Astronomy in the SKA Era: SKA-low Mini Project

CRSiD: tmb76

University of Cambridge

Contents

Gain Calibration of a SKA-low station

0.1 Introduction

In this mini project, an algorithm for the retrieval of gain solutions for a single SKA-low station is implemented. One SKA1-low station comprises 256 antennae that cover a frequency range of 50-350 MHz. The gain retrieval algorithm is used in order to calibrate the stations, to account for known instrumental effects which occur in the analog chain: Low-Noise Amplifiers (LNA), cables, and other analog components [2]. Because it can be summarised into a series of linear transformations of the input signal, the gain calibration can be done with a single complex-valued gain for each antenna.

0.2 Calibration Problem

In this short section, equations defining the problem of calibration are listed. First, the voltage that is the input of the analog chain, for antenna i, frequency f, and feed port p, is given by:

$$v_{i,p} = G_i \mathbf{F}_{i,p}(\theta, \phi) \cdot \mathbf{E}(\theta, \phi) \tag{1}$$

where θ and ϕ are the zenith and azimuth angle, respectively. $\mathbf{E}$ is the incident electric field from the sky. $\mathbf{F}_{i,p}$ is the Embedded Element Pattern (EEP) of antenna i for feed port p. The feed ports are a result of the array antennae having two feeds, which are the cables or conductors that connect the antenna to the receivers [5] [8]. And finally, G_i is the complex gain for antenna i.

Then comes the visibilities which are the time cross-correlation of the voltage signals from two antennae, i and j, and feed port p [5]. There are the measured visibilities $R_{ij,p}$ which can be modeled as $R_{ij,p} = G_i G_j^* M_{ij,p}$ where $M_{ij,p}$ are model visibilities and they are given by:

$$M_{ij,p} = \int\int (\mathbf{F}_{i,p}(\theta,\phi) \cdot \mathbf{F}^*_{j,p}(\theta,\phi)) T_b(\theta,\phi) e^{-j\mathbf{k}\cdot(\mathbf{r_i}-\mathbf{r_j})} \sin\theta d\theta d\phi \tag{2}$$

Where $\mathbf{F}^*_{j,p}(\theta,\phi)$ is the complex conjugate of the EEP of antenna j for feed port p, $T_b(\theta,\phi)$ is the brightness temperature of the sky, and $\mathbf{r}_i$ is the position of antenna i and $\mathbf{k}$ is the wavevector with wavenumber k such that: $\mathbf{k} = k\sin\theta\cos\phi\hat{\mathbf{x}} + k\sin\theta\sin\phi\hat{\mathbf{y}} + k\cos\theta\hat{\mathbf{z}}$.

By combining visibilities for all feed ports, the measured visibilities can be written as $R_{ij} = R_{ij,X} + R_{ij,Y}$ and the model visibilities: $M_{ij} = M_{ij,X} + M_{ij,Y}$. Thus, equations (1) and (2) can be written in matrix form as:

$$\mathbf{R} = \mathbf{G}\mathbf{M}\mathbf{G}^H \tag{3}$$

The calibration problem is to find the gains G_i that minimize the difference between the measured visibilities and the model visibilities. And this, taking equation (3), can be written as:

$$\hat{\mathbf{G}} = \arg\min_{\mathbf{G_i}} ||\mathbf{R} - \mathbf{G}\mathbf{M}\mathbf{G}^H||_F \tag{4}$$

Where $\hat{\mathbf{G}}$ is therefore an estimator of the gain that solves Eq (3), and $||\cdot||_F$ is the Frobenius norm of the matrix.

Chapter 1

Questions

1.1 Equation (4) and multiplying all gains by the same phase factor

Equation (4) summarises the calibration problem. It states that the estimate for the gain solution should minimise the difference between the measured visibilities and the modeled visibilities, with respect to the gains $\mathbf{G_i}$. The Frobenius norm is used to measure this difference between the two matrices and is described below in equation (1.1) [6].

$$||A||_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2} \tag{1.1}$$

for any matrix A of size m×n.

For the SKA-Low station, all matrices in Eq. (4) are of size 256×256 [2]. The gains matrix is diagonal, with diagonal elements being G_i, the gain for each antenna i [5]. Measured and modelled visibilities matrices R & M are complex valued, and symmetric. This is because the cross-correlation of two antennae should be undirected: $R_{ij} = R_{ji}$, and the same goes for M. As discussed above, the model visibilities are calculated using the EEPs of the antennae, and the brightness temperature of the sky. By applying the Hermitian transpose to the gains matrix, it ensures the gains of each antenna in the concerned pair are used, as the transpose will flip the order of the diagonal elements of the matrix.

Considering the residual errors as what is being minimised in Eq.(4), and multiplying all gains by the same phase factor, $e^{j\phi}$, the following can be shown by considering a single pair of antennae, i and j [3]:

$$R_{ij} - e^{j\phi}G_iM_{ij}(e^{j\phi}G_j)^* = R_{ij} - e^{j\phi}G_iM_{ij}(e^{j\phi})^*G_j^* \tag{1.2}$$

$$R_{ij} - e^{j\phi} G_i M_{ij} e^{-j\phi} G_j^* = R_{ij} - G_i M_{ij} G_j^* \tag{1.3}$$

which ends up being the same as without multiplying by the phase factor. Therefore, the residual errors are unaffected by the phase factor, so long as it is the same for all gain values.

1.2 Power of Embedded Element Patterns (EEPs) and the Average Element Pattern (AEP)

The EEPs are a way to account for the impact of antennas being closed to each other, and therefore the mutual coupling that occurs in arrays of antennae [2] [1]. By considering antenna i to be the only active antenna and the others passive, one obtains a representation of the electric field generated by that antenna and affected by the surrounding ones [2]. Here, the `compute_EEP` function given in the starter code was used to plot the EEPs of the 256 antennae, as well the Average Element Pattern (AEP), which is obtained by taking the average of all EEPs. The `compute_EEP` function takes in station characteristics data such as the position of the antennae and coefficients for each feed (X and Y), as well as arrays of values of θ and ϕ for which the EEPs need to be computed. It then returns θ and ϕ components of the EEPs for each antenna, each feed, and of course each angular coordinates given. Thus, 4 arrays of number of (θ, ϕ) pairs$\times$256 EEP values are obtained. One important note is that `compute_EEP` will use θ and ϕ values as angular coordinates, in the order they are given in the arrays, and will not compute EEPs for each possible combination of θ and ϕ values in the arrays.

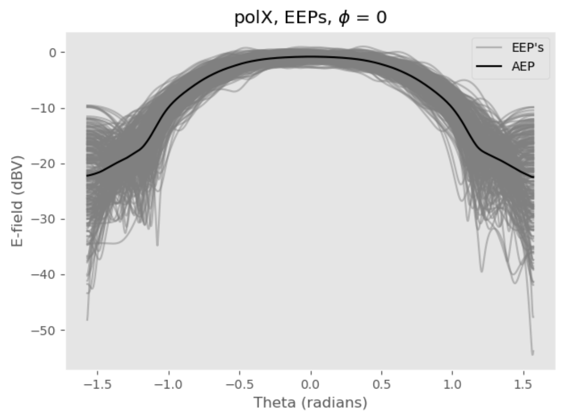

Figure 1.1: The X feed EEPs of each individual antenna (grey), and the AEP (black).

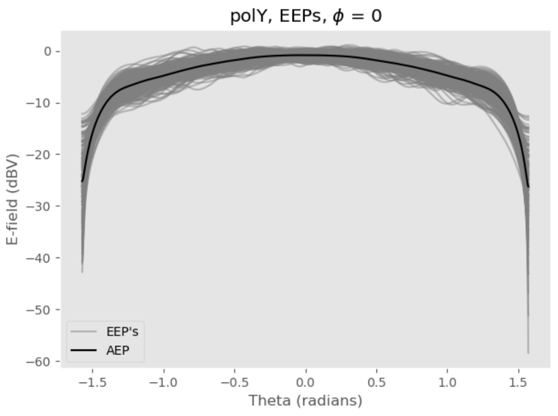

Figure 1.2: The Y feed EEPs of each individual antenna (grey), and the AEP (black).

Fig 1.1 and 1.2 show the EEPs obtained for $\phi = 0°$ and $\theta = [-\pi/2, \pi/2]$ of the 256 antennae for both feed X and Y. The AEP is also shown in black and was simply obtained by averaging the EEPs. As can be seen, there is quite a lot of variability in the EEPs, especially for the X feed, where some have side peaks for large values of $|\theta|$, and others have very low EEP values for these θ ranges. This could be explained by some antennae being at the edges of the array while others are at the center, surrounded by reflecting antennae (see Fig. 1.3).

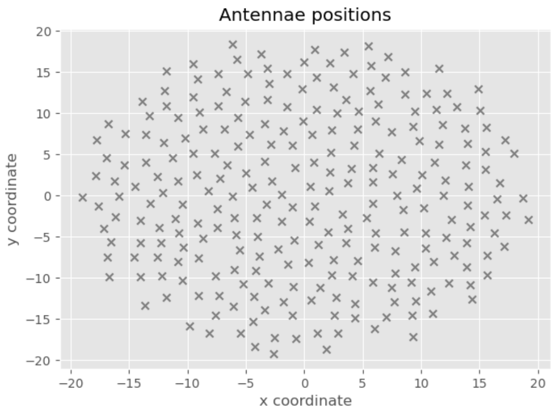

Figure 1.3: The layout of the SKA-Low station.

1.3 The StEFCal Algorithm

In this section the Stastistically Efficient and Fast Calibration (StEFCal) algorithm is implemented and discussed [4]. The StEFCal algorithm is an attempt to solve the calibration problem in Eq. (4) by using an Alternating Direction Implicit (ADI) method, which is often used to solve large matrix equations or partial differential equations [7]. It is a two-step algorithm, that first solves for G^H, with G fixed, and then solves for G, with G^H fixed [4]. With some algebra, Salvini & Wijnholds show that each iteration is the same as updating each gain value by one that solves a linear least squares problem of the form:

$$||\mathbf{\Delta}||_F = ||\hat{\mathbf{R}} - \mathbf{Z}\mathbf{G}^H||_F = \sqrt{\sum_{p=1}^{P} ||\hat{\mathbf{R}}_{:,\mathbf{p}} - \mathbf{Z}_{:,\mathbf{p}} g_p^*||_2^2} \tag{1.4}$$

where $\mathbf{Z}$ is given by $\mathbf{Z} = \mathbf{G}^{[\mathbf{i}]}\mathbf{M}$, with $\mathbf{G}^{[\mathbf{i}]}$ being the updated gains matrix. Thus, the objective is to find g_p^* which is given by:

$$g_p^* = \frac{\hat{\mathbf{R}}_{:,\mathbf{p}} \cdot \mathbf{Z}_{:,\mathbf{p}}{}^{[i-1]}}{(\mathbf{Z}_{:,\mathbf{p}}{}^{[i-1]})^H \cdot \mathbf{Z}_{:,\mathbf{p}}{}^{[i-1]}} \tag{1.5}$$

And this can then be iteratively determined whithin a antennae, p, loop giving the algorithm:

StEFCal Algorithm

```
1:  $\mathbf{G}^{[0]} \leftarrow \mathbf{I}$                                      ▷ Initialisation
2:  for $i = 1, 2, \ldots, i_{max}$ do                              ▷ Iterate
3:      for $p = 1, 2, \ldots, P$ do                          ▷ Loop over antennae
4:          $\mathbf{z} \leftarrow \mathbf{G}^{[i-1]} \cdot \mathbf{M}_{:,p} \equiv \mathbf{g}^{[i-1]} \odot \mathbf{M}_{:,p}$
5:          $g_p \leftarrow (\hat{\mathbf{R}}_{:,p}^H \cdot \mathbf{z})/(\mathbf{z}^H \cdot \mathbf{z})$
6:      end for
7:      if $i \equiv 0[2]$ then                              ▷ Every even iteration
8:          if $||\mathbf{g}^{[i]} - \mathbf{g}^{[i-1]}||_F/||\mathbf{g}^{[i]}||_F < \tau$ then       ▷ Convergence check
9:              Convergence reached
10:         else
11:             $\mathbf{G}^{[i]} \leftarrow (\mathbf{G}^{[i]} + \mathbf{G}^{[i-1]})/2$       ▷ Average gains of last 2 iterations
12:         end if
13:     end if
14: end for
```

Line 11 of the algorithm is a solution to an issue where the basic iteration would converge very slowly and could even get stuck between 2 vectors $\mathbf{g}$. The solution proposed was to take the average of the last 2 iterations, at every even iteration [4]. The reason why it is every even iteration is to ensure that this does let the vector get updated. This is akin to making jumps in gradient descent algorithms, to avoid getting stuck in local minima. This enables the algorithm to converge much faster.

The StEFCal algorithm was implemented, setting the τ threshold at 10^{-5}, based on the paper's findings on convergence requirements [4, p.7]. The algorithm was tested on the AEP and EEPs model visibilities, and the convergence plots are shown in Fig. 1.4 and 1.5.

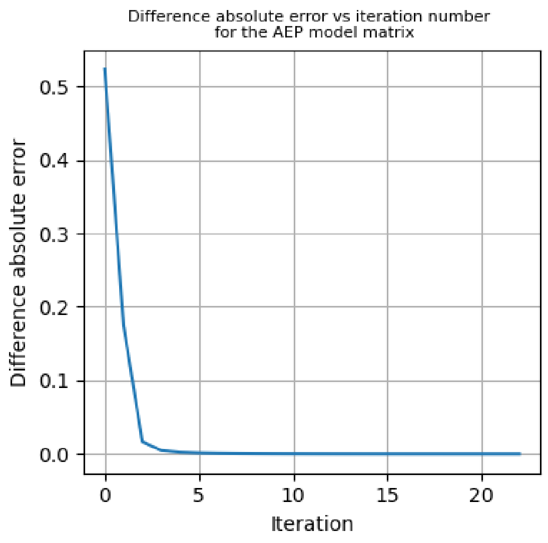

Figure 1.4: Convergence plot of the StEFCal algorithm, showing the difference between the current gains and the previous iteration's as a function of iteration number, for the AEP model visibilities.

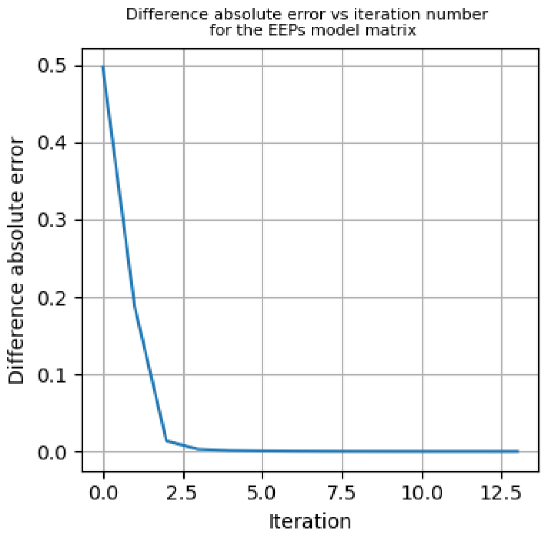

Figure 1.5: Convergence plot of the StEFCal algorithm, showing the difference between the current gains and the previous iteration's as a function of iteration number, for the EEPs model visibilities.

Both show very fast convergence, though the EEPs model visibilities converge almost twice as fast as the AEP model visibilities. Bear in mind that those differences between iteration were obtained every even iteration, so the total iteration number is twice the ones shown in the plots. The total number of iterations until convergence were 27 for the EEPs model and 45 for the AEP model. This is likely due to the AEP being quite a strong assumption, with all EEPs being the same average value. Taking the EEPs case, we can also look at the logarithm of the difference between 2 iterations of **g**. This is shown in Fig. 1.6 which can be compared to Fig. 14 of the paper [4]. The plot shows that the algorithm converges very quickly, with the error decreasing logarithmically with the number of iterations.

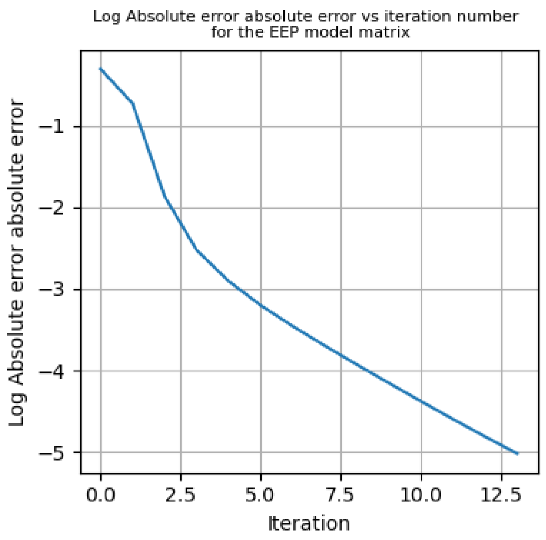

Figure 1.6: Logarithm of the absolute error between 2 iterations of $\mathbf{g}$, for the EEPs model visibilities.

1.4 A discussion of the absolute errors between the estimated gains and the true gains

In this section, the absolute errors between the estimated gains and the true gains are plotted as a function of iteration number. The absolute error is calculated as:

$$\sigma_{abs} = \sum_{i=1}^{P=256} |\mathbf{g}_i - \mathbf{g}_{\text{true}i}| \tag{1.6}$$

where $\mathbf{g}_{\text{true}}$ is the true gains matrix, and the subscript i denotes the i-th element of the vector. Since the gains are complex values, it is worth looking at 2 other absolute errors: the absolute error in the amplitude of the gains, and the absolute error in their phase. For a complex number $z = a + ib$, the amplitude is given by $|z| = \sqrt{a^2 + b^2}$ and the phase by $z_{\text{phase}} = \arctan(b/a)$. The absolute errors in amplitude and phase are then given by:

$$\sigma_{\text{amp}} = \sum_{i=1}^{P=256} ||\mathbf{g}_i| - |\mathbf{g}_{\text{true}i}|| \tag{1.7}$$

$$\sigma_{\text{phase}} = \sum_{i=1}^{P=256} |\mathbf{g}^i_{\text{phase}} - \mathbf{g}_{\text{true}}{}^i_{\text{phase}}| \tag{1.8}$$

with the i switched to superscript for clarity. Plotting these for both model visibilities, the results are shown in Fig. 1.7 for the AEP model, and in Fig. 1.8 for the EEPs model.

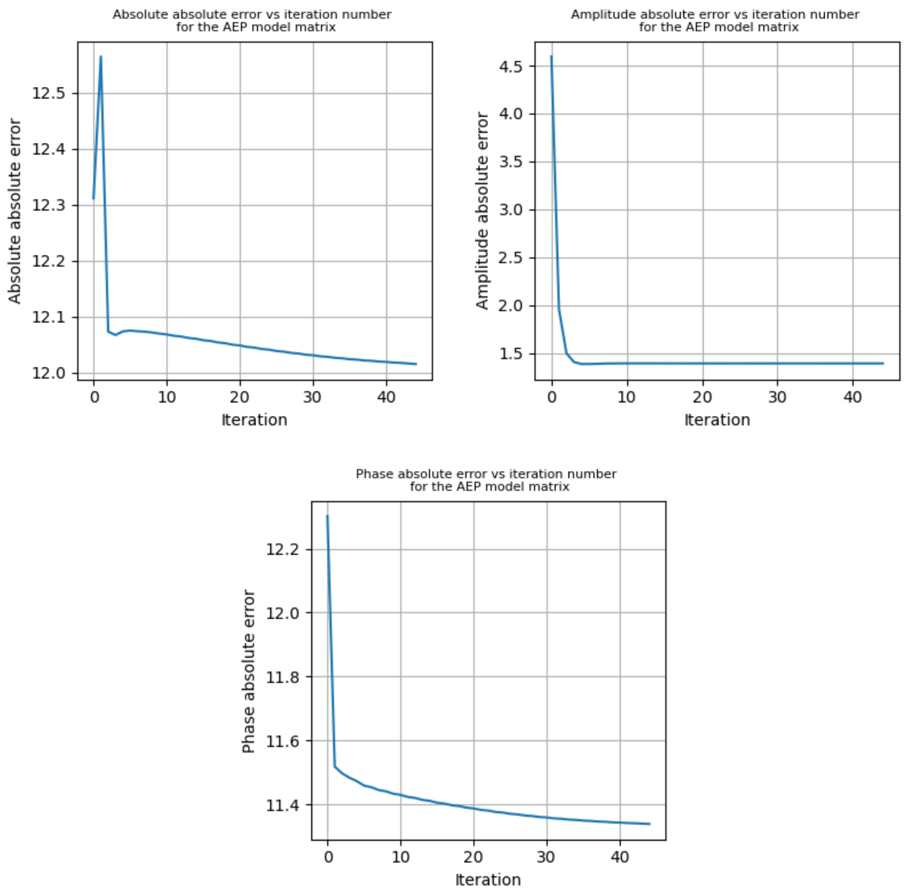

Figure 1.7: Absolute errors between the estimated gains and the true gains, the absolute errors in amplitude, and the absolute errors in phase, as a function of iteration number, for the AEP model visibilities.

Both show an interesting result in the absolute error as this one, for the first iteration, actually gets worse before decreasing very fast. And though both models converged, the absolute errors of the gain solution, their amplitude, and their phase still have non-negligible values at convergence. However, one can argue that for the sum of 256 errors, these are still reasonable, and this will be further discussed in the next sections.

The minimum absolute error achieved using both models is 12.02 for the AEP model and 11.90 for the EEP model. Which again is consistent with our assumptions based on the relaxation the AEP model provides. Both these errors were obtained for

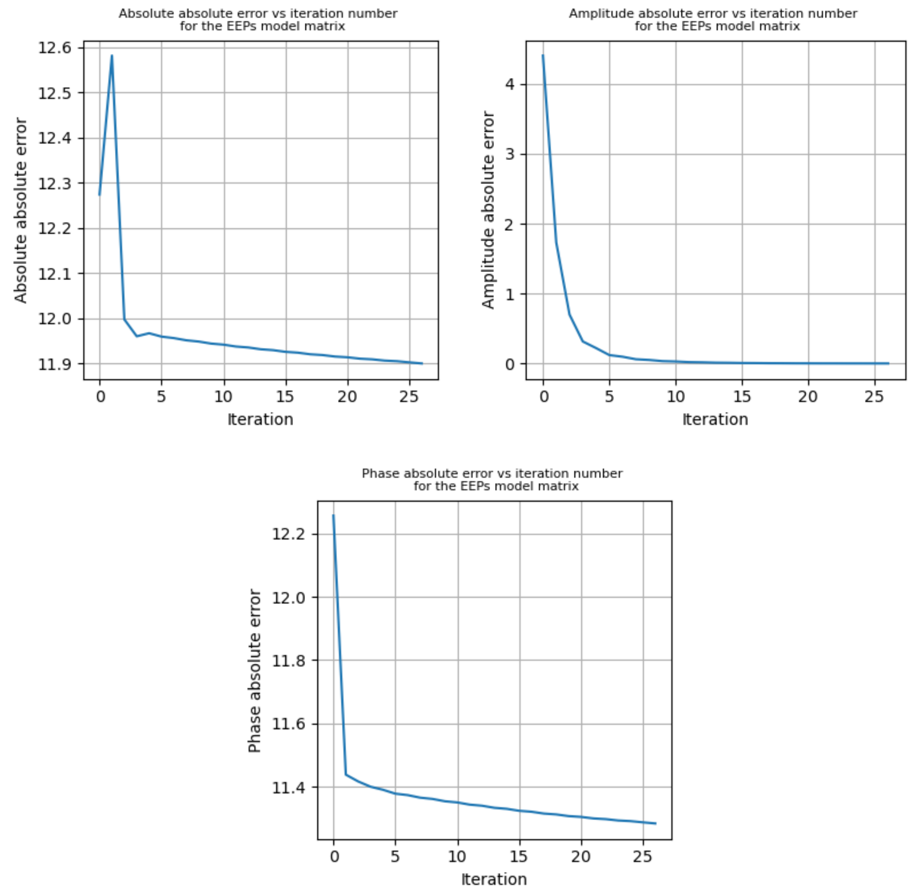

Figure 1.8: Absolute errors between the estimated gains and the true gains, the absolute errors in amplitude, and the absolute errors in phase, as a function of iteration number, for the EEPs model visibilities.

the last iteration, 44 (AEP) and 26 (EEPs) respectively, which were the penultimate iterations before the algorithm stops due to convergence.

1.5 Comparing station beams obtained

In this section, using the best gain solutions obtained from the StEFCal algorithm for both models, the station beams for the X and Y feeds were computed. Similar to section 2, only a slice of the station beam was obtained for $\phi = 0$ and $\theta = [-\pi/2, \pi/2]$. At this point, gain solution estimates have been obtained and the EEP of each antenna has been computed. To compute the total array beam voltage, for a direction (θ, ϕ), the following equation is used:

$$\vec{\mathbf{P}}(\theta,\phi) = \sum_{p=1}^{P=256} \hat{w_p} G_p \vec{\mathbf{F}}_p(\theta,\phi) e^{-ik(\sin(\theta)\cos(\phi)x_p + \sin(\theta)\sin(\phi)y_p} \tag{1.9}$$

where $\hat{w_p}$ is the weight attributed to the p-th antenna, and is a function of the antenna's position and which direction the beam is pointing:
$\hat{w_p} = e^{ik(\sin(\theta_0)\cos(\phi_0)x_p + \sin(\theta_0)\sin(\phi_0)y_p}$. G_p is the gain of the p-th antenna, $\vec{\mathbf{F}}_p(\theta,\phi)$ is the EEP of the p-th antenna, for direction (θ,ϕ), and x_p and y_p are the coordinates of antenna p. The total array beam voltage is then the sum of the beam voltages of each antenna for a specified direction. Since EEP is given for feed X and Y, each model gives 2 beams to plot.

Figure 1.9 shows the overlapped station beams for the true and modelled gain solutions. The beams are quite similar with the EEPs beam being close to perfectly overlapping the true gain beam. The AEP beam is however slightly off, especially for large positive values of θ. This can again be explained with how big of an assumption the AEP model is as one moves towards the edge of the array and large values of θ are reached. These locations are where the EEPs are the most different from the AEP (see Fig. 1.1-1.2), and therefore the beam is the most different from the true beam.

Coming back to minimum absolute error achieved for the gain values in section 4, it can be seen here that though the errors were not negligible, the beams obtained are still very close to the true beam, for the EEP model. This is both an indication that the StEFCal algorithm works well, and that the EEP model is usually a better option to use for calibration.

1.6 Visualising the 2D total array beam

In this section, the total array beam is visualised in 2D, for the most accurate of the gain solutions obtained, i.e. the EEPs gain solution. For this plot, the two feeds were combined. To plot this, the EEPs had to be calculated for repeating values vectors of θ and ϕ, then reshaped into a 2D array. Since ϕ goes from 0 to 2π, in order for the directions to cover the entire sky, θ only needs to go from 0 to $\pi/2$, allowing for better resolution for the same number of points for which to calculate the EEPs. Also, the beam was set up to be steered at angles $\theta = 40°$ and $\phi = 80°$, to see how that affected the beam. The results are shown in Fig. 1.10.

As expected, the main lobe of the beam is steered away from the zenith. And though this plot is in sine-cosine coordinates, it is clear to see that ϕ is arount 80°, with 90° being at the top of the plot. Comparing Fig 1.10 to the true gain beam (see Fig. 1.11, left), the 2D beam is again very similar. Plotting the difference of the two beams, the result is shown in Fig. 1.11, right. The difference is very small, with

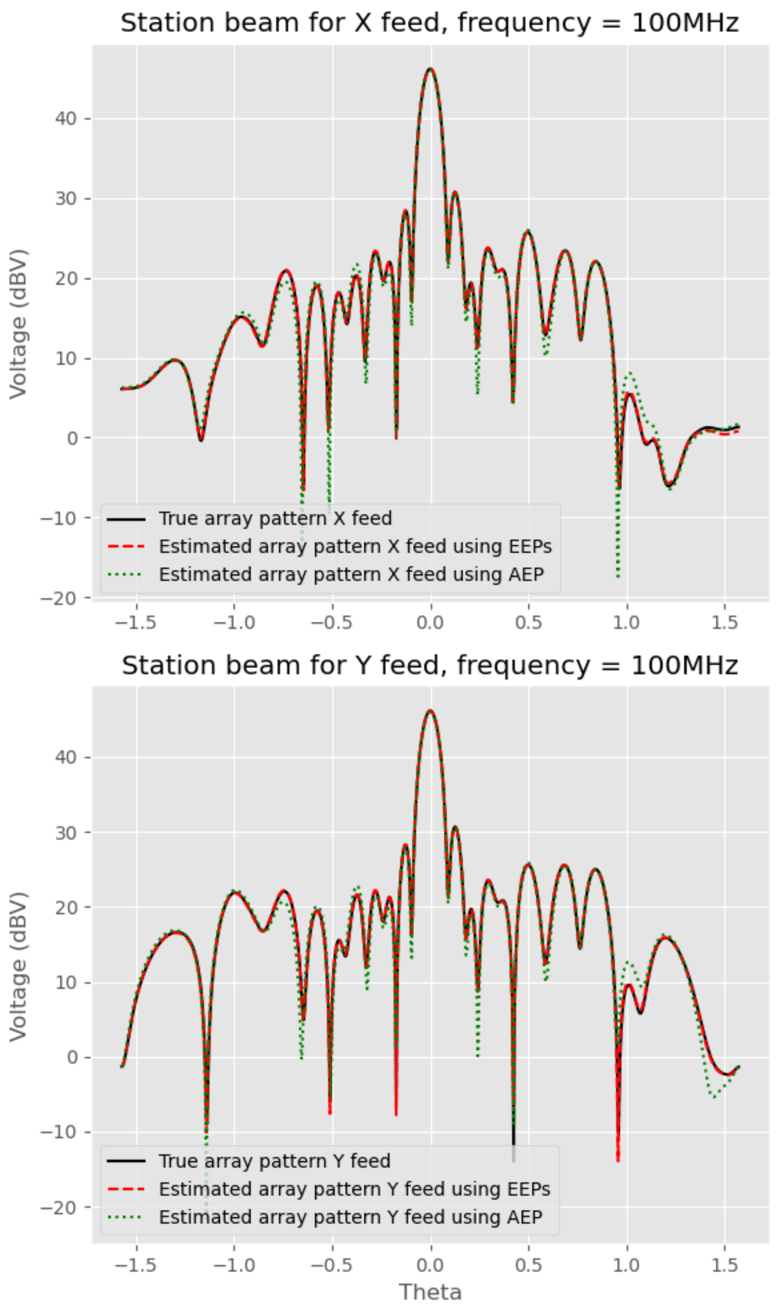

Figure 1.9: The station beams for the true (black) and 2 model gain solution (AEP=green, EEPS=red).

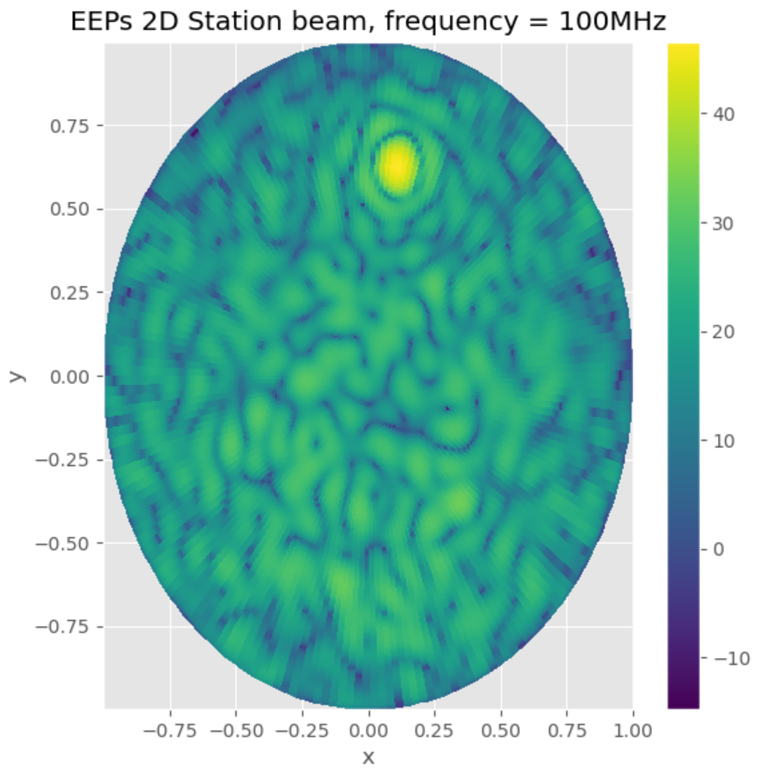

Figure 1.10: The 2D total array beam for the EEPs gain solution.

the overall difference staying below 0 dBV, or simply small values, except for a few singular directions.

Now, these results were obtained for a frequency of 100 MHz. It is worth seeing what different frequencies mean in terms of the beam generated. This was done for the 2 extremes of the SKA-Low station frequency range, 50 MHz and 350 MHz. The results are shown in Fig. 1.12. As can be seen, the higher the frequency, the narrower the lobes of the beam. This is due to the fact that the higher the frequency, the higher the wavenumber, and therefore the phase factor in the beam calculations is much more sensitive to the value of the angles.

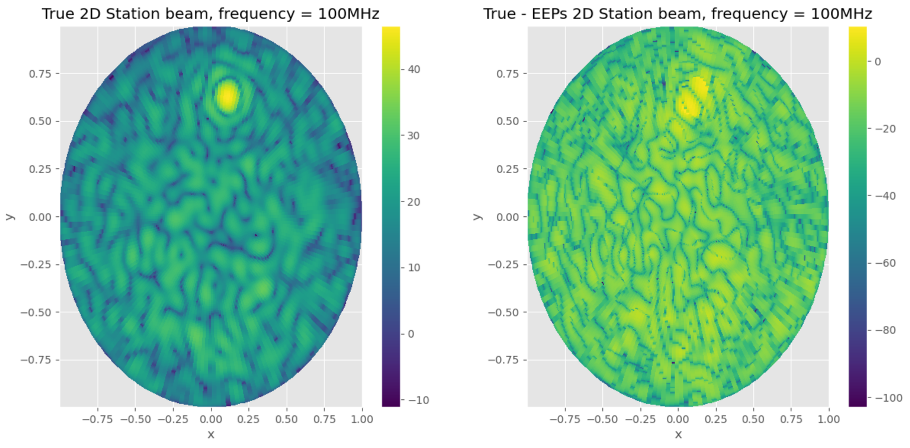

Figure 1.11: The 2D total array beam for the true gain solution (left), and the difference between the true and EEPs gain solutions beams(right).

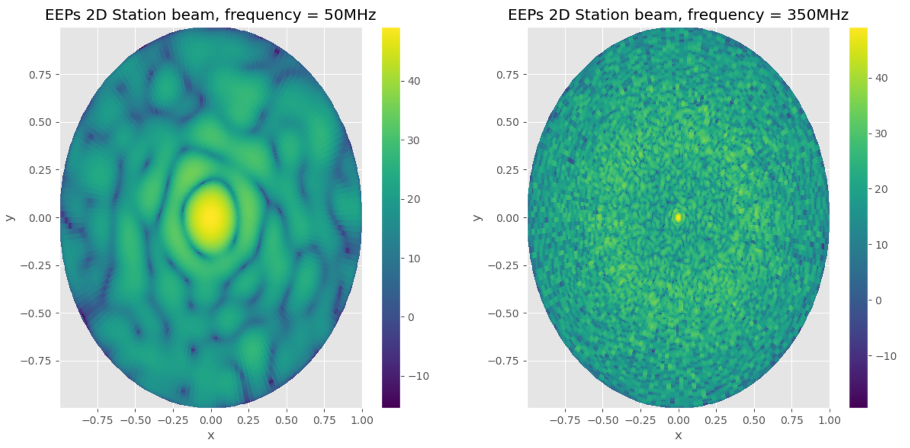

Figure 1.12: The 2D total array beam for the EEPs gain solution at 50 MHz (left) and 350 MHz (right).

Chapter 2

Appendix

2.1 README

2.1.1 SKA Coursework - Gain Calibration of a SKA-low Station

Description

This reporsitory contains the code and written report for the SKA Coursework. The aim was to gain calibrate one SKA-low station using an algorithm for the retrieval of gain solutions.

Contents

Inside this `tmb76/` directory, there are a few sub-directories one can explore. There's the code directory (`harp_beam/`), which contains all the code used in the solver. An important note is that the code will give outputs in the command line but also store the plots in a `Plots/` directory which will be created as the first code file is run. So if there is no `Plots/` directory in the repository yet, running one of the `question_*.py` once should lead to creating one. Then, there is a `test/` directory which contains some unit test for the code. The last one is the `Report/` directory, which contains the LaTeX file for the report, as well as the pdf version of it, along with the references `.bib` file. More importantly, there are an `environment.yml` and `Dockerfile` files, which one is advised to use.

How to run the code

For permissions reasons, the `Dockerfile` is not set up to pull the repository directly as it builds the image. Therefore, one must first download this repository to their local machine and then are free to build the Docker image from the `Dockerfile`.

To run the solver on a Docker container, one first has to build the image and run the container. This can be done as follows:

```
$ docker build -t ska_coursework .
$ docker run --rm -ti ska_coursework
```

Listing 2.1: Building the Docker image and running the container.

The `ska_coursework` is not a strict instruction, it can be set to any other name the user may prefer.

As you run this, the Docker image will get built and the container ran, providing the user with a bash terminal-like interface where the solver code can be run as follows:

```
  $ python harp_beam/question_*.py
```

Listing 2.2: Running the code.

where `*` can be from `2` to `6`. Each contain the code to get the results for questions 2 to 6 of the mini-project.

If there is a need to get the plots back on the local machine, the second line above can be ran without the `--rm` and also set the container name using `--name=container_name` (any valid name is fine). From there, run all the code as instructed below. Once all desired outputs and plots have been obtained. One can exit the container and then run:

```
$ docker cp docker cp container_name:/SKA_Coursework/Plots ./Plots
```

Listing 2.3: Copying the plots back to the local machine.

The `Plots/` directory will get copied into the local folder the container was ran from.

Note on time: Running the `question_[2-5].py` files all take only a few seconds. Running the `question_6.py` file however can take up to a minute. This is based on running all of these on a MacBook Air M2 (2022, Ventura 13.2.1), with 8 GB of Memory, so this may be slower on a container.

Further development

If one wishes to further develop this code, such as trying other gain self-calibration algorithms, when the image is built, git is installed and initialized and the pre-commit hooks are installed.

Use of Generative AI

GitHub Copilot's autocompletion feature was used in coding the project, when writing docstrings for the functions, though sometimes adding elements ourselves, and for repetitive parts of the code, such as question 6 with the multiple `compute_array_pattern()` function calls. It was used to generate code to suppress the user warnings in question 6. ChatGPT was also used to help in debugging the code, by providing the traceback as a prompt when an error was difficult to understand, asking to explain what the error refers to. One example is when dealing with the `np.diag()` function, which was proposed by ChatGPT, when asked how one could extract the diagonal elements of a square array. The vectors defined with it were immutable through some authorization problem. ChatGPT did identify that the np.diag() was the cause. However, the fix proposed which was to write `g_new.setflags(write=1)` was not used. And the choice was made to simply try and limit switching between matrix form and vector form in the StEFCal algorithm.

Bibliography

[1] Constantine A. Balanis. Antenna theory: Analysis and design. *Radio Science*, 45(6):RS6S02, November 2010.

[2] J. Borg, A. Magro, K. Zarb Adami, E. de lera Acedo, A. Sutinjo, and D. Ung. On-sky calibration of a ska1-low station in the presence of mutual coupling. *Monthly Notices of the Royal Astronomical Society*, 496(1):933–942, July 2020.

[3] ProofWiki. Product of complex conjugates, 2021. ProofWiki.

[4] Stefano Salvini and Stefan J. Wijnholds. Fast gain calibration in radio astronomy using alternating direction implicit methods: Analysis and applications. *Astronomy & Astrophysics*, 571, 2014.

[5] O. M. Smirnov. Revisiting the radio interferometer measurement equation: A full-sky jones formalism. *Astronomy & Astrophysics*, 527(March):A106, 2011. Published online 04 February 2011.

[6] Eric W. Weisstein. Frobenius norm, 2021. MathWorld–A Wolfram Web Resource.

[7] Wikipedia. Alternating-direction implicit method, 2021. Wikipedia.

[8] Wikipedia. Antenna feed, 2021. Wikipedia.