LWA and SSINS

Final project write up

Jonathan Julius

2018

Contents

[Original Proposal 1](#_Toc532505666)

[Familiarization 2](#_Toc532505667)

[Compatibility Issues 3](#_Toc532505668)

[Deciphering File Information 4](#_Toc532505669)

[Combining Files 5](#_Toc532505670)

[Results 6](#_Toc532505671)

[Conclusion 7](#_Toc532505672)

[Resources and Code 7](#_Toc532505673)

[Cheat Sheet: 9](#_Toc532505674)

[Figures 10](#_Toc532505675)

# Original Proposal

Recently data was received from the LWA (Long Wavelength Array) telescope in New Mexico as part of an astrophysics research project. For my project I will be using the SSINS (Sky Subtract Incoherent Noise Spectrum) python package developed my Mike Wilensky to identify rfi (radio frequency interference). After identifying the rfis I will create a dictionary to help use methods in the SSINS package to filter them out from the background. The LWA data will have unique properties that we will need to consider while going through our analysis.

The data from the LWA measures a lower frequency of radio waves then the ones previously studied from HERA and the MWA. The type of rfi would be different because of that. Whereas the rfi for HERA and MWA came from television sources, the lower frequencies from LWA would probably experience rfi signals from FM radio or other sources. Also, the location of LWA is going to in general have more rfis due to its location in the more populated America. Once done hopefully we will have developed a tool to help us better analysis the LWA data.

This project took place in five phases: Familiarization, Compatibility Issues, Deciphering File Information, Combining Files, and Final results. This document will go over all five phases in details going over the process of how I was able to get the new data to work with the SSINS package, as well as briefly go over ideas for further work needed to be done. At the end of the document there will be three appendices, one with all the figures, one with a list of the code files I used and their descriptions, and a quick cheat sheet on how to run further LWA data through SSINS.

# Familiarization

Coming into this project I had no familiarity with the EoR research project. Therefore, the first step I needed to take in this project was to learn all I could about the tools I would using. In addition to being unfamiliar with SSINS, while I had only a rudimentary knowledge of python. I have experience with code, but more so with object-oriented languages such as Java. To add to the difficulties, the data and packages I would be using were on a mac, and my primary experience if with windows. In short, I had to quickly learn or remember computer skills that I was rusty on. This task made up the first part of my project. Working with Mike, he walked me through the concepts behind his SSINS package and helped set up a work station on the shrivelfig mac in the UW physics building. I would either work on this machine or I would ssh into it from my windows device, where I would then run the necessary python scripts.

From there my first task was to learn how to use SSINS. Fortunately, Mike wrote up a basic tutorial (Link to it resources section) for me to use. The tutorial was informative and once you learned the basic commands using the SSINS package was easy. I then took data from that was supplied by Mike from the MWA observatory and ran it through the SSINS package. While I was not breaking new ground with these techniques, it did provide me a solid foundation for the rest of the project. Working with SSINS not only firm up my understanding of SSINS itself, but it also helped me gain a greater footing working with python and macs. This helped me moving forward into the further stages in the project. An example of the type of figures I made in this stage can be seen in fig 1.

As well as working with the SSINS package I worked with Mike himself as he walked me through the more high-level concepts of the project. One of the main things we talked about was how to calculate the significance threshold for using SSINS match filter. While his tutorial gave a generic number to use, we talk about how to calculate this statistic given a set of data. We discussed how given a gaussian distribution and a total number of counts, how to calculate at what interval could we expect less than one count. This was then calculated further on for the lwa data with more accuracy.

# Compatibility Issues

We received the LWA from Jayce Dowell and immediately ran into an issue running the data into a pyuvdata object. The CASA measurement sets would not read into pyuvdata. Mike Wilensky investigated the error as he had more knowledge of pyuvdata. He found that pyuvdata was making assumptions about the history and organization of the measurement set, specifically an empty history table was producing an error. It turns out pyuvdata was expecting a “weights spectrum” column to be filled in the measurement set, however this was not required as long as the less specific “weights” column was filled out.

With the source of the error discovered Mike made the changes and created a branch of the pyuvdata repo. He then tested the changes to ensure that the branch still passed existing unit tests. Once those tests were passed, he submitted a pull request with the master branch. Bryna Hazelton and other collaborators then double checked the test. They decided an additional unit test should be constructed in order to cover the new code. Bryna wrote this test. After this test was constructed the branch was successfully merged into the master pyuvdata branch. With the compatibility issues out the way, the next steps could be taken.

# Deciphering File Information

The data received from the LWA was in a folder that contained hundreds of files. All the information we had given on these files is that one of the frequency sets was off-set in time compared to the other and had less data. The rest we had to gleam from running it through a script and file names. An example of a file name we had would be “test\_adp2\_0\_00300673800807520000\_58342\_05\_00\_14.ms “. Before diving into the code, I took a good hard look at the file names. Based off their names I created a set of hypotheses that I then tested in my code to help ascertain the nature of the data. Going through the file name, a pattern became clear based of the last series of numbers. The last number in the file name would increment in a series of ten but was never high then sixty. When it would increment past sixty it would go down to “06” or “04” and the number to the left of it would increase by one. That number would then increment up by one until it hit fifty-nine when it then would reset back to “00” and the number immediately to its left would go up by one. Using this it was hypothesized that the files were separated by ten seconds and that the entire folder was the data from an hour of observation.

The next thing noticed was that there was an “adp#” in every file, and there was an adp one, two, three, four, five, and six. In addition I noticed adp1’s time stamps were offset by 2 seconds and start at minute 38 while the rest of the adps started at minute 00. Seeing as the time off-set was given to me by Jayce, I hypothesized the adps to be the frequency sets, and that adp1 was the frequency set that was offset in time.

With these hypotheses in mind I then went to test them in python. I uploaded the “first time” from each “frequency set”, printed out their frequencies and times. I found each adp did in fact represent a different frequency. The frequency range of the entire set ranged from 34.6-45.4 MHz. Each adp represented a 1.8 MHz slice, each with internal intervals of 25 KHz. I then checked the first two and last “times” for adp2 to find that they were offset by ten seconds and that the entire data set occurred over an hour. With our hypothesis confirmed it was time to move on to the next step.

# Combining Files

Now with all the information regarding the data known, it was time to run it through SSINS. However, each file of LWA data had only one-time integration, meaning it was incompatible with SSINS. In order to run it through SSINS I would need to combine the data. Fortunately, pyuvdata had an add method which was simple enough to use. Or so it would seem. When I first tried to add two pyuvdata objects together it would not let me due to the object names on the pyuvdata objects not being the same. After looking into the issue with Mike it came to light that the LWA did not follow a source when it took its data. It simply left the sky move around it. This means the polarizations of the files would all be different. In order to deal with this, each pyuvdata object had to be unphased to drift, and then phased back to the same orientation. Since the object name of the pyuvdata object was set according to those polarizations I set them all equal as well. With the meta-data finally situated the adding was nearly ready to begin. The final hitch, code wise, was that a pyuvdata object constructed of multiple time files of a frequency set would not add in a single time file of another frequency set. In order to add them together, 6 separate pyuvdata objects of the same size had to be constructed (one for each frequency set) and then added together.

Once some objects were finally combined and ran through SSINS, I ran into some issue with the data. While the ‘XX’, ‘YY’ and ‘XY’ Polarizations looked find, the ‘YX’ polarization seemed to have some error in it that I needed to fix as seen in fig. 2. Something was happening to leave the data fields in ‘YX’ seemingly blank. The ‘YX’ polarization was then tested for the presence of nands in the data and came back positive. When tested if all data were nands it came back negative. After thinking long and hard about this Mike came up with an idea. He thought since ‘XY’ has the same autocorrelations as ‘YX’, the lwa just filled in the autocorrelations for ‘YX’ with nands. The data set was ran again, this time removing all autocorrelation data and it returned data from fig 2, which was much cleaner. It was at this point I finally had the ability to run the lwa through SSINS and receive an INS to analysis for RFIs. It was time to move on to the next step.

The program was set and began to run but ran into another hitch. This program was initially running over SSH, and after a while, the program slowed down considerably and then the connection would break. The chunk of time was cut down to 5 then 4 minutes and connection tests were in place, but issues were still had. Finally, I ran the program on shrivelfig in person and found that my program would take up too much ram if it tried to chunk together more than a minute worth of data. In order to rectify this, minute chunks of data were added together as pyuvdata objects, then converted to INS objects from the SSINS package, and then combined that way. The INS objects took less ram to combine then the pyuvdata objects, making this a feasible work around. However, the one downside to using SSINS to concatenate is that the concat method does not take the differences between the end points of the INS objects in concatenates. Therefore, it is missing those times. So finally, after all that I got the data for six 10-minute chunk of time across roughly 10 MHz. I left out data from adp1 as to not have to deal with the time off set and the issues it would bring.

# Results

The figures 4-9 are the plots made of the 10-minute combinations of the INSs. As can be seen the data is rather noisy. Despite that, there were a few specific reoccurring rfi that begin to pop out. The most prevalent of these rfis occurs at 44 MHz. This signal appears in every chunk of data. Another reoccurring, but smaller rfi seems to occur around 40.4-40.7 MHz. With these in mind I calculated the sig threshold needed for the match filter. This was done by multiplying the Ntimes-1 with the Nfreq and the NPols for each SSINS object. I then multiplied that by the number of SSINS object I concated together to get 72,000 counts. Plugging that in to find my interval my significance threshold ended up being around 4.35. I then took the dictionary and significance threshold and plugged it into a match filter and set the streak and point filters off. The results of that are figures 10-15. While still very noisy, the filters effectively removed from each data the two key rfis found. It looks like other rfi might be there, but they were more acute. Further data would be needed to pin down if they were regular rfis.

# Conclusion

With the dictionary constructed we begin to look forward to what’s next Looking through further LWA data at different times to see if other rfis are present or if the same rfis are still present. Investigating possible causes of rfis in the dictionary to see if they cannot be mitigated. The 44MHz rfi seems very present. However, I could not pin down what a possible source might be. And of course, now that SSINS has been implemented on lwa data, applying that to the actual data to see what we can observe.

This project taught me a lot of techniques in large data analysis. Mainly this project had me work a lot with provenance. One of the main take-aways I have from working on this project is the importance of good documentation of your work in code. The tutorials and guides for pyuvdata and SSINS were invaluable as I tried to work my way through the project, while the lack of documentation regarding the data from the LWA was the greatest hurtle. With that in mind I attempted to create a quick and easy cheat sheet for those who follow in my footsteps. I hope my quick guide will aid those in an attempt to work with the LWA data in the future.

# Resources and Code

https://ssins.readthedocs.io/en/latest/tutorial.html?highlight=tutorial

This was Mike’s tutorial for SSINS. This notation proved vital and learning how to implement SS and Match filter.

Jon\_kickstart.py

This code was used primarily for familiarization. In it I took data from the MWA and ran it through the SSINS package and match filter. Mainly I followed the steps listed in Mike’s tutorial.

lwa.py

This code was used to add all the lwa files together and run them through SSINS. It creates five arrays of path names that then get used to create a minute long uvdata object for adp2-6. At the end it then combines the 5 frequency sets and runs it through SSINS and saves the data.

test.py

This code was used to test initial hypothesis and trouble shoot issues when combining data. Individual uvdata objects were fed in specific file names and then either their data or meta data were tested. Additionally, they would be combined in order to troubleshoot errors occurring.

INScombine.py

This code took the INS data of 10 minutes’ worth of files and combined them together using the concat method in SSINS INS\_helper. Additionally, once rfis were identified this is where the match filter was ran on the data from the LWA.

LWA Cheat Sheet

test\_adp2\_0\_00300673800807520000\_58342\_05\_00\_14.ms

The adp refers to the file’s frequency:

|  |  |
| --- | --- |
| Adp1 | 34.6-36.4 MHz |
| Adb2 | 36.4-38.2 MHz |
| Adb3 | 38.2-40 MHz |
| Adb4 | 40-41.8 MHz |
| Adb5 | 41.8-43.5 MHz |
| Adb6 | 43.6-45.4 MHz |

File’s time stamp in HR\_MN\_SC in 10 second intervals

# Combining Data:

For SSINS to work you need multiple time integration, and each data file from the LWA contains only 1, so they must be combined in order to work. Some tips:

1. The polarizations of the files don’t match. In order to add them you must unphase them to drift before combining them. In addition, because of this their object names don’t match, you have to set them to equal each other as well.
2. Combine uvdata objects of each frequency for the amount of time you require, then add all the objects at the end.
3. **Do not include autocorrelations**
4. Unless your machine has more then 20 GB of ram do not combine more then a minute worth of UVdata objects

Known probably RFI frequencies: 44MHz, 40.4 MHz-40.7MHz

# Figures

Note: Figure sizes are hard to see on documents. I have also attached all the figures used here in a separate folder, I have their names in folder listed.

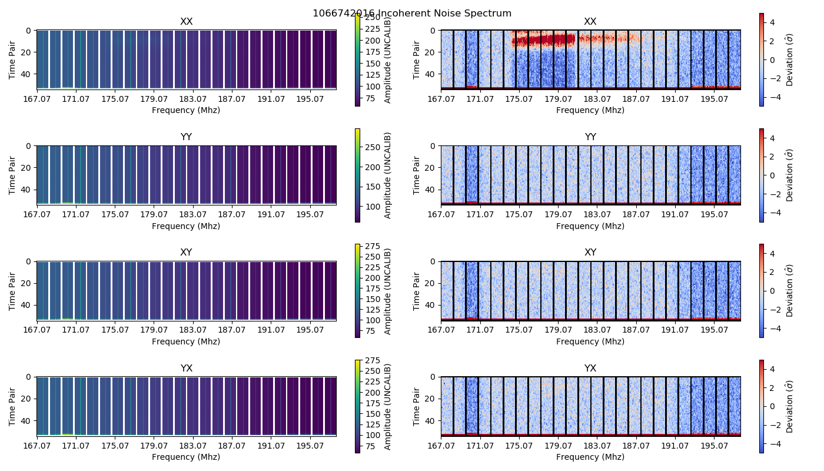


Fig. 1: A basic figure made from the familiarization process of the SSINS package. Data taken from the MWA. Labeled Familiarization.

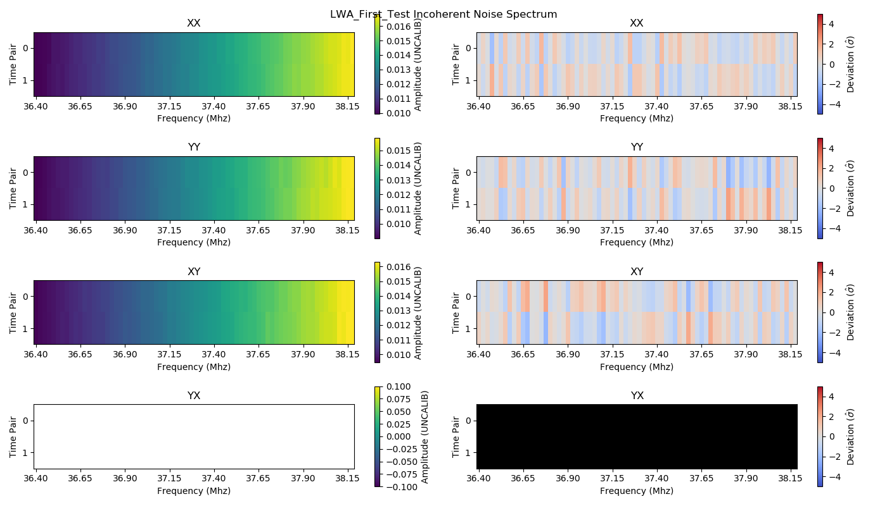


Fig 2. First test of combined lwa data. Only 3 baselines here. Notable issue with YX pol. Labeled Bad XY

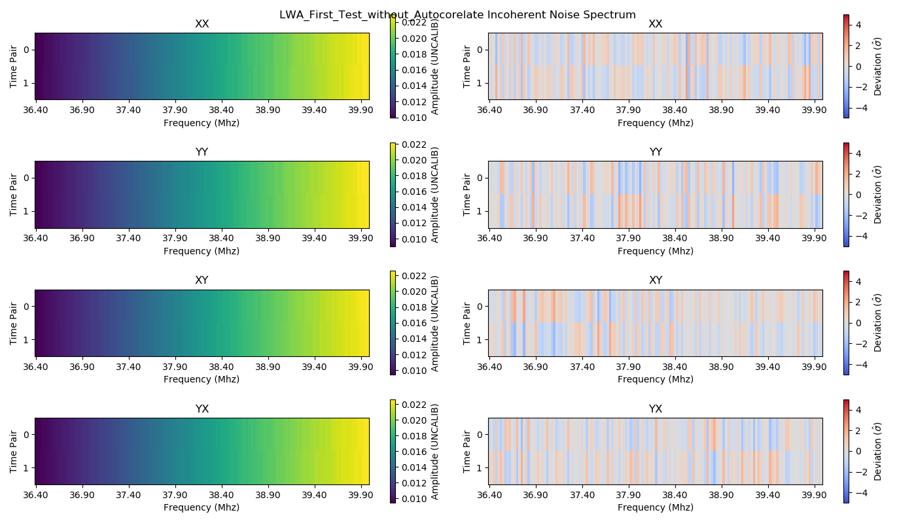


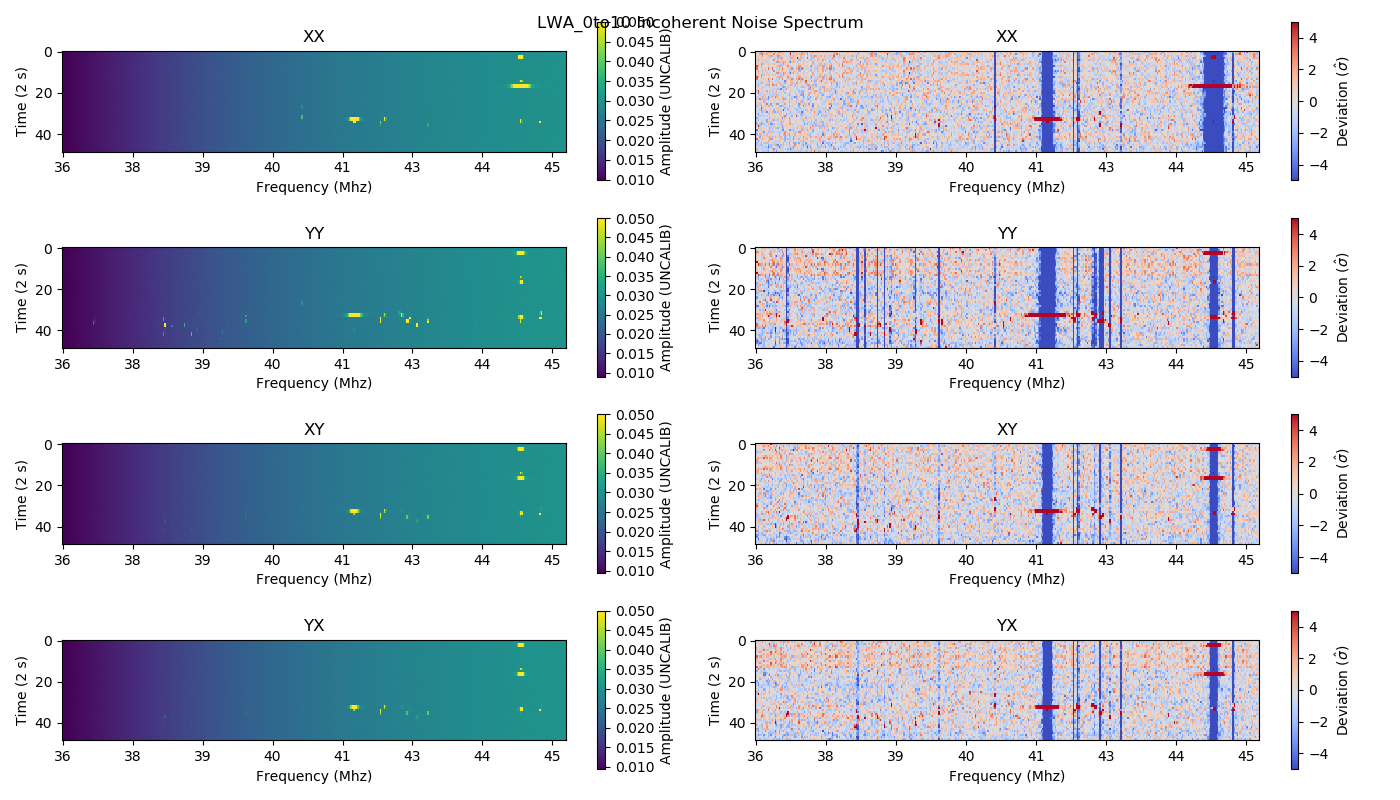
Fig 3. Test without Auto correlates. Again only 3 baseline, but no issues. Labeled Without Autocorrelates.

Fig 4. LWA Data, from 0 to 10 minutes.

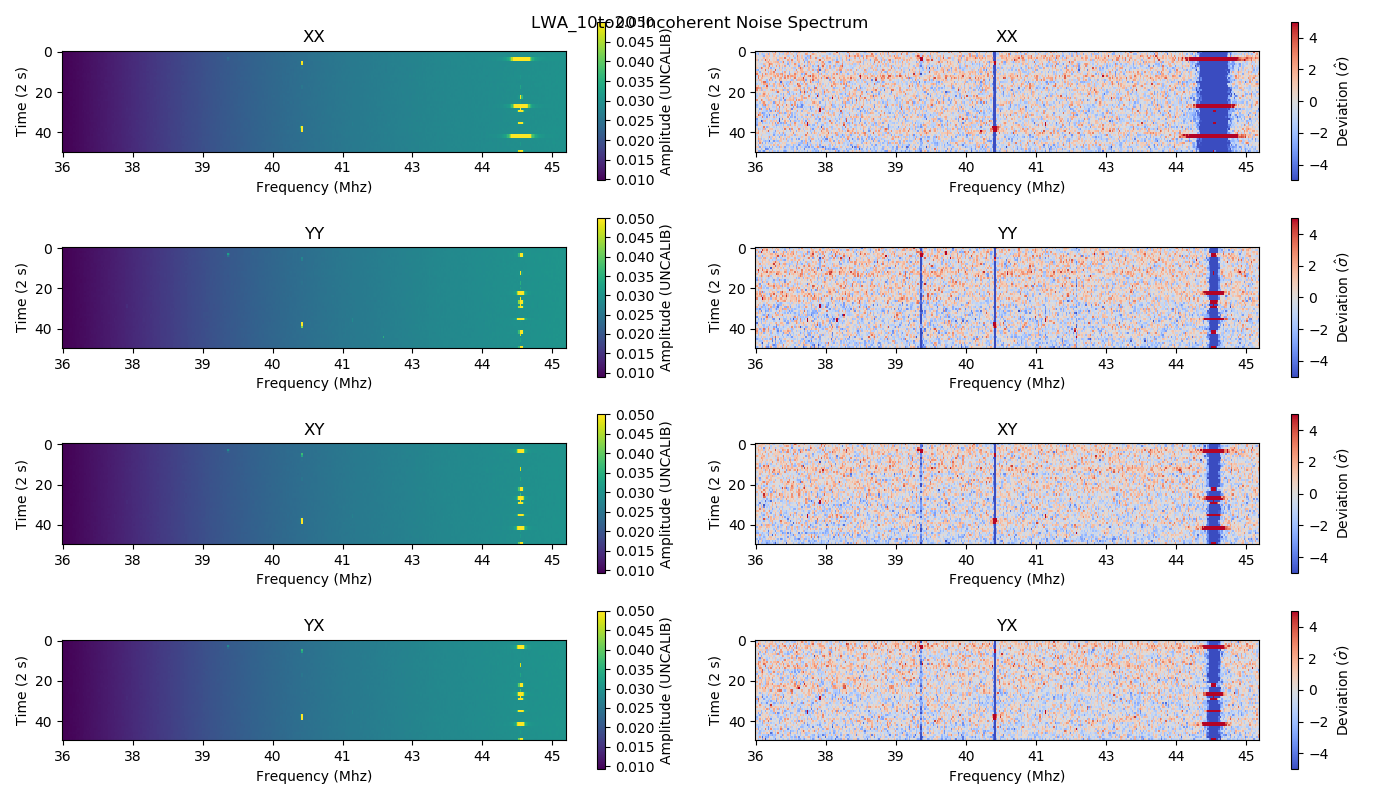


Fig. 5 LWA Data from 10 minutes to 20 minutes.

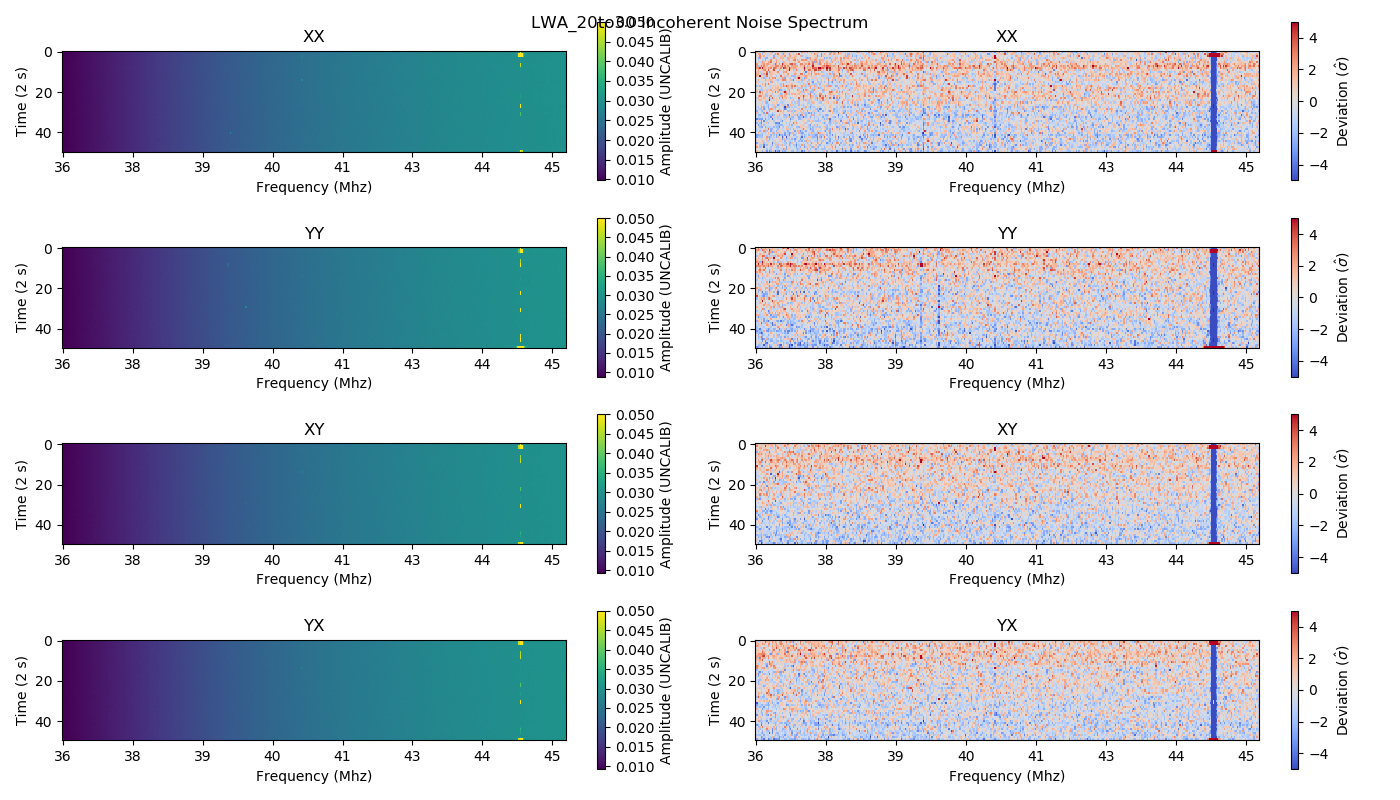


Fig 6. LWA Data from 20 minutes to 30 minutes

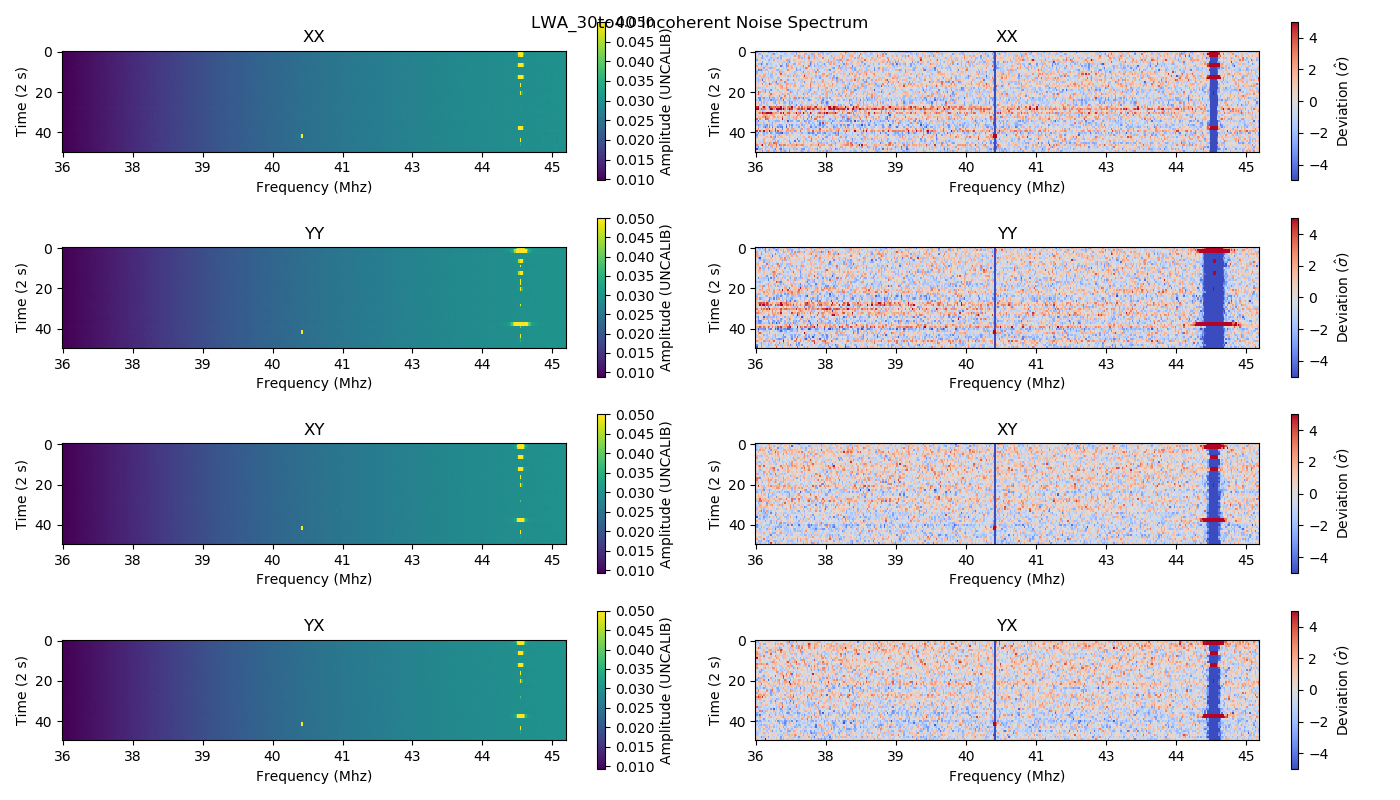


Fig 7. LWA data from 30 minutes to 40 minutes

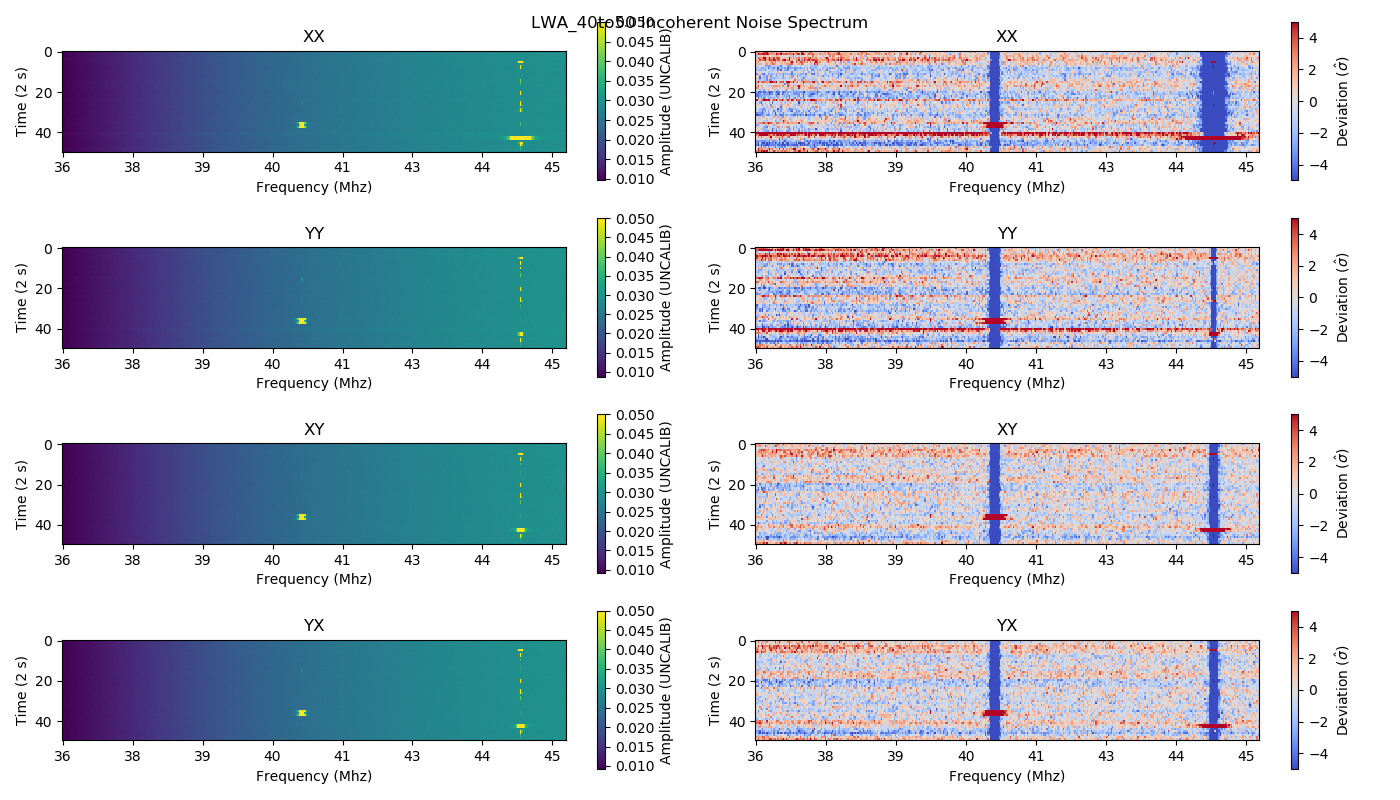


Fig 8. LWA data from 40 minutes to 50 minutes

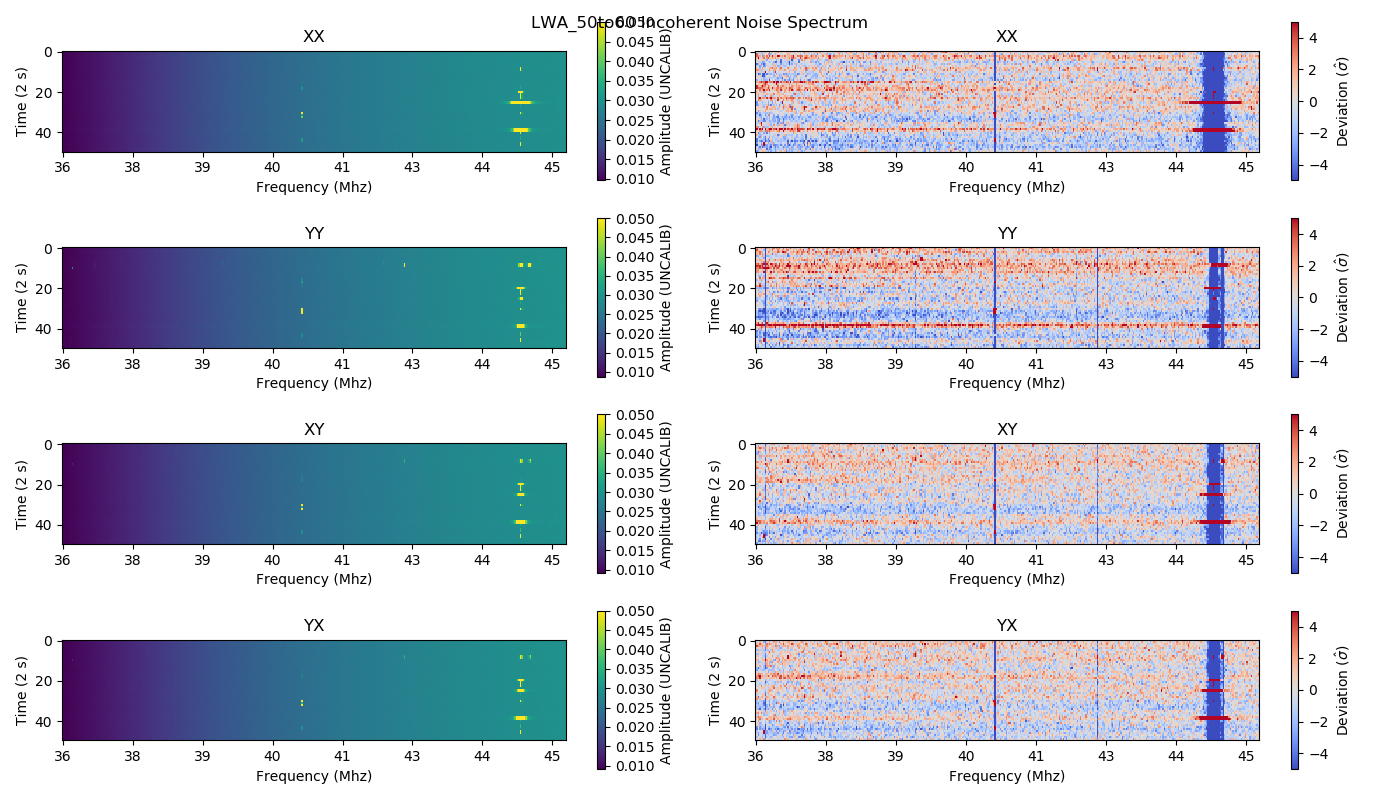


Fig 9. LWA data from 50 minutes to 60 minutes

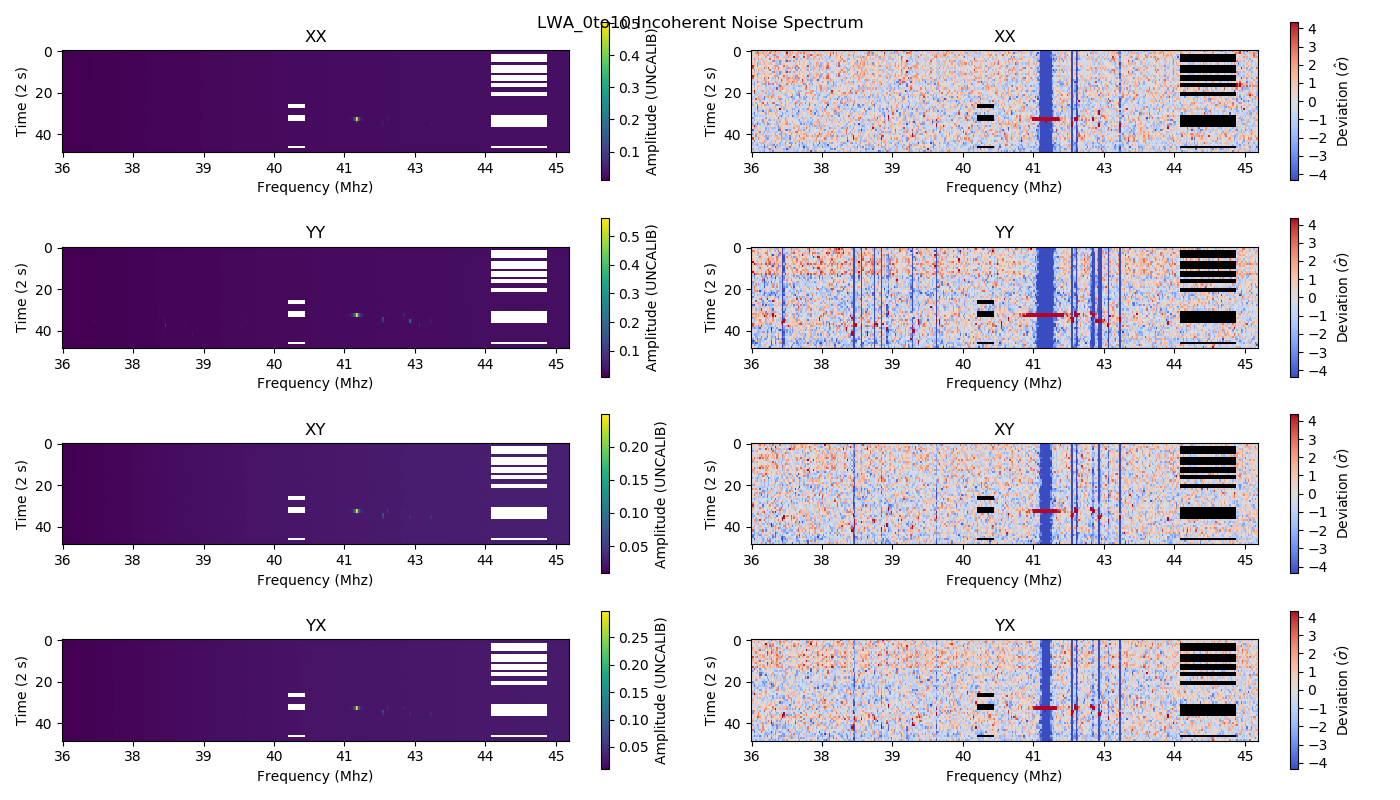


Fig 10. LWA minutes 0 to 10 with filter.

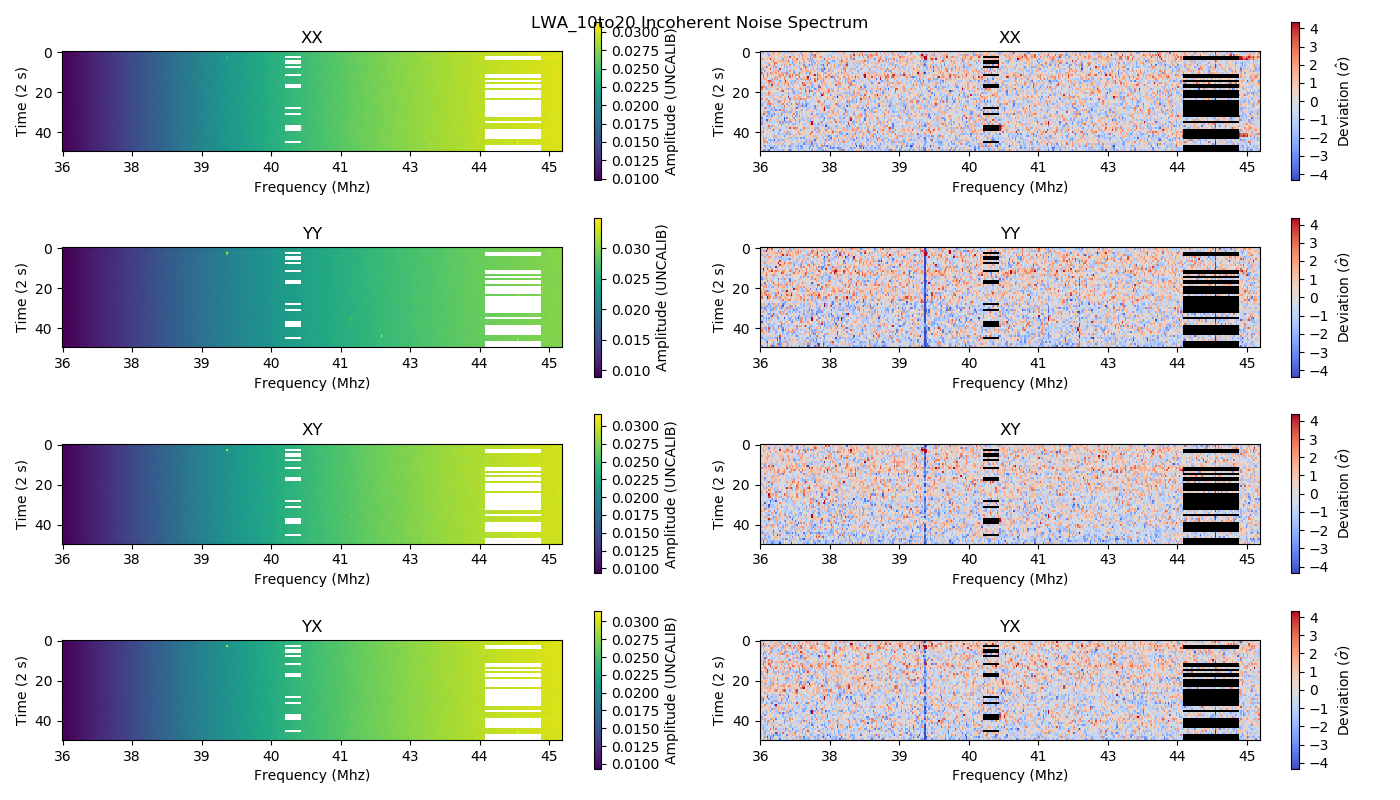


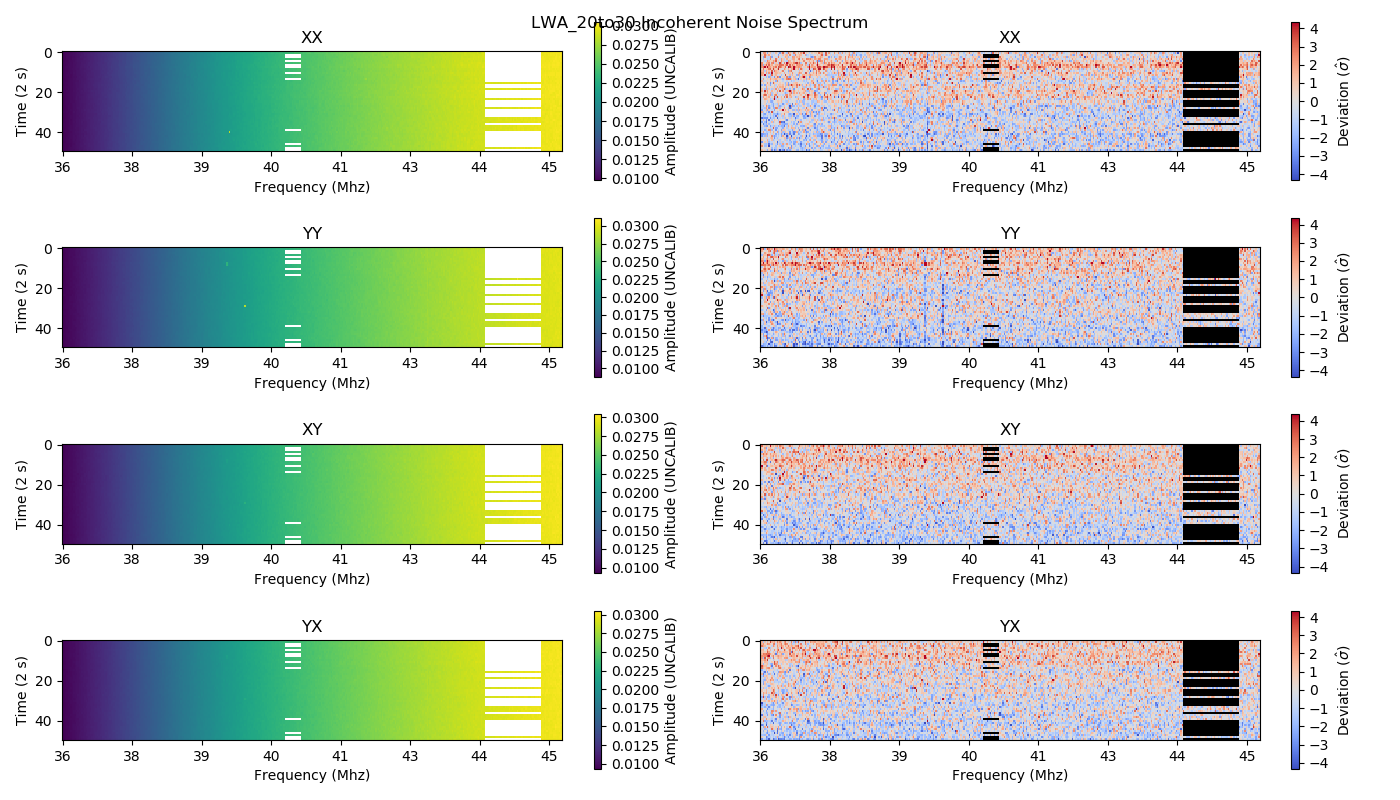
Fig 11. LWA minutes 10 to 10 with filter.

Fig 12. LWA minutes 20 to 30 with filter.

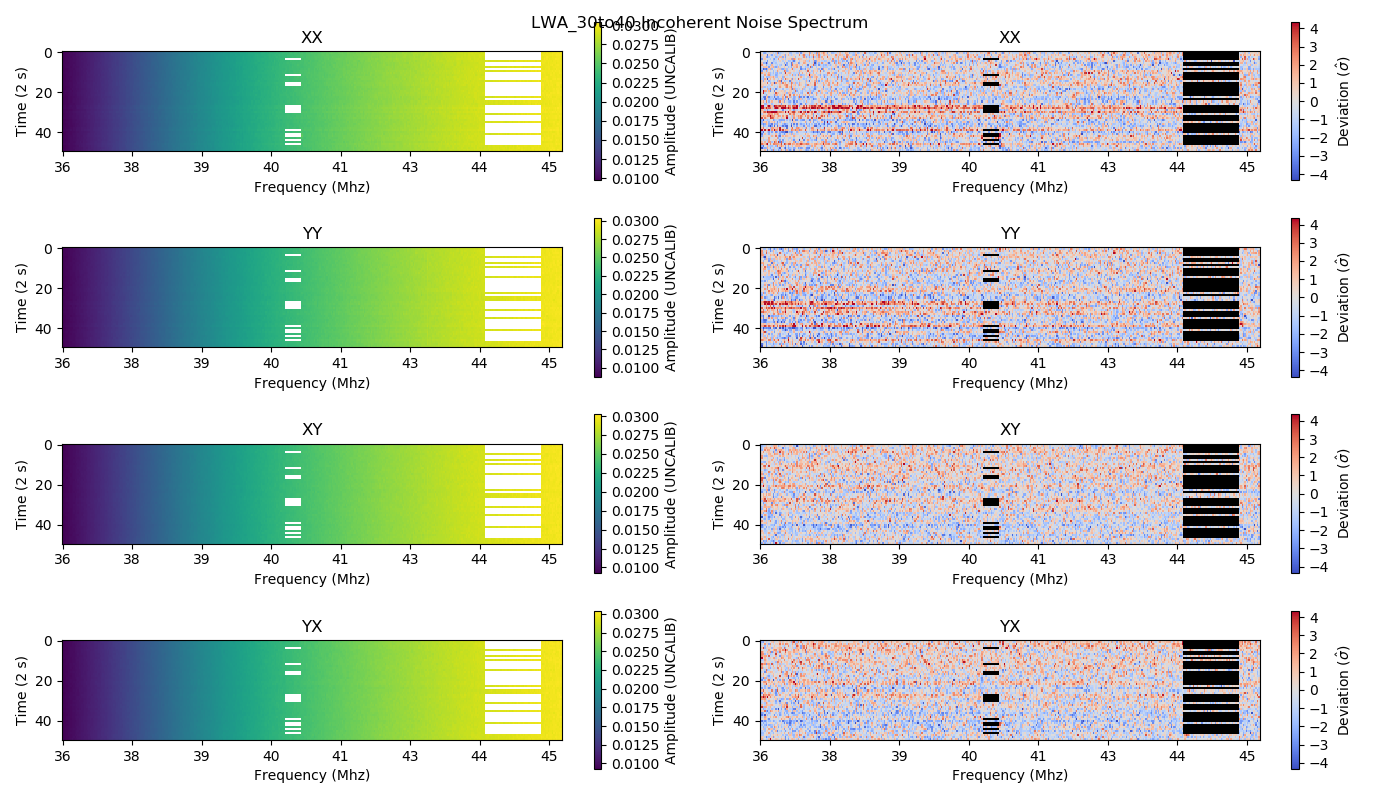


Fig 13. LWA minutes 30 to 40 with filter.

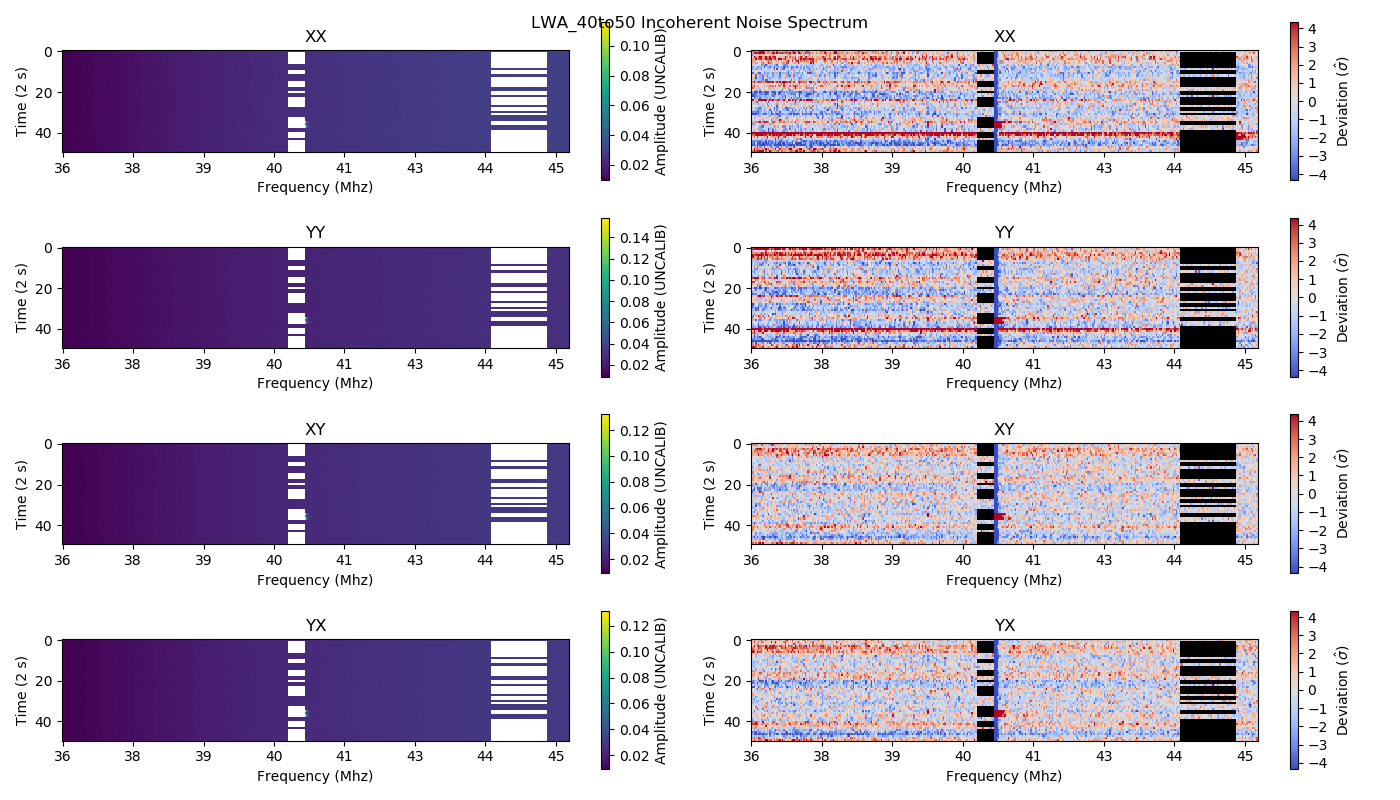


Fig 14. LWA minutes 40 to 50 with filter.

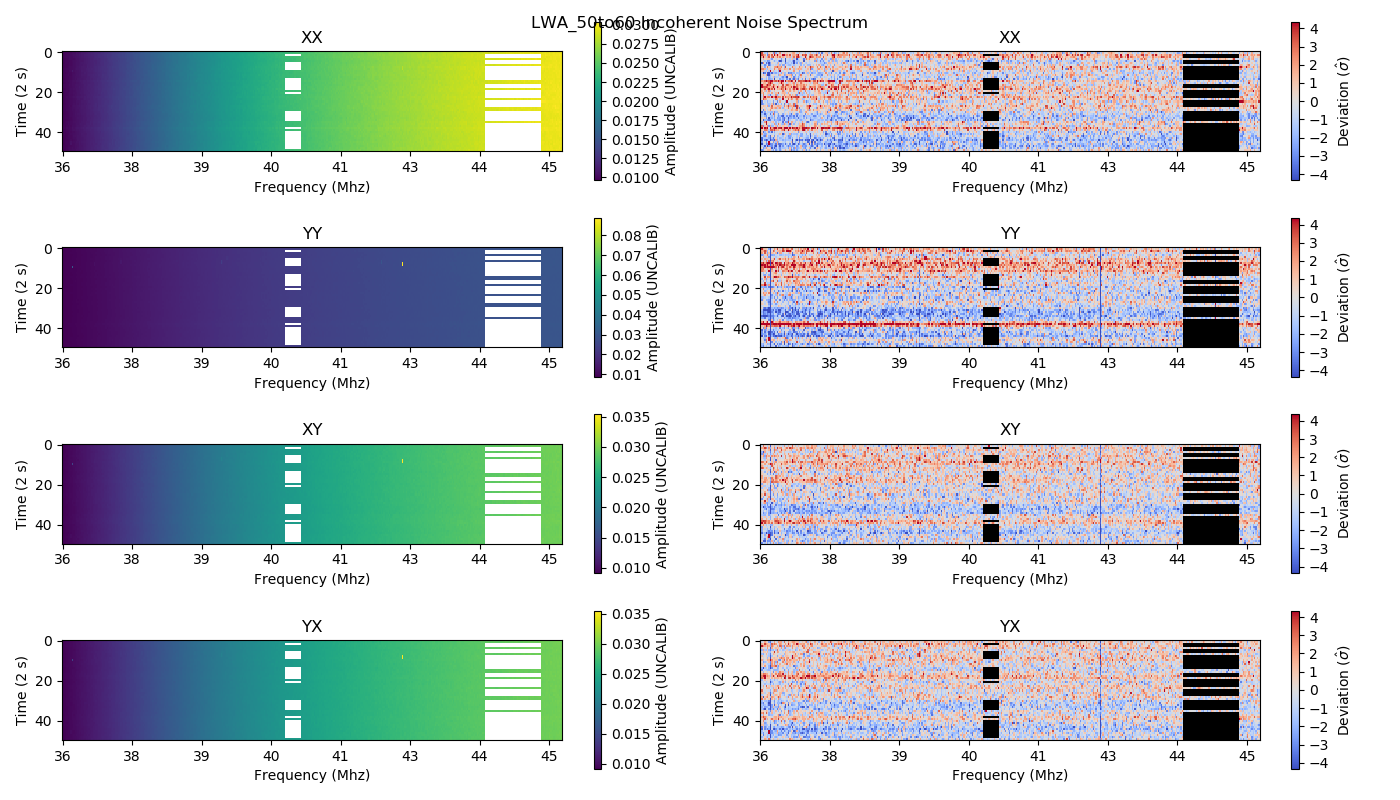


Fig 14. LWA minutes 50 to 60 with filter.