Core Cosmology Library: Precision Cosmological Predictions for LSST

Husni Almoubayyed,[1] David Alonso,[2] Jonathan Blazek,[3] Philip Bull,[4,5] Jean-Éric Campagne,[6] N. Elisa Chisari,[2] Alex Drlica-Wagner,[7] Tim Eifler,[8,9] Renée Hlozek,[10] Mustapha Ishak,[11] David Kirkby,[12] Elisabeth Krause,[13] C. Danielle Leonard,[1] Phil Marshall,[14] Thomas McClintock,[15] Sean McLaughlin,[16] Jérémy Neveu,[6] Stéphane Plaszczynski,[6] Javier Sanchez,[12] Sukhdeep Singh,[1] Anže Slosar,[17] Antonio Villarreal,[18] Michal Vrastil,[19] 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] California Institute of Technology, Pasadena, CA 91125, USA
[5] Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, California, USA
[6] Laboratoire de l'Accélérateur Linéaire, Université Paris-Sud, CNRS/IN2P3, Université Paris-Saclay, Orsay, France
[7] Fermi National Accelerator Laboratory, P. O. Box 500, Batavia, IL 60510, USA
[8] Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109, USA
[9] Department of Physics, California Institute of Technology, Pasadena, CA 91125, USA
[10] Dunlap Institute for Astronomy and Astrophysics & Department for Astronomy and Astrophysics, University of Toronto, ON M5S 3H4
[11] Department of Physics, The University of Texas at Dallas, Richardson, TX 75083, USA
[12] Department of Physics and Astronomy, University of California, Irvine, CA 92697, USA
[13] Kavli Institute for Particle Astrophysics and Cosmology, Stanford, CA 94305-4085, USA
[14] SLAC National Accelerator Laboratory, Menlo Park, CA 94025, USA
[15] Univeristy of Arizona, Tucson, AZ 85721, USA
[16] Stanford University, Stanford, CA, 94305, USA
[17] Brookhaven National Laboratory, Physics Department, Upton, NY 11973, USA
[18] Department of Physics and Astronomy, University of Pittsburgh, Pittsburgh PA 15260
[19] Institute of Physics CAS, Prague, 182 21, CZ
[20] Institute for Astronomy, Royal Observatory Edinburgh, Edinburgh EH9 3HJ, UK

The Core Cosmology Library (`CCL`) provides routines to compute basic cosmological observables with validated numerical accuracy. These routines have been validated to a hereby documented accuracy level against the results of the Code Comparison Project. In the current version, predictions are provided for distances and background quantities, angular auto- and cross-spectra of cosmic shear and clustering and the halo mass function. Fiducial specifications for the expected LSST galaxy distributions and clustering bias are also included, together with a facility to compute redshift distributions for a user-defined photometric redshift model. `CCL` is written in C with a Python interface. In this note, we explain the functionality of the first release (`CCL` v0.1) of the library.

Contents

Introduction

In preparation for constraining cosmology with the Large Synoptic Survey Telescope (LSST), it is necessary to be able to produce theoretical predictions for the cosmological quantities which will be measured. The Core Cosmology Library[1] (CCL) aims to provide, in one library, predictions which are validated to a well-documented numerical accuracy for the purpose of constraining cosmology with LSST. By constructing a cosmology library 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 available 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[2]. A Python wrapper is also provided for improved ease of use.

This note describes how to install CCL (Section 2), its functionality (Section 3), the relevant unit tests (Section 4), the default configuration (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) and the license under which CCL is released (Section 10).

Installation

Dependencies

- GNU Scientific Library GSL[3], GSL-2.1 or higher
- Simplified Wrapper and Interface Generator SWIG[4]
- FFTlog[5] is provided within CCL with minor modifications
- SWIG is not needed to run CCL, but if you are developing new features, you will need SWIG.
- A version of Angpow (Campagne et al. 2017) is also incorporated within CCL.

Installation Procedure

[1] https://github.com/LSSTDESC/CCL
[2] Future versions of the library will incorporate other power-spectrum libraries and methods.
[3] https://www.gnu.org/software/gsl/
[4] http://www.swig.org/
[5] http://casa.colorado.edu/~ajsh/FFTLog/

CCL can be installed through an autotools-generated configuration file. UNIX users should be familiar with the process: navigate to the directory containing the library and type

```
$ ./configure
$ make
$ make install
```

(You may need to pre-append sudo to the last command, depending on your default privileges.) Users without admin privileges can install the library in a user-defined directory (e.g. /home/desc_fan/) by running

```
$ ./configure --prefix=/home/desc_fan
$ make
$ make install
```

This will create two directories (if not present already): /home/desc_fan/include and /home/desc_fan/lib where the header and lib files will be placed after running make install. CCL has been successfully installed in different Linux and Mac OS X systems[6].

After installing the C library you can make sure it is running as it should by typing make check, which will run the unit tests described in Section 4. You are now ready to install the Python wrapper following the steps described in Section 7.

Alternative installation and pip

There exists also the possibility of installing both the C library and the Python wrapper together by running

```
$ python setup.py install --prefix=/home/desc_fan
```

This will perform automatically the aforementioned steps to install the C library and will also install the Python library in /home/desc_fan/lib/python2.7/site-packages. You might need to add this

[6] We know of one case with Mac OS where libtools had the “lock” function set to “yes” and this caused the installation to stall. However, this is very rare. If this happens, after the configure step, edit libtool to set the “lock” to “no”.

path to your environment variable `$PYTHONPATH` to be able to use it. In the near future the package is planned to be `pip` installable.

Once installed you can check the installation status by typing

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

This will run the embedded unit tests. Using this last method to install the Python library allows you to uninstall it just by running

```
$ python setup.py uninstall
```

Compiling against an external version of CLASS

`CCL` has a built-in version of `CLASS` that is used to calculate power spectra and other cosmological functions. This is compiled by default. Optionally, you can also link `CCL` against an external version of `CLASS`. This section useful if you want to use a modified version of `CLASS`, or a different or more up-to-date version of the standard `CLASS`. For example, we have sucessfully compiled and ran `CCL` with `hiCLASS` (Zumalacarregui et al. 2016) for Horndeski models.

To compile `CCL` with an external version of `CLASS`, you must first prepare the external copy so that it can be linked as a shared library. By default, the `CLASS` build tools create a static library. After compiling `CLASS` in the usual way (by running `make`), look for a static library file called `libclass.a` that should have been placed in the root source directory. Then, run the following command from that directory (Linux only):

```
$ gcc -shared -o libclass.so -Wl,--whole-archive libclass.a \
                              -Wl,--no-whole-archive -lgomp
```

This should create a new shared library, `libclass.so`, in the same directory. (N.B. The `-lgomp` flag has to appear at the end of the command; otherwise the linker can fail.) If you are running Mac OS X, use the following command instead:

```
$ gcc -fpic -shared -o libclass.dylib -Wl,-all_load libclass.a -Wl,-noall_load
```

Next, change to the root CCL directory and run `make clean` if you have previously run the compilation process. Then, set the CLASSDIR environment variable to point to the directory containing `libclass.so`:

```
export CLASSDIR=/path/to/external/class
```

Then, run `./configure` and compile and install CCL as usual. The CCL build tools should take care of linking to the external version of CLASS.

Once compilation has finished, run `make check` to make sure everything is working correctly. Remember to add the external CLASS library directory to your system library path, using either `export LD_LIBRARY_PATH=/path/to/external/class` (Linux) or `export DYLD_FALLBACK_LIBRARY_PATH=/path/to/external/class` (Mac). The system must be able to find both the CCL and CLASS libraries; it is not enough to only add CCL to the library path.

Creating a Docker Image

CCL supports the use of the Docker project for the deployment of applications inside software containers. The included Dockerfile allows for quick creation of an image file that will allow for easy realization of virtual machines that require no additional dependencies beyond the Docker software. The Docker website[7] details the installation process for most common operating systems.

With Docker is installed, creating an image is relatively straightforward:

```
docker build -t ccl .
```

This will begin the process of creating the image file for CCL locally. This Dockerfile contains all installed C libraries in CCL as well as the Python wrapper. It currently utilizes `python:2.7` as a base image and supports both ipython and Jupyter notebook. The virtualization process should have minimum impact on performance.

The resulting Docker image has two primary functionalities. The first is a command that will open a Jupyter notebook tied to a port connection on your local machine. This can be utilized with the following command on run:

[7] https://www.docker.com

```
docker run -p 8888:8888 ccl
```

This Jupyter notebook can be accessed on a browser of your choice at `localhost:8888`. The second possible run is to access the bash itself, which can be done with the command:

```
docker run -it ccl bash
```

This will allow for full access to the virtual machine for installation of any additional desired dependencies.

Functionality

Supported cosmological models

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

- Flat, vanilla ΛCDM
- wCDM and the CPL model ($w_0 + w_a$)
- Non-zero curvature (K)
- All the above plus an arbitrary, user-defined, modified growth function (see description in Section 3.4)
- Relativistic neutrinos and / or a single massive neutrino or multiple equal-mass massive neutrinos

Not all functionalities are available for all models. For a reference of what predictions are available for each model, see Table 1.

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

Table 1. Cosmologies implemented in CCL.

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

`ccl_parameters` objects contain information about the cosmological parameters, 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_nu_rel, double N_nu_mass, double mnu, double w0, double wa, double h, double norm_pk, double n_s,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 passed the power spectrum normalization parameterized by σ_8 or A_s, `ccl_parameters_create` switches to σ_8 normalization if `norm_pk` $> 1.e-5$, and to A_s normalization otherwise.

To include neutrinos (massive or massless), the suffix '`_nu`' may be appended to the latter four `ccl_configuration` objects above (for example: `ccl_parameters_create_lcdm_nu(...)`). Utilizing the corresponding `ccl_configuration` object without this `_nu` suffix will set the contribution from both massless and massive neutrinos to zero. In the case of non-zero massive neutrinos, it may be desirable to set `N_nu_rel` such that $N_{\rm eff} = 3.046$ in the early universe. In agreement with `CLASS`, when using the default value of $T_{\rm NCDM}$ as described in section 3.2 below, N_nu_rel can be set to 2.0328, 1.0196, and 0.00641 for 1, 2 and 3 massive neutrinos respectively to achieve this.

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 readily available as default_config, for which transfer functions are computed with CLASS, the HaloFit prediction is used for the matter power spectrum and a number of prescriptions can be used to compute the halo mass function.

After initializing an instance of ccl_parameters and ccl_configuration, the function ccl_cosmology_create(ccl_parameters,ccl_configuration) returns a pointer to a ccl_cosmology structure, which you will need to pass around to every CCL function.

Directions to an example of CCL script are provided in Section 6. The README file has additional extensive documentation for the example run and also regarding the installation.

Distances

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

The Hubble parameter is calculated via

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

The radial comoving distance is calculated via a numerical integral

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

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

$$r(\chi) = \begin{cases} k^{-1/2}\sin(k^{1/2}\chi) & k>0 \\ \chi & k=0 \\ |k|^{-1/2}\sinh(|k|^{1/2}\chi) & k<0 \end{cases} \tag{3}$$

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

CCL can compute the distance modulus, as defined (with d_L in units of parsecs):

$$\mu = 5 \log_{10}(d_L) - 5 \tag{4}$$

CCL also contains capability to compute $a(\chi)$ (i.e. the inverse of $\chi(a)$).

Density parameter functions

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

The density parameter functions $\Omega_X(a)$ are 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 $\Omega_\Lambda(a) = \Omega_{\Lambda,0} H_0^2/H^2(a)$,
- 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 $\Omega_{\nu,\mathrm{m}}(a)$,

using the Hubble parameter defined in equation 1.

For massive neutrinos, $\Omega_{\nu,\mathrm{m}}(a)$ is calculated by calling a set of functions contained in `ccl_neutrinos.c`. First, we assume that the mass of one neutrino is equal to $m_\nu^{\mathrm{tot}}/N_{\mathrm{m}}^\nu$ (recalling that we assume a single massive neutrino or equal-mass neutrinos). We then compute a quantity called the effective temperature:

$$T_\nu^{\mathrm{eff}} = T_{\mathrm{CMB}} T_{\mathrm{NCDM}}. \tag{5}$$

T_{NCDM} here is used to explicitly set the value of m_ν/Ω_ν^0. We choose a default value of $T_{\mathrm{NCDM}} = 0.71611$, which corresponds to $m_\nu/\Omega_\nu^0 = 93.14$ eV, to agree with the default value set by `CLASS`. We define

$$\mu = \frac{m_\nu^{\mathrm{tot}} a}{N_{\mathrm{m}}^\nu T_\nu^{\mathrm{eff}}} \tag{6}$$

in units such that μ is dimensionless. We then conduct the phase-space integral required to get the neutrino density, and multiply by appropriate factors to obtain $\Omega_{\nu,\mathrm{m}}(a)$:

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

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 $\sim 0.2776\mu$.

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:

$$\frac{d}{da}\left(a^3 H(a) \frac{dD}{da} \right) = \frac{3}{2} \Omega_M(a) a H(a) D. \quad (8)$$

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

$$f(a) = \frac{d \ln D}{d \ln a}. \quad (9)$$

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 only for a Universe containing just dust-like components. A scale-independent growth rate is, for example, ill-defined in the presence of massive neutrinos.

Currently CCL allows for an alternative 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), however 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.

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`.

BBKS

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

$$T(q \equiv k/\Gamma h\mathrm{Mpc}^{-1}) = \frac{\ln[1+2.34q]}{2.34q}[1+3.89q+(16.2q)^2+(5.47q)^3+(6.71q)^4]^{-0.25} \tag{10}$$

where $\Gamma = \Omega_m h$. The power spectrum is related to the transfer function by $\Delta(k) \propto T^2(k)k^{3+n}$ and $\Delta^2(k) \propto k^3 P(k)$. The normalization of the power spectrum is achieved 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. 4), and it is not recommended for other uses.

Eisenstein and Hu

CCL also provides an approximation to the matter power spectrum as implemented by Eisenstein & Hu (1998) (we refer the reader to this paper for a detailed discussion of the fitting formulae)[8].

It is worth reminding the reader that the Eisenstein & Hu and the BBKS approximations are not accurate and should not be used to derive scientific results. They are, however, computationally faster, and therefore useful for code testing and plotting.

CLASS

Secondly, there is the option to call the CLASS software (Blas et al. 2011) within CCL to obtain either linear or nonlinear matter power spectra at given redshifts. For speed, the linear power spectrum is obtained at redshift $z = 0$ and re-scaled to

[8] 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 $k < 10\,h\,\mathrm{Mpc}^{-1}$.

a different redshift using the growth function. In the case of the nonlinear matter power spectrum, upon setting up the cosmology object, we construct a bi-dimensional spline in k and the scale-factor which is then called 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`. Currently CLASS computes the non-linear power spectrum using the HaloFit prescription of Takahashi et al. (2012).

Nonlinear low and high wavenumber extrapolation

To do: This section needs careful checking, many things have changed here.

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.

We describe first 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 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. We then apply a second order Taylor expansion to extrapolate the matter power spectrum to $k >$`K_MAX_SPLINE`. The Taylor expansion gives

$$\ln P(k,z) \simeq \ln P(k_0,z) + \frac{d\ln P}{d\ln k}(\ln k_0, z)(\ln k - \ln k_0) + \frac{1}{2}\frac{d^2\ln P}{d\ln k^2}(\ln k_0, z)(\ln k - \ln k_0)^2. \tag{11}$$

The results of this approximation are shown in Figure 1 for redshifts $z = 0$ and $z = 3$ To do: This figure needs to be updated. We compare the nonlinear matter power spectrum at $z = 0$ computed with the previously described approximation, to the matter power spectrum obtained by directly evaluating `CLASS` at the desired k value. (The benchmark in reality uses a value of `K_MAX_SPLINE`$= 10^3$/Mpc, well beyond our range of application.) We find that for typical values of $\Delta \ln k$ is 10^{-2} and `K_MAX_SPLINE`$= 50$/Mpc has converged to an accuracy that surpasses the expected impact of baryonic effects on the matter power spectrum at $k > 10/$Mpc. (For a plot showing the impact of baryons on the matter power spectrum, see Schneider

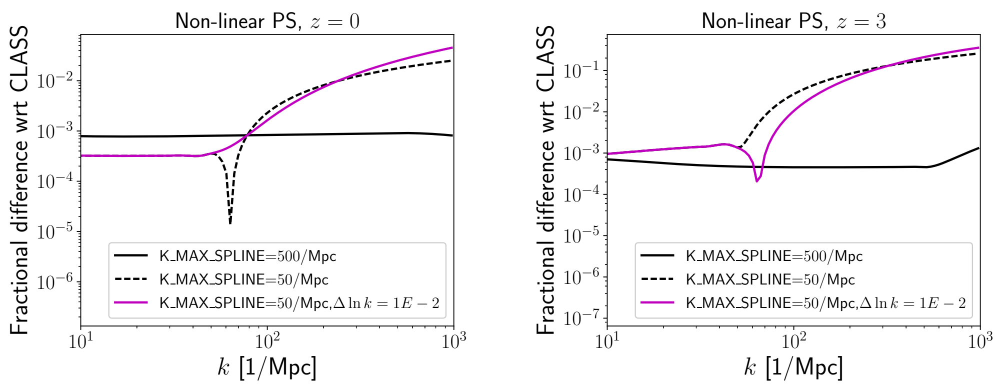

Figure 1. The relative error produced by splining the nonlinear matter power spectrum up to `K_MAX_SPLINE` and extrapolating beyond this value with a second order Taylor expansion the natural logarithm of the matter power spectrum. The left panel shows the results at $z = 0$. The right panel shows the results at $z = 3$. The fiducial parameters adopted are those corresponding to the magenta curve.

& 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 adopt a similar approach to extrapolate the power spectrum at small scales. In this case, the power spectrum below `K_MIN` is obtained by a power-law extrapolation with index n_s:

$$\log P(k < \mathtt{K\_MIN}, z) = \log P(\mathtt{K\_MIN}, z) + n_s(\log k - \log \mathtt{K\_MIN}) \tag{12}$$

To do: Explain choice of default values for `K_MIN`.

Linear extrapolation—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. We have tested here how the procedure described in the previous section affects the convergence of the linear matter power spectrum. We compare the fiducial `CCL` output to the case where we set `K_MAX_SPLINE`$= 10^3$/Mpc. The result is shown in Figure 2. Although there is a significant difference ($\gtrsim 10\%$) between the linear power spectra at large k, we have confirmed that the difference in σ_8 is negligible. Nevertheless, for other applications that use the linear power spectrum, the user might need to increase the value of `K_MAX_SPLINE`.

As in the previous section, the power spectrum at small wavenumber is extrapolated using a power-law. To do: Explain choice of `K_MIN` too.

To do: Different choices of K_MIN and K_MAX for BBKS and EH implementations. Check whether we are using extrapolation for those cases.

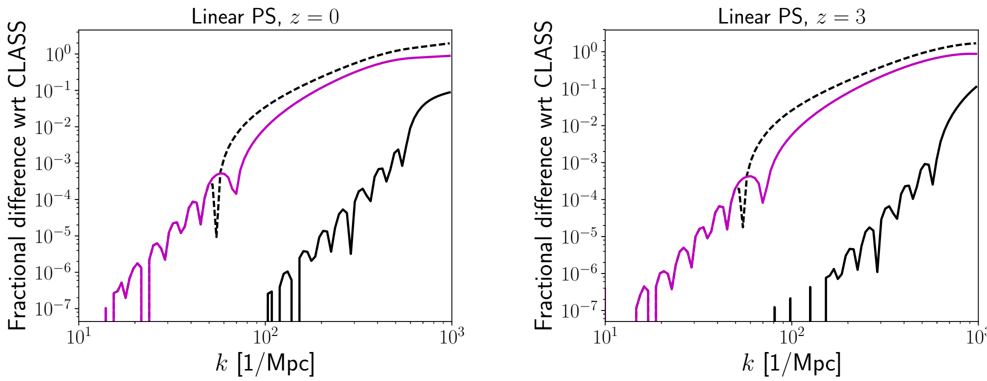

Figure 2. Same as Fig. 1 but for the linear matter power spectrum at $z = 0$ (left) and $z = 3$ (right).To do: This figure needs to be updated and to include the low k extrapolation.

Wishlist for the future

This is a list of some power spectrum methods that we would like to implement in the future:

- CAMB,
- Cosmic emulators,
- halo model/HOD.

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, 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 3.8.

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` $> 1.e-5$, and to $A_{\rm s}$ normalization otherwise.

Angular power spectra

In this section we will distinguish between *observables* (inseparable 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 main density term in number counts, redshift-space distortions, magnification, ISW, etc.). The routines described in this subsection are implemented in `ccl_cls.c`.

Exact expressions

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

$$C_\ell^{ab} = 4\pi \int_0^\infty \frac{dk}{k}\, \mathcal{P}_\Phi(k) \Delta_\ell^a(k) \Delta_\ell^b(k), \tag{13}$$

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:

$$\Delta_\ell^{\rm D}(k) = \int dz\, p_z(z)\, b(z)\, T_\delta(k,z)\, j_\ell(k\chi(z)), \tag{14}$$

  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:

$$\Delta_\ell^{\rm RSD}(k) = \int dz\, p_z(z) \frac{(1+z)p_z(z)}{H(z)} T_\theta(k,z)\, j_\ell''(k\chi(z)), \tag{15}$$

  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; this is why C_ℓ spectra involving number count tracers are not supported when massive neutrinos are present.

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

$$\Delta_\ell^{\rm M}(k) = -\ell(\ell+1)\int\frac{dz}{H(z)}W^{\rm M}(z)T_{\phi+\psi}(k,z)j_\ell(k\chi(z)), \tag{16}$$

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

$$W^{\rm M}(z) \equiv \int_z^\infty dz' p_z(z')\frac{2-5s(z')}{2}\frac{r(\chi(z')-\chi(z))}{r(\chi(z'))}. \tag{17}$$

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. 3).

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, and therefore it is safe to work without them for the time being.

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:

$$\Delta_\ell^{\rm L}(k) = -\frac{1}{2}\sqrt{\frac{(\ell+2)!}{(\ell-2)!}}\int\frac{dz}{H(z)}W^{\rm L}(z)T_{\phi+\psi}(k,z)j_\ell(k\chi(z)), \tag{18}$$

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

$$W^L(z) \equiv \int_z^\infty dz' p_z(z')\frac{r(\chi(z')-\chi(z))}{r(\chi(z'))}. \tag{19}$$

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

$$\Delta_\ell^{\rm IA}(k) = \sqrt{\frac{(\ell+2)!}{(\ell-2)!}}\int dz\, p_z(z)\, b_{\rm IA}(z)\, f_{\rm red}(z)\, T_\delta(k,z)\,\frac{j_\ell(k\chi(z))}{(k\chi(z))^2}. \tag{20}$$

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 an issue for the time being, but it will be revisited in future versions of CCL.

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:

$$j_\ell(x) \simeq \sqrt{\frac{\pi}{2\ell+1}}\,\delta\left(\ell+\frac{1}{2}-x\right). \tag{21}$$

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

Substituting this in the expressions above, it is possible to see that they can be written as follows in the Limber approximation. First, the power spectrum can be rewritten as

$$C_\ell^{ab} = \frac{2}{2\ell+1}\int_0^\infty dk\, P_\delta\,(k, z_\ell)\,\tilde{\Delta}_\ell^a(k)\tilde{\Delta}_\ell^b(k). \tag{22}$$

where

$$\tilde{\Delta}_\ell^{\rm D}(k) = p_z(z_\ell)\, b(z_\ell)\, H(z_\ell) \tag{23}$$

$$\tilde{\Delta}_\ell^{\rm RSD}(k) = \frac{1+8\ell}{(2\ell+1)^2}\, p_z(z_\ell)\, f(z_\ell)\, H(z_\ell) - \tag{24}$$

$$\frac{4}{2\ell+3}\sqrt{\frac{2\ell+1}{2\ell+3}} p_z(z_{\ell+1})\, f(z_{\ell+1})\, H(z_{\ell+1}) \tag{25}$$

$$\tilde{\Delta}_\ell^{\rm M}(k) = 3\Omega_{M,0}H_0^2\frac{\ell(\ell+1)}{k^2}\,\frac{(1+z_\ell)}{\chi_\ell}W^{\rm M}(z_\ell) \tag{26}$$

$$\tilde{\Delta}_\ell^{\rm L}(k) = \frac{3}{2}\Omega_{M,0}H_0^2\sqrt{\frac{(\ell+2)!}{(\ell-2)}}\frac{1}{k^2}\,\frac{1+z_\ell}{\chi_\ell}W^{\rm L}(z_\ell) \tag{27}$$

$$\tilde{\Delta}_\ell^{\rm IA}(k) = \sqrt{\frac{(\ell+2)!}{(\ell-2)!}}\frac{p_z(z_\ell)\, b_{\rm IA}(z_\ell) f_{\rm red}(z_\ell) H(z_\ell)}{(\ell+1/2)^2} \tag{28}$$

Beyond Limber: angpow

To do: Comment out this section if we don't make it to public release.

Our aim is to compute the angular over density power spectrum $C_\ell(z_1, z_2)$ as a cross-correlation between two z-shells with mean values (z_1, z_2) and also the auto-correlation $C_\ell(z_1)$ with $z_1 = z_2$, taking into account, in both cases, possible redshift selection functions and physical processes as redshift space distortions without any Limber numerical approximation. For this putpose, `CCL` has been linked to the`Angpow` code (Campagne et al. 2017), which is briefly described here.

The angular power spectrum for two shells, $C_\ell(z_1, z_2)$, is computed according to the following expression

$$\begin{aligned} & C_\ell(z_1, z_2; \sigma_1, \sigma_2) \\ & \quad = \iint_0^\infty \mathrm{d}z\mathrm{d}z' \, W_1(z; z_1, \sigma_1) W_2(z'; z_2, \sigma_2) \\ & \qquad \times \int_0^\infty \mathrm{d}k \, f_\ell(z,k) f_\ell(z',k). \end{aligned} \tag{29}$$

where we have introduced two normalized redshift selection functions $W_1(z; z_1, \sigma_1)$ and $W_2(z'; z_2, \sigma_2)$ around z_1 and z_2 with typical width σ_1 and σ_2, respectively. The auxiliary function $f_\ell(z,k)$ can be defined without loss of generality as

$$f_\ell(z,k) \equiv \sqrt{\frac{2}{\pi}} \, k \sqrt{P(k,z)} \, \widetilde{\Delta}_\ell(z,k) \tag{30}$$

with

- $P(k,z)$: the power spectrum at redshift z which is defined either using the primordial power spectrum $P_{in}(k)$ and the transfer function $T(k,z)$ as

  $$P(k,z) = P_{in}(k) T(k,z), \tag{31}$$

  or using an approximation valid at low redshift with the power spectrum at $z = 0$ ($P(k)|_{z=0}$) and the growth factor $G(z)$ as

  $$P(k,z) = P(k)|_{z=0} G^2(z); \tag{32}$$

  Their are abstract classes that one may tuned according to the use-case (see `angpow_integrand_base.h` and `angpow_powspec_base.h`). For instance we have specialized in `angpow_powspec.h` the computation of a $P(k)$ from an external file and a growth rate from Lahav et al. (1991); Carroll et al. (1992), and in `angpow_integrand.h` we have included the RSD term and a constant bias.

- $\widetilde{\Delta}_\ell(z,k)$: a function describing the physical processes 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). As example To do: example for galaxy clustering? with a constant bias b and using the growth factor rate $f_a(z) = d \log G(a(z))/d \log(a(z))$, we can approximate

  $$\widetilde{\Delta}_\ell(z,k) \approx b j_\ell(k\chi(z)) - f_a(z) j''_\ell(k\chi(z)) + \dots \tag{33}$$

with $j_\ell(x)$ and $j''_\ell(x)$ the spherical Bessel function of order ℓ and its second derivative, and $\chi(z)$ is the comoving distance at redshift z.

To proceed to a numerical evaluation of equation 29, we first conduct inside the rectangle $[z_{1\min}, z_{1\max}] \times [z_{2\min}, z_{2\max}]$ given by the W 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, we use the Clenshaw-Curtis quadrature. 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 $i = 0, \dots, N_{z_1} - 1$ and $j = 0, \dots, N_{z_2} - 1$. Then, one gets the following approximation:

$$C_\ell^{\text{thick}}(z_1, z_2) \approx \sum_{i=0}^{N_{z_1}-1} \sum_{j=0}^{N_{z_2}-1} w_i w_j W_1(z_i, z_1) W_2(z_j, z_2) \widehat{P}_\ell(\chi_i, \chi_j) \tag{34}$$

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

$$\widehat{P}_\ell(z_i, z_j) = \int_0^\infty dk\, f_\ell(z_i, k) f_\ell(z_j, k), \tag{35}$$

defined with the $f_\ell(z, k)$ function of equation 30. 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 (35) 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.

To do: Can we have a sentence or two with motivation and comparison of angpow vs other methods? We have tested the code both on laptop (Linux, MacOSX) as well as on Computer Center (CCIN2P3 in France and NERSC in the USA).

We use OpenMP to distribute the computation of a given C_ℓ on a single thread. The code wall time decreases reasonably well with the number of threads and a wall time at the $\mathcal{O}$(1s) level can be reached to reconstruct these accurate spectra. Such performances are much higher than those obtained with `CLASSgal` when not using the Limber approximation. For instance using`CLASSgal` to compute the auto-correlation with a Gaussian selection function with ($z_{\mathrm{mean}} = 1$, $\sigma_z = 0.02$, $5\sigma_z$-cut) and $k_{\max} = 1\,\mathrm{Mpc}^{-1}$ and a radial quadrature based on $N_{\mathrm{pts}} = 139$ sample points, it takes about 15s on 16 threads, which is to be compared to about 0.5s with `Angpow`.

Correlation functions

To do: This section is new. Sukhdeep and Elisa to update.

In the Limber approximation, the angular correlation function between any two tracers 1 and 2 is given by[9]

$$C_{12}(\theta) = \int d\chi' q_1(\chi') q_2(\chi') \int dk \frac{k}{2\pi} P_\delta(k, \chi') J_0[f_K(\chi')\theta k], \tag{36}$$

where J_0 is the Bessel function of order 0, χ is the comoving distance and $f_K(\chi)$ is the radial function that multiplies the spatial metric element, which is different from χ in the case of a cosmology with non-zero curvature.

Clustering. For clustering, the relevant weight is given by $q_1(\chi) = b_1(\chi) dN_1/d\chi(\chi)$, the comoving distance probability distribution times the bias. The angular correlation function can also be re-written as

$$C_{12}(\theta) = \int dl \frac{l}{2\pi} C_{gg}(l) J_0(l\theta), \tag{37}$$

where C_{gg} is the galaxy clustering angular power spectrum.

Lensing. Lensing correlation functions are [10]

$$\xi_+(\theta) = \int_0^\infty dl \frac{l}{2\pi} J_0(l\theta) P_\kappa(l), \tag{38}$$

$$\xi_-(\theta) = \int_0^\infty dl \frac{l}{2\pi} J_4(l\theta) P_\kappa(l), \tag{39}$$

[9] See Bartelmann and Schneider (1999) weak lensing review, page 44. See also Joachimi & Bridle (2010)

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

where the angular lensing convergence power spectrum is given by

$$P_\kappa(l) = \frac{9H_0^4\Omega_m^2}{4c^4}\int_0^{\chi_h}\frac{d\chi}{a^2(\chi)}P_\delta\left(\frac{l}{f_K(\chi)},\chi\right)\left[\int_\chi^{\chi_h}d\chi' p_\chi(\chi')\frac{f_K(\chi'-\chi)}{f_K(\chi')}\right]^2 \tag{40}$$

(41)

For numerical integration of the correlation functions, we make use of the public code `FFTlog`[11]. A version of this code is included in `CCL` with minor modifications.

To do: Describe Legendre polynomial option

Halo mass & halo bias functions

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

The halo mass function is incorporated using several 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 $\Delta = 200\mathrm{m}$, where a halo is defined with:

$$\bar{\rho}(r_\Delta) = \Delta * \rho_\mathrm{m}, \tag{42}$$

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, ρ_m. Note that another common definition utilizes the critical density of the universe, ρ_c; currently `CCL` requires an adjustment to the Δ value provided if this definition is used. We may look into additional methods of implementation in the future that will remove this additional calculation.

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. We choose not to use the interpolation fitting functions included in this work due to the introduced inaccuracies from the specified values at distinct Δ values and instead utilize a spline fitting method. We look toward extending to more general halo definitions in the future, though this implementation is not yet in practice.

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

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:

$$f(\sigma) = A\left[\left(\frac{\sigma}{b}\right)^{-a} + 1\right] e^{-c/\sigma^2}, \tag{43}$$

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

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

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 $z = 0$:

$$M = \frac{H_0^2}{2G} R^3 \rightarrow \frac{M}{M_\odot} = 1.162 \times 10^{12} \Omega_M h^2 \left(\frac{R}{1\,\mathrm{Mpc}}\right)^3. \tag{45}$$

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

$$\sigma_R^2 = \frac{1}{2\pi^2} \int dk\, k^2\, P_k\, \tilde{W}_R^2(k) \tag{46}$$

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

$$\tilde{W}_R(k) = \frac{3}{(kR)^3}[\sin(kR) - kR\cos(kR)] \tag{47}$$

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

$$\underline{}(\nu) = 1 - A\frac{\nu^a}{\nu^a + {\delta_c}^a} + B\nu^b + C\nu^c, f(\nu) = \alpha[1 + (\beta\nu)^{-2\phi}]\nu^{2\eta}e(-\gamma\nu^2/2). \quad (48)$$

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.

Photo-z implementation

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

LSST galaxy redshifts will be obtained using photometry. However, analytic forms of galaxy redshift distributions are usually known in terms of spectroscopic redshifts. A model is therefore required for the probability of measuring a photometric redshift $z_{\rm ph}$ for an object with hypothetical spectroscopic redshift $z_{\rm s}$. CCL allows the user to flexibly provide their own photometric redshift model.

To do so, 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, this function should take the form:

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

The user must then also provides the structure of further parameters (`user_par`). This model can be incorporated when computing $\frac{dN}{dz}^i$ in photometric redshift bin i, as given by equation 51, below.

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 predictions or 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 spectroscopic 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:

$$\frac{dN}{dz} \propto z^{\alpha} \exp\left(-\frac{z}{z_0}^{\beta}\right). \tag{49}$$

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):

$$\frac{dN}{dz} \propto \frac{1}{2z_0}\left(\frac{z}{z_0}\right)^2 \exp\left(-\frac{z}{z_0}\right) \tag{50}$$

with $z_0 = 0.3$. The above $\frac{dN}{dz}$ for lensing sources in fact represents a subset of the $\frac{dN}{dz}$ for clustering.

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 3.9. 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 wishes the assume 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:

$$\frac{dN^i}{dz} = \frac{\frac{dN}{dz}\int_{z_i}^{z_{i+1}} dz' p(z, z')}{\int_{z_{\rm min}}^{z_{\rm max}} dz \frac{dN}{dz} \int_{z_i}^{z_{i+1}} dz' p(z, z')} \tag{51}$$

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):

$$b(z) = \frac{0.95}{D(z)} \tag{52}$$

where $D(z)$ in the linear growth rate of structure.

Tests and validation

Our goal is for outputs of CCL to be validated against the results of the code comparison project down to a 10^{-4} or better 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 $z = 0, 1, 2, 3, 4, 5$,
2. comoving radial distance [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 (Mpc$/h)^3$ at $z = 0, 2$ in the range $10^{-3} \leq k \leq 10h/$Mpc with 10 bins per decade, and
4. 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 forecasts 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 (with one exception), and for $P(k)$ and $\sigma(M)$ among codes which use the same BBKS conventions.

Cosmological models for code comparison project								
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

We noticed that there are 2 typos for the BBKS transfer function in "Modern Cosmology" (Dodelson & Efstathiou 2004) compared to the original BBKS paper. The quadratic term should be $(16.1q)^2$ and the cubic term should be $(5.46q)^3$. On the other hand, 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 $\{k_{\rm min}, k_{\rm max}\}$. 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 submission.

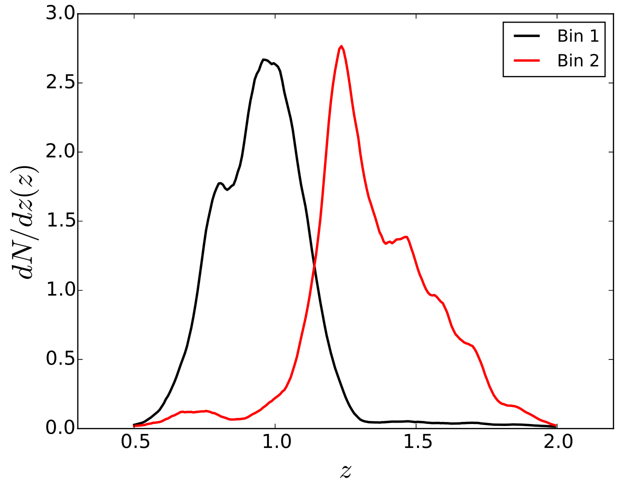

Figure 3. Binned redshift distributions used for code comparison project.

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 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 3.

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:

- The cross-correlation between bins is particularly sensitive to having enough points to sample the lensing kernel.
- The nonlinear behaviour is sensitive to $l_{\max}$, we had to go up to 30,000 to get convergence (and we could not achieve 0.01% convergence).
- The large scales are sensitive to $l_{\min}$ (which also prompts a question about using the Limber approximation or not).
- The correlation functions are sensitive to how the power spectrum is interpolated. For example, in one case we had fewer l's and we had to use an order 5 spline. If we sample at all l's then a linear interpolation is enough.

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),

$$\log[(M^2/\bar{\rho}_m)dn/dM]. \tag{53}$$

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.

Finally, we note that formal tests for predictions in cosmologies with neutrinos are not yet included. The support for neutrino cosmologies in the current release is therefore not yet formally validated, although outputs have been informally compared against other codes where available. Formal validation of this functionality is ongoing.

`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`.

To do: What new checks were added?

To do: Add correlation function benchmarks (Elisabeth)

Default configuration

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

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 galaxy lensing.

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

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 (with OpenMP enabled), and that you have a working Python 2.x installation with numpy and distutils with swig. 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 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.

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 Parameters object, containing cosmo parameter values
p = ccl.Parameters(Omega_c=0.27, Omega_b=0.045, h=0.67, A_s=2e-9, n_s=0.96)

# Create new Cosmology object with these parameters. This keeps track of
# previously-computed cosmological functions
cosmo = ccl.Cosmology(p)

# 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, z_n, n)
lens2 = ccl.ClTracerLensing(cosmo, False, 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 `tests/` directory. These are:

```
Photo-z example.ipynb,
```

`Power spectrum example.ipynb`,

`Distance Calculations Example.ipynb`,

`Lensing angular power spectrum.ipynb`.

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.

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 FAST-PT (McEwen et al. 2016) for implementation of perturbation theory,
- a more realistic photo-z model,
- support for cosmologies with multiple unequal-mass neutrinos,
- and more power spectrum methods (see 3.5.5).

Feedback

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

License

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: 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: Contributed to planning of code capabilities and structure.
Philip Bull: Implemented the Python wrapper and wrote documentation for it.
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.
Tim Eifler: reviewed/tested code
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."
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.
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 build 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.
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

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

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

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

Carroll, S. M., Press, W. H., & Turner, E. L. 1992, ARA&A, 30, 499

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

Dodelson, S., & Efstathiou, G. 2004, Physics Today, 57, 60

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

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

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

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

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

Lahav, O., Lilje, P. B., Primack, J. R., & Rees, M. J. 1991, MNRAS, 251, 128

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

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

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

Zumalacarregui, M., Bellini, E., Sawicki, I., Lesgourgues, J., & Ferreira, P. G. 2016, ArXiv e-prints, arXiv:1605.06102