Calibrating Data with the ATA RFSoC Correlator

Pranav Premnath
Wael Farah

SETI Institute
Allen Telescope Array

February 24, 2022

1 Introduction

The new online correlator with the Allen Telescope Array (ATA) uses the RFSoc boards, and has 672 MHz of effective bandwidth. This document discusses using CASA to calibrate the data, and obtaining the delay and phase solutions that can be used to update the delay engine.

1.1 Concatenate data to form a Measurement Set

We must first concatenate the data from all nodes, Each node contains 96MHz, so we concatenate nodes 1-4 for LOb and nodes 5-8 for LOc. The concatenation scripts produce a combined UVH5 file, as well as a CASA measurement set. They are located in:

```
/home/sonata/corr_data/scripts/

python concat_all_b.py uvh5_file
python concat_all_c.py uvh5_file
```

2 Calibrate with CASA

Finally, once the Measurement Set (MS) has been produced, we can start calibrating with CASA, a python based data reduction software. We define the base as the UVH5 file name.

```
base="uvh5_file"
```

2.1 Flag RFI Channels

We can inspect the data using the *plotms* function. The parameters we need to input besides the MS are the axes, the antennas to be used and the correlation, what we want to iterate over. The following call simply inspects the data to see if we need to correct for delays/phase/bandpass. The plot can be seen in Figure 1.

```
plotms(vis=base+".ms", xaxis='frequency', yaxis='phase',
antenna='!*&&&', correlation='xx,yy', avgtime='300',
iteraxis='baseline', gridrows=4, gridcols=4, coloraxis='corr')
```

More often than not, we will see that Radio Frequency Interference (RFI) is present in the data. We can flag the frequency channels that contain RFI by plotting it against the amplitude, till we see a nice bandpass structure.

```
plotms(vis=base+'.ms', xaxis='channel', yaxis='amplitude',
antenna ='!*&&&', correlation='xx,yy', avgtime='300', coloraxis='corr')
```

The following command flags the data in the channels specified in Spectral Window (spw) 0.

```
flagdata(vis=base+'.ms', spw="0:225~250;260~285;850~900;1170~1180")
```

And we can unflag data simply by specifying the channels again, and adding the unflag mode.

```
flagdata(vis=base+'.ms', spw="0:225~250;260~285;850~900;1170~1180", mode='unflag')
```

2.2 Calibrate for Delays

Once RFI is cleaned up, we can correct for the delays. This is done by performing the task *gaincal*, with the delay correction, K. The following command will do the above.

```
gaincal(base+".ms/", caltable="cal.K", refant="3", gaintype="K")
```

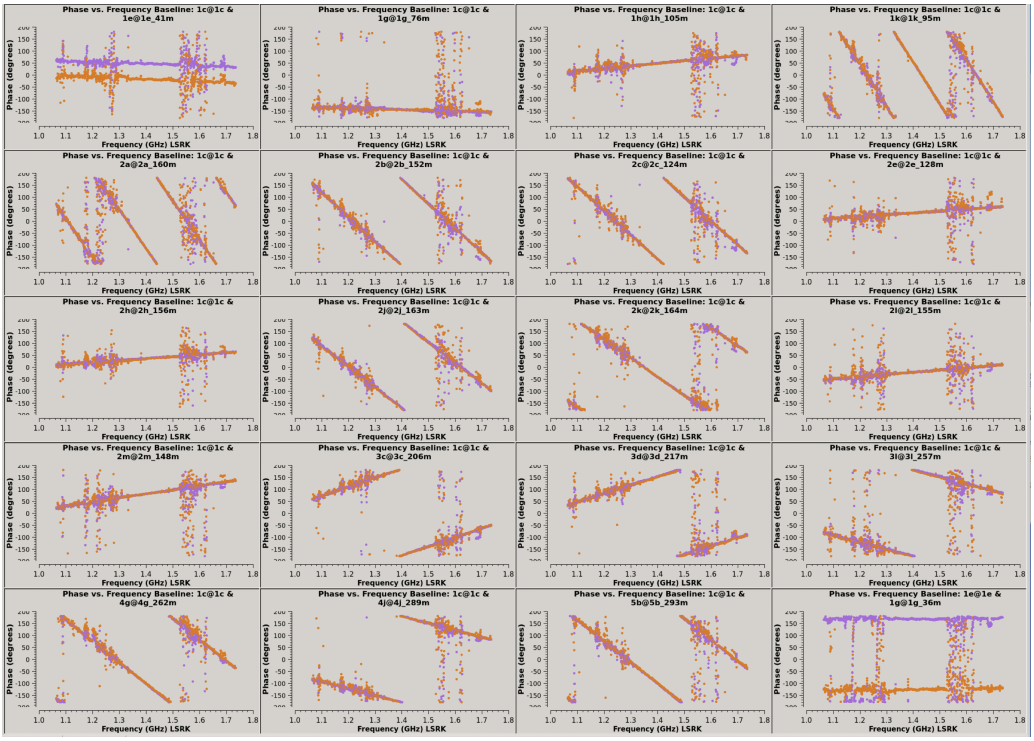

Figure 1: The data before any calibration. As the boards are out of sync, we can see bad delays.

We need to input the MS, the caltable to write out the calibration solutions to, the reference antenna, and the gaintype, which in this case is K, corresponding to delays.

Then, we apply the calibration table to the MS.

```
applycal(base+".ms/", gaintable='cal.K', calwt=False)
```

Finally, we split the corrected column from the measurement set, to obtain the delay calibrated MS.

```
split(base+".ms", outputvis=base+"_k.ms", datacolumn='corrected')
```

Looking at the split MS (Figure 2), we will notice that all antennas look flattened, but not centered at 0. Hence, we move on to phase calibration.

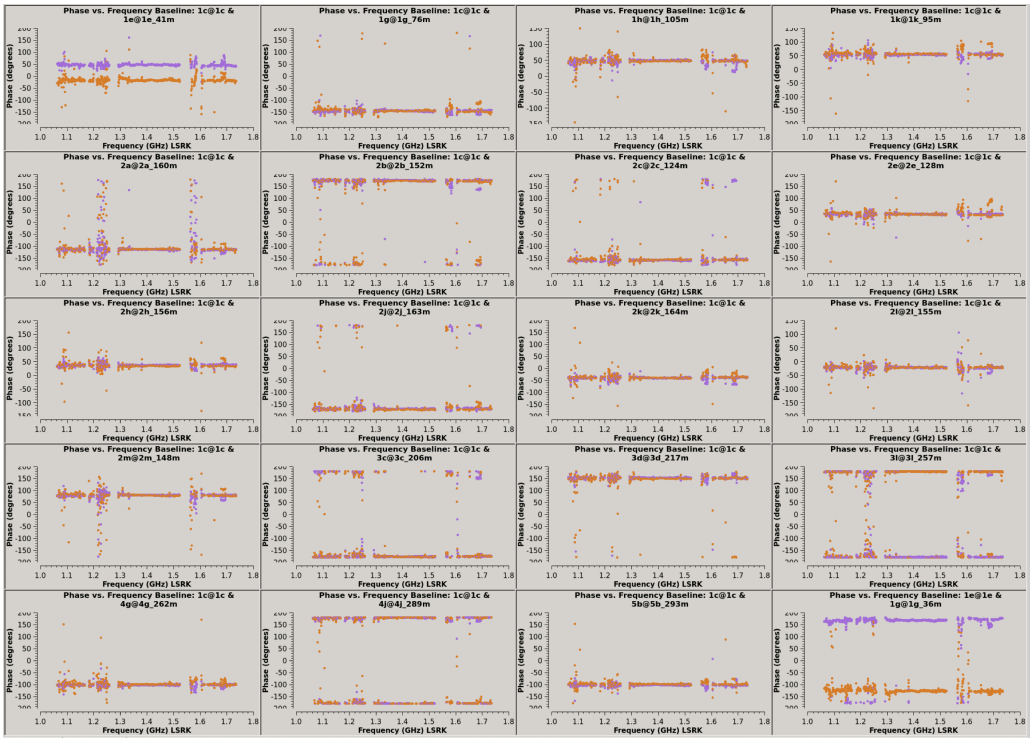

Figure 2: After delay calibration

2.3 Calibrate for Phases

Using the split MS, we then proceed to correcting for phases. This time, the gaintype is G, corresponding to phases.

```
gaincal(base+"_k.ms/", caltable="cal.G", refant="3", gaintype="G")
```

Then, we apply the calibration table to the MS.

```
applycal(base+"_k.ms/", gaintable='cal.G', calwt=False)
```

Finally, we split the corrected column from the measurement set, to obtain the phase and delay calibrated MS.

```
split(base+"_k.ms", outputvis=base+"_kg.ms", datacolumn='corrected')
```

Plotting the split MS (Figure 3), we notice that all antennas are centered at 0. But, there is still some ripples, which we need to correct for using the bandpass correction.

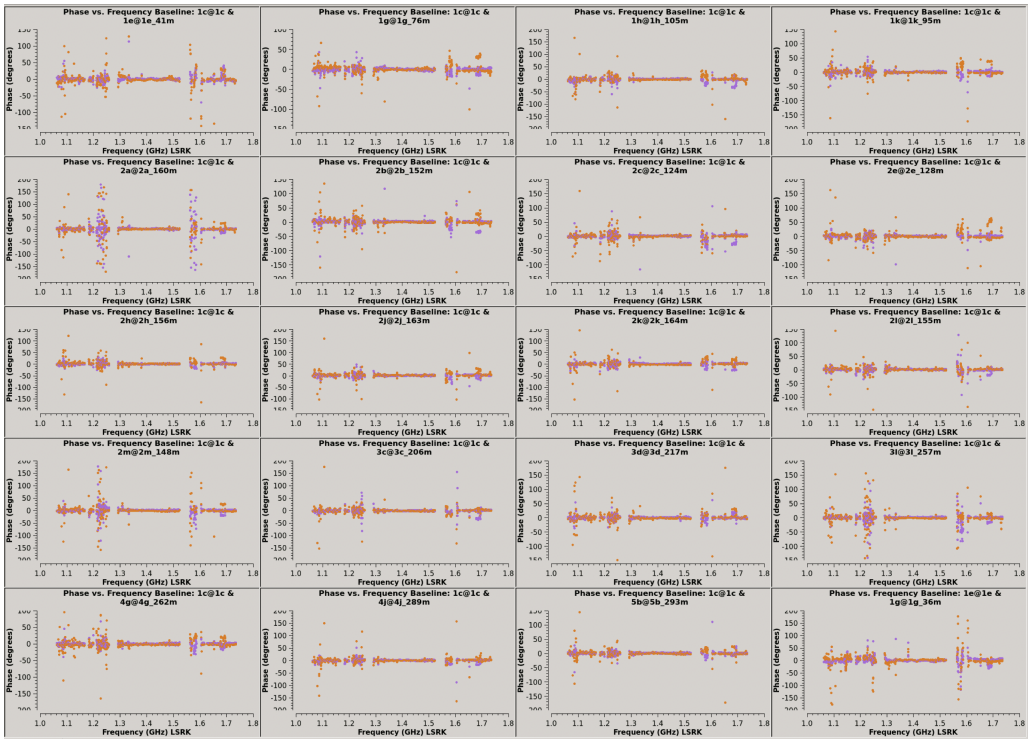

Figure 3: After phase calibration

2.4 Apply Bandpass Correction

Using the split MS, we then proceed to correcting for the bandpass. We use the task *bandpass* for this. Similar to gaincal, it produces a calibration table. The bandtype we use is *B*, which does a channel by channel solution for each specified spectral window.

```
bandpass(base+"_kg.ms", caltable='cal.BP', refant='3', bandtype='B')
```

Then, we apply the calibration table to the MS, similar to delay and phase calibration.

```
applycal(base+"_kg.ms/", gaintable='cal.BP', calwt=False)
```

Finally, we split the corrected column from the measurement set, to obtain the phase, delay and bandpass calibrated MS, just to inspect the final calibrated data.

```
split(base+"_kg.ms", outputvis=base+"_kgbp.ms", datacolumn='corrected')
```

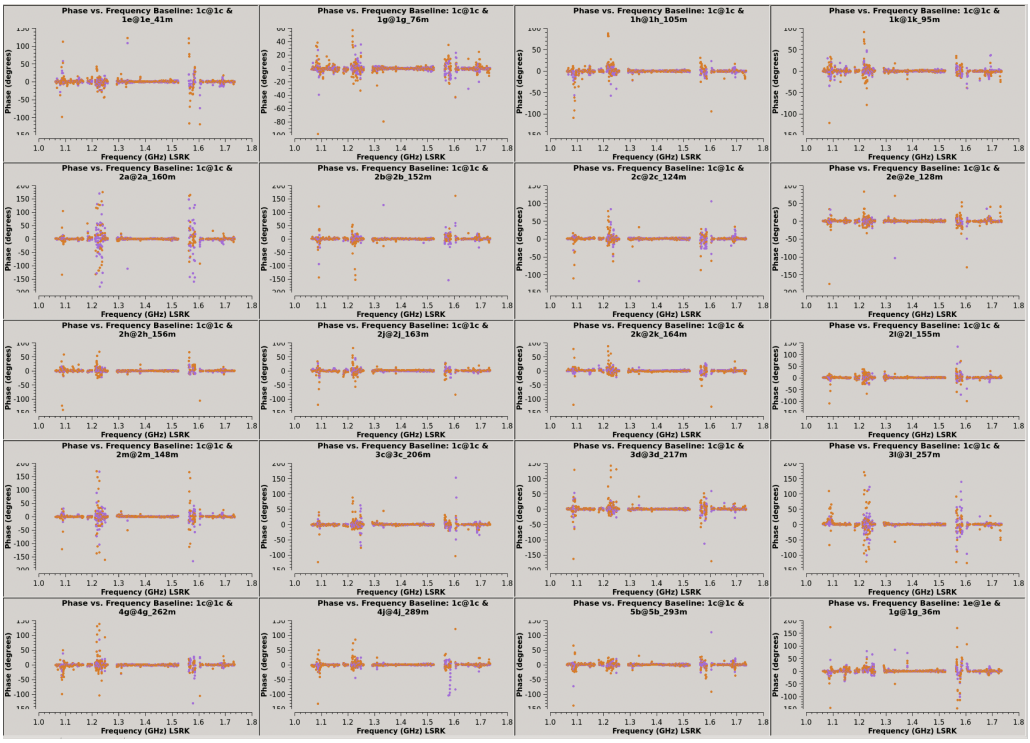

Figure 4: After bandpass calibration

We can plot the split MS (Figure 4), just to confirm that the calibration worked. Ideally, we notice flat phases for all the antennas, centered at 0, with minimal ripples. If we see this, our calibration using CASA worked.

3 Get Solutions from Calibration Tables

Once we get all three tables, we can obtain the new delay and phase solutions that we want to input into the delay engine for the next observation. The following code takes the three calibration tables, and applies the residuals to the current delay and phase solutions. We must also specify the center frequency.

```
~/corr_data/scripts/calibrate.py --delay_table cal.b.K --bf_delays
~/src/delay_engine/delays_b.txt --phase_table ./cal.b.G/
--bandpass_table cal.b.BP --bf_phases ~/src/delay_engine/phases_b.txt
--cfreq 1400
```

4 Update the Delay Engine Solution and Rerun

The output of the calibration script, i.e. the new solutions, remains in the directory that we run it from. We must then copy it to the delay engine directory and replace the old solutions. An important note: **make sure to backup the old solutions before replacing them**.

We can then rerun the delay engine and start a new observation with the new solution. Ideally, the next dataset should be corrected for delays and phases.