NaMaster: C API and code documentation

March 26, 2017

Contents

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. After installation (and assuming it has been installed in an accessible directory), the program can be invoked by typing `namaster`.

The user interacts with `namaster` by providing parameters as command-line arguments in the form of a pair: `-<param_name> <param>`. After typing

```
~$ namaster -h
```

the user is presented with a list of all available input parameters:

```
Usage: namaster -<opt-name> <option>
Options:
  -map      -> path to file containing map(s)
  -map_2    -> path to file containing 2nd map(s) (optional)
  -mask     -> path to file containing mask
  -mask_2   -> path to file containing mask for 2nd map(s) (optional)
  -temp     -> path to file containing contaminant templates (optional)
  -temp_2   -> path to file containing contaminant templates
               for 2nd map(s) (optional)
  -pol      -> spin-0 (0) or spin-2 (1) input map(s)
  -pol_2    -> spin-0 (0) or spin-2 (1) 2nd input map(s)
  -pure_e   -> use pure E-modes for 1st maps? (0-> no or 1-> yes (default->no))
  -pure_b   -> use pure B-modes for 1st maps? (0-> no or 1-> yes (default->no))
  -pure_e_2 -> use pure E-modes for 2nd maps? (0-> no or 1-> yes (default->no))
  -pure_b_2 -> use pure B-modes for 2nd maps? (0-> no or 1-> yes (default->no))
  -cl_noise -> path to file containing noise Cl(s)
  -cl_guess -> path to file containing initial guess for the Cl(s)
  -coupling -> path to file containing coupling matrix (optional)
  -out      -> output filename
  -binning  -> path to file containing binning scheme
  -nlb      -> number of ells per bin (used only if -binning isn't used)
  -h        -> this help
```

Some clarification is in order regarding some of these parameters:

- `map` and `map_2`: these arguments should point to FITS files containing the maps of the two fields to correlate. These files should contain one map if `pol` (or `pol_2`) is 0 and two maps (Q and U) if `pol` (or `pol_2`) is 1. If `map_2` is not passed, the auto-correlation of `map` will be computed (and all other `_2` options will be ignored).
- `mask` and `mask_2`: these arguments should point to FITS files containing the masks of the two fields to correlate (one map per file).
- `temp` and `temp_2`: these arguments should point to FITS files containing the contaminant templates you suspect are poluting your fields. The number of contaminants will be derived by the code based on the number of maps in the files. Note that an exception will occur if this number is not a multiple of the number of maps corresponding to the field's spin (see above).

- `cl_noise`: this file should contain the expected noise bias (see discussion below Eq. 4) to be subtracted from the estimated power spectrum. The format of this file should be $N+1$ columns, with the first column being the multipole number ℓ (starting at $\ell = 0$), and the next N being the different component of the power spectrum. N should correspond to the number of power spectra expected given the spins of the two fields being correlated. I.e. $N = 1$ if `pol=pol_2=0` ((TT)), $N = 2$ if `pol=0`, and `pol_2=1` (or vice-versa, (TE, TB)), and $N = 4$ if `pol=pol_2=1` ((EE, EB, BE, BB)). If this option is not passed, it is assumed to be 0. This file should contain 3`nside` rows, where `nside` is the HEALPix resolution parameter of the input maps.
- `cl_guess`: proposal power spectrum (i.e. best guess of true power spectrum) used to compute the deprojection bias (same format as `cl_noise` above). This is only relevant for contaminated fields. If not passed, the proposal power spectrum is set to 0.
- `out`: path to output ASCII file. First column will correspond to the effective ℓ in the bandpower. All subsequent columns correspond to the different components of the power spectrum (as described above depending on the field spins).
- `binning`: path to ASCII file describing the binning scheme defining the output bandpowers. This file should contain three columns: the first column should contain the bandpower index of the different multipoles, the second column should contain the multipoles and the third column should contain the weight corresponding to each multipole (the sum of weights in each bandpower is automatically normalized to 1).
- `nlb`: number of multipoles per bandpower. In this case the output power spectrum will be computed by binning multipoles in sets of `nlb` from $\ell = 2$ to $\ell = 3$`nside` $- 1$.

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,double ***temp,double *beam,int pure_e,int pure_b)
```

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.
- `beam`: spherical harmonic transform of the instrumental beam (assumed to be rotationally symmetric - i.e. no m dependence). If `NULL`, no beam will be corrected for. Otherwise, this array should have $3 \times$ `nside` elements, corresponding to multipoles $\ell \in [0, 3\,$`nside` $- 1]$.
- `pure_e`, `pure_b`: set to a value different from 0 if you want to use pure-E/B modes for this field. Note that, in this case you should make sure the mask and its first derivatives are continuous on its boundary (see `nmt_apodize_mask` below).

nmt_field_read

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

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. Pass "none" if no contaminants are needed (in which case deprojection and debiasing will not take place).
- `fname_beam`: file name pointing to an ASCII file containing the spherical harmonic transform of the instrumental beam (assumed to be rotationally symmetric - i.e. no m dependence). If ``none'', no beam will be corrected for. Otherwise, this file should contain two columns (ℓ and b_ℓ) and $3 \times$ `nside` columns, corresponding to multipoles $\ell \in [0, 3\,$`nside` $- 1]$.
- `pure_e`, `pure_b`: set to a value different from 0 if you want to use pure-E/B modes for this field. Note that, in this case you should make sure the mask and its first derivatives are continuous on its boundary (see `nmt_apodize_mask` below).

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:

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

   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. Once the pseudo-C_ℓ coupling matrix corresponding to two fields has been computed and stored in an `nmt_workspace`, the same workspace can be re-used to compute the pseudo-C_ℓ of any pair of fields as long as they have the same polarizations, masks and choices for E and B purification as the original fields.

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 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

[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.

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 $C_\ell^{\rm out} \to$ `cl_out` and $C_\ell^{\rm in} \to$ `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 simple code using the NaMaster library. This code takes, as command-line options, filenames for two FITS files containing HEALPix maps for a spin-0 field and a mask, and writes the MASTER estimate of its power spectrum into an ASCII file. This file can be found in `test/sample.c`, and can be compiled as described in Section 1.2.

```c
#include <stdlib.h>
#include <stdio.h>
#include <namaster.h>
#include <math.h>

int main(int argc,char **argv)
{
  long i,nside;
  if(argc!=4) {
    fprintf(stderr,"Usage: ./sample <fname_map> <fname_mask> <fname_out>\n");
    exit(1);
  }

  //Create spin-0 field
  nmt_field *fl1=nmt_field_read(argv[2],argv[1],"none","none",0);

  //Create a binning scheme (20 multipoles per bandpower)
  nmt_binning_scheme *bin=nmt_bins_constant(20,fl1->lmax);
  //Compute array of effective multipoles
  double *ell_eff=calloc(bin->n_bands,sizeof(double));
  nmt_ell_eff(bin,ell_eff);

  //Allocate memory for power spectrum
  double *cl_out=malloc(bin->n_bands*sizeof(double));

  //Dummy array to be passed as proposal and noise power spectrum
  //These are not needed here, because we don't want to remove noise bias
  //and we have assumed the field is not contaminated
  double *cl_dum=calloc((fl1->lmax+1),sizeof(double));

  //Compute pseudo-Cl estimator
  nmt_workspace *w=nmt_compute_power_spectra(fl1,fl1,bin,NULL,&cl_dum,&cl_dum,&cl_out);

  //Write output
  FILE *fo=fopen(argv[3],"w");
  for(i=0;i<bin->n_bands;i++)
    fprintf(fo,"%.2lE %lE\n",ell_eff[i],cl_out[i]);
  fclose(fo);

  //Free stuff up
  nmt_workspace_free(w);
  free(cl_dum);
  free(cl_out);
  free(ell_eff);
  nmt_bins_free(bin);
  nmt_field_free(fl1);

  return 0;
}
```