VLBI simulator documentation

Dr. Jack F. Radcliffe

1 Introduction

This simple simulator is designed to allow measurement sets to be created that can replicate VLBI arrays at low frequencies and can include custom arrays. It was borne out of initial simulations to look at the effect of primary beam attenuation on wide-field VLBI arrays but has been now developed as a tool to assist with the production of simple VLBI data sets and mosaics.

2 Installation instructions

2.1 Prerequisites

The majority of the work in getting this package installed is the various packages that are required for it to function. As a result, we recommend using `singularity` as the various packages require some shared environments which can clash with each other. These singularity images are available here.

If you decide not to use singularity or want to build optimised images based on your computer architecture, then the packages you will need are the following:

- `simms` - https://github.com/ratt-ru/simms. This package is used to generate custom empty measurement sets. YOu can also get this installed as part of the `stimela` package.
- `CASA` - https://casa.nrao.edu. We recommend the modular version of CASA as you have more control over the packages that are installed with pip.
- `python v3.10+` - you will need the `pandas`, `astropy`, `numpy`, `scipy` and `matplotlib` packages which you can install using `pip` or `conda` (depending on how you installed python).
- `wsclean` with the `Image Domain Gridder` - https://wsclean.readthedocs.io/en/latest/. This is required if you are needing to simulate the primary beam effects and other direction-dependent effects.

2.2 Installation

Installation should be fairly simple and requires just a git clone of this repository (which you may have already done!). Nothing else is needed!

3 Quick start guide

The simulator is pretty simple to use and just needs to have the two input files (`simulator_inputs.txt` and `simulator_advanced_inputs.txt`) edited to select the array of your choice. You can copy the input files to whatever directory you want as long as they are specified when you run the simulator script to produce the job files. The various inputs in this file and descriptions are described in Section 4. The simulator comprises three different steps that are controlled by a `bash` script per step. To make these scripts, we can use the following syntax on the command line,

```
singularity exec <path_to_casa_image> python <path_to_git_repository>/vlbi_simulator
    .py <path_to_inputs>/simulator_inputs.txt <path_to_inputs>/
    simulator_advanced_inputs.txt <step_number>
```

The step number corresponds to the part of the simulation you want to perform and the number corresponds to the following,

1. **make ms** - generates a measurement set, inputs noise and optionally imports a model.
2. **single pointing** - s image of a single pointing including primary beam attenuation. If a mosaic is not needed, sensitivity maps are made.
3. **mosaic** - produces measurement sets corresponding to a mosaic defined in the input file. Makes images of each measurement set.
4. **make image** - combines the mosaic pointing into a single mosaic and generates sensitivity maps.

Once this script is run, then it will make a file in your current folder called `job_<step_name>.<job_manager>`. You then want to run this via the job submission command. For example, if you are running `make_ms` using Slurm, then you would type the following to run the script,

```
sbatch job_make_ms.slurm
```

After this has been completed, then you can run the next step to make the job script that will be submitted to your job manager and submit that. If you just want to generate a measurement set and include a model.

4 Input files

The simulator takes two input files, the `simulator_inputs.txt` file, which you will need to edit, while the `simulator_advanced_inputs.txt` file is probably ok being left as default (unless you want to make exact, large data size measurement sets). The `simulator_inputs.txt` file comprises a range of parameters that you need to set and the following subsections will describe these.

4.1 Software and paths

- `CASA_exec`, `wsclean_exec`, `stimela_exec`, and `rms_exec` - the full executable commands on how to open CASA, wsclean and stimela/simms. These must be the full command i.e., if you are using singularity then the CASA command in the input file may look something like,

  ```
  CASA_exec = singularity exec /idia/software/containers/casa-6.3.simg python
  ```

- `output_path` - set this to put all outputs into this folder
- `repo_path` - path to the github repository and must include the `vlbi_simulator` part.
- `prefix` - string to append to the start of all output file names. Will be set the 'sim' is left empty.

4.2 HPC options

The simulator is designed to be usable on both your local computer and any high-performance computing architecture which uses Slurm or PBS Pro as its job manager. For the following inputs, leaving a parameter blank means that the parameter will not be specified to the job manager (and may be set as the default value on the cluster).

- `job_manager` - defines the job manager software (options are `slurm` or `pbspro`). If this is set to `bash` then the codes can be run on your local machine. You can also ignore the rest of these inputs if using `bash`.
- `partition` - selects the partition of the cluster to use.

- `walltime` - maximum time requested on the cluster to run the job.
- `nodes` - number of nodes requested.
- `cpus` - number of CPUs per node required.
- `mpiprocs` - number of tasks per node (normally good to set to the same as the number of CPUs).
- `nodetype` - defines the type of node to use (not used in Slurm)
- `max_jobs` - for the steps that use multiple jobs (e.g., the mosaic step), this specifies the maximum number of jobs to be submitted at once. Setting this to `-1` means that there is no maximum limit.
- `mem` - defines the maximum RAM requested to the node (suffix determines the order of magnitude e.g., `50G` requested 50 gigabytes)
- `email_progress` - determines which email the progress notice will be sent to.

4.3 Array and data set up

These inputs determine the data structure of the measurement set and include support for data rates as well as the selection of bandwidths. The inputs are as follows,

- `antennae` - python list of antennae to be included. The antenna codes currently follow the EVN calculator[1] names. The currently supported antennae and their parameters can be found in Appendix A and the data are located in the `master.itrf` and `ant_info.json` files.
- `data_rate` - the data rate of the observation in $\mathrm{Mbit\,s^{-1}}$ that is related to the bandwidth ($\Delta\nu$), bit sampling (N_{bit}) and polarisations (N_{pols}) by data rate $= \Delta\nu \cdot N_{\mathrm{pols}} \cdot N_{\mathrm{bit}} \cdot 2$.
- `bit_sampling` - determines the bit sampling as used in the data rate calculation.
- `bandwidth` - directly input a pre-determined bandwidth in MHz. If this is set, then the `data_rate` parameter will be ignored!
- `npols` - number of polarisation products in the MS. Note that N_{pols} in the data-rate equation will equal 2 regardless of whether cross or/and parallel hands are present.
- `obs_freq` - central observing frequency in GHz.
- `input_model` - sky model file that is to be gridded into the visibilities. Model should be either in a FITS or CASA image format and should be in units of Jy per pixel. If this parameter is left empty, then no model will be inputted.
- `wf_ITRF` - if True, the simulated measurement set will use a VLA B-array position instead of the VLBI geodetic coordinates. Useful for doing rms maps but not for modelling the effect of spatial filtering.

[1] http://old.evlbi.org/cgi-bin/EVNcalc

4.3.1 Observing scheme

These parameters specify how you want your observing scheme to be. It supports a single source and single pointing as well as an optimal mosaic which is determined from the primary beam size.

- `field_centre` - determines the central pointing location or mosaic and should be a two-element list with RA and Dec (e.g., `["12h02m21.6s", "70d11m56.5s"]`). The coordinates should be for the J2000 epoch.
- `total_time_on_source` - total integration time on source in hours. If this is a mosaic, then the time per pointing will be this value divided by the number of pointings.
- `time_multiplier` - A method of approximating large hour integrations through hour-angle averaging. If this is set >1 then the `total_time_on_source` is multiplied by this value but the data-size remains the same.
- `mosaic` - if set to True, then a mosaic is assumed to be the desired outcome.
- `mosaic_area` - two element list with the size of the mosaic sides in degrees. This will assume to be centred on the `field_centre`.
- `mosaic_filling_factor` - for a purely Gaussian beam, the optimal value for this parameter is 1.2. Lower values will cause the mosaic to be oversampled and vice-versa.
- `custom_mosaic` - if this is set, then it needs to define a text file that contains the positions of the mosaics. Each line of this file needs to specify the RA and Dec separated by a space (e.g., `12h00m00s 70d00m00s`)

4.4 Imaging

- `size` - size of the image in pixels. If a quantity is defined then then the cell size will be automatically determined.
- `cell` - angular size of each pixel. If you have set `wf_itrf = True` then this value should be set to around 1.5arcsec as a VLA B-array configuration is used.
- `clean_rms` - makes a clean rms map rather than an rms map produced using box-car smoothing of the map.

A Supported antennae and properties

Antenna	**Code**	**Bands**	**Geodetic coordinates**			**SEFD**	D_{eff}
			x (m)	y (m)	z (m)	(Jy)	(m)
Effelsberg	Ef	L, C	4033947.1955	486990.8687	4900431.0438	19, 20	76

Need to finish this table