# MeqSilhouette Documentation

*Release 2.7*

**Iniyan Natarajan, Roger Deane**

**Jun 23, 2021**

# Contents:

# CHAPTER 1

# Introduction to MeqSilhouette

MeqSilhouette is a radio interferometry observation simulator. Blecher et al. (2017) presents an earlier version which could generate synthetic data and apply corruptions such as scattering by interstellar medium (ISM), atmospheric effects, and antenna pointing offsets. The current version, v2, is capable of generating fully polarised, time-variable, spectrally-resolved sky models and propagation path effects including (but not limited to) instrumental polarisation, updated atmospheric effects and pointing offsets, time-variable antenna gains, and bandpass effects.

This documentation explains the steps necessary for installing and using MeqSilhouette.

CHAPTER 2

# Requirements & Installation

## 2.1 Ubuntu 18.04 + Python 2.7

It is recommended to install the dependencies via the KERN-6 software suite:

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository -s ppa:kernsuite/kern-6
$ sudo apt-add-repository multiverse
$ sudo apt-add-repository restricted
$ sudo apt-get update
```

Install the following dependencies via *apt-get*:

```
$ sudo apt-get install meqtrees meqtrees-timba tigger tigger-lsm python-astro-tigger \
python-astro-tigger-lsm casalite wsclean pyxis python-casacore
```

**Note:** The casacore data must be kept up-to-date. This can be done by following the instructions on the CASA website.

Optionally, install Latex (for creating paper-quality plots):

```
$ sudo apt-get install texlive-latex-extra texlive-fonts-recommended dvipng
```

*AATM v0.5* can be obtained from here. AATM cannot create the executables necessary for running MeqSilhouette without the *boost* libraries. In Ubuntu 18.04, ensure that the packages *libboost-program-options-dev*, *libboost-program-options1.65-dev*, and *libboost-program-options1.65.1* are installed. Once these are installed, proceed as follows:

```
$ cd /path/to/aatm-source-code
$ ./configure --prefix=/path/to/aatm-installation
$ make
```

(continues on next page)

(continued from previous page)

```
$ make install
$ export PATH=$PATH:/path/to/install/aatm-installation/bin
```

If using a virtual environment, the following steps are necessary (skip ahead if not using *virtualenv*):

```
$ virtualenv /path/to/env
$ source /path/to/env/bin/activate
$ pip install -U pip setuptools wheel # recommended
```

**Note:** If –system-site-packages is not passed to virtualenv, the global packages installed via apt-get above will not be available and must be installed manually from source.

Now, check out MeqSilhouette version 2.7 from GitHub and install using pip:

```
$ git clone --branch v2.7 https://github.com/rdeane/MeqSilhouette.git
$ cd MeqSilhouette
$ pip install .
```

The *turbo-sim.py* script from MeqTrees is included in the *framework* directory. If you do not have it, add a symbolic link to the copy in your *meqtrees-cattery* installation:

```
$ ln -s /path/to/meqtrees-cattery/Cattery/Siamese/turbo-sim.py /path/to/MeqSilhouette/
↪framework/turbo-sim.py
```

## 2.2 Building Singularity image

The recommended way to run MeqSilhouette is via *Singularity*. The Singularity definition file *singularity.def* is shipped with the repository. If you do not have Singularity installed on your system, follow the installation instructions on the Singularity website. Once Singularity is installed, the singularity image file (SIF) can be created as follows:

```
$ sudo singularity build meqsilhouette.sif singularity.def
```

Note that the build process automatically ensures that *casacore* data are up-to-date. If these data are missing, and if you do not have a working casa installation from which to obtain this information, simply rebuild the image to eliminate this warning thrown by *CASA*.

## 2.3 Building Docker image

*Docker* is also supported. Docker can be installed on your system via *apt-get*. Once installed, build the docker image as follows:

```
$ cd /path/to/Dockerfile
$ docker build -t meqsilhouette .
```

As before, the build process ensures that *casacore* data are up-to-date.

## 2.4 Known installation issues

1. If MeqTrees cannot see the *TiggerSkyModel* module that ought to load when *turbo-sim.py* is run (i.e. when an ASCII sky model is used), the parent directory of *Tigger* must be added to PYTHONPATH. Bear in mind that this may cause python version conflicts with other packages. In that case, it is recommended to have Tigger installed in a separate directory such as /opt/Tigger. For manual installation of Tigger and tigger-lsm, refer to their respective repositories. Without this, MeqSilhouette will still work with FITS images as input sky models.
2. If MeqSilhouette cannot find aatm, modify LD_LIBRARY_PATH as follows:

```
export LD_LIBRARY_PATH=/path/to/aatm-0.5/lib:$LD_LIBRARY_PATH
```

3. If the error *Incorrect qhull library called* is thrown, ensure **scipy==0.17** is installed.
4. MeqSilhouette will soon be ported to *astropy.fits* and *pyfits* will no longer be a dependency. As of now though, *pyfits* is still required. If *pyfits* throws an ImportError for the modules *gdbm/winreg*, a quick and dirty fix is to open the following file:

```
/path-to-virtualenv/lib/python2.7/site-packages/pyfits/extern/six.py
```

   and comment out the lines:

```
MovedModule("dbm_gnu", "gdbm", "dbm.gnu")
MovedModule("winreg", "_winreg")
```

CHAPTER 3

# Usage

MeqSilhouette can be run in a number of ways. They all require that a JSON parset file is passed as the sole argument.

## 3.1 On the local machine

If MeqSilhouette is installed on the local machine, it should already have the command *meqsilhouette* in PATH and can be run as follows:

```
$ meqsilhouette obs_settings.json
```

## 3.2 Via Singularity

There are two ways to run MeqSilhouette via Singularity. In both cases, it is the responsibility of the user to ensure that the relevant paths in the JSON parset file are writable.

### 3.2.1 Interactive mode

Interactively using the *shell* option:

```
$ singularity shell meqsilhouette.sif # drops the user inside the container
> meqsilhouette obs_settings.json
```

### 3.2.2 Command-line mode

Invoke the *run* option:

```
$ singularity run meqsilhouette.sif obs_settings.json
```

## 3.3 Via Docker

There are two ways to run MeqSilhouette via Docker as well. The interactive route is a bit more involved while the command-line mode is more straightforward.

As before, it is the responsibility of the user to ensure that the relevant paths in the JSON parset file are writable.

### 3.3.1 Interactive mode

A docker volume is the best way to share data between the container and the host. First copy the relevant input files to a directory with write access. Assuming that this directory is *~/data*, start the docker container as follows:

```
$ docker run -dit -P -v ~/data:/meqsdata meqsilhouette
```

where */meqsdata* is an arbitrary mount point in the running container to which the host directory *~/data* is to be bound. The above command will print the container ID which is a long string of characters (this can also be obtained by typing *docker ps -a* on the terminal). Once started, attach to the container using the first four characters *xxxx* of this ID:

```
$ docker attach xxxx
```

This will drop the user into the shell from which MeqSilhouette can be run as follows:

```
$ meqsilhouette /meqsdata/obs_settings.json
```

Since */meqsdata* is where *~/data* is mounted, any output files must be written to */meqsdata* for them to persist after the container is stopped. If the output files are written elsewhere, the execution will still be successful, but the files will not persist in host storage. To avoid this, the data generated must be manually copied into */meqsdata*.

### 3.3.2 Command-line mode

More easily, the following command on the terminal (with all the caveats about paths to input/output files mentioned above) executes MeqSilhouette:

```
$ docker run -v ~/data:/meqsdata meqsilhouette meqsilhouette /meqsdata/obs_settings.
↪json
```

Note that the first *meqsilhouette* is the name of the image to be run and the second *meqsilhouette* is the command that must be run within the container, since no default entrypoint/command is defined.

## 3.4 IPython/Jupyter Notebook

Running MeqSilhouette from any Python interpreter is as easy as firing up the interpreter and typing the following:

```
from meqsilhouette.driver import run_meqsilhouette
run_meqsilhouette.run_meqsilhouette('/path/to/JSON/parset/file')
```

The above command will run the default driver script shipped with the source code. This is the same script run by the command *meqsilhouette* on the command-line in the above cases.

### 3.4.1 For advanced users

By default, the *driver* script included with the source code will be used to parse the JSON file and generate the synthetic data. Advanced users can construct their own version of the driver script by importing the *framework* module in their code directly. For instance, additional operations on the Measurement Set such as flagging or averaging can be performed by an enhanced driver script tailored to the needs of the user.

CHAPTER 4

# Inputs

MeqSilhouette accepts as inputs various telescope, sky, and observation parameters in different formats. The master input file is a JSON parset containing many input parameters. Sample input files and settings can be found in the *data* subdirectory. Each input file type is explained in detail below.

## 4.1 JSON parset file

The input parset file is in JSON format with parameters that are loosely grouped into the following sets:

- General I/O parameters
- Measurement Set (MS) parameters (prefixed *ms_*)
- Imaging parameters (prefixed *im_*)
- Tropospheric parameters (prefixed *trop_*)
- Antenna pointing error parameters (prefixed *pointing_*)
- Bandpass corruption parameters (prefixed *bandpass_*)

| Parameter | Type | Units | Explanation |
|---|---|---|---|
| *outdirname* | string | | Output directory with absolute or relative path. Must be writable. |
| *input_fitsimage* | string | | Directory containing sky models in FITS image format or ASCII/Tigg |
| *input_fitspol* | bool | | Indicate that the FITS sky models are polarised. Used only when *input* |
| *input_changroups* | int | | The number of groups into which the total number of frequency chann |
| *output_to_logfile* | bool | | Write output messages to logfile instead of to the terminal. |
| *add_thermal_noise* | bool | | Add baseline-dependent thermal noise, calculated using station SEFDs |
| *exportuvfits* | bool | | Export MS to UVFITS format. |
| *station_info* | string | | Name of the file containing individual station information such as SEF |
| *bandpass_enabled* | bool | | Add complex bandpass corruptions. |
| *bandpass_table* | string | | Name of the ASCII file containing bandpass gain amplitudes for each s |

| Parameter | Type | Units | Explanation |
|---|---|---|---|
| *bandpass_freq_interp_order* | int | | Order of spline interpolation. Integer between 1 and 5. |
| *bandpass_makeplots* | bool | | Generate bandpass plots. |
| *elevation_limit* | double | *radians* | Flag visibilities below this elevation limit. |
| *corr_quantbits* | int | | Number of bits used for quantisation by the correlator (e.g. 2 for four l |
| *predict_oversampling* | int | | Oversampling factor to improve the accuracy of forward modelling wi |
| *predict_seed* | int | | Seed for random number generation with numpy. Set to -1 will disable |
| *ms_antenna_table* | string | | Name of *CASA* ANTENNA table to use for creating the MS, with abso |
| *ms_datacolumn* | string | | Name of the MS column to write the output visibilities to. Commonly |
| *ms_RA* | double | *degrees* | Right Ascension of the pointing centre of the observation. |
| *ms_DEC* | double | *degrees* | Declination of the pointing centre of the observation. |
| *ms_polproducts* | string | | Specify the polarisation feed type in *CASA* recognisable format (e.g. 'I |
| *ms_nu* | double | *GHz* | Centre frequency of the bandpass. |
| *ms_dnu* | double | *GHz* | Bandwidth of the spectral window. |
| *ms_nchan* | int | | Number of channels. |
| *ms_obslength* | double | *hours* | Duration of the observation. |
| *ms_tint* | double | *seconds* | Integration time. |
| *ms_StartTime* | string | | Starting time of the observation (e.g. 'UTC,2017/04/01/00:00:00.00'). |
| *ms_nscan* | int | | Number of scans in the observation. |
| *ms_scan_lag* | double | *hours* | DEPRECATED. Left intact for backward compatibility. |
| *ms_makeplots* | bool | | Generate plots of the data such as uv-coverage and uv-distance sensitiv |
| *ms_correctCASAoffset* | bool | | In a two-step process, correct for the spurious offset introduced by CA |
| *make_image* | bool | | Make dirty image using lwimager. The *im_* parameters are used only w |
| *im_cellsize* | multi | | Cell size to be used for imaging with units (e.g. '3e-6arcsec'). |
| *im_npix* | int | *pixels* | Image size. |
| *im_stokes* | string | | Stokes parameter to image. Allowed values are 'I', 'Q', 'U', or 'V'. |
| *im_weight* | string | | Weighting scheme to use for imaging. Allowed values are 'uniform', ' |
| *trop_enabled* | bool | | Enable corruptions by the troposphere. The other *trop_* parameters are |
| *trop_wetonly* | bool | | Simulate only the wet component (i.e. the component due to water vap |
| *trop_attenuate* | bool | | Enable attenuation by the troposphere. |
| *trop_noise* | bool | | Include sky noise from the troposphere. |
| *trop_turbulence* | bool | | Add Kolmogorov turbulence to the simulated visibility phases. |
| *trop_mean_delay* | bool | | Add mean (i.e. non-turbulent) delays due to the mean tropospheric cor |
| *trop_percentage_calibration_error* | float | | DEPRECATED. Left intact for backward compatibility. |
| *trop_fixdelays* | bool | | Insert time-invariant delays computed by taking the mean over the spe |
| *trop_fixdelay_max_picosec* | int | *picoseconds* | DEPRECATED. Maximum absolute value of the constant delays gene |
| *trop_makeplots* | bool | | Generate troposphere-related plots such as zenith opacity, elevation-de |
| *pointing_enabled* | bool | | Enable pointing errors. The other *pointing_* parameters are used only v |
| *pointing_time_per_mispoint* | float | *minutes* | Generate new pointing error per station every this minute. |
| *pointing_makeplots* | bool | | Generate pointing offset-related plots. |
| *uvjones_g_on* | bool | | Add time-varying station-based complex gains (G-Jones). The per-stat |
| *uvjones_d_on* | bool | | Add instrumental polarisation. Polarisation leakage (D-Jones) and para |
| *parang_corrected* | bool | | Indicate if the correction for parallactic angle rotation has already beer |

**Note:** Ensure that *ms_nchan* != 1 when *trop_enabled* = True. AATM may fail while when there is only one frequency channel present in the MS. If only one (e.g. averaged) frequency channel is desired, the tropospheric corruptions must be calculated at a higher frequency resolution and the channels manually averaged at the end of the simulation run.

## 4.2 Sky models

The parameter *input_fitsimage* points to sky models in one of three formats that are recognisable by MeqSilhouette.

### 4.2.1 FITS format

Sky models in FITS format are forward-modelled using *WSClean* under the hood. The directory pointed to by *input_fitsimage* must contain all FITS files that constitute the sky model, named according to the following convention:

- If there is no time-variability or polarisation, then *input_fitsimage* contains only one FITS image named *t0000-model.fits*.
- If the sky model is time-variable, the FITS files named *txxxx-model.fits*, where xxxx=0000, 0001, ... . The total number of unique times in the MS are divided evenly into N groups, where N is the number of times for which FITS files are present.
- If the sky model is polarised, the FITS images are named *txxxx-[IQUV]-model.fits*, representing each Stokes component [I, Q, U, V]. All Stokes components must be present for each time and frequency.
- If the sky model is frequency-variable, the FITS files are named *t0000-yyyy-model.fits*, where yyyy=0000, 0001, ... . The number of frequencies must be equal to *input_changroups*.

Following *WSClean*, MeqSilhouette does not care about the actual frequencies in the FITS headers. This means that the input channels in the MS will be divided evenly into *input_changroups* groups and each FITS image along the frequency axis will be used to predict visibilities into the appropriate group, regardless of the frequencies in the FITS headers.

Putting all of the above together, a time and frequency varible polarised sky model will consist of a series of FITS files named *txxxx-yyyy-[I,Q,U,V]-model.fits*, where xxxx=0000, 0001, .... (as many as needed to replicate intrinsic source variability) and yyyy=0000, 0001, .... (must be equal to *input_changroups*).

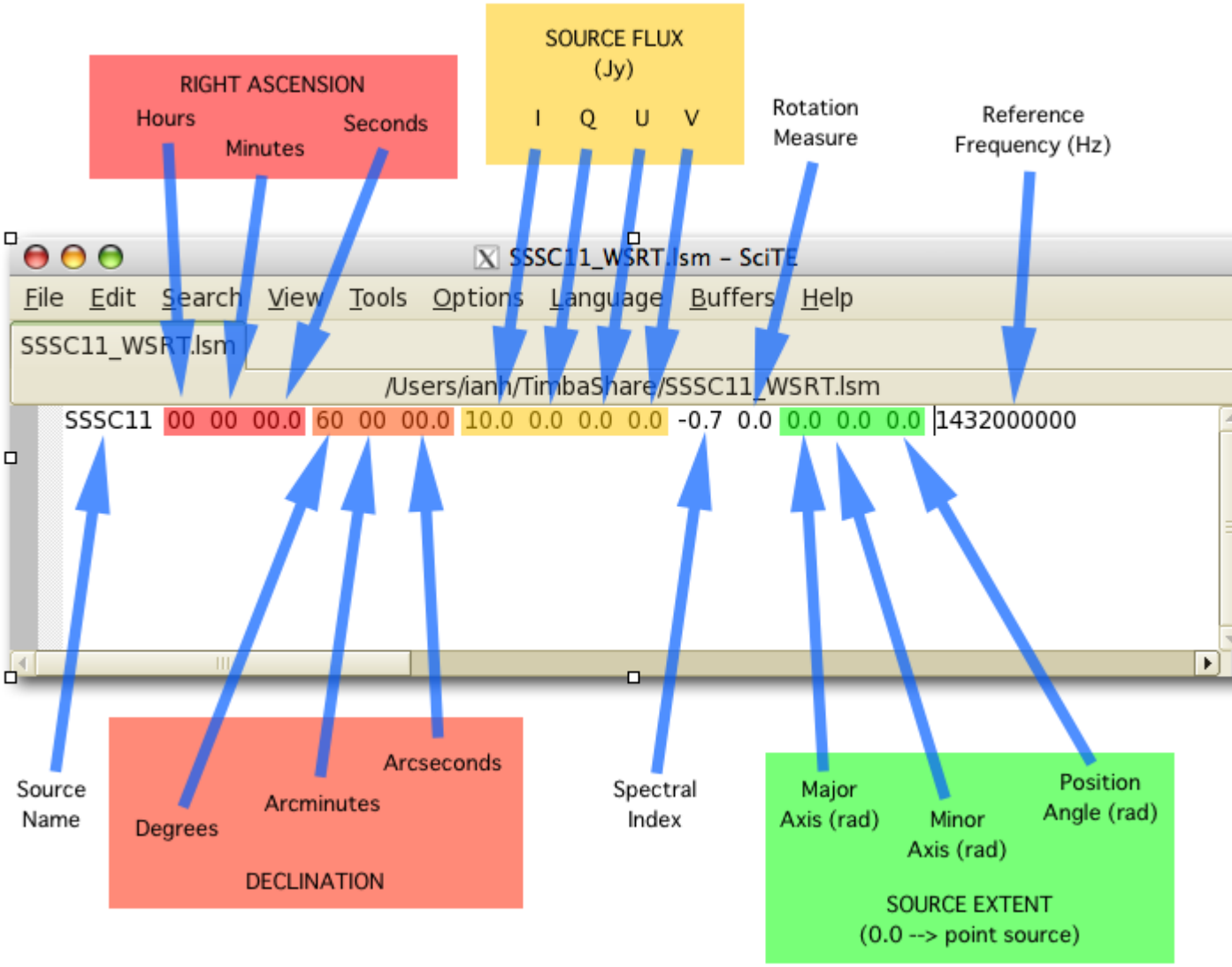
---

**Note:** *WSClean* can predict visibilities only into the MODEL_DATA column. MeqSilhouette will copy them into *ms_datacolumn*, after which the signal corruptions are applied only to *ms_datacolumn*. Hence, the uncorrupted visibilities are available in MODEL_DATA column for inspection. If *MeqTrees* is used instead, the uncorrupted visibilities from *ms_datacolumn* are copied to MODEL_DATA.

---

### 4.2.2 ASCII / Tigger LSM format

The ASCII / Tigger LSM file with extensions *.txt* / *.lsm.html* respectively, are sky model files recognisable by the *tigger-lsm* module used by *MeqTrees*. This file contains a list of sources, with each row corresponding to one source. The columns are as shown in the figure below:




**Note:** It is recommended to use FITS images as inputs (in which case *WSClean* is used for predicting visibilities) for EHT simulations. MeqTrees has been observed to occasionally give rise to precision errors of up to ~1 micro-arcsecond. Hence, when using ASCII / Tigger LSM files, additional sanity checks must be performed to ensure that the source positions are not offset from the expected values. This is an outstanding issue with MeqTrees and will be resolved in a future version.

## 4.3 Station and site information

The parameter *station_info* points to an ASCII file containing additional information about the participating stations and the site weather parameters for generating the Jones matrices for corrupting the visibilities. Each row corresponds to one station. The details of each column are given below.

Table 2: Station and site information

| Column | Type | Units | Explanation |
|---|---|---|---|
| station | string | | Station name or code. |
| sefd | float | Jansky | System Equivalent Flux Density. |
| pwv | float | millimetres | Precipitable water vapour. |
| gpress | float | millibar | Ground pressure at site. |
| gtemp | float | Kelvin | Ground temperature at site. |
| c_time | float | seconds | Tropospheric coherence time. |
| ptg_rms | float | arcseconds | RMS error in pointing. |
| PB_FWHM230 | float | arcseconds | Full Width at Half-Maximum of the primary beam. |
| PB_model | string | | Geometric model to be used for the primary beam ('gaussian' and 'cos3' are available; hardwired to *gaussian* for now). |
| ap_eff | float | | Aperture efficiency. |
| g[RL]_mean, g[RL]_std | float | | Mean and standard deviation of the normal distribution from which to draw time-varying real/imag parts of the G-Jones terms for R and L feeds. |
| d[RL]_mean, d[RL]_std | float | | Mean and standard deviation of the normal distribution from which to draw frequency-varying real/imag parts of the D-Jones terms for R and L feeds. |
| feed_angle | float | degrees | Initial feed angle offset. |
| mount | string | | Mount type of each station. Valid values are 'ALT-AZ', 'ALT-AZ+NASMYTH-R', 'ALT-AZ+NASMYTH-L'. |

# CHAPTER 5

## Example input JSON file

Examples for all input types can be found in the source code. The following is an example JSON file that simulates a polarised sky model and corrupts the complex visibilities with SEFD-based thermal noise.

```
{
 "outdirname":"EHTsim",
 "input_fitsimage":"old_grmhd_pol",
 "input_fitspol":1,
 "input_changroups":1,
 "output_to_logfile":0,
 "add_thermal_noise":1,
 "make_image":0,
 "exportuvfits":0,
 "station_info":"eht_betterweather.antennas",
 "bandpass_enabled":0,
 "bandpass_table":"eht_bandpass.txt",
 "bandpass_freq_interp_order":1,
 "bandpass_makeplots": 0,
 "elevation_limit":0.174,
 "corr_quantbits":2,
 "predict_oversampling":8191,
 "predict_seed":42,
 "ms_antenna_table":"ANTENNA_EHT2017",
 "ms_datacolumn":"DATA",
 "ms_RA":187.70591666666667,
 "ms_DEC":12.391122222222222,
 "ms _polproducts":"RR RL LR LL",
 "ms_nu":228,
 "ms_dnu":2,
 "ms_nchan":64,
 "ms_obslength":4,
 "ms_tint":10,
 "ms_StartTime":"UTC,2017/04/11/00:32:00.00",
 "ms_nscan":1,
 "ms_scan_lag":0,
```

(continues on next page)

(continued from previous page)

```
 "ms_makeplots": 1,
 "ms_correctCASAoffset":1,
 "im_cellsize":"3e-6arcsec",
 "im_npix":64,
 "im_stokes":"I",
 "im_weight":"uniform",
 "trop_enabled":0,
 "trop_wetonly":0,
 "trop_attenuate":1,
 "trop_noise":1,
 "trop_turbulence":1,
 "trop_mean_delay":1,
 "trop_percentage_calibration_error":100,
 "trop_fixdelays":0,
 "trop_fixdelay_max_picosec": 0,
 "trop_makeplots": 0,
 "pointing_enabled":0,
 "pointing_time_per_mispoint": 10,
 "pointing_makeplots": 0,
 "uvjones_g_on": 0,
 "uvjones_d_on": 0,
 "parang_corrected": 1
}
```

CHAPTER 6

# Outputs

The primary output product of MeqSilhouette is a CASA Measurement Set containing the complex visibilities, with all the user-requested corruptions applied to the data. The Measurement Set v2 specification can be found here. The tables and subtables in the MS are filled as follows:

- The value of *ms_datacolumn* in the input JSON file must correspond to an existing column in the MAIN table. This column is filled with the corrupted complex visibilities.
- The MODEL_DATA column is filled with uncorrupted visibilities. It is recommended not to set *ms_datacolumn='MODEL_DATA'* so that the uncorrupted visibilities are available to the user for later inspection.
- The SIGMA column is filled with the standard deviation of the baseline-based complex visibilities computed using the SEFD. If mulltiple frequency channels are present, then SIGMA_SPECTRUM is also filled with these values.
- The WEIGHT column is filled with inverse-variance weighting with the variance computed using the SIGMA values. If mulltiple frequency channels are present, then WEIGHT_SPECTRUM is also filled with these values.
- The ANTENNA subtable is the same as the input antenna table pointed to by *ms_antenna_table*.

Optionally, there a few other outputs that MeqSilhouette can generate for recording the synthetic data generation process and verifying the contents of the MS.

- The numerical values of all the Jones matrices are saved as numpy arrays.
- A number of plots illustrating the properties of the complex visibilities and the various effects applied to them.
- A preliminary image of the simulated data.
- Export the MS into UVFITS format, for compatibility with other calibration packages such as eht-imaging and AIPS.

# CHAPTER 7

# RIME Components

The various components of the Radio Interferometer Measurement Equation (RIME) (Smirnov 2011, and references therein) are implemented in MeqSilhouette. The generic RIME is given by

$$V_{pq} = G_p \left( \sum_s E_{ps} K_{ps} B_s K_{qs}^H E_{qs}^H \right) G_q^H$$

where for source $s$ and antenna $p$, $G_p$ and $E_{ps}$ represent the direction-independent effects (DIEs) and direction-dependent effects (DDEs) respectively, $K_{ps}$ represents the scalar phase delay matrix, and $B_s$ represents the brightness matrix.

For more details, refer to Natarajan et al., (in prep).

# CHAPTER 8

# Integration with pipelines

MeqSilhouette can be used in tandem with other radio astronomy simulation/calibration software packages in software pipelines for data synthesis/reduction/analysis.

The *SYnthetic Measurement creator for long Baseline Arrays* (SYMBA) pipeline (Roelofs et al., 2019) uses MeqSilhouette for synthetic data generation for the Event Horizon Telescope (EHT).

CHAPTER 9

# Contributors

## 9.1 Code

- Iniyan Natarajan
- Roger Deane
- Freek Roelofs
- Michael Janssen
- Tariq Blecher (MeqSilhouette v1)

## 9.2 Containerisation

- Iniyan Natarajan
- Robin Hall

# CHAPTER 10

# History

## 10.1 2.7.1 (2021)

- Update how plotting modules handle non-existent arrays
- New tropospheric turbulence module

## 10.2 2.7 (2021)

- Package MeqSilhouette
- Update singularity recipe and Dockerfile
- Update casa data while building images
- Update documentation

## 10.3 2.6.2 (2021)

- Extensive updates to documentation
- Singularity containerisation
- Add option to run with Jupyter notebook
- Add license
- Update ms plotting module
- Ensure uncorrupted vis are copied to MODEL_DATA always

## 10.4 2.6.1 (2021)

- Improve output path handling
- Synchronize sample input files and default input settings
- Update documentation

## 10.5 2.6 (2021)

- Implement frequency-dependent polarization leakage and remove time dependence
- Improve error handling for memory errors
- Chunk data to fit in memory
- Add paper-friendly plots

## 10.6 2.5 (2020)

- Generate real and imaginary parts of orthogonal polarization feeds independently for time-varying antenna gains and polarization leakage
- Generate interpolated bandpass gains independently for orthogonal polarization feeds
- Clean up input files

## 10.7 2.4 (2020)

- Handle frequency-dependent source models
- Verify existence of bandpass gains table
- Add new bandpass plotting capability
- Implement time-varying complex antenna gains
- Remove deprecated functions

## 10.8 2.3 (2020)

- Streamline random seed initialization
- Handle potential rounding errors in antenna pointing offsets
- Add sky noise to visibility weight estimation
- Include CASA time offset correction

## 10.9 2.0 (2019)

- Depend mainly on WSClean for forward modelling (MeqTrees only for txt sky models)
- Full polarimetric simulations
- Simulate time-variable sources
- Add complex bandpass gains
- Add instrumental polarization and parallactic angle rotation (write visibilities in both antenna and sky frames)
- Improve pointing error module
- Improve tropospheric corruption and thermal noise modules
- Add plotting modules
- Remove scattering screen
- Refactor code for seamless integration within pipelines such as SYMBA

## 10.10 1.0 (2016)

- Tropospheric corruptions
- Basic pointing error module
- ISM scattering

CHAPTER 11

# Indices and tables

- genindex
- modindex
- search