clik Documentation

Release 1.0

The Planck collaboration

March 05, 2013

CONTENTS

clik is a C library containing codes to compute different approximations for the Planck likelihood. Wrappers are provided so that the functions can also be called from python and f90.

CHAPTER

ONE

DESIGN CHOICES

1.1 A likelihood is entirely defined by a likelihood file.

The likelihood files (in fact directories, containing data as fits files and metadata as ascii files) contain all the info needed to define the likelihood. This encompass both the data needed to compute the likelihood,but also parameters describing the type of mathematical approximation used to compute this particular likelihood, and parameters describing the expected input of the likelihood.

A likelihood file can also be the combinaison of various underlying likelihoods, the computation from the top likelihood will be the product of each of the underlying one. A tool is distributed to allow joining and disjoining likelihoods. This means that a likelihood file containing the some of different likelihood approximation will have to contain the data for each of those approximation. This will translate into possibly huge file. And variations of those likelihoods will contains yet more copy of this huge data. To solve partially this problem, it is possible for the likelihoods that need a lot of data to optionally refer to external files. This will be the case in particular for the WMAP and lowlike likelihoods. In those cases, the data can either be included in the file (as described above) or installed in some directory, in which case the likelihood file simply refers to the path of this directory.The latter case improve the efficiency of the initialization of clik when using this kind of likelihood files.

In order to use the library, the first step is thus to initialize it with such a file. Functions or subroutine to perform this initialization are available in each language. Another function is provided to cleanup the memory at the end of the use of a given likelihood. Some likelihoods can be initialized within the same session, allowing to perform comparison between different likelihood approximation whithin the same run.

1.2 The library computes an approximation of the log likelihood

And nothing else. The library does not compute minus the log likelihood or a chi2 like value. Just the log likelihood. A function is provided in each language to compute the log likelihood, given a set of parameters.

1.3 The input of the compute function are multipoles of the power spectra and nuisance parameters.

The `compute` function expect one single vector of double. This vector must contains power spectra one after the other, starting al l=0, in microK^2, and then the nuisance parameters. Since the likelihood is defined by the likelihood file the exact range of power spectra needed by this function can vary from likelihood approximation to likelihood approximations. The same holds for the nuisance parameters. Functions are provided to query a likelihood file and obtain this info. More precisely, three such function are available.

- clik_get_has_cl: retrieve an array of 6 flags, describing the power spectra needed (1 if needed 0 otherwise). The order is TT, EE, BB, TE, TB, EB. Thus if this function answers (1,1,0,1,0,0) it means that the input vector of the compute function must contain, in that order, the power spectra for TT, EE an TE.
- clik_get_lmax: retrieve an array of 6 integer giving the lmax value for each power spectra. Order is same as above. -1 means that the spectra is not needed. Thus if this function answer (2000,1000,-1,2000,-1,-1). This means that the vector must contain, in that order, the first 2001 multipoles of the TT power spectra (from 0 to 2000 included) followed by the first 1001 multipoles of the EE power spectra (0 to 1000 included) and next by the first 2001 multipoles of the TE power spectra (0 to 2000 included). Thus the first 5003 elements of the parameter vectors are the values of different power spectra. Note that this is also the sum of the result array of clik_get_lmax plus 6. Isn't this fantastic ?
- clik_get_parameter_names: returns the number of nuisance parameters and fills an array of string giving their names. For example, if this function returns 2, and ('sigma8', 'fwhm_error') it means that the last two elements of the parameter vector must be the value of sigma8 and fwhm_error, whatever those parameters mean.

To sumarize, the input vector of the compute function must be an array of N = Ncl + Nnuis doubles. Ncl being the sum + 6 of the return array of clik_get_lmax, and Nnuis is the return of clik_get_parameter_names. The power spectra must be the Ncl first elements of that array. They start at C0 and en up at some Clmax[i] (included), the ith elements of the return of clik_get_lmax. The ordering of the power spectra is always TT EE BB TE TB EB. The Cls are in microK^2. The nuisnce parameters are the Nnuis last element of the parameter vector. Their names are given by the return array of the function clik_get_parameter_names.

1.4 Pitfalls

The function computes the log likelihood ?

The Cls must be given in that order TT, EE, BB, TE, TB, EB. And, yes, the library expects C0 and C1 for each power spectra.

The library expect power spectra and not l(l+1)Cl/2pi or other combination.

The library really wants power spectra in microK^2.

1.5 Lensing likelihood

The behaviour of the lensing likelihood is similar to the one of the CMB likelihoods except that it does not have nuisance parameters and expected the phiphi Cl as well as the TT Cl.

CHAPTER

TWO

INSTALLING

The package can be installed using two different tools, waf or make. Using waf, the installer will test for the different dependencies and try to install them if they are missing. Using make, one would have to modify the `Makefile` file for your particular computer and install the dependencies. Besides, when using make, no test of the availability of the requisite will be made.

The package has a set of core utilities, consisting in the `clik` library with a C and F90 API, as well as the programs **clik_example_C** and **clik_example_F90** which allows to compute a log likelihood for a Cl and nuisance parameter file, and also doubles as example of how to interface with the library in this two languages.

The package also have a set of optional utilities, consisting in a python wrapper of the library, along with a few python scripts allowing to explore the content of likelihood files, and to manipulate them. Those tools will have a few more requirements to be built.

Finally, when using waf, and provided that the optional python utilities can be installed, a wrapper to the wmap9 likelihood will also be installed.

2.1 Requisites

2.1.1 Mandatory requisites

modern c and fortran compilers (icc, gcc, ifort anf gfortran are ok) are absolute requisites. Gcc version must be >= 4.2 and gfortran version must be >=4.3.

To use the waf tool, one also need Python (>=2.5).

2.1.2 Mandatory requisites that can be installed when using waf

Those requisites will not be installed automatically when using make, and must be manually installed. **‘cfitsio‘_**, as well as blas and lapack distribution (preferably intel MKL on linux machine. Macos computers already have a reasonnable parallelized blas/lapack that clik can use) are needed for the core functionalities of clik. If absent (or not available in the correct flavor), they can be installed automatically using waf and the options described below.

2.1.3 Optional requisites

A python distribution including the header and library (if absent, this will not be installed automatically by waf), and the pyfits (2.4<=version), numpy (version>1.1) and cython python package (version>1.12) are needed to provide the (optional) clik python wrapper and tools. The three python package will only be installed automatically when using waf if the python header and library are available on the system.

2.2 Install with waf

The library and executables must be installed with the ‘waf’ tool. It is distributed in the package. Please have a look at the waf webpage.

The installer must first be configured using:

```
$> ./waf configure
```

This will test for the presence of all the required dependencies (as described above). Command line options are available to help the configuration by setting the location of some of those dependency, and or chose between different compiler options. An option can also be used to cause waf to try to download, compile and install for the user the required dependencies that are not found by the automatic discovery system. For a complete list of options do:

```
$> ./waf configure --help
```

After this (possibly lengthy) configuration step, clik per-se can be compiled and installed with:

```
$> ./waf install
```

2.2.1 Simplest case

In the simplest case, we will assume that you want all the dependencies absent from the usual locations to be installed automatically. Note that this translate into a rather slow clik library, since the lapack library will probably be compiled from a simple, non-parallel, version of the lib. That being said , the simplest configuration (if slow) line would be:

```
$> ./waf configure --install_all_deps
```

followed by:

```
$> ./waf install
```

Note that the option `--install_all_deps` is more powerful than simply installing all the dependency, this will be described in a latter section.

2.2.2 Simplest case with mkl

Using the automatically installed lapack library results in a slow clik library. One can solve easily this problem by letting the configuration know about an existing mkl library. To do so the configuration line has to be changed into:

```
$> ./waf configure --install_all_deps \
      --lapack_mkl=/PATH/TO/YOUR/MKL/ --lapack_mkl_version=MKL.VERSION
```

The `/PATH/TO/YOUR/MKL/` can be replaced by `${MKLROOT}` for most recent mkl installs. Otherwise, it has to be a directory containing a `lib` and an `include` subdirectories. The `MKL.VERSION` can be any of 10.0, 10.1, 10.2, 10.3.

2.2.3 Simplest case with apple lapack

MacOS X has a standard parallelized lapack distribution. One can use it to accelerate clik. To do so the configuration line has to be changed into:

```
$> ./waf configure --install_all_deps --lapack_apple
```

2.2.4 (planck insider) I am installing on magique3

Use this:

```
$> /softs/python/2.7.2/bin/python waf configure \
      --install_all_deps --cfitsio_prefix=/softs/cfitsio/3.24/
      --lapack_mkl=/softs/intel/mkl/10.2.6.038 --lapack_mkl_version=10.2
```

and then:

```
$> ./waf install
```

See below for the other otions used here.

2.2.5 (planck insider) I am installing on ccin2p3

Use this:

```
$> ./waf configure --install_all_deps \
         --lapack_mkl=/usr/local/intel/mkl/10.3.8/ --lapack_mkl_version=10.3
```

and then:

```
$> ./waf install
```

2.2.6 ADVANCED: Installing with a particular Python executable

It is possible to install clik with a python install different from the default one. For example if the default python installation does not contains the required header and libraries. To do so, call waf this way:

```
$> /path/to/special/python waf configure
```

and then:

```
$> /path/to/special/python waf install
```

2.2.7 ADVANCED: Bypassing the default compilers

To bypass the c compiler detection, set the `CC` environment variable. To bypass the fortran compiler detection, set the `FC` environment variable. Beware, you can only set the `FC` environment variable to either an intel fortran compiler or a gfortran compiler.

Shortcuts for some classical cases are provided:

- `--icc` causes the installer to use icc as c compiler.
- `--ifort` causes the installer to use ifort as fortran compiler.
- `--gcc` causes the installer to use gcc as c compiler.
- `--gfortran` causes the installer to use gfortran as fortran compiler.

2.2.8 ADVANCED: Setting the architecture

The architecture (32 or 64bits) can be set using the `--m32` or `--m64` flags. 64bits is the default.

2.2.9 ADVANCED: Setting installation path

The installation path can be set using the `--prefix=SOMEPATH` option. Default is to install in the current directory.

2.2.10 ADVANCED: More on the automatic installation of dependencies

There are three levels of automatic installation. If one wants to *always* install the dependencies, one can use the `--force_install_all_deps`:

```
$> ./waf configure --forceinstall_all_deps
```

If one wants to install only the dependencies that are not present in the usual location (or that are present, but not compiled in a way suitable for clik), one can use the `--install_all_deps` option, already described above. Since this option first tests for the presence of each library, it can be used to upgrade a clik install, avoiding to reinstall everything.

Finally, each dependency can be installed on a dependency by dependency basis, using the `--XXX_install` or `--XXX_installifneeded` options where `XXX` is the name of the dependency. The former install all the time the dependency, the latter install it only if it is not found in the usual locations. In that sense, `--forceinstall_all_deps` works as if all possible `--XXX_install` options has been set, and `--install_all_deps` as if all `--XXX_installifneeded` options have been set.

One should also note that `--forceinstall_all_deps` and `--install_all_deps` are also unactivated on a dependency by dependency basis if any of the `--XXX_prefix`, `--XXX_lib`, `--XXX_include`, or other dependency specific options are present. In that case, the the `XXX` dependency, the configuration script will look in the locations described by those option and if the package is not found will report an error.

2.2.11 ADVANCED: Setting the location of a library

The location of the library dependencies (gsl, hdf5, healpix, blas/lapack) must be known to the installer. By default, it will look for them in the classical system locations: `/usr/lib`, `/usr/lib64`, `/usr/local/lib`, `/usr/local/lib64` for the library, `/usr/include` and `/usr/local/include` for the include files. One can change the lookup path on a library by library basis. If a given dependency, `XXX`, is installed on the system such that its lib are in `SOMEPREFIXPATH/lib` and its include files in `SOMEPREFIXPATH/include`, setting the command line option `--XXX_prefix=SOMEPREFIXPATH` will allow the clik install system. If `SOMEPREFIXPATH` is identical to the the install path of clik, this option can be replaced by `-XXX_islocal`.

If the library are at `SOMEWEIRDPATH` and the includes at `SOMEDIFFERENTPATH`, then setting the two options `--XXX_lib=SOMEWEIRDPATH --XXX_include=SOMEDIFFERENTPATH` will allow the clik install system to find them.

Finally, if the name of the library files differs from the usual ones one can set the option `--XXX_link=THELINKLINE`.

Using these options allow to point the installer to a pmclib install in order to allow the linking of clik with pmclib.

2.2.12 ADVANCED: Special case: the mkl library

This option is only for advanced users. The blas/lapack distribution installed automatically is a very inefficient one. To improve the performance of clik (especially the low-l pixel based likelihood), one is advised to use the MKL library, which is fully supported and allow the use of shared memory computer architectures.

A special option is present to simplify the install using the intel MKL library: setting the option `--lapack_mkl=PATH_OF_THE_MKL_INSTALL` together with `--lapack_mkl_version=SOMEVERSION` will allow clik to pick the correct set of libraries for the particular version of the mkl package (version 10.0, 10.1, 10.2 and 10.3 only). Setting this option will cancel the `--install_all_deps` option for the lapack dependency only.

On a MacOS X computer, one can use Apple provided lapack by setting `--lapack_apple`.

2.2.13 ADVANCED: Special case: WMAP likelihood

Clik can provide a wrapper to the wmap9 likelihood. It need to now where the sources of the likelihood are located to compile against them. One must set the option `--wmap_src=WMAP7SRCPATH` or let the install system download it for you by setting the option `--wmap_install`. Note that to actually use this likelihood, one must also download the data files and prepare clik likelihood files from them. Look at *Using WMAP9 likelihood*. The `--install_all_deps` and `--forceinstall_all_deps` options will automatically download the sources, as if `-wmap_install` was set.

2.2.14 ADVANCED: Putting it all together

The following command:

```
$> ./waf configure --install_all_deps
```

will tell the clik install system to install all the possible external dependency in the current directory.

The following command:

```
$> ./waf configure --lapack_mkl=/opt/intel/mkl \ --lapack_mkl_version=10.2
   --cfistio_prefix=/usr/local/cfitsio --cython_install
```

will tell the clik install system to install cython. The cfitsio library will be looked for in the unusual dir `/usr/local/cfitsio`. /All the other dependency will be looked up in the classical locations. The blas/lapack library will be the one from an mkl install located at –lapack_mkl=/opt/intel/mkl. Clik will be compiled in 64bit and installed in the current directory.

2.2.15 ADVANCED: Best advanced choice

Use a mkl lapack install and let the other dependencies on auto install:

```
$> ./waf configure --install_all_deps  \
      --lapack_mkl=/opt/intel/mkl --lapack_mkl_version=10.2
```

This will use your mkl libraries from `/opt/intel/mkl`, test if numpy, cython and gsl are installed on your computer (often the case) if not install them, and finally install all the other requirements (helpaix, hdf5 and its python wrapper).

2.3 Installing with make

The first lines of the `Makfile` file must be checked and modified before compiling. In particular, one must set,

- the location of the cfitsio library
- the location and version of the mkl library (or other lapack library)
- the location and list of library needed to link c with fortran

To build and install the core utilities, use the following command:

```
$> make install
```

To build and install the optional utilities, use the following command:

```
$> make install_python
```

2.4 Environment variables

Depending of your shell, a configuration file named `clik_profile.sh` of `clik_profile.csh` will be installed in the `bin` directory at the install location of clik. One can source it on the command line, or include it in its startup configuration file to set the environment variable needed by clik. This tool is installed both bty waf and make.

CHAPTER

THREE

INTERFACING THE LIBRARY WITH A C EXECUTABLE

The following gives a description of the c API of the library, and how to correctly compile and link against it.

3.1 Compiling and linking

The program *clik-config* (installed in PREFIX/bin) spits out on the standard output the barbaric option and link line to give to your prefered c compiler when compiling and linking against the clik lib.

The file `click_example_c.c` gives a simple example of the use of the c API. It is compiled and installed as **clik_example_C**.

3.2 API - CMB likelihood

All codes calling clik functions must

```
include "clik.h"
```

The library can initialize more than one likelihood. Likelihood are represented by a variable (in the following, named `clikid`) of type `clik_object*`.

3.2.1 Initialization

The library must be initialized by calling

`clik_object* clik_init(char* hdffilepath, error **err);`

The function returns a pointer on an object containing the definition of the likelihood. It expects two arguments, `hdffilepath` a string containing the path to a likelihood file, and `err` a c structure allowing error tracking. The error tracking system is provided by pmclib, please refer to its doc it for more info. If you don't which to use the error tracking system, set this argument to `NULL`. In this case, the library will only print out a message and force the calling program to exit in case of an error.

3.2.2 Querying the likelihood object

`void clik_get_has_cl(clik_object *clikid, int has_cl[6],error **err);`

This function fills the array `has_cl` with flags describing which power spectra are needed by the likelihood compute function (see *The input of the compute function are multipoles of the power spectra and nuisance parameters.*). The first argument of the function must be the return value from a previous call to `clik_init()`. The last argument allows error tracking. It can be left to `NULL`, in which case no error tracking is performed and the program exit with an explaining message in case of an error.

`void clik_get_lmax(clik_object *clikid, int lmax[6],error **err);`

This function fills the array `lmax` with the lmax value for each power spectra needed by the likelihood compute function (see *The input of the compute function are multipoles of the power spectra and nuisance parameters.*). The first argument of the function must be the return value from a previous call to `clik_init()`. The last argument allow to track errors. It can be left to `NULL`, in which case no error tracking is performed and the program exit with an explaining message in case of an error.

`int clik_get_extra_parameter_names(clik_object* clikid, parname **names, error **err);`

This function returns the number of nuisance parameters needed by the likelihood compute function (see *The input of the compute function are multipoles of the power spectra and nuisance parameters.*) and fills with their names the array `*names`. This array is an array of parname, who are `char[_pn_size]`. It is allocated by the function and MUST be deallocated by the caller after use. The first argument of the function must be the return value from a previous call to `clik_init()`. The last argument allow to track errors. It can be left to `NULL`, in which case no error tracking is performed and the program exit with an explaining message in case of an error.

3.2.3 Computing the log likelihood

`double clik_compute(clik_object *clikid, double *cl_and_pars,error **err);`

This function returns the value of the log likelihood for the parameter vector `cl_and_pars`. The content of this vector is desribed in *The input of the compute function are multipoles of the power spectra and nuisance parameters.*. The first argument of the function must be the return value from a previous call to `clik_init()`. The last argument allow to track errors. It can be left to `NULL`, in which case no error tracking is performed and the program exit with an explaining message in case of an error. This function can be called as many time as the user wants.

3.2.4 Cleanup

When a likelihood object is no more needed (i.e. when no more computation will be needed in the program), the memory it uses can be cleaned up calling

`void clik_cleanup(clik_object** pclikid);`

The first argument of the function must be the pointer on a variable containg the return value from a previous call to `clik_init()`. Upon return, the content of this variable will be changed to `NULL`.

3.3 API - lensing likelihood

All codes calling clik functions must

```
include "clik.h"
```

The library can initialize more than one likelihood. Likelihood are represented by a variable (in the following, named `clikid`) of type `clik_lensing_object*`.

3.3.1 Testing whether a file contains a lensing likelihood

One can test whether a file contains a lensing likelihood by calling

int clik_try_lensing(char* hdffilepath, error **err);

Return 1 or 0 depending if the file `hdffilepath` constains a lensing likelihood. If the file does not exist or cannot be read, an error us raised the usual way.

3.3.2 Initialization

The lensing likelihood must be initialized by calling

clik_lensing_object* clik_lensing_init(char* hdffilepath, error **err);

The function returns a pointer on an object containing the definition of the likelihood. It expects two arguments, `hdffilepath` a string containing the path to a lensing likelihood file, and `err` a c structure allowing error tracking. The error tracking system is provided by pmclib, please refer to its doc it for more info. If you don't which to use the error tracking system, set this argument to `NULL`. In this case, the library will only print out a message and force the calling program to exit in case of an error.

3.3.3 Querying the lensing likelihood object

int clik_lensing_get_lmax(clik_lensing_object *clikid, error **err);

This function returns the lmax value for both clpp and cltt.

int clik_get_lensing_extra_parameter_names(clik_lensing_object* clikid, parname **names, e

This function returns the number of nuisance parameters needed by the lensing likelihood compute function and fills with their names the array `*names`. This array is an array of parname, who are `char[_pn_size]`. It is allocated by the function and MUST be deallocated by the caller after use. The first argument of the function must be the return value from a previous call to `clik_lensing_init()`. The last argument allow to track errors. It can be left to `NULL`, in which case no error tracking is performed and the program exit with an explaining message in case of an error.

3.3.4 Computing the log likelihood

double clik_lensing_compute(clik_lensing_object *clikid, double *cl_and_pars,error **err);

This function returns the value of the log likelihood for the parameter vector `cl_and_pars`. This vector must have 2*(lmax_lensing+1) + number_of_lensing_extra_parameters elements. They are first the lensing_lmax+1 values of clpp, then the lensing_lmax+1 values of the cltt, the the extra parameter values. The first argument of the function must be the return value from a previous call to `clik_lensing_init()`. The last argument allow to track errors. It can be left to `NULL`, in which case no error tracking is performed and the program exit with an explaining message in case of an error. This function can be called as many time as the user wants.

3.3.5 Cleanup

When a lensing likelihood object is no more needed (i.e. when no more computation will be needed in the program), the memory it uses can be cleaned up calling

void clik_lensing_cleanup(clik_lensing_object pclikid);**

The first argument of the function must be the pointer on a variable containing the return value from a previous call to `clik_lensing_init()`. Upon return, the content of this variable will be changed to `NULL`.

CHAPTER

FOUR

INTERFACING THE LIBRARY WITH A F90 EXECUTABLE

The following gives a description of the f90 API of the library, and how to correctly compile and link against it.

4.1 Compiling and linking

The program *clik_f90-config* (installed in PREFIX/bin) spits out on the standard output the barbaric option and link line to give to your prefered c compiler when compiling and linking against the clik lib.

The file `click_example_f90.f90` gives a simple example of the use of the f90 API. It is compiled and installed as **clik_example_f90**.

4.2 API - CMB likelihood

All codes calling clik functions must

```
use clik
```

The library can initialize more than one likelihood. Likelihood are represented by a variable (in the following, named `clikid`) of type `type(clik_object)`.

4.2.1 Initialization

The library must be initialized by calling

subroutine **`clik_init`**(clikid, hdffilepath)

The subroutine sets the argument `clikid`, which is of type `type(clik_object)` to a handle on an object containing the definition of the likelihood. It expects two arguments, `hdffilepath` a string containing the path to a likelihood file. In case of error, the library will only print out a message and force the calling program to exit.

4.2.2 Querying the likelihood object

subroutine **`clik_get_has_cl`**(clikid, has_cl)

This function fills the `integer(kind=4), dimension(6):: has_cl` array with flags describing which power spectra are needed by the likelihood compute function (see *The input of the compute function* *are multipoles of the power spectra and nuisance parameters.*). The first argument of the function must be the return value from a previous call to `clik_init()`. In case of error the program exit with an explaining message.

subroutine **`clik_get_lmax`**(clikid, lmax)
This function fills the array `integer(kind=4), dimension(6):: lmax` with the lmax value for each power spectra needed by the likelihood compute function (see *The input of the compute function are multipoles of the power spectra and nuisance parameters.*). The first argument of the function must be the return value from a previous call to `clik_init()`. In case of error the program exit with an explaining message in case of an error.

subroutine **`clik_get_extra_parameter_names`**(clikid, names, numnames)
This function sets `integer::numnames` to the number of nuisance parameters needed by the likelihood compute function (see *The input of the compute function are multipoles of the power spectra and nuisance parameters.*) and fills with their names the array `character(len=256), dimension(numnames)::names`. This array is allocated by the function and MUST be deallocated by the caller after use. The first argument of the function must be the return value from a previous call to `clik_init()`. In case of error the program exit with an explaining message.

4.2.3 Computing the log likelihood

`real`(kind=8) function *clik_compute(clikid*, cl_and_pars)
This function returns the value of the log likelihood for the parameter vector `cl_and_pars`. The content of this vector is described in *The input of the compute function are multipoles of the power spectra and nuisance parameters.*. The first argument of the function must be the return value from a previous call to `clik_init()`. In case of error the program exit with an explaining message. This function can be called as many time as the user wants.

4.2.4 Cleanup

When a likelihood object is no more needed (i.e. when no more computation will be needed in the program), the memory it uses can be cleaned up calling

subroutine **`clik_cleanup`**(clikid)

The first argument of the function must be the return value from a previous call to `clik_init()`.

4.3 API - lensing likelihood

All codes calling clik functions must

```
use clik
```

The library can initialize more than one likelihood. Likelihood are represented by a variable (in the following, named `clikid`) of type `type(clik_object)`.

4.3.1 Testing whether a file contains a lensing likelihood

One can test whether a file contains a lensing likelihood by calling

`subroutine clik_try_lensing(hdffilepath, is_lensing);`
On return the logical argument `is_lensing` is set to true or false depending whether the `hdffilepath`

argument points toward a lensing likelihood file. In case of error, the library will only print out a message and force the calling program to exit.

4.3.2 Initialization

The library must be initialized by calling

subroutine **`clik_lensing_init`**(clikid, hdffilepath)

The subroutine sets the argument `clikid`, which is of type `type(clik_object)` to a handle on an object containing the definition of the likelihood. It expects two arguments, `hdffilepath` a string containing the path to a likelihood file. In case of error, the library will only print out a message and force the calling program to exit.

4.3.3 Querying the lensing likelihood object

subroutine **`clik_get_lmax`**(clikid, lmax)

On return the integer argument `lmax` take as value the lmax of both the clpp and cltt. The first argument of the function must be the return value from a previous call to `clik_lensing_init()`. In case of error the program exit with an explaining message in case of an error.

subroutine **`clik_lensing_get_extra_parameter_names`**(clikid, names, numnames)

This function sets `integer::numnames` to the number of nuisance parameters needed by the likelihood compute function (see *The input of the compute function are multipoles of the power spectra and nuisance parameters.*) and fills with their names the array `character(len=256), dimension(numnames)::names`. This array is allocated by the function and MUST be deallocated by the caller after use. The first argument of the function must be the return value from a previous call to `clik_init()`. In case of error the program exit with an explaining message.

4.3.4 Computing the log likelihood

`real`(kind=8) function *clik_lensing_compute(clikid*, cl_and_pars)

This function returns the value of the log likelihood for the parameter vector `cl_and_pars`. This vector must have 2*(lmax_lensing+1) + number_of_lensing_extra_parameters elements. They are first the lensing_lmax+1 values of clpp, then the lensing_lmax+1 values of the cltt, the the extra parameter values. The first argument of the function must be the return value from a previous call to `clik_init()`. In case of error the program exit with an explaining message. This function can be called as many time as the user wants.

4.3.5 Cleanup

When a likelihood object is no more needed (i.e. when no more computation will be needed in the program), the memory it uses can be cleaned up calling

subroutine **`clik_lensing_cleanup`**(clikid)

The first argument of the function must be the return value from a previous call to `clik_lensing_init()`.

CHAPTER

FIVE

INTERFACING THE LIBRARY WITH PYTHON

This python library is only available when the optional python tools are installed either by make or waf.

5.1 API - CMB

The module clik contains the wrapper to the clik c library. It contains only one object called `clik`, which is initialized with a string containing the path to a likelihood file.

```
import clik

clikid = clik.clik("clikidfile")
```

The `has_cl`, `lmax` and parameter names array (see *The input of the compute function are multipoles of the power spectra and nuisance parameters.*) can be queried by simpliy reading the `has_cl`, `lmax` and `extra_parameter_names` attributes of the object

```
has_cl = clikid.has_cl
print has_cl
```

A log likelihood is computed by calling the object with a list-like object (`tuple`, `list` of `numpy.ndarray` objects) containing the vector of parameters as described in *The input of the compute function are multipoles of the power spectra and nuisance parameters.*.

```
loglkl = clikid(cl_and_pars)
```

The file `click_example_py.py` gives a simple example of the use of the python API. It is compiled and installed as **clik_example_py**.

5.2 API - lensing

Similarly a lensing likelihood can be initialized by

```
import clik

clikid = clik.clik_lensing("clikidfile")
```

The `lmax` and parameter names array (see *The input of the compute function are multipoles of the power spectra and nuisance parameters.*) can be queried by simpliy reading the `lmax` and `extra_parameter_names` attributes of the object.

The log likelihood is computed by calling

```
loglkl = clikid(cl_and_pars)
```

`cl_and_pars` must be a (lensing_lmax+1)+number_of_extra_parameters elements array. The first lmax+1 elements must be the clpp, the next the cltt. Optionnaly if providing only lmax+1+number_of_extra_parameters the likelihood will be computed using the fiducial cltt spectrum.

CHAPTER

SIX

USING WMAP9 LIKELIHOOD

This utility is only available when the optional python tools are installed using waf.

If the sources for the `wmap9` likelihood are available, one can also use this likelihood code along with the WMAP9 likelihood data file whithin clik. To do so, one must also create likelihood files that refers to the WMAP9 dataset. The tool **prepare_wmap** allows to prepare this files. It needs a parameter file describing the location of the WMAP data the description of the range of ells to use and a few flags. Example parameter files are present in the `examples` directory. See for `wmap_full.par` that defines likelihood using all of the WMAP9 data for all ells.

```
# prepare a file for calling the wmap full likelihood through clik

res_object = wmap_9_full.clik

ttmin = 2
ttmax = 1200
temin = 2
temax = 1200
use_gibbs = 0
use_lowl_pol = 1

wmap_data = /THE/PATH/TO/wmap_likelihood_v5
cl_save = wmap_9_full.cltest
```

CHAPTER SEVEN

USING ACTPST LIKELIHOOD

This utility is only available when the optional python tools are installed either by make or waf.

Source for the actspt likelihood described in (CITE) are included in the clik package. To prepare a likelihood file, one has to run the too **prepare_actspt** using a parameter file similar to

```
# prepare a file for calling the wmap full likelihood through clik

res_object = actspt.clik
actspt_data =  path/to/the/data

#optional
# l ranges for the act data
lmin11 = 1000
lmin12 = 1500
lmin22 = 1500
lmax11 = 10000
lmax12 = 10000
lmax22 = 10000

#lmax used for the CMB
tt_lmax_mc = 5000

# add or remove datasets
use_act_south = 1
use_act_equa = 1
use_spt_highell = 1

# set to one to copy the data in the clik file, 0 to simply refer to it.
include = 1
```

CHAPTER

EIGHT

PLAYING AROUND WITH LIKELIHOOD FILES

8.1 Computing a log likelihood from the command line

The example codes, **clik_example_C**, **clik_example_f90** and **clik_example_py** allow to compute a the log likelihoods for any numbers of files containing Cls andforeground parameters.

clik_example_C *usage:*

```
clik_example_C lkl_file.clik [clfile1 ...]
```

`lkl_file.clik` is the likelihood file. The `clfile1 ...` files must be ascii and contains Cls from 0 to the lmax (included) of the likelihood file, followed by the nuisance parameter values in the order shown when using **clik_print** or using of the the query function (for example, in c **:cfunction:‘clik_get_extra_parameter_names‘**).

The program **clik_example_py** is only available when the optional python tools are installed either by make or waf.

8.2 Printing info about a file

This utility is only available when the optional python tools are installed either by make or waf.

The tool **clik_print** displays some information on the content of a likelihood files. The range of modes for each power spectrum, the list of extra parameters, and for each component of the full likelihood, the same info.

Usage:

```
clik_print somelikelihoodfile.clik
```

`somelikelihoodfile.clik` is a likelihood file.

Example output:

```
$> clik_print ../release/clik_7.4/CAMspec_v6.2TN_2013_02_26.clik/
----
clik version 5869
  CAMspec e61cec87-3a37-43ca-8ed1-edcfcaf5c00a
Checking likelihood '../release/clik_7.4/CAMspec_v6.2TN_2013_02_26.clik/' on test data. got -3910.03
----
clik lkl file =  ../release/clik_7.4/CAMspec_v6.2TN_2013_02_26.clik/
  number of likelihoods = 1
  lmax ( TT = 2500 )
  number of varying extra parameters 15
```

```
    A_ps_100
    A_ps_143
    A_ps_217
    A_cib_143
    A_cib_217
    A_sz
    r_ps
    r_cib
    n_Dl_cib
    cal_100
    cal_143
    cal_217
    xi_sz_cib
    A_ksz
    Bm_1_1

  lkl_0
    lkl_type = CAMspec
    unit = 1
    TT = [50 , 2500]
    number of extra parameters = 15 ('A_ps_100', 'A_ps_143', 'A_ps_217', 'A_cib_143', 'A_cib_217', 'A
```

8.3 Modifying the content of a likelihood file

This utility is only available when the optional python tools are installed either by make or waf.

The tools **clik_join** and **clik_disjoin** allow to either join toghether one or more likelihood files in a single one, or cut a likelihood files into as many files as it has components.

clik_join *usage:*

```
clik_join lkl_file_1.clik lkl_file_2.clik [lkl_file_3.clik ...] result_lkl_file.clik
```

`lkl_file_1.clik`, `lkl_file_2.clik`... are input likelihood files. The resulting file `result_lkl_file.clik` defines a likelihood file so that the log likelihood a Cl (+extra parameters) is the sum of the log likelihood of each input files.

clik_disjoin *usage:*

```
clik_disjoin lkl_file.clik
```

The input file is `lkl_file.clik` is split in as many likelihood as it has component. Each likelihood is saved in its own file, named `lkl_file.lkl_X.clik` where `X` is a number between 0 and the number of components.

8.4 Dealing with likelihood files with external data

This utility is only available when the optional python tools are installed either by make or waf.

This is only valid for likelihood files containing only one component and when this component is either a BOPIX or WMAP likelihood. In both cases, the likelihood relies on external data. This data is either included in the file (as a big tarfile) or install somewhere in the file system. the tools **clik_extract_external** and **clik_include_external** allows to go from one choice to the other. It is either, when distribution, to include the external data whithin the file, and more efficient to run with the external data installed somewhere in the file system.

clik_extract_external *usage:*

```
clik_extract_external parameterfile
```

Example parameter file

```
input_object = wmap_7_full.clik              # input likelihood file. Data is included
install_path = /data/wmap_likelihood_data    # where to install the data
res_object = wmap_7_full.external.clik       # output likelihood file. Data is no more included
```

clik_include_external *usage:*

```
clik_include_external parameterfile
```

Example parameter file

```
input_object = wmap_7_full.external.clik   # input likelihood file. Data is installed somewhere
res_object = wmap_7_full.clik              # output likelihood file. Data is included
```

8.5 Extracting the test Cl from a likelihood file

This utility is only available when the optional python tools are installed either by make or waf.

clik_get_selfcheck *usage:*

```
clik_get_selfcheck lkl_file.clik clfile
```

`lkl_file.clik` is the likelihood file. `clfile` is the cl+nuisance parameter array used to compute the selfchek displayed at each initialization of the likelihood. Same format as the one needed for **clik_example_C**

8.6 Computing a slice through a log likelihood from the command line

This utility is only available when the optional python tools are installed either by make or waf.

One can quickly compute conditionals through a likelihood along the direction of one of the nuisance parameter using **clik_explore_1d**.

clik_explore_1d *usage:*

```
clik_explore_1d parfile
```

`parfile` is a parameter file similar to:

```
# slice

#lkl
input_object = CAMspec_v6.2TN_2013_02_26.clik

#data for the other dimensions. Same format as for clik_example_C.
initdata = bestfilcl.camspec

#name of the varying parameter
parameter = r_cib

#begin and end values
beg = -1
```

```
end = 1.5

#number of computations
step = 300

#ascii file that will hold the result as a 2d array, parameter value, lkl value
res = myresult.txt
```

CHAPTER

NINE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

INDEX