NaMaster: API documentation

February 16, 2017

1 Introduction

NaMaster is a C library, Python module and standalone program to compute full-sky angular cross-power spectra of masked, spin-0 and spin-2 fields with an arbitrary number of known contaminants using a pseudo-C_ℓ (aka MASTER) approach.

1.1 Dependencies

The following packages need to be installed before NaMaster.

- **GSL**: the GNU Scientific Library. This should be available in your usual software repositories (e.g. synaptic for linux), but you can also download and install it from `https://www.gnu.org/software/gsl/` (the installation follows the usual `./configure`, `make`, `make install` procedure.
- **HEALPix**: the C HEALPix subroutines are also needed. HEALPix can be downloaded from `http://healpix.jpl.nasa.gov/`, and the C library can be compiled following the instructions after typing `./configure` and then `make c-all`. The header and library files will then be placed in two local folders called `include` and `lib`. The user should then feel free to move these files to a different location.
- **CFITSIO**: a FITS file subroutine library. This is used to read/write HEALPix maps in FITS format. `cfitsio` can be downloaded from `http://heasarc.gsfc.nasa.gov/fitsio/fitsio.html`.
- **Libsharp**: a fast C library for spherical harmonic transforms. Libsharp can be downloaded from its github repository: `https://github.com/dagss/libsharp`. After cloning it, you should run `autoreconf -i` to generate the configuration file. Then run `./configure` and `make`, which will compile the library and place all compilation products in the folder `./auto`. The corresponding header and library files should then be manually moved to the desired installation directories.

1.2 Compilation and usage

Installing the C library and binaries

NaMaster uses `autotools` for installation, which means that you should be able to install it by simply typing

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

If you don't have default admin privileges you may have to precede the last command by `sudo`. If you don't have admin privileges at all (i.e. you can't modify the contents of standard directories such as `/usr/lib`), you can still install NaMaster by substituting the first command by `./configure --prefix=/path/to/dir`, where `/path/to/dir` should be the full path of the directory where you want to install this package. This will create three sub-directories: `/path/to/dir/include`, `/path/to/dir/lib`, `/path/to/dir/bin`, where the header, library and binary files will be placed respectively.

Note that, if you don't have admin privileges, probably some of the dependencies listed in the previous sections will also be installed in non-standard paths. If that is the case, you should make sure the

environment variables `CPPFLAGS` and `LDFLAGS` contain the corresponding `-I/path/to/dir2/include` and `-L/path/to/dir2/lib` tags that point to the directories where these dependencies are installed (see this link for more details on `make` implicit variables).

Installing the python wrapper

NaMaster comes equipped with a python wrapper. This is installed by running

```
~$ python setup.py install
```

Without admin privileges you can still make this work by running

```
~$ python setup.py install --user
```

The python wrapper needs to link with NaMaster's library. If the latter was installed in a non-standard path (e.g. `/path/to/install`), you'll need to pass the corresponding directory to `setup.py`. You can do so by running

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

before the install commands above.

Note that currently the python wrapper requires the user to have the SWIG package installed (this will be changed in the future). SWIG can be found in the standard software repositories and at `http://www.swig.org/`.

Linking with the C library

If you want to use NaMaster on your own C code you'll need to be able to link with `libnmt` (the NaMaster C library). There are two main things to do:

1. Make sure to include the NaMaster header in any C file that makes use of any of the NaMaster subroutines:

   ```
   #include <namaster.h>
   ```

2. When compiling your code, make sure you link to `libnmt` and all dependencies. In the simplest case, assuming you have written a C script called `min_code.c`, the following should work:

   ```
   gcc -fopenm -I/path/to/nmt/include min_code.c -o min_code
       -L/path/to/nmt/lib -lnmt -lsharp -lfftpack -lc_utils
       -lchealpix -lcfitsio -lgsl -lgslcblas -lm
   ```

   where `/path/to/nmt/include` and `/path/to/nmt/lib` are the directories where `namaster.h` and `libnmt.so` are installed.

Section 6 below contains a fully working C script that calls the NaMaster library.

2 C documentation

Important note: all HEALPix maps passed to NaMaster routines should be in RING order.

2.1 Fields

The definition of the fields to be correlated (including their masks and possible contaminants) is handled through a C structure called `nmt_field`. The following routines exist to manage this structure:

nmt_field_alloc

```
nmt_field * nmt_field_alloc(long nside,double *mask,int pol,double **maps,
int ntemp,flouble ***temp)
```

This is the constructor for `nmt_field`. The input variables are:

- `nside`: the HEALPix resolution of all maps involved
- `mask`: sky mask (as a single scalar HEALPix map).
- `pol`: set to 0 if this is a spin-0 field. Set to 1 if it's a spin-2 field.
- `maps`: set of maps corresponding to observed field. This would correspond to one map for a scalar field or two maps for spin-2 quantities (e.g. Q and U for polarization maps or γ_1, γ_2 for cosmic shear). The first dimension of this double array would correspond to the number of maps, while the second dimension runs through the different pixels of each map.
- `ntemp`: number of contaminant templates for this field.
- `temp`: contaminant templates as HEALPix maps. The first dimension should run through the different templates, the second dimension corresponds to the number of maps per template (e.g. 1 for spin-0 and 2 for spin-2) and the third dimension corresponds to the number of pixels.

nmt_field_read

```
nmt_field * nmt_field_read(char *fname_mask,char *fname_maps,char *fname_temp,int pol)
```

As `nmt_field_alloc`, this returns a pointer to a `nmt_field` structure based on:

- `pol`: set to 0 if this is a spin-0 field. Set to 1 if it's a spin-2 field.
- `fname_mask`: file name pointing to a FITS file containing the sky mask (as a single scalar HEALPix map).
- `fname_maps`: file name pointing to a FITS file containing the maps of the observed field. This file should contain a single map for `pol`=0 and two maps for `pol`=1.
- `fname_temp`: file name pointing to a FITS file containing the contaminant templates as HEALPix maps. Each template should contain N maps with $N = 1$ for `pol`=0 and $N = 2$ for `pol`=1.

nmt_field_free

```
void nmt_field_free(nmt_field *fl)
```

This frees up all memory associated to a previously-allocated `nmt_field`.

2.2 Binning scheme

The definition of bandpowers is managed through C structures called `nmt_binning_scheme`. The following routines allow you to interact with this structure:

nmt_bins_constant

```
nmt_binning_scheme * nmt_bins_constant(int nlb,int lmax)
```

Creates an `nmt_binning_scheme` structure (and returns a pointer to it) where the bandpowers are constant intervals of `nlb` multipoles with equal weights between $\ell = 2$ and $\ell =$ `lmax`.

nmt_bins_create

```
nmt_binning_scheme * nmt_bins_create(int nell,int *bpws,int *ells,
double *weights,int lmax)
```

Creates an `nmt_binning_scheme` structure (and returns a pointer to it) with bandpowers defined by the following parameters:

- `ells`: array of multipole indices
- `bpws`: array containing the band power each ℓ in `ells` corresponds to.
- `weights`: array containing the weight for each ℓ in `ells`. These need not be normalized, but they will be normalized such that the sum of weights within each bandpower equals 1.
- `nell`: number of elements in the three previous arrays.
- `lmax`: all multipoles $\ell >$ `lmax` will be ignored.

nmt_bins_read

```
nmt_binning_scheme * nmt_bins_read(char *fname,int lmax)
```

Creates an `nmt_binning_scheme` structure (and returns a pointer to it) with bandpowers defined by the contents of an ASCII file with name `fname`. This file should contain three columns corresponding to the arrays `bpws`, `ells` and `weights` passed to `nmt_bins_create`. All multipoles $\ell >$ `lmax` will be ignored.

nmt_bins_free

```
void nmt_bins_free(nmt_binning_scheme *bin)
```

Frees all memory associated with an allocated `nmt_binning_scheme` structure.

nmt_bin_cls

```
void nmt_bin_cls(nmt_binning_scheme *bin,double **cls_in,double **cls_out,int ncls)
```

Performs a binning operation:

$$B_k = \sum_{\ell\in\vec{\ell}_k} w_\ell C_\ell. \tag{1}$$

Here, $C_\ell \to$ `cls_in` is a set of `ncls` angular power spectra, $B_k \to$ `cls_out` is a set of bandpowers and w_ℓ and $\vec{\ell}_k$ are the weights and multipole ranges defining the binning scheme `bin`. Both `cls_in` and `cls_out` should have been previously allocated. The first dimension of both `cls_in` and `cls_out` should run from 0 to `ncls` $-$ 1. Their second dimension should correspond to the number of multipoles and bandpowers used to create `bin` respectively (for instance, the latter can be accessed as `bin->n_bands`).

nmt_unbin_cls

```
void nmt_unbin_cls(nmt_binning_scheme *bin,double **cls_in,double **cls_out,int ncls)
```

Performs a un-binning operation:

$$C_\ell = \sum_k B_k \Theta(\ell \in \vec{\ell}_k) \tag{2}$$

Here, $B_k \to$ `cls_in` is a set of `ncls` bandpowers, $C_\ell \to$ `cls_out` is a set of angular power spectra and w_ℓ and $\vec{\ell}_k$ are the weights and multipole ranges defining the binning scheme `bin`. The function $\Theta(\ell \in \ell_k)$

is 1 for all multipoles contained in the k-th bandpower and zero otherwise. Both `cls_in` and `cls_out` should have been previously allocated. The first dimension of both `cls_in` and `cls_out` should run from 0 to `ncls` − 1. Their second dimension should correspond to the number of bandpowers and multipoles used to create `bin` respectively (for instance, the former can be accessed as `bin->n_bands`).

nmt_ell_eff

```
void nmt_ell_eff(nmt_binning_scheme *bin,double *larr)
```

This function returns, in the output array `larr`, the effective multipole corresponding to each bandpower defined by `bin`. This is computed as:

$$\ell_k^{\rm eff} = \sum_{\ell\in\vec{\ell}_k} \ell\, w_\ell, \tag{3}$$

where w_ℓ are the bandpower weights. `larr` should have been previously allocated to the number of bandpowers defined by `bin`.

2.3 Pseudo-C_ℓ

The implementation of the pseudo-C_ℓ estimator can be split into the following steps:

1. Clean up your best guess of the known contaminants in your data maps. This step can be skipped if you think your maps are clean of contaminants. This step is automatically carried out when initializing an `nmt_field` structure with template contaminants.
2. Compute the cross-pseudo-C_ℓ of the cleaned maps $\tilde{C}_\ell^{\rm clean}$.
3. Compute the bias on the pseudo-C_ℓ caused by the statistical residual contaminants $\tilde{C}_\ell^{\rm cont}$.
4. Compute the mode-coupling matrix associated with the field masks $\mathsf{M}_{\ell\ell'}$.
5. Compute the de-coupled and de-biased bandpowers:

$$B_k = \sum_{k'} (\mathcal{M})^{-1}_{kk'} \sum_{\ell\in\vec{\ell}_{k'}} w_{\ell'} \left[\tilde{C}_{\ell'}^{\rm clean} - \tilde{C}_{\ell'}^{\rm bias} - \tilde{C}_{\ell'}^{\rm noise}\right], \tag{4}$$

   where $\mathcal{M}$ is the binned coupling matrix:

$$\mathcal{M}_{kk'} \equiv \sum_{\ell\in\vec{\ell}_k} \sum_{\ell'\in\vec{\ell}_{k'}} w_\ell \mathsf{M}_{\ell\ell'}. \tag{5}$$

   When auto-correlating a field with noise, it is in general also desirable to remove the noise bias on the power spectrum. This has been included in Eq. 4 above as $\tilde{C}_\ell^{\rm noise}$. This should be the pseudo-C_ℓ of the noise component (i.e. the angular power spectrum of masked noise realizations), which can be computed from Monte-Carlo simulations (or analytically for sufficiently simple noise models).

In NaMaster, these computations are carried out through a C structure called `nmt_workspace`. The relevant functions are described below.

`nmt_apodize_mask`

3 Macros

The following preprocessor macros, defined in the header file, are used by CosmoMad and may be used by any code linked to it:

- `CSM_FOURPITHIRD` : $\frac{4\pi}{3}$

- CSM_TWOPIPIINV : $\frac{1}{2\pi^2}$
- CSM_TWOPIPIINVLOGTEN : $\frac{\ln(10)}{2\pi^2}$
- CSM_LOGTEN : $\ln(10)$
- CSM_RTOD : $\frac{180}{\pi}$
- CSM_DTOR : $\frac{\pi}{180}$
- CSM_HGYR : H_0^{-1} in units of Gyr/h
- CSM_HMPC : $c\,H_0^{-1}$ in units of Mpc/h

4 Csm_params

In its current version (> 0.5), CosmoMad defines the structure `Csm_params`, which contains all the necessary information to calculate all the supported quantities for a given cosmological model. It is not our intention to describe here the elements of this structure, since the user is not supposed to meddle with it. However, its definition and those of all related structures are given in the header file `cosmo_mad.h`. Most of the functions described below accept a `Csm_params` struct as their first argument, which defines the cosmological model for which the calculation must be done (once the model has been initialized). This prevents the use of global variables and allows the user to compute the same quantity in different cosmological models simultaneously.

A `Csm_params` structure contains all the information about the background cosmological parameters, power-spectrum and 2-point correlation function information.

5 Routines

All the functions provided by CosmoMad start with the prefix `csm_`.

5.1 General behavior

csm_unset_gsl_eh

```
void csm_unset_gsl_eh(void)
```

A call to this function disables the default GSL error handler. This error handler is very strict and will exit the program if any problem (regarding, for example, the accuracy of an integral) is met. Since sometimes these problems are not so important (an integral reaching a $10^{-3}\%$ accuracy instead of $10^{-4}\%$ may not be problematic), you may want the program to continue its execution anyway. When called, a taylored error-handler will be used for the current run. This error handler will output error messages to `stderr` beginning with "CosmoMad: ", giving a hint as to what the encountered problem was, and it will exit the program if the error found is clearly important (like finding a NaN). It is recomended to call this function at the beginning of any program using CosmoMad.

csm_set_verbosity

```
void csm_set_verbosity(int verb)
```

Determines the amount of information output. In the current version there are only two levels, 0 (nothing) and 1 (everything). The default level of verbosity is 1 (all messages are output).

csm_params_new

```
Csm_params *csm_params_new(void)
```

Returns an initialized Csm_params structure. Notice that this returns an empty structure, with no associated cosmological information.

csm_params_free

```
void csm_params_free(Csm_params *pars)
```

Frees up all the memory associated with a `Csm_params` structure.

5.2 Mathematical functions

These functions return the result given by the analogous `GSL` routines and are only provided for convenience.

csm_p_leg

```
double csm_p_leg(int l,double x)
```

Returns the `l`-th Legendre polynomial evaluated at `x`: $L_l(x)$.

csm_j_bessel

```
double csm_j_bessel(int l,double x)
```

Returns the `l`-th spherical Bessel function evaluated at `x`: $j_l(x)$.

5.3 Background evolution

csm_background_set

```
void csm_background_set(Csm_params *pars,double OM,double OL,double OB,double w0,double
wa,double hh,double T_CMB)
```

Sets the background cosmology for the structure `pars`: $\Omega_M =$ `OM`, $\Omega_{DE} =$ `OL`, $\Omega_b =$ `OB`, $h =$ `hh`, $w_0 =$ `w0`, $w_a =$ `wa`, $T_{\rm CMB} =$ `TCMB`, with the CMB temperature given in Kelvin. This function must be called for any `Csm_params` used.

csm_cosmic_time

```
double csm_cosmic_time(Csm_params *pars,double aa)
```

Returns the cosmic time corresponding to the scale factor `aa` by calculating the integral

$$t(a) = \int_0^a \frac{da'}{a'\,H(a')} = H_0^{-1} \int_0^a \left(\frac{x}{\Omega_M + \Omega_k\,x + \Omega_{DE}\,x^{-3w}} \right)^{1/2} dx \tag{6}$$

csm_scale_factor

```
double csm_scale_factor(Csm_params *pars,double t)
```

For cosmic time `t` in Gyr/h, this function returns the value of the scale factor. The first time this function is called, the integral (6) is used for several values of a from 0 to 1 and a spline object is created to calculate $a(t)$ faster in all subsequent calls.

csm_hubble

```
double csm_hubble(Csm_params *pars,double aa)
```

Returns the inverse Hubble horizon $H(a)$ at $a =$ `aa` in inverse length units.

csm_omega_matter

```
double csm_omega_matter(Csm_params *pars,double aa)
```

Returns the matter parameter $\Omega_M(a)$ at $a =$ `aa`.

csm_particle_horizon

```
double csm_particle_horizon(Csm_params *pars,double aa)
```

Returns the comoving particle horizon (the maximum distance a particle can have travelled since $a = 0$) at $a =$ `aa` by calculating the integral

$$\chi_p(a) = c \int_0^a \frac{da'}{a'^2\,H(a')} = \frac{c}{H_0} \int_0^a \frac{dx}{x\sqrt{\Omega_M + \Omega_k\,x + \Omega_{DE}\,x^{-3w}}}$$

csm_radial_comoving_distance

```
double csm_radial_comoving_distance(Csm_params *pars,double aa)
```

Returns the radial comoving distance $\chi(a) = \chi_p(1) - \chi_p(a)$ for $a =$ `aa`.

csm_curvature_comoving_distance

```
double csm_curvature_comoving_distance(Csm_params *pars,double aa)
```

Returns the curvature comoving distance at $a =$ `aa`

$$r(a) = \frac{c}{H_0\sqrt{|\Omega_k|}}\,\mathrm{sinn}(H_0\sqrt{|\Omega_k|}\,\chi(a)/c)$$

csm_angular_diameter_distance

```
double csm_angular_diameter_distance(Csm_params *pars,double aa)
```

Returns the angular diameter distance at $a =$ `aa`

$$d_A(a) = a\, r(a)$$

csm_luminosity_distance

```
double csm_luminosity_distance(Csm_params *pars,double aa)
```

Returns the luminosity distance at $a =$ `aa`

$$d_L(a) = \frac{r(a)}{a}$$

csm_growth_factor_and_growth_rate

```
void csm_growth_factor_and_growth_rate(Csm_params *pars,double aa,double *gf,double *fg)
```

Returns the growth factor $D(a)$ and the growth rate $f(a)$ at $a =$ `aa` in the variables `gf` and `fg` respectively. If both quantities are required at the same time it is more efficient to call this function than the two functions below, since both quantities are obtained at the same time when solving the differential equation for the growth of matter perturbations:

$$\frac{d}{da}\left(a^3\, H(a)\frac{dD}{da}\right) = \frac{3}{2}\Omega_M(a)\, H(a)\, a\, D \tag{7}$$

Note that $D(a)$ is normalized to $D(a \to 0) \to a$, and not $D(1) = 1$.

csm_growth_factor

```
double csm_growth_factor(Csm_params *pars,double aa)
```

Returns the growth factor at $a =$ `aa`.

csm_f_growth

```
double csm_f_growth(Csm_params *pars,double aa)
```

Returns the growth rate $f(a)$ at $a =$ `aa`.

csm_theta_BAO

```
double csm_theta_BAO(Csm_params *pars,double aa)
```

Returns the angular position (in degrees) of the BAO peak in the angular correlation function at $a =$ `aa`:

$$\theta_{BAO}(a) = \frac{a\, r_s}{d_A(a)}$$

csm_Dz_BAO

```
double csm_Dz_BAO(Csm_params *pars,double aa)
```

Returns the position (in Δz) of the BAO peak in the radial correlation function at $a =$ `aa`:

$$\Delta z_{BAO}(a) = \frac{H(a)\, r_s}{c}$$

5.4 Power spectrum

csm_set_linear_pk

```
void csm_set_linear_pk(Csm_params *pars,char *fname,double lkmn,double lkmx,
double dlk,double nns,double s8)
```

This function sets the linear matter power spectrum at $a = 1$. There exist several options:

- If `fname` is "BBKS" the power spectrum will be calculated from the BBKS transfer function ([1]) in the interval `lkmn` $< \log_{10}(k) <$ `lkmx`, in intervals of $\Delta \log_{10}(k) =$ `dlk`.
- If `fname` is "EH" the power spectrum will be calculated from the Eisenstein & Hu transfer function [3] in the same fashion.
- If `fname` is "EH_smooth" the power spectrum will be calculated from the Eisenstein & Hu transfer function **without acoustic oscillations**.
- Finally `fname` can be set to the path to a file containing the power spectrum. This file must be in CAMB format, i.e.: two columns (k, $P(k)$) with k in h/Mpc and its values evenly spaced in $\log_{10}(k)$.

Once the $P(k)$ is read (or calculated) it is normalized to $\sigma_8 =$ `s8`. After that a spline object is created for faster interpolation thereafter. The normalization for $P(k)$ used here is such that

$$\langle \delta(\mathbf{x})\delta(\mathbf{x}+\mathbf{r})\rangle \equiv \xi(r) = \frac{1}{2\pi^2}\int_0^\infty P(k)\,\frac{\sin(kr)}{kr}k^2 dk$$

csm_set_nonlinear_pk

```
void csm_set_nonlinear_pk(Csm_params *pars,char *fnamePkHFIT)
```

This function sets the non-linear matter power spectrum at $a = 1$. Three options are available: if `fnamePkHFIT` is set to "RPT" the mildly non-linear power spectrum is approximated by including a Gaussian damping term arising in renormalized perturbation theory ([2]):

$$P(k,z) = P^L(k,z)\,e^{-k^2\sigma_v^2(z)},$$

where

$$\sigma_v^2(z) = \frac{1}{6\pi^2}\int_0^\infty P^L(k,z)dk.$$

Thus in this case $\sigma_v^2(z=0)$ is calculated and used in this way when calling `csm_Pk_nonlinear_0` (below).

If `fnamePkHFIT` is set to "RPT_ss" this Gaussian damping factor is also used, however the small scales are recovered by adding a no-BAO power spectrum:

$$P(k,z) = [P^L(k,z) - P^L_{\rm noBAO}(k,z)]\,e^{-k^2\sigma_v^2(z)} + P^L_{\rm noBAO}(k,z).$$

The no-BAO $P(k)$ is obtained using the Eisenstein & Hu [3] fitting formula without acoustic oscillations. This way only the BAO wiggles are damped.

The last option is to set `fnamePkHFIT` to the path to a file containing a non-linear power spectrum (for example using HALOFIT [8]). The format for this file must be the same as the one used in `csm_set_linear_pk`. Note that in this case there is no way to normalize $P(k)$ to the value of σ_8 used for the linear power-spectrum, but CosmoMad will use the same normalization factor used for the linear case, so one should make sure that both the linear and non-linear $P(k)$'s were generated with the same normalization.

csm_Pk_linear_0

```
double csm_Pk_linear_0(Csm_params *pars,double kk)
```

Returns the linear matter power spectrum at $a = 1$ and $k =$ `kk`. If `kk` is larger than the interpolation limits for $P(k)$ it is approximated by $P(k) \propto k^{n_s}$ for small k and $P(k) \propto k^3$ for large k.

csm_Pk_nonlinear

```
double csm_Pk_nonlinear(Csm_params *pars,double kk)
```

Returns the non-linear power spectrum at $k =$ `kk`. If `kk` is larger than the interpolation limits for $P(k)$ it is approximated by $P(k) \propto k^{n_s}$ for small k and $P(k) \propto k^3$ for large k. This function returns the power spectrum normalized with the growth factor given in `csm_set_Pk_params`, but without bias or RSDs.

csm_set_Pk_params

```
void csm_set_Pk_params(Csm_params *pars,double beta,double gf, double bias,int l_max)
```

Sets the parameters necessary to calculate the full power spectrum in redshift space: $\beta(a) =$ `beta`, $D(a) =$ `gf` and $b =$ `bias` (see equation (8)). `l_max` is the maximum multipole that will be used in the calculation of the power spectru and 3D correlation funciton (e.g. 4 for the Kaiser approximation or 0 for the real-space case – $\beta = 0$).

csm_Pk_full

```
double csm_Pk_full(Csm_params *pars,double kk,double muk)
```

Returns the full redshift-space power spectrum in the Kaiser approximation ([4]):

$$P_s(a,k,\mu_k) = b^2\,(1+\beta(a)\,\mu_k^2)^2\,P_r(a,k), \tag{8}$$

with

$$P_r(a,k) = [D(a)]^2\,P_{NL}(a,k),$$

and

$$P_{NL}(a,k) \equiv P_L(a=1,k)\exp\left(-[D(a)\,\sigma_v(0)]^2k^2\right)$$

if RPT was used to set the non-linear power spectrum.

csm_Pk_multipole

```
double csm_Pk_multipole(Csm_params *pars,double kk,int l)
```

Returns the `l`-th multipole of the power spectrum:

$$P_l(k) = \frac{2l+1}{2}\int_{-1}^{1} L_l(\mu_k)\,P(k,\mu_k),$$

where $L_l(x)$ is the l-th Legendre polynomial.

5.5 Correlation functions

csm_xi2p_L

```
double csm_xi2p_L(Csm_params *pars,double r,double R1,double R2,
char *wf1,char *wf2,double errfac)
```

Let $\delta(\mathbf{x}; R, T)$ be the density contrast smoothed with a window function of type T and smoothing scale R. This function returns the value of the correlation function between $\delta(\mathbf{x}; \mathtt{R1}, \mathtt{wf1})$ and $\delta(\mathbf{x} + \mathbf{r}; \mathtt{R2}, \mathtt{wf2})$. To be more specific, the return value is

$$\xi(r; R_1, R_2) \equiv \frac{1}{2\pi^2} \int_0^\infty P_L(k, z=0)\, W_{T_1}(kR_1)\, W_{T_2}(kR_2)\, j_0(kr)\, k^2 dk$$

The possible values for `wf1` and `wf2` are "TopHat" and "Gauss":

$$W_{TH}(x) = 3\,\frac{\sin x - x\cos x}{x^3}, \quad W_G(x) = \exp(-x^2/2).$$

For some values of the parameters it may be impossible for the GSL integrator to obtain the required accuracy, in which case the error requirement can be altered through `errfac`: the relative error will then be `errfac` 10^{-4} (the recommended value for `errfac` is thus 1).

csm_sig0_L

```
double csm_sig0_L(Csm_params *pars,double R1,double R2,char *wf1,char *wf2)
```

With the notation above, this function returns the value of the covariance of between $\delta(\mathbf{x}; \mathtt{R1}, \mathtt{wf1})$ and $\delta(\mathbf{x}; \mathtt{R2}, \mathtt{wf2})$. I.e. this is equivalent to `csm_xi2p_L(0,R1,R2,wf1,wf2,1)`.

csm_xi_multipole

```
double csm_xi_multipole(Csm_params *pars,double rr,int l)
```

Returns the `l`-th multipole of the redshift-space correlation function. This is done by performing the integral

$$\xi_l(r) = \frac{i^l}{2\pi^2} \int_0^\infty P_l(k)\, j_l(kr)\, k^2\, dk, \tag{9}$$

where $P_l(k)$ is the l-th multipole of the redshift-space power spectrum (as returned by `csm_Pk_multipole`). The first time this function is called a spline is created for each power spectrum multipole in order to accelerate the calculation of the integral above.

csm_set_xi_multipole_splines

```
double csm_set_xi_multipole_splines(Csm_params *pars)
```

If the correlation function multipoles must be calculated repeatedly, it may be faster to calculate first the multipoles once for a set of r-values and then interpolate between these values. This function initializes a set of spline objects that are used thereafter when calling `csm_xi_multipole`. Specifically, a logarithmic-spaced spline is used for $0.1\,\mathrm{Mpc}/h < r < 15\,\mathrm{Mpc}/h$, and a linear-spaced spline is used for $15\,\mathrm{Mpc}/h < r < 500\,\mathrm{Mpc}/h$. Hence subsequent calls to this function will not calculate the integral (9), but a much faster interpolation. If this function is called, for $r > 500\,\mathrm{Mpc}/h$, `csm_xi_multipole` will return 0, and for $r < 0.1\,\mathrm{Mpc}/h$ it will return the value at 0.1 Mpc.

csm_unset_xi_multipole_splines

```
double csm_unset_xi_multipole_splines(Csm_params *pars)
```

Frees up all the memory associated to the splines created when calling `csm_set_xi_multipole_splines`. It is not necessary to call this function at the end of each program, since `csm_params_free` will also take care of this.

csm_xi_3D

```
double csm_xi_3D(Csm_params *pars,double rr,double mu)
```

Returns the anisotropic 3-D correlation function $\xi(r,\mu)$ as a sum over multipoles:

$$\xi(r,\mu) = \sum_{l=0}^{\infty} \xi_l(r)\, L_l(\mu).$$

Note that under the Kaiser approximation (the one used in the present version of CosmoMad) only the first three multipoles ($l = 0,\ 2,\ 4$) are used. When many calls to this function are necessary it may be wise to call `csm_set_multipole_splines` first for a better performance.

csm_xi_pi_sigma

```
double csm_xi_pi_sigma(Csm_params *pars,double pi,double sigma,int use_multipoles)
```

Returns the anisotropic 3-D correlation function $\xi(\pi,\sigma)$ using longitudinal ($\pi \equiv \mu$) and transverse ($\sigma \equiv \sqrt{r^2-\pi^2}$) coordinates. If `use_multipoles` is set to 1 the sum over multipoles described above is used. If set to 0 the following double integral is performed:

$$\xi(\pi,\sigma) = \frac{1}{2\pi^2}\int_0^\infty dk_\parallel \cos(k_\parallel\,\pi)\int_0^\infty dk_\perp\, k_\perp\, J_0(k_\perp\,\sigma) P(k_\parallel,k_\perp),$$

where $J_0(x)$ is the 0-th order cylindrical Bessel function. Note that the latter approach, although exact, will be much slower than the former, unless a large number of multipoles is needed.

5.6 Halo mass function

csm_M2R

```
double csm_M2R(Csm_params *pars,double mass)
```

Returns the comoving radius of a sphere of mass `mass` (in units of $M_\odot/h$). These two quantities are related through

$$M = \frac{4\pi}{3}\Omega_M\,(2.776\times10^{11}\,M_\odot/h)\left(\frac{R}{1\,\mathrm{Mpc}/h}\right)^3 \tag{10}$$

csm_R2M

```
double csm_R2M(Csm_params *pars,double radius)
```

Returns the mass of a sphere of comoving radius `radius`.

csm_collapsed_fraction

```
double csm_collapsed_fraction(Csm_params *pars, double mass,char *mf_model)
```

Returns the fraction of the Universe that has collapsed into halos of mass larger than `mass` according to the mass function parametrization given by `mv_model`. Three models are supported:

- "PS", [6]:

$$F_{\rm PS}(<M) = \mathrm{erfc}(\nu/\sqrt{2}) \tag{11}$$

- "JAP", [5]:

$$F_{\rm JAP}(<M) = \frac{\exp(-c\,\nu^2)}{1+a\,\nu^b}, \tag{12}$$

  with $(a,b,c) = (1.529, 0.704, 0.412)$.

- "ST", [7]:

$$F_{\rm ST}(<M) = A\left[\mathrm{erfc}\left(\sqrt{\frac{a}{2}}\nu\right) + \frac{\Gamma(1/2-p, a\,\nu^2/2)}{\sqrt{\pi}\,2^p}\right], \tag{13}$$

  with $(A,a,p) = (0.322, 0.707, 0.3)$.

6 Sample program

Here's a sample code using this library. This code takes a redshift as a command-line argument and calculates several background quantities at that redshift, as well as the power spectrum and correlation functions (which are written into ASCII files):

```
#include "utils.h"

void run_master(nmt_field *fl1,nmt_field *fl2,
     char *fname_cl_noise,
     char *fname_cl_proposal,
     char *fname_coupling,
     char *fname_out,
     char *fname_bins,
     int n_lbin)
{
  FILE *fi;
  int ii;
  int lmax=fl1->lmax;
  int nspec=fl1->nmaps*fl1->nmaps;
  flouble **cl_noise,**cl_proposal,**cl_out,**cl_bias,**cl_data;

  if(fl1->nside!=fl2->nside)
    report_error(1,"Can't correlate fields with different resolution\n");

  //Binning
  nmt_binning_scheme *bin;
  if(!strcmp(fname_bins,"none"))
    bin=nmt_bins_create(n_lbin,fl1->lmax);
  else
    bin=nmt_bins_read(fname_bins,fl1->lmax);

  //Allocate cl
  cl_noise=my_malloc(nspec*sizeof(flouble *));
  cl_proposal=my_malloc(nspec*sizeof(flouble *));
  cl_bias=my_malloc(nspec*sizeof(flouble *));
  cl_data=my_malloc(nspec*sizeof(flouble *));
  cl_out=my_malloc(nspec*sizeof(flouble *));
  for(ii=0;ii<nspec;ii++) {
    cl_noise[ii]=my_calloc((lmax+1),sizeof(flouble));
    cl_proposal[ii]=my_calloc((lmax+1),sizeof(flouble));
    cl_bias[ii]=my_calloc((lmax+1),sizeof(flouble));
    cl_data[ii]=my_calloc((lmax+1),sizeof(flouble));
    cl_out[ii]=my_calloc(bin->n_bands,sizeof(flouble));
  }

  printf("Reading noise pseudo-cl\n");
  if(strcmp(fname_cl_noise,"none")) {
    fi=my_fopen(fname_cl_noise,"r");
    int nlin=my_linecount(fi); rewind(fi);
    if(nlin!=lmax+1)
      report_error(1,"Wrong number of multipoles for noise p.spec.\n");
    for(ii=0;ii<lmax+1;ii++) {
      int status,jj;
      flouble l;
      status=fscanf(fi,"%lf",&l);
      if(status!=1)
  report_error(1,"Error reading file %s\n",fname_cl_noise);
      for(jj=0;jj<nspec;jj++) {
  status=fscanf(fi,"%lf",&(cl_noise[jj][ii]));
  if(status!=1)
    report_error(1,"Error reading file %s\n",fname_cl_noise);
      }
    }
    fclose(fi);
  }

  printf("Reading proposal Cl\n");
  if(strcmp(fname_cl_proposal,"none")) {
    fi=my_fopen(fname_cl_proposal,"r");
```

```
65      int nlin=my_linecount(fi); rewind(fi);
66      if(nlin!=lmax+1)
67        report_error(1,"Wrong number of multipoles for noise p.spec.\n");
68      for(ii=0;ii<lmax+1;ii++) {
69        int status,jj;
70        flouble l;
71        status=fscanf(fi,"%lf",&l);
72        if(status!=1)
73    report_error(1,"Error reading file %s\n",fname_cl_proposal);
74        for(jj=0;jj<nspec;jj++) {
75    status=fscanf(fi,"%lf",&(cl_proposal[jj][ii]));
76    if(status!=1)
77      report_error(1,"Error reading file %s\n",fname_cl_proposal);
78        }
79      }
80      fclose(fi);
81    }
82
83    nmt_workspace *w;
84    if(access(fname_coupling,F_OK)!=-1) { //If file exists just read matrix
85      printf("Reading coupling matrix\n");
86      w=nmt_workspace_read(fname_coupling);
87      if(w->bin->n_bands!=bin->n_bands)
88        report_error(1,"Read coupling matrix doesn't fit input binning scheme\n");
89    }
90    else {
91      printf("Computing coupling matrix \n");
92      w=nmt_compute_coupling_matrix(fl1,fl2,bin);
93      if(strcmp(fname_coupling,"none"))
94        nmt_workspace_write(w,fname_coupling);
95    }
96
97    printf("Computing data pseudo-Cl\n");
98    he_anafast(fl1->maps,fl2->maps,fl1->pol,fl2->pol,cl_data,fl1->nside,fl1->lmax);
99
100   printf("Computing deprojection bias\n");
101   nmt_compute_deprojection_bias(fl1,fl2,cl_proposal,cl_bias);
102
103   printf("Computing decoupled bandpowers\n");
104   nmt_decouple_cl_l(w,cl_data,cl_noise,cl_bias,cl_out);
105
106   printf("Writing output\n");
107   fi=my_fopen(fname_out,"w");
108   for(ii=0;ii<bin->n_bands;ii++) {
109     int jj;
110     double l_here=0;
111     for(jj=0;jj<bin->nell_list[ii];jj++)
112       l_here+=bin->ell_list[ii][jj]*bin->w_list[ii][jj];
113     fprintf(fi,"%.2lf ",l_here);
114     for(jj=0;jj<nspec;jj++)
115       fprintf(fi,"%lE ",cl_out[jj][ii]);
116     fprintf(fi,"\n");
117   }
118   fclose(fi);
119
120   nmt_bins_free(bin);
121   nmt_workspace_free(w);
122   for(ii=0;ii<nspec;ii++) {
123     free(cl_noise[ii]);
124     free(cl_proposal[ii]);
125     free(cl_bias[ii]);
126     free(cl_data[ii]);
127     free(cl_out[ii]);
128   }
129   free(cl_proposal);
130   free(cl_bias);
131   free(cl_data);
132   free(cl_noise);
133   free(cl_out);
134 }
```

```
135
136 int main(int argc,char **argv)
137 {
138   int n_lbin=1,pol_1=0,pol_2=0,is_auto=0,print_help=0;
139   char fname_map_1[256]="none";
140   char fname_map_2[256]="none";
141   char fname_mask_1[256]="none";
142   char fname_mask_2[256]="none";
143   char fname_temp_1[256]="none";
144   char fname_temp_2[256]="none";
145   char fname_bins[256]="none";
146   char fname_cl_noise[256]="none";
147   char fname_cl_proposal[256]="none";
148   char fname_coupling[256]="none";
149   char fname_out[256]="none";
150   nmt_field *fl1,*fl2;
151
152   if(argc==1)
153     print_help=1;
154
155   char **c;
156   for(c=argv+1;*c;c++) {
157     if(!strcmp(*c,"-map"))
158       sprintf(fname_map_1,"%s",*++c);
159     else if(!strcmp(*c,"-map_2"))
160       sprintf(fname_map_2,"%s",*++c);
161     else if(!strcmp(*c,"-mask"))
162       sprintf(fname_mask_1,"%s",*++c);
163     else if(!strcmp(*c,"-mask_2"))
164       sprintf(fname_mask_2,"%s",*++c);
165     else if(!strcmp(*c,"-temp"))
166       sprintf(fname_temp_1,"%s",*++c);
167     else if(!strcmp(*c,"-temp_2"))
168       sprintf(fname_temp_2,"%s",*++c);
169     else if(!strcmp(*c,"-pol"))
170       pol_1=atoi(*++c);
171     else if(!strcmp(*c,"-pol_2"))
172       pol_2=atoi(*++c);
173     else if(!strcmp(*c,"-cl_noise"))
174       sprintf(fname_cl_noise,"%s",*++c);
175     else if(!strcmp(*c,"-cl_guess"))
176       sprintf(fname_cl_proposal,"%s",*++c);
177     else if(!strcmp(*c,"-coupling"))
178       sprintf(fname_coupling,"%s",*++c);
179     else if(!strcmp(*c,"-out"))
180       sprintf(fname_out,"%s",*++c);
181     else if(!strcmp(*c,"-binning"))
182       sprintf(fname_bins,"%s",*++c);
183     else if(!strcmp(*c,"-nlb"))
184       n_lbin=atoi(*++c);
185     else if(!strcmp(*c,"-h"))
186       print_help=1;
187     else {
188       fprintf(stderr,"Unknown option %s\n",*c);
189       exit(1);
190     }
191   }
192
193   if(!strcmp(fname_map_1,"none")) {
194     fprintf(stderr,"Must provide map to correlate!\n");
195     print_help=1;
196   }
197   if(!strcmp(fname_mask_1,"none")) {
198     fprintf(stderr,"Must provide mask\n");
199     print_help=1;
200   }
201   if(!strcmp(fname_out,"none")) {
202     fprintf(stderr,"Must provide output filename\n");
203     print_help=1;
204   }
```

```
205
206   if(print_help) {
207     fprintf(stderr,"Usage: namaster -<opt-name> <option>\n");
208     fprintf(stderr,"Options:\n");
209     fprintf(stderr,"  -map      -> path to file containing map(s)\n");
210     fprintf(stderr,"  -map_2    -> path to file containing 2nd map(s) (optional)\n");
211     fprintf(stderr,"  -mask     -> path to file containing mask\n");
212     fprintf(stderr,"  -mask_2   -> path to file containing mask for 2nd map(s) (optional
          )\n");
213     fprintf(stderr,"  -temp     -> path to file containing contaminant templates (
          optional)\n");
214     fprintf(stderr,"  -temp_2   -> path to file containing contaminant templates\n");
215     fprintf(stderr,"               for 2nd map(s) (optional)\n");
216     fprintf(stderr,"  -pol      -> spin-0 (0) or spin-2 (1) input map(s)\n");
217     fprintf(stderr,"  -pol_2    -> spin-0 (0) or spin-2 (1) 2nd input map(s)\n");
218     fprintf(stderr,"  -cl_noise -> path to file containing noise Cl(s)\n");
219     fprintf(stderr,"  -cl_guess -> path to file containing initial guess for the Cl(s)\n
          ");
220     fprintf(stderr,"  -coupling -> path to file containing coupling matrix (optional)\n"
          );
221     fprintf(stderr,"  -out      -> output filename\n");
222     fprintf(stderr,"  -binning  -> path to file containing binning scheme\n");
223     fprintf(stderr,"  -nlb      -> number of ells per bin (used only if -binning isn't
          used)\n");
224     fprintf(stderr,"  -h        -> this help\n\n");
225     return 0;
226   }
227
228   if(n_lbin<=0)
229     report_error(1,"#ell per bin must be positive\n");
230
231   fl1=nmt_field_read(fname_mask_1,fname_map_1,fname_temp_1,pol_1);
232
233   if(!strcmp(fname_map_2,"none")) {
234     fl2=fl1;
235     is_auto=1;
236   }
237   else {
238     if(!strcmp(fname_mask_2,"none"))
239       sprintf(fname_mask_2,"%s",fname_mask_1);
240     if(!strcmp(fname_temp_2,"none"))
241       sprintf(fname_temp_2,"%s",fname_temp_1);
242     fl2=nmt_field_read(fname_mask_2,fname_map_2,fname_temp_2,pol_2);
243   }
244
245   run_master(fl1,fl2,
246        fname_cl_noise,
247        fname_cl_proposal,
248        fname_coupling,
249        fname_out,fname_bins,n_lbin);
250
251   nmt_field_free(fl1);
252   if(!is_auto)
253     nmt_field_free(fl2);
254
255   return 0;
256 }
```

This code, together with its compilation script is included in the present version of CosmoMad in the directory `sample`.

References

[1] Bardeen J. M., Bond J. R., Kaiser N., Szalay A. S. (1986). *The statistics of peaks of gaussian random fields*. ApJ, **304**:15

[2] Crocce M., Scoccimarro R. (2006) *Renormalized cosmological perturbation theory*. PRD, **73**:063519

[3] Eisenstein D. J., Hu W. (1998). *Baryonic features in the matter transfer function.* ApJ, **496**:605

[4] Kaiser N. (1987). *Clustering in real space and in redshift space.* MNRAS **227**:1

[5] Peacock J. A. 2007, MNRAS, 379, 1067

[6] Press W. H., Schechter P., 1974, ApJ, 187, 425

[7] Sheth R. K. & Tormen G. 2002, MNRAS, 329, 61

[8] Smith R. et al. (2003) *Stable clustering, the halo model and nonlinear cosmological power spectra.* MNRAS, **341**:1311