NaMaster: API documentation

February 19, 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 4 below contains a fully working C script that calls the NaMaster library.

2 NaMaster - the program

NaMaster comes with its own executable that computes the pseudo-C_ℓ power spectrum of two input masked fields with possible contaminants.

3 C documentation

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

3.1 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`.

3.2 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`.

3.3 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:

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

   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.

Note that the input and output power spectra are given as 2D arrays. The first dimension runs through $N_{\rm spec}$, the number of different cross-spectra:

1. For two spin-0 fields f_1 and f_2, $N_{\rm spec}=1$: $C_\ell = \left(C_\ell^{f_1 f_2}\right)$.
2. For a spin-0 field f_1 and a spin-2 field f_2, $N_{\rm spec}=2$: $C_\ell = \left(C_\ell^{f_1 f_2^E}, C_\ell^{f_1 f_2^B}\right)$, where $f_2^{E,B}$ are the E and B-modes of f_2.
3. For two spin-2 fields f_1 and f_2, $N_{\rm spec}=4$: $C_\ell = \left(C_\ell^{f_1^E f_2^E}, C_\ell^{f_1^E f_2^B}, C_\ell^{f_1^B f_2^E}, C_\ell^{f_1^B f_2^B}\right)$, where $f_x^{E,B}$ are the E and B-modes of f_x.

The second dimension runs through the different multipole indices or bandpowers. For power spectra, before binning into bandpowers, this index runs from 0 to $\ell_{\rm max} = 3\,$`nside` $- 1$, where `nside` is the HEALPix resolution of the fields. For bandpowers, this index runs through the number of different bandpowers defined by the associated `nmt_binning_scale` structure.

nmt_compute_coupling_matrix

```
nmt_workspace * nmt_compute_coupling_matrix(nmt_field *fl1,nmt_field *fl2,
nmt_binning_scheme *bin)
```

Computes the coupling matrix and the binned coupling matrix for the two fields `fl1` and `fl2` and the binning scheme `bin`. Note that the only information needed from the two fields is their masks and spins. These matrices are stored internally in the `nmt_workspace` structure returned by this function.

nmt_workspace_write

```
void nmt_workspace_write(nmt_workspace *w,char *fname)
```

Writes an `nmt_workspace` structure into a file `fname` (using an internal binary format).

nmt_workspace_read

```
nmt_workspace * nmt_workspace_read(char *fname)
```

Returns a pointer to a `nmt_workspace` structured read from file `fname`. This file should have been generated by `nmt_workspace_write`. These two functions are useful when computing the power spectrum of several pairs of fields with the same pairs of masks, for which the coupling matrices only need to be computed once.

nmt_workspace_free

```
void nmt_workspace_free(nmt_workspace *w)
```

Frees up all memory associated with an `nmt_workspace` structure.

nmt_compute_deprojection_bias

```
void nmt_compute_deprojection_bias(nmt_field *fl1,nmt_field *fl2,
double **cl_proposal,double **cl_bias)
```

Estimates the bias to the cross-power spectrum of two fields `fl1` and `fl2` induced by the contaminant cleaning (i.e. $C_\ell^{\rm cont}$ in Eq. 4). This is returned into the variable `cl_bias`, which should have been previously allocated (see description in the introduction to this section). The estimate of this bias depends on a guess for the true power spectrum of both fields, given by `cl_proposal`[1]. Note that this operation does not require knowledge of the mode-coupling matrix, and therefore no `nmt_workspace` structure is needed.

nmt_compute_coupled_cell

```
void nmt_compute_coupled_cell(nmt_field *fl1,nmt_field *fl2,double **cl_out,int iter)
```

This computes the full-sky angular cross-power spectrum of two masked fields `fl1` and `fl2` without aiming to deconvolve the mode-coupling matrix. Effectively, this is equivalent as calling the usual HEALPix `anafast` routine on the masked and contaminant-cleaned maps. The coupled power spectrum is returned in `cl_out`, which should have been previously allocated (see description in the

[1] Thus, the pseudo-C_ℓ can be thought of as a recursive algorithm, where the estimate of the true power spectrum in a previous iteration is used as a proposal for the computation of the contaminant bias in the next one.

introduction to this section). The variable `iter` corresponds to the number of iterations used to compute the spherical harmonic transform. A value of 0 will correspond to the fastest but most inaccurate computation. Higher values will yield more accurate results at high-ℓ (`niter` $= 3$ is usually enough in most cases).

Since no attempt is made to deconvolve the mode-coupling matrix, this function does not require a `nmt_workspace` structure.

nmt_decouple_cl_l

```
void nmt_decouple_cl_l(nmt_workspace *w,double **cl_in,double **cl_noise_in,
double **cl_bias,double **cl_out)
```

This function performs the operation in Eq. 4: debiasing and decoupling of a power spectrum computed from `nmt_compute_coupled_cell`. The coupled power spectrum $\tilde{C}_\ell^{\rm clean}$ must be provided in `cl_in`. The contaminant bias $\tilde{C}_\ell^{\rm cont}$ (e.g. computed through `nmt_compute_deprojection_bias`) and noise bias $\tilde{C}_\ell^{\rm noise}$ must be provided through `cl_bias` and `cl_noise_in` respectively. The mode-coupling matrix M (and its binned version $\mathcal{M}$) are stored within `w`, and the de-coupled bandpowers are returned in `cl_out`.

nmt_compute_power_spectra

```
nmt_workspace * nmt_compute_power_spectra(nmt_field *fl1,nmt_field *fl2,
nmt_binning_scheme *bin,nmt_workspace *w0,double **cl_noise,double **cl_proposal,
double **cl_out)
```

Carries out steps 2-5 of the pseudo-C_ℓ estimator described in the introduction of this section. `fl1` and `fl2` are the two fields to correlate, `bin` defines the output bandpowers, `cl_noise` is the noise bias, `cl_proposal` is the best guess for the true power spectrum needed to estimate the contaminant bias $\tilde{C}_\ell^{\rm cont}$ (see `nmt_compute_deprojection_bias`). The output bandpowers are stored in `cl_out`, which should have been pre-allocated.

This function also accepts an input pointer to a `nmt_workspace` structure, `w0`. If a `NULL` pointer is passed, the function will compute the mode-coupling matrix and return a newly-allocated `nmt_workspace` structure containing this information. Otherwise, the function will skip this computation and use the mode-coupling matrix stored in `w0`. In this latter case, the function would return a pointer to `w0`. Note that a call to this function is equivalent to a successive call to `nmt_compute_coupling_matrix`, `nmt_compute_deprojection_bias`, `nmt_compute_coupled_cell` and `nmt_decouple_cl_l`.

nmt_couple_cl_l

```
void nmt_couple_cl_l(nmt_workspace *w,double **cl_in,double **cl_out)
```

Convolves an input power spectrum `cl_in` with the mode-coupling matrix stored in `w`, and provides the output in `cl_out`. I.e.:

$$C_\ell^{\rm out} = \sum_{\ell'} \mathsf{M}_{\ell\ell'} C_{\ell'}^{\rm in} \tag{6}$$

where $C_\ell^{\rm out} \rightarrow$ `cl_out` and $C_\ell^{\rm in} \rightarrow$ `cl_in`.

3.4 Utility functions

nmt_apodize_mask

```
void nmt_apodize_mask(long nside,double *mask_in,double *mask_out,double aposize,
char *apotype)
```

This function apodizes an input mask, provided in `mask_in` as a HEALPix map, and stores the result in `mask_out`. The apodization is defined by an apodization scale `aposize` (in degrees) and an apodization type `apotype`. Three different apodization types are supported (in what follows θ_* will be the apodization scale `aposize`):

- `apotype`=**"C1"**. All pixels are multiplied by a factor f given by:

$$f = \begin{cases} x - \sin(2\pi x)/(2\pi) & x < 1 \\ 1 & \text{otherwise} \end{cases} \tag{7}$$

  where $x \equiv \sqrt{(1-\cos\theta)/(1-\cos\theta_*)}$, and θ is the angular separation between the pixel and its closest masked pixel (i.e. the closest pixel where the mask is zero.

- `apotype`=**"C2"**. All pixels are multiplied by a factor f given by:

$$f = \begin{cases} \frac{1}{2}\left[1 - \cos(\pi\, x)\right] & x < 1 \\ 1 & \text{otherwise} \end{cases} \tag{8}$$

  where $x \equiv \sqrt{(1-\cos\theta)/(1-\cos\theta_*)}$, and θ is the angular separation between the pixel and its closest masked pixel (i.e. the closest pixel where the mask is zero.

- `apotype`=**"Smooth"**. This apodization is carried out in three steps:

  1. All pixels within a disc of radius $2.5\theta_*$ of a masked pixel (i.e. where the mask is zero) are masked.
  2. The resulting map is smoothed with a Gaussian window function with standard deviation $\sigma = \theta_*$.
  3. One final pass is made through all pixels to ensure that all pixels that were originally masked remain masked after the smoothing operation.

4 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_constant(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,3);
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