1. Need some background on MeqTrees and its architechre. Check Oleg/Trienko papers.

2. Diagram on Classes

3. Summary of types of Jones matrices used.

Chapter 3

Software implementation

3.1 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 diversity 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 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,
- be organised in a modular ~~modularised~~ structure so that it is flexible, extendable and could be incorporated into ~~by~~ 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.

3.2 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 formalism (see section 2.1.1). Where this is not possible, i.e. for processes which can not be described with Jones matrix, to fit those signal corruptions into the casually correct position in the signal transmission chain, with proper consider given to non-communitivity of elements in the signal transmission path. 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.

only a scaffolding right?

, particularly within astronomy.

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 (including astronomers). 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 subroutines which are written in ~~the faster~~ C++ ~~language~~. We use MS as our data format as it is directly accessible via the PYRAP library and is the data format used by MEQTREES which performs the visibility generation and pointing error simulation. Although in the mm-VLBI subfield other data formats are currently still more popular than the MS, i.e. UVFITS or IOFITS, but with the completion of ALMA, the MS format should become ~~the next modern data format and is already in use at the Joint Institute for VLBI in Europe (JIVE)~~.

because MeqTrees is written in python...

more computationally efficient

fitsidi

largely due to ~~standard set~~, the lack of a fringe fitter in CASA, however this is expected to change for CASA V5 thanks to development at JIVE.

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. 3.1. To emphasise, the framework is not restricted to this sequence of operations, allowing the exact pipeline to be quite general. This flexibility is made possible through extensive Object Orientation.

no -

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

All inputs to the simulator are specified by a configuration file, containing a dictionary, 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 ~~and antenna~~ table are located in the filesystem. The antenna table is in the CASA 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 $\Delta t_{\mathrm{src}} = t_{\mathrm{obs}}/N_{\mathrm{src}}$, where t_{obs} is the observation length and N_{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.

need table of all json file inputs

MS?

The primary outputs of the pipeline are an interferometric dataset in MS format along with the closure phases (including uncertainties) and a dirty and/or deconvolved image. 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 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. 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

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

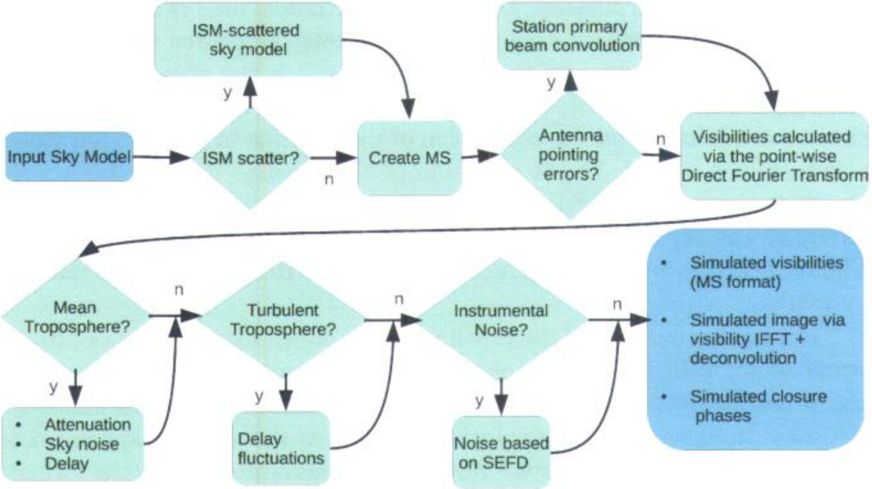

Figure 3.1: Flow diagram showing basic sequence of a MeqSilhouette simulation pipeline. The specific sequence is determined by the driver script whereas the logic of each step is contained in an object-oriented framework. The details of the station information, observation strategy, tropospheric and ISM conditions are specified in a user-defined input configuration file. The pipeline is extendable, allowing any additional, arbitrary Jones matrices to be incorporated.

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 is provided in section 3.2.2.

are

an

The third MS-related class, *SimCoordinator*~~, is~~ a child of the *TropMS* class. *SimCoordinator* is designed to make arbitrary simulations 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 3.2.1), and MEQTREES simulation functionality. Specifically within MEQTREES we make use of the *turbo-sim* script which evaluates the RIME to generate visibilities and to simulate antenna pointing errors(see section 3.2.3), where the visibilities are calculated through direct evaluation of the Fourier Transform at each UVW coordinate in the dataset. MEQTREES allows its tasks to be called by a PYTHON function and so is naturally included in the pipeline.

simple diagram needed ?

3.2.1 ISM scattering

wavelengths

As described in section ?? observations of Sgr A* at sub(mm) is subject to ISM scattering in the strong scattering regime. Due to the size 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 Johnson and Gwinn [2015]. In this algorithm a phase screen is created based on the two dimensional spatial power spectrum [see Johnson and Gwinn, 2015, Appendix C] which incorporates inner and outer turbulent lengths scales. With the screen generated, the original image is scattered according to equation 2.21. In practice equation 2.21 is implemented using an interpolation

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

Table 3.1: The list of the parameters, aside from the source model, needed to initialise and run SCATTERBRANE. Time variability is made possible as N_{pix} can be a 2-tuple (i.e. a rectangular screen can be created).

r_0	N_{pix}
r_{in}	principal angle
r_{out}	anisotropy of scattering kernel
D_{os}	λ
R	β
screen resolution	

function which is modified by the values on the phase screen. SCATTERBRANE allows variation in all parameters (see table 3.2.1) 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. Ortiz-León et al., 2016], within 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 Δt_{src} and screen variability Δt_{ism} are unequal, we set

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

Operator, not var

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.

3.2.2 Atmospheric corruption simulator

Our focus in this module is to model the three primary, interrelated (see section 2.2.3) 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_{atm}. Our approach is to model

due to non-zero opacity

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. The turbulent simulation module takes a scattering approach to account for the decoherence that results from power-law turbulence.

As described in section 2.2.3, we use the ATM package to perform radiative transfer through the realisation of the mean atmosphere. In order to calculate 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. Both opacity and time delay are separated into wet (water) and dry components. These outputs are calculated for a list of frequencies. 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 $1/\sin\theta$.

Following from section 2.2.3, we derive a weak scattering formalism to calculate station dependent visibilities phase variations which result from observing through a turbulent troposphere. Specifically we simulate random walks in visibility phase with variance given by equation 2.37 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.

3.2.3 Pointing error simulator

To simulate pointing errors, we use the implementation built into the MeqTrees turbo-sim task. This functionality includes the capability to convolve station primary beams with the sky model. The beam models available through this function are sinc, Gaussian and the analytic WSRT beam model, however there is not much difference between the different beam models up until the first null. The standard beam model which we will make use is the analytic

quantify & justify

of

WSRT beam model [Popping and Braun, 2008]

$$E(l,m) = \cos^3(C\nu\rho), \qquad \rho = \sqrt{\delta l_p^2 + \delta m_p^2} \tag{3.1}$$

where C is a constant, with value $C \approx 130$ GHz^{-1}. Note that the power beam EE^H becomes $\cos^6$, resulting in a FWHM = 6.5 arcsec at 230 GHz. One drawback of the MeqTrees implementation is that it is incompatible with the FITS format and so we are limited to point and Gaussian parameteric sources for the pointing error simulations. However this is not a significant issue as the pointing error should be constant across the FOV and hence source structure observable with mm-VLBI is unimportant to any pointing error analysis.

within mmVLBI framework ? mentioned

Furthermore "turbo-sim" allows constant offset or time variable primary beam, where the time variability can be either an up-to-third order polynomial or a sinusoidal function. We have opted to incorporate 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, sinusoidally variable and stochastically variable pointing errors on the LMT which is the EHT station with the most narrow beam, and could be used as a referenced station due to its centrality the array.

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

~~more detail~~

more detail