Chapter 1

Software implementation

1.1 Design objectives

Our primary aim is to test and research mm-VLBI calibration, imaging and parameter estimation algorithms/strategies through the construction of a synthetic data simulation framework. To address the many questions within the wide scope of this objective, one must be able to setup and run a diverse set of experiments within the simulation framework. This places definite constraints on the software architechure. In particular, the framework should :

- enable the implementation of the dominant classes of signal corruption within a formalism which ensures consistency with the causal signal transmission chain,
- be compatible with time-variable GRMHD source models which are to be used as inputs,
- be organised in a modular structure so that it is flexible, extendable and could be incorporated into other interferometric algorithms or vice-versa (e.g. a calibration or a parameter estimation algorithm),

- the modular structure should also enable the construction and execution of arbitrary observations.

1.2 Architechure and Workflow

In this section, we will review how the architechural design and workflow of the simulator has been designed to meet the above objectives. To fulfill the first objective, we cast signal corruptions in the RIME formalism (see section **??**). This is not currently possible for the case of ISM-scattering, however we do apply it in the casually correct position in the signal transmission chain, with proper consideration given to non-communitivity of elements. The implementation of each signal corruption is described in the following subsections. The remaining objectives fall into the realm of software design and will be discussed in this subsection.

We have chosen to write the high level simulation code using the PYTHON language. PYTHON is a general purpose language, is geared towards readability, and is well supported by a comprehensive library and wide user base, particularly within astronomy. Even though the higher level functionality is written in PYTHON, the bulk of the computational load is called through subroutines which are written in the more computationally efficient C++ language.

We use the interferometric toolbox, MEQTREES (**?**), which can simulate and calibrate radio interferometic data through evaluation of a user-defined RIME. The evaluation draws on directed-acyclic-graphs (or compute trees) where each element of the graph can process a multi-dimensional grid of values (typically specified along axes of ν and t). Although we have decided to construct a large portion of our RIME outside of MEQTREES, we call prebuilt MEQTREES RIMEs to augment our pipeline in two ways. Firstly to perform the Fourier transform to generate visibilities from the input sky model (FFT for fits files; DFT for parametric sources), where degridding

and interpolation steps are also called when the FFT is used. The second is to simulate the primary beam and antenna pointing error, which will be discussed later in this chapter. On a side note, use of MeqTrees and measurement set also lends itself to investigating a range of different techniques that are used in other areas of interferometry (e.g. **?**).

Our data formats of choice are: fits for image cubes and the measurement set[1] (ms) for visibilities. We use ms as our data format as it is directly accessible in Python via the pyrap library and is the data format used by MeqTrees. Although in the mm-VLBI subfield, other data formats are currently still more popular than the ms, i.e. fitsidi, uvfits or iofits, largely due to the lack of a fringe-fitter in casa and incomplete meta-data fields. However, thanks to development at the Joint Institute for VLBI in Europe (JIVE), we expect this to change in casa v5.

To create a flexible and modular structure necessary to be able to run a diversity of experiments, the software implementation is divided into 2 components:

- an object-oriented framework into which is programmed the logic of each individual step in the signal propagation chain,

- a driver script which initialises the most abstract class in the framework with the required inputs and determines the signal propagation chain relevant to that particular pipeline.

The conceptual flow diagram of one realisation of a meqsilhouette simulation pipeline is shown in Fig. 1.1. To emphasise, the framework is not restricted to this sequence of operations, allowing the exact pipeline to be quite general. Note that this diagram is not entirely accurate as antenna pointing errors are not compatible with the fits sky model output by the ISM scattering module.

[1] https://casa.nrao.edu/Memos/229.html

All inputs to the simulator are specified by a configuration file (.json text file), containing a dictionary (see table 1.1 as an example), which is the sole input to the driver script. This dictionary contains everything needed by the pipeline to determine the particular observation configuration (frequency, bandwidth, start time, etc), which signal corruption implementation should be employed and where the sky model, station weather information and antenna table are located in the filesystem. The antenna table is in the MS format, and can readily be created or altered using the PYRAP library using the station coordinates. The primary sky model used is a time-ordered list of FITS images, where each image represents the source total intensity over a time interval $\Delta t_{\rm src} = t_{\rm obs}/N_{\rm src}$, where $t_{\rm obs}$ is the observation length and $N_{\rm src}$ is the number of source images (**?**). Currently the pipeline only supports total intensity and the conversion of the pipeline to support full Stokes is discussed in section **??**. A variation of the pipeline has also been written which uses a parametric source model consisting of Gaussians or point sources as the sky model. This functionality was needed for the simulation of pointing errors as the MEQTREES beams model does not support the FITS sky model.

Table 1.1: **Top section:** keywords of the input configuration dictionary used in a standard simulation similar to that depicted in Fig. 1.1 but without antenna pointing errors. The variable names have been expanded for clarity and keywords which have boolean values have written as questions. In practice, most of these keywords are predeced by an index, which allows this dictionary to be filtered into several sub-dictionaries which can be easily passed to functions or classes. **Middle section:** the list of the parameters, aside from the source model, needed to initialise and run SCATTERBRANE. These parameters are within the main dictionary but have been separated here for comprehension. Time variability is made possible as $N_{\rm pix}$ can be a 2-tuple (i.e. a rectangular screen can be created). **Bottom section:** the headings of station weather table whose path is indicated in the input dictionary.

parameter	comment
name of output folder	to keep track of simulations

input fits folder path	folder containing intrinsic sky model in FITS format
antenna table path	CASA format
integration time (seconds)	time averaging interval for each visibility data point
start time	time coordinate system (e.g. UTC) and the observation start date and time
channel width (GHz)	
number of channels	Note that currently MEQSILHOUETTE only simulates one spectral window
number of scans	splits observation for into different scans of equal length
scan lag (hours)	lag time between scans
elevation limit (radians)	limit below which all data is automatically flagged
central observing frequency (GHz)	
total observation length (hours)	
apply atmospheric attentuation?	Uses weather table data.
add thermal noise?	calculated with station SEFD
apply atmospheric thermal noise?	Uses weather table data. Derived from atmospheric brightness temperature
apply turbulent tropospheric delays?	Based on Kolmogorov screen statistics.
apply mean atmospheric time delays?	Calculated through radiative transfer.
percentage tropospheric phase	alter the 'amount' of turbulence applied
output image pixel size	Specified as a string in [value][unit] format e.g. '1e-6arcsec'
output image N_{pix}	
station information file path	station dependent details, headings of this table shown in bottom section
apply ISM scattering?	

number of ISM screen realisations	
ISM screen speed	will come into effect only for multiple ISM screens
ISM N_{pix}	size of screen for one sky model
r_0 (km)	coherence length on the scattering scattering screen
N_{pix}	pixel number for one realisation of the screen
r_{in} (km)	inner scale of turbulence
principal angle of Gaussian scattering kernel	
anisotropy of Gaussian scattering kernel	
r_{out}	Outer scale of turbulence
d (pc)	Observer-Source distance
r (pc)	Screen-Source distance
R	
β	turbulent exponent
screen resolution (units of r_0)	
station name	ensure that the order of stations matches the order in the antenna table
average atmospheric coherence time (seconds)	
sefd	
mean pwv content	
ground pressure (mbars)	
ground temperature (K)	

The primary outputs of the pipeline are an interferometric dataset in MS format along with the closure phases and uncertainties, a dirty and/or deconvolved image, and tropospheric phase delays, opacities etc. Closure phases

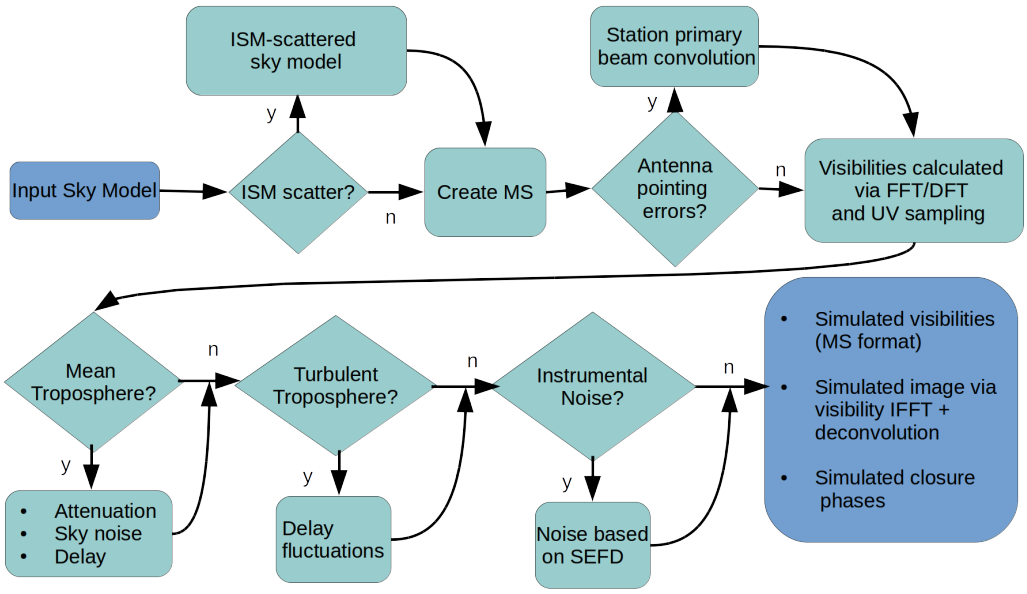

Figure 1.1: Flow diagram showing basic sequence of a MEQSILHOUETTE simulation pipeline. The specific sequence is determined by the driver script whereas the logic of each step is contained in an object-oriented framework. The details of the station information, observation strategy, tropospheric and ISM conditions are specified in a user-defined input configuration file. The pipeline is extendable, allowing any additional, arbitrary Jones matrices to be incorporated.

and uncertainties are calculated in a model-dependent way as described by section **??**. The modular structure of the pipeline allows for additional imaging and deconvolution algorithms to be easily appended to the final data processing steps. Noting that there are other data formats widely used in mm-VLBI, we make use of the CASA task for conversion to UVFITS. Similiarly other data products can be easily produced as needed e.g. polarisation ratios.

An important step to reproduce realistic observations is to be able create a comprehensive MS with arbitrary scan lengths, start times, channel and

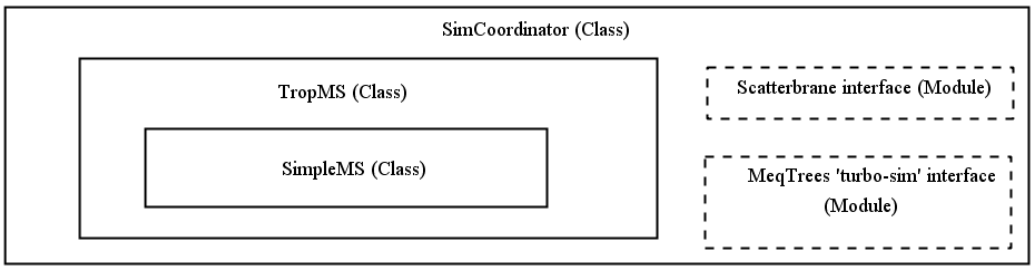

Figure 1.2: The basic class and module structure of the MEQSILHOUETTE implementation. Solid lines represent a class while dashed lines represent a module. The *SimCoordinator* class is initialised in the driver script.

bandwidth structure. This is performed using the SIMMS[2] tool. SIMMS provides an easy to use command line interface to construct a general MS, given the appropriate antenna table. The call to SIMMS is located within the driver script.

In order to make the framework as clean and modular as possible we have made extensive use of object orientation, see Fig. 1.2 for a basic overview. The first major class, *SimpleMS*, was intended to abstract and modularise the MS and MS-only derived attributes (e.g. visibility data and station positions) and methods (e.g. functions to calculate station elevations and closure phases) as well as expose these attributes and methods more efficiently than following PYRAP procedures which become verbose when used frequently. This is especially useful when accessing baseline-indexed quantities.

The second MS-related class, *TropMS*, handles the calculations relevant to tropospheric and thermal noise corruptions. This class is a child of *SimpleMS* and is initialised with weather and station information. Note that a child contains all the methods and attributes of its parent. This allows the tropospheric corruption implementation to use, whilst being separated from, the core MS functionality. The details of the tropospheric corruption are provided in section 1.2.2.

[2]https://github.com/radio-astro/simms

The third MS-related class, *SimCoordinator*, is a child of the *TropMS* class. *SimCoordinator* is designed to make a simulation, with an arbitrary experimental setup, easy and efficient to construct and execute on a high level. It is the only MS class directly initialised in the driver script and hence the low level functionality and attributes of its parents are abstracted from the user. In addition to inherited functionality, *SimCoordinator* can call the ISM-scattering task (see subsection 1.2.1), and MeqTrees simulation functionality.

1.2.1 ISM scattering

As described in section **??**, observations of Sgr A* at (sub)mm wavelengths are subject to ISM scattering in the strong scattering regime. Due to the solid angle of Sgr A* at mm-wavelengths, a single epoch observation of the scattering screen is further defined as falling into the *average regime*, wherein diffractive scintillation is averaged out but refractive scintillation is still present. As mm-VLBI observations can resolve the scatter-broadened image of Sgr A*, an implementation of scattering is needed which approximates the subtle changes in its extended source structure. Such an approximation has been implemented in the Python-based ScatterBrane[3] package, and is based on **?**. In this algorithm a phase screen is created based on the two dimensional spatial power spectrum (see **?**, Appendix C) which incorporates inner and outer turbulent lengths scales. With the screen generated, the original image is scattered according to equation **??**. In practice equation **??** is implemented using an interpolation function which is modified by the values on the phase screen. ScatterBrane allows variation of all parameters (see table **??**) associated with the scattering screen, which is essential as aspects of the scattering towards the Galactic Centre are still unconstrained (e.g. **?**).

We include the ScatterBrane software, which has already yielded important context for mm-VLBI observations towards Sgr A* (e.g. **?**), within

[3]http://krosenfeld.github.io/scatterbrane

the MEQSILHOUETTE framework. Our ISM module interfaces the SCATTERBRANE code within an interferometric simulation pipeline. This module enables simultaneous use of time-variable ISM scattering and time-variable intrinsic source structure within a single framework. The user is able to select a range of options relating to the time-resolution and epoch interpolation/averaging of both. By default, if the time resolution chosen to sample the source variability $\Delta t_{\rm src}$ and screen variability $\Delta t_{\rm ism}$ are unequal, we set

- $\Delta t_{\rm ism} = \Delta t_{\rm src}$ if $\Delta t_{\rm src} < \Delta t_{\rm ism}$
- $\Delta t_{\rm ism} = {\rm R}(\frac{\Delta t_{\rm src}}{\Delta t_{\rm ism}})\Delta t_{\rm src}$ if $\Delta t_{\rm src} > \Delta t_{\rm ism}$,

where R rounds the fraction to the nearest integer. This modification to the ISM sampling resolution avoids interpolation between different snapshots of the intrinsic source structure.

1.2.2 Atmospheric corruption simulator

Our focus in this module is to model the three primary, interrelated (see section **??**) observables which are the most relevant to mm-VLBI: turbulence-driven fluctuations in the visibility phase $\delta\phi$; signal attenuation due to the atmospheric opacity τ; and the increase in system temperature due to atmospheric emission at a brightness temperature $T_{\rm atm}$ due to non-zero opacity. Our approach is to model these observables as being separable into mean and turbulent components which are simulated independently (**?**). The mean tropospheric simulation module performs radiative transfer with a detailed model of the electromagnetic spectrum of each atmospheric constituent (i.e. O_2, H_2O, N_2, etc.). The turbulent simulation module uses a scattering formalism to account for the decoherence that results from power-law turbulence.

As described in section **??**, we use the ATM package to perform radiative transfer through the realisation of the mean atmosphere. In order to cal-

culate atmospheric temperature and pressure profiles, ATM is input several station dependent parameters, namely, ground temperature and pressure, PWV depth, water vapour scale height, tropospheric lapse rate and altitude. The lapse rate refers to the linear relation at which temperature decreases with height. Through experimentation, we have found that the first 3 variables most significantly effect the results of the simulation and opt to keep the latter variables at their default values which were set for application to ALMA. The outputs of this procedure are mean values for opacity, time delay and atmospheric brightness temperature at each station towards zenith. Both opacity and time delay are separated into wet (water vapour) and dry (other) components. Furthermore, The time delay is subdivided into dispersive and non-dispersive components. These outputs are calculated for a list of frequency values. Hence, the non-dispersive component of the mean atmosphere is accounted for. We perform this calculation using representative climate conditions taken from the literature. This final step is to account for elevation effects by multiplying by the airmass $1/\sin\theta$. Since all stations have elevation limits of $> 10^{\circ}$, this is a reasonable assumption.

Following from section **??**, we derive a weak scattering formalism to calculate station dependent visibility phase variations which result from observing through a turbulent troposphere. Specifically, we simulate random walks in visibility phase with variance given by equation **??** for each antenna. These phase-time series are combined to form a multiplicative complex gain corruption, with amplitude of unity i.e. a diagonal Jones matrix. In section **??** we explore the effect of the mean and turbulent atmosphere on observables.

This framework would enable the exploration of an arbitrary or typical range of weather conditions on mm-VLBI observations for each unique station.

1.2.3 Antenna pointing error simulator

To simulate pointing errors, we use the implementation built into the MeqTrees package. This functionality includes the capability to convolve station primary beams with the sky model, which is implemented as an E-Jones matrix in the RIME (see section **??**). The beam models available through this function are sinc, Gaussian and the analytic WSRT beam model. The standard beam model which we will make use of is the analytic WSRT beam model (**?**)

$$E(l,m) = \cos^3(C\nu\rho), \qquad \rho \equiv \sqrt{\delta l_p^2 + \delta m_p^2}, \tag{1.1}$$

where C is a constant, with value $C \approx 65$ GHz^{-1} for a dish diameter of 25 m. Note that the power beam EE^H becomes $\cos^6(\rho)$. One drawback of the MeqTrees implementation is that it is incompatible with the FITS format and so we are at present limited to point and Gaussian parameteric sources for the pointing error simulations. However this is not a significant issue as the primary beam should be constant across the synthesised FOV, effectively reducing to a Direction Independent Effect (DIE), and hence source structure is unimportant to pointing error analysis within the mm-VLBI framework.

Furthermore, Meqtrees allows a constant offset or time-variable primary beam, where the time variability can be either a polynomial (up to third order) or a sinusoid. We have opted to use only the sinusoidal variability for simplicity. To simulate stochastic variability i.e. pointing error due to slew between calibrator and source, we use a constant offset which is resampled per user-specified time interval. In section **??** we demonstrate the effect of constant, sinusoidal and stochastic pointing error variablility on the Large Millimeter Telescope (LMT) within the context of the EHT array.

Appendix A

Software documentation

The following appendix serves as additional documentation for the installation and use of the MEQSILHOUETTE simulator. Please reference (**?**) when publishing results based on this code.

A.1 Installation

The MEQSILHOUETTE repository is currently private, however if made public it is maintained here[1]. For bug reports, if public, open an issue on github/submit a pull request, alternatively if private, please contact Tariq Blecher[2] or Roger Deane[3]. Currently, the simulator is running on UBUNTU 14.04 and has not been tested on other operating systems. Software requirements which are being maintained elsewhere and are publicly available:

- SIMMS[4]
- MEQTREES[5]

[1] https://github.com/ratt-ru/MeqSilhouette
[2] tariq.blecher@gmail.com
[3] deane.roger@gmail.com
[4] https://github.com/radio-astro/simms
[5] https://ska-sa.github.io/meqtrees/

- SCATTERBRANE[6]

Note that SIMMS in turn requires CASA, where we have had success with version 4.2.2. Several different routes for the installation of MEQTREES are available[7]. As AATM is not currently being maintained or easily available, we include it within the MEQSILHOUETTE repository.

Once the MEQSILHOUETTE repository has been cloned, either add the framework module to the PYTHONPATH directly or install using pip,

```
$ cd MeqSilhouette/framework/; sudo pip install .
```

A.2 Usage

We will first discuss the simple case of running a simulation with the canonical pre-written driver script. Following this, we will discuss how to write one's own driver script. We also list various miscellaneous notes which one should to be aware of when using MEQSILHOUETTE.

A.2.1 Running a standard simulation

We will focus on the standard 'azishe' pipeline, the central concepts are captured in Fig. 1.1, however antenna pointing errors are not included and the sky model needs to be in FITS format.

To run in the MeqSilhouette repository we pass the driver script and parameter dictionary to PYTHON,

```
$python driver/azishe.py input/parameters.json
```

The content of the parameter dictionary is shown, slightly altered, in table 1.1. The parameters in the dictionary are can be edited directly.

[6] http://krosenfeld.github.io/scatterbrane

[7] https://github.com/ska-sa/meqtrees/wiki/Installation

The primary log is set by 'v.LOG' variable, initialised in the driver script. If the variable is commented out, the logs will display to screen.

The output directory is set by the 'v.OUTPUT' variable, initialised in the driver script. In this directory there will be a variety of files a list and explanation of which is given in table A.1.

Table A.1: List and explanation of files output by a standard simulation. There are several file formats and data types in parenthesis for easier comprehension. (antenna-based) NUMPY arrays have a shape corresponding to indexing (Time, Frequency, Antenna); (MS-structured) NUMPY arrays reflect the data format of the MS (i.e. Row, Frequency, Polarisation); (baseline dictionary) refers to PICKLE dictionaries where the key is (Station 1, Station 2); (triangle dictionary) is same as a (baseline dictionary) except with an extra dimension i.e. (Station 1, Station 2, Station 3).

name	comment
Measurement Set (.MS)	Final simulated Measurement set
Sky models (.fits)	Sky models which were observed
parameters (.json)	input parameter dictionary
(atm/antenna number)atm abs (.txt)	zenith atmospheric opacity and sky brightness temperature output by ATM
(atm/antenna number)atm disp (.txt)	zenith atmospheric delays output by ATM
closure phases (.p)	Dictionary of closure phases where keys are a tuple of station names (triangle dictionary)
closure phase uncertainties (.p)	Dictionary of closure phase uncertainties where keys are a tuple of station names (triangle dictionary)
Thermal noise (.p)	Dictionary of expected thermal noise levels used to generate closure phase uncertainties (baseline dictionary)
SNR (.p)	Dictionary of SNR values (see equation **??**) used to generate closure phase uncertainties (baseline dictionary).

receiver noise (.npy)	thermal noise generated from SEFDs (MS-structured)
sky noise (.npy)	thermal noise generated from atmosphere (MS-structured)
transmission (.npy)	atmosphere transmission (antenna-based)
turbulent phases (.npy)	atmospheric phase terms (antenna-based)
Stokes I (.p)	visibilities in total intensity (baseline dictionary)
phase standard deviations (.npy)	standard deviations of the phase error between two data points at zenith (antenna-based)
turbulent phases (.npy)	turbulent contribution to the atmospheric phase error (antenna-based)
phase normalisation (.npy)	mean contribution to atmospheric phase error (antenna-based)

Further notes and reminders

- Note that if the content of antenna table is changed, so should the content of station information file which contains station $SEFD$ and weather information.
- Ensure that the order of stations matches the order in the antenna table.
- Ensure that the antenna table is complete, including names, positions in metres and dish diameters.
- Ensure that the FITS file is complete with satisfactory header and data shape (see examples in the input dictionary for guidance).
- Currently one spectral window can be simulated at a time.
- Currently only total intensity can be simulated.

- There are several cautions related to usage of ScatterBrane, see original documentation [8].

- The path of the folder containing the sky model is also located in the parameter dictionary, for multiple fits files, i.e. a time-variable source, the order of their implementation is the same as if their names with sorted with numpy.sort, where each fits files is observed for approximately the same amount of time. If the ISM-scattering is turned on, the extra logistics of moving and creating fits files and folders are automatically handled.

A.2.2 Writing a driver script

Often it is useful to perform iterations of subset of steps of the standard pipeline which is straightforward with a *for* loop and several simple functions or clearing and copying measurement sets. A number of different pipelines are available in the driver scripts folder, which should be provide 'worked examples' when writing a novel driver script. The most important steps are:

1. setup parameter dictionary as well as sub-dictionaries
2. create measurement set
3. initialise SimCoordinator class
4. Use SimCoordinator to generate visibilities and apply signal corruptions

Also, it is also possible to save/load from most of the signal corruptions if you needed to ensure that the corruption used is exactly the same in different realisations.

If additional instrumental corruptions are implemented, this should be within the TropMS class for consistency. Also it is important that the pipeline

[8] http://krosenfeld.github.io/scatterbrane/current/

respects the causality of signal transmission path and/or the commutativity of Jones-matricies.