Modified SEEK

Lucas Collis Olivari

October 31, 2018

1 Introduction

SEEK is a flexible and easy-to-extend data processing pipeline for single-dish radio-telescopes. It takes an observed (or simulated) TOD in the time-frequency domain as an input and processes it into HEALPix maps, while applying calibration and automatically masking RFI. The data processing is parallelized using Ivy's parallelization scheme.

In the following, I describe a modified version of SEEK suited to process the TOD of BINGO or any other experiment with multiple horns. I first explain how install and run the software and then explore the general structure of it.

2 Installing the Modified SEEK

First, you will download and install SEEK. To do so just do the following:

i) git clone https://github.com/cosmo-ethz/seek

ii) cd seek

iii) pip install -r requirements.txt

iv) python setup.py install

Then you will download and install SEEK4BINGO. To do so just do the following:

i) git clone https://lolivari@bitbucket.org/lolivari/seek4bingo.git

ii) cd seek4bingo

To modify SEEK then type in a terminal:

i) python seek4bingo_install.py

The result will be a modified SEEK, i.e., the original SEEK with some files changed or added. See below for the details.

3 Running Modified SEEK: a Quick Guide

The spectrometer model is chosen in the "run_seek.py" file. It can either be 1 or 2 (see the details below for the description of these Fake BINGO models). The spectrometer gain template should be simulated **before** running SEEK. To do so go to the "data" directory and type "python make_fake_bingo_model_1" to simulate the model 1, for instance. Note that there are some parameters that must be chosen in these simulation files. Of course, you can add your own model and modify "run_seek.py" accordingly.

Note that, in the case that you want to have your calibration to perfectly correct for the gain, you must use the same template that you have used on your HIDE simulation. In this case, do not do the step above and just copy your gain template into the directory "data".

The configuration files are located in the "config" directory. To simulate the BINGO experiment, use the file "bingo.py". It is in this file that you should modify the main parameters of the data processing. Please, do not fill in the parameters that are going to be filled in by "run_seek.py" (they are are the blank ones).

To actually **run** the software just type: python run_seek.py

The **output** will be located in current path and will consist of a set of HDF5 files (one for each horn). Within each file, in turn, you will find a set of channel maps. An example of how to handle the HDF5 files is presented in the "plot_maps.py" file. If you want to combine the set of horn maps into one set of channel maps, use the "combine_maps.py" file (the details of how this combination is done can be found below).

4 Code Structure

I describe now the main modules that are needed to generate maps from TODs for BINGO.

4.1 Experimental Setup: BINGO

Please see HIDE's documentation for a detailed description of the experiment.

4.1.1 Code Implementation

SEEK, originally, assumes a single feed horn. To the best of my knowledge, however, a feed horn is just a device to increase the field-of-view of the experiment, with each one working as an independent telescope (**QUESTION: IS THIS CORRECT?**). In this case, the solution found by me has been:

i) Read the TOD of each horn as generated by HIDE or by the experiment backend;

ii) Produce several SEEK configuration files, one for each horn (saved as: "bingo_horn_i.py", where i is the number of the horn);

iii) Run SEEK several times, once for each configuration file, producing at the end maps for each horn separately (suggested notation: "bingo_maps_horn_modeḣdf", where mode is the number of the horn).

The use of SEEK, therefore, will be "hidden" from the user, who will use an simple python script (name: "run_seek.py") instead.

The inputs of SEEK will then be:

i) A standard BINGO configuration file (name: "bingo.py"), where the user will specify several parameters of the experiment.

ii) The TOD data.

iii) There are also several templates that must be provided at the directory "data" so the spectrometers (one for each horn) can be simulated (see below for details of how to simulate them).

Files of interest: "run_seek.py" from "seek" and "bingo.py" from "config".

4.2 Loading Data

The data is loaded from the file system (from the "bingo_data" directory) into memory. SEEK is currently able to process both FITS and HDF5 (default) data formats. The design concept of Ivy ensures that the other plugins do not depend on the origin of the data. Therefore extending the support for further file formats can be implemented without interfering with the other functionalities.

Files of interest: "find_nested_files.py" and "load_data.py" from "plugins".

4.3 Gain

A gain factor is computed either by using special calibration data that must be collected on dedicated calibration days in the survey or by using externally provided template can be loaded from the file system. This gain factor (actually its inverse) is applied to the TOD to convert the instrument-recorded values (ADU) to physical units (Kelvins).

Originally, HIDE uses data obtained with the Bleien telescope. I added two fakes BINGO gain templates as options to both HIDE and SEEK. Both assume a total bandwidth of 300 MHz and a channel width of 1 MHz as default.

In the first model, the gain is given by

$$G(\nu) = 1 + \epsilon(\nu) + \epsilon_{\text{osc}} \sin\left(\frac{\pi\nu}{\Delta\nu_{\text{osc}}}\right), \tag{1}$$

where $\epsilon(\nu)$ is a parameter that is drawn from a certain Gaussian distribution for each frequency ν and adds random errors to the gain, ϵ_{osc} is the oscillatory amplitude of the gain, and Δv_{osc} is a parameter that controls the period of the oscillation of the gain. Note that in this model the mean relation between Kelvin and ADU is 1.

In the second model, the gain is given by

$$G(\nu) = A\left(1 + \epsilon(\nu) + \epsilon_{\text{osc}} \sin\left(\frac{\pi\nu}{\Delta\nu_{\text{osc}}}\right)\right), \tag{2}$$

where A is the gain amplitude (the Bleien data mean value), $\epsilon(\nu)$ is a parameter that is drawn from a certain Gaussian distribution for each frequency ν and adds random errors to the gain, ϵ_{osc} is the oscillatory amplitude of the gain, and Δv_{osc} is a parameter that controls the period of the oscillation of the gain.

For more detail, see either "make_fake_bingo_model_1.py" or "make_fake_bingo_model_2.py" in the "data" directory.

Files of interest: "make_fake_bingo_model_1.py" and "make_fake_bingo_model_2.py" from "data", and "pre_process_tod.py" from "plugins".

4.4 Coordinate Transformation

The telescope pointing at each given time is connected to the TOD to give each pixel in the TOD a terrestrial coordinate. These coordinates are now transformed into equatorial coordinates corresponding to a given point in the celestial sphere (this information is also in the TOD originally).

Files of interest: "process_coords.py" from "plugins".

4.5 Masking Objects and Bad Frequency Channels

The TOD is masked if known bright objects such as the Sun and the Moon are too close to the telescope pointing. Furthermore, frequency bands known to be unusable (e.g., seriously contaminated by satellite communication bands) are masked.

This step **must** be investigated. As a default, it is **not** in the list of plugins of "bingo.py".

Files of interest: "mask_objects.py" and "mask_artefacts.py" from "plugins".

4.6 Masking RFI

The TOD is analyzed and pixels identified as contaminated by RFI are masked.

The SEEK package was designed to be able to process data that is heavily contaminated with RFI. SEEK's automated RFI masking mechanism is based on two steps: the SumThreshold algorithm and a Healpix pixel-based outlier rejection. (**QUESTION: WILL WE USE THIS PROCEDURE OR SHOULD WE CREATE AN ALGORITHM THAT IS BETTER SUITED FOR BINGO?**)

SumThreshold gradually builds a mask that flags the unwanted RFI in the data. The underlying assumption of the algorithm is that the astronomical signal is relatively smooth in both time and frequency (i.e., on the TOD plane), while RFI signals have sharp edges. Under this assumption, SumThreshold interactively improves a model of the true astronomical signal by smoothing the TOD with a Gaussian filter. It then clips pixels that lie above a certain threshold after subtracting this model from the data. The masking naturally starts with localized, strong RFI bursts. The threshold is progressively lowered in each iteration such that more RFI is detected and masked along the process. It also takes into account that RFI often extends in both time and frequency directions. That is, the algorithm combines neighboring pixels and masks them if their sum exceeds a threshold. Due to this underlying assumption SumThreshold has to be applied with precaution to spectral line data, which can show non-smooth signatures. RFI originating from radio point sources, such as planes or satellites is well captured. However, depending on the instrument configuration natural point sources could be incorrectly masked by the algorithm.

Finally, SEEK applies a morphological dilation of the mask, i.e., it enlarges the mask depending on the density of masked pixels in a region in order to capture possible missed RFI leakage. The iterative Gaussian smoothing in the presence of a mask is the most computationally intensive part. In order to accelerate the process while keeping the code fully Python, SEEK use the HOPE just-in-time compiler package, which translates the Python code into C++ and compiles at runtime. This allows SEEK to perform the RFI mitigation step at a rate of 190-200 GB/h/CPU, which is faster than other existing Python implementations of the similar algorithm.

The second step in the RFI mitigation is applied during the HEALPix map-making process and it is done in order to remove outliers.

This step **must** be investigated. As a default, it is **not** in the list of plugins of "bingo.py".

Files of interest: all files from "mitigation" and "remove_RFI.py" from "plugins"

4.7 Baseline Removal

The baseline level per frequency is estimated from the median value over time of the cleaned TOD. This baseline is subtracted from the TOD.

This step **must** be investigated. As a default, it is **not** in the list of plugins of "bingo.py". It will be interesting to see if is not better to remove the baseline at the component separation step.

Files of interest: "background_removal.py" from "plugins".

4.8 Map Making: TODs to Horn-Frequency Maps

For every horn and frequency channel the TOD is processed into a HEALPix map. By default, each pixel in the HEALPix map is filled with the mean value of all measurements in that pixel. One can also invoke an outlier-rejection step to avoid uncleaned RFI contaminating the mean.

Files of interest: all files of "mapmaking" and "create_maps.py" from "plugins".

4.9 Map Making: Combining the Horn Maps

As BINGO will observe the sky with the help of feed horns, we must combine the horn-frequency maps into a single set of frequency maps. To do this, I chose to fill the pixels with the mean of all measurements in that pixel.

Files of interest: "combine_horn_maps.py". from "seek".