ECLIPSE User Guide

A fast Quadratic Maximum Likelihood Estimator for CMB intensity and polarization power spectra.

Version 1.02

Juan Daniel Bilbao Ahedo

Instituto de Física de Cantabria
CSIC - Univ. de Cantabria
Edificio Juan Jorda
Avenida de los Castros
39005 Santander
Spain

May 2022

Contents

Chapter 1

Introduction

The code ECLIPSE (Efficient Cmb poLarization and Intensity Power Spectra Estimator) implements an optimized version of the Quadratic Maximum Likelihood (QML) method for the estimation of the power spectra of the Cosmic Microwave Background (CMB) from masked skies. It is written in Fortran language and designed to run parallel on many MPI tasks. Therefore, ECLIPSE can be used in a personal computer but also benefits from the capabilities of a supercomputer to tackle large scale problems.

The QML method was first introduced in [1] for intensity and in [2] for intensity and polarization. ECLIPSE implements a very efficient approach of QML, as described in [3].

There are three versions of the code

- ECLIPSE_TEB
- ECLIPSE_EB
- ECLIPSE_T

The first version computes the power spectrum of intensity and polarization, the second implementation computes the polarization spectrum and the last version provides the spectrum of intensity. In addition, the different implementations also differ in the first multipole computed by each of the codes. The first two versions — which compute polarization spectrum— start at $\ell = 2$, while ECLIPSE_T starts at $\ell = 1$, that is, it allows us to obtain the dipole in intensity maps.

The three versions analyze masked CMB maps, in which the signal can be affected by the beam and pixel window functions. The masks of intensity and polarization can be different and the noise can be isotropic or anisotropic. The program can estimate auto and cross-correlation power spectrum, that can be binned or unbinned.

The latest version of the ECLIPSE source code, documentation, and example programs are available on GitHub

https://github.com/CosmoTool/ECLIPSE

Any questions, bug reports, or suggested enhancements related to the ECLIPSE code should be sent to

bilbao@ifca.unican.es

Acknowledgments

The development of this User Guide was made possible through collaboration and especially significant contributions from Ana Belén Hoya, Jaime Finat and Belen Barreiro.

Chapter 2

Building and running ECLIPSE

As an illustration, we will show in detail how to compile and run the code in a Linux personal computer as well as in the NERSC[1] and Altamira[2] supercomputers. Future versions of this document will incorporate additional systems.[3] Note that we do not intend to be exhaustive or to explore all possible options, but rather show some examples in which the code can be built and run. The essential point is that it meets the Fortran, ScaLAPACK, BLACS and FITSIO standards, so it can run in any machine and configuration compatible with them.

2.1 Building the code

The program is written in Fortran language and uses functions of the ScaLAPACK and BLACS libraries. In addition, it works with maps in FITS format. Thus the FITSIO[4] library is also needed.

Regarding ScaLAPACK, BLACS and MPI libraries, these are usually installed in supercomputers, as is the case of NERSC at Lawrence Berkeley National Laboratory and Altamira at IFCA. We offer detailed information about using `ECLIPSE` in both machines in this document. In order to build and run the code in a personal computer equipped with Linux operating system, the required set of libraries, compilers and instructions can be downloaded and installed through Intel® oneAPI Base Toolkit and the HPC Toolkit, among many other options.

2.1.1 Building ECLIPSE on a personal computer with Linux Operating System

We will show how to build `ECLIPSE` assuming that the GCC compiler is installed. To build the code the user can follow these steps

[1] NERSC

[2] Altamira

[3] To run it in a personal computer equipped with Windows operating system, one can install the Linux Subsystem for Windows and operate as if it were a Linux machine.

[4] The latest version of the CFITSIO source code, documentation, and example programs are available on: `http://heasarc.gsfc.nasa.gov/fitsio`

1. Build the FITSIO library. To do so, the user has to download the software and follow the instructions. For example, the `FITSIO` library can be built on Linux systems by typing

   ```
   > ./configure --prefix=your_installation_path [--enable-sse2] [--enable-ssse3]
   > make
   > [sudo] make install
   ```

   After these steps, the FITSIO libraries should be built and installed in the selected folder. In this section we will assume that `your_installation_path` is the usual `/usr/local/lib`.
2. In the next step the libraries ScaLAPACK and BLACS and the MPI system must be installed on the computer. For example, we can install Intel® oneAPI Base toolkit and HPC Toolkit. In this section, we assume that the software development package is located at `/opt/intel/oneapi/`.
3. As pointed out in oneAPI HPC Toolkit instructions, once this is installed, the `setvars.sh` script must be executed (sourced) to configure the local environment variables in order to reflect the needs of the installed Intel oneAPI development tools. In our case

   ```
   > source /opt/intel/oneapi/setvars.sh
   ```

   Steps 1 and 2 must be done only once. The step 3 script has to be sourced every time the Linux Bash Shell is launched.
4. `ECLIPSE` code can be compiled and linked by launching a script that accompanies the code[5]

   ```
   > ./cipLinux.sh ECLIPSE_TEB
   ```

   Im order to compile any other version of the code, `ECLIPSE_T` or `ECLIPSE_EB`, one has to type something of the sort

   ```
   > ./cipLinux.sh ECLIPSE_T
   ```

   The script contains the code

   ```
   mpiifort "$1".f90 -o "$1" -O3  -I"${MKLROOT}/include"
   -L/usr/local/lib/libcfitsio.a
   ${MKLROOT}/lib/intel64/libmkl_scalapack_lp64.a -Wl,
   --start-group ${MKLROOT}/lib/intel64/libmkl_intel_lp64.a
   ${MKLROOT}/lib/intel64/libmkl_intel_thread.a
   ${MKLROOT}/lib/intel64/libmkl_core.a
   ${MKLROOT}/lib/intel64/libmkl_blacs_intelmpi_lp64.a
   -lcfitsio -Wl,--end-group -liomp5 -lpthread -lm -ldl
   ```

   Bear in mind the root path for `libcfitsio.a` is assumed to be `/usr/local/lib`. Modify your script accordingly should it be placed in any other folder.

[5] There is another slightly different script, `cipLinux_Alt.sh`, in the repository. It can be useful, if the previous one does not work.

There are many other options to compile the code: one can choose among multiple configurations and find the adequate instruction at Intel® oneAPI Math Kernel Library Link Line Advisor.

If all these steps are successful, an executable version of ECLIPSE_TEB will be built. This can be tested by writing

```
> mpirun -np 4 ./ECLIPSE_TEB
```

This should give the following error messages

```
ERROR, TWO COMMAND-LINE ARGUMENTS REQUIRED, STOPPING
ERROR, TWO COMMAND-LINE ARGUMENTS REQUIRED, STOPPING
ERROR, TWO COMMAND-LINE ARGUMENTS REQUIRED, STOPPING
ERROR, TWO COMMAND-LINE ARGUMENTS REQUIRED, STOPPING
```

which indicates that we have launched the code incorrectly, but that it was successfully built.

2.1.2 Building ECLIPSE on the NERSC supercomputer

NERSC provides compiler wrappers on Cori which combine the native compilers (Intel, GNU, and Cray) with MPI and other libraries. In this section we will show how to compile ECLIPSE using the Cray compiler wrappers, the default (and recommended) compiler wrappers on the Cori system.

When the user connects to NERSC, the computer sets the environment variables. Thus, to build ECLIPSE one just has to execute the script cCrayNERSC.sh

```
> ./cCrayNERSC.sh ECLIPSE_TEB
```

The code in the script is

```
ftn "$1".f90 -o "$1"
-L/usr/common/software/cfitsio/3.47/lib -lcfitsio
-Wl,-R//usr/common/software/cfitsio/3.47/lib
```

The script points out to the actual location of the FITSIO libraries in NERSC.

After following these steps, an executable version of ECLIPSE_TEB will be built in our folder in NERSC.

2.1.3 Building ECLIPSE on the Altamira supercomputer

Altamira is an HPC cluster that belongs to the RES (Spanish Supercomputing Network) and is located at the University of Cantabria. The main compute nodes are IBM dx360 and have two Intel Sandybridge E5-2670 processors, each one with 8 cores operating at 2.6 GHz and a cache of 20MB, 64 GB of RAM memory (i.e. 4 GB/core) and 500 GB local disk.

Altamira provides with all required software. To build `ECLIPSE`, follow these steps

1. Load Intel software in Altamira.[6]

   ```
   > source  /gpfs/res_apps/INTEL/oneapi/2021.3/setvars.sh
   ```

2. The next step is to compile the code

   ```
   > ./cipAltamira.sh ECLIPSE_TEB
   ```

   The script points out to the actual location of the FITSIO libraries in Altamira.

The previous steps will build an executable version of the code called `ECLIPSE_TEB`.

2.2 Running the code

The code is designed to be executed in parallel and makes use of BLACS,[7] a message-passing library designed for linear algebra in which the computational model consists of a one- or two-dimensional process grid, where each process stores pieces of the matrices and vectors. Therefore, in order to execute the program, the number of MPI processes on which to run the executable and the dimensions of the grid —rows and columns of MPI processes— need to be specified.

The user also has to indicate the file from which to read the configuration data. That file specifies all the aspects of the computation, such as the folder from which to read the data, the resolution of the maps, the $\ell_{\max}$ up to which to estimate the power spectrum, etc.

Let us suppose that we are running the code from a folder that contains a subfolder named `NSide16` in which we have saved a configuration file named `Satellite.ini`. If we want to run the program in a personal computer using six MPI processes distributed in a grid made of two rows of three columns of processes, we must type the sentence

```
> mpirun -np 6 ./ECLIPSE_TEB 2 NSide16/Satellite.ini
```

where `-np 6` indicates that we want to compute in parallel using six MPI processes, and the number `2` means that we want to distribute them in two rows —therefore the number of columns will be `3`—. The string `NSide16/Satellite.ini` indicates the location and name of the configuration file. Details about the configuration file and how to determine the required number of processors are given in sections 4 and 6, respectively.

Below we show in more detail how to execute the code in a personal computer, NERSC and Altamira. For the last two, we show only some of the possibilities since we do not aim to be exhaustive about all the options offered by those machines. For further detail, we refer the interested reader to NERSC - Running Jobs and Altamira Users Guide.

[6]At the moment this manual was written, the chosen Intel compiler version was INTEL/2021.3. Check Altamira website to find the latest or best match for your configuration.

[7]BLACS

2.2.1 Running ECLIPSE on a personal computer with Linux Operating System

As previously explained, before running the program, the user has to launch the script to properly configure the local environment variables

```
> source /opt/intel/oneapi/setvars.sh
```

The following order executes the program

```
> mpirun -np n_procs ./ECLIPSE_TEB n_rows <Folder>/<ParamsFile>.ini
```

where `n_procs` is the number of MPI processors, `n_rows` is the number of rows and `Folder` and `ParamsFile.ini` indicate the names of the folder and the parameters file for the considered problem. The number of rows has to be equal or a divisor of the number of processors.

2.2.2 Running ECLIPSE on the NERSC supercomputer

Cori at NERSC is a Cray XC40 comprised of two type of processor nodes, Haswell and KNL. The following script, `NERSCJob.sh`, shows an example on how to submit a job to be executed on the Haswell nodes

```
#!/bin/bash
#SBATCH --job-name=ECLIPSE
#SBATCH --output=ECLIPSE64.out
#SBATCH --error=ECLIPSE64.err
#SBATCH --qos=regular
#SBATCH --time=03:00:00
#SBATCH --nodes=3
#SBATCH --tasks-per-node=32
#SBATCH --constraint=haswell
#SBATCH --mail-user=<user>@<domain>.com
#SBATCH --mail-type=ALL

srun ./ECLIPSE_TEB 12 NSide64/TEB.ini
```

In the above script, the 32 physical processors of 3 Haswell nodes are reserved. The 96 cores are distributed in a grid of 12 rows and 8 columns ($96/12 = 8$). The job is submitted to the `Regular` queue, the standard queue for most production workloads. Other lines of the script set typical job directives such as the process name, the console output files names, the maximum walltime requested... (more details about `slurm` directives can be found in Slurm Workload Manager).

To submit the job the user has to type

```
> sbatch NERSCJob.sh
```

Let us mention that the previous example corresponds with a case where the intensity and polarization power spectrum of 1000 simulated masked maps at resolution $N_{\rm side} = 64$ with 29009 observed pixels was estimated up to $\ell_{\rm max} = 192$. The full process took 63 minutes.

To see how the number of operations scales approximately with the parameters of the problem, see Table **??** of [3]. In section 6 we offer some indications to determine the memory and the number of cores required by a computation.

2.2.3 Running ECLIPSE on the Altamira supercomputer

As previously explained, before running the program in Altamira, the user has to launch the script to configure the local environment variables

```
> source  /gpfs/res_apps/INTEL/oneapi/2021.3/setvars.sh
```

An example of a script, `AltamiraJob.sh`, to execute the code in Altamira is shown below

```
#!/bin/bash
#
#SBATCH --job-name=qml
#SBATCH --output=qmlsrun.out
#SBATCH --error=qmlsrun.err
#SBATCH --ntasks=64
#SBATCH --mem-per-cpu=3500
#SBATCH --time=1:30:00
# From here the job starts
srun ./ECLIPSE_TEB 8 NSide64/TEB.ini
```

In this example, the user wants to run the code using 64 cores distributed on a grid of 8 rows and 8 columns.

To submit the job the user has to type

```
> sbatch AltamiraJob.sh
```

Let us mention that the previous example correspond to a case where the intensity and polarization power spectrum of 5000 simulated masked maps at resolution $N_{\rm side} = 64$ with 29009 observed pixels was estimated up to $\ell_{\rm max} = 192$. The full process took 51 minutes.

Chapter 3

The QML estimator and ECLIPSE functionalities

Before describing how to use `ECLIPSE` and the files the code reads and write, let us describe the mathematics of the Quadratic Maximum Likelihood (QML) estimator.

3.1 General description of the QML

The QML is a method for obtaining an optimal estimation of the CMB power spectra and its covariance matrix from a map, which is well suited to deal with incomplete sky coverage. Assuming that the CMB fluctuations are Gaussian and isotropic, [1] and [2] show that given a CMB temperature map or CMB temperature and polarization maps $\mathbf{x}$, an estimation of the power spectra can be found in a two-step process

1. Starting from the pixels in the map, compute an angular quantity y_i that is related to the power spectrum.
2. Given this quantity, define an estimator of the power spectrum.

Before describing the method, let us first establish some basic notation (details can be found in [2]). In the only-temperature case, the map $\mathbf{x}$ is an N-dimensional vector of elements $T_i = T(r_i)$; in the full case, the map is a $3N$-dimensional vector of values T_i, Q_i, U_i.[1] The method requires a model of the signal and of the noise, that are both introduced through the signal $\mathbf{S}$ and the noise $\mathbf{N}$ covariance matrices, respectively.

The statistical properties of $\mathbf{x}$ are characterized by the power spectra C_i, and assuming that the signal and the noise are uncorrelated, we have

$$\mathbf{C} \equiv \langle \mathbf{x}\mathbf{x}^t \rangle = \mathbf{S} + \mathbf{N} = \sum_i C_i \mathbf{P}_i + \mathbf{N}. \tag{3.1}$$

Note that in the only-temperature case the index i can be directly substituted by the multipole index ℓ, while in the full case i includes ℓ and one of the six pairs TT, EE, BB,

[1]Actually, `ECLIPSE` can work with a different number of pixels in intensity and polarization, but for simplicity we will assume that they are the same.

TE, TB, EB; thus $C_i \leftrightarrow C_\ell^{XY}$, XY $\in$ {TT, EE, BB, TE, TB, EB}. The $\mathbf{P}_i$ matrices (see eq. (**??**) of [3]) connect the covariance in the harmonic space to the covariance in the pixel space. Each of them is the product of some subset of the columns of the matrix of the spherical harmonics by their transpose. Naturally, for each value of ℓ we have six matrices $\mathbf{P}_i \leftrightarrow \mathbf{P}_\ell^{XY}$ in the full case.

When estimating the power spectra, one usually needs to explicitly consider the effects of the instrumental beam and of the pixel window function. In addition, it is also quite common to describe the angular power spectra per logarithmic interval as $D_\ell = \ell(\ell+1)C_\ell/2\pi$. Therefore, it may be convenient to implement the QML method in terms of the D_ℓ variables and/or include the instrumental resolution effects. This can be easily done by introducing some additional factors in the $\mathbf{P}_i$ matrices of eq. (3.1).

Let us first denote by B_ℓ the beam and pixel instrumental effects, such that the harmonic coefficients (see appendix **??** of [3]) of the observed signal are given by $a_{\ell m}^{\text{Observed}} = B_\ell a_{\ell m}^{\text{Signal}}$. Analogously, let us define $W_i = B_\ell^X B_\ell^Y$, which encodes these effects in the power spectra. In this way, we can write the covariance matrix of the observed (smoothed) signal as

$$\mathbf{S} = \sum_i C_i W_i \mathbf{P}_i. \tag{3.2}$$

We can also write the previous equation in terms of the D_i variables

$$\mathbf{S} = \sum_i D_i \frac{2\pi}{\ell(\ell+1)} W_i \mathbf{P}_i = \sum_i D_i \check{\mathbf{P}}_i, \tag{3.3}$$

where we have defined the new matrices $\check{\mathbf{P}}_i$.

In order to estimate the power spectrum of a map $\mathbf{x}$ in terms of the variables D_i of eq. (3.3), we first need to obtain the angular quantity y_i related to the anisotropies of the map but translated to the harmonics space, which is achieved by

$$y_i \equiv \mathbf{x}^t \mathbf{E}_i \mathbf{x} - b_i, \tag{3.4}$$

where

$$\mathbf{E}_i \equiv \frac{1}{2} \mathbf{C}^{-1} \check{\mathbf{P}}_i \mathbf{C}^{-1} \tag{3.5}$$

and b_i takes into account the presence of noise

$$b_i = \operatorname{tr} \mathbf{N}\mathbf{E}_i. \tag{3.6}$$

Arranging the sets of D_i and y_i in the vectors $\mathbf{d} = \{D_1, D_2, \dots\}$ and $\mathbf{y} = \{y_1, y_2, \dots\}$ respectively, ref. [1] shows that the intermediate power coefficients y_i are related to the power spectrum as

$$\langle \mathbf{y} \rangle = \mathbf{F}\mathbf{d} \tag{3.7}$$

and the covariances satisfy

$$\langle \mathbf{y}\mathbf{y}^t \rangle - \langle \mathbf{y} \rangle \langle \mathbf{y} \rangle^t = \mathbf{F}, \tag{3.8}$$

where $\mathbf{F}$ is the Fisher information matrix, which, from eq. (3.1) and (3.3), can be written as

$$\mathbf{F}_{ii'} = \frac{1}{2} \operatorname{tr} \left[\mathbf{C}^{-1} \check{\mathbf{P}}_i \mathbf{C}^{-1} \check{\mathbf{P}}_{i'} \right]. \tag{3.9}$$

If $\mathbf{F}$ is regular, the power spectrum estimator can be defined as

$$\hat{\mathbf{d}} \equiv \mathbf{F}^{-1}\mathbf{y}. \tag{3.10}$$

Combining this definition with eq. (3.7), we get $\langle\hat{\mathbf{d}}\rangle = \mathbf{d}$. The covariance matrix of this estimator is the inverse of the Fisher matrix

$$\langle(\hat{\mathbf{d}} - \mathbf{d})(\hat{\mathbf{d}} - \mathbf{d})^t\rangle = \mathbf{F}^{-1}[\langle\mathbf{y}\mathbf{y}^t\rangle - \langle\mathbf{y}\rangle\langle\mathbf{y}\rangle^t]\mathbf{F}^{-1} = \mathbf{F}^{-1}. \tag{3.11}$$

Therefore, the estimator is unbiased and, by the Cramer-Rao inequality, of minimal error bars: QML is mathematically equivalent to the Maximum Likelihood Estimator, but, since it does not require a brutal force maximization, it reduces the computational costs.

Note that from eq. (3.1) and (3.3) the covariance matrix of the signal $\mathbf{S}$ is computed from the fiducial model the user provides with. Therefore, an initial guess for the sought power spectra is required in order to compute the QML estimator. This leads naturally to the possibility of using an iterative scheme in order to update the initial fiducial model and, thus, to improve the final estimation of the power spectra.[2]

It is essential to mention that `ECLIPSE` incorporates an implementation of the method such that the covariance matrix of the signal on the observed maps is computed from the last term of eq. (3.3). Therefore, the fiducial model needed to compute the covariance matrix is given in the form of the variables D_i, and must reflect the power in the signal. In parallel, `ECLIPSE` estimates the power spectra of the signal of the maps —corrected from the beam and pixel effects— in format D_i.[3]

The QML estimator requires matrices $\mathbf{C}$ and $\mathbf{F}$ to be regular. If necessary, the covariance matrix can be regularized by adding a small amount of noise (a detailed analysis on the conditions on which $\mathbf{C}$ is regular can be found in [4]). If the Fisher matrix is singular, an optimal binned QML can be implemented as described below.

3.1.1 The binned estimator

In section **??** of [3] we describe an optimal binned estimator of the power spectrum

$$\hat{\mathbf{b}} = [\mathbf{R}^t\mathbf{F}\mathbf{R}]^{-1}\mathbf{R}^t\mathbf{y}, \tag{3.12}$$

where $\hat{\mathbf{b}}$ is the vector of the estimated binned power spectrum, $\mathbf{F}$ is the Fisher matrix and $\mathbf{y}$ is the vector of elements y_i of eq. (3.4). The matrix $\mathbf{R}$, computed internally by `ECLIPSE`, is defined as

$$R_{ib} = \left\{ \begin{array}{ll} f_i^b & \text{if} \quad \ell \in L_b \\ 0 & \text{otherwise} \end{array} \right. , \tag{3.13}$$

where i refers to the index of the multipoles, b to the index of the bins and L_b is the collection of indexes i in the bin, that is, a collection of some consecutive ℓ's of some of the six components of the power spectrum. The factors f_i^b are

[2] `ECLIPSE` does not implement an iterative scheme. Details about iterative QML can be found in section **??** of [3].

[3] Section 3.2.5 shows how to work in terms of the observed power in the maps for both fiducial and estimated power spectrum.

$$f_i^b = D_i / B_b, \tag{3.14}$$

where D_i is the fiducial power spectrum provided by the user and B_b is the fiducial power-band, computed from the fiducial spectrum. In the configuration file there is a set of `key = value` pairs which defines how B_b will be computed.

Considering

`Type_of_Bin_Center = 1`

the values B_b would be

$$B_b = \frac{\sum_{i \in L_b} \frac{D_i}{(\Delta D_i)^2}}{\sum_{i \in L_b} \frac{1}{(\Delta D_i)^2}} \tag{3.15}$$

and the positions of the centers of the bins

$$\ell_b^* = \frac{\sum_{i \in L_b} \frac{\ell(i)}{(\Delta D_i)^2}}{\sum_{i \in L_b} \frac{1}{(\Delta D_i)^2}}, \tag{3.16}$$

where ΔD_i are the theoretical errors [5] taking into account cosmic variance, noise and sky fraction. $\ell(i)$ makes reference to the value of ℓ that corresponds to each index i.[4]

In the case of `Type_of_Bin_Center = 0` the means above are computed assuming $\Delta D_\ell = 1$.

The Fisher matrix of the binned spectrum, which we will call $\mathbf{G}$, is computed from the Fisher matrix of the multipoles and the matrix $\mathbf{R}$

$$\mathbf{G} = \mathbf{R}^t \mathbf{F} \mathbf{R}. \tag{3.17}$$

The covariances of the binned spectrum are

$$\langle (\hat{\mathbf{b}} - \mathbf{b})(\hat{\mathbf{b}} - \mathbf{b})^t \rangle = [\mathbf{R}^t \, \mathbf{F} \, \mathbf{R}]^{-1}, \tag{3.18}$$

that is, they are equal to the inverse of the Fisher matrix of the estimated binned spectrum.

The two binning options in ECLIPSE

Although the optimal binned estimator is achieved being the factors f_ℓ^b as defined if eq. (3.14), there is a key in the configuration file that tells `ECLIPSE` how to compute them. If `Type_Of_Grouping = 1`, eq. (3.14) is used. If `Type_Of_Grouping = 0`, the code assumes that the non null values of f_ℓ^b are equal to 1. This option allows to instruct `ECLIPSE` to compute the binned estimation without using the information in the fiducial spectrum.

[4] For example, in the case of an intensity and polarization estimation there are six different values of i that point to the same value $\ell(i)$, one for each component of the power spectrum.

3.2 How ECLIPSE works

The code computes some of the expressions of the section above, although some of them are not calculated as described in this document but rather as indicated in the appendix **??** of [3], where they are computationally more efficient. In parallel to the description of the computations, we will refer to some of the files the code reads an writes; details about the type of data and format of the files are given in another section. The code deals with double precision floating point numbers.

3.2.1 Covariance matrix

$$\mathbf{C} = \mathbf{S} + \mathbf{N} \tag{3.19}$$

To compute the covariance matrix of the signal of the maps $\mathbf{S}$, eq. (3.3), the user has to provide `ECLIPSE` with the following data

- The fiducial power spectrum.
- The value of the beam of the experiment.
- The set value of two keys in the configuration file: one which determines whether the maps are affected by the pixel window of the resolution and a second one which defines the path to the pixel window data files.

Eq. (3.3) shows that the fiducial spectrum `ECLIPSE` expects is the **fiducial model of the signal in the maps**, variables $\mathbf{D}_\ell = \ell(\ell+1)C_\ell/2\pi$, as pointed out in section 3.1. The value of the beam and whether the maps are smoothed by the pixel window function of the resolution are determined by two keys in the configuration file. Using this information, the code computes W_i.[5] Of course, matrices $\mathbf{P}_i$ are computed taking into account N_{side} and the masks.

All the versions of `ECLIPSE` expect a fiducial containing the six components of the power spectrum. That is, the components TB and EB can be different from zero. If the code detects not null values, it computes their contribution to the covariance matrix.

In all three versions of `ECLIPSE` the noise matrix $\mathbf{N}$ of (3.1) is diagonal, made of uncorrelated noise. It can be isotropic or anisotropic. In the isotropic case the values of the noise per pixel for both intensity and polarization are read from the configuration file whereas, in the case of anisotropic noise, the code reads the data from a file pointed to by the user in the configuration file.

3.2.2 Anisotropies in the harmonics space

The code computes and saves, in this order

1. The part $\mathbf{x}^t\mathbf{E}_i\mathbf{x}$ of eq. (3.4) and saves the data in the file **CoupledPower.dat**.

[5]As pointed out, in section 3.2.5 is shown how to work in terms of the observed power in the maps for both fiducial and estimated power spectrum.

2. The noise bias b_i as described in eq. (3.6) and saves the data in the file **NoiseBias.dat**.
3. The Fisher matrix, following eq. (3.9), and saves the data in the file **FisherMatrix.dat**.

Although in the next step the code computes the (un)binned power spectrum, making use of the information in the three files above the user can compute the unbinned estimation of the power spectrum using any other software, following eq. (3.10). Alternatively, the user can compute the binned estimation by their own means following the steps described in section 3.1.1.

In the next section we will illustrate how to configure `ECLIPSE` with the user's preferred type of estimation.

3.2.3 Unbinned or binned estimated power spectrum

Depending on the value of the key `Binned` in the configuration file, the code computes the binned or unbinned power spectrum.

If `Binned = 0`, the code computes and saves

1. The deconvolved power spectrum of the maps, using the estimator of eq. (3.10), and saves the data in the file **Dl.dat**. Depending on the value of the key `Remove_Noise_Bias`, the code computes $\mathbf{c} = \mathbf{F}^{-1}\mathbf{y}$ assuming $y_i \equiv \mathbf{x}^t\mathbf{E}_i\mathbf{x} - b_i$, if the key takes value `1`, or $y_i \equiv \mathbf{x}^t\mathbf{E}_i\mathbf{x}$, if the key is `0`. That is, the user can tell `ECLIPSE` whether to remove the noise bias or not.
2. If the number of maps to process is higher than one, the code also computes the mean and dispersion of the estimated power spectra, and saves the values in the files **MeanDl.dat** and **SigmaDl.dat**.
3. Equation (3.11) states that the covariance matrix of the estimated power spectrum is the inverse of the Fisher matrix. The code computes the square root of the diagonal elements of $\mathbf{F}^{-1}$ and saves the data in the file **FisherErrorDl.dat**.

If `Binned = 1`, the code computes the binned power spectrum. Depending on the value of the variable `Type_of_Bin_Center`, the code estimates the weighted or unweighted mean of the power spectrum. Depending on the value of the variable `Type_of_Grouping`, the code computes the optimal binned spectrum using the information in the fiducial, or else performs a reduction of the Fisher matrix size ignoring the data in the fiducial. The data the code computes and saves are

1. The positions of the bins centers, eq. (3.16), in file **Positions.dat**.
2. The fiducial powerbands, eq. (3.15), in file **BinnedFiducial.dat**.
3. The binned estimated power spectrum, eq. (3.12), in file **BinnedDl.dat**.
4. If the number of maps to process is higher than one, the code also computes the mean and dispersion of the binned estimated power spectra and saves the values in files **MeanBinnedDl.dat** and **SigmaBinnedDl.dat**.
5. The code computes the square root of the diagonal elements of $\mathbf{G}^{-1}$ and saves the data in file **FisherErrorBinnedDl.dat**.

3.2.4 Auto and cross-correlation power spectrum

The program can both compute the auto-correlation power spectra from a set of maps and the cross-correlation power spectra from two different sets of maps. In the auto-correlation case, being m the index of the maps in the files, the code computes the power of the anisotropies in the harmonic space, map by map, as

$$\mathbf{x}_m^t \mathbf{E}_i \mathbf{x}_m. \tag{3.20}$$

To compute cross-correlation, two different files with maps should be provided. Thus, the cross-correlations between the maps in the first file and those in the second file are calculated. Being $\mathbf{x}_m$ the maps loaded from the first file and $\mathbf{z}_m$ the maps from the second file, the code computes

$$\mathbf{x}_m^t \mathbf{E}_i \mathbf{z}_m. \tag{3.21}$$

3.2.5 Signal or observed power spectrum?

As pointed out, `ECLIPSE` loads the fiducial spectrum of the signal on the maps and estimates the deconvolved power spectrum, i.e. the power of the signal on the maps. In some situations the user might simply prefer to work with the power spectrum in the observed maps. From eq. (3.3), setting the variables W_i to value 1 forces the code to work in terms of the power spectrum in the maps. This can be achieved by setting to 0 both the value of the beam and the key that instructs the code about the pixel window function in the configuration file (keys `Beam_FWHM` and `Pixel_Window`). In this configuration both the fiducial and estimated power spectra reflect the observed power in the maps. Once the observed power in the map has been estimated, the user can deconvolve it using adequate factors.

3.2.6 Intensity and polarization masks

`ECLIPSE_TEB` differentiates the masks of intensity and polarization. The other two versions of the code only read the mask that corresponds to the type of data that each version processes.

3.2.7 χ^2 control

If the fluctuations are Gaussian, $\mathbf{x}^t\mathbf{C}^{-1}\mathbf{x}$ follows a χ^2 probability distribution and, on average, is expected to have a value close to the dimension of the covariance matrix. `ECLIPSE` computes and saves this value in the file **ChiSquare.dat** and shows the mean value on the console output. This number can be used as a test on how the covariance matrix matches the maps. Therefore, this can be used as a test on the fiducial spectrum and the noise model.

3.2.8 The three versions of ECLIPSE

All the equations of section 3.1 are valid for the three versions `TEB`, `EB` and `T`, which compute the same mathematical expressions. However, each version works with matrices and vectors of the size and data related to the type of estimation it computes.

3.2.9 Different limits for the covariance matrix and for the power spectrum

`ECLIPSE` uses different values of the $\ell_{\max}$ to determine the limit up to which to compute the covariance matrix and the limit up to which to estimate the power spectrum. Although both values should be the same, this feature provides with versatility to carry out tests. Two keys in the configuration file keep this information.

Chapter 4

The configuration file

The program reads the configuration file in the first place. It is written in ASCII format and contains the information all three versions of `ECLIPSE` need to launch a computation. It can contain comments and annotations inserted along the document. Comments are preceded by the symbol "#". The information the program reads from the file is in format `key = value`, where `key` indicates the type of information and `value`, the information itself. For a correct reading of the data in the file, it is essential that the character strings `key =` remain unmodified.

Some of the keys are read by all three versions of the code, some others are only read by one or two of the versions. A configuration file can contain more keys than those strictly required by the version of the code which will read it, but never less. A configuration file containing all the required keys can be read by the three versions of the code; depending on which version reads the file, a TEB, EB or T computation will be launched.

4.1 Sections in the file

The file contains the next sections

4.1.1 Principal

With the `keys` and, for example, the `values`

```
#Data folder
Data_Folder = NSide64

#NSide
NSide = 64

#Fiducial spectrum filename
Fiducial_FileName = PlanckModel_Dl.dat

#Lmax covariance matrix
```

```
Lmax_Covariance_Matrix = 128

#Lmax power spectrum
Lmax_Power_Spectrum = 128

#Compute Fisher matrix
# 1 -> Yes
# 0 -> No
Compute_Fisher_Matrix = 1

#Compute power spectrum
# 1 -> Yes
# 0 -> No
Compute_Spectrum = 1

#Binned spectrum?
# 1 -> Yes
# 0 -> No
Binned = 0
```

The information in this section indicates the folder where the program must read and write the data, the `HEALPix` resolution of the maps, the file from which to read the fiducial, the limits $\ell_{\max}$ up to which to compute the covariance matrix and the power spectrum.

The last three keys indicate whether or not to perform certain calculations.

- `Compute_Fisher_Matrix` tells the program whether the Fisher matrix must be computed or not. `Compute_Fisher_Matrix = 0` can be a good choice if one is processing maps using masks, noise and fiducial identical to those used to process a previous map: in this case the Fisher matrix and the elements b_i are the same as the ones formerly computed.
- If `Compute_Spectrum = 0` the code does not compute the power spectrum. It only computes the Fisher Matrix, the coupled power and the noise bias, then saves the data. In this case, the user must compute the binned, eq. (3.12), or unbinned power spectrum, eq. (3.10), by their own means.
- The key `Binned` tells the program whether to compute a binned or unbinned estimation. In the first case, the program reads data from another section of the configuration file.

4.1.2 Maps

With the `keys` and, for example, the `values`

```
#Maps section

#Type of analysis
```

```
# 0 -> Auto-correlation spectrum
# 1 -> Cross-correlation spectrum
Type_of_Analysis = 0

#Number of maps to analyze
Number_of_Maps = 2001

#Type of maps file
#This key is only read by ECLIPSE_T and ECLIPSE_EB
#ECLIPSE_TEB ignores it
# 0 -> Maps of TQU
# 1 -> Maps of only T or only QU
Type_of_Data = 0

#Maps filename
Maps_FileName = MapsTQU.fits

#Maps to cross filename
Maps2Cross_FileName = MapsTQU2Cross.fits

#FWHM Beam (ArcMin)
Beam_FWHM = 131.922678213788

#Pixel window
# 1 -> Yes
# 0 -> No
Pixel_Window = 1
```

The keys tell the program the type of analysis to perform —auto or cross-correlation—, the number of maps to process, the name of the file containing the maps to read, the name of the file containing the maps to cross-correlate (required by the code only if the user wants to estimate cross-correlation), the FWHM size of the Gaussian beam (in arcmin) and whether the power in the maps has been reduced due to the pixel window function of the resolution.

The key related to the data type functions is explained below

- `ECLIPSE_TEB` ignores the data type key since the code will undoubtedly expect maps containing TQU data.
- The codes `ECLIPSE_T` and `ECLIPSE_EB` read it. The key tells whether the file contains maps of TQU data or whether the file contains just T maps or just QU maps. That is, `ECLIPSE_EB` can read QU maps from a file containing just QU maps —in this case the key should be `Type_of_Data = 1`— or from a file containing TQU maps —in this case the key should be `Type_of_Data = 0`—. The key works the same for `ECLIPSE_T`.

If the key `Pixel_Window` has value 1, the program loads the data from the corresponding `HEALPix`[1] file (see [6]), according to the resolution fixed in the section above. In the last

[1]HEALPix

section of the configuration file, the user has to tell the program where to find the files.

4.1.3 Masks

With the `keys` and, for example, `values`

```
#Mask section

#Intensity masks filename
Intensity_Mask_FileName = Masks_Intensity.fits

#Number of mask to be read
Intensity_Mask_NumMap = 1

#Polarization masks filename
Polarization_Mask_FileName = Masks_Polarization.fits

#Number of mask to be read
Polarization_Mask_NumMap = 2
```

The program can use different masks for intensity and polarization. They can be loaded from different files (or from the same one) and the files can contain more than one mask. The program loads the masks located in the positions the keys point to.

4.1.4 Noise

With the `keys` and, for example, `values`

```
#Noise section

#Type of noise
# 0 -> Isotropic
# 1 -> Anisotropic
Type_of_Noise = 1

#If the noise is isotropic, noise per pixel in Intensity
Intensity_Noise = 1.0

#If the noise is isotropic, noise per pixel in Polarization
Polarization_Noise = 0.01

#If the noise is anisotropic, noise maps filename
Noise_Map_FileName = NoiseMaps.fits

#Type of noise maps in the file
```

```
# 0 -> TQU
# 1 -> T or QU
Type_of_Noise_Data = 0

#Remove noise bias
# 1 -> Yes
# 0 -> No
Remove_Noise_Bias = 1
```

This section tells the program the type of noise in the maps. If the noise is isotropic, the program reads the value of the noise dispersion per pixel (i.e., the square root of the diagonal elements of the noise covariance matrix). If the noise is anisotropic, the program reads the name and the type of data in the noise maps file that contains the noise dispersion per pixel. The key `Type_of_Noise_Data` operates in the same way as the key `Type_of_Data` of the maps section.

The last key tells the program whether to subtract the noise bias when estimating the power spectrum.

4.1.5 Binning

With the `keys` and, for example, `values`

```
#Binning section

#File containing the superior limits in \ell of the bins
Binnig_Limits_FileName = BinsLimites.dat

#How to compute the binned fiducial
# 1 -> Weights given by the theoretical error on D_\ell
# 0 -> Weights equal to 1
Type_of_Bin_Center = 1

#How to compact the Fisher matrix: either using the fiducial or not
# 1 -> Using the data in the fiducial: optimal binned estimator
# 0 -> The fiducial is not used
Type_of_Grouping = 0
```

The key `Binnig_Limits_FileName` indicates the file where the superior limits of the bins are to be found. The key `Type_of_Bin_Center` tells the program whether to compute the weighted mean of the fiducial —and, consequently, to estimate the weighted mean of the power spectrum— or whether to compute the arithmetic mean of the multipoles in the bins —and, therefore, to estimate the arithmetic mean of the multipoles—. The key `Type_of_Grouping` tells the program whether to compact the Fisher matrix introducing information from the fiducial model —the optimal binning estimator— or just by setting all the weights to value 1. The meaning of both options is explained in section 3.1.1.

4.1.6 Other information

That includes

```
#Size of the blocks of the distributed matrices
Matrices_Cyclic_Block_Size = 11

#Control of the regularity of the covariance matrix?
# 1 -> Yes
# 0 -> No
Inverse_Covariance_Matrix_Control = 1

#Show memory allocated by matrices?
# 1 -> Yes
# 0 -> No
Show_Memory_Allocated = 1

#Where to find the Pixel Window data
Healpix_Data_Folfer = '/home/<user>/Healpix_3.70/data/'
```

The first key tells the program the size of the blocks in which the matrices are distributed through the grid processors. `ECLIPSE` breaks the matrices into blocks of size `Matrices_Cyclic_Block_Size` and distributes them cyclically through the grid (more information in The Two-dimensional Block-Cyclic Distribution). The value of the key needs to be increased as the size of the matrices increases (with $N_{\rm side}$, with the number of observed pixels and/or with $\ell_{\rm max}$) and has to be such that each processor contains several blocks of the matrices. For example, the parameter cannot be greater than the number of rows in the covariance matrix divided by the number of rows in the grid of processors, otherwise the last rows of processors would not participate in the calculations. The user must take this point into account for two reasons: to achieve the highest efficiency in the calculations and because otherwise the program could crash.

The key `Inverse_Covariance_Matrix_Control` tells the program whether to check the regularity of the covariance matrix as explained in the section 7.1. If we are certain that the matrix is regular, the key can take the value 0. If we are not or just want to check it anyway, its value must be set to 1. The ScaLAPACK functions used to invert the matrices execute a control of the regularity, but our additional control is a good complement and we have observed it to be stronger.[2]

The key `Show_Memory_Allocated` tells the program to show, in different stages of the calculation, the memory allocated by the entire processors grid to store the matrices. This information gives a very good estimate of the memory required by the code in each step. The user must take into account that there are some stages of the computation where the memory used by the program is larger than the one directly allocated by the `ECLIPSE` code: this happens because ScaLAPACK internally allocates temporal memory to compute matricial operations.

[2]In some calculations the two ScaLAPACK functions returned value 0, but the product of the diagonal elements differed significantly from 1, which happens when the covariance matrix is in the limit singular/regular, evaluated in terms of floating point data.

The key `Healpix_Data_Folfer` tells the program where to find the FITS files with the `HEALPix` pixel window function data.[3]

4.2 About the use of the configuration file

The configuration file is multi-purpose. That is, it contains all the keys can be used by the three versions of the code. If the maps file contains TQU columns, the user can execute each versión of `ECLIPSE` by

```
  ECLIPSE_TEB num_rows Folder/Config.ini
```

or

```
  ECLIPSE_EB num_rows Folder/Config.ini
```

or

```
  ECLIPSE_T num_rows Folder/Config.ini
```

and the code will respectively compute the six, three or one components of the power spectrum of the same maps using the specific masks, noise and fiducial spectrum.

On the other hand, the user can write a specific configuration file for any of the three versions of the code, which only needs to contain the keys that version of the code reads.

[3]The FITS files can be found in the folder `HEALPix_Data` at https://github.com/CosmoTool/ECLIPSE

Chapter 5

Input and output data files

In this section we describe in detail the format of the files which `ECLIPSE` works with.

5.1 Input files

The first file the program reads is the configuration file. The name and location are passed to the program when it is launched, as shown in section 2.2.

All the names of the input files are specified in the configuration file, whereas the names of the data files the program saves are fixed. The program expects to find and save them in the folder given by the key `Data_Folder`.

Referred to CMB, masks and noise maps, they are loaded as `HEALPix` FITS files. All maps (CMB, noise or mask) read by the program should be in full-sky format (even if null values are provided for those pixels that are discarded by the mask) and `RING` ordering. `ECLIPSE` first loads the masks for intensity and polarization and selects those pixels with value 1. Later, after loading the CMB and noise maps, only those pixels allowed by the mask will be kept in the computer´s memory.

The files and format the program reads are

5.1.1 Masks

- **Intensity mask file**: the file to which points the `Intensity_Mask_FileName` key, from which the program reads the intensity mask. It is a `HEALPix` maps file —FITS format— with type `REAL` data. The file can have several full sky maps. The maps are made of ones and zeroes (the NullVall of `HEALPix` is admitted, and the program interprets it as zero). The number of masks —maps— in the file must be equal to or greater than the value of the key `Intensity_Mask_NumMap`.
- **Polarization mask file**: the file to which points the `Polarization_Mask_FileName` key, from which the program reads the polarization mask. It has the same format as the previous file. The number of masks in the file must be equal to or greater than the value of the key `Polarization_Mask_NumMap`.

  The code `ECLIPSE_TEB` loads both masks. Each of the other two codes loads the relevant mask and ignores —it does not read— the keys that point to the other mask.

5.1.2 Pixel Window

- **pixel_window_nXXXX.fits**: A data file from **HEALPix**. The program loads the file only when the key `Pixel_Window` takes the value 1. **XXXX** makes reference to the resolution ($N_{\rm side}$) of the maps. The location of the file is shown in the key `Healpix_Data_Folder`.

5.1.3 Fiducial spectrum

- **Fiducial data file**: The program loads data from the file to which the key `Fiducial_FileName` points to. The file contains seven columns of numbers in ASCII format. The first one contains the value of ℓ, the other six contain the six components of the fiducial power spectrum of the signal in the maps, in D_ℓ format. The order of the columns is

$$\ell \quad D_\ell^{TT} \quad D_\ell^{EE} \quad D_\ell^{BB} \quad D_\ell^{TE} \quad D_\ell^{TB} \quad D_\ell^{EB}$$

It is mandatory for the fiducial to include the columns TB and EB. These two columns will typically consist of zeroes but, if the program detects non null values, it will include these spectrum components in the computation of the covariance matrix. The file must have rows from $\ell = 0$ up to at least the value of the key `Lmax_Covariance_Matrix`. The file cannot contain headers, it can just contain numbers. For example, the first rows could be

0	0.	0.	0.	0.	0.	0.
1	50.00	0.	0.	0.	0.	0.
2	1017.64	0.03090	0.000046	2.623	0.	0.
3	963.98	0.03971	0.000047	2.940	0.	0.

The `ECLIPSE_TEB` and `ECLIPSE_EB` versions read data from $\ell = 0$ to the $\ell_{\rm max}$ fixed in the key `Lmax_Covariance_Matrix`, but ignore the values that correspond to $\ell = 0$ and $\ell = 1$. The `ECLIPSE_T` version reads data from $\ell = 0$ as well but, unlike the other two, it saves the value of the spectrum at $\ell = 1$, since it estimates the spectrum from $\ell = 1$ and computes the covariance matrix starting from the same value. This is the reason why the column TT of the example has power at $\ell = 1$.

5.1.4 Noise maps

- **Noise maps file**: The program loads data from files only when the key `Type_of_Noise` gets the value 1. The code expects a FITS file of `HEALPix` maps. The data in the maps represent the noise dispersion per pixel. The file must contain one or two columns of full sky data. In the case of executing `ECLIPSE_TEB` or when `Type_of_Noise_Data = 0` for any version of the code, it will be interpreted that the first column contains the noise dispersion per pixel in intensity and the second one, the polarization noise. In the latter case, `ECLIPSE_EB` loads the second column and ignores the first one and `ECLIPSE_T` loads the first column and ignores the second one. When `Type_of_Noise_Data = 1`, both `ECLIPSE_EB` and `ECLIPSE_T` will specifically read the first column of the file, which then must contain just polarization or just intensity noise respectively.

5.1.5 CMB maps

- **Maps for auto-correlation file**: The program loads maps from the file the key `Maps_FileName` points to, a FITS file of maps in the usual `HEALPix` format. It must include, at least, as many maps as the value of the key `Number_of_Maps`. The maps must be full sky.

  When `ECLIPSE_TEB` loads the maps or when they are loaded by the other two versions of the code and `Type_of_Data = 0`, each map consists of three consecutive full sky columns of T, Q and U. If the file contains multiple maps, the order of the columns is

  $$T_1, Q_1, U_1, T_2, Q_2, U_2 \ldots$$

  When `ECLIPSE_EB` loads the maps and `Type_of_Data = 1`, if the file contains more than one map, the order of the columns is

  $$Q_1, U_1, Q_2, U_2 \ldots$$

  On the other hand, when `ECLIPSE_T` loads the maps and `Type_of_Data = 1`, if the file contains more than one map, the order of the columns is

  $$T_1, T_2 \ldots$$

  Due to the limit fixed in the number of columns in the extensions of the FITS files, each extension of the file to be read by `ECLIPSE` can only have a maximum of 50 maps. Therefore, a maps file must have as many extensions as the result of dividing the number of maps by 50 —plus one, if the number of maps is not a multiple of 50—. Each extension can only contain a maximum of 150 columns if the file is made of maps T, Q and U, a maximum of 100 columns if the maps are Q and U; and 50 columns if the maps are T. If one is interested in simulating maps with their own code to be later processed by `ECLIPSE`, the maps can be adequately saved by using the function `output_map` of `HEALPix`.

  In order to compute cross-correlation spectrum, the program correlates the maps in this file with the maps in the next file of the list.

- **Maps for cross-correlation file**: The maps to be cross-correlated must be saved in a different file. The program correlates the first map in one file (a map with three, two or one columns) with the first map in the other file, the second map with the second, etc. The format of the file that contains the maps to cross-correlate must obviously be the same as the format of the previous file.

5.1.6 Bin limits

- **Bin limits file**: When `Binned = 1`, the program loads data from the file pointed to by `Binning_Limits_FileName`. The data in the file must be in ASCII format and consist of a column of integer numbers in ascending order. Each number in the file indicates the superior limit of the bin, specifically, the value of ℓ^b_{high} defined in section **??** of [3], Independently of the values provided in the file, the superior limit of the last bin is given by the value $\ell_{\max}$ of the key `Lmax_Power_Spectrum`. For example, if it contains the data

```
5
9
13
19
```

and `Lmax_Power_Spectrum` $= 16$ the versions `ECLIPSE_TEB` and `ECLIPSE_EB` will work with bins from 2 to 5, from 6 to 9, from 10 to 13 and from 14 to 16. However, the first `ECLIPSE_T` bin goes from 1 to 5.

5.2 Output files

The program saves the following list of files

5.2.1 QML kernel

The three files which follow contain all the essential information QML generates. They contain the anisotropies coupled power in the harmonic space, the noise power in the harmonic space and the Fisher matrix. From the information registered in the maps the power spectrum can be computed, unbinned (eq. (3.10)) or binned (eq. (3.12)).

All the crucial information can be be found in these files should the user be interested in computing the power spectrum using some other software.

The files are

- **CoupledPower.dat**. An ASCII file that registers the values $\mathbf{x}^t\mathbf{E}_i\mathbf{x}$ in an auto-correlation estimation, or $\mathbf{x}^t\mathbf{E}_i\mathbf{z}$ in a cross-correlation estimation (see eq. (3.4) and eq. (3.21)). That is, it contains the information about the coupled power —signal plus noise— of the anisotropies in the harmonic space. The number of data and structure of the file depend on the version of the code that generates the file
  - `ECLIPSE_TEB`: the file contains a list of $N_{\text{maps}} \times 6 \times (\ell_{\max} - 1)$ elements, where N_{maps} is the number of maps. The data are organized as follows: the first $6 \times (\ell_{\max} - 1)$ numbers are the values $\mathbf{x}^t\mathbf{E}_i\mathbf{x}$ (or $\mathbf{x}^t\mathbf{E}_i\mathbf{x}_{\text{To cross}}$) computed from the first map; the next $6 \times (\ell_{\max} - 1)$ numbers are the values of the second map, etc. The structure within the data of any of the maps is: the first $\ell_{\max} - 1$ numbers are TT data —from $\ell = 2$ to $\ell = \ell_{\max}$—, the next group of $\ell_{\max} - 1$ numbers are EE data, followed by data groups of BB, TE, TB and EB, in this order.
  - `ECLIPSE_EB`: the file contains a list of $N_{\text{maps}} \times 3 \times (\ell_{\max} - 1)$ elements. The data are organized as follows: the first $3 \times (\ell_{\max} - 1)$ numbers are, for example, the values $\mathbf{x}^t\mathbf{E}_i\mathbf{x}$ computed from the first map, the next group contains the values

obtained from the second map, etc. The structure within the data of any of the maps is: the first $\ell_{\max} - 1$ numbers are EE data —from $\ell = 2$ to $\ell = \ell_{\max}$—, the next group of $\ell_{\max} - 1$ numbers are BB data, and the last group of $\ell_{\max} - 1$ numbers are EB data.

 – `ECLIPSE_T`: the file contains a list of $N_{\text{maps}} \times \ell_{\max}$ elements. The data are organized as follows: the first $\ell_{\max}$ numbers are, for example, the values $\mathbf{x}^t\mathbf{E}_i\mathbf{x}$ computed from the first map —ordered from $\ell = 1$ to $\ell_{\max}$—; the next group contains the values obtained from the second map, etc.

- **NoiseBias.dat**. A file in ASCII format. It contains the values of eq. (3.6). Depending on the program that saves the file, the data are
  - `ECLIPSE_TEB`: a list of $6 \times (\ell_{\max} - 1)$ elements. The first $\ell_{\max} - 1$ elements are the TT values —ordered from $\ell = 2$ to $\ell_{\max}$ —. The rest of the data are the EE, BB, TE, TB and EB values, in this order.
  - `ECLIPSE_EB`: a list of $3 \times (\ell_{\max} - 1)$ elements. The first $\ell_{\max} - 1$ elements are the EE values —ordered from $\ell = 2$ to $\ell_{\max}$ —. The rest of the data are the BB, and EB values.
  - `ECLIPSE_T`: a list of $\ell_{\max}$ elements, the values of TT ordered from $\ell = 1$ to $\ell_{\max}$.

- **FisherMatrix.dat**. A file in ASCII format that contains the data given by eq. (3.9). The saved data do not conform a matrix structure, they must be interpreted as a simple list of numbers whose structure depends on the version that saves the file
  - `ECLIPSE_TEB`: the file — the list of numbers — contains $[6 \times (\ell_{\max} - 1)] \times [6 \times (\ell_{\max} - 1)]$ elements. Each group of $6 \times (\ell_{\max} - 1)$ numbers makes up a row of the matrix. The rows (and columns) of the matrix are organized as follows: six groups of $\ell_{\max} - 1$ numbers. The groups are ordered: TT, EE, BB, TE, TB, EB. The $\ell_{\max} - 1$ numbers in each group range from $\ell = 2$ to $\ell_{\max}$.
  - `ECLIPSE_EB`: the file contains $[3 \times (\ell_{\max} - 1)] \times [3 \times (\ell_{\max} - 1)]$ numbers. Each group of $3 \times (\ell_{\max} - 1)$ numbers makes up a row of the matrix. The rows (and columns) of the matrix are organized as follows: three groups of $\ell_{\max} - 1$ numbers. The groups are ordered: EE, BB, EB. The $\ell_{\max} - 1$ numbers in each group range from $\ell = 2$ to $\ell_{\max}$.
  - `ECLIPSE_T`: the file contains $\ell_{\max} \times \ell_{\max}$ numbers. Each group of $\ell_{\max}$ numbers makes up a row of the matrix, from $\ell = 1$ to $\ell_{\max}$.

  `ECLIPSE` saves this file when the key `Compute_Fisher_Matrix` takes the value 1.

5.2.2 Control

The program computes and saves $\mathbf{x}^t\mathbf{C}^{-1}\mathbf{x}$, a list of N_{maps} numbers. This data can be used to check the concordance between the maps and the covariance matrix. If cross-correlation is computed, the program saves $\mathbf{x}^t C^{-1}\mathbf{z}$, where $\mathbf{z}$ is the map to cross. Therefore, another file saved by the program is

- **ChiSquare.dat**. A file in ASCII format made of N_{maps} numbers.

5.2.3 Power spectrum

Depending on the value of the key `Binned`, the program computes the binned or unbinned form of the power spectra of the maps.

Complete spectrum

If `Binned = 0`, the code saves

- **Dl.dat**. A file in ASCII format that contains the deconvolved power spectrum — according to the values of the keys `Beam_FWHM` and `Pixel_Window`— of the maps in terms of the variables D_ℓ. For the three versions, the number of elements and the structure of the file are the same as those of the `CoupledPower.dat` file.
- **MeanDl.dat**. A file in format ASCII that contains the mean value of D_ℓ. The program saves this file when the number of maps is greater than one. The structure of the file is the same as that of the one-map `Dl.dat` file.
- **SigmaDl.dat**. A file in format ASCII that contains the standard deviation of D_ℓ. The program saves this file when the number of maps is greater than one. The structure of the file is the same as that of the `MeanDl.dat` file.
- **FisherErrorDl.dat**. A file in ASCII format that contains the error bar of the estimation, computed from the Fisher matrix. That is, the data are the square root of the diagonal elements of the inverse of the Fisher matrix. The structure of the file is the same as that of the `MeanDl.dat` file.

Binned spectrum

If the binned spectrum has been estimated, the program saves the files described in this section. The data in the files depends on the keys `Type_of_Bin_Center` and `Type_of_Grouping` and are related to the deconvolved variables D_ℓ.

- **Positions.dat**. A file in ASCII format that contains the positions in ℓ of the bins, calculated according to eq. (3.16). Depending on the version, we may have the following cases
  - `ECLIPSE_TEB`: Saves $6 \times N_{\text{bins}}$ elements, in the order TT, EE, BB, TE, TB y EB.
  - `ECLIPSE_EB`: Saves $3 \times N_{\text{bins}}$ elements, in the order EE, BB y EB.
  - `ECLIPSE_TEB`: Saves N_{bins} TT elements.
- **BinnedFiducial.dat**. A file in ASCII format that contains the values of the binned fiducial spectrum computed from eq. (3.15). The number of elements and the structure are the same as those of the `Positions.dat` file.
- **BinnedDl.dat**. A file in ASCII format that contains the binned estimation of the power spectrum of each map, computed from eq. (3.12), where the elements of the matrix $\mathbf{R}$ depend on the value of the key `Type_of_Grouping`. The file contains a sequence of N_{maps} consecutive vectors with the same structure as of the vector saved in `BinnedFiducial.dat`, whatever the version of the code.

- **FisherMatrixBinnedDl.dat**. A file in ASCII format that contains the data given by eq. (3.17). The saved data do not conform to a matrix structure, they must be interpreted as a simple list of numbers whose structure depends on the version that generates the file
  - `ECLIPSE_TEB`. The file contains $[6 \times N_{\rm bins}] \times [6 \times N_{\rm bins}]$ numbers. Each group of $6 \times N_{\rm bins}$ numbers makes up a row of the matrix. The rows (and columns) of the matrix are organized: TT, EE, BB, TE, TB, EB. Each group contains $N_{\rm bins}$ numbers.
  - `ECLIPSE_EB`. The file contains $[3 \times N_{\rm bins}] \times [3 \times N_{\rm bins}]$ numbers. Each group of $3 \times N_{\rm bins}$ numbers makes up a row of the matrix. The rows (and columns) of the matrix are organized: EE, BB, EB. Each group contains $N_{\rm bins}$ numbers.
  - `ECLIPSE_T`. The file contains $N_{\rm bins} \times N_{\rm bins}$ numbers. Each group of $N_{\rm bins}$ numbers makes up a row of the matrix.
- **MeanBinnedDl.dat**. A file in ASCII format that contains the mean value of the binned estimation, with the same structure as that of the `BinnedFiducial.dat` file.
- **SigmaBinnedDl.dat**. A file in ASCII format that contains the standard deviation of the binned estimation, with the same structure as that of the `BinnedFiducial.dat` file.
- **FisherErrorBinnedDl.dat**. A file in ASCII format that contains the error bar on the estimation of the binned power spectrum, computed from the compacted Fisher matrix. That is, the square root of the diagonal elements of the inverse of the matrix of eq. (3.17). The structure of the data is the same as that of the `BinnedFiducial.dat` file.

Chapter 6

Computer requirements

Depending on the maps dimension and the $\ell_{\max}$ up to which to compute the power spectrum, the dimensions of the matrices the code stores can vary significantly. The user needs some rules to determine the memory required and, consequently, the number of processors. The package `ECLIPSE` contains a small program, `ECLIPSE_Memory.f90`, which reads any given configuration file and —according to eq. (6.2)-(6.9)— tells the user the memory required by all three versions of the code to store matrices at two crucial steps of the computation. For example, if typing

```
> ./ECLIPSE_Memory NSide64/Example.ini
```

the program shows

```
Loading configuration from: NSide64/TEB.ini
                       Data_Folder:   NSide64
                             NSide:   64
               Lmax_Power_Spectrum:   192
           Intensity_Mask_FileName:   MascaraSateliteN64.fits
             Intensity_Mask_NumMap:   1
        Polarization_Mask_FileName:   MascaraSateliteN64.fits
          Polarization_Mask_NumMap:   1
  Inverse_Covariance_Matrix_Control:   1
*******************************************************************
Loading masks
Number of observed pixels in temperature:                  29009
Number of observed pixels in polarization:                 29009
Lmax power spectrum:                                         192
Number of spherical harmonics:                             37245
*******************************************************************
*******************************************************************
ECLIPSE_TEB - [Gb]
Eq. (6.1):    112.856905594468
Eq. (6.2):    222.766825564206
Eq. (6.3):    252.822962254286
*******************************************************************
ECLIPSE_EB - [Gb]
Eq. (6.4):    50.1586247086525
Eq. (6.5):    121.678194284439
Eq. (6.6):    190.810735747218
****************************************************************
ECLIPSE_T - [Gb]
Eq. (6.7):    12.5396561771631
Eq. (6.8):    30.4214937761426
Eq. (6.9):    52.8762931823730
Number of spherical harmonics:                              37248
****************************************************************
```

That is, the program reads the configuration file, determines the number of pixels allowed by the intensity and polarization masks and takes the value of $\ell_{\max}$ up to which the power spectrum must be estimated. From these values the program calculates the memory required to store matrices —at two stages of the computation— by all three versions of the code. In the example, if one needs to estimate the six components of the power spectrum in a case determined by the configuration file `Example.ini`, at least 7.36 Gb of memory are required; if one needs to compute the intensity power spectrum, at least 1.17 Gb are required.

Note that these values only consider the memory required to store the matrices, and not the memory required to run the code. However, they can be taken as a very valuable reference to determine the total memory that needs to be requested.

In order to build `ECLIPSE_Memory.f90`, the user has to run the same script which builds `ECLIPSE`. For example, in NERSC

```
> ./cCrayNERSC.sh ECLIPSE_Memory
```

For the interested readed, in the next section we explain where these numbers come from. However, if not interested in the details, one can go directly to section 6.2.

6.1 Determining the memory required

The size of the matrices depends on three principal numbers

- Number of observed pixels in the intensity maps: P_I.
- Number of observed pixels in the polarization maps: P_P.
- Multipole up to which to compute the power spectrum: $\ell_{\max}$.

When the code estimates polarization components, the number of spherical harmonics, L, depends on $\ell_{\max}$

$$L = \sum_{\ell=2}^{\ell_{\max}} 2\ell + 1.$$

The summation starts from $\ell = 1$ when the code estimates only intensity.

In the following sections we detail the critical stages regarding memory requirements for the three versions of `ECLIPSE`.

6.1.1 ECLIPSE_TEB

According to these critical quantities, the number of elements of the blocks and matrices that `ECLIPSE_TEB` stores are

1. Covariance matrix in pixel space:	$(P_I + 2P_P)^2$
2. TT block of the covariance matrix:	$P_I \cdot P_I$
3. TQ type block of the covariance matrix:	$P_I \cdot P_P$
4. QQ type block of the covariance matrix:	$P_P \cdot P_P$
5. TT type block of the spherical harmonics matrix:	$P_I \cdot L$
6. QE type block of the spherical harmonics matrix:	$P_P \cdot L$
7. Matrix $\mathbf{Y}^\dagger\mathbf{C}^{-1}\mathbf{Y}$:	$(3L)^2$
8. One of the nine blocks of the matrix $\mathbf{Y}^\dagger\mathbf{C}^{-1}\mathbf{Y}$:	$L \cdot L$

The elements of the matrices in cases 1, 2, 3 and 4 are real numbers, whereas the elements in cases 5, 6, 7 and 8 are complex numbers.

There are tree stages in the code where the number of stored elements can reache the highest values

- In **Step 2**: when the program is about to compute the inverse of the covariance matrix and the key `Inverse_Covariance_Matrix_Control` takes the value 1. The code requires to store in memory two covariance matrices. The number of reals is

$$2 \cdot (P_I + 2P_P)^2. \tag{6.1}$$

- In **Step 4**: when the program is about to compute the imaginary part of the product $\mathbf{C}^{-1}\mathbf{Y}$.

  At this point, the code stores the blocks TT, TQ, TU, QQ, QU and UU of the inverse of the covariance matrix, the imaginary part of the spherical harmonics matrix and the real and imaginary parts of $\mathbf{C}^{-1}\mathbf{Y}$. The number and type of elements are

  - Blocks TT, TQ, TU, QQ, QU and UU of the inverse of the covariance matrix: $P_I \cdot P_I + 2 \cdot P_I \cdot P_P + 3 \cdot P_P \cdot P_P$ elements, real numbers.
  - Imaginary part of the spherical harmonics matrix: $P_I \cdot L + 4 \cdot P_P \cdot L$ elements, real numbers.
  - Matrix $\mathbf{C}^{-1}\mathbf{Y}$: $(P_I + 2P_P) \cdot (3L)$ elements, complex numbers.

  Therefore, multiplying the number of complexes by two, the amount of real numbers the code stores at this point is

$$P_I \cdot P_I + 2 \cdot P_I \cdot P_P + 3 \cdot P_P \cdot P_P + P_I \cdot L + 4 \cdot P_P \cdot L + 2\left[(P_I + 2P_P) \cdot (3L)\right]. \tag{6.2}$$

- In **Step 6**: when the program is about to compute the blocks EE, BB and EB of $\mathbf{Y}^\dagger\mathbf{C}^{-1}\mathbf{Y}$.

At this point, the code stores the complex numbers of six blocks of the matrix $\mathbf{Y}^{\dagger}\mathbf{C}^{-1}\mathbf{Y}$ — blocks TT, TE, TB, EE, EB and BB—, the real and imaginary parts of the polarization block of the spherical harmonics matrix —blocks QE, QB, UE and UB— and the blocks QE, QB, UE and UB of $\mathbf{C}^{-1}\mathbf{Y}$. The number and type of elements are

– Blocks TT, TE, TB, EE, EB and BB of $\mathbf{Y}^{\dagger}\mathbf{C}^{-1}\mathbf{Y}$: $6 \cdot L \cdot L$ elements, complex numbers.
– Polarization block of the spherical harmonics matrix: $4 \cdot P_P \cdot L$, complex numbers.
– Blocks QE, QB, UE and UB of $\mathbf{C}^{-1}\mathbf{Y}$: $4 \cdot P_P \cdot L$, complex numbers.

Therefore, multiplying the number of complexes by two, the amount of real numbers the code stores at this point is

$$2\left[6 \cdot L \cdot L + 4 \cdot P_P \cdot L + 4 \cdot P_P \cdot L\right] \tag{6.3}$$

To get a first estimate of the required memory, one has to calculate the expressions (6.2) and (6.3) and take the highest value. Since in `ECLIPSE_TEB` the floating point variables are double precision, in order to estimate the number of gigabytes one has to multiply the said highest value by 8 and divide the result by 1024^3.

As an example, in a case where the number of observed pixels in intensity and polarization is 29009 and $\ell_{\max} = 192$, eq. (6.1) takes the value 113 Gb; eq. (6.2), 223 Gb and eq. (6.3), 253 Gb.

6.1.2 ECLIPSE_EB

According to the critical quantities, the number of elements of the blocks and matrices that `ECLIPSE_EB` stores are

1. Covariance matrix in pixel space:	$(2P_P)^2$
2. Spherical harmonics matrix:	$(2P_P) \cdot (2L)$
3. Matrix $\mathbf{Y}^{\dagger}\mathbf{C}^{-1}\mathbf{Y}$:	$(2L)^2$
4. One of the four blocks of the matrix $\mathbf{Y}^{\dagger}\mathbf{C}^{-1}\mathbf{Y}$:	$L \cdot L$

The elements of the matrices in case 1 are real numbers, whereas the elements in cases 2, 3 and 4 are complex numbers.

There are two stages in the code where the number of stored elements reaches the highest values

- In **Step 2**: when the program is about to compute the inverse of the covariance matrix and the key `Inverse_Covariance_Matrix_Control` takes the value 1. The code requires to store in memory two covariance matrices. The number of reals is

$$2 \cdot (2P_P)^2. \tag{6.4}$$

- In **Step 4**: when the program is about to compute the imaginary part of the product $\mathbf{C}^{-1}\mathbf{Y}$.

  At this point, the code stores the inverse of the covariance matrix, the spherical harmonics matrix and the real part of $\mathbf{C}^{-1}\mathbf{Y}$. Therefore, multiplying the number of complexes by two, the amount of real numbers the code stores at this point is

$$(2P_P)^2 + 2 \cdot (2P_P) \cdot (2L) + (2P_P) \cdot (2L). \tag{6.5}$$

- In **Step 6**: when the program is about to compute the blocks EE, BB and EB of $\mathbf{Y}^\dagger\mathbf{C}^{-1}\mathbf{Y}$.

  At this point, the code stores in memory the matrix $\mathbf{C}^{-1}\mathbf{Y}$, the spherical harmonics matrix $\mathbf{Y}$ and three blocks of product $\mathbf{Y}^\dagger\mathbf{C}^{-1}\mathbf{Y}$. Since all values are complexes, the amount of real numbers the code stores at this point is

$$2\left[2 \cdot (2P_P) \cdot (2L) + 3 \cdot (L \cdot L)\right]. \tag{6.6}$$

To get a first estimate of the required memory, one has to calculate the expressionss (6.5) and (6.6) and take the highest value. Since in `ECLIPSE_EB` the floating point variables are double precision, in order to estimate the number of gigabytes one has to multiply the said highest value by 8 and divide the result by 1024^3.

In a case where the number of observed pixels in intensity and polarization is 29009 and $\ell_{\max} = 192$, eq. (6.4) takes the value 50 Gb; eq. (6.5), 122 Gb and eq. (6.6), 191 Gb.

6.1.3 ECLIPSE_T

According to the critical quantities, the number of elements of the blocks or matrices that `ECLIPSE_T` stores are

1. Covariance matrix in pixel space:	P_I^2
2. Spherical harmonics matrix:	$P_I \cdot L$
3. Matrix $\mathbf{Y}^\dagger\mathbf{C}^{-1}\mathbf{Y}$:	L^2

The elements of the matrices in case 1 are real numbers, whereas the elements in cases 2 and 3 are complex numbers.

There are two stages in the code where the number of stored elements reaches the highest values

- In **Step 2**: when the program is about to compute the inverse of the covariance matrix and the key `Inverse_Covariance_Matrix_Control` takes the value 1. The code requires to store in memory two covariance matrices. The number of reals is

$$2 \cdot P_I^2. \tag{6.7}$$

- In **Step 4**: when the program is about to compute the product $\mathbf{C}^{-1}\mathbf{Y}$.

At this point, the code stores the inverse of the covariance matrix, the spherical harmonics matrix and the real part of $\mathbf{C}^{-1}\mathbf{Y}$. Therefore, multiplying the number of complexes by two, the amount of real numbers the code stores at this point is

$$P_I^2 + 2 \cdot P_I \cdot L + P_I \cdot L. \tag{6.8}$$

- In **Step 6**: when the program is about to compute $\mathbf{Y}^\dagger \mathbf{C}^{-1}\mathbf{Y}$.
  At this point, the code stores in memory the matrix $\mathbf{C}^{-1}\mathbf{Y}$, the spherical harmonics matrix $\mathbf{Y}$ and $\mathbf{Y}^\dagger \mathbf{C}^{-1}\mathbf{Y}$. Since all values are complexes, the amount of real numbers the code stores at this point is

$$2\left[2 \cdot P_I \cdot L + L \cdot L\right]. \tag{6.9}$$

To get a first estimate of the required memory, one has to calculate the expressions (6.8) and (6.9) and take the highest value. Since in `ECLIPSE_T` the floating point variables are double precision, in order to estimate the number of gigabytes one has to multiply the said highest value by 8 and divide the result by 1024^3.

As an example, in a case where the number of observed pixels in intensity and polarization is 29009 and $\ell_{\max} = 192$, eq. (6.7) takes the value 13 Gb; eq. (6.8), 30 Gb and eq. (6.9), 53 Gb.

6.2 Computer requirements

`ECLIPSE_Memory.f90` encodes the equations (6.2)-(6.9). Once the user knows the maximum memory required to store the matrices, the number of cores needed to compute the problem can be calculated. As mentioned, it is important to note that the memory required to run the code is larger than the limit given by the storage of matrices, so the memory requested for the job must be larger than the value of reference. No further instructions can be given which determine the total memory, since this will depend on the computer and configuration we use. As pointed out, the number of gigabytes needed to store the matrices must be taken as an inferior limit.

To determine the number of nodes and cores, the user needs to know the number of cores and the memory in the nodes. Once the memory required to store the matrices is stated, one can estimate the number of cores. For example, a computation requiring 191 Gb to store the matrices, executed in Haswell nodes of NERSC —with 128 Gb of memory and 32 cores—, can be done requesting 2 nodes and 64 cores. This guarantees 256 Gb to run `ECLIPSE`. The cores are best distributed as a 8×8 grid.

The script could be

```
#!/bin/bash
#SBATCH --job-name=ECLIPSE
#SBATCH --output=Job.out
#SBATCH --error=Job.err
#SBATCH --qos=regular
#SBATCH --time=03:00:00
```

```
#SBATCH --nodes=2
#SBATCH --tasks-per-node=32
#SBATCH --constraint=haswell
#SBATCH --mail-user=<user>@<domain>.com
#SBATCH --mail-type=ALL

srun ./ECLIPSE_TEB 8 NSide64/Config.ini
```

Chapter 7

Controls in the code

The code checks the regularity of the matrices to be inverted and whether the fiducial and noise models match the maps.

7.1 The inversion of the covariance matrix

Once the covariance matrix is computed, `ECLIPSE` proceeds to invert it. The operation is performed by two ScaLAPACK subroutines. Both subroutines return a parameter that indicates whether the operations were successfully accomplished. When the matrices are regular, the parameter value is zero. At this point the code displays information such as

```
Cholesky factorization:      0
Inversion result:            0
```

If one of the parameters is zero, the matrix is singular. In this case, the program displays a warning message and stops.

The program executes an additional test of the regularity of the covariance matrix: it can check whether the diagonal of $\mathbf{C}\cdot\mathbf{C}^{-1}$ is made of ones. In particular, the program calculates the sum and the product of the elements of the diagonal and displays the result. This computation provides an additional test of the regularity which is stricter than the one provided by the ScaLAPACK functions. The test is run when `Inverse_Covariance_Matrix_Control = 1` in the configuration file.

7.2 The inversion of the Fisher matrix

The program also computes the inverse of the Fisher matrix —when binning, the inverse of the compacted Fisher matrix—. In both cases the inverse is computed in two steps and the program displays the values of the parameters given by the ScaLAPACK functions. When the matrix is singular, it displays a warning message and stops.

7.3 Test χ^2 from maps and $\mathbf{C}^{-1}$

As pointed out in section 3.2.7, the code helps the user to check whether the fiducial and the noise models match the maps.

Chapter 8

Examples of execution of the code

The folder `Example_NSide8` in https://github.com/CosmoTool/ECLIPSE contains fiducial, masks, maps, bin limits and configuration files, as well as instructions to practice with `ECLIPSE`. The folder also contains an example of the output generated by the code.

On the other hand, the following lines show the output generated by `ECLIPSE_TEB` running on NERSC on a computation of high dimension.

```
*****************************************************************************************
 Start time: Wed Jul 28 13:12:43 2021
   Number of cores:             64
   Number of rows:               8
   Number of columns:            8
*****************************************************************************************
 Loading configuration from: NSide64/TEB.ini
                      Data_Folder:   NSide64
                            NSide:   64
                Fiducial_FileName:   ModeloDl_PlanckTodos.dat
           Lmax_Covariance_Matrix:   128
              Lmax_Power_Spectrum:   128
                 Type_of_Analysis:   0
            Compute_Fisher_Matrix:   1
                 Compute_Spectrum:   1
                   Maps_FileName:   MapasTQU.fits
                  Number_of_Maps:   1000
                    Pixel_Window:   1
                       Beam_FWHM:   131.922678213788
          Intensity_Mask_FileName:   MascaraSateliteN64.fits
            Intensity_Mask_NumMap:   1
       Polarization_Mask_FileName:   MascaraSateliteN64.fits
         Polarization_Mask_NumMap:   1
                   Type_of_Noise:   0
                 Intensity_Noise:  4.548120000000000E-003
              Polarization_Noise:   4.548120000000000E-004
               Remove_Noise_Bias:   1
        Matrices_Cyclic_Block_Size:   677
 Inverse_Covariance_Matrix_Control:   1
             Show_Memory_Allocated:   1
               Healpix_Data_Folfer:   /global/homes/j/jdbilbao/Descargas/Healpix_3.70/data/
*****************************************************************************************
  Loading masks
  Number of observed pixels in temperature:                                   29009
  Number of observed pixels in polarization:                                  29009
 Loading HEALPix Pixel Window
 Loading Fiducial Power Spectrum
*****************************************************************************************
 Step 1. Computing covariance matrix
   Computing blocks of the spherical harmonics matrix                            0s
                                                                            35.96 GB
    Block YTT done                                                                2s
    Block YPP done                                                                7s
   Computing blocks of the covariance matrix                                      8s
                                                                            92.39 GB
    Block TT done                   57 s
    Block TQ done                  107 s
    Block QT done                  109 s
    Block TU done                  159 s
    Block UT done                  160 s
    Block QQ done                  259 s
    Block QU done                  358 s
    Block UQ done                  360 s
    Block UU done                  459 s
  Intensity diagonal element:      3073.14818800938
  Polarization diagonal element:   5.834018299670649E-002
 Covariance matrix already computed
                                                                            56.43 GB
*****************************************************************************************
 Step 2. Inverting the covariance matrix                                        459s
                                                                           112.86 GB
   Cholesky factorization:      0                                               728s
   Inversion result:            0                                              1164s
                                                                            56.43 GB
   Diagonal product:   0.999692798028858
   Diagonal sum:        87026.9999865500
   Matrix size:            87027
 Covariance matrix inverted                                                    1167s
                                                                            56.43 GB
*****************************************************************************************
 Step 3. Computing coupled power in the harmonics space
   Computing blocks of the spherical harmonics matrix
                                                                            92.39 GB
    Block YTT done                                                             1169s
    Block YPP done                                                             1175s
   Loading maps
                                                                            93.04 GB
   Computing C^-1 Maps                                                         1211s
    <m^t C^-1 m>:    87042.2726098239
    Matrix size:         87027
                                                                            93.79 GB
   Maps transformed to harmonis space
 Auto-corrrelation power already computed                                      1273s
                                                                            92.39 GB
*****************************************************************************************
 Step 4. Computing product C^-1 Y
  Moving from matrix C^-1 to blocks of the matrix C^-1                         1273s
                                                                           130.01 GB
   TT
   TQ
   TU
   QQ
  QU
  UU
 Matrix C^-1 moved to blocks                                                     1277s
                                                                              73.58 GB
 Computing blocks of the product C^-1 Y
  Real part                                                                      1277s
                                                                             105.94 GB
   TT                   1305 s
   TE                   1356 s
   TB                   1407 s
   QT                   1433 s
   QE                   1486 s
   QB                   1539 s
   UT                   1566 s
   UE                   1619 s
   UB                   1673 s
  Imaginary part                                                                 1673s
                                                                             120.32 GB
   TT                   1701 s
   TE                   1752 s
   TB                   1803 s
   QT                   1829 s
   QE                   1882 s
   QB                   1936 s
   UT                   1962 s
   UE                   2016 s
   UB                   2070 s
Blocks of product C^-1 Y already computed                                        2070s
                                                                              64.72 GB
**************************************************************************************
 Step 5. Computing noise bias                                                    2070s
 Noise bias already computed
**************************************************************************************
 Step 6. Computing the Fisher matrix
  Moving blocks of C^-1 Y to complex form                                        2073s
  Computing blocks of the harmonic matrix in complex form
  and multiplications Y^H (C^-1 Y)
   Computing block TT of Y in complex form                                       2074s
   Computing blocks TT, TE and TB of Y^H C^-1 Y                                  2076s
                                                                              69.91 GB
   TT                   2125 s
   TE                   2175 s
   TB                   2225 s
   Computing blocks QE, QB, UE y UB of Y in complex form                         2225s
   Computing blocks EE, BB and EB of Y^H C^-1 Y                                  2231s
                                                                              82.28 GB
   EE                   2332 s
   BB                   2433 s
   EB                   2535 s
                                                                              24.75 GB
  Moving blocks to real and imaginary part                                       2535s
  Building transposed blocks                                                     2535s
  Computing blocks of the Fisher matrix
   TTTT
   EEEE
   BBBB
   TTEE
   TTBB
   EEBB
```

```
    TTTE
    TTTB
    TTEB
    EETE
    EETB
    EEEB
    BBTE
    BBTB
    BBEB
    TETE
    TETB
    TEEB
    TBTB
    TBEB
    EBEB
    Blocks of the Fisher matrix already computed                   2538 s
    Saving the Fisher matrix
  Fisher matrix already computed                                                  2568s
                                                                                0.28 KB
*****************************************************************************************
 Computing power spectrum
  Loading Fisher matrix
  Inverting Fisher matrix
  Cholesky factorization:           0
  Inversion result:                 0
  Loading Bl
  Loading YlTotal
  Computing Dl
  Saving Dl
 Elapsed time:                  2665 s
 End: Wed Jul 28 13:57:08 2021
*****************************************************************************************
```

Bibliography

[1] M. Tegmark, *How to measure CMB power spectra without losing information*, *PhRvD* **55** (1997) 5895 [astro-ph/9611174].

[2] M. Tegmark and A. de Oliveira-Costa, *How to measure cmb polarization power spectra without losing information*, *Physical Review D* **64** (2001) .

[3] J. Bilbao-Ahedo, R. Barreiro, P. Vielva, E. Martínez-González and D. Herranz, *ECLIPSE: a fast quadratic maximum likelihood estimator for CMB intensity and polarization power spectra*, *Journal of Cosmology and Astroparticle Physics* **2021** (2021) 034.

[4] J. D. Bilbao-Ahedo, R. B. Barreiro, D. Herranz, P. Vielva and E. Martínez-González, *On the regularity of the covariance matrix of a discretized scalar field on the sphere*, *JCAP* **2** (2017) 022 [1701.06617].

[5] D. J. Eisenstein, W. Hu and M. Tegmark, *Cosmic Complementarity: Joint Parameter Estimation from Cosmic Microwave Background Experiments and Redshift Surveys*, *ApJ* **518** (1999) 2 [astro-ph/9807130].

[6] K. M. Górski, E. Hivon, A. J. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke et al., *HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere*, *ApJ* **622** (2005) 759 [astro-ph/0409513].