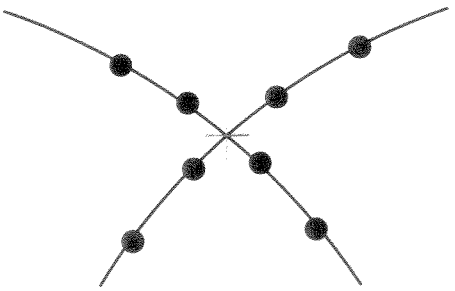
Internal Report ITeSRE/CNR 310/2001

March 2001

PLANCK-LFI **Destriping Code**
Users's Guide
Issue 2.0

G. STANGHELLINI[1], C. BURIGANA[1], D. MAINO[2],
M. MALASPINA[1], M. MALTONI[3], M. MARIS[2]



Document: PL-LFI-TES-MA-002
Revision: Issue 2.0; March 27, 2001
Prepared by: Giuseppe Stanghellini, Carlo Burigana, Davide Maino
Marco Malaspina, Michele Maltoni, Michele Maris
Authorized by: Fabio Pasian

[1] *Istituto TeSRE/CNR, via P. Gobetti 101, I-40129 Bologna, Italy*
[2] *Osservatorio Astronomico di Trieste, OAT, via G.B. Tiepolo 11, I-34131 Trieste, Italy*
[3] *Instituto de Física Corpuscular - CSIC/UVEG, Edificio Institutos de Paterna, Apt. 22085, E-46071 Valencia, Spain*

Document: PL-LFI-TES-MA-002

Revision: Issue 2.0; March 27, 2001

Prepared by: Giuseppe Stanghellini, Carlo Burigana, Davide Maino
Marco Malaspina, Michele Maltoni, Michele Maris

Authorized by: Fabio Pasian

Current code version: 2.0 - March 2001

Current code authors: Giuseppe Stanghellini, Carlo Burigana, Davide Maino,
Marco Malaspina, Michele Maltoni.

Internal Report ITeSRE/CNR 310/2001

March 2001

PLANCK-LFI Destriping Code
Users's Guide
Issue 2.0

G. STANGHELLINI[1], C. BURIGANA[1], D. MAINO[2],
M. MALASPINA[1], M. MALTONI[3], M. MARIS[2]

[1] *Istituto TeSRE/CNR, via P. Gobetti 101, I-40129 Bologna, Italy*
[2] *Osservatorio Astronomico di Trieste, OAT, via G.B. Tiepolo 11, I-34131 Trieste, Italy*
[3] *Instituto de Física Corpuscular - CSIC/UVEG, Edificio Institutos de Paterna, Apt. 22085, E-46071 Valencia, Spain*

SUMMARY – In this document we describe the improvements recently implemented in the PLANCK-LFI **destriping** code to make it more efficient from the computational point of view, to partially overcome the RAM requirements and to make it more versatile and general. We outline the guidelines for the installation and running of the code.
The PLANCK-LFI DPC of Trieste has the responsability of the software which is stored in the PLANCK-LFI Software Repository system under CVS.
Account to the LFI Software Repository can be obtained from the LFI DPC for the LFI consortium members.

1 Introduction

One of the most important issues in the context of PLANCK-LFI experiment (Mandolesi et al. 1998) is related to the $1/f$ noise fluctuations in LFI receivers that, when coupled with PLANCK scanning strategy, produces artifacts in the final maps such as stripes. These stripes can increase the overall noise level and introduce correlations which may affect the statistical analysis of the CMB pattern in the sky. The relevant radiometer characteristic can be combined into a single parameter, the knee-frequency f_k, which has to be kept as low as possible compared to the spinning frequency f_s of the spacecraft. It has been shown (Janssen et al. 1996) that for $f_k \gtrsim f_s$ a degradation in the final sensitivity will result.

2 The Flight Simulator

In order to assess the impact of the $1/f$ noise on PLANCK-LFI observations, within the LFI Consortium, we developed a Fortran code, the so-called "Flight Simulator" (hereafter FS), that is able to reproduce the sky observations performed by a given horn in the PLANCK focal plane. This code is the core of all the simulations and includes all the relevant geometrical and instrumental properties: beam location on the sky field of view, beam response function, instrumental noise, telescope configuration. A more detailed description of this code is reported in Burigana et al. (1997) and Maino et al. (1999).

The relevant point here is the data in output from the FS. These are several (typically four) matrices with a number of rows equal to the number of spin axis positions ($n_s \sim 10200$ for $2.5'$ shift of the spin-axis and a mission observation duration of 14 months) and a number of columns n_p equal to the total number of samplings on a given scan circle (*e.g.* for a simulation at 30 GHz with 3 samplings per FWHM ($\simeq 33'$), $n_p \sim 1980$, weakly dependent on the angle α between pointing and spin axes). These matrices contain the pixel identification number $\mathbf{N}$ in a given pixelisation scheme at the desired final map resolution, and, typically, the observed sky temperature with full noise (white + $1/f$ noise) $\mathbf{T}$, the observed signal with only white noise $\mathbf{W}$ and the pure signal without instrumental noise $\mathbf{G}$. These last two matrices are considered in order to evaluate the degradation of $1/f$ noise with respect the pure white noise case and to study the impact of the scanning strategy geometry on the observed pixel signal.

3 Destriping technique: concept

In this section we discuss how to eliminate the effects of gain drifts on timescales greater than that for which the spin axis is fixed at a given direction (1 hour in the current baseline). Usually we work with averaged scan circles *i.e.* we took the average over the 60 scans for a given spin axis direction. This however is a working hypothesis for this particular case and the code can work with all the single scan circles. Averaging is like a low-pass filtering and, as long as f_k is not far greater than f_s this ensures that only the very low frequencies of the $1/f$ noise survive this filter operation. It has been shown by Janssen et al. (1996) that the residual $1/f$ noise in this case can be well approximated as an additive level A_i related to the "mean" level of $1/f$ noise during the period of observation in that scan circle. Of course these levels A_i are different for different circles due to gain fluctuations. The goal is to obtain the levels for each circle and subtract them from the corresponding scan circle. It is also possible to search for more than one baseline, say n_l, per circle. This is exactly equivalent to rearrange the matrices $\mathbf{N}$, $\mathbf{T}$, $\mathbf{W}$ and $\mathbf{G}$, dividing their rows into n_l parts that have to be properly relocated to construct matrices with $n_s \times n_l$ rows and n_p/n_l columns. The analysis then follows in the same way of a single baseline case.

For estimating all these levels we use a computation scheme able to simultaneously find the pixels in common between different scan circles. In the following N_{il}, T_{il} and E_{il} will denote the pixel number, the temperature and white noise level for the pixel in the $i^{\rm th}$ row and $l^{\rm th}$ column. The pixel identification number can be also stored into a matrix $\mathbf{N}'$ at resolution levels different (typically higher) from that desired for the final map. This allows to produce maps at a given resolution and searching the common pixels at another, higher, level.

A generic pair of different observations of the same pixel is identified by π that will range between 1 and n_c, the total number of pairs found. In this notation π is related to two elements of $\mathbf{N}$:$\pi \rightarrow (il, jm)$ where i and j identify two different scan circles while l and m are the relative position in each of the two scan circles.

We want to minimize the quantity:

$$\begin{aligned} S &= \sum_{\rm allpairs} \left[\frac{[(A_i - A_j) - (T_{il} - T_{jm})]^2}{E_{il}^2 + E_{jm}^2} \right] \\ &= \sum_{\pi=1}^{n_c} \left[\frac{[(A_i - A_j) - (T_{il} - T_{jm})]^2}{E_{il}^2 + E_{jm}^2} \right]_\pi \end{aligned} \tag{1}$$

with respect to the unknown levels A_i. The sub-index π indicates that each set of (il, jm) pairs is used in the summation. From Eq.(1) it is clear that S is quadratic in the unknowns A_i and that only differences between A_i enter into the equation. Therefore the solution of the system will be determined up to an arbitrary additive constant with no meaning in anisotropy measurements. We choose to remove this level of uncertainty requiring that $\sum_{h=1}^{n_s} A_h = 0$. This is equivalent to replace Eq.(1) with $S' = S + (\sum_{h=1}^{n_s} A_h)^2$. After some algebra we get:

$$\frac{1}{2}\frac{\partial S'}{\partial A_k} = \sum_{\pi=1}^{n_c} \left[\frac{[(A_i - A_j) - (T_{il} - T_{jm})] \cdot [\delta_{ik} - \delta_{jk}]}{E_{il}^2 + E_{jm}^2} \right]_\pi + \sum_{h=1}^{n_c} A_h = 0 \tag{2}$$

for all the $k = 1, ..., n_s$ (here the δ are the usual Kronecker symbols). This translate into a set of n_s linear equations:

$$\sum_{h=1}^{n_s} C_{hk} A_h = B_k, \qquad k = 1, ..., n_s \tag{3}$$

which can be easily solved. We denote with $\mathbf{C}$ and $\mathbf{B}$ the matrix of the coefficients C_{kt} and the vector of coefficients B_k.

We show here how $\mathbf{C}$ and $\mathbf{B}$ are formed as we extract pixels in common. This is the way adopted in the code. First of all we set $\mathbf{B}$=0 and $C_{kt} = 1$ for each k, t (setting all $C_{kt} = 1$ takes into account the second term of Eq.(2). Then for each couple π of pixels in common between two scan circles we define:

$$\chi_\pi = \left[\frac{1}{E_{il}^2 + E_{jm}^2} \right]_\pi \tag{4}$$

and

$$\tau_\pi = \left[\frac{T_{il} - T_{jm}}{E_{il}^2 + E_{jm}^2} \right]_\pi . \tag{5}$$

From the above equations we have that a given pair π contribute only to two equations of our linear system: those for $k = i$ or $k = j$. If we iteratively increment $\mathbf{C}$ and $\mathbf{B}$ as we find a couple, explicitely we have:

$$C_{ii} \rightarrow C_{ii} + \chi_\pi \tag{6}$$

$$C_{ij} \rightarrow C_{ij} - \chi_\pi \tag{7}$$

$$C_{ji} \rightarrow C_{ji} - \chi_\pi \tag{8}$$

$$C_{jj} \rightarrow C_{jj} + \chi_\pi \tag{9}$$

$$B_i \rightarrow B_i + \tau_\pi \tag{10}$$

$$B_j \rightarrow B_j - \tau_\pi \tag{11}$$

Therefore each pair contributes to only six terms and the resulting system shows a complete symmetry with respect to the exchange of the rows indexes i and j. The linear system have in fact some useful properties:

- is *symmetric*: we can hold in memory only half of the matrix (*e.g.* upper-right) and solve the system speeding up the calculations computing only half of the matrix coefficients. This is possible when using the Gauss reduction algorithm since it preserves the symmetry of the remaining part of the matrix;
- is *positive defined* so there is no null pivot when reducing a non-singular matrix (Strang 1976). It is possible to solve the system without exchange rows and columns, so preserving symmetry;
- is *not singular* provided that enough pairs are present since the only indetermination is removed.

When the system is solved, we end up with the baseline levels A_i which have to be subtracted from the matrix $\mathbf{T}$.

4 Destriping technique: the code

Here we report the structure of the code and the main flow of data. See also the users's guide of the PLANCK-LFI destriping code, issue 1.0 (Maino et al. 2000).

1. `int main (int argc, char **argv)`

   this is the `main` section of the code. First of all the code read from the file [`ReadParams()`] provided in the calling sequence the required inputs. Then create and sort the vector of pixel numbers [`CreateSortedVector( , , )`]. It also reads the other TOD with full signal [`ReadFortranMatrix( , , , , )`] . Creates the symmetric matrix $\mathbf{C}$ and the matrix $\mathbf{B}$ of coefficients [`CreateMatrix( , , ,)`] , call for solving the system [`Solve( , , ,)`], produce the destriped matrix [`DestripeMatrix( , )`] and save the results into FITS or binary files [`WriteFits( , , , )`].

2. `CreateSortedVector (struct VECTOR vec, char *pix_file1, char * pix_file2)`

   this routine read from `FORTRAN` files `pix_file1` and `pix_file2` the matrices with pixel numbers at different map resolution (typically $\mathtt{n}_{side}$ = 256 and 512 for a 30 GHz simulation). The structure `vec` has four attributes that are the pixel numbers at different resolution for the corresponding position in the `vec` (`vec[idx].pixel` and `vec[idx].hires`) and the original row and column in the matrix $\mathbf{N}$ (`vec[idx].row` and `vec[idx].col`). The structure is ordered according to the `vec[idx].pixel` attribute.

3. `CreateMatrix(char *filename, double *mtx, struct VECTOR vec, unsigned int *cnt)`

   this routine create the matrix of coefficient C_{hk} and known terms elements B_k. It works as follows. A memory buffer is created large enough to keep L lines; the search for pairs starts and the quantities χ_π and τ_π are evaluated; if $i \in [0, ..., L-1]$ then Eqs. (6,7,10) are evaluated; if $j \in [0, ..., L-1]$ the Eqs.(8,9,11) are applied. After all couples are evaluated the memory buffer is saved into a file. These steps are repeated for i and j in the range $[L, ..., 2L-1]$ and so on until the matrix coefficients is completed. This is then saved into a binary file that have n_s rows and $n_s + 1$ columns where the last column contains the known terms.

4. `Solve(char *temporary, char *filename, double * solution, bool_do_verify)`

   this is the routine that solve the system using the Gauss elimination method. As for the creation of the symmetry matrix a memory buffer which contains L lines is created. These first lines are loaded and complete Gauss elimination is performed: each line is reduced by the preceding lines and used to reduce the following. Each of the remaining $n_s - L$ lines is sequentially loaded into memory an reduced by each of the L lines stored into the buffer. The buffer is flushed and the next L lines are considered and reduced. These steps are repeated until to system is completely reduced.

5. `DestripeMatrix(double *temp, double *solut)`

   this routine subtract from the matrix $\mathbf{T}$ the baseline levels stored in `solut`. The resulting matrix is then converted into a map with `Matrix2Map( , , )` that simply coadd pixels to form a sky map.

6. `WriteFits(double *map_buff, map_size, sizeof(double), n_fc_twhite )`

   this routine write the output map in a FITS file using the CFITSIO 2.0 library. Since the input map of the FS code are in HEALPix pixelisation scheme (Górski et al. 1998), the same will be for the output map. Therefore maps can be stored as C/IDL binary file or as binary tables, in which case the header of the FITS file is according to HEALPix header style. The input FS code data streams are presently assumed to be binary files, written row by row (Integer*4 for pixel numbers and Real*8 for temperature values). Future code releases will consider different input data format (typically in FITS), according to the conventions of PLANCK observation simulation codes available in the future. We note that this code can work with arbitrary pixelisation schemes, the first version being in fact written assuming QuadCube pixelisation. An equal area pixelisation scheme is particularly advantageous for a uniform applicability on the sky of the crossing condition rule, since it is actually implemented by using pixel number only to have computational efficiency. Also it is very useful to work with pixelisations offering a wide choice of resolutions; a hierarchic structure, although simplifies the construction of input pixel matrices at different resolutions, is not a requirement for this code.

5 Changes of the input parameter file

Here we report the changes of the 2.0 version over the 1.0 and how them reflects to the input parameter file.

1. `Characterization of input/output files`

   An improvement over the old version of the code is that now the program is able to read/write the input or output files in single or double precision, the user can choose amongst them by specifying, into the input parameter file, the keywords OUTPUT_DATA_PRECISION and INPUT_DATA_PRECISION as single or double, and, is of course possible, to have single precision input files and double precision output file, as well as the opposite. We must state at this point that the internal computations that the program does, during the coadding and the destriping phase are always in double precision, regardless of the choice of that two parameters; in fact the program translates the input files in double precision format, during the reading operation, and does, double to single conversion, during writing of the output files, letting the arithmetical precision of the results, virtually inalterate. The keywords INPUT_DATA_FORMAT and OUTPUT_MAP_FORMAT specifies the format respectively for input data files (globally for all input files) and for the ouput map files (globally for all output map files). They can get the values "fortran_binary" or "raw_binary" for the input data format, and "raw_binary" or "fits" for output map format.

2. `Input temperature files`

   With the 2.0 version of the software it is possible to tell the program that some input files were not acquired, by putting the string "n/a" as filenames in the input parameter file. In the same way if we don't want to generate some output file we can use the same string "n/a" in the same way as for the input files, to tell the program that the output is not applicable. In this way is no longer required to generate all ouput maps, saving cpu time and disk space.

   In this version 2.0 of the LFI destriping code it is possible to give as input of the program the TEMP_NO_NOISE_FILE and to specify if this stream must be added to the streams TEMPERATURE_FILE and TEMP_WHITE_NOISE_FILE. In this way, we can have for example as input a stream with pure sky temperatures (or other kind of signals, i.e. signals from periodic fluctuations – see the last section) and two streams with pure noises only.

   These options are controlled by the the keyword:

   `ADD_NONOISE_TO_NOISE  yes/no`

   for the stream associated with TEMPERATURE_FILE, and by the keyword

   `ADD_NONOISE_TO_WHITENOISE  yes/no`

   for the stream associated with TEMP_WHITE_NOISE_FILE.

   The user can use these options according to the specific problem to solve.

   As illustrative example we can, with these possibilities for the input parameters, to manage the case in which TEMP_NO_NOISE_FILE contains only pure sky signal and TEMPERATURE_FILE (TEMP_WHITE_NOISE_FILE) contains pure noise (pure white noise) or the sum of the noise (pure white noise) and pure sky signal.

3. `Pixel files and related keywords`

Four new pixel related keywords have been introduced to allow to the user more flexibility on pixel input files. They are:

```
PIXEL_USAGE_METHOD     resample/as_is
PIXEL_NSIDE            integer
PIXEL_CROSSING_NSIDE   integer
PIXEL_MAP_NSIDE        integer
```

We remember that in previous version two pixel input files were requested from the program: the pixel file and the pixel hires file. With these new possbilities the user can choose two operating mode: with PIXEL_USAGE_METHOD equal "as_is" we have the old behaviour, with PIXEL_USAGE_METHOD equal "resample" we have the following behaviour: given the PIXEL_NSIDE equal tho the nside of the input pixel file, the pixel file for crossing computation/crossing searching will be the pixel input file resampled with PIXEL_CROSSING_NSIDE value, and the pixel file for map computation will be the pixel input file resampled with PIXEL_MAP_NSIDE value.

As illustrative example we can do that with the following choices for pixel related parameters:

```
PIXEL_USAGE_METHOD     resample
PIXEL_NSIDE            4096
PIXEL_CROSSING_NSIDE   512
PIXEL_MAP_NSIDE        256
PIXEL_FILE             pix_4096.bin
PIXEL_HIRES_FILE       n/a
```

In this case the original pixel file "pix_4096.bin" had a NSIDE equals to 4096 while the pixels used for crossing searching will have NSIDE equals 512 and pixels use for map computations will have NSIDE equals 256.

4. `Additional input streams`

   Two new input streams were added to the program:

   ```
   FLAG_MATRIX_FILE          flag.asc
   SENSITIVITY_MATRIX_FILE   err.bin
   ```

   The flag matrix contains, line by line, pairs, each of them identifies row and column of pixels that have to be considered erroneous, thus not applicable in further computations.

   This feature also allows to work with input matrices with a different number of columns in the different rows (i.e. as in the case of not regular sky sampling).

   The sensitivity file is a matrix that contains for each pixel of input streams, a sensitivity value to get rid of instrumental sensitivity oscillations that could arise during very long observations.

5. `Old and new output maps`

   The old output maps, the related keywords and their contents are (see also the point 2. of this section):

TEMPERATURE_MAP	map_t.bin	noise (+ sky signal) map
TEMP_NO_NOISE_MAP	map_tnonoise.bin	sky signal map
TEMP_WHITE_NOISE_MAP	map_twhite.bin	white noise (+ sky signal) map
DESTRIPPED_TEMP_MAP	map_tdestr.bin	destriped noise (+ sky signal) map
SENSITIVITY_MAP	map_nobs.bin	map of sensitivity per pixel
COUPLE_RECURRENCE_MAP	map_xing.bin	# of times in which pixel=crossing

One new output map was added to the program:

```
POINTINGS_MAP   npoint_map.bin   # of pointings per pixel
```

The pointing map is a map that contains for each pixel the number of pointings of that pixel.

[The third column are comments inserted here for more clarity of the contents of the `destriping.par` file. The relevant columns are only the first two.]

6. `Solving the system`

   It is now possible to tell the program to not solve the system, by specifying the keyword SOLVE_SYSTEM yes/no in the par file. With this new feature is it possible to load a previously computed solution by file and then proceed with the destriping phase, saving all the time to solve the system, of course, is necessary that the system was solved at least one time for each set of input streams. This is obtained by specifying in the input parameter file the following keywords:

```
SOLVE_SYSTEM                no
LOAD_SOLUTIONS_FROM_FILE    solut.bin          recovered baselines
SOLUTIONS_FILE              n/a
DESTRIPPED_TEMP_MAP         t_destr_map.bin
```

   With these parameters the program will try to load the solution from the solut.bin file that in this case is not any more an output file but becomes an input file, if this is succesfull then the program continues with the last step of the destriping phase; the application of the solution to the input stream.

   The buffer related keywords are used to set the maximum amount of physical memory that the program will try to allocate in the process of system solving.

   The N_CREATE_BUFFER_MEMBERS refers to buffer size used in the phase of system creation, while the others two, to buffer sizes used in the phase of system solving. All numbers refers to number of elements that the system will try to allocate, thus to have the absolute number of bytes of memory allocated these numbers should be multiplied by 8 (the size in byte of the double precision internal representation of a number).

```
N_CREATE_BUFFER_MEMBERS   10000000   # of buffer members
N_BIG_BUFFER_MEMBERS      4000000    # of big buffer members
N_SMALL_BUFFER_MEMBERS    1000000    # of small buffer members
```

   One of the bigger changes to the program is that now the solving of the system performed during the destriping phase can be obtained by dividing the linear system in subsystems, and by solving them one at a time, reducing drastically (in the order of ten to hundred times) the cpu time required for destriping; loosing, of course, something on final precision.

   This is obtained by specifying in the input parameter file the following keyword:

```
ROWS_PER_ITERATION   integer_number
```

   [The third column are comments inserted here for more clarity of the contents of the `destriping.par` file. The relevant columns are only the first two.]

6 Code requirements and compilation

This code can run virtually on every machine, for what concern the system solution, almost regardless of its RAM, thanks to an interesting use of memory buffer. Reducing the size of

the buffer, will produce only a small increase of the computation time. An optimal choice has to be taken considering the RAM of the machine and the proper buffer size in order to do not make the code swapping on disk.

In the current version it is possible to write file in FITS format. It is therefore necessary to have installed the software package that allows to manipulate FITS file: CFITSIO V 2.0 library (`http://heasarc.gsfc.nasa.gov/docs/software/fitsio/`). [It is however possible to work only with binary files properly selecting the format of output files and commenting the routine which write FITS files.]

The code now compiles through a makefile in several architecture, the compilation is done with the following command:

```
make arch
```

Where arch is the name of the architecture; by simply issuing the command "make" it is possible to gain information on supported architectures (currently linux, digital unix and irix). The makefile get rid also of generating the fitsio library, as with this version of symmetry it is included in a subdir at the same level of the symmetry package.

The code needs an input file with several parameters defined. The calling sequence will be:

```
./symmetry symmetry.par
```

where all the mentioned parameters are written into the ".par" file.

6.1 Parameter file

Here we report the structure of the parameter file `symmetry.par`.

```
#
# Parameters file for symmetry
#
# Input file names (these files are supposed to be in Fortran format)
#
 PIXEL_FILE                 pix_map_resolution.bin        matrix N
 PIXEL_HIRES_FILE           pix_search_resolution.bin     matrix N'
 TEMPERATURE_FILE           t_signal_and_noise.bin        matrix T
 TEMP_NO_NOISE_FILE         t_onlysignal.bin              matrix G
 ADD_NONOISE_TO_NOISE       yes/no
 TEMP_WHITE_NOISE_FILE      t_signal_whitenoise.bin       matrix W
 ADD_NONOISE_TO_WHITENOISE  yes/no
 FLAG_MATRIX_FILE           rows and columns of pixels.   ascii file
 SENSITIVITY_MATRIX_FILE    err.bin                       matrix E
 LOAD_SOLUTIONS_FROM_FILE   solut.bin                     recovered baselines
#
# Output file names (these files are supposed to be in C/IDL format)
# Remember to create the appropriate subdirectory for output files !!!
#
 SYMM_TEMPORARY_FILE  temp_symmetry_file.bin  temporary file
 SYMM_MATRIX_FILE     symm.bin                symmetric matrix when solving
 SOLUTIONS_FILE       solut.bin               recovered baselines
```

```
 TEMPERATURE_MAP          map_t.bin           noise (+ sky signal) map
 TEMP_NO_NOISE_MAP        map_tnonoise.bin    sky signal map
 TEMP_WHITE_NOISE_MAP     map_twhite.bin      white noise (+ sky signal) map
 DESTRIPPED_TEMP_MAP      map_tdestr.bin      destriped noise (+ sky signal) map
 SENSITIVITY_MAP          map_nobs.bin        map of sensitivity per pixel
 COUPLE_RECURRENCE_MAP    map_xing.bin        map of the # of times in which that pixel is a crossing
 POINTINGS_MAP            map_npoint.bin      # of pointings per pixel

#
# Input parameters
#
 PIXEL_USAGE_METHOD       resample/as_is              Method for pixel usage
 PIXEL_NSIDE              integer                     nside of PIXEL_FILE
 PIXEL_CROSSING_NSIDE     integer                     nside for crossing searching
 PIXEL_MAP_NSIDE          integer                     nside for map computation
 INPUT_DATA_FORMAT        raw_binary/fortran_binary   Input format of all files
 INPUT_DATA_PRECISION     single/double               precision of all input files
 OUTPUT_DATA_PRECISION    single/double               precision of all output files
 OUTPUT_MAP_FORMAT        raw_binary/fits             output files: fits or IDL (raw binary)?
 MATRIX_ROWS              integer                     n_s
 MATRIX_COLOUMNS          integer                     n_p
 ROWS_PER_ITERATION       integer                     # of recovered baselines of each subsystem
 N_PIXELS_MAP             integer                     N_pix of the map
 SPIN_ERROR               double                      (averaged) TOD sample sensitivity
 GAUSS_THRESHOLD_PERCENT  double                      % accuracy in method
 N_CREATE_BUFFER_MEMBERS  integer                     # of buffer members
 N_BIG_BUFFER_MEMBERS     integer                     # of big buffer members
 N_SMALL_BUFFER_MEMBERS   integer                     # of small buffer members
 SOLVE_SYSTEM             yes/no                      solve or not the system
 VERIFY_AFTER_SOLVE       yes/no                      verify solution
```

[The third column are comments inserted here for more clarity of the contents of the `destriping.par` file. The relevant columns are only the first two.]

7 Tests

Some successfull tests of this version of the LFI destriping code has been carried out in the context of the simulation work to remove systematic effects induced by periodic fluctuations (possibly jointed to white and $1/f$ noise) in PLANCK-LFI data (Mennella et al. 2001).

Please, send any possible bug report to C. Burigana (e-mail: burigana@tesre.bo.cnr.it) and G. Stanghellini (e-mail: gstanghe@igm.bo.cnr.it).

References

[1] Burigana C., Malaspina M., Mandolesi N., et al., 1997, Int. Rep. TeSRE/CNR 198/1997

[2] Górski K.M., Hivon E., Wandelt B.D., 1998, to appear in "Proceedings of the MPA/ESO Conference on Evolution of Large-Scale Structure: from Recombination to Garching", Banday A.J. et al. (Eds.), astro-ph/9812350

[3] Janssen M., Scott D., White M., et al., 1996, astro-ph/9602009

[4] Maino D., Burigana C., Maltoni M., et al., 1999, A&AS, 140, 383

[5] Maino D., Maris M., Burigana C., Stanghellini G., Maltoni M., 2000, PL-LFI-OAT-MA-002, Issue 1.0, April, "PLANCK-LFI Destriping code Users's Guide"

[6] Mandolesi N., et al., 1998, PLANCK Low Frequency Instrument, A Proposal Submitted to the ESA

[7] Mennella A., Bersanelli M., Burigana C., Maino D., Mandolesi N., Morgante G., Stanghellini G., 2001, A&A, in preparation.

[8] Strang G., 1976, "Linear Algebra and Its Applications", Academic Press, Inc.