MeerKATHI Documentation

Paolo Serra

Feb 12, 2020

Contents

MeerKATHI is a pipeline for radio interferometry data reduction. It works on data from any radio interferometer as long as they are in "measurement set" format.

This documentation is work in progress. Please bear with us.

CHAPTER 1

Download & Install

1.1 Usage and publication policy

When using MeerKATHI/CARACal please be aware of and adhere to the MeerKATHI publication policy.

1.2 On Linux

1. Clone this repository Use https and your github credentials, then go to the pipeline folder 'meerkathi'.

```
$ git clone https://github.com/ska-sa/meerkathi.git
$ cd meerkathi
```

1. Start and activate virtual environment outside the meerkathi directory

```
$ cd ..
$ virtualenv meerkathi-venv
$ source meerkathi-venv/bin/activate
$ pip install pip wheel setuptools -U
```

1. If working from master branch it may be necessary to install bleeding edge fixes from upstream dependencies. Please install the requirements.txt requirements:

```
$ pip install -U -r <absolute path to meerkathi folder>/requirements.txt
```

1. Install meerKATHI

```
$ pip install <absolute path to meerkathi folder>"[extra_diagnostics]"
$ export PYTHONPATH='' # Ensure that you use venv Python
```

If the requirements cannot be installed on your system you may omit [extra_diagnostics]. This will disable report rendering.

1. Pull and/or build stimela images

- **Podman**[Recommended]

```
$ stimela pull -p
```

- **Singularity**[Recommended]

```
$ stimela pull --singularity --pull-folder <folder to store stimela singularity
↪images>
```

- **uDocker**[Recommended]<note: no python3 support>

```
$ stimela pull
```

- **Docker**

```
$ stimela pull -d
$ stimela build
```

1. run meerkathi

- **Podman**[Recommended] `$ meerkathi -c path_to_configuration_file --container-tech podman`
- **Singularity**[Recommended] `$ meerkathi -c path_to_configuration_file --container-tech singularity -sid <folder where singularity images are stored>`
- **uDocker**[no python3 support] `$ meerkathi -c path_to_configuration_file --container-tech udocker`
- **Docker** `$ meerkathi -c< path to configuration file>`

1.3 Troubleshooting

- **Stimela cache file** When re-building/pullng/updating stimela (any stimela call above), sometimes problems will arise with the cache file of stimela, whose standard location is

```
~/.stimela
```

If you run into unexplicable errors when installing a stimela version, including a failed update (possibly resulting in a repeating error when running CARACal), do:

```
> rm ~/.stimela/*
> stimela ...
```

before re-building. If that does not work, re-building the dependencies might help.

```
> pip install --upgrade --force-reinstall -r <absolute path to meerkathi folder>/
↪requirements.txt
> rm ~/.stimela/*
> stimela ...
```

1.4 On Mac

1. create a python environment

```
$ conda create env --name meer_venv
```

1. activate environment

```
$ source activate meer_venv
```

1. clone `meerkathi`

```
$ git clone https://github.com/ska-sa/meerkathi.git
$ cd meerkathi
```

1. Start and activate virtual environment

```
$ virtualenv meerkathi-venv
$ source meerkathi-venv/bin/activate
$ pip install pip wheel setuptools -U
```

1. If working from master branch it may be necessary to install bleeding edge fixes from upstream dependencies. Please install the requirements.txt requirements:

```
$ pip install -U -r <absolute path to meerkathi folder>/requirements.txt
```

1. Install meerKATHI

```
$ pip install <absolute path to meerkathi folder>
$ export PYTHONPATH='' # Ensure that you use venv Python
```

1. Pull and/or build stimela images

- **uDocker**[Recommended]

```
$ stimela pull
```

- **Singularity**[Recommended]

```
$ stimela pull --singularity --pull-folder <folder to store stimela singularity␣
↪images>
```

- **Docker**

```
$ stimela pull
$ stimela build
```

1. run meerkathi

- **uDocker**[Recommended] `$ meerkathi -c path_to_configuration_file --container-tech udocker`
- **Singularity**[Recommended] `$ meerkathi -c path_to_configuration_file --container-tech singularity -sid <folder where singularity images are stored>`
- **Docker** `$ meerkathi -c< path to configuration file>`

CHAPTER 2

Manual

2.1 Introduction

2.1.1 What is MeerKATHI?

MeerKATHI is a pipeline to reduce radio interferometry continuum and spectral line data in full polarisation. It works on data from any radio interferometer as long as they are in "measurement set" format.

In the simplest terms, MeerKATHI is a collection of Python/Stimela scripts. Stimela is a platform-independent radio interferometry scripting framework based on Python and Docker/Singularity. Stimela allows users to execute tasks from many different data reduction packages in Python without having to install those packages individually (e.g., CASA, MeqTrees, AOflagger, SoFiA, etc.). Using Stimela, the different software packages are available through a unified scheme. MeerKATHI consists of a sequence of Stimela scripts, which it links and runs sequentially.

Within MeerKATHI – and throughout this documentation – the individual Stimela scripts are called "workers". Each MeerKATHI worker corresponds to a specific section of the data reduction process (e.g., flagging, cross-calibration, spectral line imaging, etc.). Each worker executes several tasks from the interferometry packages included in Stimela (e.g., the cross-calibration worker can calibrate delays, bandpass, gains and flux scale).

In practice, users tell MeerKATHI what to do – and how to do it – via a YAML configuration file. The configuration file has one section for each run of a worker. By editing the configuration file users control the workers' options, deciding which tasks to run and with what settings. A detailed explanation of the configuration file syntax is given in the *Configuration file* section of this manual.

Normally, users will not have to touch anything but the configuration file. They can check what has happened through a variety of data products, including images, diagnostic plots and log files. A list of all MeerKATHI data products is available at the *Data products* section of this manual.

In the rest of this Introduction we give a brief description of each worker. A more comprehensive description is available in the *Data reduction* section of this manual, which follows the flow of a typical data reduction process. The full list of parameters available for the individual workers through the configuration file can be found at the *Workers parameters* section of this manual or following the links below.

2.1.2 Brief description of workers

The following workers are available in MeerKATHI. Typically, they are executed in the same order in which they are given below. Only the first three workers (general, get_data and observation_config) should always be executed. All other workers are optional.

general

This worker sets up the name of various input/output directories and the prefix used for the output data products (e.g., diagnostic plots, images, etc.).

get_data

This worker sets up the name of the files to be processed and whether any conversion to .MS format is necessary. It can also virtually concatenate several .MS files together.

observation_config

This worker collects basic information on the content of the .MS files to be processed (e.g., target and calibrators' name, channelisation, etc.). The worker can also extract this information automatically from the .MS metadata. Finally, it can create a primary beam image cube on a user-defined pixel- and frequency grid.

prepare_data

This worker prepares the data for calibration and imaging. For example, it can recalculate UVW coordinates, add a BITFLAG column to the input .MS files, or add spectral weights based on Tsys measurements.

flagging

This worker flags the data and returns statistics on the flags. As all other workers, it can be run multiple times within a single MeerKATHI run as explained at *Configuration file* (though this feature is not necessarily useful for many other workers). It can flag data based on, e.g., channel-, antenna- and time selection, or using automated algorithms that run on autocorrelations (to catch antennas with clear problems) or crosscorrelations.

cross_cal

This worker cross-calibrates the data. Users can calibrate delays, bandpass, gains and flux scale. The calibration can be applied to the calibrators' visibilities (for later inspection) and to the target. Numerous parameters are available for users to decide how to calibrate. Flagging based on closure errors is available in this worker.

polcal

TBD

inspect_data

This worker produces diagnostic plots based on the calibrated calibrators' visibilities.

split_target

This worker creates new .MS files which contain the targets' calibrated visibilities only. Time and frequency averaging is available, as well as phase rotation to a new phase centre. Crosscalibration can be applied on the fly while splitting.

masking

This worker creates an a-priori clean mask based on NVSS or SUMSS catalogues, to be used during the continuum imaging/self-calibration loop. It can also merge the resulting mask with a mask based on an existing image.

self_cal

This worker performs continuum imaging and standard (i.e., direction-independent) self-calibration. Automated convergence of the calibration procedure is optionally available. This worker can also interpolate and transfer sky model and calibration tables to another .MS (e.g., from a coarse- to a fine-channel .MS file).

image_line

This worker creates spectral-line cubes and images. It can subtract the continuum via both model and UVLIN-like subtraction, Doppler correct, flag solar RFI, perform automated iterative cleaning with 3D clean masks, and, finally, run a spectral-line source finder.

2.2 Configuration file

TBD

2.3 Data products

The following data products are written by MeerKATHI.

2.3.1 log files

TBD

2.3.2 continuum images

TBD

2.3.3 spectral-line cubes

TBD

2.3.4 spectral-line moment images

TBD

2.4 Data reduction

2.4.1 Prepare pipeline and data

[relevant workers: *general*, *get_data*, *observation_config*, *prepare_data***]**

Directories and input file names

A run of MeerKATHI must always start by setting up a number of directory and file names. This is done through the *general* and *get_data* workers.

In the *general* worker users give:

- if necessary, the *data_path* directory where to find the files that should be converted to .MS format (*general: data_path*);
- the *msdir* directory where to find/write .MS files (*general: msdir*);
- the *output* directory where to write all output data products (*general: output*);
- the *input* directory where to find various input files, such as AOflagger strategies etc. (*general: input*);
- the prefix for the output data products (*general: prefix*).

If they do not exist yet, the above directories can be created by setting *general: init_pipeline* to *true*. This also copies files from the meerkathi/data/meerkat_files directory to the *input* directory set above.

In the *get_data* worker users give the name of the .MS files to be processed (*get_data: dataid*). Furthermore, the following optional steps are available:

- convert from HDF5/MVF format to .MS (*get_data: mvftoms*); this step includes the following additional conversion options:
  - create a .MS.TAR file,
  - convert only visibilities in a selected channel range,
  - discard cross-polarisation products;
- untar an existing .MS.TAR file (*get_data: untar*);
- virtually concatenate all .MS input files (*get_data: combine*); optionally users can delete an existing concatenate file and tar/untar the concatenated file.

Metadata

Finally, before starting the actual data processing users need to provide some info about the content of the input .MS files through the *observation_config* worker (e.g., target and calibrators name, channelisation, etc.). In principle, this can be done by editing the relevant parameters of this worker:

- *observation_config: target*
- *observation_config: bpcal*
- *observation_config: fcal*
- *observation_config: gcal*
- *observation_config: reference_antenna*

In fact, MeerKATHI can automatically extract most of these info from the .MS files themselves. To do so users should enable the *observation_config: obsinfo* parameter. This writes a .JSON and a .TXT file to disc. Once the .JSON file is on disc, parameters set to *auto* (or in some cases to 0) in the *observation_config* worker will be automatically read from that file.

Note that the reference antenna information is often missing from the metadata. In those cases, users should carefully select a good reference antenna for calibration and set it with the *observation_config: reference_antenna* parameter.

[missing from this page: vampirisms, primary beam]

2.4.2 Flag data

[relevant workers: *flagging***]**

The *flagging* worker can run on the input .MS files or on .MS files created by MeerKATHI at various stages of the pipeline (e.g., by the *split_target* worker). In the latter case the name of the .MS files to be flagged is based on that of the input .MS files, with a label added before the extension. Users can set the label with the *flagging: label* parameter in this worker.

The *flagging* worker allows users to flag the data in a variety of ways. Unless otherwise stated below, flagging is done with the CASA task FLAGDATA. Follow the links below for a detailed documentation of the individual flagging modes.

- Flag on autocorrelations to catch antennas with obvious problems using the custom program POLITSIYAKAT (*flagging: autoflag_autocorr_powerspectra*). Individual scans are compared to the median of all scans per field and channel; and individual antennas are compared to the median of all antennas per scan, field and channel. Both methods have their own flagging threshold, which users can tune. Users can also set which column and which fields to flag.
- Flag all autocorrelations (*flagging: flag_autocorr*).
- Flag specific portions of the beginning and/or end of each scan (*flagging: quack_flagging*). As in the CASA task FLAGDATA, users can set the time interval that should be flagged and the quackmode.
- Flag shadowed antennas (*flagging: flag_shadow*). Users can tune the amount of shadowing allowed before flagging an antenna. For observations obtained with a MeerKAT subarray it is possible to include offline antennas in the shadowing calculation.
- Flag selected channel ranges (*flagging: flag_spw*).
- Flag selected time ranges (*flagging: flag_time*).
- Flag selected antennas (*flagging: flag_antennas*). Within this task, users can limit the flagging of selected antennas to a selected timerange.
- Flag selected scans (*flagging: flag_scan*).
- Flag according to a static mask of bad frequency ranges using the custom program RFIMASKER (*flagging: static_mask*). The mask file should be located in the *input* directory set by *general: input*. Users can decide to limit the flagging to a selected UV range. This could be useful to flag short baselines only.
- Flag with AOFlagger (*flagging: autoflag_rfi*). The AOFlagger strategy file should be located in the *input* directory set by *general: input*. MeerKATHI comes with a number of strategy files, which are located in the meerkathi/data/meerkat_files directory and are copied to the *input* directory by the *general* worker. However, users can copy their own strategy file to the same *input* directory and use it within MeerKATHI. Additional parameters allow users to limit the execution of AOFlagger to selected columns, fields or frequency bands of the .MS files. AOFlagger is described by Offringa et al. (2012), A&A, 539, A95.

Finally, a summary of the flags can be obtained with *flagging: flagging_summary*. The summary is available at the relevant log file (see *Data products*).

2.4.3 Cross-calibration

[relevant workers: *cross_cal*, *inspect_data*]

Cross-calibration runs largely on CASA tasks. Using these tasks, MeerKATHI allows users to solve for delays, bandpass, gains and flux scale in several different ways.

Just as an example, it is possible to solve for: 1) time-independent antenna delays and normalised bandpass based on the observation of a bandpass calibrator; 2) time-dependent antenna flux scale based on the observation of a flux calibrator; 3) time-dependent antenna gains based on the observation of a secondary calibrator; 4) time-dependent antenna flux scale at fine time resolution obtained by scaling the gains from step 3 above to the gains from step 2 above.

Variations on the above scheme are possible by tuning the parameters of the various steps. However, be aware that MeerKATHI does not check that the selected combination of settings makes physical sense.

Preliminary steps

Before starting users can take the following optional steps.

- Set up a label which will be appended to all cross-calibration products (i.e., tables and plots; *cross_cal: label*)
- Reinitialise the .MS for calibration with the CASA task CLEARCAL (cross_cal: clear_cal). Given the users' field selection for this task (with the *field* parameter) MeerKATHI sets the MODEL_DATA column to unity in total intensity and zero in polarization, sets the CORRECTED_DATA column equal to the DATA column, and optionally adds a MODEL_DATA column if not already present. Selected fields can be specified with the field number or name as it appears in the metadata, or with the field code within MeerKATHI (typically "fcal", "bpcal"; field codes are set by the *observation_config* worker). Multiple fields can be selected.
- Select the UV range that should be used to calibrate the data (*cross_cal: uvrange*). This can be useful, for example, to exclude short baselines from the calibration.
- Fill the MODEL_DATA column based on a calibrator model with the CASA task SETJY (*cross_cal: set_model*). Various models are available and can be chosen using a number of additional parameters. For PKS 1934-638, users can choose a sky model which includes confusing sources and their spectral shape as seen through a frequency-dependent MeerKAT primary beam. This is recommended for MeerKAT observations. Alternatively, users can choose a point source model. When available, SARAO models are adopted. Alternatively, MeerKATHI uses the NRAO models.

Delay calibration

Antenna-based delay calibration is performed with the CASA task GAINCAL. It results in the creation of a delay calibration table, which subsequent calibration steps can apply the on the fly (e.g., bandpass, gain and flux calibration). In order to do this set *cross_cal: otfdelay* to *true*.

In GAINCAL, the delay calibration is performed using only baselines with the reference antenna. Therefore, the choice of reference antennas is very important. For example, reference antennas involved in short baselines should probably be avoided. Also, care should be taken that the reference antenna is not involved in heavily flagged baselines, as this could result in loss of a much larger fraction of the array due to missing delay calibration.

The delay calibration options are set at cross_cal: delay_cal. Users can set the following parameters:

- Field to use (*field*). This can be the field number or name as it appears in the metadata, or the field code within MeerKATHI ("fcal", "bpcal", "gcal"; field codes are set by the *observation_config* worker). Multiple fields can be used for the delay calibration.
- Solution time interval (*solint*). Set this to 'inf' for time-independent delays. Note that for fully time-independent delays you may need to set the *combine* option below to an appropriate value.

- Whether and how to combine the data (*combine*). If this is an empty string then data are not combined and at least one delay solution is calculated per observation, field, scan, spw (even for *solint="inf"*).
- Minimum S/N ratio (*minsnr*). No delay is calculated for solution intervals with a S/N ratio below this value.

Once a delay calibration has been obtaiend users can flag the data based on the delay solutions (*flag*). This is done with the CASA task FLAGDATA. For *mode="clip"* all visibilities with a delay outside the *clipminmax* range are flagged. Other delay flagging modes are also available.

Finally, the solutions can be plotted (*plot*). Normally, all plotting options can be kept to their default values.

Bandpass calibration

Antenna-based bandpass calibration is performed with the CASA tasks BANDPASS and, optionally, GAINCAL. Its options are set at cross_cal: bp_cal. The following bandpass calibration options are similar to those for the delay calibration (see above):

- Field to use (*field*). This can be the field number or name as it appears in the metadata, or the field code within MeerKATHI (typically "bpcal"; field codes are set by the *observation_config* worker). Multiple fields can be used for the bandpass calibration.
- Solution time interval (*solint*). Set this to 'inf' for a time-independent bandpass. Note that for a fully time-independent bandpass you may need to set the *combine* option below to an appropriate value.
- Whether and how to combine the data (*combine*). If this is an empty string then data are not combined and at least one bandpass solution is calculated per observation, field, scan (even for *solint="inf"*).
- Minimum S/N ratio (*minsnr*). No bandpass is calculated for solution intervals with a S/N ratio below this value.

Bandpass calibration has the following additional options:

- Minimum number of baselines (*minnrbl*). Given a solution interval, no bandpass is calculated for antennas with less baselines than this minimum number.
- Whether to set the reference antenna to the value given in the *observation_config* worker (*set_refant*). If *false* then CASA BANDPASS will decide which reference antenna to use.
- Whether to normalise the bandpass to have average amplitude and phase of 1 and 0, repsectively (*solnorm*).

MeerKATHI can calibrate the bandpass with the above options in a single run of the CASA task BANDPASS. However, a single run of BANDPASS may not be ideal when calculating a time-independent bandpass from calibrator scans spread over a long time interval. This is because the vector time-average of the calibrator's raw visibilities, which is calculated before solving for the time-independent bandpass, may be corrupted by the variation of the antenna gains with time. In this case it is better to correct for such gain variations *before* time-averaging the calibrator's visibilities and calculating the time-independent bandpass. MeerKATHI allows users to do this through the parameter *remove_ph_time_var*. Setting this parameter to *true* results in the following three steps (instead of a single run of BANDPASS):

- A first run of BANDPASS with the solution interval set by *solint* and with *combine=""* (regardless of what users set *combine* to be). This results in a preliminary bandpass solution per observation, field, scan even for *solint="inf"*.
- A run of the the CASA task GAINCAL applying the above bandpass on the fly and solving for time-dependent antenna gains. The use of the preliminary bandpass ensures a good quality of the time-dependent gain calibration.
- A second run of BANDPASS with the solution interval set by *solint* and combining the data as requested by the user with *combine*. The above gain calibration is applied on the fly, resulting in a more accurate time-independent bandpass.

Finally, the solutions can be plotted (*plot*). Normally, all plotting options can be kept to their default values.

Flux scale calibration

Antenna-based flux scale calibration is performed with the CASA task GAINCAL. Its options are set at cross_cal: gain_cal_flux.

The flux scale calibration options are similar to those for the bandpass calibration (see above):

- Field to use (*field*). This can be the field number or name as it appears in the metadata, or the field code within MeerKATHI (typically "fcal"; field codes are set by the *observation_config* worker). Multiple fields can be used for the flux scale calibration.
- Solution time interval (*solint*). Set this to 'inf' for a time-independent calibration. Note that for a fully time-independent calibration you may need to set the *combine* option below to an appropriate value.
- Whether and how to combine the data (*combine*). If this is an empty string then data are not combined and at least one flux scale solution is calculated per observation, field, scan, spw (even for *solint="inf"*).
- Minimum S/N ratio (*minsnr*). No flux scale calibration is calculated for solution intervals with a S/N ratio below this value.
- Minimum number of baselines (*minnrbl*). Given a solution interval, no flux scale calibration is calculated for antennas with less baselines than this minimum number.
- Whether to set the reference antenna to the value given in the *observation_config* worker (*set_refant*). If *false* then CASA GAINCAL will decide which reference antenna to use.

Finally, the solutions can be plotted (*plot*). Normally, all plotting options can be kept to their default values.

Gain calibration

Antenna-based gain calibration is performed with the CASA task GAINCAL. Its options are set at cross_cal: gain_cal_gain.

The gain scale calibration options are identical to those for the flux calibration (see above):

- Field to use (*field*). This can be the field number or name as it appears in the metadata, or the field code within MeerKATHI (typically "gcal"; field codes are set by the *observation_config* worker). Multiple fields can be used for the gain calibration.
- Solution time interval (*solint*). Set this to 'inf' for a time-independent calibration. Note that for a fully time-independent calibration you may need to set the *combine* option below to an appropriate value.
- Whether and how to combine the data (*combine*). If this is an empty string then data are not combined and at least one gain solution is calculated per observation, field, scan, spw (even for *solint="inf"*). This is probably the mode of interest in most cases.
- Minimum S/N ratio (*minsnr*). No gain calibration is calculated for solution intervals with a S/N ratio below this value.
- Minimum number of baselines (*minnrbl*). Given a solution interval, no gain calibration is calculated for antennas with less baselines than this minimum number.
- Whether to set the reference antenna to the value given in the *observation_config* worker (*set_refant*). If *false* then CASA GAINCAL will decide which reference antenna to use.

Finally, the gain solutions can be plotted (*plot*). Normally, all plotting options can be kept to their default values.

Flux scale bootstrapping

Typically, the calibrator used for the flux scale calibration above is observed at low cadence. This results in a coarse time resolution (if any) for the flux calibration. Higher resolution mapping of the variation of the gain amplitude with

time might be provided by the gain calibration step above, which is based on the typically more frequent observation of a gain calibrator. These gain amplitudes do not give a reliable, absolute flux scale, but can be scaled to the gain amplitudes obtained from the flux calibrator. Within MeerKATHI, this step is performed with the CASA task FLUXSCALE. Its options are set at cross_cal: transfer_fluxscale.

The only availabel options are the field(s) from which the flux scale was derived (*reference*) and those whose gains should be scaled (*transfer*). As for all other steps above, fields can be specified with the field number or name as it appears in the metadata, or the field code within MeerKATHI. Typically this will be *reference="fcal"* and *transfer="gcal"*.

The CASA FLUXSCALE algorithm works initially on each antenna and each polarisation product separately. It calculates the ratio R (i , x) between the time-averaged gain amplitude derived from the flux calibrator and the time-averaged gain amplitude derived from the gain calibrator for antenna i and polarisation product x. This results in N (ant) x N (pol) values of the ratio R (i , x). The algorithm then takes the median of all such values, F = MEDIAN[R (i , x)]. Finally, all values of the gain amplitude derived from the gain calibrator for all antennas and polarisation products are multiplied by F. This means that the bootstrapping is done globally for the array as a whole, and not individually for each antenna and each polarisation product.

The final flux scale solutions can be plotted (*plot*). Normally, all plotting options can be kept to their default values.

Apply the cross-calibration and diagnostic plots

MeerKATHI can apply the cross calibration tables to all calibrators (useful for diagnostics) and to the science target. In doing so it will use the following interpolation rules:

- Delay calibration: applied to the fields bpcal, gcal, target with nearest, linear, linear interpolation, respectively.
- Bandpass calibration: applied to the fields bpcal, gcal, target with nearest, linear, linear interpolation, respectively.
- Gain calibration before bootstrapping the flux scale: applied to the fields bpcal, gcal, target with linear, linear, linear interpolation, respectively.
- Gain calibration after bootstrapping the flux scale: applied to the fields bpcal, gcal, target with linear, nearest, linear interpolation, repsectively.

The calibrated caibrators' visibilities can be inspected to check whether the calibration is correct. This is done by the *inspect_data* worker. A number of .PNG plots are produced, such as phase-vs-amplitude and real-vs-imaginary.

[missing from this page: flag on closure errors and flag statistics]

2.4.4 Continuum imaging and self-calibration

[relevant workers: *split_target*, *flagging*, *self_cal***]**

Split, average and flag target visibilities

Following cross-calibration MeerKATHI creates a new .MS file which contains the cross-calibrated target visibilities only. This is done by the *split_target* worker. In case the cross-calibration tables have not been applied to the target by the *cross_cal* worker, *split_target* can do so on the fly while splitting using the CASA task MSTRANSFORM.

Optionally, the *split_target* worker can average in time and/or frequency while splitting. Depending on the science goals, it might be useful to run this worker more than once. E.g., the first time to create a frequency-averaged dataset for continuum imaging and self-calibration, and the second time to create a narrow-band dataset for spectral-line work. The possibility of running this worker multiple times within a single MeerKATHI run allows users to design the best strategy for their project.

Before self-calibrating it might also be good to flag the target's visibilities. (Typically the target is not flagged before applying the cross-calibration.) This can be done with the *flagging* worker (which was probably already run on the calibrators' visibilities before cross-calibration) setting fields to target in *flagging: autoflag_rfi*.

Image the continuum and self-calibrate

Having split, optionally averaged and flagged the target, it is now possible to iteratively image the radio continuum emission and self-calibrate the visibilities. The resulting gain tables and continuum model can also be transferred to another .MS file (particularly useful for spectral line work). All this can be done with the *self_cal* worker.

Several parameters allow users to set up both the imaging and self-calibration according to their needs. Imaging is done with WSclean, and the parameters of this imaging software are available in the *self_cal* worker. Calibration is done with either Cubical or MeqTrees, and also in this case the *self_cal* worker includes the parameters available in those packages.

Additional parameters allow users to decide how many calibration iterations to perform through the parameter *self_cal: cal_niter*. For a value N, the code will create N+1 images following the sequence image1, selfcal1, image2, selfcal2, ... imageN, selfcalN, imageN+1.

Optionally, users can enable *self_cal: aimfast*, which at each new iteration compares the new continuum image with the previous one and decides whether the image has improved significantly. In case it has not, no further iterations are performed. In this case therefore *self_cal: cal_niter* is the maximum number of iterations.

While imaging, WSclean auto-mask and auto-threshold can be used, but it is also possible to use a clean mask made by SoFiA from the previous continuum image. This functionality is controlled through *self_cal: sofia_mask*.

[missing a description of additional functionalities]

Gain and model transfer

If the self-cal loop was executed on a frequency-averaged .MS file, it might be necessary to transfer the resulting gains and continuum model back to the original .MS file. This is done with *self_cal: transfer_apply_gains* (using Cubical) and *self_cal: transfer_model* (using Crystalball), respectively. The latter allows users to limit the model transfer to the N brightest sources, to sources in a region, or to point sources only.

2.4.5 Spectral line imaging

[relevant workers: *image_line*]

Spectral line imaging runs on a combination of custom software, CASA, WSclean, SunBlocker and SoFiA in order to subtract the continuum, Doppler correct, flag solar RFI, create cleaned spectral line cubes and moment images. It can run on the input .MS files or on .MS files created by MeerKATHI at various stages of the pipeline (e.g., by the *split_target* worker). In the latter case the name of the .MS files to be imaged is based on that of the input .MS files, with a label added before the extension. Users can set the label with the *image_line: label* parameter in this worker.

The input .MS files may contain several targets. In this case, MeerKATHI makes one HI cube per target, using all available visibilities for that target from all input .MS files. This worker does not mosaic line cubes made for different targets.

Continuum subtraction

Continuum subtraction might be necessary before imaging the spectral line of interest. MeerKATHI can do this using two standard methods, which can be run sequentially within a single MeerKATHI run: i) subtraction of the continuum model visibilities from the field visibilities (*image_line: subtractmodelcol*); and ii) fitting and subtracting

polynomials from the individual real and imaginary visibility spectra (parameter *uvlin* in *image_line: mstransform*). A third standard method currently NOT implemented in MeerKATHI consists of fitting and subtracting polynomials from individual image spectra in the data cube. This may be implemented in the future.

In practice, the first method consists of subtracting the MODEL_DATA column from the CORRECTED_DATA column of the .MS files. MeerKATHI writes the resulting visibilities in the CORRECTED_DATA column itself.

Users should therefore be aware that the CORRECTED_DATA column gets overwritten.

The MODEL_DATA column contains the continuum model to be subtracted. Within MeerKATHI, the MODEL_DATA column should have been filled in with the continuum model resulting from the continuum imaging and self-calibration done by the *self_cal* worker.

When running the second continuum subtraction method, which uses the CASA task MSTRANSFORM, users can select the order of the fit, the channels that should be included in the fit, and the column that should be considered. This method writes a new file with an 'mst' suffix appended to the file name. Subsequent steps of this worker can be instructed to run on the file written by MSTANSFORM.

Doppler correction

Doppler correction is performed with the CASA task MSTRANSFORM (parameter *doppler* in *image_line: mstransform*) in the same run of this task used to perform continuum subtraction (see above). Users can select the telescope (choosing from a list of available names, see parameter *telescope* in *image_line: mstransform*), as well the regridding mode (channel, frequency or velocity), velocity type and output frame. Users can also set the frequency (or velocity, ...) grid they want to Doppler correct to, but they can also let MeerKATHI fin the optimal one given the input files. In the latter case, visibilities will be regridded to the widest Doppler-corrected spectral interval common to all input .MS files, at the worse Doppler-corrected spectral resolution of them all.

Solar RFI flagging

MeerKATHI flags solar RFI using SunBlocker (*image_line: sunblocker*). The main idea of SunBlocker is that, because solar RFI is broadband, averaging visibilities in frequency should enhance its detectability. However, the phase of solar RFI changes rapidly with frequency, leading to vectorial averages with very low amplitude. In order to enhance the detectability of the solar RFI SunBlocker performs a scalar average. It does so in uv cells of the visibility plane (i.e., on gridded visibilities). Once that is done, UV cells with anomalously high (scalar) average ampliude are flagged. This method has been shown to work well on continuum-subtracted data. Users have control over some of the SunBlocker settings, such as flagging threshold and gridding. It is also possible to run this task on day-time data only.

Imaging

Spectral line imaging is done with WSCLEAN. This software produces a set of individual .FITS images per channel (i.e., dirty image, psf, clean model, restored image), and when that is done MeerKATHI stacks them all together in .FITS image cubes. Cleaning is done iteratively by making a first cube using WSClean algorithms for a blind clean, then making a clean mask with SoFiA and running WSClean again with that clean mask, and so on. Users can let MeerKATHI continue iterating until the noise in the residual cube converges (up to a maximum number of iterations) or perform a fixed number of iterations ignoring noise convergence. Users also have full control of all WSCLEAN imaging parameters.

Imaging can also been done in CASA, but this has not received as much attention as WSCLEAN imaging and may be obsolete.

Several additional steps are availabel and can be run once the .FITS image cubes are ready. This includes removing the trivial Stokes axis from the cubes, convert the frequency axis from frequency to velocity, and create a primary beam cube on the same WCS grid of the image cube. At the moment the primary beam cube is calculated assuming a Gaussian primary beam with FWHM = 1.02 * lightspeed / frequency / dishdiameter.

As a final step, MeerKATHI can run SoFiA in order to make a line detection mask and the corresponding moment images. Users have control over several (but not all) SoFiA settings.

Diagnostics

As a useful diagnostic, MeerKATHI can run SHARPENER, which extracts 1D spectra at the position of bright continuum sources in the field. This is helpful to assess the quality of the continuum subtraction. It can also create one last flagging summary.

2.5 Workers parameters

2.5.1 general

General pipeline information, data IDs, prefixes for output

data_path

str, *optional*, *default* = ‘ ‘

where MeerKATHI (over-) writes HDF5 files and JSON info files downloaded by get_data below

msdir

str, *optional*, *default* = *msdir*

where MeerKATHI will write and expect to find measurement set (MS) files

input

str, *optional*, *default* = *input*

where MeerKATHI expects to find various input files (e.g., RFI flagging strategy files).

output

str, *optional*, *default* = *output*

where MeerKATHI writes output products

prefix

str, *optional*, *default* = *meerkathi*

Prefix for MeerKATHI output products

init_pipeline

bool, *optional*, *default* = *True*

Initialise pipeline by copying input files (meerkat specific; flagging strategies, beam model, etc.)

init_notebooks

list of str, *optional*, *default = std-progress-report*

Install standard radiopadre notebooks, given by list of basenames

2.5.2 get_data

Download and/or convert/unarchive data so that its in the MS format for further processing

dataid

list of str

Basename of MS. For MeerKAT data to be downloaded by MeerKATHI, this should be the data ID of the observation

mvftoms

Convert HDF5/MVF files in data_path to MS files; the latter are written to msdir; also creates a MS.TAR file. (This only works for MeerKAT HDF5 files)

enable

bool, *optional*, *default = False*

Execute this segment

tar

bool, *optional*, *default = False*

Create a tarbal of the converted MS.

channel_range

str, *optional*, *default = all*

Only exctract channels in this range (0-based, inclusive; comma seperated string)

full_poll

bool, *optional*, *default = False*

Extract all four correlations instead of only the XX,YY

untar

Unarchive from MS from a archive file.

enable

bool, *optional*, *default = False*

Execute this segment

tar_options

str, *optional*, *default = -xvf*

Options to parse to 'tar' command

combine

Virtually concatenate MSs and proceed with the combined MS

enable

bool, *optional*, *default = False*

Execute this section

reset

bool, *optional*, *default = False*

Delete concatenated MS if it exists. Else, proceed with existing MS

tar

Create a tarbal of the converted MS

enable

bool, *optional*, *default = False*

Execute this section

tar_options

str, *optional*, *default = -cvf*

Options to parse to the tar command

untar

Unarchive from MS from a archive file.

enable

bool, *optional*, *default = False*

Execute this section

tar_options

str, *optional*, *default = -xvf*

Options to parse to the tar command

2.5.3 observation_config

Setup some basic observation information

obsinfo

Get observation information

enable

bool

Execute this section

listobs

bool, *optional*, *default = True*

Run CASA listobs task to get observation information

summary_json

bool, *optional*, *default = True*

Run MSUtils summary function to get observation information as JSON file which can be used to automatically configure pipeline

vampirisms

bool, *optional*, *default = False*

Returns sun free time range

plot_elevation_tracks

bool, *optional*, *default = False*

Make Elevation vs Hour angle plots for observed fields

plotter

{"plotms", "owlcat"}, *optional*, *default = owlcat*

Application to use for making plots

target

list of str, *optional*, *default = all*

Field name of target field. Or 'all' for all the target fields.

gcal

list of str, *optional*, *default = all*

Field name of gain (amplitude/phase) calibrator field. Or set as 'all' for all the gcal fields, 'longest' to setlect the gcal field observed for the longest time, 'nearest' to select the gcal field closest to the target. Note that if multiple targets and gcals are present, then 'all' (for both) means each target will be paired with the closest gcal.

bpcal

list of str, *optional*, *default = longest*

Field name of bandpass calibrator field. Or set as 'all' for all the bpcal fields, 'longest' to setlect the bpcal field observed for the longest time, 'nearest' to select the bpcal field closest to the target.

fcal

list of str, *optional*, *default = longest*

Field name of fluxscale calibrator field. Or set as 'all' for all the fcal fields, 'longest' to setlect the fcal field observed for the longest time, 'nearest' to select the fcal field closest to the target.

xcal

list of str, *optional*, *default = longest*

Crosshand phase angle calibrator. This calibrator must be linearly polarized and have a non-zero parallactic angle coverage at the time of observation to solve for the X-Y offsets in digitizers and the absolute polarization angle of the system. Successful calibration derotates U from V.

reference_antenna

str

Reference antenna.

2.5.4 prepare_data

Prepare the data for calibration and imaging.

enable

bool

Executes the data preparation step.

fixvis

Fixes the UVW coordinates through the CASA task fixvis.

enable

bool, *optional*, *default = False*

Enable execution of fixvis.

clear_cal

bool, *optional*, *default = False*

Clears out calibrated data and resets previous predicted model

manage_flags

Manage MS flags

enable

bool, *optional*, *default = true*

enable this section

add_bitflag_column

bool, *optional*, *default = true*

Add BITFLAG and BITFLAG_ROW columns

init_legacy_flagset

bool, *optional*, *default = true*

Save all current flags in a legacy flagset if it does not exist

remove_flagsets

bool, *optional*, *default = true*

Remove all existing flagsets, except legacy flags

spectral_weights

How to initialize spectral weights

enable

bool, *optional*, *default = False*

Enable this segment

mode

{"uniform", "estimate", "delete"}, *optional*, *default = uniform*

uniform: Set all weights to unity, estimate: Estimate spectral weights from frequency-dependent SEFD/Tsys/Noise values, delete: Delete WEIGHT_SPECTRUM column if it exists. if 'estimate', then further settings are available in the 'estimate' segement of this section

estimate

Estimate spectral weights from frequency-dependent SEFD/Tsys/Noise values

stats_data

str, *optional*, *default = use_package_meerkat_spec*

File with SEFD/Tsys/Noise data. If data is from MeerKAT telescope, you can specify 'use_package_meerkat_spec' to use package data.

weight_columns

list of str, *optional*, *default = WEIGHT, WEIGHT_SPECTRUM*

column names

noise_columns

list of str, *optional*, *default = SIGMA, SIGMA_SPECTRUM*

column names for noise

write_to_ms

bool, *optional*, *default = True*

write columns to file

2.5.5 flagging

Flagging of the data.

enable

bool

Execute flagging of the data.

label

str, *optional*, *default = ‘ ‘*

The label is added to the input .MS file name to define the name of the .MS file that should be flagged, <input>-<label>.ms. Default is an empty string, i.e., the original .MS is flagged.

autoflag_autocorr_powerspectra

Flags antennas based on drifts in the scan average of the auto correlation spectra per field. This doesn't strictly require any calibration. It is also not field structure dependent, since it is just based on the DC of the field. Compares scan to median power of scans per field per channel. Also compares antenna to median of the array per scan per field per channel. This should catch any antenna with severe temperature problems.

enable

bool, *optional*, *default = False*

Enables flagging of antennas based on drifts in the scan average of the auto correlation spectra per field.

scan_to_scan_threshold

int, *optional*, *default = 3*

Threshold for flagging in sigma above the rest of the scans per field per channel.

antenna_to_group_threshold

int, *optional*, *default = 5*

Threshold for flagging in sigma above array median power spectra per scan per field per channel.

column

str, *optional*, *default = DATA*

Data column to flag.

fields

str, *optional*, *default = auto*

Fields to flag. Given as ‘auto’ or comma-seperated keys (keys in gcal, bpcal, target).

calibrator_fields

str, *optional*, *default = auto*

Calibrator fields. Given as ‘auto’ or comma-seperated keys (keys in gcal, bpcal).

threads

int, *optional*, *default = 8*

Number of threads to use.

flag_autocorr

Flag autocorrelations. Through CASA flagdata task.

enable

bool, *optional*, *default = True*

Enables flagging of autocorrelations.

quack_flagging

Do quack flagging, i.e. flag the begining and/or end chunks of each scan. Again, through FLAGDATA.

enable

bool, *optional*, *default = False*

Enable quack flagging.

quackinterval

float, *optional*, *default = 8.*

Time interval (in seconds) to flag.

quackmode

{"beg", "endb", "end", "tail"}, *optional*, *default = beg*

Quack flagging mode. Either 'beg', which flags scan begining, 'endb', which flags end of the scan, 'end', which flags everything but the first specified seconds of the scan and 'tail' which flags all but the last specified seconds of the scan.

flag_elevation

Flag antennas with pointing elevation outisde the selected range through CASA FLAGDATA.

enable

bool, *optional*, *default = False*

Enable flagging based on pointing elevation.

low

float, *optional*, *default = 0*

Lower elevation limit. Antennas pointing at elevation below this value are flagged.

high

float, *optional*, *default = 90*

Upper elevation limit. Antennas pointing at elevation above this value are flagged.

flag_shadow

Flag shadowed antennas through the CASA task FLAGDATA.

enable

bool, *optional*, *default = False*

Enables flagging of shadowed antennas.

tolerance

float, *optional*, *default = 0.*

Amounts of shadow allowed (in metres). Default is 0. A positive number allows antennas to overlap in projection. A negative number forces antennas apart in projection.

include_full_mk64

bool, *optional*, *default = False*

Consider all MeerKAT-64 antennas in the shadowing calculation even if only a subarray is used. Default is False.

flag_spw

Flag spectral windows/channels. Of course, through FLAGDATA.

enable

bool, *optional*, *default = False*

Enable flagging spectral windows/ channels.

channels

str, *optional*, *default = *:856~880MHz , *:1658~1800MHz, *:1419.8~1421.3MHz*

Channels to flag. Given as "spectral window index:start channel ~ end channel" e.g. "*:856~880MHz". End channel not inclusive.

ensure_valid_selection

bool, *optional*, *default = False*

Check whether the channel selection returns any data. If it does not FLAGDATA is not executed preventing the pipeline from crashing. This check only works with the following spw formats (multiple, comma-separated selections allowed), "*:firstchan~lastchan"; "firstspw~lastspw:firstchan~lastchan"; "spw:firstchan~lastchan"; "firstchan~lastchan". Channels are assumed to be in frequency (Hz, kHz, MHz, GHz allowed; if no units are given it assumes Hz).

flag_time

Flag timerange in the data using CASA FLAGDATA task.

enable

bool, *optional*, *default = False*

Enabla flagging timeranges.

timerange

str, *optional*, *default = ' '*

Timerange to flag. Required in the format 'YYYY/MM/DD/HH:MM:SS-YYYY/MM/DD/HH:MM:SS'.

ensure_valid_selection

bool, *optional*, *default = False*

Check whether the timerange is in the ms being considered. This stops the pipeline from crashing when multiple dataset are being processed.

flag_antennas

Flag bad antennas. Or just the ones you have sworn a vendetta against.

enable

bool, *optional*, *default = False*

Enables flagging of bad antennas.

antennas

str, *optional*, *default = 0*

Antennas to flag. Follows the CASA Flagdata syntax.

timerange

str, *optional*, *default = ‘ ‘*

Timerange to flag. Required in the format ‘YYYY/MM/DD/HH:MM:SS-YYYY/MM/DD/HH:MM:SS’.

ensure_valid_selection

bool, *optional*, *default = False*

Check whether the timerange is in the ms being considered. This stops the pipeline from crashing when multiple dataset are being processed.

flag_scan

Flag bad scans. Uses CASA Flagdata task.

enable

bool, *optional*, *default = False*

Enables flagging of bad scans.

scans

str, *optional*, *default = 0*

Scans to flag. CASA flagdata syntax.

static_mask

Apply static mask to flag out known RFI, Meerkat specific.

enable

bool, *optional*, *default = False*

Enables the application of static mask on the data.

mask

str, *optional*, *default = labelled_rfimask.pickle.npy*

The mask to apply.

uvrange

str, *optional*, *default = ‘ ‘*

UV range to select (CASA style range, e.g. lower~upper) for flagging. Leave blank for entire array.

autoflag_rfi

Flag RFI using AOFlagger software.

enable

bool, *optional*, *default = True*

Enable RFI flagging with AOFlagger or tricolour (not active yet)

flagger

{“aoflagger”, “tricolour”, “tfcrop”}, *optional*, *default = aoflagger*

Choose flagger for automatic flagging

strategy

str, *optional*, *default = firstpass_QUV.rfis*

The AOFlagger strategy file to use.

column

str, *optional*, *default = DATA*

Specify column to flag

fields

str, *optional*, *default = auto*

comma separated list of (zero-indexed) field ids to process

calibrator_fields

str, *optional*, *default = auto*

comma separated list of (zero-indexed) field ids to process

bands

str, *optional*, *default = auto*

comma separated list of (zero-indexed) band ids to process

window_backend

{“numpy”, “zarr-disk”}, *optional*, *default = numpy*

Visibility and flag data is re-ordered from a MS row ordering into time-frequency windows ordered by baseline.

tricolour_calibrator_strat

str, *optional*, *default = mk_rfi_flagging_calibrator_fields_firstpass.yaml*

usewindowstats

{“none”, “sum”, “std”, “both”}, *optional*, *default = std*

Calculate additional flags using sliding window statistics

combinescans

bool, *optional*, *default = False*

Accumulate data across scans depending on the value of ntime

flagdimension

{“freq”, “time”, “freqtime”, “timefreq”}, *optional*, *default = freqtime*

Dimensions along which to calculate fits (freq/time/freqtime/timefreq)

timecutoff

float, *optional*, *default = 4.0*

Flagging thresholds in units of deviation from the fit

freqcutoff

float, *optional*, *default = 3.0*

Flagging thresholds in units of deviation from the fit

correlation

str, *optional*, *default = ‘ ‘*

Correlation

rfinder

A tool to investigate the presence of RFI

enable

bool, *optional*, *default = False*

Enable invsetigation of rfi with rfinder

telescope

str, *optional*, *default = MeerKAT*

Name of telescope

field

str, *optional*, *default = target*

Field to get flag stats. Given as a key (key in [gcal, bpcal, target]).

polarization

{“xx”, “XX”, “yy”, “YY”, “xy”, “XY”, “yx”, “YX”, “q”, “Q”}, *optional*, *default = q*

Select polarisation e.g. xx, yy, xy, yx, q (also in CAPS)

spw_enable

bool, *optional*, *default = True*

Enable spw for rebinning

spw_width

int, *optional*, *default = 10*

Channel width of rebinned output table (MHz)

time_enable

bool, *optional*, *default = True*

Enable time chunking

time_step

int, *optional*, *default = 5*

Time chunks in minutes

movies_in_report

bool, *optional*, *default = True*

Generate movies in a repo

flagging_summary

Write flagging summary at the end of the pre-calibration flagging. Uses CASA FLAGDATA in "summary" mode.

enable

bool, *optional*, *default = True*

Enables the writing of flagging summary.

2.5.6 cross_cal

Carry out Cross calibration of the data (delay, bandpass and gain calibration)

enable

bool

Execute this segment.

otfdelay

bool, *optional*, *default = True*

Set whether to apply the delay calibration on the fly when solving for other calibration terms.

uvrange

str, *optional*, *default = >50*

Set the U-V range for data selection, e.g. '>50'.

label

str, *optional*, *default = 1gc1*

Label for output files.

casa_version

str, *optional*, *default = 47*

Casa version to carry out cross-calibration. '47' means use CASA 4.7, which is recommended, unless you enjoy your data extra flag-gy. Leave empty to use the latest CASA.

set_model

Essentially setjy task from CASA.

enable

bool, *optional*, *default = True*

Execute the setjy task.

meerkathi_model

bool, *optional*, *default = False*

Force disable built-in models in MeerKATHI (NOT RECOMMENDED!)

no_verify

bool, *optional*, *default = False*

Enables setting standard manually.

field

str, *optional*, *default = fcal*

Set the field to carry out setjy on. Specify either the field number, name or even as 'fcal' corresponding to field specification in observation config.

threads

int, *optional*, *default = 8*

Set the number of threads to use when predicting local sky model using MeqTrees.

primary_cal

Calibrating on the bandpass calibrator field

enable

bool, *optional*, *default = True*

Execute this section

reuse_existing_gains

bool, *optional*, *default = False*

Reuse gain tables if they exist

order

str, *optional*, *default = KGB*

Order in which to solve for gains for this field. E.g, if order is set to 'KGB', the we solve for delays, then the phase and amplitude, and finally the bandpass. The full options are: K-delay calibration; G-amplitude and phase calibration; B-bandpass calibration; A-automatic flagging (existing gains will be applied first).

solnorm

bool, *optional*, *default = False*

Normalise average solution amplitude to 1.0

combine

list of str, *optional*, *default = '', '', scan*

Parameter to combine different data axis for solving. Options are ['','obs', 'scan', 'spw', 'field', 'obs,scan', 'scan,ob']

solint

list of str, *optional*, *default = 120s, 120s, inf*

Solution interval for delay-correction calibration.

calmode

list of str, *optional*, *default = a, ap, ap*

Type of solution

B_fillgaps

int, *optional*, *default = 70*

Fill flagged solution channels by interpolation

plotgains

bool, *optional*, *default = True*

Plot gains

flag

Apply existing gains and flag corrected data

column

{"corrected", "residual"}, *optional*, *default = corrected*

Data column to flag on

usewindowstats

{"none", "sum", "std", "both"}, *optional*, *default = std*

Calculate additional flags using sliding window statistics

combinescans

bool, *optional*, *default = False*

Accumulate data across scans depending on the value of ntime

flagdimension

{“freq”, “time”, “freqtime”, “timefreq”}, *optional*, *default = freqtime*

Dimensions along which to calculate fits (freq/time/freqtime/timefreq)

timecutoff

float, *optional*, *default = 4.0*

Flagging thresholds in units of deviation from the fit

freqcutoff

float, *optional*, *default = 3.0*

Flagging thresholds in units of deviation from the fit

correlation

str, *optional*, *default = ‘ ‘*

Correlation

secondary_cal

Calibrating on the amplitude/phase calibrator field

enable

bool, *optional*, *default = True*

Execute this section

reuse_existing_gains

bool, *optional*, *default = False*

Reuse gain tables if they exist

order

str, *optional*, *default = KG*

Order in which to solve for gains for this field. E.g, if order is set to ‘KGB’, the we solve for delays, then the phase and amplitude, and finally the bandpass. The full options are: K-delay calibration; G-amplitude and phase calibration; B-bandpass calibration; A-automatic flagging (existing gains will be applied first); I-Do a self-callibration

solnorm

bool, *optional*, *default = False*

Normalise average solution amplitude to 1.0

combine

list of str, *optional*, *default = ‘’, scan*

Parameter to combine different data axis for solving. Options are [‘’,’obs’, ‘scan’, ‘spw’, ‘field’, ‘obs,scan’, ‘scan,ob’]

solint

list of str, *optional*, *default = 120s, 120s*

Solution interval for delay-correction calibration.

calmode

list of str, *optional*, *default = a, ap, ap*

Type of solution

apply

str, *optional*, *default = B*

Gains to apply from calibration of bandpass field

plotgains

bool, *optional*, *default = True*

Plot gains

flag

Apply existing gains and flag corrected data

column

{"corrected", "residual"}, *optional*, *default = corrected*

Data column to flag on

usewindowstats

{"none", "sum", "std", "both"}, *optional*, *default = std*

Calculate additional flags using sliding window statistics

combinescans

bool, *optional*, *default = False*

Accumulate data across scans depending on the value of ntime

flagdimension

{"freq", "time", "freqtime", "timefreq"}, *optional*, *default = freqtime*

Dimensions along which to calculate fits (freq/time/freqtime/timefreq)

timecutoff

float, *optional*, *default = 4.0*

Flagging thresholds in units of deviation from the fit

freqcutoff

float, *optional*, *default = 3.0*

Flagging thresholds in units of deviation from the fit

correlation

str, *optional*, *default = ' '*

Correlation

apply_cal

Apply calibration

enable

bool, *optional*, *default = True*

Execute this section

applyto

list of str, *optional*, *default = bpcal, gcal, target*

Fields to apply calibration to

calmode

{"=", "calflag", "calflagstrict", "trial", "flagonly", "flagonlystrict"}, *optional*, *default = calflag*

Calibration mode, the default being "calflag" - calibrates and applies flags from solutions. See CASA documentation for info on other modes.

flagging_summary

Prints out the buther's bill, i.e. data flagging summary at the end of cross calibration process.

enable

bool, *optional*, *default = True*

Execute printing flagging summary.

2.5.7 polcal

Carry out crosshand calibration of the data (X and D) on boresight, after parallel hand calibration has been performed

enable

bool

Execute this segment.

label

str, *optional*, *default = pol*

Label for this worker and its associated datasets

solve_uvdist

str, *optional*, *default = 100~100000000m*

UV cutoff for solver. This should be large enough to wash out RFI

preaverage_time

str, *optional*, *default = 30s*

Preaveraging time interval (like 30s) used in solving for X, D and plots

preaverage_freq

int, *optional*, *default = 4*

Preaveraging channel interval (like 4) used in solving for X, D and plots

timesol_solfreqsel

str, *optional*, *default = ‘ ‘*

Subband selection to be used whenever solving for the DC component of either X or D. This could be picked to avoid RFI

timesol_soltime

str, *optional*, *default = inf*

Time solution interval to be used whenever solving for the DC component of either X or D

freqsol_soltime

str, *optional*, *default = inf*

Time solution interval to be used when solving for per channel solutions for either X or D. This ought to be as long as possible

feed_angle_rotation

float, *optional*, *default = -90.*

Apply feed angle rotation matrix to all stations. MeerKAT X and Y feeds are flipped, so apply a -90 offset here

do_dump_precalibration_leakage_reports

bool, *optional*, *default = False*

Write out report estimating residual quadrature leakages prior to crosshand calibration

set_model

Predict model for calibrators into preaveraged dataset

enable

bool, *optional*, *default = True*

Execute this segment.

meerkathi_model

bool, *optional*, *default = True*

Use model, if available full-sky model, built into southern calibrators database, instead of any NRAO model

threads

int, *optional*, *default = 8*

When predicting full sky model (lsm) use these many threads in MeqServer

do_phaseup_crosshand_calibrator

bool, *optional*, *default = True*

Phaseup diagonal of calibrator that is used to calibrate crosshand phase prior to solving for slope and offset in crosshand phase

do_solve_crosshand_slope

bool, *optional*, *default = True*

Solve for a frequency slope (X-Y digitizer delay) on the crosshand. This requires SNR on the crosshands.

do_solve_crosshand_phase

bool, *optional*, *default = True*

Solve for an absolute phase on the crosshand, necessary for polarization angle measurements and to derotate U from V. This requires SNR on the crosshands.

do_solve_leakages

bool, *optional*, *default = True*

Solve for leakages from I into U and V. If your calibrator polarization model for your flux scale reference is accurate this shouldalways be enabled.

do_apply_XD

bool, *optional*, *default = True*

Apply off-diagonal corrections

do_dump_postcalibration_leakage_reports

bool, *optional*, *default = True*

Write out report estimating residual quadrature leakages post crosshand calibration

flagging_summary_crosshand_cal

Summarize data flagged at the end of crosshand calibration

enable

bool, *optional*, *default = True*

Execute this segment.

2.5.8 inspect_data

Dignostic plots of the first-pass cross-calibrated data.

enable

bool

Executes dignostic plotting of the first-pass cross-calibrated data.

label

str, *optional*, *default = ‘ ‘*

Label for output products (plots etc.) for this step.

plotter

{“plotms”, “shadems”}, *optional*, *default = plotms*

Application to use for making plots

correlation

str, *optional*, *default = XX, YY*

Label specifying the correlations.

uvrange

str, *optional*, *default = ‘ ‘*

Set the U-V range for data selection, e.g. ‘>50’.

fields

list of str, *optional*, *default = bpcal, gcal*

Fields to plot. Specify by field id, index or keys like, gcal, bpcal.

real_imag

Plot real vs imaginary parts of data.

enable

bool, *optional*, *default = True*

Executed the real v/s imaginary data plotting.

fields

list of str, *optional*, *default = ‘ ‘*

Fields to plot. Specify by field id, index or keys like, gcal, bpcal.

column

str, *optional*, *default = corrected*

Data column to plot.

avgtime

str, *optional*, *default = 10*

Time to average for plotting, in seconds.

avgchannel

str, *optional*, *default = 10*

Number of channels to average for plotting.

amp_phase

Plot Amplitude vs Phase for data.

enable

bool, *optional*, *default = True*

Executes the plotting of amplitude v/s phase for data.

fields

list of str, *optional*, *default = ' '*

Fields to plot. Specify by field id, index or keys like: gcal, bpcal.

column

str, *optional*, *default = corrected*

Data column to plot.

avgtime

str, *optional*, *default = 10*

Time to average for plotting, in seconds.

avgchannel

str, *optional*, *default = 10*

Number of channels to average for plotting.

amp_uvwave

Plot data amplitude v/s uvwave.

enable

bool, *optional*, *default = True*

Executes plotting data amplitude as a function of uvwave.

fields

list of str, *optional*, *default = ' '*

Fields to plot. Specify by field id, index or keys like: gcal, bpcal.

column

str, *optional*, *default = corrected*

Data column to plot.

avgtime

str, *optional*, *default = 10*

Time to average for plotting, in seconds.

avgchannel

str, *optional*, *default = 10*

Number of channels to average for plotting.

amp_ant

Plot data amplitde v/s antenna.

enable

bool, *optional*, *default = True*

Executes plotting data amplitude v/s antennas.

fields

list of str, *optional*, *default = ‘ ‘*

Fields to plot. Specify by field id, index or keys like: gcal, bpcal.

column

str, *optional*, *default = corrected*

Data column to plot.

avgtime

str, *optional*, *default = 10*

Time to average for plotting, in seconds.

avgchannel

str, *optional*, *default = 10*

Number of channels to average for plotting.

phase_uvwave

Plot data phase v/s uvwave.

enable

bool, *optional*, *default = True*

Executes plotting data phase v/s uvwave.

fields

list of str, *optional*, *default = ‘ ‘*

Fields to plot. Specify by field id, index or keys like: gcal, bpcal.

column

str, *optional*, *default = corrected*

Data column to plot.

avgtime

str, *optional*, *default = 10*

Time to average for plotting, in seconds.

avgchannel

str, *optional*, *default = 10*

Number of channels to average for plotting.

amp_scan

Plot data amplitude v/s scan number.

enable

bool, *optional*, *default = True*

Executes plotting data amplitude v/s scan number.

fields

list of str, *optional*, *default = ' '*

Fields to plot. Specify by field id, index or keys like: gcal, bpcal.

column

str, *optional*, *default = corrected*

Data column to plot.

avgtime

str, *optional*, *default = 10*

Time to average for plotting, in seconds.

avgchannel

str, *optional*, *default = 10*

Number of channels to average for plotting.

amp_chan

Plot Amplitude vs Channel data.

enable

bool, *optional*, *default = True*

Executes the plotting of amplitude v/s phase for data.

fields

list of str, *optional*, *default = ' '*

Fields to plot. Specify by field id, index or keys like: gcal, bpcal.

column

str, *optional*, *default = corrected*

Data column to plot.

avgtime

str, *optional*, *default = 10*

Time to average for plotting, in seconds.

avgchannel

str, *optional*, *default = 10*

Number of channels to average for plotting.

phase_chan

Plot Phase vs Chan.

enable

bool, *optional*, *default = True*

Executes the plotting of amplitude v/s phase for data.

fields

list of str, *optional*, *default = ' '*

Fields to plot. Specify by field id, index or keys like: gcal, bpcal.

column

str, *optional*, *default = corrected*

Data column to plot.

avgtime

str, *optional*, *default = 10*

Time to average for plotting, in seconds.

avgchannel

str, *optional*, *default = 10*

Number of channels to average for plotting.

2.5.9 split_target

Split and average target data

enable

bool

Execute this worker

label_in

str, *optional*, *default = ‘ ‘*

Label of the input dataset

label_out

str, *optional*, *default = corr*

Label of the output dataset

split_target

Split data

enable

bool, *optional*, *default = True*

Execute this section

time_average

str, *optional*, *default = ‘ ‘*

Time averaging

freq_average

int, *optional*, *default = 1*

Frequency averaging

column

str, *optional*, *default = corrected*

Column to split, default is ‘corrected’.

correlation

str, *optional*, *default = ‘ ‘*

Select correlations

usewtspectrum

bool, *optional*, *default = True*

Create a WEIGHT_SPECTRUM column in the output MS.

spw

str, *optional*, *default = ‘ ‘*

Select spectral windows and channels

otfcal

Apply OTF calibration

enable

bool, *optional*, *default = False*

Execute this section

callabel

str, *optional*, *default = 1gc1*

Label of calibration tables to be used

apply_delay_cal

Apply the delay correction calibration table to specified fields via the CASA applycal task.

enable

bool, *optional*, *default = True*

Executes application of delay correction calibration table.

field

list of str

Field to select in the delay correction calibration table. Specify either the field number, name or as corrsponding to field spec in observation config, e.g. 'bpcal'.

apply_bp_cal

Apply the bandpass table to specified fields via the CASA applycal task.

enable

bool, *optional*, *default = True*

Executes application of bandpass table.

field

list of str

Field to select in the bandpass table. Specify either the field number, name or as corrsponding to field spec in observation config, e.g. 'bpcal'.

apply_gain_cal_gain

Apply the gain calibration table to specified fields via the CASA applycal task.

enable

bool, *optional*, *default = False*

Executes application of gain calibration table.

field

list of str

Field to select in the gain calibration table. Specify either the field number, name or as corrsponding to field spec in observation config, e.g. 'gcal'.

apply_transfer_fluxscale

Apply the fluxscale table to specified fields via the CASA applycal task.

enable

bool, *optional*, *default = True*

Executes application of fluxscale table.

field

list of str, *optional*, *default = gcal*

Field to select in the fluxscale table. Specify either the field number, name or as corrsponding to field spec in observation config, e.g. 'gcal'.

changecentre

changes the phase centre

enable

bool, *optional*, *default = False*

Execute this section

ra

str, *optional*, *default = 0h0m0.0s*

J2000 RA of new phase centre, format XXhXXmXX.XXs, default is empty string

dec

str, *optional*, *default = 0d0m0.0s*

J2000 Dec of new phase centre, format XXdXXmXX.XXs, default is empty string

obsinfo

Get observation information

enable

bool, *optional*, *default = True*

Execute this section

listobs

bool, *optional*, *default = True*

Run CASA listobs

summary_json

bool, *optional*, *default = True*

Run MSUtils function

2.5.10 masking

Create mask from catalog and/or merge with mask of extended source

enable

bool

Execute this segment

centre_coord

list of str, *optional*, *default = HH:MM:SS , DD:MM:SS*

Coordinates of the centre of the field of view read from reference_dir by default

label

str, *optional*, *default = corr*

Label of the .MS file where to find information about the target

mask_size

int, *optional*, *default = 900*

Number of pixels in the mask (must be the same as img_npix in selfcal worker)

cell_size

float, *optional*, *default = 2.*

Size of pixel in the mask (arcsec, must be the same as img_cell in selfcal worker)

name_mask

str, *optional*, *default = catalog_mask.fits*

Name of the output mask generated from catalog

extended_source_input

str, *optional*, *default = Fornaxa_vla.FITS*

Name of the input mask for particularly extended sources in the field

final_mask

str, *optional*, *default = final_mask.fits*

Name of the output final mask

query_catalog

Query catalog to select field/sources from which extract the mask

enable

bool, *optional*, *default = true*

Execute this worker

catalog

{"NVSS", "SUMSS"}, *optional*, *default = SUMSS*

Name of catalog to query [NVSS/SUMSS]

width_image

str, *optional*, *default = 1.2d*

Width of the region of sky we want to mask (keep larger than dirty image)

thresh_nvss

float, *optional*, *default = 10e-3*

Cutoff to select sources in the SUMSS map, corrected for the primary beam (Jy) or cutoff in sigmas for sofia source finder

pb_correction

Correct input image for primary beam before exctracting mask

enable

bool, *optional*, *default = true*

Execute this worker

frequency

float, *optional*, *default = 1.420405752*

Primary beam size changes with frequency, provide central frequency of considered dataset

make_mask

Build mask from existing image using SoFiA and/or threshold cutoff

enable

bool, *optional*, *default = true*

Execute this worker

mask_with

{"thresh", "sofia"}, *optional*, *default = sofia*

Tool to use for masking

input_image

{"pbcorr", "path_to_mask"}, *optional*, *default = pbcorr*

Input image where to create mask ???? what is this ???

thresh_lev

int, *optional*, *default = 5*

Cutoff to select sources in the SUMSS map, corrected for the primary beam (Jy) or cutoff in sigmas for sofia source finder

scale_noise_window

int, *optional*, *default = 101*

window size where SoFiA measures the local rms, units of pixels

merge_with_extended

Merge with mask of extended source

enable

bool, *optional*, *default = False*

Execute this worker

extended_source_input

str, *optional*, *default = extended_mask.fits*

name of image of extended source to merge with current image

mask_with

{"thresh", "sofia"}, *optional*, *default = thresh*

Tool to use for masking

thresh_lev

float, *optional*, *default = 8e-2*

Cutoff to select sources in the SUMSS map, corrected for the primary beam (Jy) or cutoff in sigmas for sofia source finder

2.5.11 self_cal

Perform Self calibration on the data

enable

bool

Execute this segment

label

str, *optional*, *default = corr*

Label of the .MS files to process

undo_subtractmodelcol

bool, *optional*, *default = False*

replace the corrected column with the sum of corrected and model columns to undo continuum subtraction that may have been done by the image HI worker.

primary_beam

bool, *optional*, *default = False*

Use primary beam

calibrate_with

{"meqtrees", "cubical"}, *optional*, *default = cubical*

Tool to use for calibration

spwid

int, *optional*, *default = 0*

Provide spectral window id

ncpu

int, *optional*, *default = 5*

number of cpu's to use

minuvw_m

int, *optional*, *default = 0*

Exclude baselines shorter than this value (given in metres) from the imaging and selfcalibration loop.

img_npix

int, *optional*, *default = 1800*

Number of pixels in output image

img_padding

float, *optional*, *default = 1.3*

Padding in WSclean

img_mgain

float, *optional*, *default = 0.99*

Image CLEANing gain

img_cell

float, *optional*, *default = 2.*

Image pixel size (arcsec)

img_weight

{"briggs", "uniform", "natural"}, *optional*, *default = briggs*

Image weighting type. If Briggs, set the img robust parameter

img_robust

float, *optional*, *default = 0.*

Briggs robust value

img_uvtaper

str, *optional*, *default = 0*

Taper for imaging (arcsec)

img_niter

int, *optional*, *default = 1000000*

Number of cleaning iterations

img_nmiter

int, *optional*, *default = 0*

Number of major cycles

img_cleanborder

float, *optional*, *default = 1.3*

Clean border

img_nchans

int, *optional*, *default = 3*

Number of channesls in output image

img_joinchannels

bool, *optional*, *default = True*

Join channels to create MFS image

img_fit_spectral_pol

int, *optional*, *default = 2*

Number of spectral polynomial terms to fit to each clean component. This is equal to the order of the polynomial plus 1.

img_pol

str, *optional*, *default = I*

Stokes image to create

cal_gain_amplitude_clip_low

float, *optional*, *default = 0.5*

Lower gain amplitude clipping

cal_gain_amplitude_clip_high

float, *optional*, *default = 2.*

Higher gain amplitude clipping

cal_niter

int, *optional*, *default = 2*

Number of self-calibration iterations to perform

start_at_iter

int, *optional*, *default = 1*

Start self-cal iteration loop at this start value, 1-based.

cal_time_chunk

int, *optional*, *default = 1*

Chunk data up by this number of timeslots. This limits the amount of data processed at once. Smaller chunks allow for a smaller RAM footprint and greater parallelism but sets an upper limit on the time solution intervals that may be employed. 0 means use full time axis but does not cross scan boundaries.

cal_freq_chunk

int, *optional*, *default = 0*

Chunk data up by this number of channels. This limits the amount of data processed at once. Smaller chunks allow for a smaller RAM footprint and greater parallelism but sets an upper limit on the frequency solution intervals that may be employed. 0 means use full frequency axis but does not cross SPW boundaries.

aimfast

Quality assessment parameter

enable

bool, *optional*, *default = False*

Execute this segment

tolerance

float, *optional*, *default = 0.02*

Relative change in weighted mean of several indicators from aimfast.

convergence_criteria

list of str, *optional*, *default = DR*

The residual statistic to check convergence against. Every criterium listed will be combined into a weighted mean. Options ["DR","SKEW","KURT","STDDev","MEAN"]. Note that when calibrate model_mode = 'vis_only' DR is not an option.

area_factor

int, *optional*, *default = 6*

Peak flux source area multiplying factor i.e tot_area = psf-size*af

normality_model

{"normaltest", "shapiro"}, *optional*, *default = normaltest*

normality test model to use. Note that normaltest is the D'Agostino

plot

bool, *optional*, *default = True*

Generate html plots for comparing catalogs and residuals

image

Imaging parameter

enable

bool, *optional*, *default = True*

Execute this segment

auto_mask

list of float, *optional*, *default = 30, 10, 7*

Auto masking threshold

auto_threshold

list of float, *optional*, *default = 0.5*

Auto clean threshold

column

list of str, *optional*, *default = DATA, CORRECTED_DATA*

Column to image

mask_from_sky

bool, *optional*, *default = False*

switch on cleaning within mask from fits file

fits_mask

list of str, *optional*, *default = catalog_mask.fits*

filename of fits mask (in output/masking folder)

multi_scale

bool, *optional*, *default = False*

switch on multiscale cleaning

multi_scale_scales

list of int, *optional*, *default = 10, 20, 30*

scales of multiscale [0,10,20,etc, etc] in pixels

local_rms

bool, *optional*, *default = False*

switch on local rms measurement for cleaning

sofia_mask

Run SoFiA source finder to produce a source mask and a Moment-0 map

enable

bool, *optional*, *default = False*

Execute segment sofia (yes/no)?

threshold

float, *optional*, *default = 4.0*

SoFiA source finding threshold.

flag

bool, *optional*, *default = False*

Use flag regions (yes/no)?

flagregion

list of str, *optional*, *default = ‘ ‘*

Pixel/channel range(s) to be flagged prior to source finding. Format is [[x1, x2, y1, y2, z1, z2], . . .].

inputmask

str, *optional*, *default = ‘ ‘*

input mask over which add Sofia’s

fornax_special

bool, *optional*, *default = False*

Activates masking of Fornax A using Sofia

fornax_thresh

list of float, *optional*, *default = 4.0*

SoFiA source finding threshold. Default is 4.0.

use_sofia

bool, *optional*, *default = False*

use sofia for mask of Fornax A instead of Fomalont mask

scale_noise_window

int, *optional*, *default = 31*

window size where to measure local rms in pixels

positivity

bool, *optional*, *default = False*

merges only positive pixesl of sources in mask

extract_sources

Source finding parameters

enable

bool, *optional*, *default = False*

Execute this segment

sourcefinder

str, *optional*, *default = pybdsm*

choose your favorite sourcefinder pybdsm, (pybdsf), sofia

local_rms

bool, *optional*, *default = False*

Execute this segment

spi

bool, *optional*, *default = False*

Extract source spectral index

thresh_pix

list of int, *optional*, *default = 5*

Source finder pixel threshold

thresh_isl

list of int, *optional*, *default = 3*

Source finder island threshold

calibrate

Calibration parameters

enable

bool, *optional*, *default = True*

Execute this segment

model

list of str, *optional*, *default = 1,2*

Model number to use [or combination e.g. '1+2' to use first and second models]

output_data

list of str, *optional*, *default = CORR_DATA*

Data to output after calibration

gain_matrix_type

list of str, *optional*, *default = GainDiagPhase*

Gain matrix type

model_mode

str, *optional*, *default = vis_only*

pybdsm_vis, pybdsm_only, vis_only are the possible options

shared_memory

str, *optional*, *default = 100Gb*

Set the amount of shared memory for cubical. Default '100Gb'

two_step

bool, *optional*, *default = False*

Trigger a two step calibration process where the phase only calibration is applied before continuing with amplitude + phase cal. When cubical is used this happens simultaneous and gain parameters can be used with DDsols parameters. Set DDsol_time to -1 one to avoid amplitude calibration in an itereation. The parameter DDjones should be set to false.

add_vis_model

bool, *optional*, *default = True*

Add/Use clean components from latest imaging step to/as sky model for calibation

Gsols_time

list of float, *optional*, *default = 1*

G-Jones time solution interval. The parameter cal_time_chunk above should a multiple of Gsols_time. 0 means a single solution for the full time chunk.

Gsols_channel

list of float, *optional*, *default = 0*

G-Jones frequency solution interval. The parameter cal_frq_chunk above should a multiple of Gsols_channel. 0 means a single solution for the full frequency chunk.

Bjones

bool, *optional*, *default = False*

Enable Bjones

Bsols_time

list of int, *optional*, *default = 0*

Gsols for individual calibration steps, if not given will default to cal_Gsols

Bsols_channel

list of float, *optional*, *default = 2*

Gsols for individual calibration steps, if not given will default to cal_Gsols

DDjones

bool, *optional*, *default = False*

Enable direction dependent calibration, currently experimental.

DDsols_time

list of float, *optional*, *default = 0*

Calibration solution intervals

DDsols_channel

list of float, *optional*, *default = 0*

Calibration solution intervals

weight_column

str, *optional*, *default = WEIGHT*

Column with weights

madmax_flagging

bool, *optional*, *default = True*

Flags based on maximum of mad

madmax_flag_thresh

list of int, *optional*, *default = 0, 10*

Threshold for madmax flagging

sol_term_iters

str, *optional*, *default = auto*

Number of iterations per Jones term. If set to 'auto', uses hardcoded iteration numbers depending on the jones chain.

dist_max_chunks

int, *optional*, *default = 4*

Maximum number of time/freq data-chunks to load into memory simultaneously. If 0, then as many as possible will be loaded.

ragavi_plot

Plotting dignostics plots for delay correction calibration.

enable

bool, *optional*, *default = False*

Enables plotting dignostics

gaintype

list of str, *optional*, *default = G*

List of gain solution types

field

list of int, *optional*, *default = 0*

Fields to plot. Specify by field id, index.

restore_model

Restore modelled to final calibrated residual image

enable

bool, *optional*, *default = False*

Execute this segment

model

str, *optional*, *default = 1+2*

Model number to use [or combination e.g. '1+2' to use first and second models]

clean_model

str, *optional*, *default = 3*

Clean model number to use [or combination e.g. '1+2' to use first and second models]

flagging_summary

Output the flagging summary

enable

bool, *optional*, *default = False*

Execute this segment

transfer_apply_gains

Interpolate gains over the high frequency resolution data

enable

bool, *optional*, *default = False*

Execute this segment

transfer_to_label

str, *optional*, *default = corr*

label of cross-calibrated .ms file to which to transfer and apply the selfcal gains

interpolate

To interpolate the gains or not to interpolate the gains. That is indeed the question.

enable

bool, *optional*, *default = True*

Enable gain interpolation.

time_int

int, *optional*, *default = 1*

Solution interval in time (units of timeslots/intergration time) to transfer gains.

freq_int

int, *optional*, *default = 0*

Solution interval in frequency (units of channels) to transfer gains.

time_chunk

int, *optional*, *default = 128*

Time chunk in units of timeslots for transferring gains with Cubical.

freq_chunk

int, *optional*, *default = 0*

Frequency chunk in units of channels for transferring gains with Cubical. '0' means the whole spw.

transfer_model

Transfer model from last WSclean imaging run to the MODEL_DATA column of another .MS

enable

bool, *optional*, *default = True*

Execute this segment (default False)

transfer_to_label

str, *optional*, *default = corr*

label of .ms file to which to transfer the model

model

str, *optional*, *default = auto*

Name of the sky model file (currently the only supported format is that of WSclean component lists). When 'auto', the pipeline builds the file name from the input parameters of the selfcal loop. The file is assumed to be in the 'output' directory.

spectra

bool, *optional*, *default = True*

Model sources as non-flat spectra. The spectral coefficients and reference frequency must be present in the sky model.

row_chunks

int, *optional*, *default = 0*

Number of rows of input .MS that are processed in a single chunk.

model_chunks

int, *optional*, *default = 0*

Number of sky model components that are processed in a single chunk.

exp-sign-convention

str, *optional*, *default = casa*

Sign convention to use for the complex exponential. 'casa' specifies the e^(2.pi.I) convention while 'thompson' specifies the e^(-2.pi.I) convention in the white book and Fourier analysis literature. Defaults to 'casa'.

within

str, *optional*, *default = ' '*

Give JS9 region file. Only sources within those regions will be included.

points_only

bool, *optional*, *default = False*

Select only point-only sources. Default is False.

num_sources

int, *optional*, *default = 0*

Select only N brightest sources. Default is 0

num_workers

int, *optional*, *default = 0*

Explicitly set the number of worker threads. Default is 0, meaning it uses all threads.

memory_fraction

float, *optional*, *default = 0.5*

Fraction of system RAM that can be used. Used when setting automatically the chunk size.

2.5.12 image_line

Process visibilities for spectral line work and create line cubes and images.

enable

bool

Execute segment image_line.

label

str, *optional*, *default = corr*

Label of names of MS data sets to be used. MS data set names will always start with the data set id, followed by a hyphen, followed by desc.

line_name

str, *optional*, *default = HI*

Line name string to be used for output file names.

restfreq

str, *optional*, *default = 1.420405752GHz*

Rest frequency default value for this worker.

subtractmodelcol

Replace the column CORRECTED_DATA with the difference CORRECTED_DATA - MODEL_DATA. This is useful for continuum subtraction as it enables the subtraction of the most recent continuum clean model.

enable

bool, *optional*, *default = True*

Execute segment subtractmodelcol.

mstransform

Perform UVLIN continuum subtraction and/or doppler tracking corrections

enable

bool, *optional*, *default = True*

Execute segment doppler correction.

telescope

{"meerkat", "vla", "gmrt", "wsrt", "atca", "askap"}, *optional*, *default = meerkat*

The name of the telescope from which observations were made. Default is the 'meerkat' telescope. Current options are gmrt, vla, wsrt, atca.

doppler

bool, *optional*, *default = True*

Transform channel labels and visibilities to a different spectral reference frame.

mode

{"frequency"}, *optional*, *default = frequency*

Regridding mode (channel/velocity/frequency/channel_b). IMPORTANT - Currently only frequency mode is supported.

outframe

{"", "topo", "geo", "lsrk", "lsrd", "bary", "galacto", "lgroup", "cmb", "source"}, *optional*, *default = bary*

Output reference frame, options '', 'topo', 'geo', 'lsrk', 'lsrd', 'bary', 'galacto', 'lgroup', 'cmb', 'source'

veltype

str, *optional*, *default = radio*

Definition of velocity (as used in mode), radio or optical.

outchangrid

str, *optional*, *default = auto*

Output channel grid for Doppler correction. Default is 'auto', and the pipeline will calculate the appropriate channel grid. If not 'auto' it must be in the format 'nchan,chan0,chanw' where nchan is an integer, and chan0 and chanw must include units appropriate for the chosen mode (see parameter 'mode' above)

uvlin

bool, *optional*, *default = True*

Perform continuum subtraction as in task uvcontsub whilst regridding within mstransform.

fitspw

str, *optional*, *default = ' '*

Spectral window channel selection for fitting the continuumSelection of line-free channels using CASA syntax (e.g. '0:0~100;150:300'). If set to null, a fit to all unflagges visibilities will be performed.

fitorder

int, *optional*, *default = 1*

Polynomial order for the continuum fits

column

str, *optional*, *default = corrected*

Data column to use.

obsinfo

bool, *optional*, *default = True*

Create obsinfo.txt and obsinfo.json of MS file created by mstransform.

sunblocker

Use sunblocker to remove solar RFI. See description of sunblocker on github repository gigjozsa/sunblocker in method phazer of module sunblocker.py.

enable

bool, *optional*, *default = False*

Execute segment sunblocker.

use_mstransform

bool, *optional*, *default = True*

Execute sunblocker on continuum-subtracted data (otherwise use non-continuum-subtracted data).

imsize

int, *optional*, *default = 900*

Image size (use the same as in wsclean_image or casa_image).

cell

float, *optional*, *default = 2.*

Cell size in arcsec (use the same as in wsclean_image or casa_image).

threshold

float, *optional*, *default = 4.*

Distance from average beyond which data are flagged in units of sigma.

vampirisms

bool, *optional*, *default = False*

Apply only to data taken during day time.

uvmax

float, *optional*, *default = 2000*

Maximum uvdistance in wavelength to be analysed.

uvmin

float, *optional*, *default = 0.*

Minimum uvdistance in wavelength to be analysed.

make_cube

Make a line cube with either WSclean + SoFiA (for clean masks) or Casa.

enable

bool, *optional*, *default = True*

Execute segment make_cube.

image_with

{“wsclean”, “casa”}, *optional*, *default = wsclean*

Choose whether to image with WSclean + SoFiA (“wsclean”) or with Casa (“casa”).

use_mstransform

bool, *optional*, *default = True*

Image the .MS file(s) made by CASA MSTRANSFORM (continuum-subtracted and/or Doppler corrected).

pol

str, *optional*, *default = I*

Polarizations in output cube (I,Q,U,V,XX,YY,XY,YX,RR,LL,RL,LR and combinations).

spwid

int, *optional*, *default = 0*

Spectral window to use.

nchans

int, *optional*, *default = 0*

Number of channels of HI cube, 0 means all channels.

firstchan

int, *optional*, *default = 0*

First channel of HI cube.

binchans

int, *optional*, *default = 1*

Integer binning of channels.

npix

seq, *optional*, *default = 900 , 900*

Image size in pixels. List of integers (width and height) or a single integer for square images.

cell

float, *optional*, *default = 2*

Scale of a pixel. Default unit is arcsec, but can be specificied, e.g. ‘scale 20asec’.

padding

float, *optional*, *default = 1.2*

Images have initial size padding*npix, and are later trimmed to npix.

weight

str, *optional*, *default = briggs*

Weightmode can be natural, uniform, briggs. When using Briggs weighting, the Robustness parameter robust has to be specified in addition.

robust

float, *optional*, *default = 0*

Robust parameter in case of Briggs weighting.

taper

float, *optional*, *default = 0*

Gaussian taper FWHM in arcsec. Zero means no tapering.

niter

int, *optional*, *default = 1000000*

Maximum number of clean iterations to perform.

gain

float, *optional*, *default = 0.1*

Fraction of the peak that will be cleaned in each minor iteration.

wscl_mgain

float, *optional*, *default = 1.0*

WSclean gain for major iterations, i.e., maximum fraction of peak that will be cleaned in each major iteration.

wscl_sofia_niter

int, *optional*, *default = 2*

Maximum number of WSclean + SoFiA iterations. The initial cleaning is done with WSclean automasking or with a user clean mask. Subsequent iterations use a SoFiA clean mask. A value of 1 means that WSclean is only executed once and SoFiA is not used.

wscl_sofia_converge

float, *optional*, *default = 1.1*

Stop the WSclean + SoFiA iterations if the cube RMS has dropped by a factor < wscl_sofia_converge when comparing the last two iterations. If set to 0 then the maximum number of iterations is performed regardless of the noise change.

wscl_keep_final_products_only

bool, *optional*, *default = False*

If set to true it deletes WSclean + Sofia intermediate cubes from the output directory, if set to false it keeps all the cubes of the WSclean + SoFiA iterations.

wscl_user_clean_mask

str, *optional*, *default = ‘ ‘*

WSclean user clean mask for first WSclean + SoFiA iteration (give filename, to be located in the output/masking folder).

wscl_auto_mask

float, *optional*, *default = 10*

WSclean option. Construct a mask from found components and when a threshold of sigma is reached, continue cleaning with the mask down to the normal threshold.

wscl_auto_threshold

float, *optional*, *default = 0.5*

WSclean option. Auto clean threshold.

wscl_make_cube

bool, *optional*, *default = True*

If set to true the output of WSclean is a data cube, if set to false the output is one fits file per spectral channel.

wscl_no_update_mod

bool, *optional*, *default = True*

If set to true, WSclean will not store the line clean model in MODEL_DATA.

wscl_multi_scale

bool, *optional*, *default = False*

Switch on WSclean multiscale cleaning.

wscl_multi_scale_scales

list of int, *optional*, *default = 0, 10, 20, 30*

List of scales of WSclean multiscale in units of pixels. Only used is wscl_multi_scale is set to True.

wscl_multi_scale_bias

float, *optional*, *default = 0.6*

Parameter to set the bias towards larger scales during multi-scale cleaning. A lower bias will give preference to larger scales.

casa_threshold

str, *optional*, *default = 10mJy*

Flux level to stop CASA cleaning, must include units, e.g. '1.0mJy'.

casa_port2fits

bool, *optional*, *default = False*

Port CASA output to fits files.

remove_stokes_axis

Remove Stokes axis from HI cube

enable

bool, *optional*, *default = False*

Execute this segment.

pb_cube

Make primary beam cube

enable

bool, *optional*, *default = False*

Execute this segment.

apply_pb

bool, *optional*, *default = False*

Whether to apply the primary beam correction to the image cube.

freq_to_vel

Convert the spectral axis' header keys of the HI cube from frequency to velocity in the radio definition, v=c(1-obsfreq/restfreq). No change of spectra reference frame is performed.

enable

bool, *optional*, *default = False*

Execute conversion.

reverse

bool, *optional*, *default = False*

Perform the inverse transformation and change the cube 3rd axis from radio velocity to frequency.

sofia

Run SoFiA source finder to produce a source mask and a Moment-0 map

enable

bool, *optional*, *default = True*

Execute segment sofia?

rmsMode

str, *optional*, *default = mad*

Method to determine rms ('mad' for using median absolute deviation, 'std' for using standard deviation, 'negative' for using Gaussian fit to negative voxels).

threshold

float, *optional*, *default = 4.0*

SoFiA source finding threshold.

flag

bool, *optional*, *default = False*

Use flag regions?

flagregion

list of int, *optional*, *default = 10, 10*

Pixel/channel range(s) to be flagged prior to source finding. Format is [[x1, x2, y1, y2, z1, z2], . . .].

merge

bool, *optional*, *default = False*

Use method to de-select and merge emission islands detected by any of SoFiA source finding algorithms. If turned on, pixels with a separation of less than mergeX pixels in x direction and less than mergeY pixels in y-direction and less than z pixels in z-direction will be merged and

identified as a single object in the mask. Detections whose extent in x-direction is smaller than minSizeX, in y direction is smaller than minSizeY, and in z-direction is smaller than minSizeZ will be removed from the mask. Parameter merge determines if the merging should be applied.

mergeX

int, *optional*, *default = 2*

Merge-'radius' in x-direction.

mergeY

int, *optional*, *default = 2*

Merge-'radius' in y-direction.

mergeZ

int, *optional*, *default = 3*

Merge-'radius' in z-direction (velocity direction).

minSizeX

int, *optional*, *default = 3*

Minimum size in x-direction.

minSizeY

int, *optional*, *default = 3*

Minimum size in y-direction.

minSizeZ

int, *optional*, *default = 3*

Minimum size in y-direction.

do_cubelets

bool, *optional*, *default = True*

Create cubelets of HI sources.

do_mom0

bool, *optional*, *default = True*

Create moment 0 map.

do_mom1

bool, *optional*, *default = True*

Create moment 1 map.

sharpener

Run sharpener to extract spectrum of all continuum sources against the lines of sight. The spectra are then plotted.

enable

bool, *optional*, *default = False*

Execute sharpener?

catalog

{"NVSS", "PYBDSF"}, *optional*, *default = PYBDSF*

Type of catalog to use (PYBDSF/NVSS).

channels_per_plot

int, *optional*, *default = 50*

Number of channels to plot per detail plot.

thresh

float, *optional*, *default = 20*

Threshold to select sources in online catalogs (mJy).

width

str, *optional*, *default = 1.0d*

Field of view of output catalog (degrees).

label

str, *optional*, *default = ' '*

Prefix label of plot names and titles.

2.5.13 mosaic

Mosaic images output by selfcal or image_line worker.

enable

bool

Execute mosaic segment(yes/no). Default is no.

mosaic_type

str

Type of mosaic to be made(continuum/spectral). Default is continuum.

domontage

Re-grid the input images, and associated beams. Default is true.

enable

bool, *optional*, *default = True*

Execute this domontage section.

cutoff

float, *optional*, *default = 0.1*

The cutoff in the primary beam to use, assuming a Gaussian at the moment. Default is 0.1 (signifying 10 per cent).

use_MFS_images

bool, *optional*, *default = False*

Indicate that the images to be mosaicked were created using MFS. Default is false.

name

str, *optional*, *default = ' '*

The prefix to be used for output files. Default is the pipeline prefix(pipeline.prefix).

target_images

seq, *optional*, *default = directory/first_image.fits, directory/second_image.fits*

List of images to be mosaicked, with suffix of image.fits being expected.

dish_diameter

float, *optional*, *default = 13.5*

If no continuum pb.fits are already in place, user needs to specify the dish diameter(in units of m) so that rudimentary primary beams can be created.

ref_frequency

float, *optional*, *default = 1383685546.875*

If no continuum pb.fits are already in place, user needs to specify the reference frequency(in units of Hz) so that rudimentary primary beams can be created.

label

str, *optional*, *default = corr*

For autoselection of images, this needs to match the label setting used for the self_cal worker(when mosaicking continuum images) or the image_line worker(when mosaicking cubes).

line_name

str, *optional*, *default = HI*

Spectral mode only - - If autoselection is used to find the final cubes, this needs to match the line_name parameter used for the image_line_worker. Default is HI.

CHAPTER 3

Acknowledgements

The MeerKATHI team acknowledges support from the following institutes:

- South African Radio Astronomy Observatory (SARAO)
- Rhodes University
- Istituto Nazionale di Astrofisica (INAF) - Osservatorio Astronomico di Cagliari
- ASTRON
- Kapteyn Astronomical Institute
- Ruhr-Universitat Bochum

and from the following funding allocations:

- Starting Grant of the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant number 679629, project name FORNAX)
- Grant from the Italian Ministry of Foreign Affairs and International Cooperation (MAECI Grant Number ZA18GR02) and the South African Department of Science and Technology's National Research Foundation (DST-NRF Grant Number 113121) as part of the ISARP Joint Research Scheme.
- At RUB this work is partly supported by BMBF project 05A17PC2