RHODES UNIVERSITY
Where leaders learn

MASTERS THESIS

A Systematic Visualisation Framework for Radio-Imaging Pipelines

Author:
Lexy A. L. ANDATI

Supervisor:
Prof. Oleg M. SMIRNOV
Dr. Sphesihle MAKHATHINI

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Centre for Radio Astronomy Technologies and Techniques
Department of Physics and Electronics

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

June 30, 2020

Declaration of Authorship

I, Lexy A. L. Andati, declare that this thesis titled, "A Systematic Visualisation Framework for Radio-Imaging Pipelines" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: L. A. L. Andati

Date: June 30, 2020

RHODES UNIVERSITY

Abstract

Department of Physics and Electronics

Master of Science

A Systematic Visualisation Framework for Radio-Imaging Pipelines

by Lexy A. L. Andati

Pipelines for calibration and imaging of radio interferometric data produce many intermediate images and other data products (gain tables, etc.) These often contain valuable information about the quality of the data and the calibration, and can provide the user with valuable insights, if only visualised in the right way.

However, the deluge of data that we're experiencing with modern instruments means that most of these products are never looked at, and only the final images and data products are examined. Furthermore, the variety of imaging algorithms currently available, and the range of their options, means that very different results can be produced from the same set of original data. Proper understanding of this requires a systematic comparison that can be carried out both by individual users locally, and by the community globally.

We address both problems by developing a systematic visualisation framework based around `Jupyter`[1] notebooks, enriched with interactive plots based on the `Bokeh`[2] and `Datashader`[3] visualisation libraries.

[1] `https://jupyter.org`
[2] `http://bokeh.org`
[3] `http://datashader.org/`

Acknowledgements

I would first and foremost like to thank God for letting me see the submission of this thesis. It has been a long journey which I never thought and never knew I would make, nonetheless this soon. But indeed, He is who He is.

And now to start, I want to thank my supervisors, Prof. Oleg Smirnov and Dr Sphe Makhathini and Javas, for their unwavering support, patience and understanding during this masters. I have a lot to say about you guys but in a nutshell, thank you very much. Thank you for allowing me to grow at my own pace, and giving me the space to wander and learn. By watching you, I have learned a lot about humility and work ethic as a person in addition to the professional knowledge you have imparted. I am very grateful because you changed my perspective on life, and changed my life as well.

A special thanks Prof. Smirnov for giving me a chance to be in the RATT team, and for the opportunities to learn in different environments, to Dr Simon Perkins, Dr Jonathan Kenyon, Prof Ian Heywood, Ben Hugo and many others who have contributed in one way or another (knowingly and unknowingly) to this work. It has been an honour and a privilege. To the entire CARACal development team, thanks for your feedback and suggestions in various ways towards this work, and for the wonderful experiences shared in being part of this team.

I also want to thank Cyndie Russeeawon, who has been so nice to me. Cyndie, if you ever read this, know that I am so grateful for everything and only God can repay you for what you've done for me. You have been more like my family in Grahamstown. #friendship goals!

Thanks to Getty, who has been one of my biggest fans until this point :), I am very blessed and happy to know you. Thanks to my entire immediate family as well for shaping me into who I have become. My friends Chelmis, Vivian who have been a source of encouragement through my adult life and who have grown become more like family.

To the entire RATT team, I didn't know if research is always this fun, but you have made it worth-wile. Ronel, Zizipo, Verushca, thank you for making our lives easier every day, I cannot put in words the magnitude of my gratitude. A special thanks to Ronel for organising my first trip away from home and to Eric and James for welcoming me to Grahamstown and answering my 1Billion questions regarding Rhodes. To those who I have shared working space with Esther, Joel, Robert, Mark, Bokang, Kate, thanks for you neighbourliness and motivation and being a source of joy and relief during difficult times. A big thanks to the Rock Family church for making my life memorable and active in Grahamstown.

Dr. G.O. Okeng'o who made me aware of the RATT bursaries and recommended me for it, I can never thank you enough, God bless you.

Finally, a big thanks to SARAO / NRF for funding towards this project. I appreciate those who encouraged, taught and supported me. This thesis is dedicated to all those who took over my well being in the absence of my father. ...

Contents

List of Figures

List of Tables

List of Abbreviations

AIPS	Astronomical Image Processing System
CASA	Common Astronomy Software Applications
CTDS	Casacore Table Data System
CLI	Command Line Interface
CPU	Central Processing Unit
GUI	Graphical User Interface
MS	MeasurementSet
PSF	Point Spread Function
OS	Operating System
RAM	Random Access Memory
SKA	Square Kilometre Array

In memory of David Andati-Amwayi

Chapter 1

Introduction

The first astronomical instrument was the human eye, sensitive to the so-called optical wavelengths of the electromagnetic spectrum. Eventually, astronomers developed optical telescopes, which are responsive to the visible light wavelength radiation emitted by celestial objects such as stars and galaxies. At the most basic level, these telescopes function by collating incoming light to some focus, where it is then magnified, thus making the target objects visible. For a long time, this "optical window" of the EM spectrum was the only viable window onto the Universe, and while great discoveries were made (and continue to be made) this way, astronomers remained blind to the rest of the EM spectrum.

However, in 1932, a radio engineer by the name Karl G. Jansky accidentally discovered extraterrestrial radio signals from the centre of the Milky Way (Jansky, 1932; Jansky, 1933). Unbeknownst to him at the time, this discovery would open a new window into the observation of the sky and beyond, and birthed the science known as radio astronomy. Henceforth, distant celestial objects could be studied with "unobscured" views, and astronomers could explore more than the visible properties of these objects. Since then, instruments known as radio telescopes (typically consisting of a single parabolic dish, and a receiver responsive to radio wavelength emissions) emerged and have been used to perform astronomical observations. However, it quickly became apparent that single radio telescopes could not provide adequate angular resolution. A technique known as aperture synthesis (see (Burke, Graham-Smith, and Wilkinson, 2019) chap 6), involving the use of multiple dishes to improve the angular resolution of the telescopes, was developed. This technique shall be discussed in Chapter 2.

A number of large aperture synthesis radio telescopes are in operation today, such as Westerbork Synthesis Radio Telescope (WSRT), Very Large Array (VLA), Australia Telescope Compact Array (ATCA), Low Frequency Array (LOFAR), MeerKAT to name just a few. These have gradually become very advanced, complex and fast systems, owing to the recent technological advancements that have resulted in a surge of ubiquitous computing faculties. As a result, massive growth in ingenious applications of these technologies has been evident. This growth can be attributed to the fact that electronic components become smaller and cheaper by each leap as predicted by Moore's law, thus allowing for more complex and compact circuitry. Component miniaturisation translates to increased processing power and consequently, an increased data production rate. The MeerKAT telescope is an example of such application ingenuity. MeerKAT is a precursor to the more powerful and anticipated Square Kilometre Array (SKA) telescope. One of MeerKAT's goals is to facilitate the development and testing of SKA technology and science (Norris, 2010). MeerKAT can generate up to 275 Gigabytes of raw data per second, while its successor, the SKA, is expected to generate up to 160 Terabytes of data per second, an incredibly

massive data load which is currently technically impossible to process in its entirety (Farnes et al., 2018). Such amounts of data undoubtedly fall into the domain of what is known as *Big Data* where meaningful data extraction and analysis is a demanding task.

To further complicate astronomers' lives, data collected using radio synthesis telescopes (known as visibilities) is often tainted by interfering signals from electrical equipment and devices which also emit EM radiation such as communication equipment. This interference is known as radio frequency interference (RFI). In addition, propagation of the signal through the Earth's atmosphere and the antenna electronics introduces its own distortions. Therefore, to make use of the data, various steps must be performed, such as RFI excision or mitigation ("flagging"), as well as calibration and correction for the effects of the signal path ("gains"). The sheer amount of data from modern telescopes makes a manual process impractical. Thus, there have been various efforts to automate the process of extracting science-ready images from raw visibility data through the development of data pipelines. An example of such a pipeline is `CARACal`[1], which shall be discussed at various points of this thesis. However, because of the time and resources consumed in processing extensive data and the need for accurate results, it is essential to monitor the pipeline data products, such as calibration solutions and calibrated visibility data, to ensure that the final data is of good quality.

One way to examine pipeline data products is through graphically visualising them before a final scientific image is produced. While multiple tools can be used to do this - the most notable being the `Plotms` tool in `CASA` (McMullin, 2007), the output generated by these tools takes the form of static plots[2]. This means that:

(i) Data views in the generated plot cannot be modified unless the data is re-plotted,

(ii) It is difficult to embed a lot of information in a plot without obscuring the data,

(iii) To get multiple views of the data, numerous plots must be generated,

(iv) It is difficult to precisely locate sources of outliers on the plot, as well as explore the data.

The goal of this work was to develop a tool, `RAGaVI` (Radio Astronomy Gains and Visibilities Inspector), that generates interactive plots of calibration solutions and visibility data. This is done to enable the embedding of multiple auxiliary data, in addition to the desired plots, which would allow a user to identify and trace problems within the data quickly and promote data exploration. Moreover, the plots do not require any special software to provide interactivity, as they are HTML based. They can thus be generated in batch mode, and examined "offline" using any browser.

1.1 Thesis Structure

The rest of this thesis is sectioned as follows:

Chapter 2: An introduction to the basic concepts of radio astronomy, visibilities, the Radio Interferometer Measurement Equation (RIME) and its relationship to calibration and calibration products, and the importance of pre-imaging data inspection.

[1] `https://github.com/caracal-pipeline/caracal`

[2] Strictly speaking, `Plotms` can be run interactively. However, this mode of operation requires an X11 session, and it is therefore challenging to fit into the batch processing paradigm of a pipeline.

Chapter 3: Here, we describe some tools that were essential in the development of `RAGaVI` as well as some justifications as to why they were chosen. We also briefly describe the structure of the Measurement Set and calibration solutions because they are the main data formats in which the data we visualise are stored.

Chapter 4: We introduce `RAGaVI`, describe its features and implementation.

Chapter 5: We present the results in terms of performance and output layouts of `RAGaVI` in comparison to `CASA Plotms`, and discuss challenges faced during this research.

Chapter 6: We present conclusions and future work.

Chapter 2

Basics of Radio Astronomy

Celestial objects emit various forms of electromagnetic (EM) waves. However, owing to the Earth's atmosphere, only waves in the optical (visible light) and the radio range of the EM spectrum are detectable from the Earth's surface. This is because the other wavelengths of the spectrum are either reflected or absorbed (Condon and Ransom, 2016).

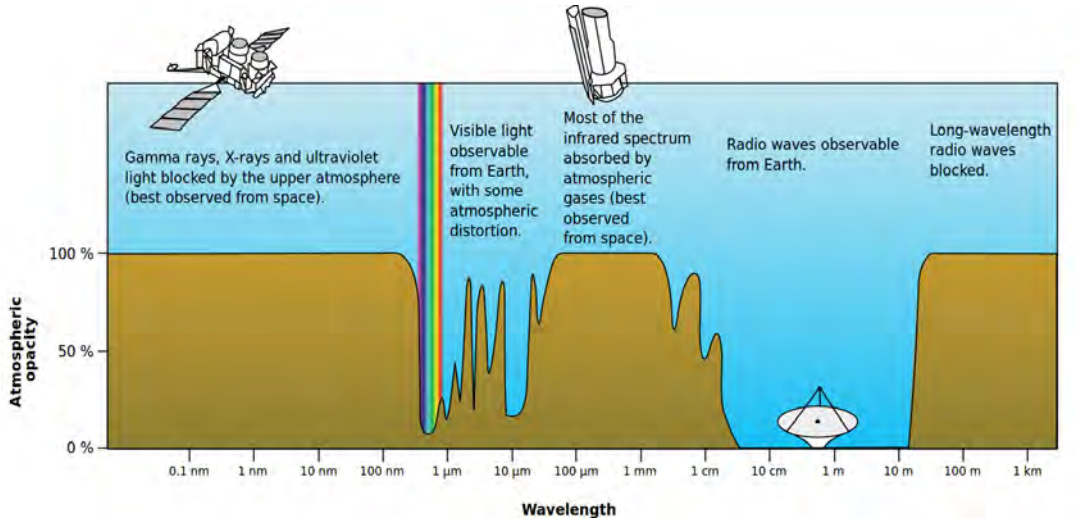


FIGURE 2.1: Frequency window within which we can observe radio waves emanating from celestial objects. Image source: Wikipedia

While optical emissions are useful in the study of celestial objects, they are limited to a relatively narrow band of the EM spectrum. In contrast, radio emissions occur over a larger range of frequencies and can avail astronomers with more information on the properties of celestial objects. Moreover, they are detectable from the Earth's surface using radio telescopes.

Therefore, radio astronomy is the observation and study of celestial objects through the use of radio waves. These waves are characterized by longer wavelengths in comparison to those of visible light, as demonstrated in Fig. 2.1 and hence, tend to be far weaker. Detected radio emissions must first be recorded and processed to produce an image which astronomers can use. The goal of this chapter is to introduce the basic concepts associated with radio signal collection, correction and, ultimately, image generation.

2.1 Single Dish Radio Telescopes

Different kinds of antennas, from simple dipoles to parabolic dishes, may be used to detect radio waves. However, of particular concern to us is the parabolic dish antenna type, which we shall refer to as a single-dish radio telescope. A typical design consists of a single antenna with a parabolically shaped dish known as an *aperture* which collects incoming radiation, and has a diameter D. Due to the aperture's shape, collected radiation is reflected towards a feed horn at the focal point. This feed horn usually houses dual feeds, which sample orthogonal polarisation states of the signal (either linear or circular polarisation) (Cotton, 1999). Parabolic dishes are used because they are highly directional. To detect radio waves from an object in the sky, the telescope must be pointed directly towards that object.

We may characterise a telescope by its *sensitivity*: a measure of the weakest source of radio emission that it can detect (Wrobel and Walker, 1999), and its *angular resolution*: the smallest angular distance that it can distinguish. Using the Rayleigh criterion, the angular resolution, θ, in radians, for a single dish is defined as:

$$\theta \approx 1.22 \times \frac{\lambda}{D}, \tag{2.1}$$

where λ is the observing wavelength, and D is the dish diameter. It is desirable to have a telescope with a smaller value of θ to increase the level of discernible detail of the observed object. Because of the longer wavelengths in the radio regime with respect to the optical regime, Eqn. (2.1) suggests that the resolution of single-dish radio telescopes is very poor.

One way to increase resolution is to increase D. Telescopes such as the Arecibo telescope in Puerto Rico (Roshi et al., 2019) and the Five-hundred-meter Aperture Spherical radio Telescope (FAST) in China undertook this challenge, with FAST currently being the largest single-dish telescope, having $D = 500$ m (Li and Pan, 2016). If we consider the FAST telescope observing at a 6 cm radio wavelength, the resulting resolution will be $\approx 0.503'$. On the other hand, a 100 times smaller optical telescope with $D = 5$ m at a 750 nm optical wavelength would have a resolution of $\theta \approx 0.0006'$. It is clear from this example that, to match the resolution available at optical wavelengths, an unrealistically *enormous* radio dish would be required. Such a large dish would also be susceptible to gravitational deformations and thermal expansion, which will introduce significant errors. It is because of the need for better resolution in radio telescopes that we look to radio interferometry.

2.2 Radio Interferometry

Radio interferometry uses a technique known as *aperture synthesis*. This involves the combination of radio signals collected separately from multiple, small, and distributed single-dish radio telescopes, thus emulating the aperture of a single large telescope known as a *radio interferometer*. The distance of separation between any two single-dish radio telescopes is known as a *baseline* and the size of the synthesized aperture is determined by the longest baseline $\mathrm{b_{max}}$. This baseline thus determines the interferometer's resolution. We can consider $\mathrm{b_{max}}$ to be the equivalent of the dish

diameter D in Eqn. (2.1). The resolution of an interferometer can thus be given by:

$$\theta \sim \frac{\lambda}{\mathrm{b_{max}}}. \tag{2.2}$$

In comparison to the single-dish telescope, an interferometer's resolution can be significantly improved, since long baselines are considerably easier to implement than large dishes.
Given a number of antennas $\mathrm{N_{ant}}$, each pair forms its own baseline. The number of unique baselines $\mathrm{N_b}$ of a radio interferometer is given by:

$$\mathrm{N_b} = \frac{\mathrm{N_{ant}}(\mathrm{N_{ant}} - 1)}{2}. \tag{2.3}$$

To demonstrate the concept of radio interferometry, we will examine an interferometer composed of two antennas, describe what it measures, and how images can be obtained from the measurements made. We will base this section on a summary of (Thompson, 1999).

2.2.1 A Two Element Interferometer

Consider the two-element interferometer shown in Fig. 2.2, which consists of two antennas, p and q, separated by a baseline vector $\mathbf{b}$, both pointing towards a radio-emitting point source e in the direction $\mathbf{s}$.

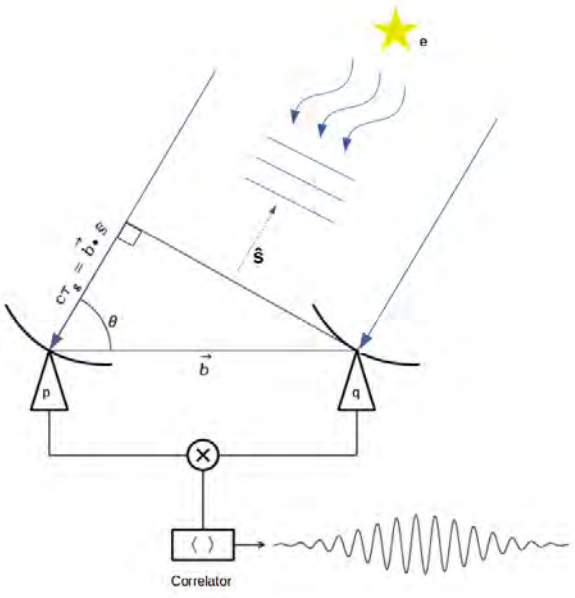


FIGURE 2.2: An interferometer consisting of 2 identical antennas p and q, separated by a baseline vector $\mathbf{b}$ and pointing towards a point source in direction $\mathbf{s}$. This image was adapted from Essential Radio Astronomy.

Because EM waves incident on antenna p take a longer path than those arriving at antenna q, they arrive at antenna p at a time τ_g later than they arrive at q. This

delay is known as *geometric delay* which we can describe as:

$$\tau_g = \frac{\mathbf{b} \cdot \mathbf{s}}{c}, \tag{2.4}$$

where c is the speed of light in a vacuum. Antennas convert the EM waves to voltages. Assuming the waves incident on each antenna are quasi-monochromatic (ranging the same wavelength and frequency), we may describe voltages V_p and V_q as a function of time t at the terminals of antennas p and q respectively by the sinusoidal wave equation as:

$$V_p(t) = v_p \cos\left(2\pi\nu(t - \tau_g)\right),$$
$$V_q(t) = v_q \cos\left(2\pi\nu t\right),$$

where ν is the frequency of observation. The signals are then multiplied and averaged over time and frequency in an instrument known as the *correlator* whose output is proportional to:

$$\langle V_p(t) V_q(t) \rangle, \tag{2.5}$$

with the angle brackets referring to a time and frequency average. The output of the correlator then becomes:

$$r(\tau_g) = v_p v_q \cos\left(2\pi\nu(t - \tau_g)\right) \cos\left(2\pi\nu t\right). \tag{2.6}$$

Using trigonometric identities:

$$\cos\left(a + b\right) = \cos a \cos\left(b\right) - \sin a \sin\left(b\right),$$
$$\cos\left(a - b\right) = \cos a \cos\left(b\right) + \sin a \sin\left(b\right),$$

we expand and simplify Eqn. (2.6) to:

$$r(\tau_g) = v_p v_q \left(\cos(2\pi\nu\tau_g)\right), \tag{2.7}$$

where $2\pi\tau_g$ represents the geometric delay, while $v_p v_q$ represents the amplitude of received power. The oscillatory part of Eqn. (2.7) represents the motion of the observed source through what is known as the interferometer's fringe pattern and is caused by the variations in τ_g as the Earth rotates (Thompson, 1999).

We now introduce the notion of intensity $I(s)$ of a source measured in $\mathrm{Wm^{-2}Hz^{-1}sr^{-1}}$. This is the amount of flux per frequency ν at a collecting area (aperture) $A(s)$, through a solid angle $d\Omega$ in a direction $\mathbf{s}$. Integrating the intensity over the entire solid angle gives rise to what is is known as flux density, a measurement used for compact sources which is measured in Jansky (Jy)[1]. Considering that the power received by a single antenna in a bandwidth $\Delta\nu$ from element $d\Omega$ is given by $A(s)I(s)\Delta\nu d\Omega$, the resulting output from the correlator for the signal from a solid angle $d\Omega$, excluding the constant amplitudes, is thus given by:

$$dr = A(s)I(s)\Delta\nu d\Omega \cos\left(2\pi\nu\tau_g\right). \tag{2.8}$$

Substituting Eqn. (2.4) in this equation, we rewrite the correlator output in terms of baseline and source position vectors. Integrating over the celestial sphere's[2] surface s

[1] 1 Jansky $= 10^{-26}\mathrm{Wm^{-2}Hz^{-1}}$

[2] Celestial sphere: An arbitrary sphere onto whose surface we assume our astronomical sources to be located.

we get:

$$r = \Delta\nu \int_s A(s)I(s)\cos\frac{2\pi\nu\mathbf{b}\cdot\mathbf{s}}{c}d\Omega. \tag{2.9}$$

We assume that:

- Bandwidth ν is small enough such that variations in A and I with respect to $\Delta\nu$ are disregarded.
- The source is in a far-field of the interferometer such that incoming wavefronts are considered planar.
- Antenna responses from different parts of the source can be independently added and are thus uncorrelated.

We specify the centre of our synthesised FoV (Field of View); known as the *phase reference position* or *phase centre* as $\mathbf{s}_0$, and some distance σ; which is the distance from $\mathbf{s}_0$ during source tracking, such that $\mathbf{s} = \mathbf{s}_0 + \sigma$. We then substitute this in Eqn. (2.9) to obtain:

$$\begin{aligned} r = \Delta\nu\cos\frac{2\pi\nu\mathbf{b}\cdot\mathbf{s_0}}{c}\int_s A(\sigma)I(\sigma)\cos\frac{2\pi\nu\mathbf{b}\cdot\sigma}{c}d\Omega \\ -\Delta\nu\sin\frac{2\pi\nu\mathbf{b}\cdot\mathbf{s_0}}{c}\int_s A(\sigma)I(\sigma)\sin\frac{2\pi\nu\mathbf{b}\cdot\sigma}{c}d\Omega. \end{aligned}$$

At this point, we introduce the term *visibility*, which is a measure of spatial coherence (Clark, 1999) of the electric field of an EM source. It is a complex quantity, which is defined as:

$$V \equiv |V|e^{i\phi v} = \int_s \mathcal{A}(\sigma)I(\sigma)e^{\frac{-2\pi i\nu\mathbf{b}\cdot\sigma}{c}}d\Omega, \tag{2.10}$$

where $\mathcal{A}(\sigma) \equiv \frac{\mathcal{A}(\sigma)}{A_0}$ is the normalised antenna reception (single beam) pattern and A_0 the response at the beam centre. We introduce a coordinate system whereby the baseline vector $\mathbf{b}$ has coordinates (u, v, w) measured in wavelengths, and w is pointing in the source direction, i.e. the phase tracking centre. We then define positions in the sky as having coordinates (l, m, n), which are directional cosines measured with respect to the formerly defined coordinates. We define some parameters:

$$\begin{aligned} \frac{\nu\mathbf{b}\cdot\mathbf{s}}{c} &= ul + vm + wn, \\ \frac{\nu\mathbf{b}\cdot\mathbf{s_0}}{c} &= w, \\ \text{and } d\Omega = \frac{dldm}{n} &= \frac{dldlm}{\sqrt{1-l^2-m^2}}, \end{aligned}$$

which we apply to Eqn. (2.10) to get the complex visibility as:

$$V(u,v,w) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\mathcal{A}(l,m)I(l,m)e^{-2\pi i[ul+vm+w(\sqrt{1-l^2-m^2}-1)]}\frac{dldm}{\sqrt{1-l^2-m^2}}. \tag{2.11}$$

It is possible to reduce Eqn. (2.11) to a 2-Dimensional Fourier transform by enforcing either of 2 conditions:

- We choose w-axis to be in the celestial pole and thus equal to 0, after which we absorb $1/\sqrt{1-l^2-m^2}$ into $I(l,m)$.

- We consider $|l|$ and $|m|$ small enough such that we image a small field in the sky and thus $(\sqrt{1-l^2-m^2}-1)w \approx -\frac{1}{2}(l^2+m^2)w \approx 0$.

We then assume the $\mathcal{A}(l,m)$ is 1 for simplicity for each condition, after which we obtain the Van Cittert-Zernike theorem:

$$V(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I(l,m)e^{-2\pi i(ul+vm)}dldm, \tag{2.12}$$

which is a Fourier transform of the sky brightness $I(l,m)$. Conversely, if we knew the value of the visibility function at each u,v, we could obtain a radio image of our sky by performing an inverse Fourier transform of Eqn. (2.12):

$$\begin{aligned} I(l,m) &= \mathcal{F}^{-1}\{V(u,v)\}, \\ I(l,m) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} V(u,v)e^{2\pi i(ul+vm)}dudv. \end{aligned} \tag{2.13}$$

2.2.2 The Radio Interferometer Measurement Equation (RIME)

In this section, we use Jones calculus, and the Hamaker, Bregman, and Sault, 1996 measurement equation, taking into account propagation effects, to reformulate the Van Cittert-Zernike theorem.

While we were able to see from the previous section that a single measurement from the interferometer corresponds to a 2-D Fourier transform of the sky, we only took into account the geometric delay effect. However, in real life, signals from the source are altered due to different types of propagation effects along the paths over which they propagate. Consequently, an interferometer does not measure an exact depiction of the sky. The RIME reconstructs the Van Cittert-Zernike theorem (see Eqn. (2.12)) while taking into account propagation effects along the signal path. Estimating these effects is the business of calibration – while calibration is not in the scope of this work, we aim to introduce some ideas within this section that have been useful to the work. This section will be based on a summary of Smirnov, 2011a which draws from Hamaker, Bregman, and Sault, 1996.

We consider a single source in Fig. 2.2, the EM field from which at a particular point in time and space can be described using a complex vector $\mathbf{e}$ in an orthonormal xyz coordinate system, where z is the direction of propagation of the signal. Since the signal propagates in the form of a plane wave, $\mathbf{e}$ lacks a z component. Therefore, we represent $\mathbf{e}$ as:

$$\mathbf{e} = \begin{pmatrix} e_x \\ e_y \end{pmatrix}. \tag{2.14}$$

We then assume that all corruptions along a signal's path are linear with respect to $\mathbf{e}$, and can thus be represented as:

$$\mathbf{e}' = \mathbf{J}\mathbf{e}, \tag{2.15}$$

where $\mathbf{J}$ is a complex 2 x 2 Jones matrix (Jones, 1941). Because the signal encounters multiple effects along its path, each of these effects will be represented by a Jones matrix. Thus, all the corruptions affecting the signal will be consequently represented by repetitive matrix multiplications of those matrices, in the order which they occur,

forming what is known as a *Jones chain*. Eqn. (2.15),therefore, becomes:

$$\mathbf{e}' = \mathbf{J}_n\mathbf{J}_{n-1}\ldots\mathbf{J}_1\mathbf{e} = \mathbf{J}\mathbf{e}. \tag{2.16}$$

When the signal arrives at an antenna, it is converted to a complex value [3]. If there are 2 feeds a and b at the antenna, we let the voltages at the feeds be v_a and v_b respectively. Additionally, we treat the process of conversion from the electric field $\mathbf{e}$ to voltage as a linear one, representing it as:

$$\mathbf{v} = \begin{pmatrix} v_a \\ v_b \end{pmatrix} = \mathbf{J}\mathbf{e}, \tag{2.17}$$

with $\mathbf{J}$ representing the linear transformation. We call the $\mathbf{J}$ the *total Jones* as it encompasses all the propagation effects. If we have two antennas p and q, their voltage vectors $\mathbf{v}_p$ and $\mathbf{v}_q$ are fed into a correlator, where as earlier intimated, they are cross-multiplied and averaged over frequency and time. The correlator output can be expressed as:

$$\langle v_{pa}v_{qa}^*\rangle, \langle v_{pa}v_{qb}^*\rangle, \langle v_{pb}v_{qa}^*\rangle, \langle v_{pb}v_{qb}^*\rangle, \tag{2.18}$$

in which the angle brackets represent the averaging, while the $(*)$ represents the complex conjugate operator. We place Eqn. (2.18) in a matrix to get:

$$V_{pq} = 2\begin{pmatrix} \langle v_{pa}v_{qa}^*\rangle & \langle v_{pa}v_{qb}^*\rangle \\ \langle v_{pb}v_{qa}^*\rangle & \langle v_{pb}v_{qb}^*\rangle \end{pmatrix}, \tag{2.19}$$

which we reduce to:

$$V_{pq} = 2\left\langle \begin{pmatrix} v_{pq} \\ v_{pb} \end{pmatrix} \begin{pmatrix} v_{qa}^* v_{qb}^* \end{pmatrix} \right\rangle = 2\langle \mathbf{v}_p\mathbf{v}_q^H\rangle, \tag{2.20}$$

where H is a Hermitian transpose. The factor of 2 is introduced as a matter of convention, to obtain unity correlations for an I of 1 Jy (Smirnov, 2011a). Because signals to both these antennas have their independent paths, we define a total Jones matrix for each signal path. Therefore, substituting the RHS of Eqn. (2.17) into Eqn. (2.20), we obtain:

$$V_{pq} = 2\langle \mathbf{J}_p\mathbf{e}(\mathbf{J}_q\mathbf{e})^H\rangle = 2\langle \mathbf{J}_p(\mathbf{e}\mathbf{e}^H)\mathbf{J}_q^H\rangle, \tag{2.21}$$

for which we assume that the Jones terms are constant over averaging and thus, can be taken out of the averaging operator. Hence, Eqn. (2.21) becomes:

$$V_{pq} = 2\mathbf{J}_p\langle \mathbf{e}\mathbf{e}^H\rangle\mathbf{J}_q^H = 2\mathbf{J}_p\begin{pmatrix} \langle e_x e_x^*\rangle & \langle e_x e_y^*\rangle \\ \langle e_y e_x^*\rangle & \langle e_y e_y^*\rangle \end{pmatrix}\mathbf{J}_q^H. \tag{2.22}$$

The terms in the brackets relate to the Stokes parameters:

$$\begin{aligned} I &= \langle e_x e_x^*\rangle + \langle e_y e_y^*\rangle, \\ Q &= \langle e_x e_x^*\rangle - \langle e_y e_y^*\rangle, \\ U &= \langle e_x e_y^*\rangle + \langle e_y e_x^*\rangle = 2\Re\langle e_x e_y^*\rangle, \\ V &= -\imath(\langle e_x e_y^*\rangle - \langle e_y e_x^*\rangle) = 2\Im\langle e_x e_y^*\rangle, \end{aligned}$$

[3] Technically, the voltage is a real measurement, but it is converted to a complex value by adding a phase delay of $\pi/2$ within the correlator.

where $\Re$ and $\Im$ represent the real and imaginary components respectively, such that:

$$2\begin{pmatrix}\langle e_x e_x^*\rangle & \langle e_x e_y^*\rangle\\ \langle e_y e_x^*\rangle & \langle e_y e_y^*\rangle\end{pmatrix} = \begin{pmatrix}I+Q & U+iV\\ U-iV & I-Q\end{pmatrix} = \mathcal{B}, \tag{2.23}$$

where I represents total intensity, Q and U are linear polarisation terms and V is the circular polarisation term. Eqn. (2.23) is called the brightness matrix, which henceforth we denote as $\mathcal{B}$. From it, we can formulate the RIME as:

$$V_{pq} = \mathbf{J}_p \mathcal{B} \mathbf{J}_q^H. \tag{2.24}$$

If we expand Eqn. (2.24) to its Jones chain:

$$V_{pq} = \mathbf{J}_{pn}(\mathbf{J}_{pn-1}(\ldots(\mathbf{J}_{p1}\mathcal{B}\mathbf{J}_{q1}^H)\ldots)\mathbf{J}_{qm-1}^H)\mathbf{J}_{qm}^H, \tag{2.25}$$

we get a form of the RIME known as the onion form, where effects with the smallest values of m and n occur at the source, while effects at the outer ends occur at the antennas. It is worth pointing out that $n \neq m$ because the paths taken by the signal to antennas p and q can completely differ (Smirnov, 2011a).

One intrinsic signal propagation effect is that of phase delay due to pathlength difference, which exists even in the absence of any corrupting factors. Phase delay is described by a K-Jones scalar diagonal matrix, given by:

$$K_p = e^{-i\kappa_p} = e^{-2\pi i(u_p l + v_p m + w_p(n-1))}, \tag{2.26}$$

whose derivation can be found in (Smirnov, 2011a). The RIME for a single uncorrupted source becomes:

$$V_{pq} = \mathbf{K}_p \mathcal{B} \mathbf{K}_q^H = \mathbf{X}_{pq}, \tag{2.27}$$

and this is the visibility that an ideal interferometer devoid of corruptions would measure. X_{pq} is called the source coherency. Recalling that a signal is corrupted along its path, we then represent all these other corruptions using a G-Jones matrix to get:

$$V_{pq} = \mathbf{G}_p \mathbf{X}_{pq} \mathbf{G}_p^H. \tag{2.28}$$

For multiple discrete sources, contributions from each source add up linearly as:

$$V_{pq} = \sum_s \mathbf{J}_{sp} \mathcal{B}_{spq} \mathbf{J}_{sq}^H. \tag{2.29}$$

If we group $\mathbf{J}_{sp}$ into the source independent $\mathbf{G}_p$ term, source dependent $\mathbf{E}_{sp}$ term and the phase term $\mathbf{K}_{sp}$ we end up with:

$$V_{pq} = \mathbf{G}_p\Big(\sum_s \mathbf{E}_{sp}\mathbf{X}_{spq}\mathbf{E}_{sq}^H\Big)\mathbf{G}_q^H, \tag{2.30}$$

the RIME for multiple discrete sources. We recognize that in reality, the sky has a continuous brightness distribution rather than discrete sources. Therefore, to get the total visibility measured by an interferometer, brightness from all directions must be considered.

Following (Smirnov, 2011a), that if we project the celestial sphere onto a plane (l, m) tangential to a particular field's centre, and we consider effects on direction w (towards the field) in the uvw coordinate system as direction-dependent effects, we obtain the

full sky RIME:

$$V_{pq} = \mathbf{G}_p\Big(\iint_{lm} \mathcal{B}_{pq} e^{-2\pi i(u_{pq}+v_{pq}m)} dldm \Big) \mathbf{G}_q^H. \tag{2.31}$$

This is yet another 2-D transform of the "apparent sky brightness as seen by a baseline pq" (Smirnov, 2011a). We will refer to this visibility as the *observed visibility*.

2.3 Calibration

The goal of radio astronomy is to observe objects in the sky through the radio window as intimated earlier. Nevertheless, signals received by radio interferometers are marred due to the existence of other external and stronger signals such as those from communication devices and aeroplanes which emit what is known as Radio Frequency Interference (RFI), as well as atmospheric and instrumental effects. As a result, an interferometer samples other unwanted signals in addition to the desired signal, which is already tainted due to the signal path it takes. Known and anticipated RFI and alterations due to the instrument may be mitigated during observation time and at the instrumentation level. However, some noise, as well as instrumental and atmospheric gains, still corrupt the recorded data. We referred to this recorded data as the observed visibilities (recall Section 2.2.2).

Thus, calibration of data is the process of attempting to estimate the desired (actual) visibility data, known as *true visibilities*, from the observed visibilities (Fomalont and Perley, 1999). We can categorise calibration into 3 regimes (Noordam and Smirnov, 2010) :

- First-generation calibration.
- Second-generation calibration.
- Third-generation calibration.

First and second-generation calibration processes are direction-independent, while third-generation calibration is direction-dependent. For this work, we shall mostly focus on the first-generation calibration. More on 2GC and 3GC can be found in (Smirnov, 2011b; Smirnov, 2011c).

2.3.1 First Generation Calibration [1GC]

1GC is a calibrator-field based calibration, and hence, one needs to observe a source, known as a *calibrator field*. This field usually has a known behaviour in frequency, as well as known flux and shape to enable the tracking of observational parameters and solve for gains which are eventually transferred and applied to the desired field. While some calibration parameters may be known a priori, those that are unknown must be solved for from the calibrator sources. We now discuss some of the quantities for which we calibrate and whose Jones terms need to be solved.

Delay Calibration

Delay errors in the signal path arise due to a number of things:

- Atmospheric delay,
- Electronic delay,

- Geometric delay due to separation of antennas in a baseline,
- Errors in delay tracking in the correlator due to inaccurate models for the geometric delay.

Although geometric delay can be accurately determined, small residual delay errors inevitably remain as a result of the other effects. Delay calibration is thus a measurement of these remaining errors (Fomalont and Perley, 1999). Delays manifest as a time constant frequency-dependent linear phase slopes in the correlated data for each baseline.

Delays are represented by a K-Jones scalar diagonal matrix:

$$\mathbf{K} = \begin{pmatrix} e^{i\phi} & 0 \\ 0 & e^{i\phi} \end{pmatrix},$$

where $e^{i\phi}$ represents the phase delay. This was defined in Eqn. (2.26). Delay calibration is performed by observing a very bright, invariant point source, and then solving for the per-antenna ϕ terms.

Bandpass Calibration

Bandpass calibration is the process of estimating and correcting for the frequency-dependent aspects of the observing instruments, which may be due to the antenna electronics or the environment. It involves observation of a bright and invariant point source whose frequency spectrum is known over the observing frequency band.

Because radio sources of interest may exhibit relatively narrow spectral features (such as absorption and emission lines), it is necessary to perform observations spanning multiple adjacent frequency channels. The channels must have enough frequency resolution to separate emission regions (Fomalont and Perley, 1999). Modern radio telescopes observe over extremely wide frequency bands (a 2:1 ratio is not unusual) with many channels. For example, MeerKAT has 1024, 4096 and 32768-channel observing modes. Bandpass calibration is therefore necessary for accurate detection and measurement of spectral features. Bandpass is represented by the B-Jones matrix:

$$\mathbf{B} = \begin{pmatrix} b_a(\nu) & 0 \\ 0 & b_b(\nu) \end{pmatrix},$$

where $b(\nu)$ is a complex, frequency variable gain and indices a and b represent the antenna feeds.

Complex Gain Calibration

This entails correcting for the time-dependent part of antenna gains. Observed signals can be time variable due to environmental and atmospheric factors, which cause continuous fluctuations in the gain amplitude and phases. Of the two, the phases are more adversely affected. Gain calibration thus serves the purpose of mitigating the large fluctuations, thus increasing coherence.

It is performed regularly throughout the observation. It also requires a moderately bright calibrator source that is close to the target field. This source is additionally used to monitor atmospheric conditions and hence, is likely to be affected by the

same environmental effects as the target field. It is described by the G-Jones complex diagonal matrix:

$$\mathbf{G} = \begin{pmatrix} g_a(t) & 0 \\ 0 & g_b(t) \end{pmatrix},$$

where $g(t)$ is a complex, time variable gain, and indices a and b again denoting the antenna feeds.

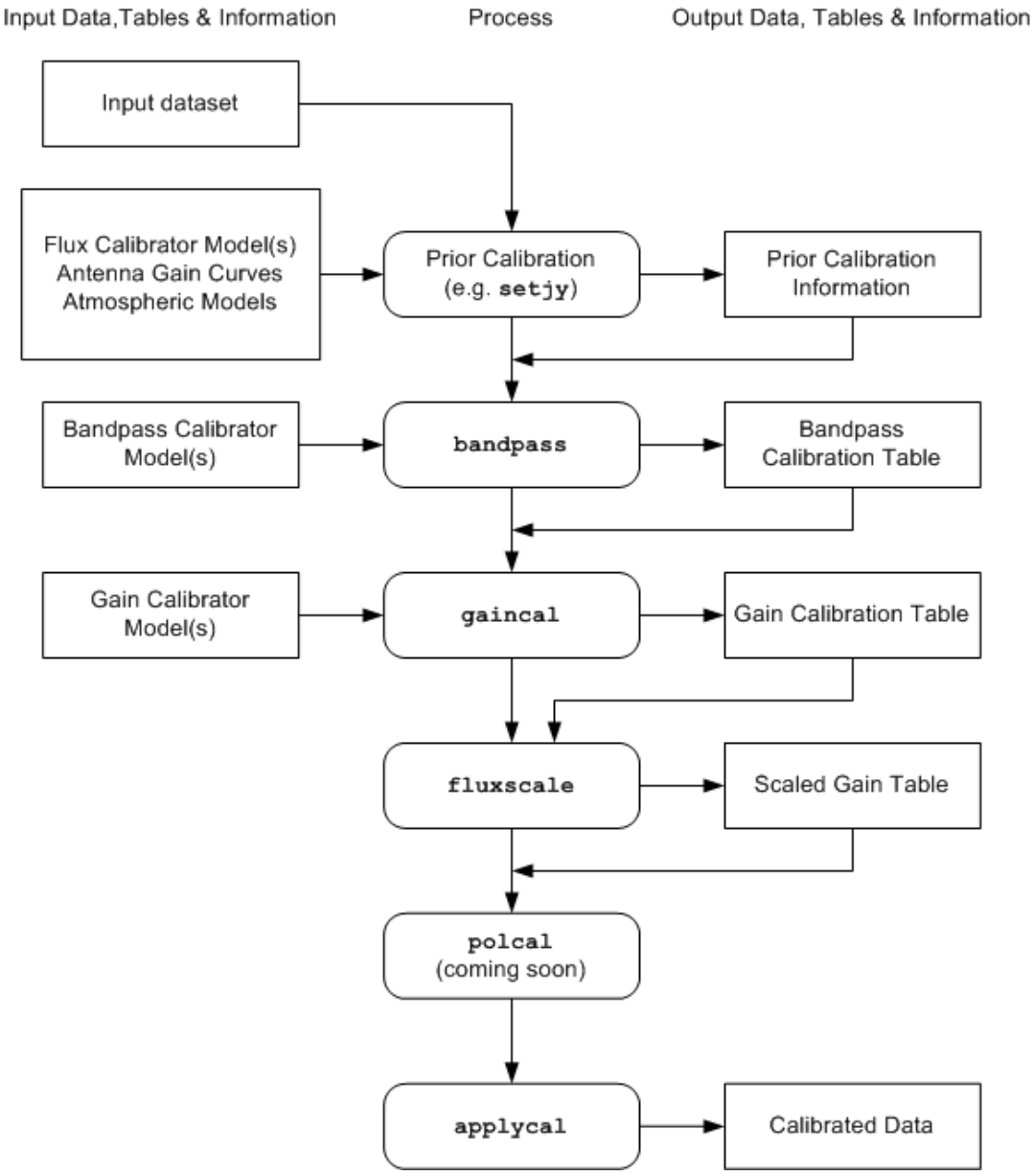


FIGURE 2.3: An example of the CASA calibration workflow. Image source CASA.

Polarisation Calibration

Antennas have receiver feeds which are orthogonal to each other. Each of these receivers is sensitive to different types of polarisation. These may be left and right circular polarisation (L, R) or linear polarisation (X, Y). Ideally, these receivers should be independent of each other and have no cross-talk. In practice, however, a small fraction of the signal from one feed leaks through to the other, and vice versa (Hamaker,

Bregman, and Sault, 1996) leading to impurities in the recorded signals. The goal of polarisation calibration is to measure and correct for these leakages. Leakage values are small (Hamaker, Bregman, and Sault, 1996) and are denoted by a D-Jones off-diagonal matrix:

$$\mathbf{D} = \begin{pmatrix} 1 & d_a \\ -d_b & 1 \end{pmatrix}.$$

Absolute Flux Calibration

Absolute flux calibration is done to determine the true flux of a source in a target field. It necessitates a bright, point-like and invariant calibrator source. Flux calibration enforces correct flux scaling.

Observed visibility data is typically recorded and stored in the Measurement Set (MS) format (see Section 3.1 for MS). From this point, the data then undergoes the series of 1GC calibration steps previously highlighted and partly illustrated by Fig. 2.3. At the end of each of the calibration steps, gain solutions corresponding to the estimated corrections applicable to the observed data are generated and stored in *gain tables* also known as *calibration tables*. After all the desired calibration steps have been performed, these solutions are then applied to the observed data, resulting in calibrated data.

2.3.2 Importance of Non-Imaging Data Inspection

Calibration is a time consuming and intricate process that requires attention and care. This is because applying improper calibration solutions to observed visibilities leads to flawed calibrated data, which may have significant effects on the science goals. Hence, it is essential to monitor and examine the quality of calibration solutions beforehand, to determine their correctness and change calibration strategy if need be. Furthermore, it allows the detection of problems earlier, before imaging the data in order to examine it and to assess residuals between modelled and calibrated visibilities. These latter processes may be computationally expensive and time-consuming (Heald et al., 2018).

Non-imaging data inspection also provides a means of quick identification and excision of corrupt data. It can be easier to inspect data in the visibility domain rather than the image domain as errors in the image domain are spread throughout the image. However, in the visibility domain, they are localized. Moreover, to the trained eye, the visibility domain can also reveal some basic properties corresponding to the Fourier transform of the brightness distribution of the observed source (Pearson, 1999). In particular, calibrator sources, owing to their simple spatial structure, have very simple signatures in the visibility domain, which make data outliers especially obvious.

2.3.3 Beyond Visibilities

While imaging is not in the scope of this work, we briefly summarise its idea for the sake of completeness. We showed that images of the sky can be produced by performing an inverse Fourier transform (see Eqn. (2.12), (2.13) and (2.7)). However, since an interferometer performs a finite and discrete sampling of the uv-plane, Eqn. (2.13)

must be modified to include the sampling function $S(u,v)$ becoming:

$$I^D(l,m) = \iint V(u,v)S(u,v)e^{2\pi i(ul+vm)}dudv, \tag{2.32}$$

$$I^D(l,m) = \mathcal{F}^{-1}\{V(u,v)S(u,v)\}, \tag{2.33}$$

$$I^D(l,m) = I(l,m) * P(l,m). \tag{2.34}$$

Therefore, an image produced at this stage is known as a *dirty image* I^D, because it is corrupted by the effects of the sampling function, known in the image domain as the *point spread function* (PSF) or the *dirty beam* $P(l,m)$. This effect is described by Eqn. (2.34), which shows that the dirty image is a convolution ($*$) of the "true sky" and the PSF. Therefore, to remove the effect of the PSF, the *deconvolution* process must be performed after which an image of the "true sky" can be obtained.

Chapter 3

Tools of the Trade

The goal of this chapter is to introduce the tools we used to accomplish this work. Data measured by a radio interferometer must first be recorded and stored to allow for calibration and visualisation. Therefore, we introduce the Measurement Set, a format that specifies how recorded telescope data is to be stored. It is then followed by a description of the general structure of Measurement Sets and calibration tables and the tools that can be used to extract data from them. We then finally describe the tools chosen for data visualisation.

3.1 Measurement Sets

Data from aperture synthesis telescopes is commonly stored in a table format known as a Measurement Set (MS). The MS was first developed as part of the AIPS++ project, which eventually became what is now known as `CASA`, and is a CTDS (Casacore Table Data System) table whose layout prescribes how telescope data should be stored regardless of how the data was recorded (Diepen, 2015; McMullin, Golap, and Myers, 2004; Diepen and Farris, 1994).

3.1.1 Structure of a Measurement Set

A Measurement Set consists of a single main table which contains the measured visibility data and sub-tables which contain auxiliary metadata about a particular observation (Kemball and Wieringa, 2000). The main table is linked to the different sub-tables through *foreign keys*. These keys, which are row numbers in the sub-tables, are stored in columns within the main tables. Fig. 3.1 shows an example MS structure, known as a schema, whereby the items in blue indicate the table names while the rest show the data columns contained in those specific tables.

Column cells can hold scalar values or N-dimensional arrays. Hence, 2-dimensional visibility data of shape $N_{\text{chans}} \times N_{\text{corrs}}$, is stored in the DATA column of the main table. In this case, the single cell contains data for a single baseline at a specific time instance. Considering a single observation with multiple baselines, the DATA column can then be seen to be 3-dimensional with a shape of $N_{\text{rows}} \times N_{\text{chans}} \times N_{\text{corrs}}$. The N_{rows} is equivalent to the number of samples recorded by unique baselines (recall Eqn. (2.3)) throughout the observation, N_{chans} is the number of frequency channels used for observation, while N_{corrs} is the number of correlations available in the data. We note that this data shape is not solely limited to the data column. Depending on factors such as the length of observation, integration time τ, number of spectral windows, number of frequency channels and the number of baselines, Measurement Sets can be massive in size (up to terabytes for a single observation). This is because the amount of recorded visibility points - and other auxiliary data - increases with

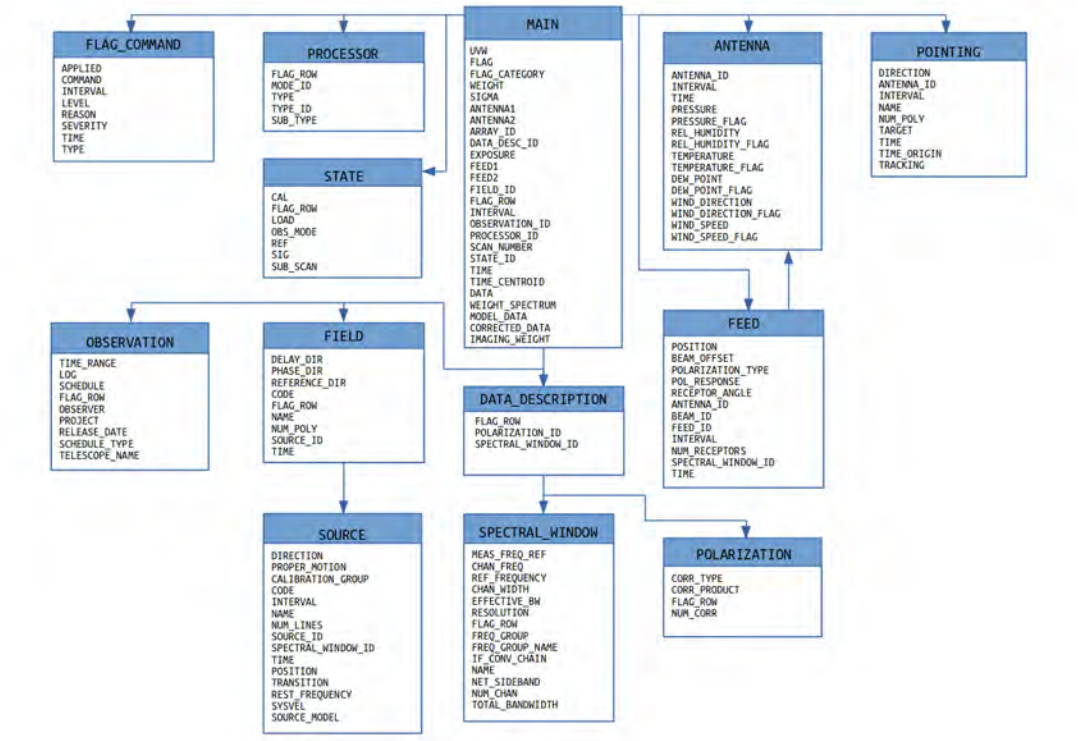


FIGURE 3.1: An example MS schema showing the structure of the main table and sub-tables. The blue colour indicates the table name, while all the items are the contents of that table. This image is adapted from (Diepen, 2015).

each of those factors. For instance, if we consider that a single visibility point is stored in an 8-byte memory space, the memory consumed by only the visibility data can be represented by:

$$\frac{T_{\text{obs}}}{\tau} \times N_{\text{spw}} \times N_{\text{bl}} \times N_{\text{chan}} \times N_{\text{corr}} \times 8 \text{ Bytes}, \tag{3.1}$$

where T_{obs} is the length of the observation τ is the integration time and $\frac{T_{\text{obs}}}{\tau} \times N_{\text{bl}} \times N_{\text{spw}}$ corresponds to N_{rows}.

Different calibration tools exist and have varying formats for storing calibration solutions. For example, the Lofar Solution Tool (LoSoTo[1]) (De Gasperin et al., 2019) stores solutions in what are referred to as solution tables (soltabs), while MeqTrees[2] (Noordam and Smirnov, 2010) stores solutions in MEP tables. For this work, we shall focus on `CASA`'s calibration tables.

These have a similar structure to MSs. That is, they have a main table – which consists of the calibration solutions, and ancillary tables – consisting of metadata such as the antennas and fields for which the solutions are available, as well as the type of

[1] https://support.astron.nl/LOFARImagingCookbook/losoto.html
[2] http://meqtrees.net/

Jones' matrix for which the solutions pertain to. As discussed in Section 2.3, calibration tables are produced at each calibration step. However, they are much smaller in size because the solutions are per antenna and per correlation. This means that N_{rows} will correspond to the number of antennas available in an array, times the number of solution intervals.

It is important to note that during the process of calibration using `CASA`, calibration solutions are calculated with respect to a reference antenna which is chosen beforehand. An appropriate choice for a reference antenna is often one that is close to the centre of the telescope array and preferably stable (with minimal variability) during the duration of the observation. It's phase is assumed to be zero.

3.1.2 Accessing Measurement Set Data

Data from MSs and calibration tables can be accessed through the use of `CASA`, which provides a Graphical User Interface (GUI), through which a user can browse a table. Additionally, `CASA` provides an enhanced interactive `Python` shell (`IPython`[3]) interface which may be used to access and perform operations on the desired data, but whose caveat is that launching the entire `CASA` application is required. The `Python-Casacore`[4] package offers an alternative to this.

`Python-Casacore` is a high-level `Python` package that provides an interface to Casacore tables such as MSs and calibration tables. Because cells in Casacore tables can contain array data, columnar data is accessed as N-dimensional `NumPy`[5] arrays which enables fast and optimised computations on the data. However, `NumPy` performs immediate evaluation operations. Thus, the result of a user-defined operation on data is returned almost immediately. Immediate evaluation, therefore, is only suitable in cases where data can fit comfortably in the Random-Access Memory (RAM), which temporarily stores data ready to be processed by a Central Processing Unit (CPU). `Python-Casacore` offers a way around this problem by providing an interface which enables users to perform operations such as aggregation and averaging of data, as well as querying (selecting) only desired or required data from MSs and gain tables before it is loaded into RAM. This can be achieved via appropriate Table Query Language (TaQL) (Diepen, 2018) command. This allows data to be loaded into RAM in smaller chunks, by specifying iterative TaQL commands which will load certain portions of the data matching the specified criterion. While this is advantageous, immediate evaluation of data still occurs and this may prove inefficient as datasets grow in size. With projects such as the SKA, it is expected that the amounts of retrieved data will only increase. Thus, an alternate method of data processing is inevitably needed.

The `Python` package `Dask`[6], offers a suitable solution. While `Dask` exposes interfaces and mechanisms similar to those of `NumPy`, it performs what are known as *lazy evaluations*. This means that operations on data are only performed when they are explicitly triggered. Therefore, in cases whereby the data to be processed is larger than the available RAM, behind the scenes that data can be split into smaller chunks that can comfortably fit into the memory. Given an operation on the data, a single

[3] https://ipython.org/
[4] https://github.com/casacore/python-casacore
[5] https://numpy.org/
[6] https://dask.org/

chunk is loaded into the RAM to be processed by the CPU, and an intermediate result from that operation is computed and stored in memory. This process is repeated for all available chunks, after which all the intermediate results are combined into a single result which is then returned (Rocklin, 2015; Dean and Ghemawat, 2008). If the data chunks are small enough such that multiple chunks can be loaded into RAM simultaneously, `Dask` is able to operate on those chunks simultaneously (in parallel). This parallelism increases the speed of computations.

`Dask-ms`[7] leverages the parallelisation offered by `Dask` to provide a wrapper around `Python-Casacore`. It uses `Dask` arrays as the back-end rather than `Numpy` arrays. Moreover, `Dask-ms` makes use of TaQL commands from `Python-Casacore` to organise how data is read by `Dask`. Because of its suitability for large data, `Dask-ms` was chosen as an interface to access the Measurement Set and calibration table data.

3.2 Visualisation Tools

3.2.1 BokehJS

`BokehJS`[8] is a `Python` based library that utilises the `JavaScript` (JS) programming language to provide high-level, web-based and interactive plots. Interactivity in this context refers to the ability to modify the information displayed on a plot without reloading and re-plotting the data.

A `BokehJS` plot can be thought of as a container for objects such as data sources, glyphs (lines, circles, rectangles) and widgets. All these are collectively known as models, which are defined in the `Python` language but have corresponding implementations in JS. When a `BokehJS` plot is defined, it serialises all the `Python` models into a format understood by JS known as JavaScript Object Notation (JSON), whereby the JS counterpart of `BokehJS` reconstitutes the models into visual objects. Plots output from `BokehJS` are then stored in the Hypertext Markup Language (HTML) format, which can be inspected using a web browser. Therefore, `BokehJS` represents each data point availed to it without any form of aggregations, and its plots operate similarly to interactive web-pages. As a result, the response of the plot to interactive actions is dependant on the amount of memory resource allocated to the web browser. Hence, we can comfortably use `BokehJS` for plotting calibration solutions because, as intimated in Section 3.1, they are relatively small in size and thus have fewer data points, and so can be rendered by a typical browser. The acceptable number of points which permits meaningful interactivity is approximately thirty thousand data points. In contrast, visibility data, which can range to over billions of points, can not be represented by `BokehJS` and thus, we look to `Datashader`[9].

3.2.2 Datashader

Contrary to `BokehJS`, `Datashader` does not plot each data point. In `Datashader`, 2-dimensional data of arbitrary size is projected to a fixed, user-defined 2-dimensional grid, where each cell corresponds to a pixel and the grid dimensions correspond to the final image size. Because of this projection, multiple points falling into the same cell are aggregated by an incremental function, such as addition. The data is then

[7] https://dask-ms.readthedocs.io/en/latest/readme.html
[8] https://bokeh.pydata.org/en/latest/
[9] http://datashader.org/

mapped to colours, with empty cells having an opacity of 0 (fully transparent) and cells containing data having an opacity of 1 (fully visible) and a plot, which is an image, is produced (Bednar, 2016). `Datashader` also uses `Dask` parallelisation - performing computations on different chunks of data simultaneously - and therefore, can be used to represent an arbitrarily large dataset in an optimised and efficient manner. This makes it suitable for plotting visibility data. Moreover, it can be linked with `BokehJS`, which consequently permits the embedding of interactive features into the resulting image. `Datashader` plots can therefore also be saved in the HTML format.

3.3 Pipelines

A data pipeline refers to a series of transformations that a raw dataset undergoes resulting in a processed dataset. The transformations may be performed by various software interlinked in such a way that the output of one process becomes the input of another, thus automating the entire data processing procedure. This fosters a systematic and reproducible process. In radio astronomy, pipelines are becoming indispensable due to the vast amounts of available data, and the need for its methodical exploitation (Ruiz et al., 2012). The procedure from a raw visibility dataset to a science-ready image typically involves the reduction in the size of the dataset, possibly at each step. Hence, pipelines are typically referred to as *data reduction pipelines*. An example of such is the `rPICARD` pipeline, which uses `CASA` to perform Very Long Baseline Interferometry[10] (VLBI) data calibration and imaging (Janssen et al., 2019).

However, depending on the specificity of research, data reduction pipelines may require the installation, co-existence and coordinated use of multiple software within the same environment. This has been challenging, especially within astronomy. To address this, an Ubuntu operating system repository known as KERN Suite[11] was developed. Here, scientific software are bundled as Ubuntu packages, thus facilitating their easy installation and enhancing their interoperability (Molenaar and Smirnov, 2018). This, as well as containerisation technologies such as Docker[12], and Singularity[13] that provide isolated environments (containers) which house software and their dependencies allowing them to execute smoothly, have propelled the emergence of automated data reduction pipelines that use multiple software to achieve science-ready data products. We describe such a pipeline, `CARACal`, which is a data reduction pipeline developed for the MeerKAT telescope, and is the basis on which this work exists.

3.3.1 A Brief Description of the `CARACal` pipeline

`CARACal` (Containerized Automated Radio Astronomy Calibration), is an open-source, end-to-end data reduction pipeline based on `Python` programming language. It was developed specifically for use with data from the MeerKAT telescope (the original name of the project was `MeerKATHI`), but has already been applied to data from other telescopes which use the Measurement Set format (Michałowski et al., 2019; Maccagni et al., 2019). `CARACal` performs a range of tasks, including data flagging (removing

[10] VLBI is a type of interferometry that involves the collection and combination of radio signals from telescopes separated by long distances across the earth, hence creating long baseline distances. It is however not in the scope of this work and shall not be discussed further.

[11] https://kernsuite.info

[12] https://www.docker.com/

[13] https://sylabs.io/singularity/

bad data), calibration and imaging. At its core is a platform-agnostic radio interferometry scripting framework - also based on `Python` and container technologies - known as `Stimela`[14], which provides an interface to the different software used within the pipeline for data processing, and manages their execution within their containers (Makhathini, 2017).

Containers are created from what are known as *images*. An image is a file that consists of specifications of the software components and dependencies to be availed inside a container. When executed, an image spawns into a container. `Stimela` consists of two types of images. The first kind are known as base images in which radio astronomy software from the KERN project installed. The second kind are cab images; also container images, which consist of `Python` scripts that perform specific functions. A cab image takes in an input file (the file to be operated on), and the parameters required to perform a specific function, executes the required function on that file, and produces its output. Different tasks using different software within `CARACal` such as calibration, data inspection and imaging are performed via these cab images in an isolated fashion. Because a container image is executed within a specific, pre-defined environment, the tasks performed are almost guaranteed[15] to execute the same way, repeatedly, therefore promoting reproducibility. Furthermore, container images can be executed on different machines and environments, thus making them portable.

To run an end-to-end data reduction with `CARACal`, a configuration file containing parameters required for the execution of each task must be defined. The parameters are then dispatched to the tasks in which they are required during run time, thus requiring little or no human intervention. However, the reduction process requires an occasional assessment to verify its quality. This can be done through the inspection of resulting data products such as calibration tables and calibrated visibility data. In the next chapter, we introduce `Ragavi`, a tool used to visualise these products within the `CARACal` pipeline.

[14] https://github.com/SpheMakh/Stimela

[15] Given sufficient resources such as RAM and disk space

Chapter 4

Pipeline Visualisation Tools

Calibration is a vital process that allows for minimisation of the errors present in observed visibility data, thus enabling the reconstruction of a *true sky* (refer to Section 2.3). Therefore, it is crucial to ensure that it is done properly and effectively. A way to ensure this is by assessing the quality of the calibration products and the final calibrated visibility data by means of plots. If visualised correctly, these may provide additional and useful insights on the observed data, in addition to an expedited error identification and resolution process. Plots from which one can directly trace the origin of a data point with the click or hover of a mouse over that data point can be beneficial[1]. Such plots are known as interactive plots, and can be dynamically modified, giving one control over what is visible within a plot.

Tools such as `JPlotter`[2], `Matplotlib`[3] and `CASA` (specifically, its `plotms` and `plotcal` tasks) have made interactive visualisation of calibration data products possible. They are partly `Python` based, optimised, robust, and reasonably intuitive to use and fairly well documented. However, the interactivity of these tools is only available in *online* mode, that is, it requires an X11 session to be running, and specific applications to be started up on a host having access to the data. These applications can be time-consuming to install and set up. Moreover, data must be loaded into the tools each time interactive plots are required. If the data is being processed on a remote compute cluster (as is increasingly the case in radio astronomy), online plotting becomes cumbersome or even impossible.

The above tools can also make plots in *offline* mode, by rendering them into a static image in a format such as Portable Network Graphics (PNG), Joint Photographic Experts Group (JPEG) or Scalable Vector Graphics (SVG). This is more practical in a remote computing environment but sacrifices the element of interactivity.

In this chapter, we present `RAGaVI`, a `Python` based visualisation tool that generates *interactive* gain and visibility plots that can be viewed *offline*, by employing the HTML (Hypertext Markup Language) format, which requires only a standard web browser (rather than online GUI sessions) to view and interact with.

Note that `RAGaVI` is the author's work.

[1] Origin here can refer to the antenna, baseline, scan, spectral window or any other information associated with that data point.

[2] https://github.com/haavee/jiveplot

[3] https://github.com/matplotlib/matplotlib

4.1 RAGaVI

RAGaVI (Radio Astronomy Gains and Visibilities Inspector)[4] is an open-source, Python based tool that generates interactive plots, given inputs as MSs or calibration tables. We previously defined the interactivity of a plot in Section 3.2.1, as the ability to dynamically modify its appearance, without the need for repeatedly reloading the data, or reproducing a new plot. RAGaVI is divided into two parts as suggested by its name: Ragavi-gains; responsible for plotting calibration solutions and Ragavi-vis; for plotting visibility data. Henceforth, we shall refer to the plots yielded by these tools as gain plots and visibility plots respectively. RAGaVI can be easily installed via the standard Python package installation tool pip as it is available on the Python repository PyPI[5].

4.2 Architecture

RAGaVI depends on a few other packages for its functionality. Fortunately, all the software is available in the Python ecosystem and easily accessible and installable via pip. We now discuss the roles of some of the Python packages that feature prominently in RAGaVI.

4.2.1 Data Access

To access MSs and calibration table data, RAGaVI makes use of the tool Dask-ms[6], which forms a proxy for Python-Casacore while taking advantage of Dask's lazily evaluated computations. This is in contrast to Python-Casacore-based applications, which perform strict computations through NumPy.

4.2.2 Computation

Computations in RAGaVI are performed by Dask[7] which provides an interface for fast and efficient computations over large amounts of data. It does so by dividing the data into smaller chunks. This allows to both process the chunks in parallel (by fitting multiple chunks into RAM), increasing the computational speed, while automatically supporting out-of-memory computation (by loading chunks from disk in an iterative manner). Dask is particularly well suited for visibility data, which can be enormous in size (compared to available memory).

4.2.3 Visualisation

BokehJS and Datashader are the visualisation back ends to Ragavi-gains and Ragavi-vis respectively. BokehJS was chosen because it offers the functionality of producing interactive plots with a reasonable number of data points as HTML files. Furthermore, it is well integrated with Datashader, whose modus operandi is fundamentally different and thus can be used to plot a much larger number of data points as compared to BokehJS. Nevertheless, its interactivity is conveniently furnished by BokehJS.

[4] https://ragavi.readthedocs.io/en/latest/
[5] https://pypi.org/project/ragavi/
[6] https://xarray-ms.readthedocs.io/en/latest/readme.html
[7] https://dask.org

4.3 User Interface

Unlike the other aforementioned visualisation tools, `RAGaVI` has a Command Line Interface (CLI) rather than a GUI. Therefore, users can use it by just specifying the required and desired arguments from the terminal, rather than launch an entire application or application suite. Fig. 4.1 and 4.2 show the arguments (options) available for `Ragavi-gains` and `Ragavi-vis` respectively.

```
optional arguments:
  -h, --help            show this help message and exit
  -a  , --ant           Plot only a specific antenna, or comma-separated list
                        of antennas. Defaults to all.
  -c  , --corr          Correlation index to plot. Can be a single integer or
                        comma separated integers e.g '0,2'. Defaults to all.
  --cmap                Matplotlib colour map to use for antennas. Defaults to
                        coolwarm
  -d  , --doplot        Plot complex values as amplitude & phase (ap) or real
                        and imaginary (ri). Defaults to ap.
  -f , --field          Field ID(s) / NAME(s) to plot. Can be specified as
                        "0", "0,2,4", "0~3" (inclusive range), "0:3"
                        (exclusive range), "3:" (from 3 to last) or using a
                        field name or comma separated field names. Defaults to
                        all
  --htmlname            Output HTMLfile name
  -p  , --plotname      Output PNG or SVG image name
  --ddid                SPECTRAL_WINDOW_ID or ddid number. Defaults to all
  --t0                  Minimum time to plot [in seconds]. Defaults to full
                        range]
  --t1                  Maximum time to plot [in seconds]. Defaults to full
                        range
  --taql                TAQL where clause

Required arguments:
  -g   [  ...], --gaintype   [  ...]
                        Type of table(s) to be plotted. Can be specified as a
                        single character e.g. "B" if a single table has been
                        provided or space separated list e.g B D G if multiple
                        tables have been specified. Valid choices are B D G K
                        & F
  -t   [  ...], --table   [  ...]
                        Table(s) to plot. Multiple tables can be specified as
                        a space separated list
```

FIGURE 4.1: Command line options for `Ragavi-gains`.

The mandatory argument for `Ragavi-gains` is:

- `table`: the calibration table that is to be plotted.

This may be supplied as a space separated list containing different table names, which instructs `Ragavi-gains` to create a single output plots of both tables. The type of gain table can also be supplied using `-gaintype`, otherwise, it is inferred from the gain table's metadata. An example call to `Ragavi-gains` is:

```
$ ragavi-gains --table path/to/table.B0 path/to/table2.G0
```

LISTING 4.1: Instructing `Ragavi-gains` to plot multiple tables and store both plots in the same single HTML file.

from which a single HTML file with plots for both the B-Jones tables and the G-Jones tables will emerge. Besides a CLI, `Ragavi-gains` can also be used in the `Jupyter` notebook environment through a single function call.

The basic arguments required to produce a visibility plot for `Ragavi-vis` are:

- `ms`: MS to be plotted.

- `xaxis`: The desired x-axis data.
- `yaxis`: Desired y-axis data.

A simple invocation of `Ragavi-vis` would then be:

```
$ ragavi-vis --ms path/to/table.ms --xaxis time --yaxis amplitude
```

LISTING 4.2: A call to plot amplitude vs time `Ragavi-vis`.

which yields a plot of amplitude against time. In the next section, we discuss some of the remaining optional arguments and dissect the features of `Ragavi-gains` and `Ragavi-vis`.

```
optional arguments:
  -h, --help            show this help message and exit
  -c , --corr           Correlation index or subset to plot Can be specified
                        using normal python slicing syntax i.e "0:5" for
                        0<=corr<5 or "::2" for every 2nd corr or "0" for corr
                        0 or "0,1,3". Default is all.
  --cbin                Size of channel bins over which to average.e.g setting
                        this to 50 will average over every 5 channels
  --chan                Channels to select. Can be specified using syntax i.e
                        "0:5" (exclusive range) or "20" for channel 20 or
                        "10~20" (inclusive range) (same as 10:21) "::10" for
                        every 10th channel or "0,1,3" etc. Default is all.
  -cs , --chunks        Chunk sizes to be applied to the dataset. Can be an
                        integer e.g "1000", or a comma separated string e.g
                        "1000,100,2" for multiple dimensions. The available
                        dimensions are (row, chan, corr) respectively. If an
                        integer, the specified chunk size will be applied to
                        all dimensions. If comma separated string, these chunk
                        sizes will be applied to each dimension respectively.
                        Default is 100,000 in the row axis.
  --cmap                Colour or colour map to use.A list of valid cmap
                        arguments can be found at:
                        https://colorcet.pyviz.org/user_guide/index.html Note
                        that if the argument "colour-axis" is supplied, a
                        categorical colour scheme will be adopted. Default is
                        blue.
  -dc , --data-column   Column from MS to use for data. Default is DATA.
  --ddid                DATA_DESC_ID(s) to select. Can be specified as e.g.
                        "5", "5,6,7", "5~7" (inclusive range), "5:8"
                        (exclusive range), 5:(from 5 to last). Default is all.
  -f , --field          Field ID(s) / NAME(s) to plot. Can be specified as
                        "0", "0,2,4", "0~3" (inclusive range), "0:3"
                        (exclusive range), "3:" (from 3 to last) or using a
                        field name or comma separated field names. Default is
                        all
  -o , --htmlname       Output HTMLfile name
  -ca , --colour-axis   Select column to colourise by. Default is None.
  -ml , --mem-limit     Memory limit per core e.g '1GB' or '128MB'
  -nf, --no-flag        Plot both flagged and unflagged data. Default only
                        plot data that is not flagged.
  -nc , --num-cores     Number of CPU cores to be used by Dask. Default is
                        half of the available cores
  -s , --scan           Scan Number to select. Default is all.
  --taql                TAQL where
  --tbin                Time in seconds over which to average .e.g setting
                        this to 120.0 will average over every 120.0 seconds
  --xmin                Minimum x value to plot
  --xmax                Maximum x value to plot
  --ymin                Minimum y value to plot
  --ymax                Maximum y value to plot

Required arguments:
  --ms [ [ ...]]        Table(s) to plot. Default is None
  -x , --xaxis          x-axis to plot
  -y , --yaxis          Y axis variable to plot
```

FIGURE 4.2: Command line options for `Ragavi-vis`.

4.4 Features

While some features are common between `Ragavi-gains` and `Ragavi-vis`, most have a different implementation or are entirely different.

4.4.1 Data Selection

For `Ragavi-gains`; `ant, field, ddid, t0` and `t1` are the arguments that *may* be used to select antennas, fields, spectral windows, and time range respectively and consequently reduce the size of the data to be plotted. If supplied, only data associated with the chosen quantities will be included in the gain plot; otherwise, all the available data is plotted by default.

Data selection in `Ragavi-vis` can be achieved by supplying values for arguments `chan, ddid, field, scan, corr, xmin, xmax, ymin` and `ymax`. These serve the purpose of channel, spectral window, field, scan, correlation, x-axis range and y-axis range selections respectively. All the data is plotted by default if no selections are made.

Because gain tables and MSs are both CTDS tables (recall Section 3.1), the selection arguments for both `Ragavi-gains` and `Ragavi-vis` are deferred to TAQL (Diepen, 2018) queries. To further reduce the data size, a supplementary argument `taql` exists in both the visibility and gain plotter, which allows a user the choice of making additional selections on the data.

4.4.2 Chunking and Averaging

Chunking, in this context, refers to the process of dividing a large dataset into small partitions known as chunks, which can fit within some given RAM size, to allow computations to be performed on them. The enormous size of visibility data necessitates chunking due to constraints on computational resources. `Ragavi-vis` exposes a chunking interface through the `chunks` command-line option, which allows a user to define the size of chunks that will be loaded into memory at any given time. The chunking back end is provided by `Dask-ms`.

Besides chunking, visibility data size can be reduced by averaging over frequency or over time. The parameters `tbin` and `bin` allow for exactly this. A user may select the number of channels of visibility data to averaged together or a time period in seconds over which the data will be averaged. `Ragavi-vis` then relays these to `Codex-Africanus` (Perkins, 2018), whose averaging tool is used. We note, however, that averaging is only performed after data selections have been made if required. Moreover, for time averaging, only data within the same scan and field are averaged together.

4.4.3 Computing Resource Allocation

`Ragavi-vis` exposes the `mem-limit` and `num-cores` arguments which give a user control over the number of CPU cores to be used and the amount of RAM available to each computation executing in a core. A CPU core is the facility of the CPU that actually performs computations.

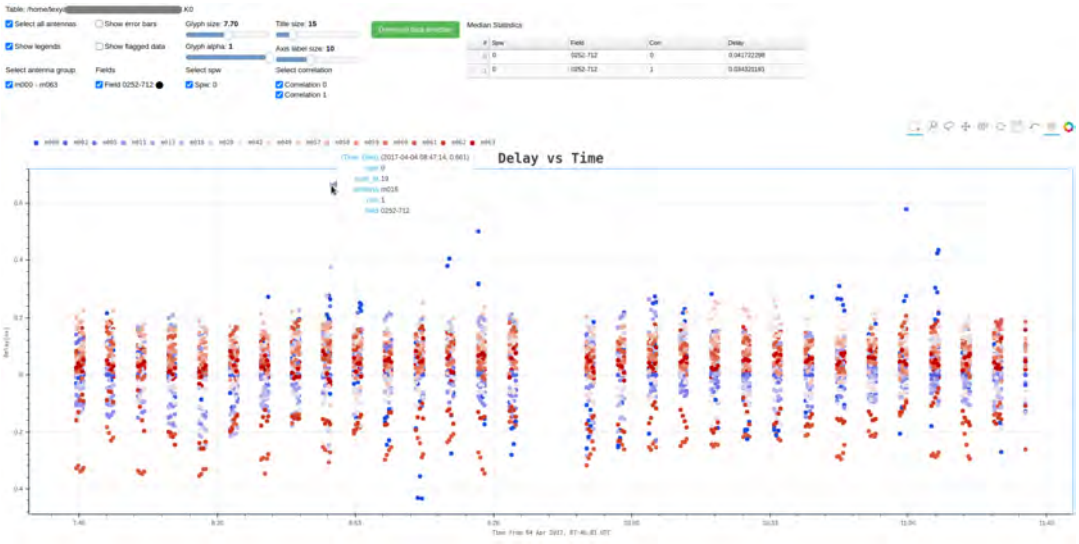
4.4.4 Layouts

Per gain table, `Ragavi-gains` produces a pair of plots placed side by side with each other, whose y-axes correspond to either amplitude and phase (which is the case by default), or the real and imaginary parts (see Fig 4.3 for a demonstration of this layout). The option `doplot` allows a user to chose between the two representations. Points on the plot are colourised based on antenna number, with colours determined by the argument `cmap`.

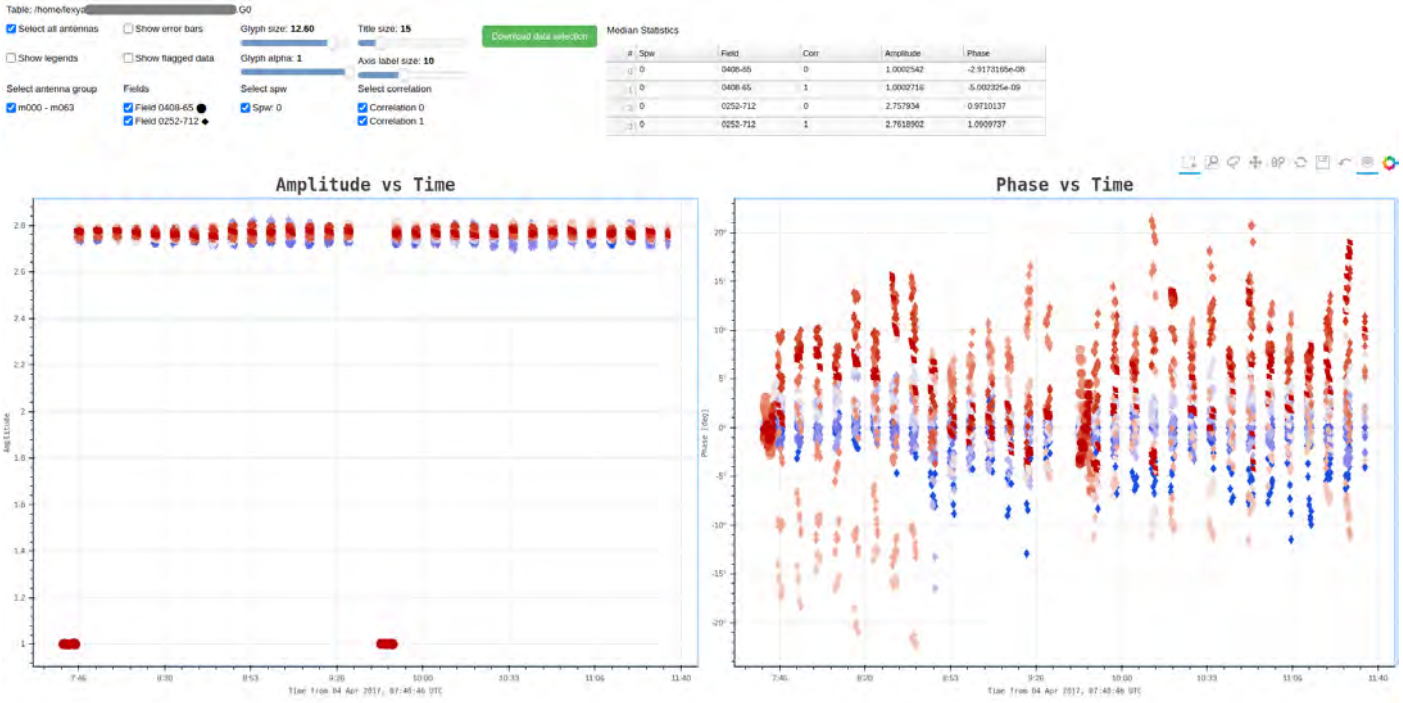
At the bottom of each generated plot, there is a table containing a summary of the median values per spectral window, field and correlation. An exception to the dual plot layout is the output yielded from the delay calibration table, the K-Jones table, whereby only a single plot is produced since the delays are real quantities. The plot sizes are not rigid and scale up or down depending on the size of a user's browser window. It is important to clarify that given a single gain table, `Ragavi-gains` will produce a single output file containing a pair of plots. However, given multiple gain tables at once, as illustrated in Listing 4.1, the output file will consist of a pair of plots for each available table. Therefore, a `Ragavi-gains` output file containing plots of calibration solutions for the entire calibration process can be generated with a single call. Currently, gain tables that can be plotted are (i) G-Jones, (ii) B-Jones, (iii) K-Jones, (iv) D-Jones, (v) F-Jones, all of which have been described in Section 2.3. A summary of supported tables is given in Table 4.1. Refer to Appendix A for more on `Ragavi-gains` layouts.

Table type	x-axis	y1-axis	y2-axis
K-Jones (Delay)	Antenna / Time	Delay (ns)	
B-Jones (Bandpass)	Frequency (GHz) & Channel	A / R	P / I
G-Jones (Complex gain)	Time	A / R	P / I
D-Jones (Leakages)	Frequency (GHz) & Channel	A / R	P / I

TABLE 4.1: A summary of calibration tables currently supported for plotting by `Ragavi-gains` and their expected axes quantities. The x-axis column denotes values found in the x-axes and are common amongst both pairs of plots, while y1 and y2 columns are the y-axes values and have been separated as they are different for each plot. The letter **A, P, R, I** stand for Amplitude, Phase, Real and Imaginary. K-Jones tables yield a single plots and hence, there is no value for the y2-axis column.



(a) A sample delay calibration solution plot layout.



(b) Sample plot showing gain calibration solutions with amplitude, phase against time. `Ragavi-gains` generates a pair of plots for each calibration table, with a summary of median values appended on a table below the plots.

FIGURE 4.3: An illustration of layouts for plots generated by `Ragavi-gains`. For each delay calibration table, a single plot is produced as in 4.3a whereas for all the other calibration tables, a pair of plots is generated with amplitude and phase or real and imaginary on the y-axes as in 4.3b.

By contrast, `Ragavi-vis` only yields a single plot per output file, depending on the users choice of the x and y axes. This plot also scales up and down with the size of the browser or monitor. Furthermore, an option `colour-axis` exists, allowing a user to choose a column in the MS over which to categorise the data to be plotted. These categories manifest as differently coloured data points on the resultant plot. A colour bar is also added to the side to aid in identification, as is illustrated in Fig. 4.5. If

`colour-axis` is not supplied, the default plot's colouring scheme is determined by the density of points at that particular position on the plot. This colour can be changed by supplying the `cmap` argument. Axes values supported by `Ragavi-vis` are shown in Table 4.2.

`xaxis`		`yaxis`	`colour-axis`	
amplitude	real	amplitude	corr	antenna1
antenna1	scan	phase	scan	baseline
antenna2	time	real	spw	
frequency	uvdistance	imaginary	field	
phase	uvwave		chan	

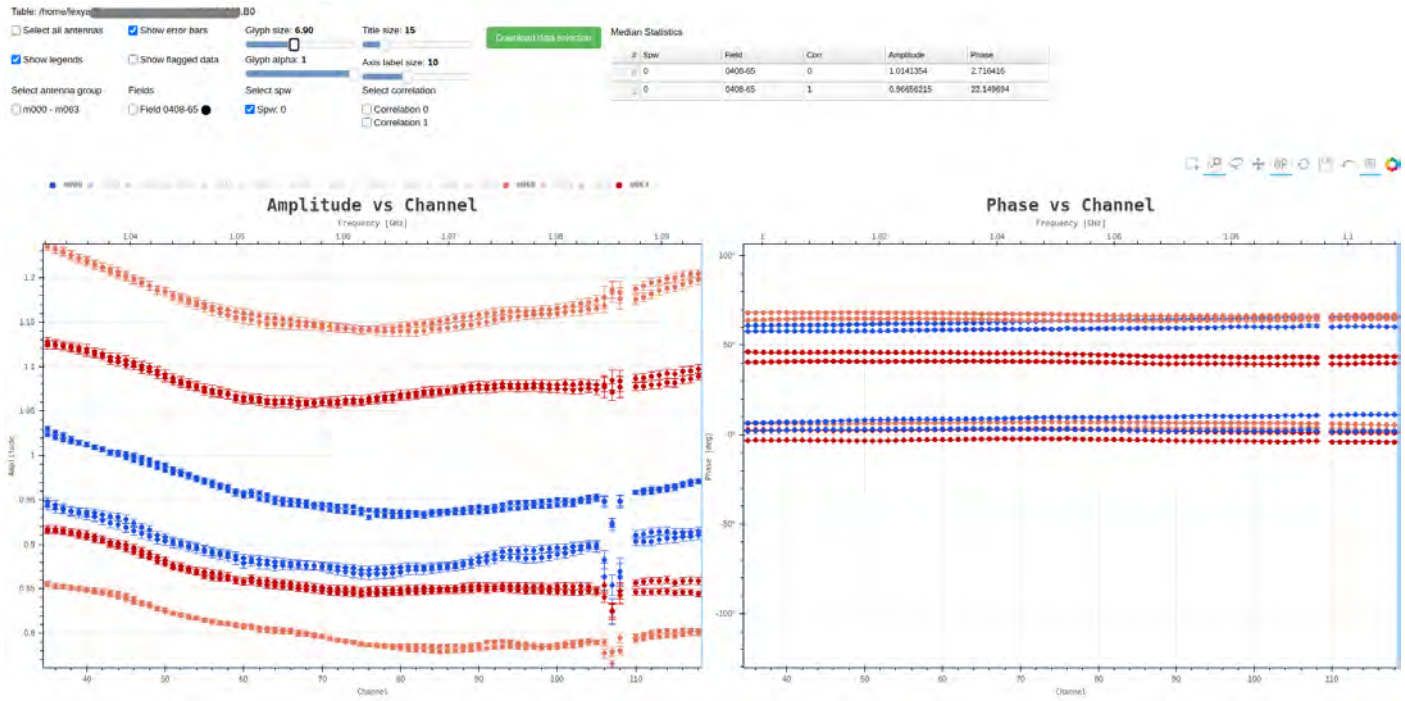
TABLE 4.2: Options supported by the `xaxis, yaxis` and `iterate` arguments of `Ragavi-vis`.

4.4.5 Interactivity

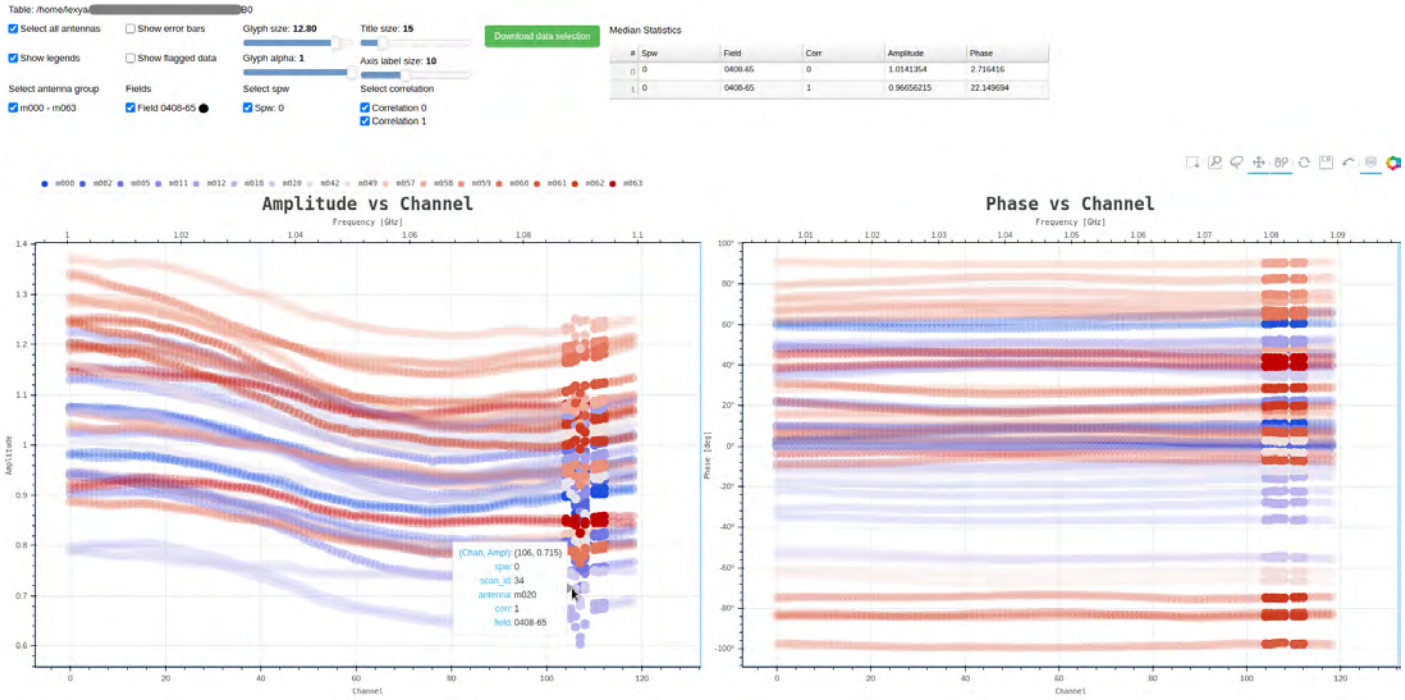
`BokehJS` makes use of `JavaScript`, a scripting language that can execute on the web browser level, to create interactive plots. Besides, it offers an interface to `JavaScript` through which the developer can define and embed custom interactive actions, known as callbacks, which only execute after certain events, such as button clicks and mouse scrolls occur. Taking advantage of this interface, `Ragavi-gains` places a control panel atop the plots which allow the user to select:

- A group of antennas whose data will be made visible. Each group consists of up to 16 antennas. In the case of more, additional selection group selection buttons are generated.
- Fields whose data will be displayed. If there are multiple fields in the MS, `Ragavi-gains` plots data for all the fields denoted by different markers. Markers associated with a field are also displayed here.
- Desired spectral window. All are plotted by default.
- Correlations to be plotted. All correlations are plotted by default.
- Whether or not to show data that is flagged out.
- The opacity and size of the markers.
- Whether plot legends will be visible or not. Legends are hidden by default as they *may* take up a lot of screen space.

Moreover, because `Ragavi-gains` produces a pair of plots for each input gain table, these plots are linked together in a way that allows some actions performed on one plot to be replicated on the other. Panning, zooming and data selection on the plot function in this way. For example, if data is selected in the first plot, the same data points are selected on the second plot, while those not selected are greyed out. Additional information about a point under observation, such as its spectral window number and scan number may be displayed by hovering the mouse pointer over that point. This is useful in identifying outliers in the data. Clicking on the legend of a specific antenna can also make that antenna's data visible or hidden.



(a) A zoomed in plot showing amplitude and phase bandpass calibration solutions whereby solutions of three antennas have been made visible while the rest have been hidden. Solutions of specific antennas can be made visible by clicking on their corresponding legends.



(b) The same plot as in 4.4a zoomed out, where solutions of all the antennas are shown. Selections made in the amplitude plot are also selected in the phase plot. This is possible as the x-axes of the plots are linked. Hover tooltip shown on the amplitude plot provides additional information about a particular point on the plot.

FIGURE 4.4: Demonstration of the interactive features of `Ragavi-gains`.

Contrary to `Ragavi-gains`, the tool panel of visibility plots controls only zooming and panning. This reduction in functionality is due to the fact that while `BokehJS`

plots every single data point it is provided with, `Datashader` aggregates the data and recasts it onto a user-defined grid, thus forming an image, which becomes the plot(see Section 3.2.2) (Bednar, 2016). The aggregation enables the plotting of extremely large datasets (case in point being visibilities) that would otherwise exceed the capabilities of the browser if rendered as individual points, but it breaks the link to individual data points, thus making some of the more advanced interactive features offered by `Ragavi-gains` difficult to implement.

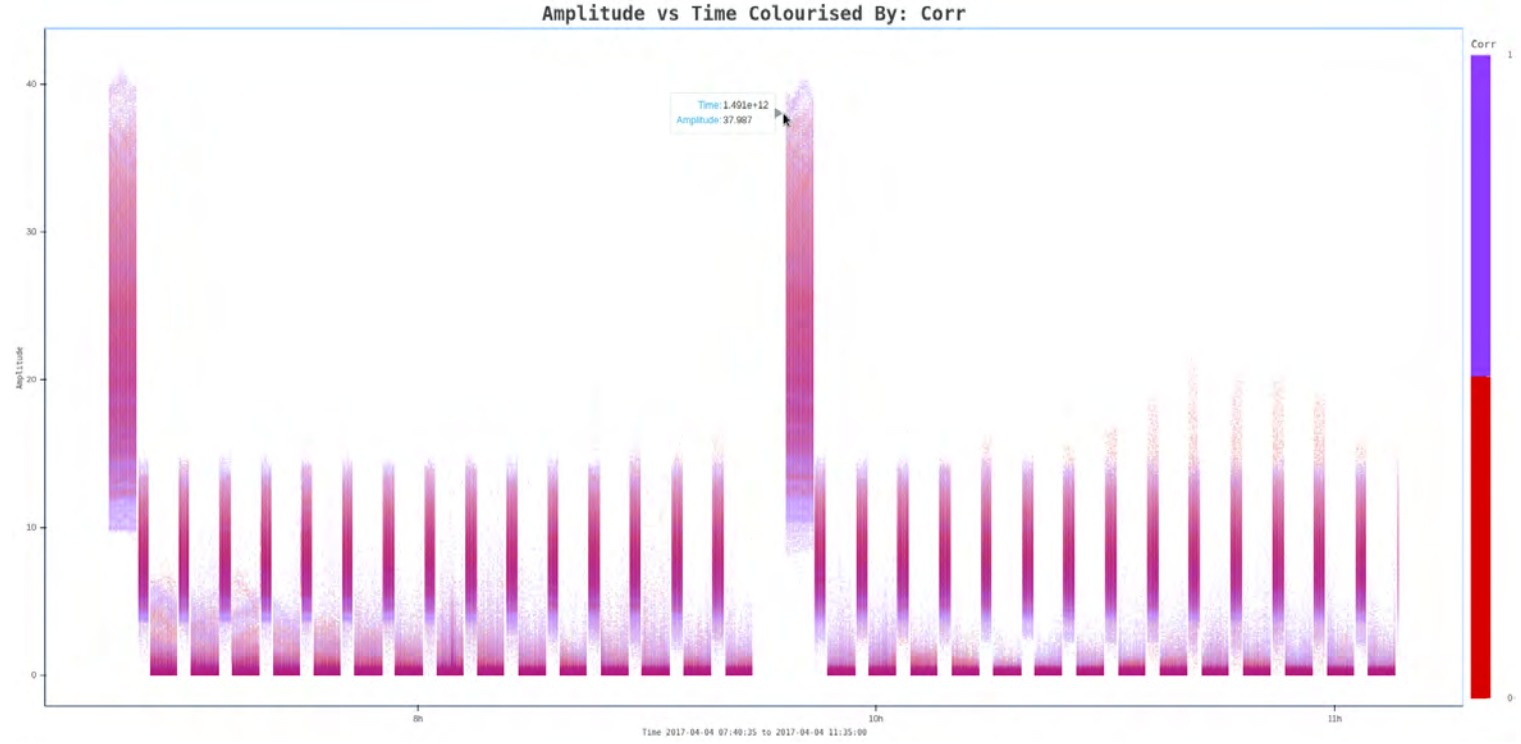


FIGURE 4.5: A plot generated by `Ragavi-vis` of amplitude vs time colourised by correlations.

4.5 Role of `RAGaVI` in the `CARACal` pipeline

`CARACal` is an automated data reduction pipeline (described in Section 3.3.1). Like any pipeline, quality assurance requires the ability to examine its data products (both intermediate and final). It is for this reason that `RAGaVI` exists. `RAGaVI` is incorporated as a `Stimela` cab image (recall `Stimela` cab images in Section 3.3.1) which performs the task of generating the plots. Arguments required for plot generation are extracted from the `CARACal` configuration file. The CLI nature of `RAGaVI` allows for straightforward invocation from within the pipeline. Moreover, the plots produced are interactive, thus enabling the encoding of multiple pieces of useful information in a single plot document. To this extent, `Ragavi-gains` allows plots of different calibration solutions to be contained in a single document. However, their interactivity is not confined to a GUI server.

Interactive plots promote data exploration because functions such as zooming, panning and filtering of visible information on the plot may help in the identification of underlying problems or trends within a given dataset. For example, visualised calibration tables can help determine the quality of calibration solutions, which then

dictates the calibration strategy. Additionally, visualising visibility datasets before the reduction process may assist in distinguishing RFI or bad scans thus guiding the flagging process.

The interactivity of `RAGaVI` plots is not limited to the machine from which they are produced. Therefore, portability of the plots means they are widely distributable, giving collaborators or peers a chance to scrutinise and review the data reduction products, in addition to the science-ready result. They are also embeddable into Jupyter notebooks and web pages and thus can be useful in cases where pipeline reports outlining information about the data reduction are generated in notebook form.

It is worth noting that `RAGaVI` is not restricted for use within the `CARACal` pipeline, but is also usable in other pipelines whose calibration tables or visibility datasets follow the CTDS table system (refer to Section 3.1).

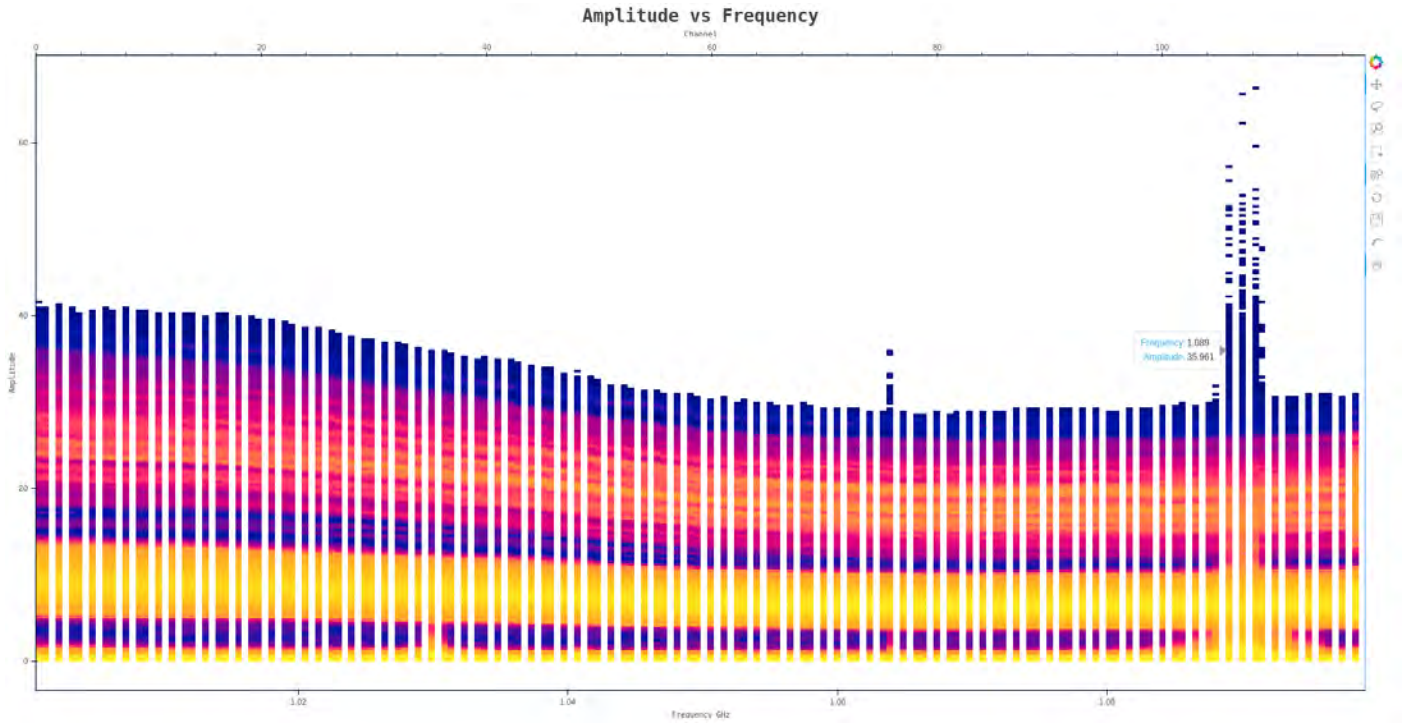
4.6 Example Use-case

Based on the definition of `RAGaVI`'s role in Section 4.5, we demonstrate the practicality of `RAGaVI`. For this example, we inspect some visibility data before and after calibration, as well gain solutions resulting from the calibration process.

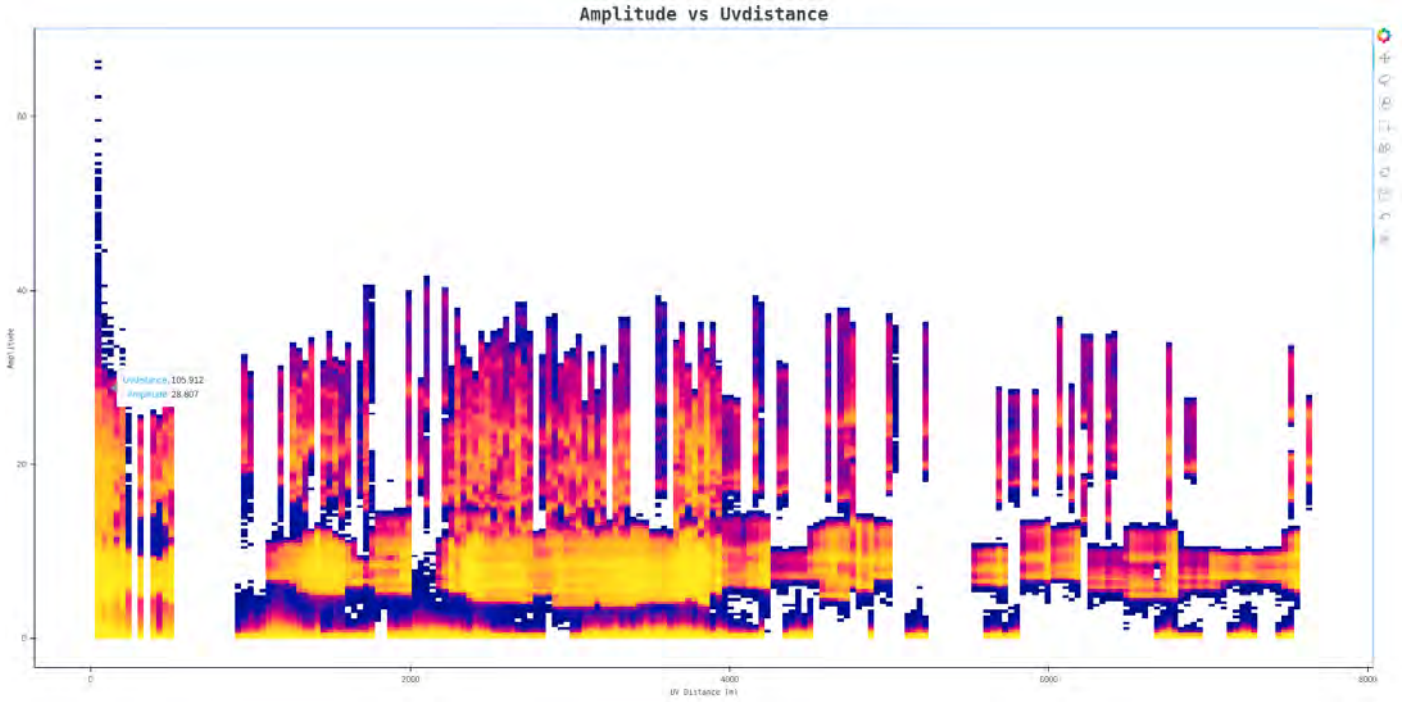
It is worth pointing out that the example presented here is not necessarily accurate and is not meant to present any scientific results. Rather, it only serves to practically exhibit the usefulness of `RAGaVI`. It is also based on a calibration and imaging tutorial demonstrated by B. Hugo and is therefore not my original work.

4.6.1 Before Calibration

One of the first steps towards data calibration is flagging. Therefore, there is a need to inspect one's data beforehand so as to determined which sections of data are to be flagged out. We use `Ragavi-vis` to first make two plots to aid in this. Note that the colours chosen in these example plots are not iterations over any of the data in the MS. Rather, they are indicative of the number of points that fall within a certain area of the plot. Hence, the colour yellow indicates where the are the most number of points, while the navy blue indicates where are the least number of points.



(a) A plot of visibility amplitude against frequency showing some RFI

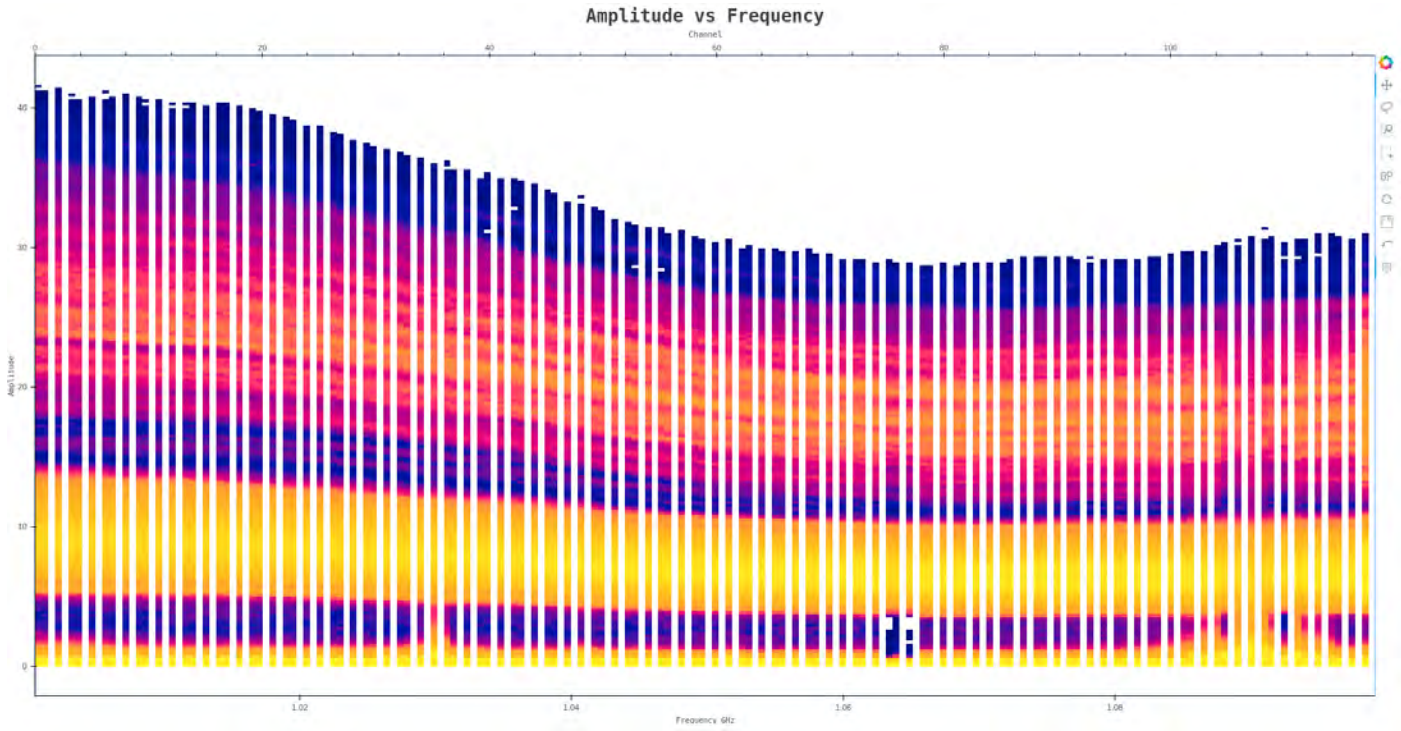


(b) Amplitude against UV Distance (m) showing the UVdistance in which RFI is visible.

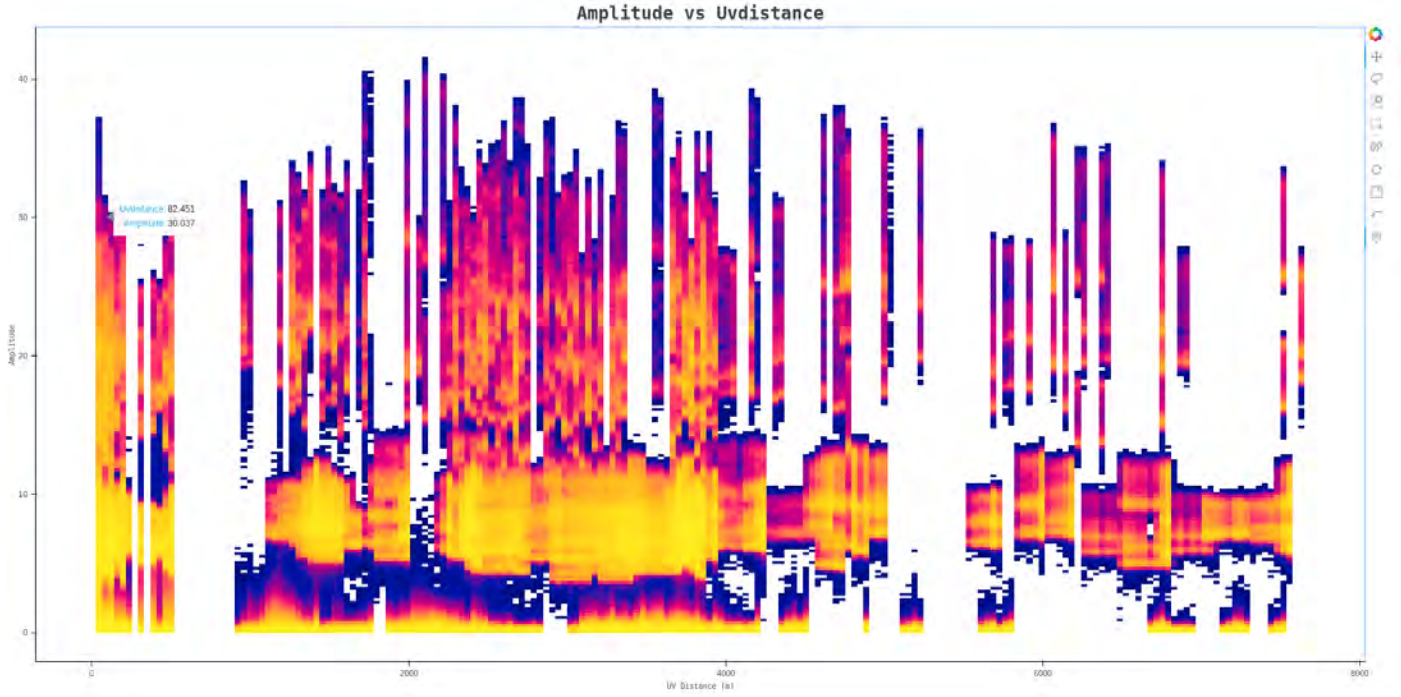
FIGURE 4.6: MS before flagging

`Ragavi-vis`'s hover tooltips in Fig. 4.6 indicate that for this dataset areas close to $\approx$ 1.089 GHz in Fig. 4.6a and around $\approx$ 105m in Fig. 4.6b may contain some bad data that needs to be flagged out. This manifests as the higher amplitude values with

respect to the rest of the data. Knowing approximately which range of frequencies and UV distance to flag, new plots of the same type were generated in Fig. 4.7.



(a) Amplitude vs Frequency after flagging



(b) Amplitude vs UV Distance (m) after flagging

FIGURE 4.7: MS after flagging

Inspection after flagging important as it helps ensure that the process is done appropriately and may immediately reveal if too much or too little data has been removed. Here, it is visible that the bad data has been excluded.

At this point, we examine plots of the raw visibility data for the primary and secondary calibrators. Ideally, it is expected that the calibrator phases are centred around a phase angle of 0^o because the calibrators are located at the phase centre of the target field so that their phases can be assumed to be zero. However, because of corruptions in the data, it is not so as Fig. 4.8 illustrates. Here, we see that the amplitude spreads widely over phase and thus is not centred around zero degrees. Calibration is thus required to correct this.

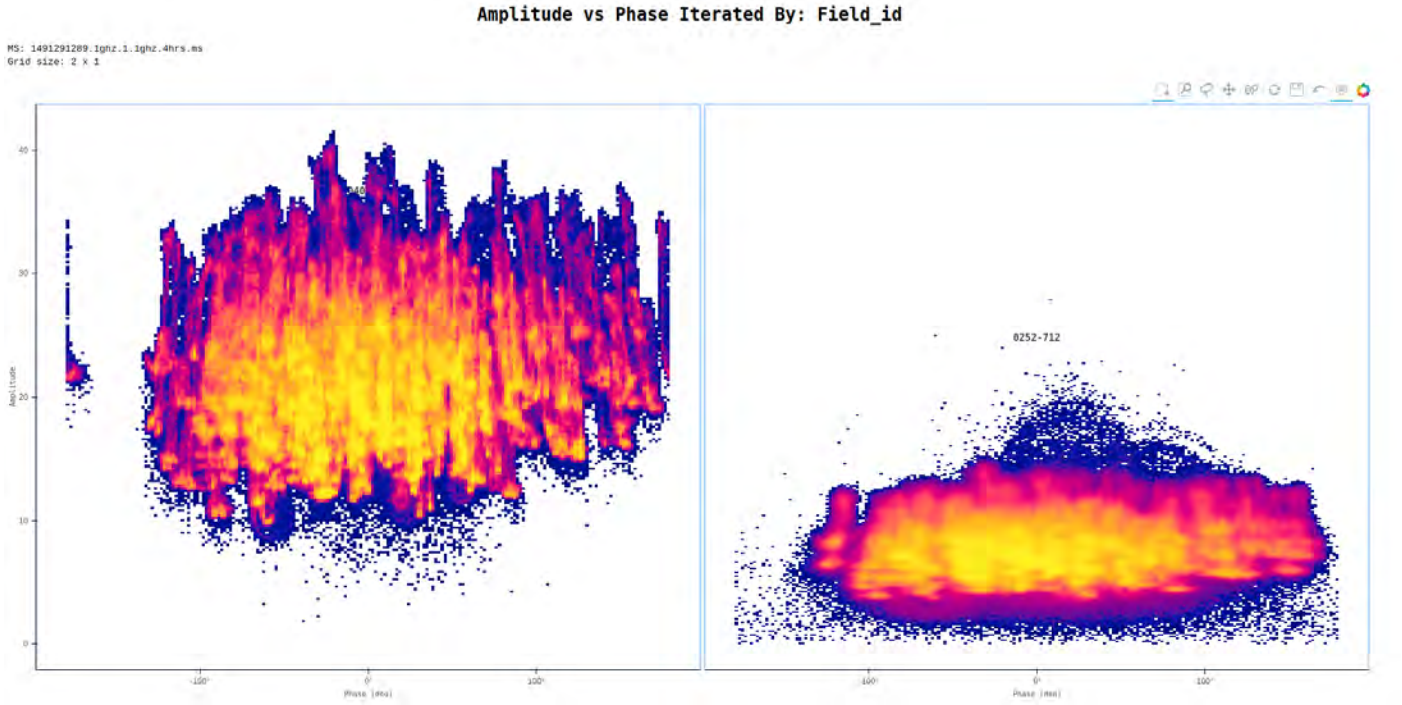
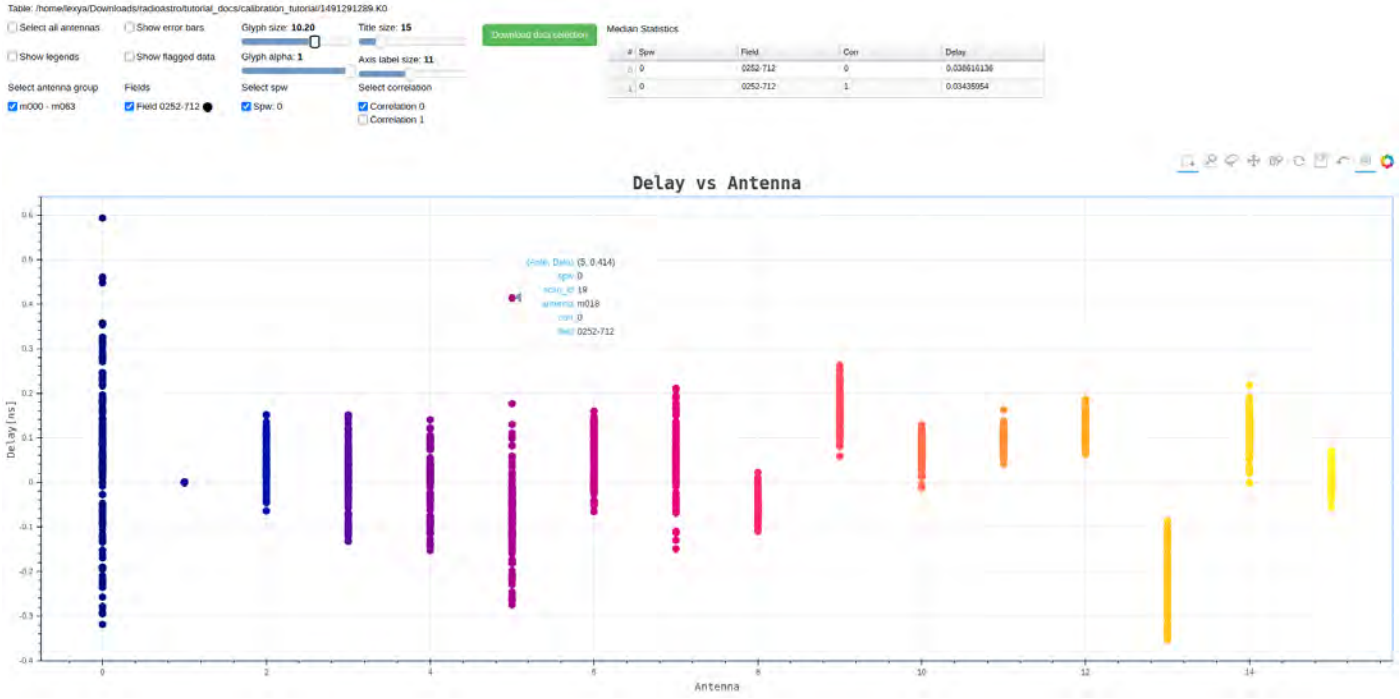


FIGURE 4.8: Amplitude vs phase for the calibrator fields before calibration. The primary (bandpass) calibrator is on the left while the secondary (gain) calibrator is on the right

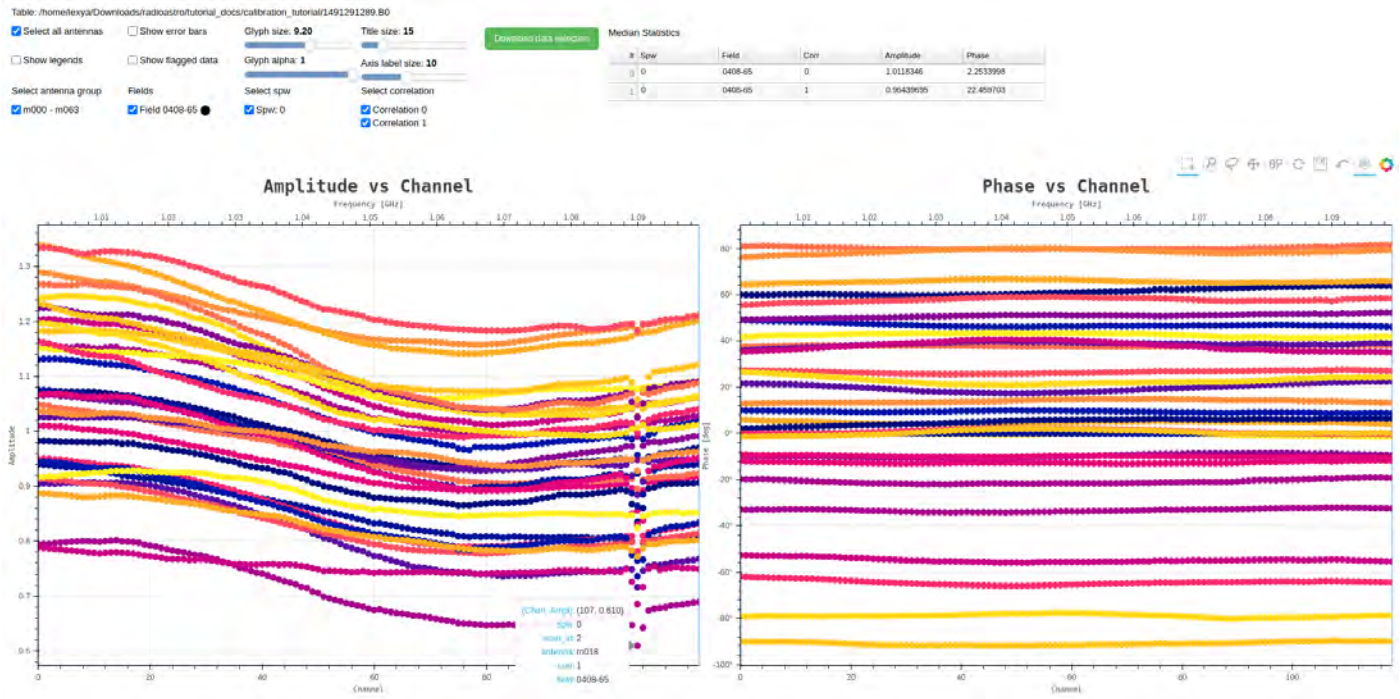
4.6.2 During Calibration

Delay calibration was performed first, followed by bandpass and complex gain calibration. Flux scaling was performed last. Before calibration solutions are applied to the data, we examine them to see if there are any irregularities. The delay, bandpass and flux scales solutions in Fig. 4.9[8].

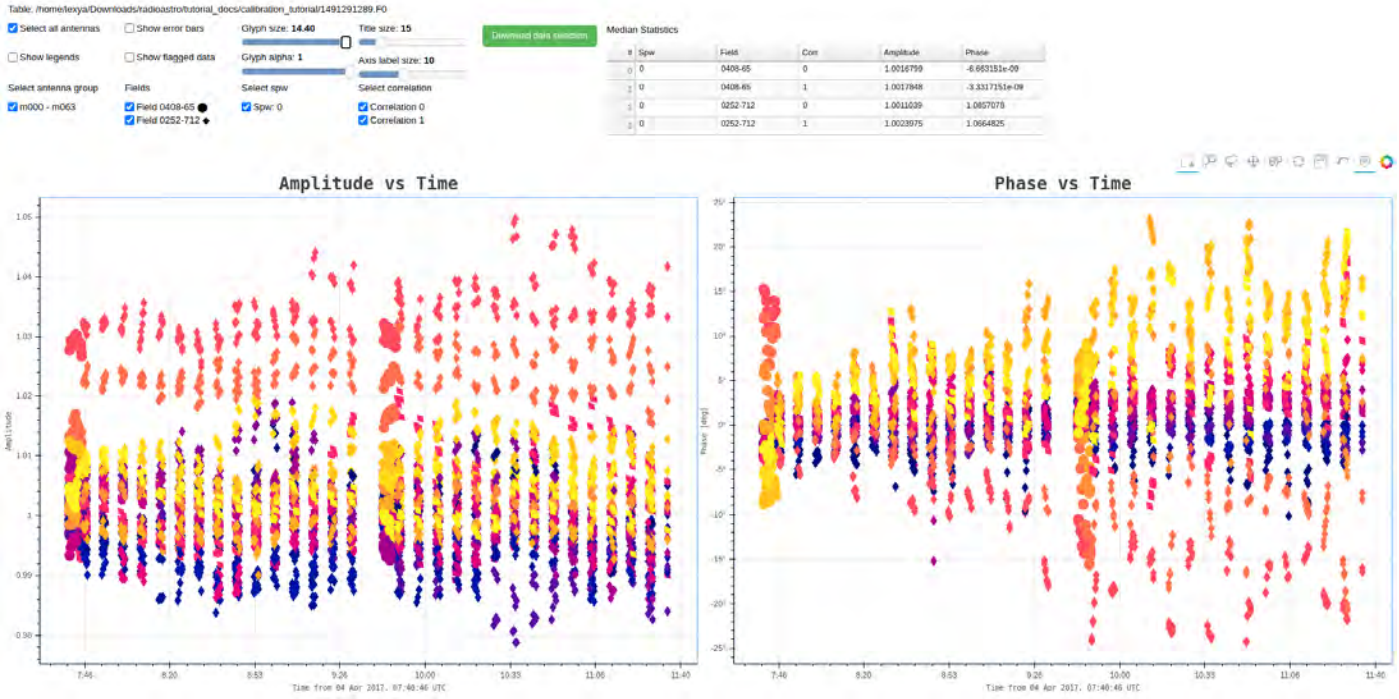
[8]The calibration solutions and visibility data presented in this section are similar to those presented in Section 4.4.



(a) Delay calibration solutions showing delays per antenna.



(b) Bandpass calibration solutions. They show some slight odd behaviour at around 1.09GHz for the amplitude vs channel plot. This could be a result of bad flagging.



(c) Flux scaled solutions for both the calibrator field. The primary calibrator is marked by the circular shape, while the secondary calibrator is marked by the diamond shape.

FIGURE 4.9: K, B and F Calibration solutions.

4.6.3 After Calibration

Generally, the solutions do not show any major deviations. For instance, all the delay values (Fig. 4.9a) lie within a few nanoseconds for all the antennas, while the bandpass solutions all surround the value of one (Fig. 4.9b). There is some odd behaviour at around channel 107 (1.09 GHz) and this could be and indication that flagging was not done properly. The fluxes where correctly scaled as we see in (Fig. 4.9c) that the primary and secondary calibrators' values are almost similar. Seeing as the gains were proper, they were then applied to the MS to calibrate the data. A look at the amplitude versus phase plot in Fig. 4.10 for the calibrated data shows that the amplitudes are now centred around zero. While there is still some spread, it is reduced as compared to that shown in Fig. 4.8.

This dataset can be further calibrated and flagged to improve its quality before it is imaged, but, because calibration is not the purview of this thesis, we shall not go into the details. Nevertheless, it is evident that visualisation tools such as `RAGaVI` play a significant role and are indispensable during the process of flagging and calibration. It is important that these tools are fast, portable and rich so as to accelerate the generation of new scientific knowledge.

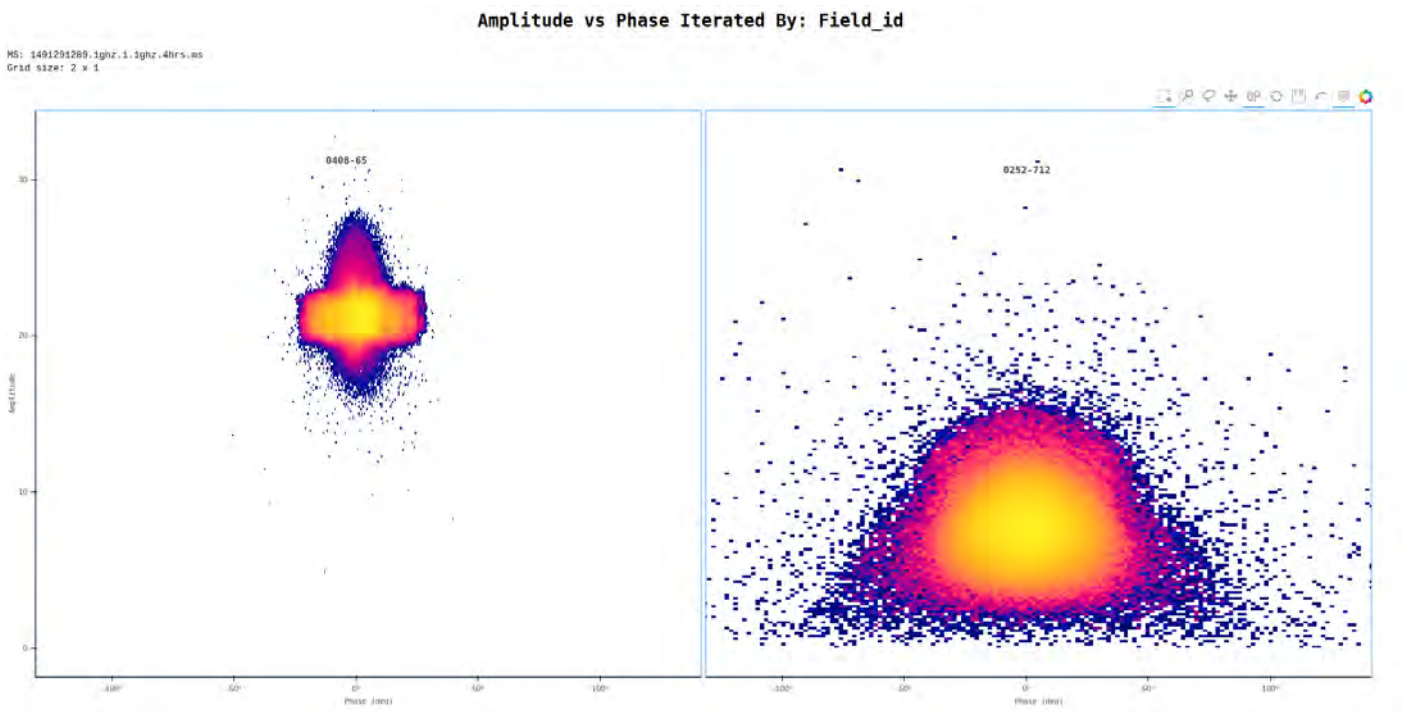


FIGURE 4.10: Amplitude vs phase for the calibrator fields after calibration

Chapter 5

Ragavi vs CASA Plotms

This chapter compares `RAGaVI` to `Plotms` in terms of their interface layout and performance. For performance metrics, we primarily compare `Ragavi-vis` to `Plotms`, owing to the size of visibility data. Metrics used are therefore related to plotting time and RAM usage and their relationship with increasing MS size. We discuss the test conditions, test metrics and results, and then interpret them.

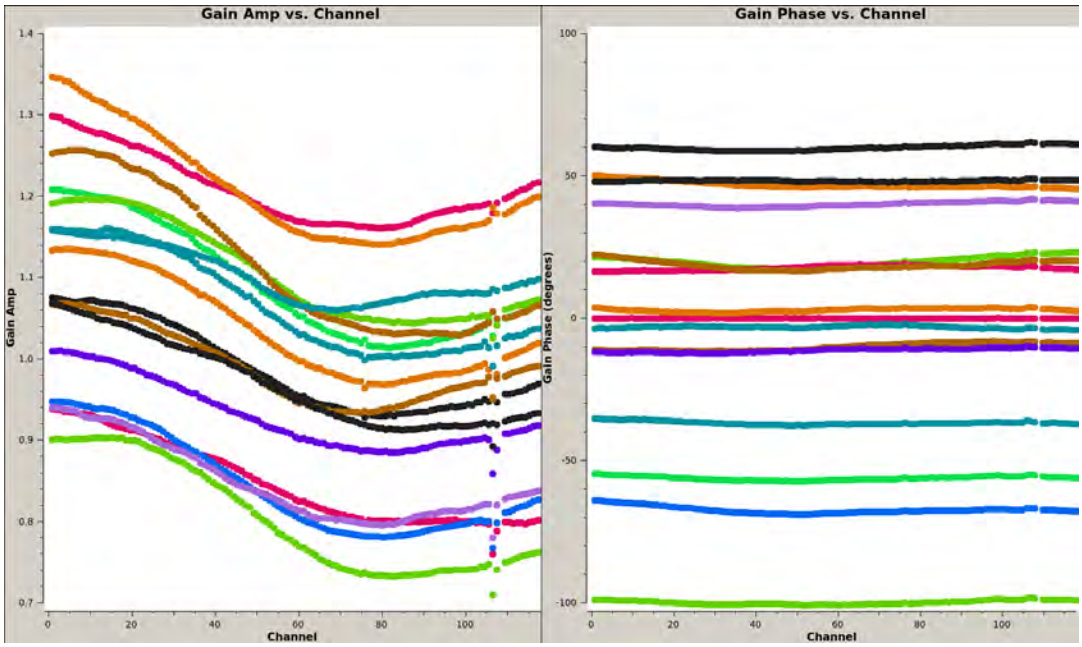
Test Benchmarks

We run both `RAGaVI` and `CASA Plotms` v.5.4.1-31 on an isolated machine with 512GB of RAM and one AMD EPYC 7702P CPU with 64 physical cores each clocked at 2.0GHz. The operating system is Ubuntu 18.04.4 LTS. We first compare the interfaces of both tools, showing the differences between the plots generated by `Plotms` and `Ragavi-vis`. We then contrast the performance of both tools in terms of the time taken to generate plots (plotting time) and their peak RAM usage during execution with an increasing number of data points to be plotted. Time profiling was performed using a simple timing function (i.e. "wall clock time"). The `memory-profiler`[1] package was used to profile the RAM. Each test was repeated approximately five times to improve the accuracy of the results.

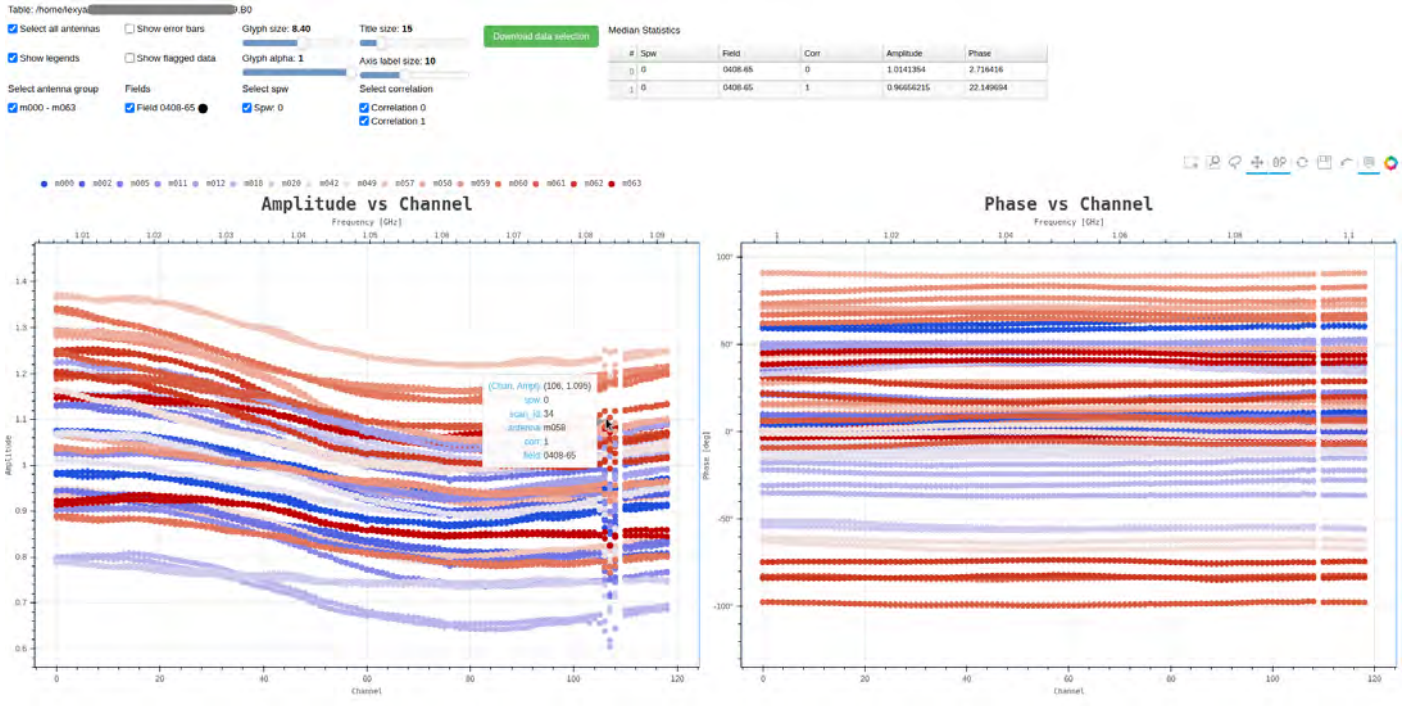
5.1 Interface

Fig. 5.1 contrasts between plots produced by `Ragavi-gains` and `Plotms`. While `Ragavi-gains` generates plots that can be modified as shown in Fig. 5.1b, `Plotms` produces static plots which constrain a user to a single view of the plotted data, as shown in Fig. 5.1a. `Plotms` does offer recourse to interactivity through the use of a GUI, as illustrated in Fig. 5.2a, which shows a plot of amplitude against time for an MS. The GUI supports multiple useful data modification and selection options which give users control over the visible aspects of data. Although useful, this may be somewhat prohibitive for use within automated data pipelines as multiple static plots may be required to capture the different data aspects. Furthermore, these plots are only interactive for as long as the GUI is active. While `Ragavi-vis` does not provide as many options as `Plotms`, it provides an avenue for data exploration without the need for re-plotting it. Plots rendered can be stored for later analysis, with the same interactivity level available offline.

[1] https://github.com/pythonprofilers/memory_profiler



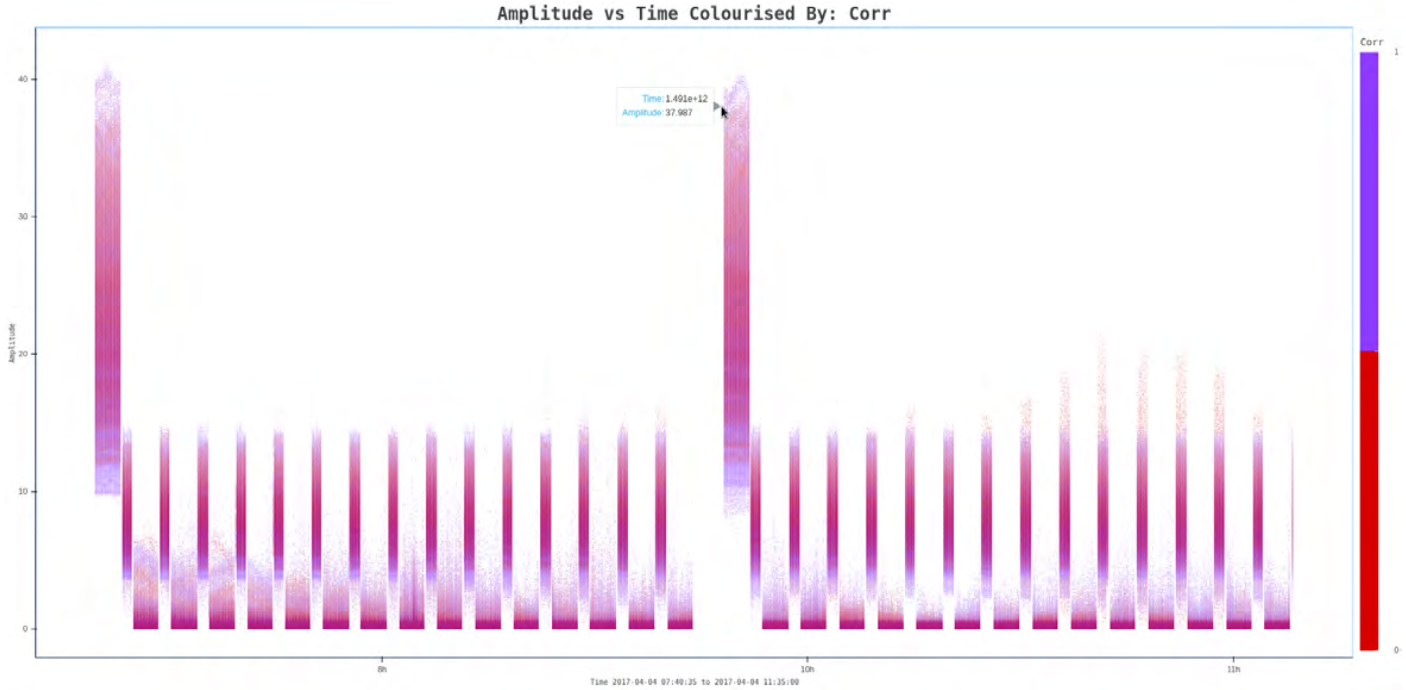
(a) Bandpass solutions plotted using `Plotms`. This is a static plot (PNG format) which is limited to a single view of the bandpass calibration solutions. To modify the view, the data must be plotted again.



(b) The same plot as that in Fig. 5.1a generated by `Ragavi-gains`. Data view can easily be changed using the interactive buttons atop the plots. Additional information about a specific data point such as the scan in which it belongs and the spectral window, can be revealed by hovering about that point.

FIGURE 5.1: A layout comparison of bandpass calibration solutions plot from `Plotms` and `Ragavi-gains`.

(a) The `Plotms` GUI featuring an Amplitude vs Time plot. This GUI exposes numerous options for selecting and modifying the visible amounts of data points.



(b) A similar plot to Fig. 5.2a, `Ragavi-vis` does not require a GUI. The plot generated is self contained and remains interactive.

FIGURE 5.2: Comparing `Plotms` and `Ragavi-vis`.

5.2 Performance

To quantify the performance of `Ragavi-vis`, we first establish the combination of plot axes that:

1. Takes the least amount of time to plot, thus implying a lesser computational load,

2. Takes the most time consuming and thus, presumably, the most computing resource-intensive.

This is done using `Plotms`, as it is the *de facto* standard visibility data plotting tool. Our test dataset has a size of ≈ 0.94GB and contains ≈ 45.44 million visibility data points. No data selections or averaging was done.

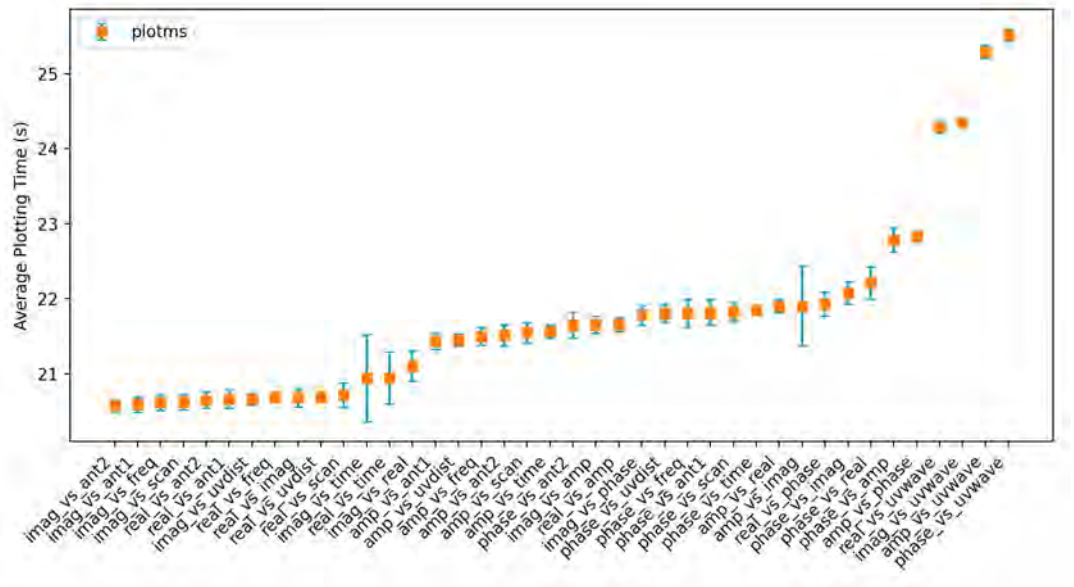


FIGURE 5.3: Average time taken by `Plotms` to plot visibility data for various combinations of x and y-axes. For context, we limited the axes to those available in `Ragavi-vis`. Plots involving UV distance in wavelengths appear to take the most time and are thus assumed to be the most resource intensive.

Based on the results shown in Fig. 5.3, we observe that the most time-consuming plots are those where the x-axis is UV distance in wavelengths (UVWave) and more specifically, the plot of phase against UVWave. This is an expected result because the data being plotted, in this case, must be computed and thus, is not readily available in the dataset. On the lower end, there are multiple plots that take approximately the same amount of time to be generated as seen in Fig. 5.3. Our plot of choice here is that of real vs imaginary data, because it is a fairly common diagnostic plot. Moreover, it is expected to require little computational effort as complex visibility data is intrinsically stored as a real and imaginary part. Consequently, we based all further performance tests on these two plot types. Note that we consider "plotting time" as the time from when the plotting tool is launched until the moment when a PNG or HTML file containing the plots is generated.

We utilise the same dataset to investigate and compare the time taken by `Ragavi-vis` and `Plotms` to generate the visibility amplitude, phase, and real data against UV distance in wavelengths and the imaginary data. This yields the results illustrated in Fig. 5.4.

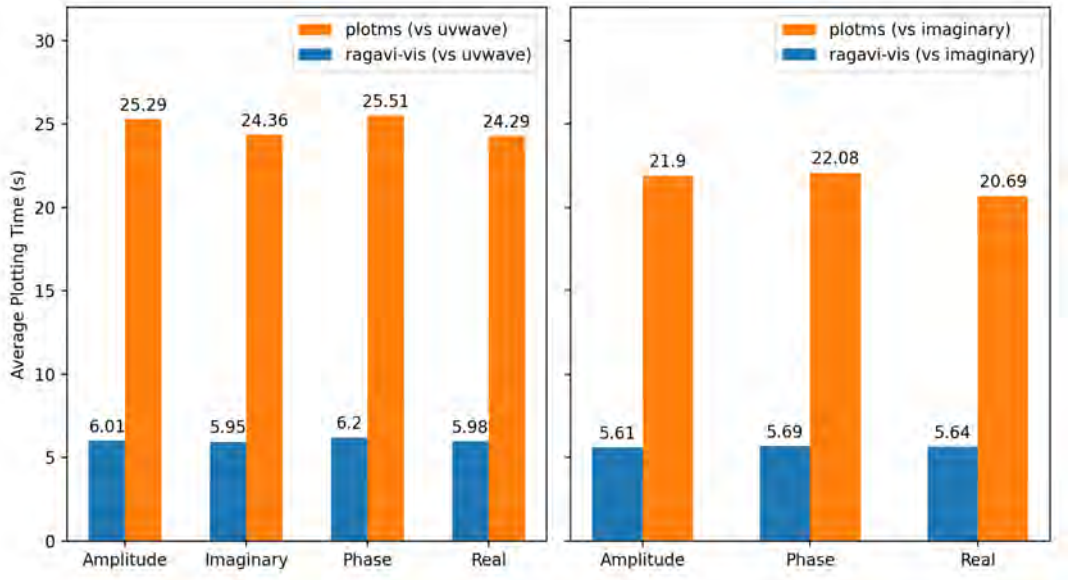


FIGURE 5.4: Comparison of average times taken by `Ragavi-vis` and `Plotms` to generate plots for amplitude, phase, real and imaginary vs UV Distance in wavelengths, and amplitude, phase and real vs imaginary data for a single dataset.

We see that `Ragavi-vis` is consistently faster by a factor of 4 or more. This performance may be attributed to two primary factors directly linked to `Ragavi-vis`'s back-ends. Firstly, as discussed in Section 4.2, `Ragavi-vis` makes use of `Dask` to perform computations. Here, data is divided into smaller sizes that can fit in RAM. `Dask`, in the case of `Ragavi-vis`, introduces parallelism through the use of threads, which allow for the processing of the multiple chunks available in RAM simultaneously, thereby improving the speed of execution (see Section 3.1.2). Secondly, the plotting back-end for `Ragavi-vis`, `Datashader`, also utilises `Dask` to perform parallel computations (see Section 3.2.2). This allows `Ragavi-vis` to better exploit the multiple cores available on our test machine.

A test dataset of 45 million data points is relatively small in comparison to a typical MeerKAT observation, which may contain billions of visibility points. While averaging or data selection may be performed to reduce the number of data points and increase the speed at which plots are generated, it may occasionally be desirable to plot a dataset in its entirety, in order to reveal its structure or problematic aspects. Hence the need for a visualisation tool that performs such plotting within a reasonable time frame.

We thus test the scalability of both `Ragavi-vis` and `Plotms` with increasing dataset sizes, i.e. increasing number of visibility data points. For this test, we consider the plot of the visibility phase versus UV distance in wavelengths as it provides an average upper limit of the plotting times, and was deemed to be the most time and resource

consuming due to the amount of computations before the data is ready for plotting (see Fig. 5.3). We also consider the plot of real versus imaginary visibility data, which provides an average lower limit of the plotting times and can give an insight into the actual plotting time without significant computations.

In our testing, we have noted that, if left to its own devices, `Ragavi-vis` could use a lot more RAM and CPU cores than `Plotms`. This behaviour is driven by the `Dask` scheduler. In comparison, `Plotms`' documentation suggests that it uses only three major threads for execution: a caching thread, a plotting thread and a plot export thread. Caching refers to the process of storing data temporarily within the RAM to improve the throughput of data processing. These threads all operate sequentially and are thus directly dependent on the output of each other. `Plotms` also has a maximum cache memory storage of 25 GB which causes the limitation on the number of visibility data points that can be plotted simultaneously (around 4.29 billion). In other words, `Plotms` will not exploit more than three cores and 25 GB of RAM, even if the machine offers considerably more (as our test machine did).

It could be said that `Dask`, and therefore `Ragavi-vis`, is simply better at exploiting RAM and CPU. In most real-life scenarios these resources are very likely to be available: large visibility datasets tend to live on large machines since small machines are anyway unable to cope with the standard steps of calibration and imaging at these data sizes. However, in the interest of a fair comparison, we tried to provide two test cases for `Ragavi-vis`. In the first case ("unconstrained"), `Ragavi-vis` was allowed to use as much RAM and as many CPU cores as available. In the second case ("constrained"), we attempt to constrain `Ragavi-vis` to the same computational resources that `Plotms` uses. For the constrained case, we limit `Ragavi-vis` to three cores, each executing a single thread. We also allocate 8 GB to each thread, constituting a total of 24 GB, which is close to `Plotms`' caching memory limit.

5.2.1 Performance Without MS Averaging

The results in this Section are based on the said test setup without any averaging being performed on the MS before plotting was done. Table 5.1 shows the details concerning the datasets used for the scaling tests, as well as results from `Plotms` and the unconstrained case of `Ragavi-vis`. The same results are illustrated by Fig. 5.5.

Tool	Dataset	Points (B)	Plot	Plotting time (min)
`Ragavi-vis`	1	≈ 0.0454	phase vs uvwave	0.092 ± 0.000
			real vs imag	0.081 ± 0.001
	2	≈ 0.2164	phase vs uvwave	0.197 ± 0.001
			real vs imag	0.176 ± 0.001
	3	≈ 1.7769	phase vs uvwave	4.985 ± 0.022
			real vs imag	4.741 ± 0.004
	4	≈ 3.4340	phase vs uvwave	1.311 ± 0.117
			real vs imag	0.751 ± 0.005
	5	≈ 4.2134	phase vs uvwave	4.583 ± 0.160
			real vs imag	4.105 ± 0.005
	6	≈ 4.3104	phase vs uvwave	4.470 ± 0.020
			real vs imag	4.106 ± 0.025
	7	≈ 4.4181	phase vs uvwave	4.501 ± 0.008
			real vs imag	4.112 ± 0.013
`Plotms`	1	≈ 0.0454	phase vs uvwave	0.424 ± 0.002
			real vs imag	0.347 ± 0.003
	2	≈ 0.2164	phase vs uvwave	1.438 ± 0.010
			real vs imag	1.182 ± 0.009
	3	≈ 1.7769	phase vs uvwave	11.319 ± 0.042
			real vs imag	8.410 ± 0.053
	4	≈ 3.4340	phase vs uvwave	19.090 ± 0.134
			real vs imag	14.224 ± 0.124
	5	≈ 4.2134	phase vs uvwave	23.544 ± 0.086
			real vs imag	17.096 ± 0.120
	6	≈ 4.3104		above limits

TABLE 5.1: Number of visibility data points (in billions) available in different datasets and the time it takes to plot those points in both `Ragavi-vis` and `Plotms`.

As expected, Fig. 5.5 shows an increase in the plotting times for both tools with increasing dataset sizes. Additionally, we see that plotting real versus imaginary data takes a shorter time in comparison to plotting phase against UVWave. However, `Ragavi-vis` is faster in all cases. Note that there is a limit to the number of points that `Plotms` can plot in a single instance, which is approximately 4.2949 billion visibility points. We see that `Ragavi-vis` is well able to plot above this limit (our testing stopped at ≈ 4.41 billion points), and is still retaining the same plotting speed.

The plotting time of real against imaginary data is lower than that of phase against UVWave is because visibility data in the former case is readily available in the real and imaginary form and thus does not need to undergo any processing to arrive at. By contrast, more computation is required to arrive at the UV Distance in wavelengths, as it necessitates calculations of the available UV data over all the available frequencies. This is further escalated by the calculation of the phase angle from visibility data.

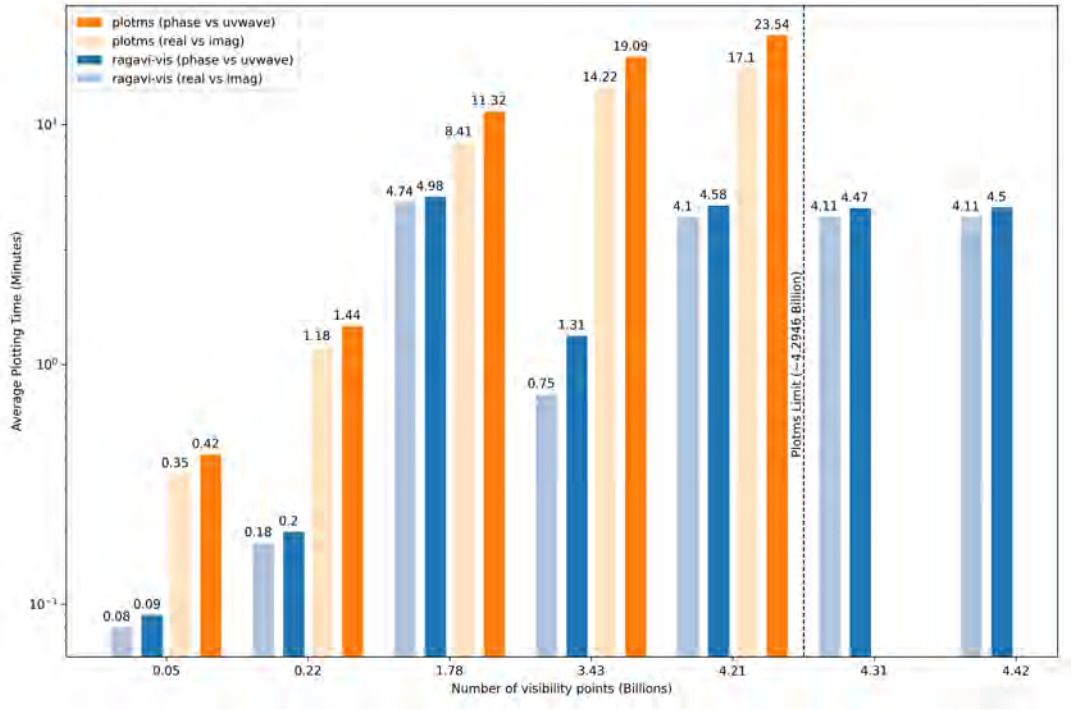


FIGURE 5.5: Comparison of average plotting times in minutes for `Ragavi-vis` case one and `Plotms` with increasing MS size.

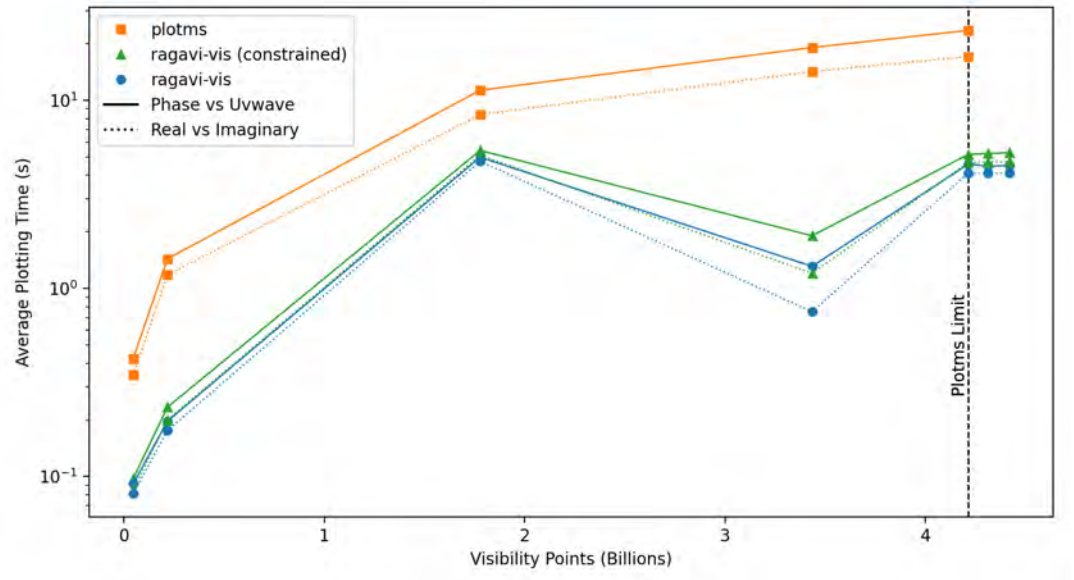


FIGURE 5.6: Representing the same information in Fig. 5.5 and including results for the constrained `Ragavi-vis` (triangle marker), in which an attempt is made to constrain `Ragavi-vis` the same computational resources as `Plotms`. In this plot, there are more data points associated with `Ragavi-vis` than `Plotms` because `Plotms` can plot a maximum of approximately 4.295 billion points.

A further look at the more comprehensive plot including the constrained case for `Ragavi-vis`, Fig. 5.6, reveals interesting and unexpected outcomes. To begin with, the difference between plotting times for the constrained and unconstrained cases for

`Ragavi-vis` seems to be very slight. Secondly, plotting a dataset of 2 billion points takes a longer time than plotting a dataset of 3.5 billion points. We suspect that the the first outcome may be an effect of increased communication between `Dask`'s scheduler and its workers. This result suggests that having more computational resources and increased parallelism does not guarantee better performance. The second result is a consequence of having more chunks in the smaller dataset than in the larger one. During testing, the same standard chunk size (5000 in the row dimension while the channel and correlation dimensions remained the same) was used for all the datasets. However, the smaller dataset had a more rows and fewer channels (8884650 x 100 x 2), while the situation was reversed for the larger dataset (419195 x 4096 x2) (see Section 3.1 for the structure of MS data columns). Because chunking mainly occurs in the row axis, the smaller dataset contained a significantly larger number of chunks, hence, taking more plotting time.

We also speculate that the minimal difference in plotting time between phase against UVWave and real against imaginary data for `Ragavi-vis` is because `Dask` chunks data in both cases the same way, eventually resulting in the same number of partitions. The difference in the plotting times between the two plot types could indicative of computational time.

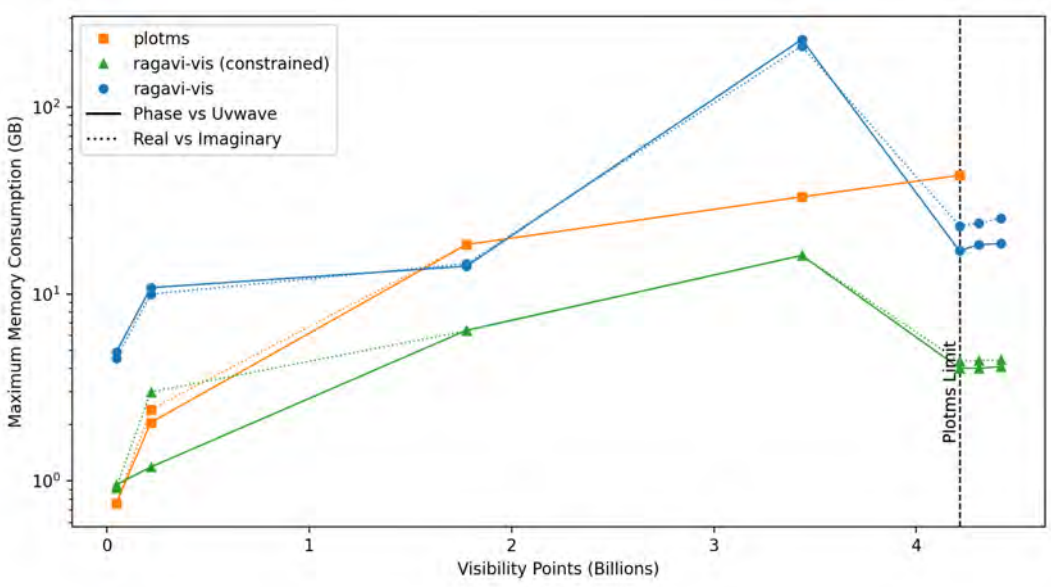


FIGURE 5.7: A comparison of the maximum RAM used by both `Plotms` and `Ragavi-vis` for an increasing number of visibility points (MS size). The line with triangular markers demonstrates the case where an attempt was made to constrain `Ragavi-vis` to the same resources as `Plotms` while that with circular markers is for the case where `Ragavi-vis` has access to all the resources.

On the other hand, an investigation on the RAM usage of `Ragavi-vis` versus `Plotms` also demonstrates some interesting results, as illustrated in Fig. 5.7. We see that there is an extreme peak in RAM usage for one dataset in the unconstrained case of `Ragavi-vis`. This is the same dataset for which there is a dip in plotting time in Fig 5.6. The reason is that the chunk size in this dataset was larger than that of its immediate neighbours because it had a greater number of channels (it had 4096 channels while its neighbours had 100 and 400 channels). This peak is also

visible in the constrained `Ragavi-vis`, revealing that larger chunk sizes will certainly result in more utilisation of the available RAM. The constrained case shows lower RAM consumption (with respect to `Plotms` and the unconstrained `Ragavi-vis`) but demonstrates an almost similar profile to that of the unconstrained case, while `Plotms`' RAM usage steadily increases with increasing dataset sizes. This figure depicts that with some fine-tuning, it is possible with `Ragavi-vis` to control the amount of RAM used even for large datasets.

5.2.2 Performance With MS Averaging

The final phase of `Ragavi-vis`'s assessment entailed testing its performance while averaging is enabled. Data averaging reduces the data size and thus, a reduction in plotting time and memory usage can be expected. For this test, all datasets were averaged over every 100 seconds and every 30 frequency channels.

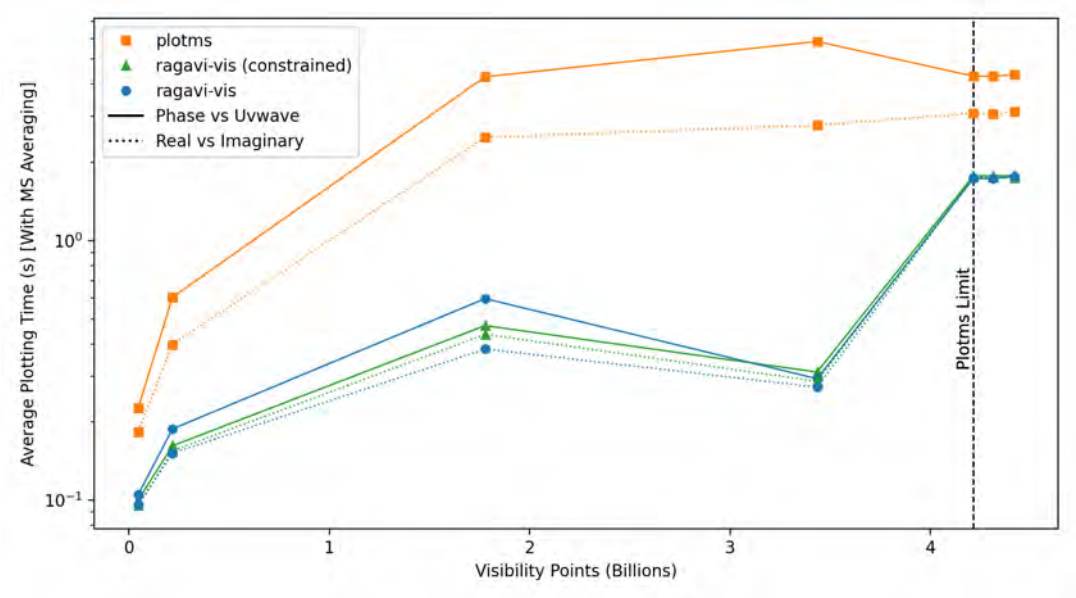


FIGURE 5.8: A comparison of the average plotting time for `Plotms` and `Ragavi-vis` (case one and two) when averaging is active for an increasing number of visibility points.

Fig. 5.8 indicates that in this regime, `Ragavi-vis` is still faster in comparison to `Plotms`, with both the constrained and unconstrained tests showing almost similar results. Once more, we ascribe this behaviour to the parallelised nature of `Dask`'s operations. As intimated in Section 4.4, averaging is tasked to `Codex-Africanus` which in turn similarly uses `Dask` to perform its functions. The reason for the dip with the plots concerning `Ragavi-vis` in Fig 5.8 is the same as that explained in the unaveraged case. It should be mentioned that, as expected in this context, `Plotms` is capable of plotting above its data points' ceiling due to the obvious reduction in data sizes, hence the availability of data after its limit.

A probe into the memory usage reveals that, despite the rise with increasing MS sizes, the memory consumption of `Ragavi-vis` considerably reduces when averaging is in use. Constraining `Ragavi-vis` leads to a further decrease in the RAM usage. Nonetheless, `Ragavi-vis`'s RAM consumption remains higher than that of `Plotms` as demonstrated by Fig. 5.9. We also highlight that the type of plot (phase versus

uvwave or real versus imaginary) appears to have little to no effect on the RAM usage for both tools.

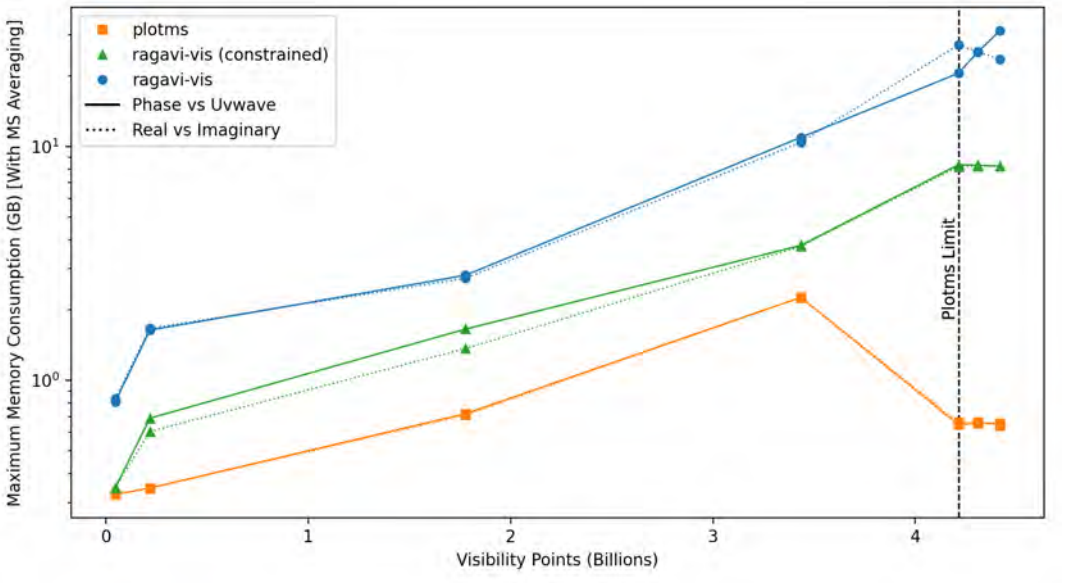


FIGURE 5.9: A comparison of the maximum RAM used by both `Plotms` and `Ragavi-vis` (case one and two) when averaging is active for an increasing number of visibility points.

5.2.3 Discussion

In general, `Ragavi-vis` emerges as a significantly faster plotting tool as compared to `Plotms`. However the price to pay for the speed so far, and in some cases, has been an increased RAM utilisation, which is directly linked to the number of cores in use at any given point. We have demonstrated that in certain instances, increasing computational resources available to `Ragavi-vis` does not necessarily result in the intended speed up, and rather has the opposite effect or no effect at all. Therefore, it is evident that the combination of the number of cores (thus workers), memory availed to a CPU core, and data chunk size have a significant influence on the achieved parallelism, and thus performance, of `Ragavi-vis`. Therefore, their choices, though tricky, may require careful consideration from the user in order to achieve the intended efficient utilisation of resources utilisation, in addition to maintaining a reasonable time frame for the generation of plots.

While it is true that `Ragavi-vis` can still be streamlined, we have shown that it is possible to generate plots faster than the available alternative tools even with exceedingly large datasets. It is also worth noting the size of output files remains constant regardless of the MS size because the former only depends on the configured canvas size. Currently, each file is approximately 11.6 MB in size.

5.3 Challenges

A major challenge encountered during the development of `Ragavi-vis` is that: not providing explicit x- and y-axis ranges beforehand also increases the time taken to

generate a plot. This is especially true for large datasets, as `Dask` will need to perform a full additional pass through the data in order to calculate the maximum and minimum values and thus define the ranges. `Ragavi-vis` partially alleviates this by allowing user-defined ranges upfront – the user is encouraged to make use of this feature for very large datasets.

Furthermore, due to data aggregation, it is challenging to add more interactive features to stand-alone visibility plots. Implementing additional levels of interactivity may require running a web server to interact with `Datashader`. The latter solution would be somewhat half-way between the offline, self-contained nature of the HTML plots currently generated by `RAGaVI`, and the online X11 sessions required by legacy tools such as `Plotms`, but may well be worth pursuing in future work.

Chapter 6

Conclusion and Future Work

Data inspection is an integral aspect of quality assurance in the calibration and imaging process. We discussed in Chapter 2 that EM radiation arriving at a radio interferometer is corrupted by the electronics of the receiving antennas and the atmosphere. The calibration process plays a pivotal role in mitigating these effects and enabling an accurate sky reconstruction. Errors in the calibration process can lead to loss of data quality, and thus a failure to achieve fundamental science goals. Such errors can be detected via the inspection of calibration solutions, as well as visibility data before and after the application of calibration solutions (recall Section 2.3.2).

Chapter 4 therefore presents `RAGaVI`, a gains and visibilities inspection tool that generates interactive plots in HTML format, given a target MS or calibration table, while making use of freely available Python packages (see Chapter 3). Due to the interactive property of the plots it generates, it is possible to embed multiple views to the same dataset in a single plot, hence reducing the number of static plots required to expose interesting data features, and reducing the need for frequent plot regeneration. For example, `Ragavi-gains` makes it possible to contain all the calibration solutions from the 1GC calibration process (recall Section 2.3) in a single HTML document. Moreover, interactive plots promote data exploration, thus hastening the process of fault detection. `RAGaVI` is well suited for use within data reduction pipelines, as it has a command-line interface and thus does not rely on the launch of an X11 session and a GUI to provide its interactive service. `RAGaVI` plots can be shared as self-contained files that retain their interactivity. `RAGaVI` is already used within the `CARACal` pipeline (Section 4.5) to generate plots of calibration solutions.

In Chapter 5, we compared `RAGaVI` to the `CASA Plotms` tool and showed `Ragavi-vis`'s ability to generate plots of visibility data faster than `Plotms`. The tests within this comparison also yielded some interesting aspects of resource usage between the two tools, clearly showing some trade-offs concerning the speed of execution and computing resource management. Here, we demonstrated that while `Ragavi-vis` is capable of generating plots faster, it can have a significantly larger RAM footprint than `Plotms`, which we attributed to the need for `Dask` to load multiple chunks of data into RAM simultaneously. We also showed that `Ragavi-vis` performs better than `Plotms` if averaging is enabled.

An effort has already been made towards improving `Ragavi-vis`'s memory and CPU usage by allowing a user to set the amount of memory associated to each CPU core, the size of data chunks and the number of CPU cores to use. However, without the right combination of the three, `Ragavi-vis` may perform sub-optimally and sometimes, erratically. Further work on automating these settings is required. We also aim to

address and optimise the speed of the aggregation of an input dataset onto a regular-sized grid. `Datashader` provides a recourse in (*Datashader Documentation: Spatial Indexing* 2019), which is yet to be looked into. Finally, it is important to emphasise that `RAGaVI` is still under active development[1]. Hence, we aim to increase interactive features as well as support a larger number of plots for `Ragavi-vis`.

[1]The author is continuing her graduate studies in radio astronomy, and remains an active member of the `CARACal` team.

Appendix A

Notes on `Ragavi-gains` Layout

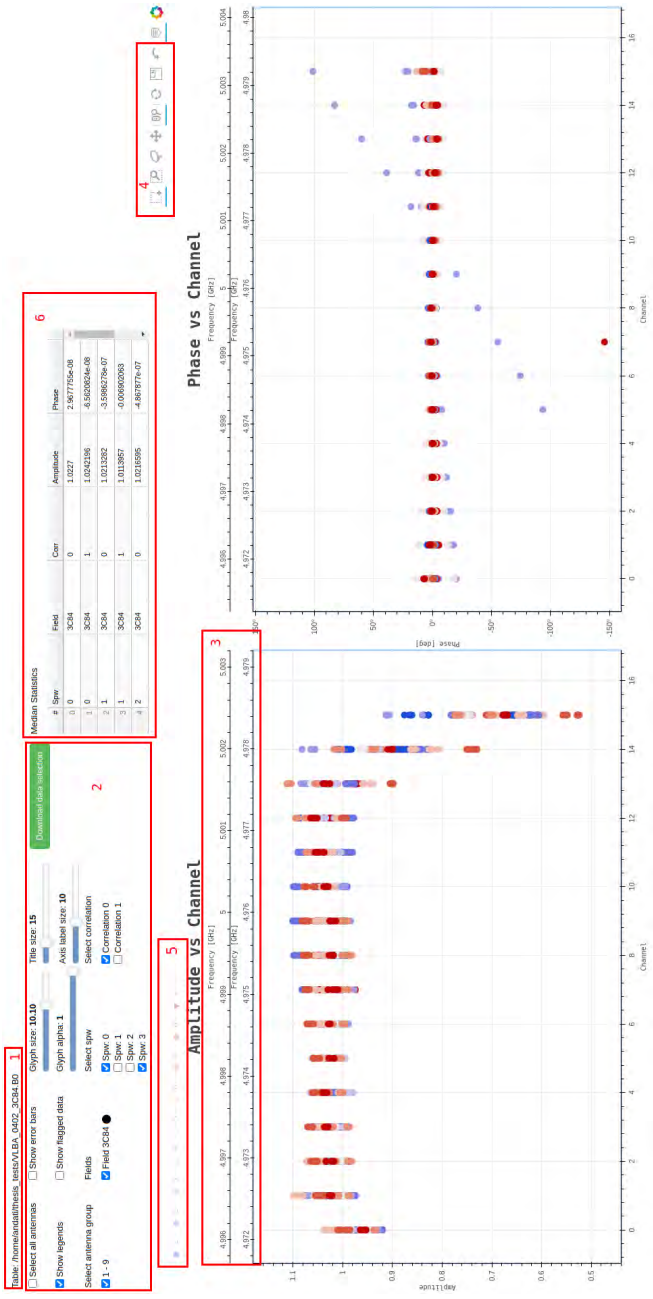


FIGURE A.1: A bandpass plot with multiple SPWs, highlighting of `Ragavi-gains` plot interface.

No		Description
1	Table name	Name of the plotted gain table.
2	Data selection panel	Widgets to select data to be made visible on the plot.
3	Extra axis	Contain frequencies corresponding to the available channels.
4	Plot tools	Allow for zooming, panning, selection highlighting etc on the plot.
5	Clickable legends	Click to make data for a specific antenna visible or hidden.
6	Median stats	Some statistics on the median values for all the antennas per SPW, field and correlation.

A.0.1 Notes on 2

By default, all correlations, fields and SPWs will be selected on the the widgets here. However at any given moment, to make a plot visible, there **must** be a selection made on **each of the three** selectors, otherwise, the plotting canvas will remain empty. The symbol beside the field selectors denote the symbols used in the plot to represent that field.

To increase the size of the plots, it is recommended to hide the antenna legends on the plot by selecting None on the *showing legends* drop-down menu. This is especially useful in the case of more than 16 antennas, as each legend bar holds a maximum of 16 antennas (See section A.0.4).

A.0.2 Notes on 3

The extra frequency axis only appears in bandpass and leakage solution plots. It contains frequencies corresponding to the available channels in a spectral window. In the case of multiple spectral windows, for instance 4 in this case, four frequency axes are shown with the plot. Axes for a particular spectral window are only made visible if that spectral window has been selected at the *spw* check-boxes. Hence, in Figure A.1, only 2 extra axes appear because two spectral windows have been selected.

A.0.3 Notes on 4

The use of these widgets on one plot induces the same reaction on its twin plot. That said, those plots are linked via the x-axis and therefore, actions performed here will be effected with respect to the x-axis. For example, panning over the y-axis in one plot will not have an effect on the other. Panning over the x-axis will.

Although not recommended, the save widget here can be used to download a PNG version of the plot. However, the downloaded version will be in a fixed resolution, and will not contain any of the interactive features. Furthermore, in the case of two plots on the canvas like in Figure A.1, two plots will be downloaded separately.

A.0.4 Notes on 5

The maximum allowable number of antennas on this legend strip is 16. This means that, in case of $>$ 16 antennas, more legend strips will be made available. This will

have a significant effect on the size of the visible plot and as such, it is recommended to hide all the legend strips if they are not in use. Selecting an antenna from the legend strip of one plot will lead to a selection of the same antenna on the twin plot.

A.0.5 General notes

When `Ragavi-gains` first generates a plot, the only data that will be visible will be that of the first antenna available. This is done to control the amount of elements visible in case of a large number of antennas. The antenna selection widgets in Section A.0.1 can then be used to select antenna data to be made visible in batches of 16.

It is also worth noting that zooming on the plots can be done in two ways: i) Zooming within the plot, which can be achieved by selecting the wheel zoom widget and scrolling up and down from inside either of the plots. ii) Zooming on the axes (outside the plot) which can be achieved by moving the mouse pointer to either the x-axis or y-axis and scrolling up and down. The latter can be especially useful in cases whereby there are multiple fields within the plot, having values that differ by a factor. If one of the fields is unselected, the plot does not automatically rescale and therefore, this type of zooming used to correct that.

Bibliography

Bednar, James (2016). *Big Data Visualization with Datashader (Whitepaper).*

Burke, Bernard F, Francis Graham-Smith, and Peter N Wilkinson (2019). *An introduction to radio astronomy.* Cambridge University Press.

Clark, BG (1999). "Coherence in radio astronomy". In: *Synthesis Imaging in Radio Astronomy II.* Vol. 180, p. 1.

Condon, J. J. and S. M. Ransom (2016). *Essential Radio Astronomy.*

Cotton, WD (1999). "Polarization in interferometry". In: *Synthesis Imaging in Radio Astronomy II.* Vol. 180, p. 111.

Datashader Documentation: Spatial Indexing (2019). https://datashader.org/user_guide/Points.html. [Online; accessed Nov-2019].

De Gasperin, F et al. (2019). "Systematic effects in LOFAR data: A unified calibration strategy". In: *Astronomy & Astrophysics* 622, A5.

Dean, Jeffrey and Sanjay Ghemawat (2008). "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1, pp. 107–113.

Diepen, Ger van and Allen Farris (1994). "AIPS++ Table Data System". In: *Astronomical Data Analysis Software and Systems III.* Vol. 61, p. 417.

Diepen, GNJ van (2015). "Casacore Table Data System and its use in the MeasurementSet". In: *Astronomy and Computing* 12, pp. 174–180.

— (2018). *Table query language. Casacore notes 199.* http://casacore.github.io/casacore-notes/199.html.

Farnes, Jamie et al. (2018). "Science Pipelines for the Square Kilometre Array". In: *Galaxies* 6.4, p. 120.

Fomalont, Ed B and Richard A Perley (1999). "Calibration and editing". In: *Synthesis imaging in radio astronomy II.* Vol. 180, p. 79.

Hamaker, JP, JD Bregman, and RJ Sault (1996). "Understanding radio polarimetry. I. Mathematical foundations". In: *Astronomy and Astrophysics Supplement Series* 117.1, pp. 137–147.

Heald, George et al. (2018). "Low Frequency Radio Astronomy and the LOFAR Observatory". In: *Springer International Publishing, doi* 10, pp. 978–3.

Jansky, Karl G (1932). "Directional studies of atmospherics at high frequencies". In: *Proceedings of the Institute of Radio Engineers* 20.12, pp. 1920–1932.

— (1933). "Electrical disturbances apparently of extraterrestrial origin". In: *Proceedings of the Institute of Radio Engineers* 21.10, pp. 1387–1398.

Janssen, Michael et al. (2019). "rPICARD: A CASA-based calibration pipeline for VLBI data". In: *arXiv preprint arXiv:1905.01905.*

Jones, R Clark (1941). "A new calculus for the treatment of optical systemsi. description and discussion of the calculus". In: *Josa* 31.7, pp. 488–493.

Kemball, AJ and MH Wieringa (2000). "MeasurementSet definition version 2.0". In: *URL: http://casa. nrao. edu/Memos/229. html*.

Li, Di and Zhichen Pan (2016). "The five-hundred-meter aperture spherical radio telescope project". In: *Radio Science* 51.7, pp. 1060–1064.

Maccagni, FM et al. (2019). "The flickering nuclear activity of Fornax A". In: *Astronomy and Astrophysics*.

Makhathini, Sphesihle (2017). "Advanced Radio Interferometric Simulation and Data Reduction Techniques". PhD thesis. Rhodes University.

McMullin, JP, K Golap, and ST Myers (2004). "The AIPS++ Project". In: *Astronomical Data Analysis Software and Systems (ADASS) XIII*. Vol. 314, p. 468.

McMullin J. P., Waters B. Schiebel D. Young W. Golap K. ed. R. A. Shaw F. Hill D. J. Bell (2007). *Astronomical Data Analysis Software and Systems XVI (ASP Conf. Ser. 376)*, p. 127.

Michałowski, Michał J et al. (2019). "On the nature of the unusual transient AT 2018cow from HI observations of its host galaxy". In: *Astronomy and Astrophysics*.

Molenaar, Gijs and Oleg Smirnov (2018). "Kern". In: *Astronomy and Computing* 24, pp. 45–51.

Noordam, Jan E and Oleg M Smirnov (2010). "The MeqTrees software system and its use for third-generation calibration of radio interferometers". In: *Astronomy & Astrophysics* 524, A61.

Norris, Ray P (2010). "Data challenges for next-generation radio telescopes". In: *2010 Sixth IEEE International Conference on e-Science Workshops*. IEEE, pp. 21–24.

Pearson, Timothy J (1999). "Non-Imaging Data Analysis". In: *Synthesis Imaging in Radio Astronomy II*. Vol. 180, p. 335.

Perkins, S (2018). *Codex-Africanus Documentation*. https://codex-africanus.readthedocs.io/en/latest/.

Rocklin, Matthew (2015). "Dask: Parallel computation with blocked algorithms and task scheduling". In: *Proceedings of the 14th python in science conference*. 130-136. Citeseer.

Roshi, D Anish et al. (2019). "Astro2020 Activities and Projects White Paper: Arecibo Observatory in the Next Decade". In: *arXiv preprint arXiv:1907.06052*.

Ruiz, JE et al. (2012). "Digital Science: reproducibility and visibility in Astronomy". In: *Highlights of Spanish Astrophysics VII, Proceedings of the X Scientific Meeting of the Spanish Astronomical Society (SEA)*.

Smirnov, Oleg M (2011a). "Revisiting the radio interferometer measurement equation-I. A full-sky Jones formalism". In: *Astronomy & Astrophysics* 527, A106.

— (2011b). "Revisiting the radio interferometer measurement equation-II. Calibration and direction-dependent effects". In: *Astronomy & Astrophysics* 527, A107.

Smirnov, Oleg M (2011c). "Revisiting the radio interferometer measurement equation-III. Addressing direction-dependent effects in 21 cm WSRT observations of 3C 147". In: *Astronomy & Astrophysics* 527, A108.

Thompson, A Richard (1999). "Fundamentals of radio interferometry". In: *Synthesis Imaging in Radio Astronomy II*. Vol. 180, p. 11.

Wrobel, JM and RC Walker (1999). "Sensitivity". In: *Synthesis Imaging in Radio Astronomy II*. Vol. 180, p. 171.