Running Analysis:

Location of data:

Data is stored on kingbee server at OSU. The 10% and 90% samples are stored at

/data/anita/btdailey/anita_data/sample10/

and

/data/anita/btdailey/anita_data/sample90/

respectively.

The main code can be found on github at

https://github.com/btdailey/4Pol_backup/Useful_files

This contains all my working code. (analysis, queuing, drawing etc)

MyCorrelator.cxx is the main code for the analysis. It can be compiled using Makefile.

MyCorrelator has a myriad of options that can be used to run. The main flags to check are the NotchFilterFlag and phase_flag. NotchFilterFlag sets the type of the amplitude filter (3= interpolated method) and phase_flag sets the phase filter (3= geometric method).

Other options can be used for the NotchFilterFlag and phase_flag. All the options are as follows.

NotchFilterFlag:

0) No-Fill: Notched region in amplitude spectrum is not filled. Phase does not matter for this since the amplitude is 0.
1) Rayleigh: Fill the notched region with Rayleigh distributed noise, based on thermal baseline. Can use any phase option.
    a. Abby's method: By going into MyCorrelator.h, you can change the variable "newnotchflag" to 0 to re-produce Abby's filter. (You may also change the normalization flag to 0 to get her interferometric mapping as well)
2) Wiener: Use this option to apply the Wiener filter to the notched region. Can use any phase option
3) Interpolated Method: The method used in my analysis. A linear interpolation across the notch in the amplitude spectrum. Can use any phase option.
4) No-Notch: No filtering done. Must use phase_flag=0, since no notch is calculated, no phase change is possible.

5) Sine Subtraction: Use implemented sine subtraction technique to remove CW. Must use phase_flag=0, since the sine subtraction also changes both amplitude and phase of waveform.

Phase_flag:

0) Old Phase: Use the original phase of the waveform.
1) Random Phase: Calculated a random phase between –Pi and Pi inside the notch region that was calculated during Amplitude Spectrum Filter steps.
2) Interpolated Phase: Linear interpolation across the notch region that was calculated during Amplitude Spectrum Filter steps
3) Geometric Method: Method used in analysis. Attempts to correct the phase inside the notch region using a derived formula.
4) Simple-Shift: Not implemented correctly in this version. Returns Geometric method.

//////RUNNING THE CODE//////
After compiling, there are various scripts I used to queue Anita data runs. They are as follows:

Batch10sample.sh: Calls **loop10sample.cc**. Qsubs 10% sample

Batchfilterpulser.sh: Calls **loopfilterpulser.cc**. Qsubs Taylor Dome (calibration pulser) sample

Batchfilterthermal.sh: Calls **loopfilterthermal.cc**. Qsubs thermal sample

Batchsimulationdata.sh: Calls **loopsimulationdata.cc**. Qsubs simulated events.

There is a second set of scripts that allows for splitting up a data run into multiple pieces. They are in the *_peices.sh codes. This reduces the time to run over the total sample.

These codes will create a temporary file on the server, then copy the output from the job to a directory you designate. Some computer systems can automatically create a temporary folder to hold the data, verify if you need that step in the script.

The code **pointeventdata.cc** allows you to look at single events. It also draws various plots such as the correlation maps and coherently summed waveforms. This option can be turned off by changing the drawMaps flag in the input to the pointhisevent function in the pointeventdata.cc file.

////////Conducting analysis////////
After having run over the entire data sample, you will apply various cuts that were not optimized for this analysis. This is done in **applyCuts.cc**. In this code, you need to read in the root files you wish to apply cuts to, specify an output directory and

name for the output root file. After running, it will place passing events into the specified directory.

The next step in the process is to place the events into HealPix bins. This is done in the subdirectory healpix_mapping. The code that performs this is **healpix_map.cc**. This code requires HealPix to be installed correctly. It will read a root file you specify (the output from appliedCuts.cc), place the events into the correct HealPix bins based on the location of the event (whole_flag==1) or weighted by the area of the error ellipse (whole_flag==0). The remaining events are placed into a root output file.

Caution, healpix_map.cc applies two analysis cuts near the beginning of the for-loop that were intended to be optimized along with the rotated cross correlation cut. These cuts (ratio of peaks and polarization fraction) can be moved to appliedCuts.cc very easily if you want.

Finding the slope of the rotated cross correlation cut was done using the code called **pval.cc**. This code reads in the output from healpix_map.cc. It loops over various slopes for all HealPix bins, creating the differential plots for those bins, and does a fit to the falling edge if it passes certain criteria.

This code produces a variety of plots for each bin: diffplots (differential plots, 1 for each bin for each slope), rotatedCuts (2D distribution of the cut we are optimizing. 1 for each HealPix bin), likelihood plots (likelihood distribution for HealPix bin, 1 for each slope for each HealPix bin), pval plots (1 for each slope), and pval_Dist (1 for each slope). The plot named pvalDist is a 2 dimensional histogram plotting the number of events in the bin against the p-value of that bin, while pval plot show only the p-value of the HealPix bins.

Again, caution. To make the correct rotatedCuts at this point, we did not apply an analysis cut (peakVal <0.075). It is applied before calculating the pval, so the results are correct, but the number of events may be inflated.

After choosing a slope and the HealPix bins you wish to use, you will use the code **optimization.cc**. In this code, the slope of the cut line and the bins you wish to use are hardcoded due to time constraints. This code reads in the output from healpix_map.cc, uses the slope and the bins you want and optimizes the y-intercept for each HealPix bin. It outputs the parameters in Cut_values.txt.

To apply the final cuts and see how many events pass, you will use **final_cuts_prebinned.cc**. You need to have the output from **healpix_map.cc** for this code to run properly. This is to help speed up the code, as a cut will remove any events that did not land in used HealPix bins.

The code will read in **Cut_values.txt** and applies all the cuts, including the rotated cut, to the events. The text output will tell you how many events were cut by each cut and if that cut were placed last on the list (significance of the cut). The last line informs you of how many events pass the analysis cuts (first number) and the fraction that passed the final cut (area>50% in the HealPix bin).