A User’s Guide for the

GBT Mapping Pipeline

Jim Braatz and Joe Masters (NRAO)

March 6, 2015

Table of Contents

Introduction 2

Getting Started 2

A First Look at the Mapping Pipeline 3

Using the Pipeline with VEGAS 4

Basic Operation of the Pipeline 4

Pipeline Options 5

Some Considerations on Pipeline options 7

Advanced Operation of the Pipeline 8

Stage 1: Calibration 8

Stage 2: File Conversion 9

Stage 3: Combining Data and Imaging 9

Flagging Data 10

Channel limitations with VEGAS data 11

Imaging Examples 11

Example: Combining Data from Multiple Maps 11

Example: Mapping Pre-calibrated Data 12

Example: Reducing channels prior to mapping VEGAS data 12

Example: Line Maps and Temperature from NH3(1,1) and NH3(2,2) 13

Optimal Observing Configuration for the Mapping Pipeline 15

Subtracting Spectral Baselines 16

Cleaning Up Disk Space 16

A Word on Observations in Galactic Coordinates 17

Units in Image FITS files 17

Pipeline Development and Help 17

Appendix 1: Calibration Equations 18

# Macintosh HD:Users:jmasters:Desktop:w51.png

# Introduction

This document describes how to use the GBT Mapping Pipeline. The pipeline calibrates and grids GBT spectral line mapping data, and creates FITS image cubes.

The pipeline was developed initially to support mapping observations made with the KFPA, but it can be applied to data taken with any of the receivers that use standard position-switched or frequency-switched observing modes. Pipeline tools can also help with reducing other types of data, such as W-band mapping observations. The pipeline can process data from either VEGAS or the GBT spectrometer.

The pipeline is designed to require minimal user input when data are taken using standard *astrid* mapping procedures. An example astrid script that makes pipeline-ready data is described later in this document. However, the pipeline is also versatile, and users in some cases will get the best results by tuning the pipeline parameters for their specific data set.

# Getting Started

To run the pipeline you must be a member of the “aipsuser” unix group. You can check whether you are a member by typing “groups” at the command line. If “aipsuser” is not listed, contact the GB helpdesk to have GB computing add you to the group. The NRAO helpdesk is accessible at: [http://help.nrao.edu](http://help.nrao.edu/)

Where to Run the Pipeline: Currently, the GBT pipeline has certain dependencies and it runs only on 64-bit, RedHat 6 computers in Green Bank. The computer *arcturus.gb.nrao.edu* is reserved for pipeline use. We recommend that you use *arcturus* for all pipeline work at this time.

Disk Space: Due to the typically large datasets produced by the pipeline, especially for VEGAS data, we recommend you make use of the pipeline scratch area.

/lustre/pipeline/scratch/<user\_id>/

where “<user\_id>” is your linux user ID, e.g. “jbraatz”. To create your own directory, at the command line type:

mkdir /lustre/pipeline/scratch/<user\_id>

# A First Look at the Mapping Pipeline

You can run the pipeline using the command:

% gbtpipeline

For quick help, type the command with no parameters.

Here is a sample pipeline execution using KFPA position-switched test data that observed NH3 toward W51 using the spectrometer:

% gbtpipeline -i /home/gbtpipeline/reference-data/TKFPA\_29/TKFPA\_29.raw.acs.fits -m 14:24 --refscan 13 --units Jy

This command maps data from scans 14-24, using scan 13 as the reference observation. The data are calibrated to units of Jy. The program creates several data products, including three maps stored as FITS files, calibrated spectra from each spectrometer sampler in separate FITS files, and logs of the pipeline execution. The output maps are:

W51\_14\_24\_23706\_MHz\_cont.fits – a single-channel “continuum” map

W51\_14\_24\_23706\_MHz\_cube.fits – a multi-channel data cube

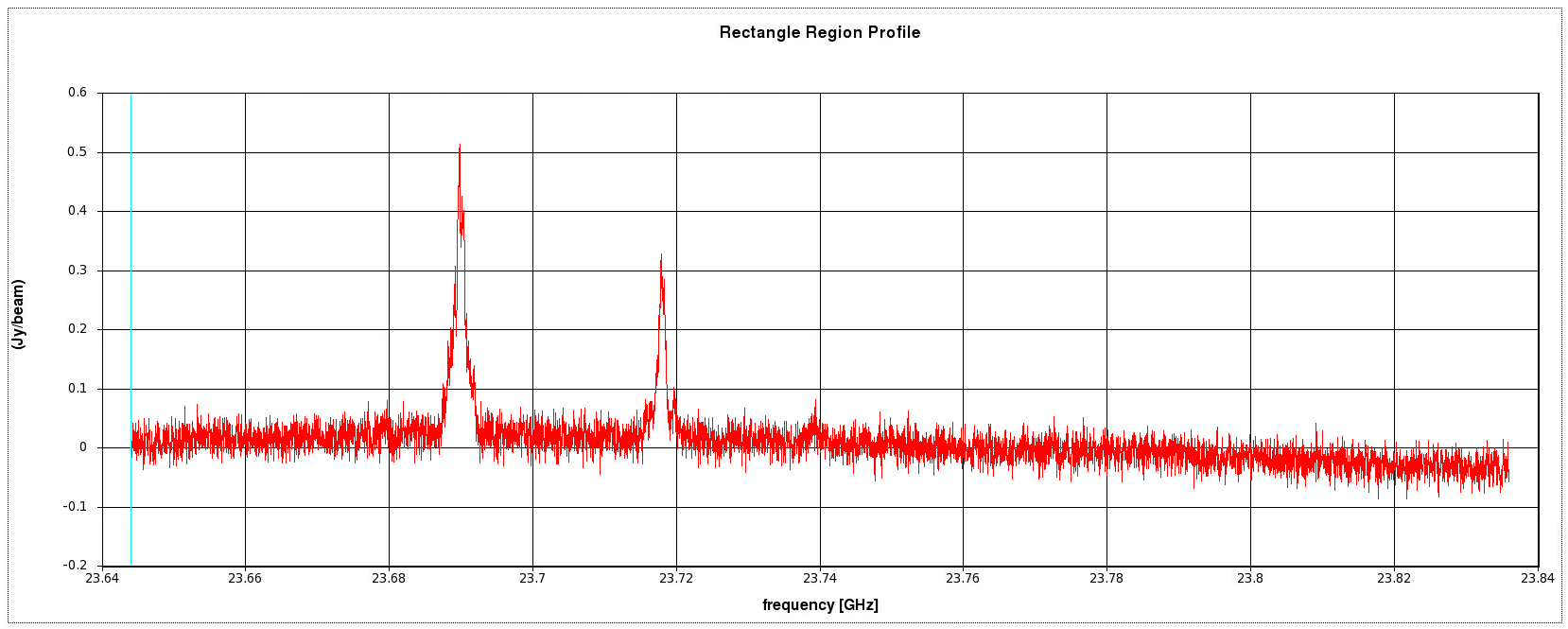
W51\_14\_24\_23706\_MHz\_line.fits – a continuum-subtracted data cube

The “continuum” map in fact is an average of all the spectral channels, and has no blanking of spectral data. So it includes both the continuum and any spectral lines in the cube. The “cube” includes both continuum and spectral data for each channel. The “line” data has a nominal continuum level subtracted. The continuum level is approximated, in this case, by looking at the flux level in channels near the edges of the spectral axis. If you want a careful continuum subtraction, you will need to modify the pipeline scripts (described later in this document).

You can examine these data cubes using the CASA viewer, e.g. :

% casaviewer W51\_23740\_MHz\_line.fits

Select the menu option “Tools -> Spectral Profile”, then left-click on the crosshair icon, and left-click on the map. This will display a spectral cut through the cube.



# Using the Pipeline with VEGAS

The pipeline works with observations using either the spectrometer or VEGAS. The syntax is identical between the two. The primary difference relating to the pipeline is that spectrometer data are filled into a single FITS file, while VEGAS data get filled into a directory of FITS files. So a pipeline call using VEGAS data can reference either the top-level directory, or one of the FITS files within that directory.

# Basic Operation of the Pipeline

The input to the pipeline is SDFITS data generated by the *sdfits* filler, from a GBT mapping observation. From that, the pipeline generates image cubes. The pipeline operates in three basic stages.

1. The first stage calibrates the spectral line data and produces FITS files compatible with the GBTIDL “keep file” format of SDFITS.
2. The second stage uses the utility program *idlToSdfits* to convert these SDFITS files into a FITS format that can be read by AIPS.
3. The third stage uses AIPS to combine and grid these calibrated data and produce the output data cubes.

In typical use, a single call to the *gbtpipeline* command executes all three stages of operation. You can specify the range of scans to be mapped and identify the reference scans used for calibration, as in the NH3 mapping example above. A standard position-switched mapping observation begins with a reference scan, executes the mapping scans, and then (optionally) finishes with a second reference scan, all using a common VEGAS (or spectrometer) configuration. The pipeline understands this sequence and will calibrate the mapping scans using the adjacent reference scans, even if the mapping and reference scans are not identified specifically in the call line. Or, you can specify the mapping scans and reference scans explicitly. When two reference scans are used for calibration, the reference data for each integration of the map will be determined by interpolating between the two reference scans.

Here is an example where the pipeline automatically detects mapping and reference scans within the FITS file:

% gbtpipeline -i /home/gbtpipeline/reference-data/TKFPA\_29/TKFPA\_29.raw.acs.fits

And here is an example pipeline call explicitly setting the mapping and two reference scans:

% gbtpipeline -i /home/gbtpipeline/reference-data/TKFPA\_29/TKFPA\_29.raw.acs.fits -m 14:24 --refscan 13,26

In the case of this FITS file, the preceding two examples produce the same maps.

## Pipeline Options

Here are the standard options available from the *gbtpipeline* command line:

**general arguments:**

-h, --help show a help message

**Input:**

-i INFILENAME, --infile INFILENAME

SDFITS file name containing map scans

**Data Selection:**

-m MAPSCANS, --map-scans MAPSCANS

map scans to process, e.g. "10:20,30:40" identifies two ranges of

11 scans each

--refscan REFSCANS, --refscans REFSCANS

reference scan(s) used for calibration, e.g. "4,13" to identify two

reference scans

-f FEED, --feed FEED feeds, e.g. "0:6" identifies feeds 0 through 6

-p POL, --pol POL polarizations, e.g. "0,1"

-w WINDOW, --window WINDOW

spectral windows, e.g. "0,1,2,3"

-c CHANNELS, --channels CHANNELS

channel selection, e.g. "100:200". (Passed directly to

idlToSdfits)

**Control:**

--imaging-off If set, will calibrate data and write calibrated

SDFITS files but will not create image FITS files.

-a AVERAGE, --average AVERAGE

average the spectra over N channels (passed directly

to idlToSdfits)

-n N, --rms-flag N flag integrations with RMS noise > N kelvin (passed

directly to idlToSdfits)

--median-baseline-subtract N

subtract median-filtered baseline, with a median

window half width of N channels (passed directly to

idlToSdfits)

--display-idlToSdfits-plots

display idlToSdfits plots of calibrated spectra

**Calibration:**

-u {ta,ta\*,tmb,jy}, --units {ta,ta\*,tmb,jy}

calibration units. Default: tmb

--no-sky-brightness-correction

Disable adjustment for sky brightness.

--spillover-factor SPILLOVER

rear spillover factor (eta\_l). Default: .99

--aperture-efficiency APERTURE\_EFF

aperture efficiency for spectral window 0. Other

window efficiencies are adjusted to this value.

(eta\_A) Default: .71

--main-beam-efficiency MAINBEAM\_EFF

main beam efficiency for spectral window 0. Other

window efficiencies are adjusted to this value.

(eta\_B) Default: .91

--beam-scaling BEAMSCALING

comma-separated beam scaling factors for each feed and

polarization in the order 0L, 0R, 1L, 1R, etc.

-t ZENITHTAU, --zenith-opacity ZENITHTAU

zenith opacity value (tau\_z). Default: determined from

GB weather prediction tools.

--smoothing-kernel-size SMOOTHING\_KERNEL

boxcar kernel size for reference spectrum smoothing. A

value <= 1 means no smoothing. Default: 3

**Output:**

-v VERBOSE, --verbose VERBOSE

set the verbosity level -- 0-1:none, 2:errors only,

3:+warnings, 4(default):+user info, 5:+debug

--clobber Overwrites existing output files if set.

--keep-temporary-files

If set, do not remove intermediate imaging files.

## Some Considerations on Pipeline options

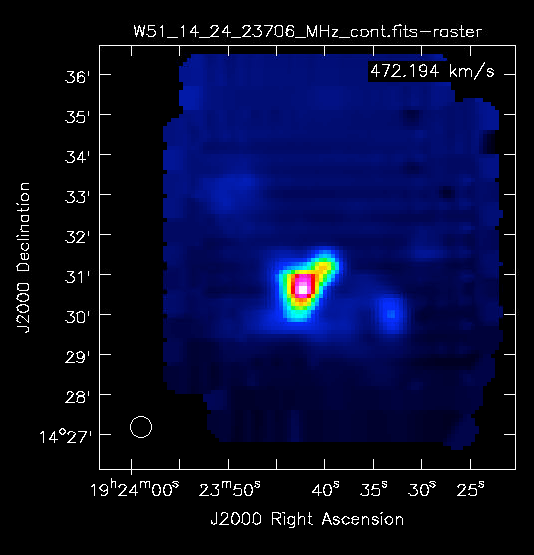
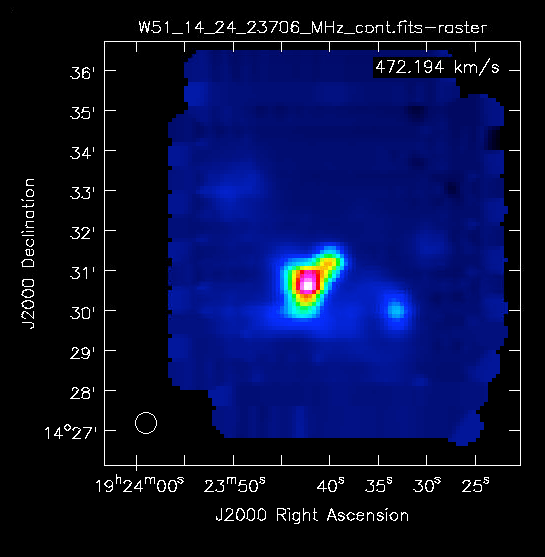
**Reference Channel Smoothing:**

By default, the pipeline applies boxcar smoothing to reference spectra prior to calibration. This is an effective technique for improving signal-to-noise in the calibrated spectra. However, it can strongly emphasize birdies and other defects in the spectra, so in some instances it may be better to turn this option off. The smoothing kernel size has a default of 3 channels, and is controlled with the “--smoothing-kernel-size” parameter. To turn off smoothing, set “--smoothing-kernel-size=0”.

**Sky Brightness Correction:**

Total power position-switched mapping observations should include observations of a reference position immediately before and (optionally) after the mapping scans. The reference observation may take place at an elevation that is not necessarily representative of the mapping scan elevations. Moreover, if the reference observations take place many minutes apart, changes in the weather conditions can lead to a poor match between the signal (map) scans and the reference scans. If the calibration does not correct for these effects, maps can have artifacts in the form of stripes and a north-south gradient in the calibrated power. The pipeline includes, by default, a correction for this effect. The effect is particularly important at weather-sensitive frequencies (e.g. K-band and above) and it is pronounced for observations that use only a single reference observation. The correction can be turned off with the “--no-sky-brightness-correction” option.

*Without* the correction: *With* the correction.

**Tcal scaling for KFPA:**

The calibration temperatures (*Tcal*) of noise diodes used in GBT receivers are measured periodically by the observatory, and applied to the data during calibration. These calibration temperatures can drift over time, leading to uncertainties in the measured flux scale. To correct for this scaling offset, the pipeline allows you to set per-beam adjustments at the command line (--beam-scaling). Each of the beam-scaling values (one for each feed and polarization pair) acts as a scaling factor on the observatory-measured *Tcal* values. If you wish to apply these calibration corrections, you should measure the beam-scaling factors by observing a calibrator during your observations. Typically, this is done in one of two ways:

1. Pointing each of the beams at a known flux calibrator source.
2. Observing an extended source with a known flux (e.g. the moon), using all beams.

You should consult your GBT project friend to determine the best calibration strategy.

By default, the pipeline does not apply any scaling factor to the beams. If you do measure and apply a beam-scaling factor, note that the pipeline nevertheless records the observatory-measured *Tcal* values in the output SDFITS spectra.

At the moment, the pipeline only applies corrections for the KFPA. Future releases will allow corrections for other GBT receivers.

**Mapping Beyond the KFPA:**

The pipeline was developed primarily to support KFPA observations, but it works well with maps observed with other receivers as well. It is important, though, to be aware that certain pipeline features are not optimized for other receivers. For example, there is no intrinsic flagging for RFI that can affect maps observed at lower frequencies, and calibration defaults are tuned to sensible values for the KFPA. Calibration parameters can be set with command-line options.

# Advanced Operation of the Pipeline

Basic usage of the *gbtpipeline* command, as described above, should meet the data reduction needs of many projects. Sometimes, however, you may need to customize the mapping results or take advantage of certain features that are not available from the standard *gbtpipeline* parameters. In this case, you can run any of the three stages of the pipeline separately, customizing the results at each stage.

### Stage 1: Calibration

The first stage of the pipeline calibrates the data and creates an SDFITS file for each sampler in the backend. You can halt the *gbtpipeline* process after Stage 1 by specifying the command-line option “--imaging-off”.

As an alternative to using the pipeline for calibration, you may use GBTIDL to produce the SDFITS files. GBTIDL gives you detailed control of the calibration on an integration-by-integration basis. The SDFITS file that you generate must contain one record for each integration in the map, and each record should contain a calibrated spectrum. You can either create one SDFITS file with all the data, or multiple SDFITS files containing partial data sets (for example, separating feeds into separate SDFITS files for individual processing). Separate SDFITS files are easily combined for gridding in a later step.

### Stage 2: File Conversion

With an SDFITS file in hand, you must convert the file to an AIPS-compatible format using the utility program *idlToSdfits*. A typical execution will look like:

% idlToSdfits –l –o myfile.aips.fits myfile.fits

The “-l” option suppresses the creation of a plot file. The *idlToSdfits* program has a host of additional options that can be examined by typing “idlToSdfits” at the linux prompt, with no parameters.

The product of Stage 2 is one “AIPS FITS” file for each SDFITS file from Stage 1.

### Stage 3: Combining Data and Imaging

The third step is to combine and image the “AIPS FITS” files generated in Stage 2 using AIPS. We access the AIPS imaging routines using a python interface. Here we describe that process.

**aipspy scripts:** The pipeline includes a utility program called *aipspy* that runs ParselTongue scripts in AIPS. To use ParselTongue scripts, you need to know the AIPS ID used on your behalf by the pipeline. Get the ID (which is just a number) from the linux prompt using:

% id –u

There are several *aipspy* scripts in the standard gbtpipeline distribution. Two important scripts are:

1. load.py : Load one or multiple data files into AIPS in advance of mapping
2. image.py : Map data in AIPS

So, you map the data by executing the following two linux commands:

% aipspy /home/gbtpipeline/release/load.py <aipsid> <file1.aips.fits> [<file2.aips.fits>]

% aipspy /home/gbtpipeline/release/image.py <aipsid>

The result of these commands is three FITS files, one with the full image data cube, one with a “continuum” map, and one with a spectral line map.

The “load” step must be executed prior to the “image” step, and you should use only one “load” to combine all of your FITS files intended for the map. You can specify as many AIPS FITS files as you need, when running the “load” step. These will be combined prior to imaging. You can get online help for either of these commands as follows:

% aipspy /home/gbtpipeline/release/load.py -h

% aipspy /home/gbtpipeline/release/image.py -h

The image script allows you to average channels prior to imaging, specify the center position of the map in RA and DEC, specify the size of the map in pixels, and specify the rest frequency of the line observed.

# Flagging Data

There are currently some restrictions on how data flagging must be accomplished with the pipeline. If you have data that has RFI in some channels, or other defects that you would like to flag, you will probably want to process your data in GBTIDL for calibration or just for the data correction. Unfortunately it is not possible to use the GBTIDL *flag* command at this time to eliminate the bad data. This limitation is introduced because flagged data are represented as NaN, and the *idlToSdfits* program rejects spectra if any of the channels have a NaN value. The *idlToSdfits* program is being rewritten for a future release. Until then, to exclude bad data from the imaging process you have a few options:

1. You can use the idlToSdfits channel filtering ‘-c’ option to select only good channels intended for imaging. This is an effective method if you are imaging contiguous channels that are clean from defects.

2. Similarly, you could use the GBTIDL procedure *sdextract* to extract certain channels for imaging. An example of using this procedure is given later.

3. You could replace bad channels with values interpolated from neighboring channels using the GBTIDL *replace* procedure.

For example, to replace a bad data value in channel 100 with interpolated values from the neighboring channels you'd do:

replace,99,101

GBTIDL documentation can be found here: [http://www.gb.nrao.edu/GBT/DA/gbtidl/release/user/](http://www.gb.nrao.edu/GBT/DA/gbtidl/release/user/" \t "_blank)

Of course, removing defects can be a tedious process if there are lots of bad data. Writing scripts in GBTIDL for data replacement is a good option.

# Channel limitations with VEGAS data

Currently, the pipeline is not able to process data sets with more than 32K channels per spectrum. Some VEGAS modes have many more channels than this. Again the *idlToSdfits* conversion program introduces this limitation, so when that program is replaced the limitation will be lifted.

Regardless of the limits, observers should note that it can take a long time to image data sets with a lot of channels, so it is generally good practice to average the data to a sensible channel spacing and limit imaging to only the channel range of interest.

You can reduce the number of channels in your data either by extracting a set of channels from the full data set, or averaging channels, or both. These operations are accomplished with the GBTIDL utility program *sdextract*. The procedure takes the following arguments:

sdextract, <sdfin>, <sdfout>, <startat>, <endat>, <boxwidth>, <clobber>

1. <sdfin> : the input sdfits file
2. <sdfout> : the output sdfits file
3. <startat> : start channel to extract
4. <endat> : end channel to extract
5. <boxwidth> : width of boxcar smooth
6. <clobber> : if True, delete an existing sdfout output file before writing

For example, to create a new data set with some channels extracted and a 4-channel boxcar smoothing across the selected channel range:

GBTIDL -> sdextract, ‘my\_cube.fits’,’avg\_cube.fits’, startat=1600, endat=1679, boxwidth=4

Below we give an example that shows how to process VEGAS data that exceeds the 32K channel limit (the “Reducing Channels” example).

# Imaging Examples

Here we give examples of several common examples of pipeline use.

## Example: Combining Data from Multiple Maps

Suppose you have several data sets from independent executions of your GBT mapping observation. You can combine and map the data using the following procedure.

First run the pipeline on each of the maps separately, specifying the “--imaging-off” option. Each execution will generate a set of SDFITS files.

gbtpipeline –i my\_sample\_data.fits –m 10:19 --refscan 9 --imaging-off

gbtpipeline –i my\_sample\_data.fits –m 21:30 --refscan 20 --imaging-off

Next create the aips files for each FITS file generated in the previous step:

idlToSdfits –l –o file1.aips.fits file1.fits

idlToSdfits –l –o file2.aips.fits file2.fits

Now combine the data using the *aipspy* command with load.py:

aipspy /home/gbtpipeline/release/load.py <aips\_id> file1.aips.fits file2.aips.fits

Finally you can map the data using the *aipspy* command with image.py:

aipspy /home/gbtpipeline/release/image.py <aips\_id>

## Example: Mapping Pre-calibrated Data

Under certain circumstances, you may wish to calibrate and process your data in GBTIDL rather than calibrate with the pipeline. For example, the pipeline cannot currently calibrate W-band data, since observations with that receiver use a unique observing sequence for calibration. Or, you may need to perform specific calibration and processing steps, such as reducing the number of channels or fitting a complex spectral baseline. To process data such as this, you must first calibrate the data in GBTIDL and produce a “keep file” that has one record per beam, per integration in the map.

You can then grid the data with the following steps. First, convert the *keep* file to a FITS format (aips file) that can be read by the AIPS FITS reader:

% idlToSdfits –o my\_aips\_file.aips.fits my\_keep\_file.fits

Next, load the data into AIPS with the *load.py* script:

% aipspy /home/gbtpipeline/release/load.py <aipsid> my\_aips\_file.aips.fits

And finally map the data with the *image.py* script:

% aipspy /home/gbtpipeline/release/image.py <aipsid>

## Example: Reducing channels prior to mapping VEGAS data

Here is an example of how to reduce the number of channels in a data set prior to mapping. This example refers to a real sample VEGAS data set stored on the GB file system.

First calibrate the data with the pipeline:

% gbtpipeline –i /home/gbtpipeline/reference-data/TREG\_140917.raw.vegas –-imaging-off

Next reduce the number of channels:

% gbtidl

GBTIDL -> sdextract, ‘W3OH\_scan\_140\_145\_window0\_feed0\_pol0.fits’, ’reduced\_pol0.fits’, startat=65000, endat=65999, boxwidth=10

GBTIDL -> sdextract, ‘W3OH\_scan\_140\_145\_window0\_feed0\_pol1.fits’, ’reduced\_pol1.fits’, startat=65000, endat=65999, boxwidth=10

You can then grid the data with the following steps. First, convert the *keep* files to the AIPS FITS format:

% idlToSdfits –o reduced0.aips.fits reduced\_pol0.fits

% idlToSdfits –o reduced1.aips.fits reduced\_pol1.fits

Then load the data into AIPS with the *load.py* script:

% aipspy /home/gbtpipeline/release/load.py <aipsid> reduced0.aips.fits reduced1.aips.fits

And finally map the data with the *image.py* script:

% aipspy /home/gbtpipeline/release/image.py <aipsid>

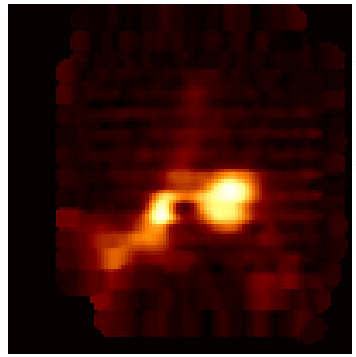
## Example: Line Maps and Temperature from NH3(1,1) and NH3(2,2)

Here we give an example of using *aipspy* scripts to generate line maps and a temperature map from the NH3 data. Glen Langston contributed the scripts. We make line maps by restricting the spectral channels used in the map to only those near the NH3 line centers. We use sample data observed with the spectrometer.

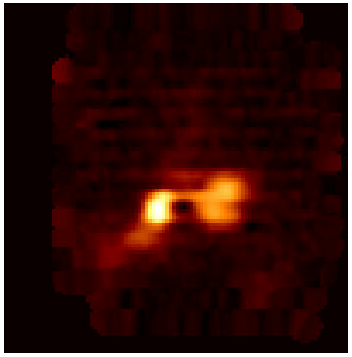
Here are the commands to run on arcturus:

% gbtpipeline -i /home/gbtpipeline/reference-data/TKFPA\_29.raw.acs.fits -m 14:24 --refscan 13 -a 3 -v 4

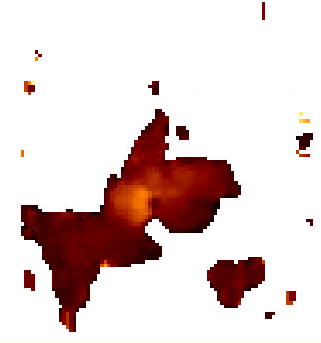
% aipspy /home/gbtpipeline/release/contrib/sumLine.py <aipsid> W51-NH3-11 23694.506 60. 20.



% aipspy /home/gbtpipeline/release/contrib/sumLine.py <aipsid> W51-NH3-22 23722.6336 60. 20.



% aipspy /home/gbtpipeline/release/contrib/tempNH3\_1122.py <aipsid> W51 60. 10. .2



Remember you must replace <*aipsid*> with your unique ID. The first command is the pipeline mapping command we encountered already in our initial example. The second and third commands use the sumLine.py script to generate single-channel maps averaged through the line centers from the NH3(1,1) and NH3(2,2) lines. The parameters of the sumLine.py procedure are:

sumLine.py <aipsNumber> <fileName> <restFreq> <lineVel> <lineWid>

1. <aipsNumber> : the AIPS ID
2. <fileName> : a name for the output file
3. <restFreq> : the rest frequency of the line in MHz
4. <lineVel> : the line center velocity in km/s, and
5. <lineWid> : the line FWHM in km/s.

The fourth command in the sequence generates the temperature map from the original line cube using the temp\_NH3\_1122.py script. The parameters for this command are:

tempNH3\_1122.py <aipsNumber> <fileName> <restFreq> <lineVel> <cutOff>

1. <aipsNumber> : the AIPS ID
2. <filename> : a name for the output file
3. <lineVel> : the line center velocity in km/s
4. <lineWid> : the line width (FWHM) to average, and
5. <cutOff> : the minimum line intensity to use in the line ratio (smaller values result in blanked pixels in the temperature map)

These python scripts are available in the */home/gbtpipeline/release/contrib* directory. Besides being useful in their own right, they also serve as good examples to help you develop your own scripts.

# Optimal Observing Configuration for the Mapping Pipeline

Previous examples demonstrated how you could identify specific mapping scans and reference scans for the pipeline’s calibration sequence. By specifying mapping and reference scans explicitly in this manner, you have direct control over the calibration process. An alternative is to allow the pipeline to determine mapping and reference scans automatically. If you do not specify scan ranges at the command line, the pipeline will examine the entire input data file and attempt to process each mapping observation it encounters there. The pipeline looks at the observing sequence and uses header information to try to identify reference scans with mapping scans.

The following sample astrid script demonstrates a KFPA observing sequence. The pipeline can properly calibrate observations from this sequence without the need to identify mapping and reference scans in the parameter list.

# Example Astrid script demonstrating KFPA Mapping for use with pipeline

# We assume that the telescope has been configured prior to running this script

# Identify the source catalog

Catalog(“/home/userid/mycatalog.cat”)

target = "W51" # define mapping target

off = "W51-Off" # define a map reference location, with no emission

Slew(target)

Balance()

# Check the levels are correct

Break("Check Balance”)

#perform a total power observation at the reference location

Track( off, None, 30.0, "1")

#Map

RALongMap( target,

Offset("Galactic", 0.33, 0.0), # This is a galactic coordinate map

Offset("Galactic", 0.0, 0.10),

Offset("Galactic", 0.0, 0.008),

140.0, "1")

#perform the final total power reference obs

Track( off, None, 30.0, "1")

The pipeline command to image such an observation is simply:

gbtpipeline –i my\_sample\_data.fits

# Subtracting Spectral Baselines

A typical GBT mapping observation uses short integration times per pixel (a few seconds), and the resulting spectral baselines are flat compared to the thermal noise. The pipeline therefore uses a very basic spectral baseline subtraction, implemented with the AIPS task *IMLIN* in the *image.py* script. The default behavior is to fit a DC baseline (polynomial of order 0) determined from the edges of the spectrum. Specifically, the baseline is fit to channels [0.04, 0.12]\* nchan and [0.81,0.89]\*nchan. You can modify this behavior by editing the *image.py* script directly. Follow these steps:

1. cd to the directory with your mapping data
2. copy /home/gbtpipeline/release/image.py to the current directory
3. edit image.py appropriately (search for the “imline” command)
4. re-run the pipeline with the “--imaging-off” option
5. run “aipspy /home/gbtpipeline/release/load.py <aipsid> <file.aips.fits>”
6. run “aipspy image.py <aipsid>” (note we are using the local copy of image.py)

A future upgrade to the pipeline will incorporate baseline-fitting parameters directly in the *gbtpipeline* command line.

For more detailed control over the baseline fitting and other aspects of the calibration and flagging, you can use GBTIDL to calibrate the data and use pipeline tools for “Stages” 2 and 3, that were discussed in the “Advanced Operation” section above.

# Cleaning Up Disk Space

The pipeline creates large data files, and may completely fill the available disk space if care is not taken to clean up unused files. You should remove all intermediate files that are not needed for additional processing. This includes “aips” files generated during the calibration of data, and AIPS files used in intermediate stages of gridding the data. A utility is available to clean up old AIPS files, and this should be run after each use of the pipeline.

% aipspy /home/gbtpipeline/release/tools/clear\_AIPS\_catalog.py <aips\_id>

Recall that you can find your AIPS ID by the linux command “id –u”.

Note: All files stored under the AIPS ID used by the pipeline will be removed with this operation, so it is important that you not store any non-pipeline AIPS files under this AIPS ID.

# A Word on Observations in Galactic Coordinates

For observations made using Galactic Coordinates, pipeline-generated FITS cubes have an error in the FITS header. Process the data through the pipeline as usual, and then fix the headers of the cubes using the *setGalCords* tool. Example:

% /home/gbtpipeline/release/tools/setGalCoords my\_image\_cube.fits

# Units in Image FITS files

The pipeline imaging engine, AIPS, always writes BUNIT=Jy/Beam to the header of FITS image cubes regardless of the actual units specified in the calibration. Despite the header, the values in the image are correct. If you calibrated your data to units other than Janskys and wish to correct the header, you can use the unix tool *fv*, the *fthedit* procedure in IDL’s modfits.pro or the python package *pyfits*.

# Pipeline Development and Help

The pipeline is a stable software product, but it is being actively improved. The pipeline developers are Joe Masters ([jmasters@nrao.edu](mailto:jmasters@nrao.edu)), Bob Garwood ([bgarwood@nrao.edu](mailto:bgarwood@nrao.edu)) and Jim Braatz ([jbraatz@nrao.edu](mailto:jbraatz@nrao.edu)).

You can get help on pipeline-related issues by using the NRAO Helpdesk.

<https://help.nrao.edu>

The pipeline team uses a mailing list (gbtpipeline-announce@nrao.edu) to keep pipeline users abreast of news and developments. Ask Joe to be on it!

# Appendix 1: Calibration Equations

The pipeline uses the following set of equations for calibration.

Calibration temperature

where is the lab-measured calibration temperature and *F* is the a scaling factor, usually determined by calibration scans on a source with a known flux. A scaling factor, *F*, is a user-provided correction factor determined for each beam (i.e. for each unique feed and polarization pair). *F* defaults to the value 1 for all feeds. The *Tcal* values in the input and output SDFITS files are the lab-measured values.

System temperature

In order to minimize edge effects in calculating system temperature, 10% of the channels

at each edge are eliminated. The notation *ref80* refers to the central 80% of the reference

data. The angle brackets represent averaging over channels. A single calibration

temperature (is applied for the entire bandpass, averaged over the frequencies observed.

Antenna temperature

where and the calculation is done on a channel-by-channel basis. The notation indicates that the system temperature calculation comes from the reference data, when the telescope is off source (position switching) or off frequency (frequency switching).

Corrected antenna temperature

where is the correction for sky brightness variation between the reference and current integration, is the elevation-adjusted atmospheric opacity and is the correction factor for rear-spillover, ohmic loss and blockage efficiency. The zenith opacity ( is determined at every integration from weather modeling code provided by Ron Maddalena.

Main beam brightness

where is the main beam efficiency. For the KFPA on the GBT, .

Flux density (Jy)

where is the (frequency-dependent) aperture efficiency.