will be used to compute the halo mass function.

After initializing instances of `ccl_parameters` and `ccl_configuration`, use the function `ccl_cosmology_create(ccl_parameters, ccl_configuration)` to return a pointer to a `ccl_cosmology` structure. You will need to pass this pointer around to every CCL function.

An example of using `CCL` is provided in Section 6. The `README` file has additional extensive documentation for the example run, as well as installation.

Creating a cosmology with massive neutrinos—In the parameter construction routines above, `mnu` is a pointer to either a single value, to be interpreted as the sum of neutrino masses, or to an array containing the three neutrino masses, both in eV. `mnu_type` is a label which indicates whether `mnu` points to sum of the masses or to an array of three mass values. In the former case, `mnu_type` also defines which convention should be used to split the sum into three neutrino masses for calculations (options are normal hierarchy, inverted hierarchy, or an equal split).

To set up a cosmology with massive neutrinos, the user should pass `mnu`, `mnu_type`, and `N_eff` to the parameter construction routine. When working from the python wrapper, it suffices to pass `mnu` and `N_eff`; `CCL` will infer whether `mnu` is a set of masses or a sum. In the latter case, the default behaviour is to split the sum into masses consistent with the normal hierarchy, but an inverted hierarchy or equal splitting can be requested by passing `mnu_type`.

In the case where `mnu` corresponds to a sum, we first allocate the three neutrino masses according to the requested convention. The default convention is the normal hierarchy, but users may also request either an inverted hierarchy or a split

of the sum into three equal masses. In the latter case it is trivial to compute the three masses. In the former two cases, it is slightly more complicated. The relevant neutrino-mass quantity which has been determined via particle physics experiments is the square of the different of masses (up to a sign for one of the differences, hence the existence of the two hierarchies). Because we know the square of the differences rather than the differences themselves, to get the neutrino masses given these differences and the sum we must solve a set of quadratic equations. This is most easily accomplished by making use the knowledge that the mass difference between two of the species is small; this allows us to taylor expand about this difference and greatly simplifies solving this system of equations. The resulting expressions for the neutrino masses are implemented.

Having then a set of three masses regardless of `mnu_type`, we check for which of the three masses the corresponding neutrino species is non-relativistic today ($m_\nu > 0.00017$, Lesgourgues & Pastor 2012), and obtain a number of massive neutrinos `N_nu_mass`. We use this along with the `N_eff` value to set the number of relativistic neutrinos species `N_nu_rel`. We must be slightly careful in doing so, as for massive neutrinos only we follow `CLASS` in modifying the relationship between the temperature of the CMB and the neutrino temperature:

$$T_\nu^{\rm eff} = T_{\rm CMB} T_{\rm NCDM} \tag{1}$$

where the above defines $T_{\rm NCDM}$, an adhoc modification to the equality between $T_{\rm CMB}$ and $T_\nu^{\rm eff}$. We follow the nomenclature of `CLASS` here, but we emphasize that $T_{\rm NCDM}$ is a dimensionless scaling factor, not a temperature. Setting $T_{\rm NCDM} = 0.71611$ ensures that $m_\nu/\Omega_\nu^0 = 93.14{\rm eV}$, in agreement with second-order theoretical calculations which correctly take into account QED effects and electron / positron annihilation (Mangano et al. 2005). Therefore to get `N_nu_rel` consistent with the `N_eff` passed by the user, we do:

$$\texttt{N\_nu\_rel} = \texttt{N\_eff} - (T_{\rm NCDM})^4 \left(\frac{4}{11}\right)^{-\frac{4}{3}} \texttt{N\_nu\_mass}. \tag{2}$$

For easier initiation of cosmologies with massive neutrinos, the suffix '`_nu`' may be appended to the last four `ccl_parameters_create` functions in the previous section. Using these functions without the `_nu` suffix will set the neutrino masses to 0 and the effective number of neutrino species to 3.046.

It may sometimes be preferable or necessarily to specify a cosmology in terms of Ω_ν^0 for massive neutrinos instead of m_ν. To facilitate this, `CCL` includes a convenience function `ccl_nu_masses`, which takes as input Ω_ν^0 for massive neutrinos, the temperature of the CMB, and a label specifying how the neutrino mass should be

split amongst species similarly to above. It then outputs a pointer to the resulting neutrino mass(es).

4.4. **Distances**

The routines described in this subsection are implemented in `ccl_background.c`.

The Hubble parameter is calculated as

$$\frac{H(a)}{H_0} = a^{-3/2}\Big(\Omega_{M,0} + \Omega_{\Lambda,0}a^{-3(w_0+w_a)}\exp[3w_a(a-1)] + \Omega_{K,0}a + (\Omega_{g,0} + \Omega_{\nu,\mathrm{rel}})a^{-1} + \Omega_{\nu,\mathrm{m}}(a)a^3\Big)^{\frac{1}{2}}. \tag{3}$$

The radial comoving distance is calculated via a numerical integral,

$$\chi(a) = c\int_a^1 \frac{da'}{a'^2 H(a')}. \tag{4}$$

The transverse comoving distance is computed in terms of the radial comoving distance as:

$$r(\chi) = \begin{cases} k^{-1/2}\sin(k^{1/2}\chi) & k > 0 \\ \chi & k = 0 \\ |k|^{-1/2}\sinh(|k|^{1/2}\chi) & k < 0 \end{cases} \tag{5}$$

The usual angular diameter distance is $d_A = a\,r(a)$, and the luminosity distance is $d_L = r(a)/a$.

`CCL` can also compute the distance modulus, defined as,

$$\mu = 5\log_{10}(d_L/\mathrm{pc}) - 5 \tag{6}$$

and $a(\chi)$, the inverse of $\chi(a)$.

4.5. **Density parameter functions**

The routines described in this subsection are implemented in `ccl_background.c`.

The density parameter functions $\Omega_X(a)$ can be calculated for six components:

- matter density parameter $\Omega_M(a) = \Omega_{M,0} H_0^2/(a^3 H^2(a))$,
- dark energy density parameter $\Omega_\Lambda(a) = \Omega_{\Lambda,0} a^{-3(1+w_0+w_a)} \exp[3w_a(a-1)] H_0^2/H^2(a)$,
- radiation density parameter $\Omega_g(a) = \Omega_{g,0} H_0^2/(a^4 H^2(a))$,
- curvature density parameter $\Omega_K(a) = \Omega_{K,0} H_0^2/(a^2 H^2(a))$,
- massless neutrino density parameter $\Omega_{\nu,\mathrm{rel}}(a) = \Omega_{\nu,\mathrm{rel},0} H_0^2/(a^4 H^2(a))$,
- massive neutrino density parameter $\Omega_{\nu,\mathrm{m}}(a)$,

all using the Hubble parameter defined in equation 3.

For massive neutrinos, $\Omega_{\nu,\mathrm{m}}(a)$ is calculated by calling a set of functions contained in `ccl_neutrinos.c`. For each species of massive neutrino with mass m_ν^i, we define

$$\mu^i = \frac{m_\nu^i a}{T_\nu^{\mathrm{eff}}} \tag{7}$$

in units such that μ is dimensionless. We conduct once the phase-space integral required to get the neutrino density (the integral over x in equation 8 below), and store the result as a `gsl` spline for subsequent access. Given then the value of this integral, we multiply by appropriate factors to obtain $\Omega_{\nu,\mathrm{m}}(a)$:

$$\Omega_{\nu,\mathrm{m}}(a) = \sum_{i=1}^{N_\nu} \frac{8\pi^2(\pi k_b)^3 k_b}{15(ch_{\mathrm{P}})^3} \frac{8\pi G}{3h^2c^2} \left(\frac{T_\nu^{\mathrm{eff}}}{a}\right)^4 \left(\frac{7}{8}\int_0^{x_{\mathrm{max}}} dx\, x^2 \frac{\sqrt{x^2+(\mu^i)^2}}{\exp(x)+1}\right) \tag{8}$$

where h_{P} is Planck's constant and h is $H_0/100$ with H_0 in units of km / s / Mpc. x_{max} is set to 1000. The final bracketed term which includes the phase-space integral can be simplified in the limit where μ is very large or very small: for small μ, it is set to $\frac{7}{8}$, and for large μ, it becomes $\frac{5\zeta(3)}{18\pi^4}\mu \sim 0.2776\mu$.

4.6. **Functions of the physical density**

The routines described in this subsection are implemented in `ccl_background.c`.

The physical density $\rho_X(a)$ can be calculated for seven components:

- critical density $\rho_{\mathrm{crit}}(a) = \frac{3H^2(a)}{8\pi G} = \rho_{\mathrm{crit},0} H^2(a)/H_0^2$,
- matter density $\rho_M(a) = \rho_{\mathrm{crit}}(a)\Omega_M(a) = \rho_{\mathrm{crit},0}\Omega_{M,0}/a^3$,

- dark energy density parameter $\rho_\Lambda(a) = \rho_{\rm crit,0}\Omega_{\Lambda,0}a^{-3(1+w_0+w_a)}\exp[3w_a(a-1)]$,
- radiation density parameter $\rho_g(a) = \rho_{\rm crit,0}\Omega_{g,0}/a^4$,
- curvature density parameter $\rho_K(a) = \rho_{\rm crit,0}\Omega_{K,0}/a^2$,
- massless neutrino density parameter $\rho_{\nu,\rm rel}(a) = \rho_{\rm crit,0}\Omega_{\nu,\rm rel,0}/a^4$,
- massive neutrino density parameter $\Omega_{\nu,\rm m}(a) = \rho_{\rm crit,0}\Omega_{\nu,\rm m}(a)H^2(a)/H_0^2$,

where $\Omega_{\nu,\rm m}(a)$ is given by equation 8 and the Hubble parameter by equation 3. CCL moreover allows for comoving physical densities $\rho_{X,\rm comoving}(a) = \rho_X(a)a^3$.

4.7. **Growth function**

The routines described in this subsection are implemented in `ccl_background.c`. To compute the growth function, $D(a)$, the growth factor of matter perturbations, CCL solves the following differential equation:

$$\frac{d}{da}\left(a^3H(a)\frac{dD}{da}\right) = \frac{3}{2}\Omega_M(a)aH(a)D, \tag{9}$$

using a Runge-Kutta Cash-Karp algorithm.

In doing this, CCL simultaneously computes the growth rate $f(a)$, defined as:

$$f(a) = \frac{d\ln D}{d\ln a}. \tag{10}$$

CCL provides different functions that return the growth normalized to $D(a=1) = 1$ and to $D(a \ll 1) \to a$.

Note that the above is strictly valid for a Universe containing only dust-like matter components. A scale-independent growth rate is, for example, ill-defined in the presence of massive neutrinos; therefore CCL will raise an error if the user attempts to calculate the growth rate or growth factor in a cosmology with massive neutrinos.

Currently CCL allows for an alternative 'modified gravity' cosmological model defined by a regular background $(w_0 + w_a)$CDM (with arbitrary K) as well as a user-defined $\Delta f(a)$, such that the true growth rate in this model is given by $f(a) = f_0(a) + \Delta f(a)$, where $f_0(a)$ is the growth rate in the background model. Note that this model is only consistently implemented with regards to the computation of the linear growth factor and growth rates (which will also scale the linear

power spectrum). All other CCL functions (including the non-linear power spectrum) will ignore these modifications. This model, and the interpretation of the predictions given by CCL, should therefore be used with care.

4.8. Matter power spectrum

There are several options for obtaining the matter power spectrum in CCL. The routines described in this subsection are implemented in ccl_power.c.

4.8.1. BBKS

CCL implements the analytical BBKS approximation to the transfer function (Bardeen et al. 1986), given by

$$T(q \equiv k/\Gamma h\mathrm{Mpc}^{-1}) = \frac{\ln[1+2.34q]}{2.34q}[1+3.89q+(16.2q)^2+(5.47q)^3+(6.71q)^4]^{-0.25} \tag{11}$$

where $\Gamma = \Omega_m h$. The power spectrum is related to the transfer function by $P(k) = 2\pi^2\Delta^2(k)/k^3 \propto T^2(k)k^{n_s}$, where $\Delta(k)$ is the dimensionless power spectrum and n_s is the spectral index. The normalization of the power spectrum is defined at $z = 0$ by setting σ_8 to its value today. The BBKS power spectrum option is primarily used as a precisely-defined input for testing the numerical accuracy of CCL routines (as described in Sect. 5), and it is not recommended for other uses.

4.8.2. Eisenstein and Hu

CCL also provides an approximation to the matter power spectrum as implemented by Eisenstein & Hu (1998) (E&H; we refer the reader to this paper for a detailed discussion of the fitting formulae).[13]

The Eisenstein & Hu and BBKS approximations are not very accurate (generally only to within a few percent), and so should not be used to derive precise cosmological constraints. They are, however, computationally faster, and therefore useful for code testing and comparison.

[13] Note that the implementation in CCL modifies Eq. 5 of Eisenstein & Hu (1998) using $a^{-1} = 1+z$ instead of the approximation $a^{-1} \sim z$. The difference in the resulting power spectra is negligible, but larger than 1 part in 10^4 for $k < 10\,h\,\mathrm{Mpc}^{-1}$.

4.8.3. **CLASS**

The fiducial configuration calls the `CLASS` software (Blas et al. 2011) within `CCL` to obtain either linear or nonlinear matter power spectra at given redshifts. On initializing the cosmology object, we construct a bi-dimensional spline in k and the scale-factor which is then evaluated by the relevant routines to obtain the matter power spectrum at the desired wavenumber and redshift. The relevant routines can be found within `ccl_power.c`. CLASS currently computes the non-linear power spectrum using the HaloFit prescription of Takahashi et al. (2012).

As discussed in Section 2.7, the user can compile `CCL` with an external version of `CLASS` as well.

4.8.4. **Cosmic emulator**

The cosmic emulator of Lawrence et al. (2017) is integrated into `CCL` and it is available as one of the matter power spectrum and transfer function methods (both have to be specified). The cosmic emulator provides predictions for the matter power spectrum based on an interpolation over simulation outputs spanning a defined set of cosmological parameter values.

The emulator provides accurate predictions for the nonlinear matter power spectrum, at the 1% level, for $z \leq 2$ and in the wavenumber range $k = [10^{-3}, 5]$ Mpc^{-1}. If a redshift above $z = 2$ is passed, the emulator will quit and return an error message. For k values below and above the previously specified range, we extrapolate in the manner specified in the following section.

The allowed range of cosmological parameters is as follows:

$$\begin{aligned} 0.12 \leq \Omega_m h^2 \leq 0.155, \\ 0.0215 \leq \Omega_b h^2 \leq 0.0235, \\ 0.7 \leq \sigma_8 \leq 0.9, \\ 0.55 \leq h \leq 0.85, \\ 0.85 \leq n_s \leq 1.05, \\ -1.3 \leq w_0 \leq -0.7, \\ -1.73 \leq w_a \leq -0.7, \\ 0.0 \leq \Omega_\nu h^2 \leq 0.01. \end{aligned} \tag{12}$$

Actually, w_a and w_0 are constrained jointly to be $0.3 \leq (-w_0 - w_a)^{1/4}$. Note that `CCL` only allows a subregion within this parameter space. For models in which

w(z) crosses -1 at some given redshift, CLASS will crash because this value corresponds to a true cosmological constant, which by definition should have no perturbations.[14] The linear matter power spectrum is still obtained from CLASS as described in the section above.

Neutrino species—The emulator is set up to consider $N_{\rm eff} = 3.04$ and to allow the user to provide $\Omega_\nu h^2$ in order to set the neutrino mass, where it is assumed that the corresponding mass is split equally amongst three neutrinos. This is different from the standard CCL configuration, which takes as input the mass(es) of neutrinos in eV, m_ν. The assumption of an equal split of masses amongst three neutrino species is also different from the default CCL choice to split the mass amongst species according to the normal hierarchy. For best compatibility with the emulator, we therefore include a ccl_configuration element emulator_neutrinos_method, which defaults to ccl_strict but can be set to ccl_equalize if desired:

- ccl_strict requires the user to pass either a set of three equal neutrino masses, or to pass a sum of masses and to set ccl_mnu_type such that the sum is split equally amongst species. If other options are selected when the emulator is in use, CCL will raise an error and exit. This default option ensures full self-consistency within quantities calculated with a single CCL cosmology.
- ccl_equalize allows the user to pass an unequal set of neutrino masses, or to pass a sum with ccl_mnu_type set such that the sum is not split equally amongst speices. In this case, CCL will pass redistributed equal neutrino masses to the emulator. This choice may result in internal inconsistency amongst different quantities calculated with the same CCL cosmology, or indeed even internal inconsistency between the linear and nonlinear parts of the same power spectrum. This option should only be selected for convenience if you are sure you don't care about the mass splitting between neutrino speices.

Note also that because the emulator is constructed with $N_{\rm eff} = 3.04$, the user is required to pass this when the emulator is in use. If this is not the case, CCL will quit and issue an error.

Notice that ideally we would like to pass a non-integer number of massive neutrino species for best compatibility with the emulator set-up. However, since this is not possible within CCL, we have opted to verify that neither the growth function nor the comoving radial distance computation are affected by this approxi-

[14] We thank Emilio Bellini and Miguel Zumalacárregui for clarifying this for us.

mation to more than 10^{-4} in the range $0.01 < a < 1$, where a is the scale factor. We have verified this by comparing this prediction for a fiducial cosmology $\{\Omega_c = 0.27, \Omega_b = 0.049, h = 0.67, \sigma_8 = 0.8, n_s = 0.96\}$ with the following neutrino parameters: $\{N_{\nu,\mathrm{rel}}, N_{\nu,\mathrm{mass}}, m_\nu\} = \{0.00641, 3, 0.06\mathrm{eV}\}$, $\{N_{\nu,\mathrm{rel}}, N_{\nu,\mathrm{mass}}, m_\nu\} = \{0, 3.04, 0.06\mathrm{eV}\}$ and $\{N_{\nu,\mathrm{rel}}, N_{\nu,\mathrm{mass}}, m_\nu\} = \{0, 3.046, 0.06\mathrm{eV}\}$. The discrepancy between distances and growth results between these choices of neutrino parameters can raise above 10^{-4} at $a < 0.01$ and the user should be mindful of this for their particular application.

4.8.5. **Impact of baryons**

`CCL` incorporates the impact of baryons on the total matter power spectrum via the "baryonic correction model" (BCM) of Schneider & Teyssier (2015). This is implemented through a flag in the configuration object, which currently accepts the following options: `ccl_nobaryons` or `ccl_bcm`. When `ccl_bcm` is set, the nonlinear matter power spectrum (whichever the method that provides it) is multiplied by a correction factor $F(k, z)$ which models the impact of baryons.

The main consequences of baryonic processes are: to suppress the power spectrum at intermediate scales ($k \sim$ a few $h/$Mpc) due to the ejection of gas by Active Galactic Nuclei feedaback, and to enhance it at smaller scales due to enhanced cooling. Three effective parameters govern the contribution of baryonic processes to modifying the total matter power spectrum:

- $\log_{10}[M_c/(\mathrm{M}_\odot/h)]$: the mass of the clusters responsible for feedback, which regulates the amount of suppression of the matter power spectrum at intermediate scales;
- η_b: a dimensionless parameter which determines the scale at which suppression peaks;
- and k_s [$h/$Mpc]: the wavenumber that determines the scale of the stellar profile.

If the BCM parameters are not specified and the user sets `CCL` to compute the power spectrym with baryonic feedback included, `CCL` will assume the default parameters of Schneider & Teyssier (2015). The user may pass these explicitly if desired.

Other baryonic impact models might be incorporated in the future as they become available.

4.8.6. **Spline parameters & the INI file**

All spline and grid parameters relevant for computing quantities such as distances, growth functions, power spectra, and halo mass functions are defined in the `ccl_params.ini` file in the `include` folder of the repository. This file allows the user to configure the following constants:

- `A_SPLINE_NLOG`: number of points in the logarithmic part of the scale factor spline. The default is `A_SPLINE_NLOG`$= 250$.
- `A_SPLINE_NA`: number of points in the linear part of the scale factor spline. The default is `A_SPLINE_NA`$= 250$.
- `A_SPLINE_MINLOG`: minimum value for the scale factor spline. The default is `A_SPLINE_MINLOG`$= 0.0001$. This only applies to background quantities.
- `A_SPLINE_MIN`: maximum value for the logarithmic part and minimum value for the linear part of the scale factor spline. The default is `A_SPLINE_MIN`$= 0.01$. This only applies to background quantities.
- `A_SPLINE_MAX`: maximum value for the scale factor spline. The default is `A_SPLINE_MAX`$= 1$.
- `LOGM_SPLINE_DELTA`: spacing of the mass function spline (logarithm). The default is `LOGM_SPLINE_DELTA`$= 0.025$.
- `LOGM_SPLINE_NM`: number of points in the mass function spline (logarithm). The default is `LOGM_SPLINE_NA`$= 440$.
- `LOGM_SPLINE_MIN`: minimum value for the mass function spline (logarithm). The default is `LOGM_SPLINE_MIN`$= 6$.
- `LOGM_SPLINE_MAX`: maximum value for the mass function spline (logarithm). The default is `LOGM_SPLINE_MAX`$= 17$.
- `A_SPLINE_NLOG_PK`: number of bins for the logarithmic part of the scale factor in the case of two-dimensional splines (where the second variable is the wavenumber). The default is `A_SPLINE_NLOG_PK`$= 11$.
- `A_SPLINE_NA_PK`: number of bins for the linear part of the scale factor in the case of two-dimensional splines (where the second variable is the wavenumber). The default is `A_SPLINE_NA_PK`$= 40$. In the case of the emulator's non-linear power spectrum, the maximum wavenumber is `K_MAX_SPLINE`$= 5/$Mpc, the interpolation is only linear and we use `A_SPLINE_NA_PK`$= 40$ as default.

- A_SPLINE_MINLOG_PK: minimum value for the scale factor spline. The default is A_SPLINE_MINLOG_PK$= 0.01$. This only applies for the 2D power spectrum splines. For the cosmic emulator and in the case of the nonlinear matter power spectrum, we use a linear spline only, adopting A_SPLINE_MIN_PK$= 1/3$.
- A_SPLINE_MIN_PK: maximum value for the logarithmic part and minimum value for the linear part of the scale factor spline. The default is A_SPLINE_MIN_PK$= 0.1$. This only applies for the 2D power spectrum splines.
- K_MAX_SPLINE: The maximum value of wavenumber considered in the spline. This is explained in more detail in the coming subsections. The default is K_MAX_SPLINE$= 50/$Mpc. In the case of the emulator's nonlinear power spectrum, the maximum wavenumber is K_MAX_SPLINE$= 5/$Mpc.
- K_MAX: The maximum value of wavenumber when integrating over k. The default is K_MAX$= 1000/$Mpc.
- K_MIN_DEFAULT: The minimum value of wavenumber when integrating over k. The default is K_MIN_DEFAULT$= 5 \times 10^{-5}/$Mpc.
- N_K: Number of bins per decade for the wavenumber. The default is N_K$= 167$.
- N_K_3DCOR: Number of bins per decade for the wavenumber in the 3d correlation function calculation. The default is N_K_3DCOR = 100,000.

The precision parameters of GSL routines can also be set in ccl_params.ini. The parameters are organized in a hierarchal fashion, such that more specialized parameters take precedence over more general ones. For example, there is a general tolerance parameter, a tolerance parameter for integrals, and a tolerance parameter for the Limber integral. By default all are set to the value of the general tolerance. Changing the integration tolerance changes the tolerance for all integrals but not for root-finding routines, for example, and changing the Limber integral tolerance only affects that integral and no other. The parameters are:

- GSL_EPSREL: Relative tolerance for GSL routines. Default is 10^{-4}.
- GSL_N_ITERATION: Maximum number of iterations allowed for GSL routines. This parameter also controls the size of the integration workspaces. Default is 1000.
- GSL_INTEGRATION_GAUSS_KRONROD_POINTS: Number of Gauss-Kronrod points in the QAG integration routines. Default is GSL_INTEG_GAUSS41 (numerical value 4).

- GSL_INTEGRATION_EPSREL: Relative tolerance for GSL integration routines. Default is GSL_EPSREL.
- GSL_INTEGRATION_DISTANCE_EPSREL: Relative tolerance for GSL integration routines for distance calculations. Default is 10^{-6}.
- GSL_INTEGRATION_DNDZ_EPSREL: Relative tolerance for GSL integration routines for $\frac{dN}{dz}$ calculations. Default is 10^{-6}.
- GSL_INTEGRATION_SIGMAR_EPSREL: Relative tolerance for GSL integration routines for σ_R calculations. Default is 10^{-5}.
- GSL_INTEGRATION_NU_EPSREL: Relative tolerance for GSL integration routines for neutrino calculations. Default is 10^{-7}.
- GSL_INTEGRATION_NU_EPSABS: Absolute tolerance for GSL integration routines for neutrino calculations. Default is 10^{-7}.
- GSL_INTEGRATION_LIMBER_GAUSS_KRONROD_POINTS: Number of Gauss-Kronrod points in the QAG integration routines for the Limber integral. Default is GSL_INTEGRATION_GAUSS_KRONROD_POINTS.
- GSL_INTEGRATION_LIMBER_EPSREL: Relative tolerance for GSL integration routines for Limber integral. Default is GSL_EPSREL.
- GSL_ROOT_EPSREL: Relative tolerance for GSL root-finding routines. Default is GSL_EPSREL.
- GSL_ROOT_N_ITERATION: Maximum number of iterations allowed for GSL root-finding routines. Default is GSL_N_ITERATION.
- GSL_ODE_GROWTH_EPSREL: Relative tolerance for GSL ODE-solving routines for growth factor calculation. Default is 10^{-6}.

The GSL parameters are initialized to their default values and do not need to be listed in ccl_params.ini if no adjustments are necessary.

Note that a copy of ccl_params.ini is installed along with the library, so changing the version of this file inside the source directory will not have any effect unless you reinstall.

For the matter power spectrum, the spline is performed in two variables: the logarithmically-spaced wavenumber and the linearly-spaced scale factor. Splining the CLASS output leads to some precision loss (compared to direct outputs from CLASS). We quantify this, along with the impact of extrapolation, in the following subsection.

4.8.7. **Extrapolation for the nonlinear power spectrum**

The computation of the power spectrum from CLASS can be significantly sped up by extrapolating in the range $k >$ K_MAX_SPLINE and $k <$ K_MIN. In this section, we describe the implementation of the extrapolation and the accuracy attained. These tests are performed in a flat ΛCDM cosmology with $\Omega_c = 0.25$, $\Omega_b = 0.05$, $A_s = 2.1 \times 10^{-9}$, $h = 0.7$ and $n_s = 0.96$.

We first describe the extrapolation at high wavenumbers. The introduction of the parameter K_MAX_SPLINE allows us to spline the matter power spectrum within the cosmo structure up to that value of k (in units of $1/$Mpc). A separate K_MAX parameter sets the limit for evaluation of the matter power spectrum. The range between K_MAX_SPLINE$< k <$K_MAX is evaluated by performing a second order Taylor expansion in $\ln k$ within the static routine ccl_power_extrapol_highk.

First, we compute the first and second derivative of the $\ln P(k,z)$ at $k_0 =$ K_MAX $- 2\Delta \ln k$ by computing the numerical derivatives by finite differences using GSL. The fiducial choice for $\Delta \ln k$ is 10^{-2}. We then apply a second order Taylor expansion to extrapolate the matter power spectrum to $k >$K_MAX_SPLINE. The Taylor expansion gives

$$\ln P(k,z) \simeq \ln P(k_0,z) + \frac{d \ln P}{d \ln k}(\ln k_0, z)(\ln k - \ln k_0) + \frac{1}{2}\frac{d^2 \ln P}{d \ln k^2}(\ln k_0, z)(\ln k - \ln k_0)^2. \tag{13}$$

The accuracy of this approximation is shown in Figure 1 for redshifts $z = 0$, $z = 3$ and $z = 20$. We compare the nonlinear matter power spectrum at these redshifts computed with the previously described approximation, to the matter power spectrum obtained by setting the power spectrum splines to high values. We find that for typical values of $\Delta \ln k = 10^{-2}$ and K_MAX_SPLINE$= 50/$Mpc, $\ln P$ has converged to an accuracy that surpasses the expected impact of baryonic effects on the matter power spectrum at $k > 10/$Mpc. (For an estimate of the impact of baryons on the total matter power spectrum, see Schneider & Teyssier 2015.) The lower K_MAX_SPLINE is, the faster CCL will run. The optimum choice of K_MAX_SPLINE is left to the user for their particular application.

We also extrapolate the power spectrum at small wavenumbers within the static routine ccl_power_extrapol_lowk. In this case, the power spectrum below K_MIN is obtained by a power-law extrapolation with index n_s:

$$\log P(k < \text{K\_MIN}, z) = \log P(\text{K\_MIN}, z) + n_s(\log k - \log \text{K\_MIN}) \tag{14}$$

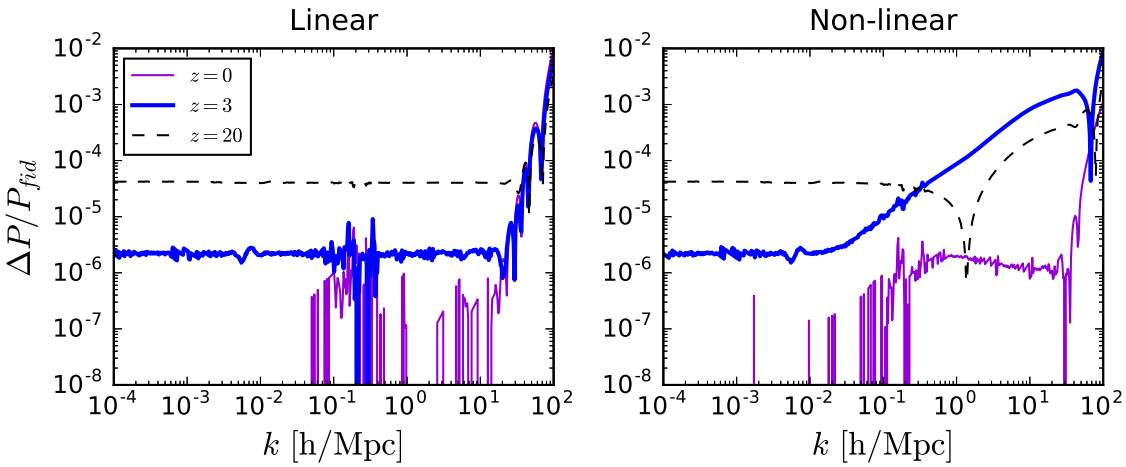

Figure 1. The relative error compared to power spectra produced with high values of the power spectrum splines, P_{fid}, produced by splining the matter power spectrum up to K_MAX_SPLINE and extrapolating beyond this value with a second order Taylor expansion the natural logarithm of the matter power spectrum. The left panel shows the relative errors for the linear matter power spectrum at $z = 0$, $z = 3$ and $z = 20$. The right panel shows the results for the non-linear matter power spectrum at the same redshifts. The standard CCL parameters adopted are those corresponding to the black dashed curve. For comparison, the impact of baryonic physics on the matter power spectrum is $\sim 10\%$ at $k = 1$/Mpc (Schneider & Teyssier 2015).

The value adopted for K_MIN depends on the choice of power spectrum method. For CLASS and the nonlinear power spectrum, we adopt K_MIN that coincides with the smallest wavenumber output by CLASS, K_MIN$= 7 \times 10^{-6}$/Mpc.[15] Note that this parameter is different from K_MIN_DEFAULT, which sets the minimum k for integrations and which is set by default to K_MIN_DEFAULT$= 5 \times 10^{-5}$/Mpc. Hence, in practice, no extrapolation is occurring in this case, unless the user specifically asks for an output power spectra below K_MIN_DEFAULT for their own purposes.

4.8.8. **Extrapolation for the linear power spectrum**

With the implementation described in the previous section, the power spectrum splines are initialized up to K_MAX_SPLINE. This is also true for the linear matter power spectrum, which is used within CCL in particular to obtain σ_8 (see Eq. 47). We have tested here how the procedure described in the previous section affects the convergence of the linear matter power spectrum. The result is shown in Figure 1. For some applications that use the linear power spectrum, the user might need to increase the value of K_MAX_SPLINE.

[15] For BBKS, the power spectrum is computed analytically at all k, there is no extrapolation. For the Eisenstein & Hu implementation, the splines of the power spectrum span K_MIN_DEFAULT$<$ k $<$K_MAX_SPLINE, so there is only extrapolation at high k.

As in the previous section, the power spectrum at small wavenumber is extrapolated using a power-law. This extrapolation is performed below a fiducial value of K_MIN_DEFAULT$= 5 \times 10^{-5}$.

We have found that changing N_A to 200, or changing the sampling of the wavenumber to 5000 points, does not change the results presented in Figures 1 in this section.

4.8.9. **Wishlist for the future**

We plan to implement the following power spectrum methods in the future:

- CAMB,
- other emulators,
- Halo model/HOD.

4.8.10. **Normalization of the power spectrum**

There are two alternative schemes for normalization of the matter power spectrum. The first one is to specify the value of A_s, the amplitude of the primordial power spectrum, which is passed directly to CLASS. This option is available in the case of the linear/nonlinear matter power spectrum implementation. For these, as well as for BBKS and E&H transfer functions, there is the additional option to set the normalization of the matter power spectrum by specifying σ_8, the RMS density contrast averaged over spheres of radius $8h^{-1}$Mpc. The computation of σ_8 is described in Section 4.11.

In practice, there is only one argument that encodes the normalization. This is the argument norm_pk, which can be passed the power spectrum normalization parameterized by σ_8 or $A_{\rm s}$. As noted above, ccl_parameters_create switches to σ_8 normalization if norm_pk $> 10^{-5}$, and to $A_{\rm s}$ normalization otherwise.

In the python implementation, CCL allows for either $\sigma 8$ or A_s to be passed as parameters.

4.9. **Angular power spectra**

In this section we will distinguish between *observables* (quantities observed on the sky, such as number counts in a redshift bin, shear or CMB temperature fluctuations) and *contributions* to the total observed fluctuations of these observables (such as the biased matter density term in number counts, redshift-space distortions, magnification, ISW, etc.). The routines described in this subsection are implemented in `ccl_cls.cc`.

4.9.1. **Exact expressions**

The angular power spectrum between two observables a and b can be written as:

$$C_\ell^{ab} = 4\pi \int_0^\infty \frac{dk}{k}\, \mathcal{P}_\Phi(k) \Delta_\ell^a(k) \Delta_\ell^b(k), \tag{15}$$

where $\mathcal{P}_\Phi(k)$ is the dimensionless power spectrum of the primordial curvature perturbations, and Δ^a and Δ^b are, using the terminology of CLASS, the transfer functions corresponding to these observables. Each transfer function will receive contributions from different terms. Currently `CCL` supports two observables (also labelled "tracers"), number counts and galaxy shape distortions, with the following contributions:

Number counts. —The transfer function for number counts can be decomposed into three terms: $\Delta^{\rm NC} = \Delta^{\rm D} + \Delta^{\rm RSD} + \Delta^{\rm M}$, where

- $\Delta^{\rm D}$ is the standard density term proportional to the matter density:

$$\Delta_\ell^{\rm D}(k) = \int dz\, p_z(z)\, b(z)\, T_\delta(k,z)\, j_\ell(k\chi(z)), \tag{16}$$

  where T_δ is the matter transfer function. Note that `CCL` currently does not support non-linear or scale-dependent bias. Here, $p_z(z)$ is the normalized distribution of sources in redshift (selection function). Thus `CCL` understands each individual redshift bin as a separate "observable".

- $\Delta^{\rm RSD}$ is the linear contribution from redshift-space distortions:

$$\Delta_\ell^{\rm RSD}(k) = \int dz\, p_z(z) \frac{(1+z)p_z(z)}{H(z)} T_\theta(k,z)\, j_\ell''(k\chi(z)), \tag{17}$$

  where $T_\theta(k,z)$ is the transfer function of θ, the divergence of the comoving velocity field. $T_\theta(k,z)$ depends on the growth, which `CCL` does not compute for massive neutrino cosmologies; therefore at this time an attempt to create a number count tracer in a cosmology with massive neutrinos will cause `CCL`

to raise an error. C_ℓ is instead computed assuming a linear-theory relation between the matter overdensity and peculiar velocity fields. While this should not be problematic for wide photometric redshift bins, users should exercise care when interpreting results for narrow window functions.

- $\Delta^{\rm M}$ is the contribution from magnification lensing:

$$\Delta^{\rm M}_\ell(k) = -\ell(\ell+1)\int \frac{dz}{H(z)} W^{\rm M}(z) T_{\phi+\psi}(k,z) j_\ell(k\chi(z)), \tag{18}$$

where $T_{\phi+\psi}$ is the transfer function for the Newtonian-gauge scalar metric perturbations, and $W^{\rm M}$ is the magnification window function:

$$W^{\rm M}(z) \equiv \int_z^\infty dz' p_z(z') \frac{2-5s(z')}{2} \frac{r(\chi(z')-\chi(z))}{r(\chi(z'))}. \tag{19}$$

Here $s(z)$ is the magnification bias, given as the logarithmic derivative of the number of sources with magnitude limit, and $r(\chi)$ is the angular comoving distance (see Eq. 5).

Note that CCL currently does not compute relativistic corrections to number counts Challinor & Lewis (2011); Bonvin & Durrer (2011). Although these should be included in the future, their contribution to the total fluctuation is largely subdominant (see Alonso et al. (2015) and the two references above), and therefore it is safe to work without them in most cases.

Galaxy shape distortions. —The transfer function for shape distortions is currently decomposed into two terms: $\Delta^{\rm SH} = \Delta^{\rm WL} + \Delta^{\rm IA}$, where

- $\Delta^{\rm L}$ is the standard lensing contribution:

$$\Delta^{\rm L}_\ell(k) = -\frac{1}{2}\sqrt{\frac{(\ell+2)!}{(\ell-2)!}}\int \frac{dz}{H(z)} W^{\rm L}(z) T_{\phi+\psi}(k,z) j_\ell(k\chi(z)), \tag{20}$$

where $W^{\rm L}$ is the lensing kernel, given by

$$W^{L}(z) \equiv \int_z^\infty dz' p_z(z') \frac{r(\chi(z')-\chi(z))}{r(\chi(z'))}. \tag{21}$$

- $\Delta^{\rm IA}$ is the transfer function for intrinsic galaxy alignments. CCL currently supports the so-called "non-linear alignment model", according to which the galaxy inertia tensor is proportional the local tidal tensor Hirata & Seljak (2004); Hirata et al. (2007).

$$\Delta^{\rm IA}_\ell(k) = \sqrt{\frac{(\ell+2)!}{(\ell-2)!}}\int dz\, p_z(z)\, b_{\rm IA}(z)\, f_{\rm red}(z)\, T_\delta(k,z)\, \frac{j_\ell(k\chi(z))}{(k\chi(z))^2}. \tag{22}$$

It is worth noting that the equations above should be modified for non-flat cosmologies by replacing the spherical Bessel functions j_ℓ with their hyperspherical counterparts Kamionkowski & Spergel (1994). Since the library currently only uses the Limber approximation (documented below), this is not currently an issue. It will be revisited in future versions of `CCL`.

CMB lensing. —The transfer function lensing convergence from a source at redshift $z_{\rm S}$ is given by:

$$\Delta^{\rm C}_\ell(k) = -\frac{\ell(\ell+1)}{2}\int_0^{\chi_{\rm S}} d\chi\, \frac{r(\chi_{\rm S})-r(\chi)}{r(\chi)r(\chi_{\rm S})} T_{\phi+\psi}(k,\chi)\, j_\ell(k\chi), \tag{23}$$

where $\chi_{\rm S} \equiv \chi(z_{\rm S})$.

4.9.2. **The Limber approximation**

As shown above, computing each transfer function involves a radial projection (i.e. an integral over redshift or χ), and thus computing full power spectrum consists of a triple integral for each ℓ. This can be computationally intensive, but can be significantly simplified in certain regimes by using the Limber approximation, given by:

$$j_\ell(x) \simeq \sqrt{\frac{\pi}{2\ell+1}}\,\delta\left(\ell+\frac{1}{2}-x\right). \tag{24}$$

Thus for each k and ℓ we can define a radial distance $\chi_\ell \equiv (\ell+1/2)/k$, with corresponding redshift z_ℓ. This approximation works best for wide radial kernels and high multipoles.

Substituting this in the expressions above, the simplified versions become:

$$C^{ab}_\ell = \frac{2}{2\ell+1}\int_0^\infty dk\, P_\delta\,(k,z_\ell)\,\tilde{\Delta}^a_\ell(k)\tilde{\Delta}^b_\ell(k). \tag{25}$$

where

$$\tilde{\Delta}^{\rm D}_\ell(k) = p_z(z_\ell)\, b(z_\ell)\, H(z_\ell) \tag{26}$$

$$\tilde{\Delta}^{\rm RSD}_\ell(k) = \frac{1+8\ell}{(2\ell+1)^2}\, p_z(z_\ell)\, f(z_\ell)\, H(z_\ell) - \tag{27}$$

$$\frac{4}{2\ell+3}\sqrt{\frac{2\ell+1}{2\ell+3}} p_z(z_{\ell+1})\, f(z_{\ell+1})\, H(z_{\ell+1}) \tag{28}$$

$$\tilde{\Delta}^{\rm M}_\ell(k) = 3\Omega_{M,0}H_0^2 \frac{\ell(\ell+1)}{k^2}\, \frac{(1+z_\ell)}{r(\chi_\ell)} W^{\rm M}(z_\ell) \tag{29}$$

$$\tilde{\Delta}^{\rm L}_\ell(k) = \frac{3}{2}\Omega_{M,0}H_0^2 \sqrt{\frac{(\ell+2)!}{(\ell-2)}} \frac{1}{k^2}\, \frac{1+z_\ell}{r(\chi_\ell)} W^{\rm L}(z_\ell) \tag{30}$$

$$\tilde{\Delta}^{\rm IA}_\ell(k) = \sqrt{\frac{(\ell+2)!}{(\ell-2)!}} \frac{p_z(z_\ell)\, b_{\rm IA}(z_\ell) f_{\rm red}(z_\ell) H(z_\ell)}{(\ell+1/2)^2} \tag{31}$$

$$\tilde{\Delta}^{\rm C}_\ell(k) = \frac{3}{2}\Omega_{M,0}H_0^2\, \ell(\ell+1) \frac{1+z_\ell}{k^2}\, \frac{r(\chi_{\rm S}) - r(\chi_\ell)}{r(\chi_\ell) r(\chi_{\rm S})}\, \Theta(\chi_\ell; 0, \chi_{\rm S}). \tag{32}$$

Here ($\Theta(x; x_i, x_f)$) is a top-hat function (1 for $x \in [x_i, x_f]$ and 0 otherwise).

4.9.3. **Beyond limber**

Native CCL computation.

`CCL` incorporates routines to compute the C_ℓ^{ab} angular power spectra as described above but without the Limber approximation. The algorithm performs first the integrals over z for both tracers, and ends with the k integral. This computation is much slower than using the Limber approximation, but it ends up with precise angular power spectra at low ℓ, and correct cross-correlations between tracers (the Limber approximation fails at reproducing the interference terms $j_\ell(x) \times j_\ell(x')$).

Some parameters are needed to define this integral. First, to fasten the computation the C_ℓ^{ab} function is computed for a selection of ℓ values (linearly spaced at low-ℓ and logarithmicly spaced at high-ℓ) and then the function is splined. Then the integration step of the redshift integrals must be specified in terms of a comoving distance. It is recommanded to start with a low value ($< 3\,$Mpc) for a high precision integration, and then to release this parameter to achieve the user needs in terms of precision and rapidity. A logarithmic step in k must also be provided: it does not correspond to the integration step but to a computation step before splining the transfer functions $\Delta_\ell^a(k)$. Last parameter, a minimum redshift can be given for the redshift integral bound so that the transfer functions are not computed near $z \approx 0$ where they can be numerically undefined.

The integration bounds in redshift are estimated automatically given the a redshift window so that the integration is preformed where the redshift window is relevant within a given precision (set by the `CCL_FRAC_RELEVANT` parameter). The integration bounds for the k integrals are defined automatically given the ℓ multipole and the comoving distances at play.

It is worth noting that the user can define a threshold multipole ℓ from which the C_ℓ^{ab} computation switches to the Limber approximation which is faster and generally relevant at high ℓ values.

Angpow.

The aim of the `Angpow` software (Campagne et al. 2017) is to compute the angular power spectra C_ℓ^{ab} without any Limber numerical approximation. `CCL` has been linked to the `Angpow` code, which is briefly described here.

The angular power spectrum for two shells C_ℓ^{ab} is computed in `Angpow` according to the following expression

$$C_\ell^{ab} = \iint_0^\infty \mathrm{d}z\mathrm{d}z' p_{z_1}(z_1) p_{z_2}(z') \times \int_0^\infty \mathrm{d}k\, f_\ell(z,k) f_\ell(z',k). \tag{33}$$

The auxiliary function $f_\ell(z,k)$ can be defined without loss of generality as

$$f_\ell(z,k) \equiv \sqrt{\frac{2}{\pi}}\, k \sqrt{P(k,z)}\, \widetilde{\Delta}_\ell(z,k) \tag{34}$$

with

- $P(k,z)$: the matter power spectrum at redshift z
- $\widetilde{\Delta}_\ell(z,k)$: a function describing the physical processes such as matter density fluctuations, redshift-space distortions as described for instance in references Durrer (2008); Yoo et al. (2009); Yoo (2010); Challinor & Lewis (2011); Bonvin & Durrer (2011). Currently, the `Angpow` version delivered with `CCL` only can deal with galaxy clustering tracers (no lensing) and this without the magnification lensing term (equation 18). The incorporation of those transfer functions is left for future work, but in principle `Angpow` has already the capability to treat them. For now, for galaxy clustering tracers we defined $\widetilde{\Delta}_\ell(z,k)$ as
$$\widetilde{\Delta}_\ell(z,k) \approx b j_\ell(k\chi(z)) - f(z) j_\ell''(k\chi(z)) \tag{35}$$
with $j_\ell(x)$ and $j_\ell''(x)$ the spherical Bessel function of order ℓ and its second derivative, and $f(a)$ the growth rate as defined subsection 4.7 (derivative of the growth function with respect to the scale factor a).

To proceed to a numerical evaluation of equation 33, `Angpow` first conducts inside the rectangle $[z_{1\min}, z_{1\max}] \times [z_{2\min}, z_{2\max}]$ given by the $p_z(z)$ selection functions a Cartesian product of one-dimensional (1D) quadrature defined by the set of sample nodes z_i and weights w_i. In practice, the Clenshaw-Curtis quadrature is used. The corresponding sampling points (z_{1i}, z_{2j}) are weighted by the product $w_i w_j$ using the 1D quadrature sample points and weights on both redshift regions with $i = 0, \ldots, N_{z_1} - 1$ and $j = 0, \ldots, N_{z_2} - 1$. Then, one gets the following approximation:

$$C_\ell^{ab} \approx \sum_{i=0}^{N_{z_1}-1} \sum_{j=0}^{N_{z_2}-1} w_i w_j p_{z_1}(z_i) p_{z_2}(z_j) \widehat{P}_\ell(\chi_i, \chi_j) \tag{36}$$

with the notations $z_i = z_{1i}$, $z_j = z_{2j}$ and $\chi_i = \chi(z_{1i})$, $\chi_j = \chi(z_{2j})$ and

$$\widehat{P}_\ell(z_i, z_j) = \int_0^\infty dk\, f_\ell(z_i, k) f_\ell(z_j, k), \tag{37}$$

To conduct the computation of such integral of highly oscillating functions we use the 3C-algorithm described in details in reference (Campagne et al. 2017). In brief this algorithm proceeds the following way:

1. the total integration k interval (eg. $[k_{\min}, k_{\max}]$) in equation (37) is cut on several k-sub-intervals;
2. on each sub-interval the functions $f_{i\,\ell}(k) = f_\ell(z_i, k)$ and $f_{j\,\ell}(k) = f_\ell(z_j, k)$ are projected onto Chebyshev series of order 2^N;
3. the product of the two Chebyshev series is performed with a 2^{2N} Chebyshev series;
4. then, the integral on the sub-interval is computed thanks to the Clenshaw-Curtis quadrature.

All the Chebyshev expansions and the Clenshaw-Curtis quadrature are performed via the DCT-I fast transform of FFTW.

Thanks to the 3C-algorithm, the `Angpow` is able to compute the C_ℓ^{ab} in a fast and accurate way. It was tested against `CLASS` and the native `CCL` computation and can performed the computation an order of magnitude faster, which is more suitable for an extensive exploration of the cosmological parameter space ($\mathcal{O}(1s)$). Note that `Angpow` is written in C++ with OpenMP, to distribute the computation of a single C_ℓ on a single thread. The computation can also be switch by the user to the faster Limber approximation setting a threshold multipole ℓ.

Precision tests.

The code has been compared with CLASS and the native CCL computation and all three softwares agrees perfectly if precision parameters are pushed to high levels.

A few parameters must be provided to set the precision of this computation. First the order of the Chebyshev polynomials is set to 2^{10} by default, and the number of k sub-intervals to 200, and we checked this is enough for the current uses. Then the redshift quadrature stepping is set automatically given the redshift windows to recover the native CCL computation boosted with high precision parameters: its precision is optimised so that the relative numerical error compared with the native method is two orders of magnitude below the relative cosmic variance $\sqrt{2/(2\ell+1)}$, from $\ell = 2$ to $\ell = 1000$. The $k_{\rm min}$ and $k_{\rm max}$ bounds are also automatically set given the current multipole ℓ and the comoving distance χ involved in the inner integral.

4.10. **Correlation functions**

The following expressions relating the angular power spectra and correlation functions are valid in the flat-sky approximation[16]. In all cases, $f_K(\chi)$ is comoving angular diameter distance, which differs from the radial comoving distance χ only in the case of cosmologies with non-zero curvature. The routines described in this subsection are implemented in ccl_correlation.c.

Galaxy-galaxy. The angular correlation function between two number-count tracers (labeled a and b here) is given by

$$\xi^{ab}(\theta) = \int d\ell \frac{\ell}{2\pi} C_\ell^{ab} J_0(\ell\theta), \tag{38}$$

where C_{ab} is the angular power spectrum between both tracers.

Lensing-lensing. The lensing correlation functions are [17]

$$\xi_+^{ab}(\theta) = \int_0^\infty d\ell \frac{\ell}{2\pi} J_0(\ell\theta) C_\ell^{ab}, \tag{39}$$

$$\xi_-^{ab}(\theta) = \int_0^\infty d\ell \frac{\ell}{2\pi} J_4(\ell\theta) C_\ell^{ab}, \tag{40}$$

where the angular lensing convergence power spectrum C_ℓ^{ab} is given above (see Equations 20 and 25).

[16] See the weak lensing review by Bartelmann & Schneider (2001), page 44 and Joachimi & Bridle (2010).

[17] from Schneider 2002 and Bartelmann & Schneider section 6.4.1

Galaxy-lensing. The correlation between a number count tracer a and a shear tracer b is given by

$$\xi^{ab}(\theta) = \int d\ell \frac{\ell}{2\pi} C_\ell^{ab} J_2(\ell\theta), \tag{41}$$

Note that, in the above, "Galaxy" and "Lensing" can be replaced by any spin-0 and spin-2 fields on the sphere respectively (e.g. the CMB lensing convergence would play the same role as the galaxy overdensity field in all the formulas above).

3d spatial correlation function. In addition to the angular correlation functions, the 3-dimensional spatial correlation function $\xi(r)$ is also calculated from the power spectrum $P(k)$ using

$$\xi(r) = \frac{1}{2\pi^2} \int dk\, k^2 P(k) \frac{\sin(kr)}{kr} \tag{42}$$

To evaluate the numerical integral in the correlation functions, we make use of the public code `FFTlog`[18](Hamilton 2000; Talman 2009). In brief, `FFTlog` works on functions periodic in log space, by writing the Hankel Transform as a convolution between Bessel functions and the function of interest (in this case either C_ℓ or $P(k)$). A version of this code is included in `CCL` with minor modifications.

4.11. Halo mass & halo bias functions

The routines described in this subsection are implemented in `ccl_massfunc.c`.

The halo mass function is implemented using several different definitions from the literature: Tinker et al. (2008), Tinker et al. (2010), Angulo et al. (2012), and Watson et al. (2013). All four models are tuned to simulation data and tested against observational results. In addition, each of these fits has been implemented using the common halo definition of $\Delta = 200$, where a halo is defined with:

$$\bar{\rho}(r_\Delta) = \Delta \times \rho_{\rm m}, \tag{43}$$

where a halo with size r_Δ has an average density $\bar{\rho}$ equal to the overdensity parameter Δ times the mean background density of the universe, $\rho_{\rm m}$. Note that another common definition utilizes the critical density of the universe, $\rho_{\rm c}$; currently `CCL` requires that an external conversion by the end user between values of Δ with respect to the critical density to values of Δ as defined with respect to the mean

[18] http://casa.colorado.edu/~ajsh/FFTLog/

density. In the future we plan to allow for self-consistent handling of critical density based definitions, though it is not implemented as of this build.

In addition to the usage of the most common definition, we have implemented an extension for two of the models. The Tinker 2010 model allows for a value of Δ to be given between the values of 200 and 3200 and interpolates the fitting parameters within this range in a space of $\log \Delta$ using splines. We also have implemented interpolation in the same range of Tinker 2008 Δ values. For both Tinker 2008 and Tinker 2010 models we have utilized spline interpolation through `GSL` routines in order to guarantee a match to specified fitting parameters at exact values of Δ. This fitting has slight deviation from the fit as expressed in Tinker 2010.

The halo mass function models implemented in CCL are tuned to simulations without massive neutrinos, therefore are not valid in cosmologies with massive neutrinos. Attempts to calculate the halo mass function, halo bias, or other related quantities within cosmologies with massive neutrinos will cause `CCL` to raise an error and quit.

With the exception of the Tinker 2010 model, we attempt to keep a common form to the multiplicity function whenever possible for ease of extension:

$$f(\sigma) = A\left[\left(\frac{\sigma}{b}\right)^{-a} + 1\right] e^{-c/\sigma^2}, \tag{44}$$

where A, a, b, and c are fitting parameters that have additional redshift scaling and σ is the RMS variance of the density field smoothed on some scale M at some scale factor a. This basic form is modified for the Angulo et al. (2012) formulation. The resulting form is

$$f(\sigma) = A\left[\left(\frac{b}{\sigma} + 1\right)^{-a}\right] e^{-c/\sigma^2}, \tag{45}$$

where the only change is in the formulation of the second term. Note that the fitting parameters in the Angulo et al. (2012) formulation do not contain any redshift dependence and the use of it is primarily for testing and benchmark purposes.

Each call to the halo mass function requires an assumed model (defined within the `ccl_configuration` structure contained in `ccl_cosmology`), in addition to a value of the halo mass and scale factor for which to evaluate the halo mass function. The currently implemented models can be called with the tags `config.mass_function_method = ccl_tinker`, `ccl_tinker10`, `ccl_angulo`, or `ccl_watson`. It returns the number density of halos in logarithmic mass bins, in the form $dn/d\log_{10} M$, where n is the number density of halos of a given mass and M is the input halo mass.

The halo mass M is related to σ by first computing the radius R that would enclose a mass M in a homogeneous Universe at $z = 0$:

$$M = \frac{H_0^2}{2G} R^3 \rightarrow \frac{M}{M_\odot} = 1.162 \times 10^{12} \Omega_M h^2 \left(\frac{R}{1\,\mathrm{Mpc}} \right)^3 . \tag{46}$$

The rms density contrast in spheres of radius R can then be computed as

$$\sigma_R^2 = \frac{1}{2\pi^2} \int dk\, k^2\, P_k\, \tilde{W}_R^2(k) \tag{47}$$

where P_k is the matter power spectrum and $\tilde{W}(kR)$ is the Fourier transform of a spherical top hat window function,

$$\tilde{W}_R(k) = \frac{3}{(kR)^3} [\sin(kR) - kR\cos(kR)] \tag{48}$$

This function is directly implemented in `CCL` as well as a specific σ_8 function.

The Tinker et al. (2010) model parameterizes both the halo mass function and the halo bias in terms of the peak height, $\nu = \delta_c / \sigma(M)$, where δ_c is the critical density for collapse and is chosen to be 1.686 for this particular parameterization. We can then parameterize the halo function and halo bias as

$$b(\nu) = 1 - A \frac{\nu^a}{\nu^a + {\delta_c}^a} + B\nu^b + C\nu^c, f(\nu) = \alpha[1 + (\beta\nu)^{-2\phi}]\nu^{2\eta} e(-\gamma\nu^2/2). \tag{49}$$

The currently implemented model in `CCL` allows for an arbitrary overdensity Δ to be chosen, using the fitting functions provided in Tinker et al. (2010). Other halo model definitions are not included in the halo bias calculation, though this remains an area of active work to improve upon.

The halo mass function and related quantities

4.12. Photo-z implementation

The functionality described in this section is implemented in `ccl_lsst_specs.c`.

LSST galaxy redshifts will be obtained using photometry. A model is therefore required for the probability of measuring a photometric redshift z_{ph} for an object with hypothetical spectroscopic redshift z_{s}. `CCL` allows the user to flexibly provide their own photometric redshift model.

To take advantage of this functionality, the user writes a function which accepts as input a photometric redshift, a spectroscopic redshift, and a void pointer to a

structure containing any further parameters of the photo-z model. This function will return the probability of measuring the input photometric redshift given the input spectroscopic redshift. Explicitly, it should take the form:

```
user_pz_probability(double z_ph, double z_s, void * user_par){...}
```

This function is responsible for extracting the photo-z model parameters from the void pointer `user_par` itself.

This model can be used when computing $\frac{dN}{dz}^i$ in photometric redshift bin i, as given by equation 52 below. An example of how the user can construct the required functions and structure can be found in `ccl_sample_run.c`. An example that uses a built-in Gaussian photo-z model is also provided.

4.13. LSST Specifications

`CCL` includes LSST specifications for the expected galaxy distributions of the full galaxy clustering sample and the lensing source galaxy sample. These enable the user to easily make forecasts for LSST. The functionality described in this section is implemented in `ccl_lsst_specs.c`.

The functional forms of the expected $\frac{dN}{dz}$ for clustering galaxies and lensing source galaxies are provided. Here, $\frac{dN}{dz}$ is the number density of galaxies as a function of true redshift.

In the case of lensing source galaxies, these forms are given in Chang et al. (2013), wherein three different cases are considered: fiducial, optimistic, and conservative. All three are included in `CCL`, and are indicated via a label of `DNDZ_WL_OPT`, `DNDZ_WL_FID`, and `DNDZ_WL_CONS` as appropriate. The functional form of $\frac{dN}{dz}$ for lensing source galaxies is given as:

$$\frac{dN}{dz} \propto z^{\alpha} \exp\left(-\frac{z}{z_0}^{\beta}\right). \tag{50}$$

The parameters, in the fiducial case, are given as $\alpha = 1.24$, $\beta = 1.01$, and $z_0 = 0.51$. In the optimistic case, this becomes $\alpha = 1.23$, $\beta = 1.05$, and $z_0 = 0.59$. The conservative case is given by $\alpha = 1.28$, $\beta = 0.97$, and $z_0 = 0.41$.

For the case of the clustering galaxy sample, the functional form is given by (LSST Science Collaboration 2009):

$$\frac{dN}{dz} \propto \frac{1}{2z_0}\left(\frac{z}{z_0}\right)^2 \exp\left(-\frac{z}{z_0}\right) \tag{51}$$

with $z_0 = 0.3$.

In order to be incorporated into forecasts or predictions, the above expressions for $\frac{dN}{dz}$ must be normalized, and the value of $\frac{dN}{dz}$ must be provided in a given photometric redshift bin. Support is provided for the user to input a flexible photometric redshift model, as described in Section 4.12. This takes the form of a function which returns the probability $p(z, z')$ of measuring a particular photometric redshift z, given a spectroscopic redshift z' and other relevant parameters. Also provided are functions to return σ_z at a given redshift for both lensing sources and clustering galaxies, for the case in which the user-defined function is a Gaussian photo-z model.

With this, $\frac{dN^i}{dz}$ of lensing or clustering galaxies in a particular photometric redshift bin i is given by:

$$\frac{dN^i}{dz} = \frac{\frac{dN}{dz}\int_{z_i}^{z_{i+1}} dz' p(z, z')}{\int_{z_{\rm min}}^{z_{\rm max}} dz \frac{dN}{dz} \int_{z_i}^{z_{i+1}} dz' p(z, z')} \tag{52}$$

where z_i and z_{i+1} are the photo-z edges of the bin in question.

Finally, the expected (linear, scale-independent) bias of galaxies in the clustering sample is also provided. It is given by (LSST Science Collaboration 2009):

$$b(z) = \frac{0.95}{D(z)} \tag{53}$$

where $D(z)$ in the linear growth rate of structure normalized to unity at z=0. Note that this bias definition requires the calculation of the linear growth rate and therefore is not yet supported for cosmologies with massive neutrinos.

5. Tests and validation

Our goal is for outputs of `CCL` to be validated against the results of a code-comparison project run within LSST-DESC down to a 10^{-4} or pre-established accuracy level if possible. In some cases, this level of accuracy is not necessary, as other systematics which have not been considered in this version of `CCL` yet are expected to have a larger fractional impact. In the cases where this applies, we make it clear below.

A code comparison project was carried out among members of TJP where the following outputs of cosmological forecast codes were compared and validated:

1. growth factor at $z = 0, 1, 2, 3, 4, 5$,

Model	Ω_m	Ω_b	Ω_Λ	h_0	σ_8	n_s	w_0	w_a
flat LCDM	0.3	0.05	0.7	0.7	0.8	0.96	-1	0
w_0 LCDM	0.3	0.05	0.7	0.7	0.8	0.96	-0.9	0
w_a LCDM	0.3	0.05	0.7	0.7	0.8	0.96	-0.9	0.1
open w_a LCDM	0.3	0.05	0.65	0.7	0.8	0.96	-0.9	0.1
closed w_a LCDM	0.3	0.05	0.75	0.7	0.8	0.96	-0.9	0.1

Table 3. Cosmological models for the code comparison project.

2. comoving radial distance $[\mathrm{Mpc}/h]$ at the same redshifts, as well as the corresponding distance moduli,
3. linear matter power spectrum, $P(k)$, from BBKS (Bardeen et al. 1986) in units of $(\mathrm{Mpc}/h)^3$ at $z = 0, 2$ in the range $10^{-3} \leq k \leq 10h/\mathrm{Mpc}$ with 10 bins per decade,
4. Eisenstein & Hu matter power spectrum in units of $(\mathrm{Mpc}/h)^3$ at $z = 0$ in the range $10^{-3} \leq k \leq 10h/\mathrm{Mpc}$ with 10 bins per decade, and
5. the mass variance at $z = 0$, $\sigma(M, z = 0)$ for $M = \{10^6, 10^8, 10^{10}, 10^{12}, 10^{14}, 10^{16}\}\mathrm{M}_\odot/h$.

These predictions were produced and compared for different cosmologies, which are listed in the table below. The results agree to better than 0.1% relative accuracy for comoving distance and growth factor among all submissions, and for $P(k)$ and $\sigma(M)$ among codes which use the same BBKS conventions.

We noticed that there are 2 typos for the BBKS transfer function in "Modern Cosmology" (Dodelson 2004) compared to the original BBKS paper. The quadratic term should be $(16.1q)^2$ and the cubic term should be $(5.46q)^3$. The BBKS equation is correct in Peacock (1999). Using the wrong equation can give differences in the results above the 10^{-4} level.

From the comparison, we were also able to identify some typical issues which affect convergence at the desired level:

- For achieving 10^{-4} precision in $\sigma(M)$ and the normalisation of the power spectrum, one should check that the integral of σ_8 and $\sigma(M)$ has converged for the chosen values of $\{k_{\mathrm{min}}, k_{\mathrm{max}}\}$. After checking convergence, we achieved the desired precision.
- Also note that for $\sigma(M)$, it is important to set the desired precision level correctly for the numerical integrator. The integral usually yields $\sigma^2(M)$, and not

$\sigma(M)$. Hence, one has to set the desired precision taking the exponent into account.

- The value of the gravitational constant, G, enters into the critical density. We found that failure to define G with sufficient precision would result in lack of convergence at the 10^{-4} level between the different submissions. Importantly, note that CAMB barely has 10^{-4} precision in G (and similarly, there might be other constants within CAMB/CLASS for which one should check the precision level). For `CCL`, we are using the value from the Particle Physics Handbook.

- Including/excluding radiation in the computation of the comoving distances and the growth function can easily make a difference of 10^{-4} at the redshifts required in this comparison.

In a second stage, we used the BBKS linear matter power spectrum from the previous step to compare two-point statistics for two redshift bins, resulting in three tomography combinations, $(1-1)$,$(1-2)$,$(2-2)$. We computed the following quantities:

- projected galaxy clustering tomography power spectra: density term only (no magnification, RSD, etc.) with non-evolving linear bias $b(z) = 1$, in the range $10 < \ell < 10000$, using 5 bins per decade,

- angular convergence tomography power spectra: leading order convergence term only (no magnification), in the same range and with the same resolution as the case above,

- angular galaxy clustering tomography correlation function, in the range $0.01\,\mathrm{deg} < \theta < 5\,\mathrm{deg}$, using 5 bins per decade, and

- angular shear tomography correlation functions (ξ_+,ξ_-), similarly to above.

We adopted the following analytic redshift distributions: a Gaussian with $\sigma = 0.15$, centered at $z_1 = 1$; and another Gaussian with the same dispersion but centered at $z_2 = 1.5$. We repeated the exercise for two redshift distribution histograms shown in Figure 2.

In this second step, only 2 codes have been compared so far. More outputs are needed to guarantee convergence. Preliminarily, from these outputs, we have concluded that:

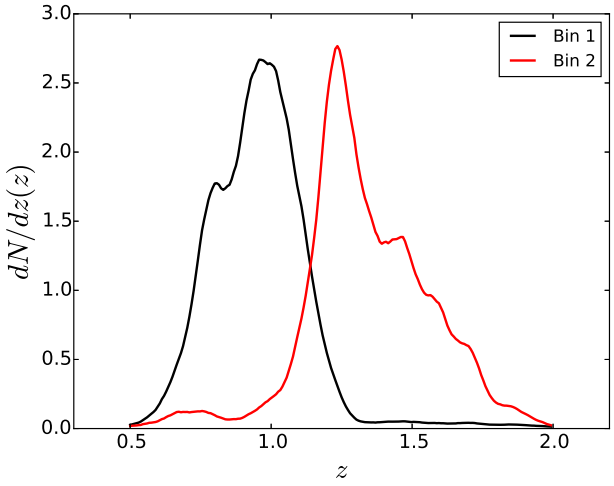

Figure 2. Binned redshift distributions used for code comparison project.

- The cross-correlation between bins is particularly sensitive to the number of points where the kernels have been sampled.
- The accuracy of the correlation function is sensitive to $\ell_{\rm max}$. We had to use up to $\ell_{\rm max} = 3 \times 10^4$ for convergence (and we could not achieve 0.01% convergence).
- The large scales of the correlation function are sensitive to $\ell_{\rm min}$. The use of the flat-sky approximation is also relevant on these scales.
- For sufficiently high precision, the correlation functions are sensitive to how the power spectrum is sampled and interpolated.

For C_ℓ computations, we required the relative difference between `CCL` and the benchmarks to be $< 10^{-3}$. We performed the test both for analytic redshift distributions and histograms.

To obtain realistic targets for the convergence of correlation function computations for LSST analyses, we calculate the expected statistical uncertainty of the clustering and lensing correlation functions of the LSST gold sample (c.f. Sect. 4.13), assuming an effective source galaxy density of $n_{\rm eff} = 26 {\rm gal/sq\,arcmin}$ for galaxy shape distortions, and galaxy density of $n_{\rm gold} = 45 {\rm gal/sq\,arcmin}$ for number counts. Specifically, we calculate the Gaussian covariance of angular correlation functions following the formalism of Joachimi et al. (2008), and note that leaving out the non-Gaussian covariance terms makes this convergence criterion more conservative. We split the galaxy samples into 10 tomography bins, defined to contain equal numbers of galaxies. The accuracy test then proceeds as follows. We compared the difference between `CCL` calculated lensing and clustering correlations

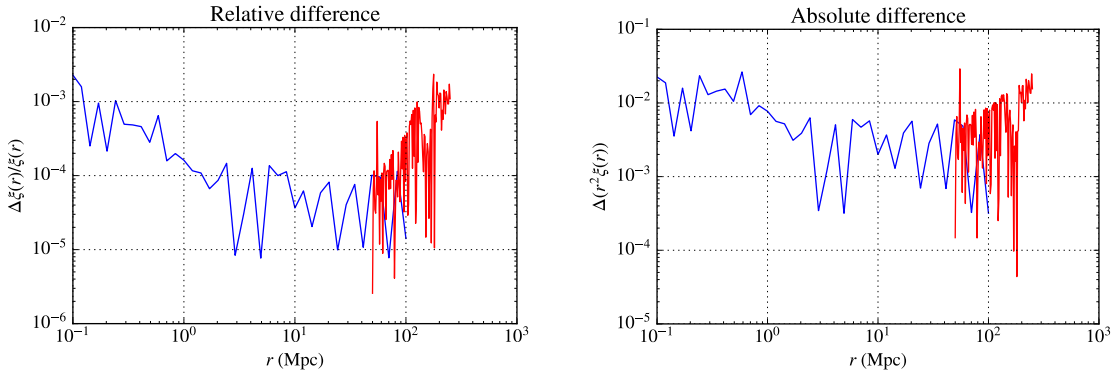

Figure 3. Comparison of the CCL calculation of the 3d spatial correlation function $\xi(r)$ with a precise, numerical transform of the CCL non-linear Halofit power spectrum. The left panel shows the relative error $\Delta\xi(r)/\xi(r)$. The right panel shows the absolute error in $r^2\xi(r)$. Both panels are for the w_a LCDM model of Table 3 at redshift zero.

and the benchmarks for the analytic redshift distributions and for auto-correlations of redshift bins only. To pass the benchmark test, we required that this difference be smaller than half of the value of the errorbar derived from the covariance for each correlation function computed. Specifically, we take the value of the covariance in the bins centered at $z = 1$ and $z = 1.5$ to compare to the benchmarks.

The 3-dimensional correlation function $\xi(r)$ was validated by comparing with an independent, precise numerical transform. We calculated $\xi(r)$ by transforming the CCL non-linear Halofit power spectrum using this independent code for the five cosmologies listed in Table 3 at redshifts $z = 0, 1, 2, 3, 4, 5$. We then compared with the $\xi(r)$ from CCL with a sampling of $P(k)$ equal to `N_K_3DCOR` bins per decade. The default value of `N_K_3DCOR` = 100,000 results in a relative agreement at the level of $\Delta\xi(r)/\xi(r) < 2.5 \times 10^{-3}$ for $0.1 < r < 250$ Mpc and redshift zero. The agreement is better for higher redshifts. We also compared the absolute value of $r^2\xi(r)$ and find a maximum difference of $\Delta(r^2\xi(r)) < 3.0 \times 10^{-2}$ for the range $r = 0.1 - 250$ Mpc. This corresponds to approximately 0.08% of the BAO peak value of $r^2\xi(r)$. At the BAO peak, the difference is only 9.0×10^{-3}, or 0.024% of the peak height. The results are shown in Fig. 3 To further validate the $P(k) \rightarrow \xi(r)$ transform we performed a test using an analytical function $\xi(r) = (r/r_0)^a$, whose inverse transform $P(k)$ has a known analytic form. We used $r_0 = 5$ Mpc^{-1} and $a = -1.67$, which gives a rough approximation to the actual 3d correlation function. We then compared the CCL calculation of $\xi(r)$ to the known analytic result. The relative difference $\Delta\xi(r)/\xi(r)$ was found to be less than 0.3% in the range $2 < r < 250$ Mpc rising to about 5% at $r = 1000$ Mpc (see Fig. 4). For $r = 0.1 - 0.8$ Mpc the relative difference is ≈8%. The accuracy at low and high distances can be improved by increasing the range over which the power spectrum splines are evaluated.

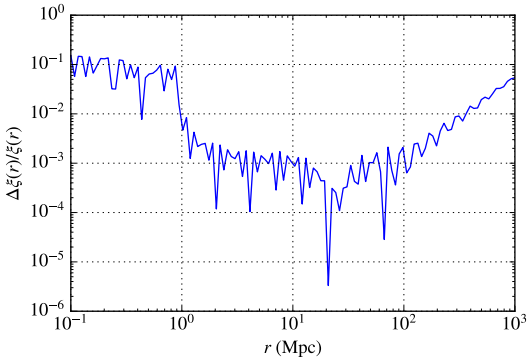

Figure 4. The relative error in the 3d spatial correlation function computed using the CCL algorithm compared to an analytic function $\xi(r) = (r/r_0)^a$ whose inverse transform $P(k)$ is known analytically. In this validation test, the known $P(k)$ was transformed with the CCL algorithm and compared to the known analytic result for $\xi(r)$.

Additionally, independent codes were utilized to test the accuracy of halo mass function predictions. For the halo mass function, we compare the value of σ, $\log(\sigma^{-1})$, and the value of the halo mass function in the form used in Tinker et al. (2008),

$$\log[(M^2/\bar{\rho}_m)dn/dM]. \tag{54}$$

We note that while we maintain the 10^{-4} for our evaluations of σ, the accuracy degrades to a value of 5×10^{-3} for the halo mass function evaluation, primarily at the high halo mass and high redshift domains. We find that this increased error is acceptable, as the level of precision is significantly better than the accuracy of current halo mass function models.

The implementation of the matter power spectrum emulation code from Lawrence et al. (2017) in `CCL` has been validated at first instance by comparing the direct output of the original code to `CCL` for the best fit cosmology (M000) in the range of wavenumbers spanned by the emulator predictions and confirming that the agreement is within the expectation from numerical errors (fractional difference of $< 10^{-5}$). We have also compared the `CCL` emulator outputs for certain cosmologies to smoothed power spectra from the simulations presented in Lawrence et al. (2017).[19] The check ensures that the `CCL` outputs when calling the emulator are within 3% of the simulation benchmarks at $z = 0$ and for the k range of validity of the emulator. This essentially corresponds to a verification of the results presented in Figure 5 and Figure 6 of Lawrence et al. (2017).

In addition to the distance tests based on the TJP code comparison project described above, we include tests based on `CLASS` that cover the benchmark cosmologies listed in Table 3, as well as cosmologies with massive neutrinos. The

[19] Courtesy of Earl Lawrence and Katrin Heitmann.

agreement between CLASS and CCL on comoving radial distances and distance moduli is better than 10^{-6}. These tests are done at 10 logarithmically spaced redshifts between 0.01 and 1000. We also include tests of the growth factor for the same redshift range.

Finally, for cosmologies involving massive neutrinos, we have included also tests which compare comoving distance and distance modulus predictions from CCL from those produced by the astropy (Astropy Collaboration et al. 2013) library. In this case, we consider these comparison tests to pass if the agreement between the two predictions is 0.001 or better; this is due to the fact that astropy uses a fitting function for the neutrino phase-space integral while we compute the integral directly. The fitting function used by astropy is expected to match the true integral at approximately this level. As a sanity check, we also include a test which verifies that the power spectrum from CLASS as called via CCL matches that as called directly from CLASS in three test cosmologies with massive neutrinos.

CCL has a suite of test routines which, upon compilation, compare its outputs to the benchmarks from code comparison. These are run with `make check`.

6. Examples for C implementation

Examples of how to run CCL are provided in the tests sub-directory of the library. The first resource for a new user should be the ccl_sample_run.c file. This starts by setting up the CCL default configuration. Then, it creates the "cosmo" structure, which contains distances and power spectra splines, for example. There are example calls for routines that output comoving radial distances, the scale factor, the growth factor and σ_8. Toy models are created for the redshift distributions of galaxies in the clustering and lensing samples, and for the bias of the clustering sample ($b(z) = 1 + z$). These are used for constructing the "tracer" structures via CCL_Cltracer, which can then be called to obtain the angular power spectra for clustering, cosmic shear and CMB lensing. All functions of redshift associated to each tracer can be retrieved at any arbitrary point using the same interpolation scheme used by CCL internally.

7. Python wrapper

A Python wrapper for CCL is provided through a module called pyccl. The whole CCL interface can be accessed through regular Python functions and classes, with all of the computation happening in the background through the C code. The func-

tions all support `numpy` arrays as inputs and outputs, with any loops being performed in the C code for speed.

7.1. **Python installation**

Before you can build the Python wrapper, you must have compiled and installed the C version of `CCL`, as `pyccl` will be dynamically linked to it. The Python wrapper's build tools currently assume that your C compiler is `gcc`, and that you have a working Python 2.x or 3.x installation with `numpy` and `distutils` with `swig` (the latter is not necessary for using `CCL`, only for development). If you have installed `CCL` in your default library path, you can build and install the `pyccl` module by going to the root `CCL` directory and choosing one of the following options:

- To build and install the wrapper for the current user only, run
  `$ python setup.py install --user`
- To build and install the wrapper for all users, run
  `$ sudo python setup.py install`
- To build the wrapper in-place in the source directory (for testing), run
  `$ python setup.py build_ext --inplace`

If you choose either of the first two options, the `pyccl` module will be installed into a sensible location in your `PYTHONPATH`, and so should be automatically picked up by your Python interpreter. You can then simply import the module using `import pyccl`. If you use the last option, however, you must either start your interpreter from the root `CCL` directory, or manually add the root `CCL` directory to your `PYTHONPATH`.

These options assume that the C library (`libccl`) has been installed somewhere in the default library path. If this isn't the case, you will need to tell the Python build tools where to find the library. This can be achieved by running the following command first, before any of the commands above:

```
python setup.py build_ext --library-dirs=/path/to/lib/
--rpath=/path/to/lib/
```

Here, `/path/to/lib/` should point to the directory where you installed the C library. For example, if you ran `./configure --prefix=/my/path/` before you compiled the C library, the correct path would be `/my/path/lib/`. The command above will

build the Python wrapper in-place; you can then run one of the `install` commands, as listed above, to actually install the wrapper. Note that the `rpath` switch makes sure that the `CCL` C library can be found at runtime, even if it is not in the default library path. If you use this option, there should therefore be no need to modify the library path yourself.

On some systems, building or installing the Python wrapper fails with a message similar to:

```
fatal error: 'gsl/gsl_interp2d.h' file not found.
```

This happens when the build tools fail to find the directory containing the GSL header files, e.g. when they have been installed in a non-standard directory. To work around this problem, use the `--include-dirs` option when running the `setup.py build_ext` step above, i.e. if the GSL header files are in the directory `/path/to/include/`, you would run

```
python setup.py build_ext --library-dirs=/path/to/install/lib/
--rpath=/path/to/install/lib/ --include-dirs=/path/to/include/
```

and then run one of the `setup.py install` commands listed above. (Note: As an alternative to the `--include-dirs` option, you can use `-I/path/to/include` instead.)

You can quickly check whether `pyccl` has been installed correctly by running `python -c "import pyccl"` and checking that no errors are returned. For a more in-depth test to make sure everything is working, change to the `tests/` subdirectory and run `python run_tests.py`. These tests will take a few minutes. Notice that these are not the same tests that are run via `make check`. In the case of the `python` tests, the library will only check for finite outputs of the routines called from `pyccl`. There is no benchmark comparison in this case.

7.2. **Python example**

The Python module has essentially the same functions as the C library, just presented in a more standard Python-like way. You can inspect the available functions and their arguments by using the built-in Python `help()` function, as with any Python module.

Below is a simple example Python script that creates a new `Cosmology` object, and then uses it to calculate the C_ℓ's for a simple lensing cross-correlation. It should take a few seconds on a typical laptop.

```
import pyccl as ccl
import numpy as np

# Create new Cosmology object with a given set of parameters. This keeps track
# of previously-computed cosmological functions
cosmo = ccl.Cosmology(Omega_c=0.27, Omega_b=0.045, h=0.67, A_s=2e-9, n_s=0.96)

# Define a simple binned galaxy number density curve as a function of redshift
z_n = np.linspace(0., 1., 200)
n = np.ones(z_n.shape)

# Create objects to represent tracers of the weak lensing signal with this
# number density (with has_intrinsic_alignment=False)
lens1 = ccl.ClTracerLensing(cosmo, False, n=(z_n, n))
lens2 = ccl.ClTracerLensing(cosmo, False, n=(z_n, n))

# Calculate the angular cross-spectrum of the two tracers as a function of ell
ell = np.arange(2, 10)
cls = ccl.angular_cl(cosmo, lens1, lens2, ell)
print cls
```

Further examples are collected in several Jupyter notebooks available in the `examples/` directory. These are:

`Correlation.ipynb`,

`Distance Calculations Example.ipynb`,

`HMFexample.ipynb`,

`Lensing angular power spectrum.ipynb`,

`MCMC Likelihood Analysis.ipynb`,

`Photo-z example.ipynb`,

`Correlation_3d.ipynb`,

`Power spectrum example.ipynb`

Note that the likelihood analysis in the last notebook is not intended to be realistic, but it gives an operational example of how `CCL` can be integrated into such an analysis. In particular, the notebook only considers cosmic shear over one wide redshift bin and the covariance matrix adopted solely includes a contribution from cosmic variance. The "data vector" is simply a simulated using `CCL` theoretical predictions. For speed, theoretical predictions use the BBKS power spectrum implementation.

7.3. **Technical notes on how the Python wrapper is implemented**

The Python wrapper is built using the `swig` tool, which automatically scans the `CCL` C headers and builds a matching interface in Python. The default autogenerated `swig` interface can be accessed through the `pyccl.lib` module if necessary. A more user-friendly wrapper has been written on top of this to provide more structure to the module, allow `numpy` vectorization, and provide more natural Python objects to use (instead of opaque `swig`-generated objects).

The key parts of the wrapper are as follows:

`setup.py` —This instructs `swig` and other build tools on how to find the right source files and set compile-time variables correctly. Most of this information is provided by header files and SWIG interface files that are included through the `pyccl/ccl.i` interface file.

Note that certain compiler flags, like `-fopenmp`, are also set in `setup.py`. If you are not using `gcc`, you may need to modify these flags (see the `extra_compile_args` argument of the `setup()` function).

Interface (`.i`) files —These are kept in the `pyccl/` directory, and tell `swig` which functions to extract from the C headers. There are also commands in these files to generate basic function argument documentation, and remove the `ccl_` prefix from function names.

The interface files also contain code that tells `swig` how to convert C array arguments to `numpy` arrays. For certain functions, this code may also contain a simple loop to effectively vectorize the function.

The main interface file is `pyccl/ccl.i`, which imports all of the other interface files. Most of the `CCL` source files (e.g. `core.c`) have their own interface file too. For other files, mostly containing support/utility functions, `swig` only needs the C header (`.h`) file to be specified in the main `ccl.i` file, however. (The C source file must also be added to the list in `setup.py` for it to be compiled successfully.)

Python module files—The structure of the Python module, as seen by the user, is organized through the `pyccl/__init__.py` file, which imports only the parts of the `swig` wrapper that are useful to the user. The complete autogenerated `swig` interface can be accessed through the `pyccl.lib` sub-module if necessary.

Individual sub-modules from `CCL` are wrapped in their own Python scripts (e.g. `power.py`), which typically provide a nicer "Pythonic" interface to the underlying `CCL` functions and objects. This includes automatically choosing whether to use the vectorized C function or not, as well as some conversions from Python objects to the autogenerated `swig` objects. Most of the core Python objects, like `Parameters` and `Cosmology`, are defined in `core.py`. These objects also do some basic memory management, like calling the corresponding `ccl_free_*` C function when the Python object is destroyed.

Auto-generated wrapper files—The `swig` command is triggered when you run `setup.py`, and automatically generates a number of C and Python wrapper files in the `pyccl/` directory. These typically have names like `ccl_*.c` and `ccl_*.py`, and should not be edited directly, as `swig` will overwrite them when it next runs.

`pyccl/pyutils.py`—This file contains several generic helper functions for passing `numpy` arrays in and out of Python functions in a convenient way, and for performing error checking and some type conversions.

The build process will also create a `pyccl/ccllib.py` file, which is the raw autogenerated Python interface, and `_ccllib.so`, which is a C library containing all of the C functions and their Python bindings. A `build/` directory and `pyccl.egg-info/` directory will also be created in the same directory as `setup.py` when you compile `pyccl`. These (plus the `pyccl/_ccllib.so` file) should be removed if you want to do a clean recompilation. Running `python setup.py clean --all` will remove some, but not all, of the generated files.

8. Future functionality to be included

In the future, we hope that CCL will include other functionalities. Functionalities which are currently under development:

- a link to angpow (Campagne et al. 2017) for going beyond the Limber approximation,
- a link to FAST-PT (McEwen et al. 2016) for efficient implementation of nonlinear perturbation theory,
- and more power spectrum methods (see 4.8.9).

9. Feedback

If you would like to contribute to CCL or contact the developers, please do so through the CCL github repository located at https://github.com/LSSTDESC/CCL.

10. Citing CCL

If you use CCL in your work, please provide a link to the repository and cite it as LSST DESC (in preparation). For free use of the CLASS library, the CLASS developers require that the CLASS paper be cited: *CLASS II: Approximation schemes*, D. Blas, J. Lesgourgues, T. Tram, arXiv:1104.2933, JCAP 1107 (2011) 034. The CLASS repository can be found in http://class-code.net.

11. License

- Neither the name of CCL (https://github.com/LSSTDESC/CCL) nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

We would like to thank the organisers of the the DESC collaboration meetings at: Oxford (July 2016), SLAC (March 2016), and ANL (2015), and the LSST-DESC Hack Week organisers (CMU, November 2016), where this work was partly developed. We would also like to acknowledge the contribution of the participants of the TJP Code Comparison Project, some of whom are among the CCL contributors, for providing the benchmarks for testing CCL. Finally, we are grateful for the feedback received from other working groups of DESC, including Strong Lensing, Supernovae and Photometric Redshifts.

Author contributions are listed below.
Husni Almoubayyed: wrote an mcmc jupyter notebook example, reviewed code/contributed to issues.
David Alonso: Co-led project; developed structure for angular power spectra; implemented autotools; integrated into LSS pipeline; contributed to: background, power spectrum, mass function, documentation and benchmarks; reviewed code
Jonathan Blazek: Planning capabilities and structure; documentation and testing.
Philip Bull: Implemented the Python wrapper and wrote documentation for it; general bug fixes, maintenance, and code review; enhanced the installer and error handling system.
Jean-Éric Campagne: Angpow builder and contributed to the interface with CCL.
N. Elisa Chisari: Co-led project, coordinated hack projects & communication, contributed to: correlation function & power spectrum implementation, documentation, and comparisons with benchmarks.

Alex Drlica-Wagner: Helped with document preparation.
Zilong Du: Implemented the 3d correlation function.
Tim Eifler: Reviewed/tested code.
John Ellison: Implemented the 3d correlation function; wrote text describing 3d correlation function for this note.
Renée Hlozek: Contributed initial code for error handling structures, reviewed other code edits.
Mustapha Ishak: Contributed to planning of code capabilities and structure; reviewed code; identified and fixed bugs.
Shahab Joudaki: Created physical density function and documentation.
Matthew Kirby: Performed comparison of physical constants.
David Kirkby: Writing, testing and reviewing code. Asking questions.
Elisabeth Krause: Initiated and co-led project; developed CLASS interface and error handling; contributed to other code; reviewed pull requests.
C. Danielle Leonard: Wrote and tested code for LSST specifications, user-defined photo-z interface, and support of neutrinos; reviewed other code; wrote text for this note.
Christiane S. Lorenz: Contributed to accurate high-redshift cosmological background quantities and benchmarked background splines.
Phil Marshall: Helped with document preparation.
Thomas McClintock: Wrote Python documentation.
Sean McLaughlin: Wrote doxygen documentation and fixed bugs/added functionality to distances.
Jérémy Neveu: Contributed to Angpow and built the interface with CCL.
Stéphane Plaszczynski: Contributed to Angpow and contributed to the interface with CCL.
Javier Sanchez: Modified setup.py to allow pip installation and uninstall.
Sukhdeep Singh: Contributed to the correlation functions code.
Anže Slosar: Wrote and reviewed code.
Tilman Tröster: Wrote code for user-changable precision parameters, added distance and growth factor tests, found and fixed bugs.
Antonio Villarreal: Contributed to initial benchmarking, halo mass function code, and general code and issues review.
Michal Vrastil: Wrote documentation and example code, reviewed code.
Joe Zuntz: Wrote initial infrastructure, C testing setup, and reviewed code.

References

Alonso, D., Bull, P., Ferreira, P. G., Maartens, R., & Santos, M. G. 2015, ApJ, 814, 145

Angulo, R. E., Springel, V., White, S. D. M., et al. 2012, MNRAS, 426, 2046

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., et al. 2013, A&A, 558, A33

Bardeen, J. M., Bond, J. R., Kaiser, N., & Szalay, A. S. 1986, ApJ, 304, 15

Bartelmann, M., & Schneider, P. 2001, PhR, 340, 291

Blas, D., Lesgourgues, J., & Tram, T. 2011, CLASS: Cosmic Linear Anisotropy Solving System, Astrophysics Source Code Library, ascl:1106.020

Bonvin, C., & Durrer, R. 2011, PhRvD, 84, 063505

Campagne, J.-E., Neveu, J., & Plaszczynski, S. 2017, ArXiv e-prints, arXiv:1701.03592

Challinor, A., & Lewis, A. 2011, PhRvD, 84, 043516

Chang, C., Jarvis, M., Jain, B., et al. 2013, Monthly Notices of the Royal Astronomical Society, 434, 2121

Chevallier, M., & Polarski, D. 2001, International Journal of Modern Physics D, 10, 213

Dodelson, S. 2004, Modern Cosmology, Vol. 57, 60–61, doi:10.1063/1.1784308

Durrer, R. 2008, The Cosmic Microwave Background (Cambridge University Press)

Eisenstein, D. J., & Hu, W. 1998, ApJ, 496, 605

Hamilton, A. J. S. 2000, MNRAS, 312, 257

Hirata, C. M., Mandelbaum, R., Ishak, M., et al. 2007, MNRAS, 381, 1197

Hirata, C. M., & Seljak, U. 2004, PhRvD, 70, 063526

Joachimi, B., & Bridle, S. L. 2010, A&A, 523, A1

Joachimi, B., Schneider, P., & Eifler, T. 2008, A&A, 477, 43

Kamionkowski, M., & Spergel, D. N. 1994, ApJ, 432, 7

Lawrence, E., Heitmann, K., Kwan, J., et al. 2017, ApJ, 847, 50

Lesgourgues, J., & Pastor, S. 2012, ArXiv e-prints, arXiv:1212.6154

Linder, E. V. 2003, Physical Review Letters, 90, 091301

LSST Science Collaboration. 2009, ArXiv e-prints, arXiv:0912.0201

Mangano, G., Miele, G., Pastor, S., et al. 2005, Nuclear Physics B, 729, 221

McEwen, J. E., Fang, X., Hirata, C. M., & Blazek, J. A. 2016, JCAP, 9, 015

Peacock, J. A. 1999, Cosmological Physics, 704

Schneider, A., & Teyssier, R. 2015, JCAP, 12, 049

Takahashi, R., Sato, M., Nishimichi, T., Taruya, A., & Oguri, M. 2012, ApJ, 761, 152

Talman, J. 2009, Computer Physics Communications, 180, 332

Tinker, J., Kravtsov, A. V., Klypin, A., et al. 2008, ApJ, 688, 709

Tinker, J. L., Robertson, B. E., Kravtsov, A. V., et al. 2010, ApJ, 724, 878

Watson, W. A., Iliev, I. T., D'Aloisio, A., et al. 2013, MNRAS, 433, 1230

Yoo, J. 2010, PhRvD, 82, 083508

Yoo, J., Fitzpatrick, A. L., & Zaldarriaga, M. 2009, PhRvD, 80, 083514