Chapter 1

Software implementation

1.1 Data simulation

1.1.1 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 diversity of experiments within the simulation framework. This places definite constraints on the software architechure. In particular, the framework should

- enable the implementation of all relevant 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,
- enable the construction and execution of arbitrary observations,
- be organised in modularised structure so that it is flexible, extendable and could be incorporated by other interferometric algorithms e.g. a calibration or parameter estimation algorithm.

1.1.2 Architechure and Workflow

In this section, we will review how the architechural design and workflow of the simulator architechure has been designed to meet the above objectives. To fulfill the first objective, we try to cast signal corruptions in the RIME (see section **??**) formalism, and where this is not possible, to fit those particular signal corruptions into the casually correct position in the signal transmission chain. 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 simulator 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 (including astronomers).

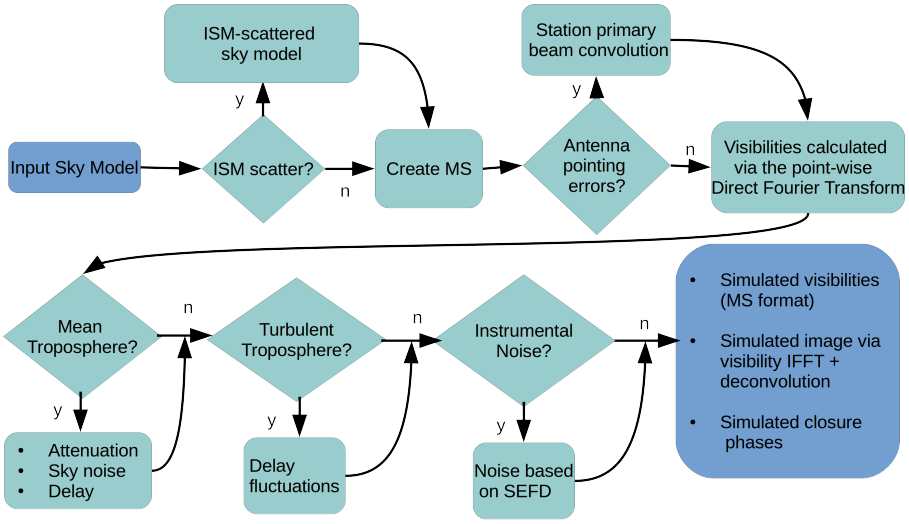

Figure 1.1: Flow diagram showing basic sequence of the MeqSilhouette simulation pipeline. The sky model could include (a) a time-ordered list of FITS images or (b) parametric source model consisting of Gaussians or point sources. The details of the station information, observation strategy, tropospheric and ISM conditions are specified in a user-defined input configuration file. The pipeline is flexible, allowing any additional, arbitrary Jones matrices to be incorporated. Further details in text.

Specifically Python interfaces well with a modern interferometric toolbox, MeqTrees, as well as our data formats of choice: FITS for image cubes and the measurement set[1] (MS) for visibilities. Although the higher level functionality is written in Python, the bulk of the computational load (MS and visibility generation) is called through lower level programs, which are written in the faster C++ language. The MS is our data format of choice as it is directly accessible via the pyrap library and is the format of choice for MeqTrees which is used for computing the RIME. Although in the mm-VLBI subfield other data formats are currently still more popular than the MS i.e. UVFITS and/or HOPS, with the completion of ALMA, the MS format will inevitably become the next generator data format and already is used at the Joint Institute for VLBI in Europe (JIVE).

A conceptual flow diagram of a MeqSilhouette simulation pipeline is shown in Fig. 1.1. The framework is not restricted to this sequence of operations and allows for the exact pipeline to be quite general. This flexibility is made possible through the use of *Object-Orientation*, which will be elaborated on later.

At highest level of the framework is the driver script and parameter dictionary. The driver script determines the major sequence of steps in the pipeline whereas the parameter dictionary specifies all parameters needed by the pipeline to determine the particular observation configuration (array, frequency, band-

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

width, start time, etc), which signal corruption implementation should be employed and where the sky model is located. The sky model is typically a time-ordered list of FITS images, where each image represents the source total intensity[2] 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.

The first major step in the pipeline is to create the MS. This is performed using the SIMMS[3] tool. SIMMS provides an easy to use interface to construct general MS, given an appropriate antenna table. This includes multiple scans, bands and channels.

In order to make the framework as 'clean' and modular as possible we have made extensive use of object orientation. The first major class, *SimpleMS*, was intended to abstract and modularise the MS and derived attributes and methods as well as expose these attributes and methods more efficiently than following PYRAP procedures. Hence this class contains only the original MS and derived attributes, e.g. visibility data and station positions, and methods, e.g. functions to calculate station elevations and closure phases.

The second MS-related class is *TropMS* which handles all troposphere and thermal noise related 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 is provided in a later subsection.

The third MS-related class is the *SimCoordinator*, and it is a child of the *TropMS* class. *SimCoordinator* is designed to make arbitrary simulations easy and efficient to construct and execute. 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 (the implementation of which is in the next subsection), and the evaluation of the RIME. The RIME is evaluated using the MeqTrees : turbo-sim script, where the visibilities are calculated through evaluation of the Fourier Transform at each UVW coordinate in the dataset, the time and frequency resolution of which is specified by the user.

The primary outputs of the pipeline are an interferometric dataset in MEASUREMENT SET format along with the closure phases and uncertainties and a dirty and/or deconvolved image (or spectral cube if desired. The modular structure of the pipeline allows for multiple imaging and deconvolution algorithms to be employed.

1.1.3 ScatterBrane

1.1.4 Atmospheric corruption simulator

1.1.5 Pointing error simulator

1.1.6 RODRIGUES interface

For community use, we host the online, RODRIGUES, interface, found at http://rodrigues.meqtrees.net/. Each of the components of the simulator run

[2]Later versions of MEQSILHOUETTE will enable the full Stokes cubes as input.
[3]https://github.com/radio-astro/simms

in Docker containers. **Looks like the infrustructure is going to change, re: discussions with Gijs and Sphe, so going to wait before writing this.

1.2 Parameter estimation

Bibliography