LSST DESC Notes

Core Cosmology Library: Precision Cosmological Predictions for LSST

David Alonso, Jonathan Blazek, Phil Bull, Nora Elisa Chisari, Tim Eifler, Renée Hlozek, David Kirkby, Elisabeth Krause, C. Danielle Leonard, Phil Marshall, Sukhdeep Singh, Anze Slosar, Antonio Villarreal, Michal Vrastil, Joe Zuntz, and Theory and Joint Probes Working Group

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 Comparsion 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

1. 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).

2. Installation

2.1. Dependencies

- GNU Scientific Library GSL[3], GSL-2.1 or higher
- Simplified Wrapper and Interface Generator SWIG[4]

2.2. Installation Procedure

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

[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/

```
$ ./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[5].

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.

3. Functionality

3.1. Supported cosmological models

Ultimately, CCL will aim to incorporate theoretical predictions for all cosmological models of interest to LSST. Currently, however, only a few 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 3.3).

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

Table 1. Cosmologies implemented in CCL.

Observable/Model	flat ΛCDM	ΛCDM+K	wCDM	$w_0 + w_a$	MG
Distances	✓	✓	✓	✓	X
Growth	✓	✓	✓	✓	✓
$P_m(k, z)$	✓	✓	✓	✓	X
Halo Mass Function	✓	✓	✓	✓	X
C_l	✓	✓	✓	✓	X

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.

`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 w0, double wa, double h, double norm_pk, double n_s,int nz_mgrowth,double *zarr_mgrowth,double *dfarr_mgrowth)`: 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.

`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.

3.2. **Distances**

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

The Hubble parameter is calculated via

$$\frac{H(a)}{H_0} = a^{-3/2}\sqrt{\Omega_m + \Omega_\Lambda a^{-3(w_0+w_a)}\exp[3w_a(a-1)] + \Omega_K a + \Omega_g a^{-1}}. \quad (1)$$

The radial comoving distance is calculated via a numerical integral

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

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

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

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

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

3.3. **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:

$$\frac{d}{da}\left(a^3 H(a)\frac{dD}{da}\right) = \frac{3}{2}\Omega_M(a)H(a)D. \quad (4)$$

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

$$f(a) = \frac{d\ln D}{d\ln a}. \quad (5)$$

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

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.

3.4. **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`.

3.4.1. **BBKS**

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

$$T(q \equiv k/\Gamma h\mathrm{Mpc}^{-1}) = \frac{\ln[1+2.34q]}{2.34q}[1+3.89q+(16.2q)^2+(5.47q)^3+(6.71q)^4]^{-0.25} \quad (6)$$

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.

3.4.2. 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 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-facor 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).

3.4.3. Nonlinear extrapolation

The computation of the nonlinear power spectrum from CLASS can be significantly sped up by extrapolating in the range $k >$K_MAX_SPLINE. In this section, we describe the implementation of the extrapolation and the accuracy attained.

The introduction of the parameter K_MAX_SPLINE allows us to spline the nonlinear 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 routine `ccl_nonlin_matter_power`.

First, we compute the first and second derivative of the $\ln P(k,z)$ at $k_0 = \mathrm{K\_MAX} - 2\Delta \ln k$ by computing the numerical derivatives by finite differences. We define the following k values:

$$k_0^{+} = k_0 + \Delta \ln k, \tag{7}$$
$$k_0^{++} = k_0 + 2\Delta \ln k, \tag{8}$$
$$k_0^{-} = k_0 - \Delta \ln k, \tag{9}$$
$$k_0^{--} = k_0 - 2\Delta \ln k, \tag{10}$$

where we evaluate the power spectra for computing the suitable numerical derivatives as

$$\frac{d\ln P}{d\ln k}(k_0,z) = \frac{-\ln P(k_0^{++},z) + 8\ln P(k_0^{+},z) - 8\ln P(k_0^{-},z) + \ln P(k_0^{--},z)}{2\Delta\ln k} \tag{11}$$

$$\frac{d^2\ln P}{d\ln k^2}(k_0,z) = \frac{\ln P(k_0^{+},z) - 2\ln P(k_0) + \ln P(k_0^{-},z)}{\Delta\ln k^2}. \tag{12}$$

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

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

The results of this approximation are shown in Figure 1. We compare the non-linear 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. Our fiducial choice for $\Delta\ln k$ is 10^{-4}, beyond which the case with `K_MAX_SPLINE`$=$ 50/Mpc seems to have converged. Our results show that the approximation is good to within 5% up to $k = 10^4$/Mpc if `K_MAX_SPLINE`$=$ 50/Mpc; and to within 0.1% up to $k = 10^4$/Mpc if `K_MAX_SPLINE`$=$ 500/Mpc. The lower `K_MAX_SPLINE` is, the faster `CCL` will run. Figure 2 shows similar results at $z = 3$. In this case, `K_MAX_SPLINE`$=$ 500/Mpc is required to maintain good accuracy. To span the approximate redshift range of LSST, we thus settle on a fiducial value of `K_MAX_SPLINE`$=$ 500/Mpc. The optimum choice of `K_MAX_SPLINE` is left to the user for their particular application. If needed, the $\Delta\ln k$ parameter chan be changed within `ccl_power.c`.

3.4.4. 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`$=$ $10^4/Mpc$. The result is shown in Figure 3. 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`.

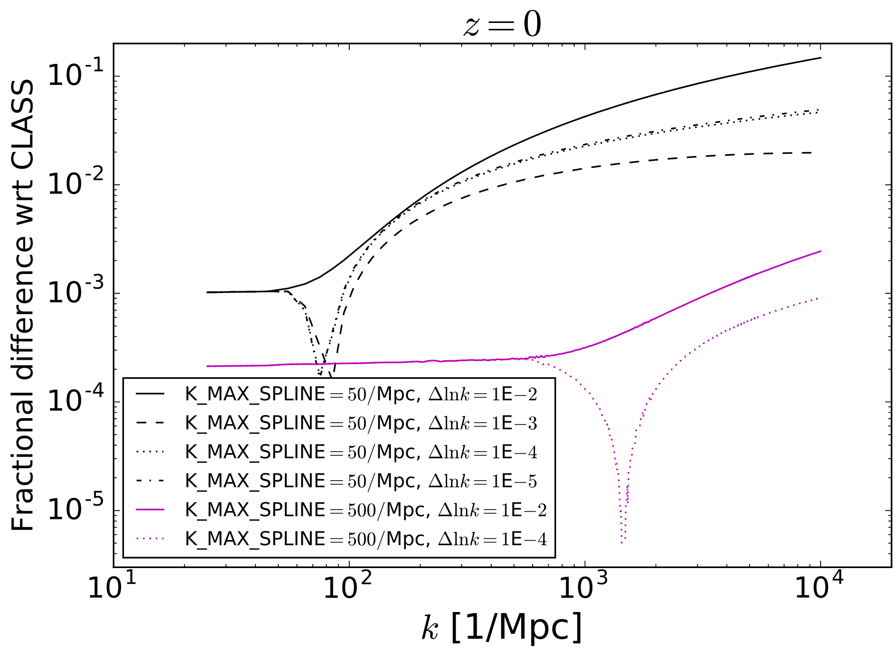

Figure 1. The relative error produced by splining the nonlinear matter power spectrum up to K_MAX_SPLINE and extrapolating beyond this value with a second order Taylor expansion the natural logarithm of the matter power spectrum.

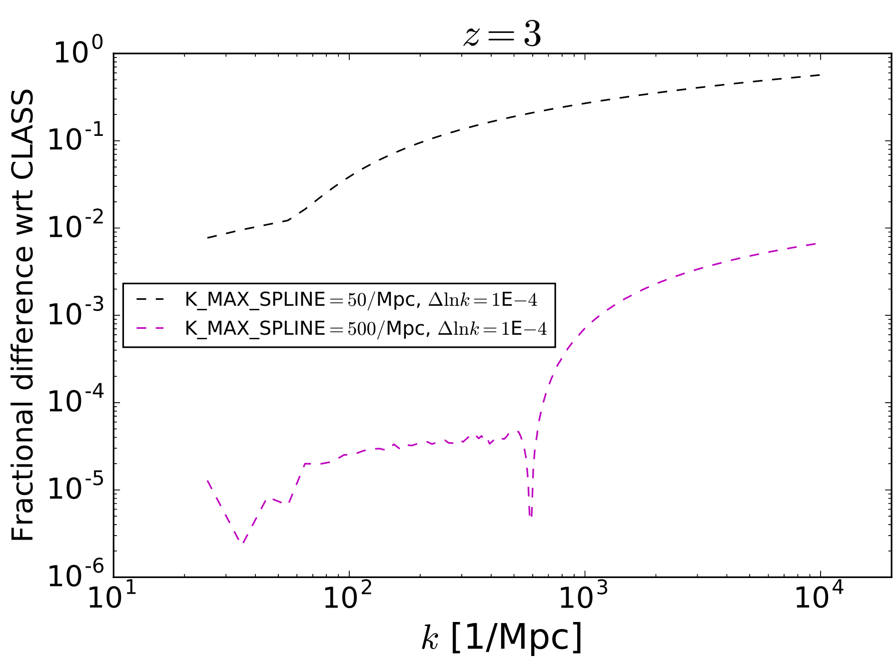

Figure 2. Similar to Figure 1 but at $z = 3$.

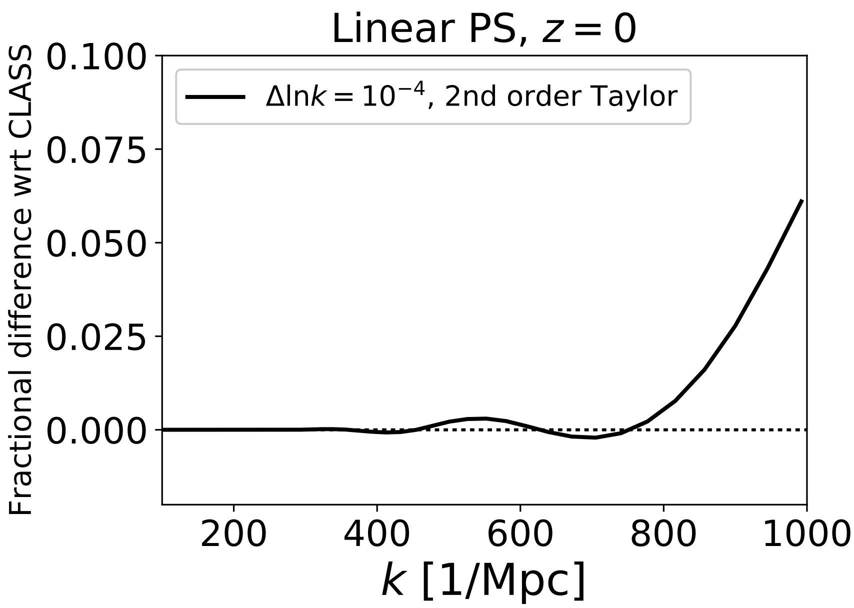

Figure 3. Convergence of the linear matter power spectrum by adopting K_MAX_SPLINE$=500/Mpc$ compared to the case with K_MAX_SPLINE$=10^4/Mpc$.

3.4.5. **Wishlist for the future**

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

- Eisenstein & Hu approximation,
- CAMB,
- Cosmic emulators,
- halo model/HOD.

3.4.6. **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 constrast averaged over spheres of radius $8h^{-1}$Mpc. The computation of σ_8 is described in Section 3.6

3.5. **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`.

3.5.1. **Exact expressions**

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

$$C_\ell^{ab} = 4\pi \int_0^\infty \frac{dk}{k}\, \mathcal{P}_\Phi(k) \Delta_\ell^a(k) \Delta_\ell^b(k), \tag{14}$$

where $\mathcal{P}_\Phi(k)$ is the dimensionless power spectrum of 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:

  $$\Delta_\ell^{\rm D}(k) = \int dz\, p_z(z)\, b(z)\, T_\delta(k,z)\, j_\ell(k\chi(z)), \tag{15}$$

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

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

  $$\Delta_\ell^{\rm RSD}(k) = \int dz\, p_z(z) \frac{(1+z)p_z(z)}{H(z)} T_\theta(k,z) j_\ell''(k\chi(z)), \tag{16}$$

  where $T_\theta(k,z)$ is the transfer function of θ, the divergence of the comoving velocity field.

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

$$\Delta^{\rm M}_\ell(k)=-\ell(\ell+1)\int\frac{dz}{H(z)}W^{\rm M}(z)T_{\phi+\psi}(k,z)j_\ell(k\chi(z)),\tag{17}$$

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

$$W^{\rm M}(z)\equiv\int_z^\infty dz'p_z(z')\frac{2-5s(z')}{2}\frac{r(\chi(z')-\chi(z))}{r(\chi(z'))}.\tag{18}$$

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:

$$\Delta^{\rm L}_\ell(k)=-\frac{1}{2}\sqrt{\frac{(\ell+2)!}{(\ell-2)!}}\int\frac{dz}{H(z)}W^{\rm L}(z)T_{\phi+\psi}(k,z)j_\ell(k\chi(z)),\tag{19}$$

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

$$W^L(z)\equiv\int_z^\infty dz'p_z(z')\frac{r(\chi(z')-\chi(z))}{r(\chi(z'))}.\tag{20}$$

- $\Delta^{\rm IA}$ 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).

$$\Delta^{\rm IA}_\ell(k)=\sqrt{\frac{(\ell+2)!}{(\ell-2)!}}\int dz\,p_z(z)\,b_{\rm IA}(z)\,f_{\rm red}(z)\,T_\delta(k,z)\,\frac{j_\ell(k\chi(z))}{(k\chi(z))^2}.\tag{21}$$

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.

3.5.2. 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:

$$j_\ell(x) \simeq \sqrt{\frac{\pi}{2\ell+1}}\,\delta\left(\ell+\frac{1}{2}-x\right). \tag{22}$$

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

$$C_\ell^{ab} = \frac{2}{2\ell+1}\int_0^\infty dk\,P_\delta\left(k,z_\ell\right)\tilde{\Delta}_\ell^a(k)\tilde{\Delta}_\ell^b(k). \tag{23}$$

where

$$\tilde{\Delta}_\ell^{\rm D}(k) = p_z(z_\ell)\,b(z_\ell)\,H(z_\ell) \tag{24}$$

$$\tilde{\Delta}_\ell^{\rm RSD}(k) = \frac{1+8\ell}{(2\ell+1)^2}\,p_z(z_\ell)\,f(z_\ell)\,H(z_\ell)- \tag{25}$$

$$\frac{4}{2\ell+3}\sqrt{\frac{2\ell+1}{2\ell+3}}p_z(z_{\ell+1})\,f(z_{\ell+1})\,H(z_{\ell+1}) \tag{26}$$

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

$$\tilde{\Delta}_\ell^{\rm L}(k) = \frac{3}{2}\Omega_{M,0}H_0^2\sqrt{\frac{(\ell+2)!}{(\ell-2)}}\frac{1}{k^2}\,\frac{1+z_\ell}{\chi_\ell}W^{\rm L}(z_\ell) \tag{28}$$

$$\tilde{\Delta}_\ell^{\rm IA}(k) = \sqrt{\frac{(\ell+2)!}{(\ell-2)!}}\frac{p_z(z_\ell)\,b_{\rm IA}(z_\ell)f_{\rm red}(z_\ell)H(z_\ell)}{(\ell+1/2)^2} \tag{29}$$

3.6. 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 $\Delta = 200$, where a halo is defined with:

$$\bar{\rho}(r_\Delta) = \Delta * \rho_m. \tag{30}$$

We look toward extending to more general halo definitions in the future, though this implementation is not yet in practice.

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:

$$f(\sigma) = A\left[\left(\frac{\sigma}{b}\right)^{-a} + 1\right]e^{-c/\sigma^2}, \tag{31}$$

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

$$f(\sigma) = A\left[\left(\frac{b}{\sigma} + 1\right)^{-a}\right]e^{-c/\sigma^2}, \tag{32}$$

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 $z = 0$:

$$M = \frac{H_0^2}{2G}R^3 \rightarrow \frac{M}{M_\odot} = 1.162 \times 10^{12}\Omega_M h^2 \left(\frac{R}{1\,\mathrm{Mpc}}\right)^3. \tag{33}$$

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

$$\sigma_R^2 = \frac{1}{2\pi^2}\int dk\, k^2\, P_k\, \tilde{W}_R^2(k) \tag{34}$$

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

$$\tilde{W}_R(k) = \frac{3}{(kR)^3}[\sin(kR) - kR\cos(kR)] \tag{35}$$

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

$$\underset{\sim}{(}\nu) = 1 - A\frac{\nu^a}{\nu^a + {\delta_c}^a} + B\nu^b + C\nu^c, f(\nu) = \alpha[1 + (\beta\nu)^{-2\phi}]\nu^{2\eta}e(-\gamma\nu^2/2). \quad (36)$$

The currently implemented model in CCL does not account for changes in halo definition, though this remains an area of active work to improve upon.

3.7. Photo-z implementation

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_{ph} for an object with hypothetical spectroscopic redshift z_{s}. CCL allows you to flexibly provide your own photometric redshift model.

To do so, you will write a function which accepts as input a photometric redshift, a spectroscopic redshift, and a void pointer to a structure containing any further parameters of your 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){...}
```

You must then also provides the structure of further parameters (`user_par`). This model can be incorporated when computing $\frac{dN^i}{dz}$ in photometric redshift bin i, as given by equation 39, below.

3.8. 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 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:

$$\frac{dN}{dz} \propto z^{\alpha} \exp\left(-\frac{z}{z_0}^{\beta}\right). \tag{37}$$

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):

$$\frac{dN}{dz} \propto \frac{1}{2z_0}\left(\frac{z}{z_0}\right)^2 \exp\left(-\frac{z}{z_0}\right) \tag{38}$$

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 3.7. 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:

$$\frac{dN^i}{dz} = \frac{\frac{dN}{dz}\int_{z_i}^{z_{i+1}} dz' p(z, z')}{\int_{z_{\rm min}}^{z_{\rm max}} dz \frac{dN}{dz} \int_{z_i}^{z_{i+1}} dz' p(z, z')} \tag{39}$$

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):

$$b(z) = \frac{0.95}{D(z)} \tag{40}$$

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

4. 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,
3. linear matter power spectrum, $P(k)$, from BBKS Bardeen et al. 1986) in units of $(\text{Mpc}/h)^3$ 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 $M = \{10^6, 10^8, 10^{10}, 10^{12}, 10^{14}, 10^{16}\}\text{M}_\odot/h$.

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.

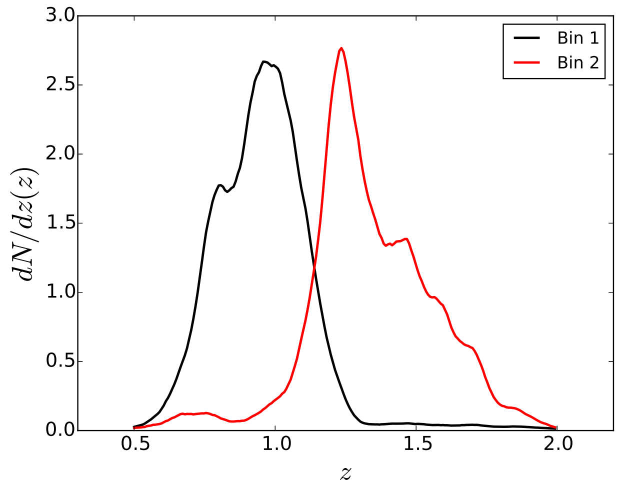

Figure 4. Binned redshift distributions used for code comparison project.

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 4.

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 $l_{\rm max}$, 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 $l_{\rm min}$ (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),

$$\log[(M^2/\bar{\rho}_m)dn/dM]. \tag{41}$$

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.

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`.

5. 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.

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 galaxy lensing.

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 functions 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` (with OpenMP enabled), and that you have a working Python 2.x installation with `numpy` and `distutils` with `swig`. To build and install the `pyccl` module, go to the root `CCL` directory and choose 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.

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.

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 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
```

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:

- correlation functions (linking to FFTlog),
- the non-Limber capability,
- a link to FAST-PT (McEwen et al. 2016) for implementation of perturbation theory,
- a photo-z model,
- support for cosmologies with neutrinos,

- and more power spectrum methods (see 3.4.5).

9. 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.

10. 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.

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

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

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

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

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