Beyond PLANCK

Deliverable 1.2: Operational Gibbs Sampler

Authors Mathew Galloway
Kristian Joten Andersen
Hans Kristian Eriksen
Trygve Leithe Svalheim
Ingunn Kathrine Wehus
Maksym Brilenkov

Date February 28th, 2019

Work Package WP1 – Gibbs Sampling Integration

Revision History

Version	Authors	Date	Changes
1.0	Mathew Galloway Kristian Joten Andersen Hans Kristian Eriksen Trygve Leithe Svalheim Ingunn Kathrine Wehus Maksym Brilenkov	February 25, 2019	Initial Version

Contents

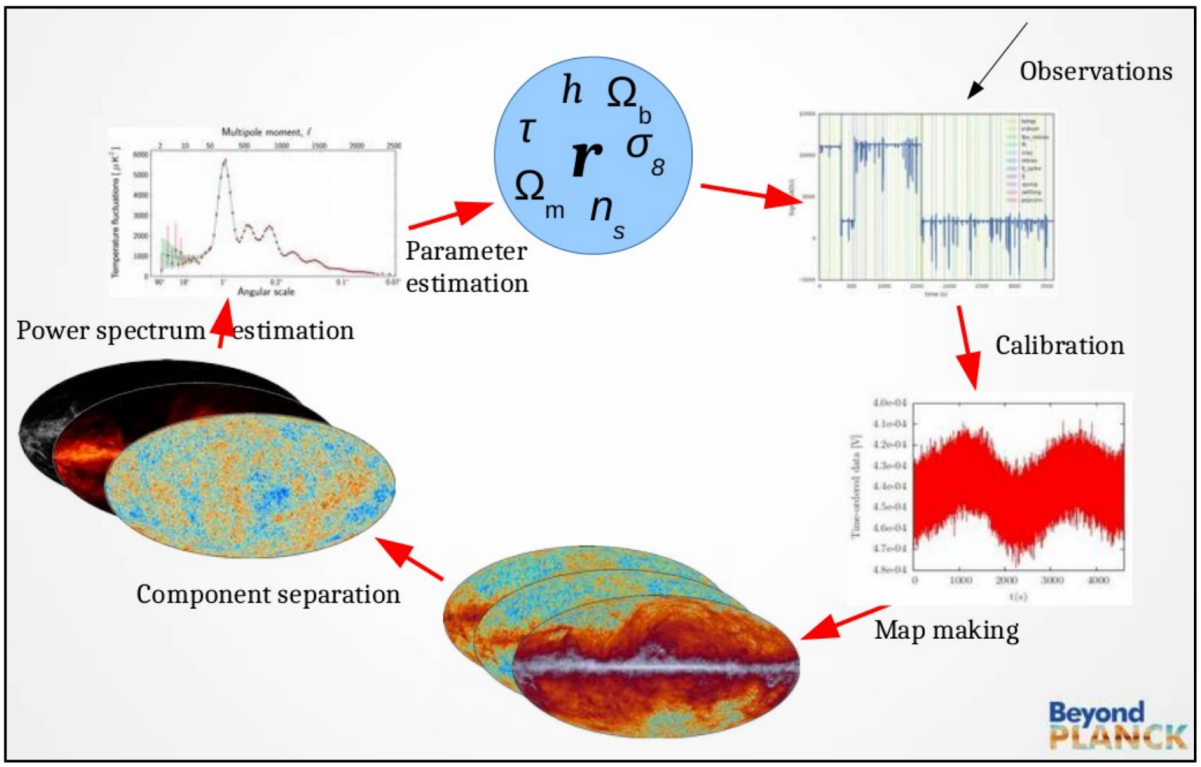

Figure 1: The high-level Gibbs sampling loop

1 Overview

The main goal of the BeyondPlanck project is to build an end-to-end Gibbs sampler for the Planck LFI data, and use this to improve the overall calibration and fidelity of the final LFI sky maps. This Gibbs sampler is illustrated in Figure 1, and may be summarized by the following steps:

1. Data selection and calibration
2. Map making
3. Component separation
4. Power spectrum estimation
5. Parameter estimation

These five steps are then iterated until convergence, where each step uses information produced by previous steps. This pipeline was first implemented in an informal manner during the Planck project, but BeyondPlanck aims to integrate the pipeline into a single code base that can run without human intervention on one computer.

To this end, this deliverable presents the BeyondPlanck Python Pipeline, a working implementation of the theoretical Gibbs sampling loop shown in Figure 1. The pipeline is written in Python, using an object-oriented style, and interfaces with the various legacy codes using the Python operating system interfaces subprocess and multiprocessing.pool. The full code base is available at:

https://gitlab.com/BeyondPlanck/repo/tree/master/pipe

2 Software

The pipeline uses the individual Gibbs sampling modules are available in the following sub-directories in the Gitlab repository:

1. Data selection and calibration: calib/
2. Map making: madam/
3. Component separation: commander/
4. Power spectrum estimation: like/
5. Parameter estimation: like/

Parallelization is accomplished through the use of MPI as well as OpenMP, and enables all codes to be run using the maximum number of cores available. Additionally, some short segments of the pipeline use Pythons multiprocessing.pool interface to run small jobs in parallel.

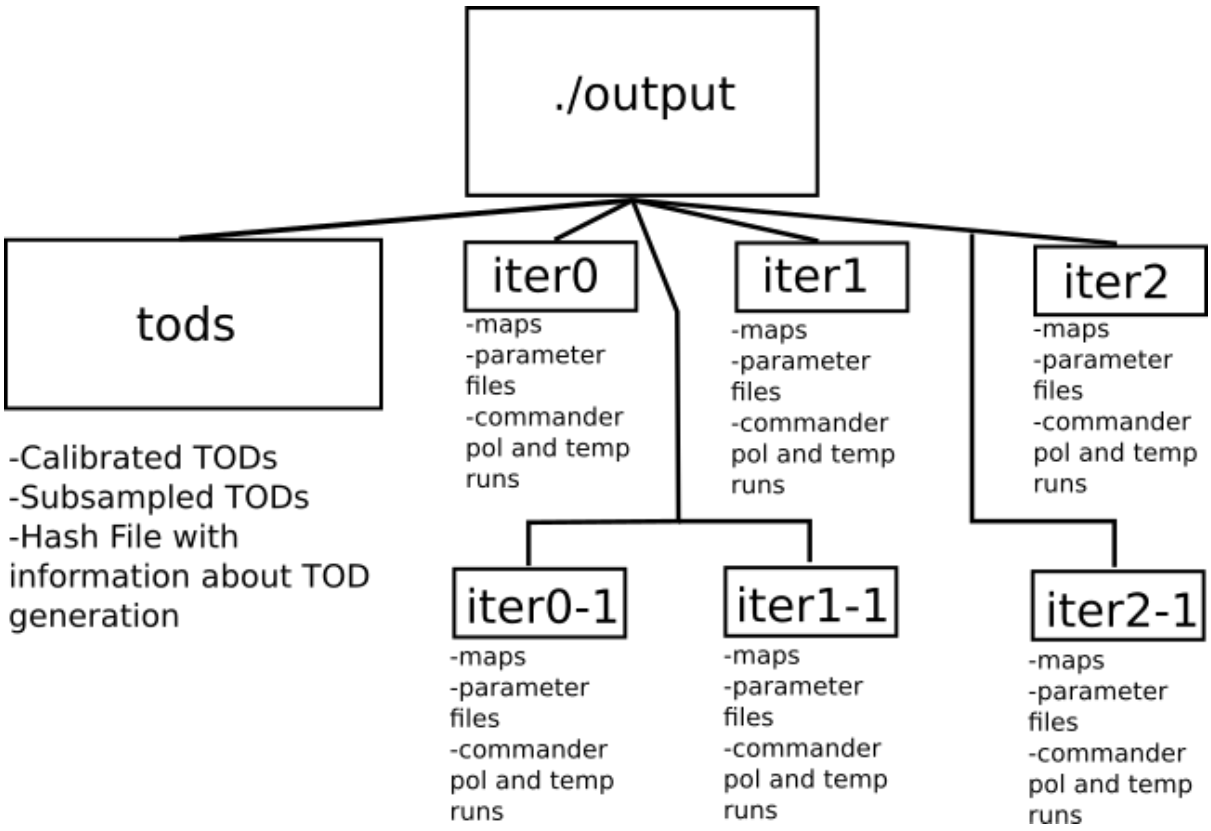

Figure 2: The data storage architecture of the python pipeline

The pipeline code can be used to run each module individually, which is helpful for testing functionality as well as for smaller use cases. By default, the entire pipeline is run, initializing from a specified directory or from the raw data. The pipeline is designed to check at each step if the outputs it would create have already been generated from a previous run, allowing it to start from where it left off.

The data storage architecture is shown in Figure 2. Each iteration generates a new output directory so that all previous runs are stored by default. The output directory names are chosen so as not to conflict with existing ones. The only exception to this is that the calibrated Time Ordered Data (TODs) are overwritten every iteration, as storing them each time would be prohibitively large. The pipeline uses a hash file stored on disk to link each iteration to the TODs that it generated. Once they are overwritten, that hash file is deleted and replaced with the one corresponding to the new iteration.

All constant pipeline input data (such as the RIMO files, raw timestreams, etc.) that do not change with iteration are stored on disk at a shared location. Each pipeline user must simply point the pipeline at this directory to access the identical files. This allows consistency in input versions between users. On the Owl cluster, that directory can be found at /mn/stornext/d14/bp/data.

2.1 The bppyp Module

The main pipeline loop is located in the file bppyp.py. This file handles all the parameter parsing that is common to the pipeline, making use of Python's argparse module. It uses sensible defaults so that the user only has to specify non-standard parameters they would like to use, reducing the burden on new users to figure out how to run it. This module also compiles all the sub-codes to ensure that the most recent code version is used. Finally, it starts the main loop, iterating through each of the steps of the pipeline until convergence.

To run the pipeline, simply navigate to the pipe subdirectory and type 'python bppyp.py'. This will execute the main loop with the default options. To adjust parameters, simply specify them at the command line. For example, to change the number of cores to use to 100, run 'python bppyp.py –num-procs 100'. For a complete list of all options and their functions, executing 'python bppyp.py -h' will display a help dialog and exit.

2.2 The calib Module

The calib.py module serves as the interface to the DaCapo calibration code provided by WPs 2 and 3. It first checks to see if the output files already exists. If not, it constructs a calibration parameter file for each of the LFI horns, and calls the calibration code once for each horn using the maximum number of MPI threads. Once it has generated all the calibrated timestreams, the code creates a hash file, linking the current TODs to the directory from which they were produced.
The calib module can also run the DataSelection code, although this is not performed by default in the Gibbs loop. This allows the pipeline to serve as a common interface to all the sub-modules so that the entire collaboration can use all the codes without knowing much about the low level interfaces.

2.3 The madam Module

The mapmaker is controlled by the madam.py module. When called, it first checks if the files it will generate are already on disk. Then it builds a parameter file and a simulation file for each of the LFI frequencies. The parameter file includes the general configuration parameters and the simulation file contains the paths to the data files and pointing information. Then, the madam module executes the madam mapmaking code provided by WP4, specifying the maximum number of MPI threads.

Once the destriped maps have been generated, the code performs a few short operations to convert them to the correct input format for commander, the next module in the Gibbs chain. Firstly, madam's output per-pixel covariance matrix, which contains I,Q and U noise terms and their cross-correlation, is converted into a set of I, Q and U RMS maps, which is the format that Commander requires the noise input. Then, the monopole values in the madam maps, which are set arbitrarily in the mapmaker, are set to the same values as in the npipe maps. This ensures that the maps are compatible with one another, and that the monopole tuning that has been done for npipe is applicable to these maps as well. This is accomplished using the map_editor software developed for QUIET, and uses a galactic mask to avoid biases.

2.4 The commander module

The commander module is the next step in the Gibbs chain. It has the capability to run both the Commander1 code and the Commander2 code, although Commander2 is what is currently used in the pipeline. Unlike the other modules, it does not

generate a commander parameter file, as these are incredibly long and writing all that code was deemed low priority. Instead, it reads in an existing parameter file from disk and changes only those parameters which are updated each loop, which are:

OUTPUT_DIRECTORY: The output directory path
BAND_MAPFILE: The new madam maps for the LFI channels
BAND_NOISE_RMS: The new madam noise files for the LFI channels
BAND_MASKFILE: The new madam masks for the LFI channels

Once the new parameter file has been written, the code calls commander2 with the maximum number of MPI threads and setting OMP_NUM_THREADS to 1. This speed trade off could be investigated to see if there are other configurations that result in faster runtimes.

Commander runs a total of four times, once to solve for the temperature components, once to solve for the polarized components, and twice more to project those components into the bandpasses of all the LFI detectors. Once this has been accomplished, the total signal model for each of the LFI detectors is summed in map space before being converted to harmonic space using the python anafast module. These harmonic representations of the signal are then passed to the convolution module.

2.5 The convolve module

The final module is the sampling loop is the convolve code, which wraps the totalconvolve_cxx code written for Planck as part of the LevelS package. It first checks if the files it would generate are already existent, as is standard. Then, it loops over each horn, then over each side and main detector, and then over sidelobes and main beams. For each of these 11*2*2 = 44 cases, it generates a totalconvolver_cxx file, and calls totalconvolver using a single thread. This is accomplished in parallel using multiprocessing.pool, so all convolution operations are run simultaneously as long as at least 44 cores are allocated. The totalconvolve code takes the signal model and the beam profiles and generates a ringset, which is then fed back to the calibration code (2.2) to close the loop.

3 Parameter Files

This section will present example parameter files that the code has generated. As the parameters of the full pipeline are still being tuned, these are not the final parameter files, but instead represent the current state of the pipeline. Additionally, the commander parameter files are incredibly long (over 1000 lines each), so they are not included here but can be accessed at:

https://gitlab.com/BeyondPlanck/repo/blob/master/pipe/

There are 4 of them and they are named:

comm2_bp_temp.txt – calculates the temperature components
comm2_bp_pol.txt – calculates the polarized components
comm2_bp_proj_temp.txt – projects the temperature components to the LFI bands
comm2_bp_proj_pol.txt – projects the polarized components to the LFI bands

```
log_level=WARNING
start_od=91
end_od=1604
aux_path=/mn/stornext/d14/bp/data/auxiliary_data
weights_table=diode_weights.fits
velocity_file=satellite_velocity.fits
mask=mask_calib_70GHz.fits
level2_data_basename=LFI_070_
subsample_basename=LFI_subsampled_070_
ringset_interpolation=9
cmb_temperature=2.72548
dipole_temperature=0.0033645
dipole_longitude=264.0
dipole_latitude=48.24
quality_flag=6111248
convolution_parameters=dipole_convolve_parameters.fits
subsample_houskeeping_name=subsampled_hk.fits
sample_list=list_sampling_70.fits
galaxy_ringsetM=full_sky_DX12_LFI18M.fits
galaxy_ringsetS=full_sky_DX12_LFI18S.fits
sidelobes_ringsetM=galactic_straylight_DX12_LFI18M.fits
sidelobes_ringsetS=galactic_straylight_DX12_LFI18S.fits
input_data_path=/mn/stornext/d14/bp/data/L2Data
window_len_hybrid=300
percentile_hybrid=0.95
window_len_dipole_minima=1200
window_len_dipole_maxima=400
window_len_slow_smoothing=300
percentile_slow_variations=0.99
min_range_dipole=0.0034
max_range_dipole=0.004
calibrated_data_basename=LFI_070_
gain_data_basename=LFI_070_
horn=18
level2_version=1
output_version=1
output_data_path=/mn/stornext/d14/bp/mathew/beyondplank/pipe/output/tods
ringset_path=/mn/stornext/d14/bp/data/ringsets/
                                                    38,1          All
```

Figure 3: An example calibration parameter file for horn 18

```
  file_pntperiod = /mn/stornext/d14/bp/data/madam/pp_new_30.dat
  file_gap = /mn/stornext/d14/bp/annastiina/DX12/mission/gaps_30GHz_DX12.dat
  kfilter = T
  base_first = 32
  temperature_only = F
  to_galactic = T
  firstPID = 1
  lastPID = 44000
  detector_1 = LFI27S
  detector_2 = LFI27M
  detector_3 = LFI28S
  detector_4 = LFI28M
  tod_0 = tod
  nside_map = 512
  simulation = /mn/stornext/d14/bp/mathew/beyondplank/pipe/output/iter0-200/simulation_30GHz.txt
  file_map = /mn/stornext/d14/bp/mathew/beyondplank/pipe/output/iter0-200/madam_map_30GHz_full.fits
  file_cov = /mn/stornext/d14/bp/mathew/beyondplank/pipe/output/iter0-200/madam_cov_30GHz_full.fits
  file_mask = /mn/stornext/d14/bp/mathew/beyondplank/pipe/output/iter0-200/madam_mask_30GHz_full.fits
```

Figure 4: An example madam parameter file for the 30GHz channels

```
#simulation
   simcomment    =
   fsample       =  32.5079365079365
   nofiles       =    1513 # number of files/detector/TOD component
#TOD component info
               tod_ID  weight  name
  tod_info     =  1  1.00000  tod
#detector info
#     #     #     det_ID  psi_pol  pol    sigma           slope       fknee       fmin    point_ID   name
    detector_info =  1      89.7  T   0.0016050498604963537 -0.92681   0.173682 1.15e-05  1       LFI27M
    detector_info =  2      0.0   T   0.0017263679342118218 -0.907266   0.109562 1.15e-05  1       LFI27S
    detector_info =  3      90.3  T   0.0018124331810740733 -0.929991   0.128468 1.15e-05  2       LFI28M
    detector_info =  4      0.0   T   0.0016309521073658605 -0.89933   0.0441449 1.15e-05  2       LFI28S
#
# TOD files
#             tod_ID   det_id
  path_tod  = 1
  file_tod =  1   1  /mn/stornext/d14/bp/mathew/beyondplank/pipe/output/tods/LFI_030_27M_L2_000_OD0091.h5
  file_tod =  1   1  /mn/stornext/d14/bp/mathew/beyondplank/pipe/output/tods/LFI_030_27M_L2_000_OD0092.h5
  file_tod =  1   1  /mn/stornext/d14/bp/mathew/beyondplank/pipe/output/tods/LFI_030_27M_L2_000_OD0093.h5
  file_tod =  1   1  /mn/stornext/d14/bp/mathew/beyondplank/pipe/output/tods/LFI_030_27M_L2_000_OD0094.h5
  file_tod =  1   1  /mn/stornext/d14/bp/mathew/beyondplank/pipe/output/tods/LFI_030_27M_L2_000_OD0095.h5
  file_tod =  1   1  /mn/stornext/d14/bp/mathew/beyondplank/pipe/output/tods/LFI_030_27M_L2_000_OD0096.h5
```

Figure 5: The top of the madam simulation file that corresponds to figure 4. The remainder is lists of files and has been omitted here.

```
fwhm_deconv=0
polarisation=T
conv_lmax=512
lmax_out=512
beammmax=100
beam_alm=FITS:/mn/stornext/d14/bp/data/beamalms/sl/sl_070_18_y_qucs-raa.alm
sky_alm=FITS:/mn/stornext/d14/bp/data/ringsets/alm/commander_gal_070-18M_full_map_TQU_deco2v2_gal.alm
ringset=FITS:/mn/stornext/d14/bp/mathew/beyondplank/pipe/output/iter0-69/sl_BP_LFI18M.fits
~
```

```
fwhm_deconv=0
polarisation=T
conv_lmax=512
lmax_out=512
beammmax=100
beam_alm=FITS:/mn/stornext/d14/bp/data/beamalms/totalAlm/totalAlm_070_18_y_qucs-raa.alm
sky_alm=FITS:/mn/stornext/d14/bp/data/ringsets/alm/commander_gal_070-18M_full_map_TQU_deco2v2_gal.alm
ringset=FITS:/mn/stornext/d14/bp/mathew/beyondplank/pipe/output/iter0-69/totalAlm_BP_LFI18M.fits
~
```

Figure 6: The totalconvolve parameter files for the sidelobes (top) and main beam (bottom) of detector 18M

4 Preliminary Maps

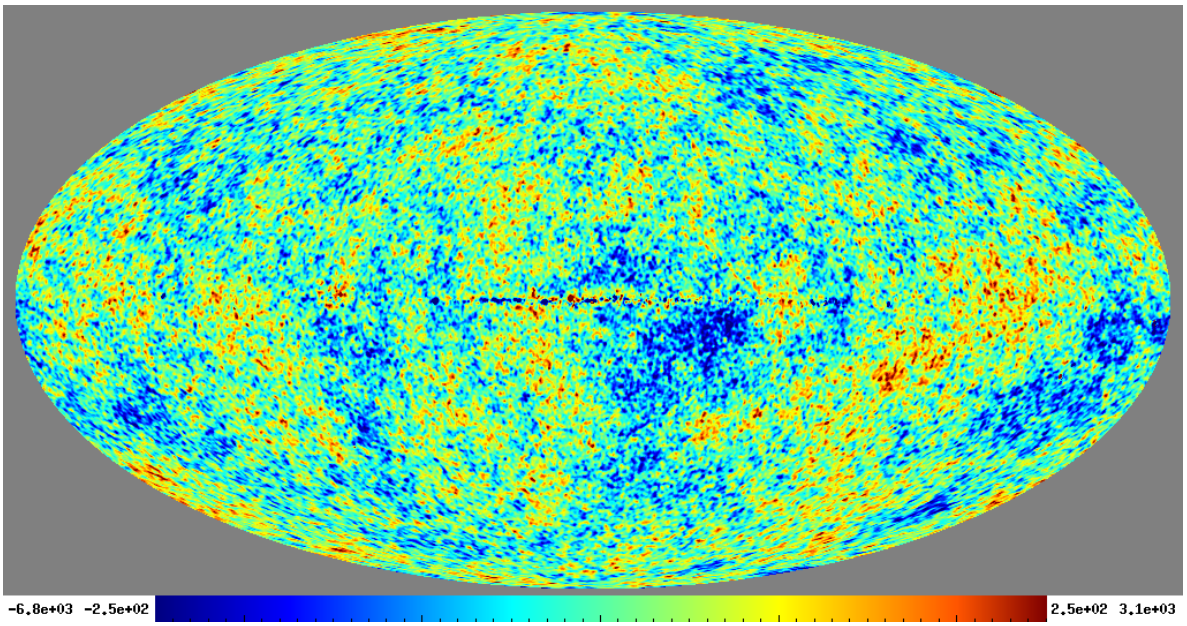

Figure 7: The CMB as seen from one of the 30GHz detectors

This section presents some preliminary pipeline outputs from the first iteration in the commander stage. These are not analysis quality products as the pipeline still has improvements that need to be made, detailed in the next section. Instead, these are included to demonstrate that the pipeline is operational and is producing sensible output at this stage.

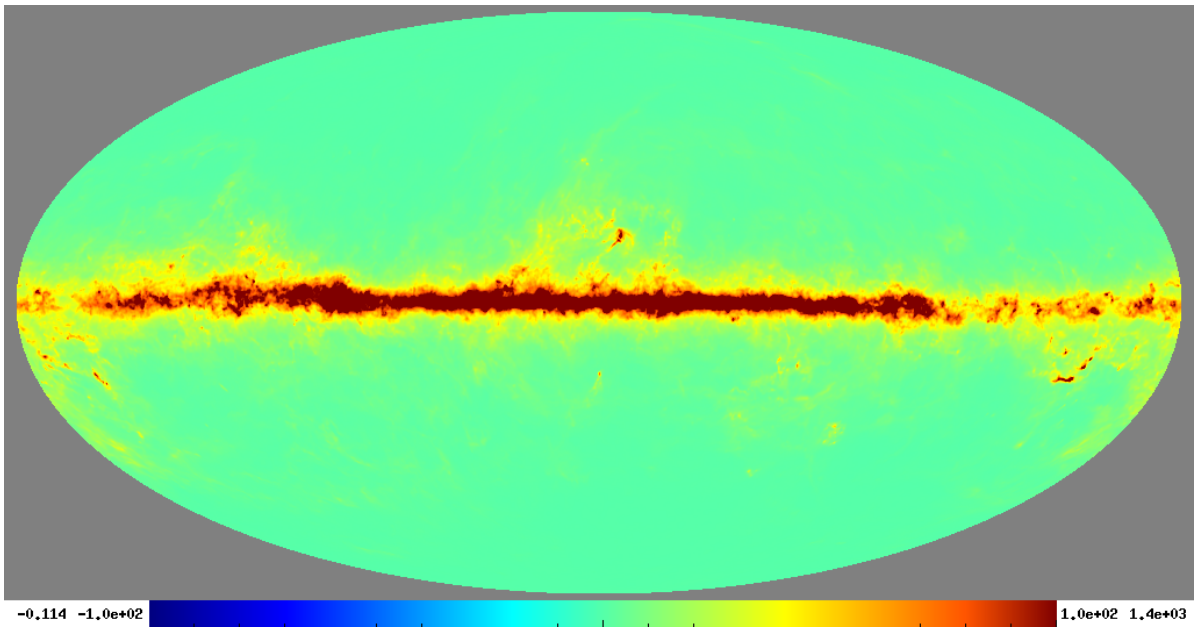

Figure 8: The dust map at 44 GHz

5 Next Steps

Now that the pipeline is operational, there are several things that must be done before the loop can be left to run. Firstly, the runtime of a single iteration the whole pipeline is currently about 2 days, broken up as follows:

Calibration: 1.5 days
Mapmaking: 5 hours
Commander: 8 hours
Misc: 1 hour

For efficiency of the loop, it would be ideal to be able to execute a full loop in around 8 hours, so that multiple iterations could be done per day. For this reason, the parameters going into each code are all currently being tuned to improve runtime, while still ensuring correctness of the outputs.

Secondly, there are still some improvements to be made on the commander runs. They should be updated to the release versions of the npipe maps when those become available. Additionally, the dipole needs to be re-estimated each iteration which in not currently being done. Finally, the T and P runs are currently independent, but could be linked through priors on dust and synchrotron amplitudes and betas.

Finally, thorough testing must be done of the entire, finalized pipeline to ensure that the results are sensible and are converging with each iteration. As the iterations get closer and closer to the optimal values, some parameters may have to be changed from their historical values as lower amplitude systematics are detected and fixed.