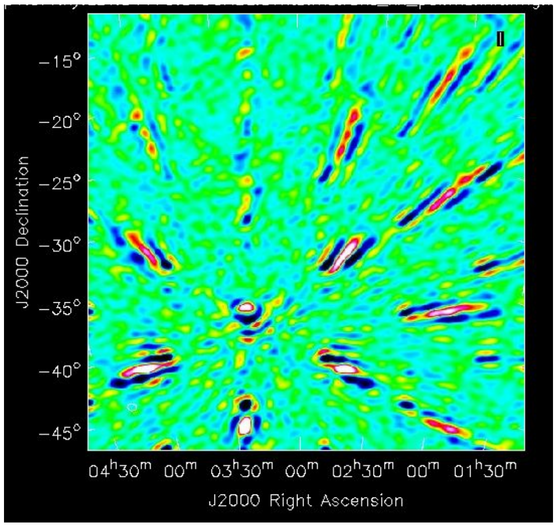
# Constant Offset in Cross-Polarized HERA IDR2.1 Data

Katherine Elder, CHAMP ASU, 08/16/18

This memo gives an overview of the project I worked on this summer with the CAMPARE/CHAMP summer internship program at ASU. There are links in this document to Jupyter notebooks and python scripts which contain the code and detailed explanations of the process. All of the notebooks can be found on GitHub in the asu_hera repo, in the folder titled crosspol_investigation.

**1) Imaging Cross-Polarized Data**

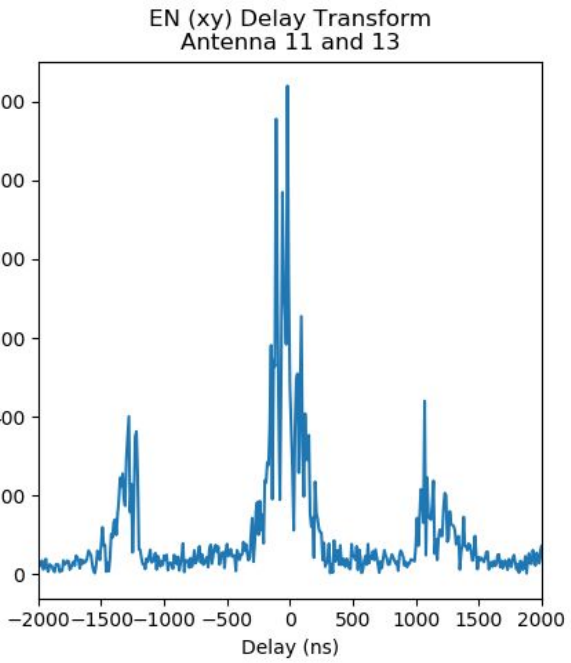


I started off this summer by trying (and failing) to clean and calibrate the HERA IDR2.1 LST binned cross-polarized data. I was attempting to do this so that we could image the data and gain a better understanding of what the polarized sky looks like. Unfortunately, the CASA software is not built in a way that allows you to directly work with cross-polarized data. Ultimately, the only way that I could generate images of the data was to trick CASA into thinking the data was parallel polarized and then running a clean command with the number of iterations set to zero so that it did not actually run a clean. The python script that does this is found here. An example of these images is shown above, with Fornax A crossing the NE (or yx) polarized field of view.

Once we had images of the uncleaned, LST binned data, we created videos of the entire run. When we watched the videos, we noticed that there were objects that didn't change with time. We already suspected that most of the data we would be seeing in these images would be noise caused by instrumentation feedback and signal bouncing. Our next step was to look at the data and identify if these time-constant objects appear in the data, or if they were simply artifacts of the movie-making process.

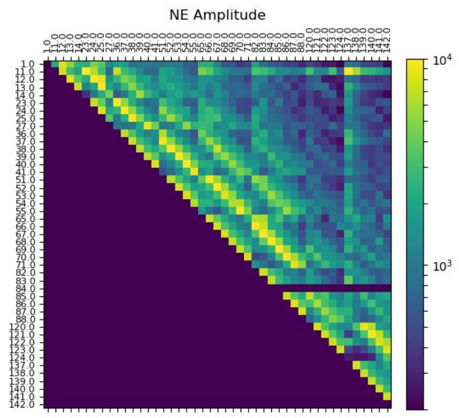
**2) Waterfalls and Spectra: (Jupyter Notebook)**

To take a closer look at the data, we created waterfall plots. These plots showed lines running through the plot which did not change with time, which told us that these objects in the movies were not artifacts in the images.



Next, we took the time average and Fourier transformed the data to plot the delay spectrum. Because most of the structure from the sky is averaged out, we are left with the additive constant offset. By looking at the delay corresponding to the peaks, we can find out how far away these signals are originating. The pattern we began noticing was that practically every antenna pair produced a prominent peak which corresponded to the length of the baseline. But antenna pairs with longer baselines would also have prominent peaks away from baseline lengths. Sometimes these peaks corresponded to the cable length, or twice the cable length, but other times they didn't seem to have any specific physical reference.

**3) Delay Spectra amplitudes and delays** (Jupyter Notebook)



We wanted to investigate of these patterns more, so we came up with the idea to plot a matrix of every antenna pair with the maximum peak amplitude from their delay spectrum as the color scale. By doing this, we could take a look at both patterns we noticed from the spectra and try to identify other patterns.

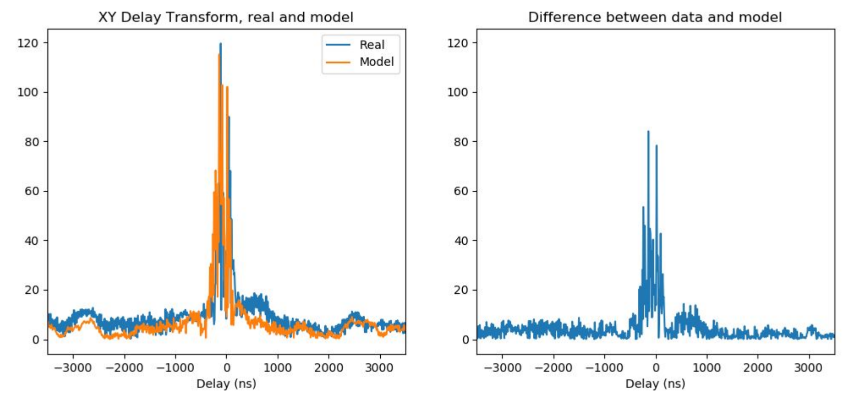
Following up on that idea, we also plotted a map of the antenna array, using the amplitudes and corresponding delays as the color scale. To better look at both patterns, baseline dependence and non-baseline dependent, we split up the window of where we looked for peaks in the delay spectrum. Lastly, we also plotted the delay distance as a function of baseline, which allowed us to easily see how far these signals were traveling before being picked up by the antenna and how it related to the baseline.

These plots confirmed that the baseline plays a large role in these constant offsets within the data. They also alerted us to the fact that the north dipole of antenna 84 is being flagged somewhere along the pipeline.

## 4) Time-Averaged Visibility Model (Jupyter Notebook)

Now that we had a better idea of what factors were involved in creating the additive constant offsets, we wanted to build a model of the time-averaged visibilities. We hypothesised that it could be built with only a few variables: the autocorrelations, a baseline dependent delay, a cable dependent delay, a baseline dependent amplitude factor, and a non-baseline dependent amplitude factor. The equation for our model looks like this:

$$\overline{V}_{ij} = \varepsilon_{ij}(\overline{V}_{ii} e^{-i2\pi\tau_{ci}\nu} + \overline{V}_{jj} e^{-i2\pi\tau_{cj}\nu}) + P_{ij} e^{-i2\pi\tau_{aij}\nu}(\overline{V}_{ii} + \overline{V}_{jj})$$



The full description of the factors in the model and how we built it can be found in the Jupyter Notebook. On the left is an example of the model and how it compares to the real time-averaged data. As you can see, the model does a decent job of modelling the real time-averaged visibility, but there is still a lot of noise and small peaks it is missing.

## 5) Next Steps

There is obviously still room for improvement in how the model is designed. The next steps for it will be to run the model through a fitter. After that, we can begin working to improve the way the baseline and non-baseline peaks are found to hopefully increase accuracy.

More generally, there is still a lot we don't understand about what is causing these additive constant offsets. There is obviously a baseline component, although our data shows that it is actually baseline plus 3 meters. But outside of the baseline, we are still not completely sure what is causing the peaks. Further investigation is needed to form a better understanding of what exactly is affecting the data.