Exploring Data

Hamish Johnson

July 2020

1 Introduction

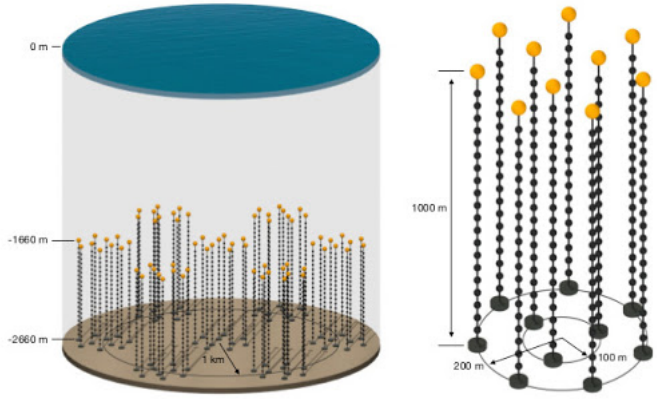


Figure 1: P-ONE Observatory

Shadowing Matthew Ens in his work with P-ONE Neutrino Observatory, specifically on dealing with noise in sDOMS.

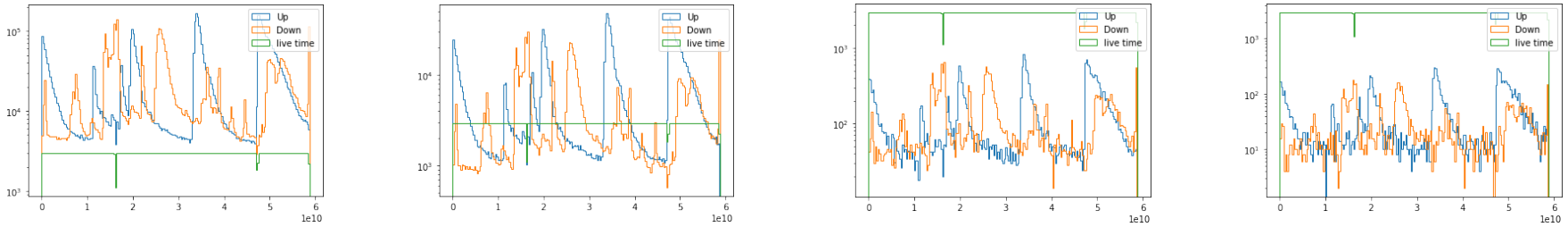
You can see various notebook files at https://github.com/Ham0osh/HamishPONE/tree/master/notebooks-%26-data.
Or contact me at: hamish_johnson@sfu.ca.

Contents

2 My Work

2.1 First Data Set

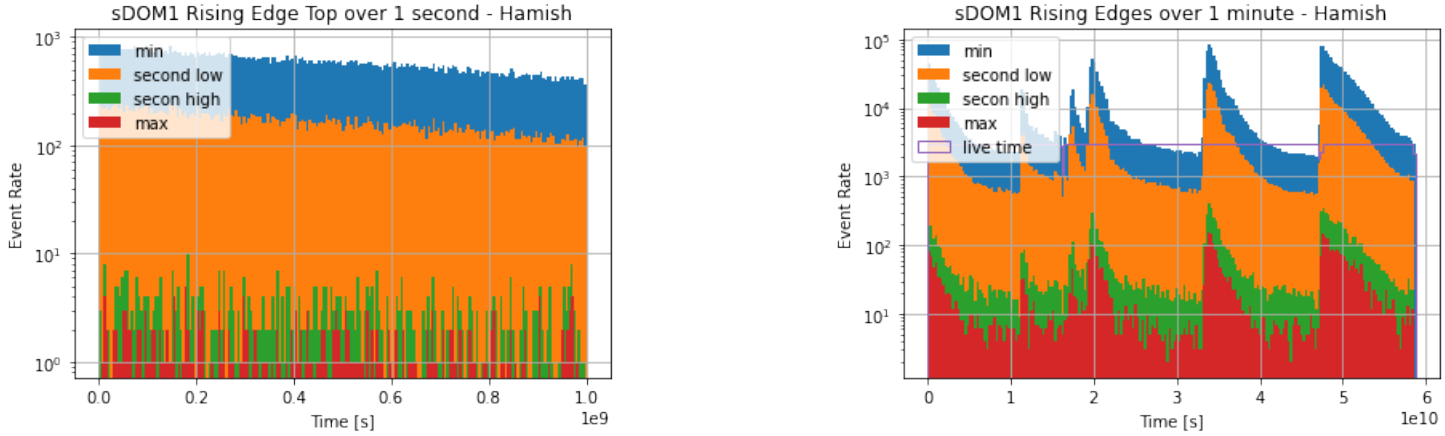
I started looking at the data file Matthew gave me by viewing the different threshold. This gave me a chance to better understand what the data points meant, some quirks of the measurements (downtime), and how to handle the file. *The horizontal is time [ns] and the vertical is number of events.*



(a) Low Threshold (b) Second Low Threshold (c) Second High Threshold (d) High Threshold

Figure 2: Trying different thresholds

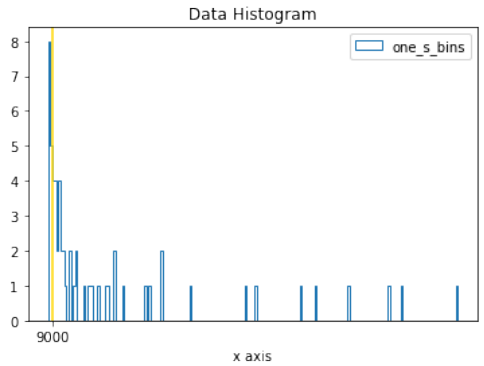
From here I wanted to see all of the events reaching each threshold together over one second to see small variations as well as the full one minute to see the spikes.



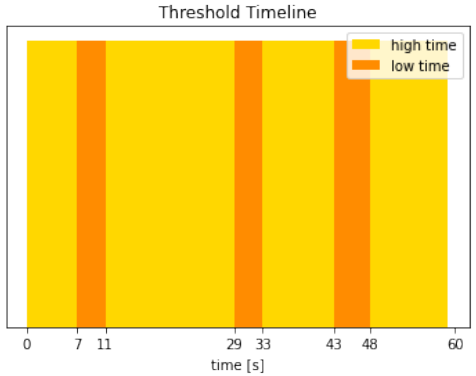
(a) One Second (b) One Minute

Figure 3: Rising edge hits coloured by threshold

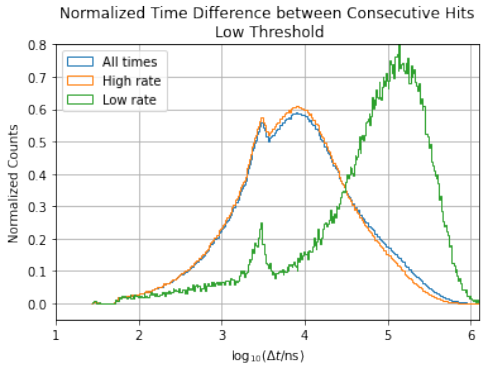
From here I needed to re-create the time differences between consecutive hits on my way to looking at coincidences. I started by reading though and rewriting the code to produce *fig 4.c*. Because data was being sorted into high times and low times I made *fig 4.b* which is just another way of seeing when "min" (blue) in *fig 3.b* passes above the threshold. I went to do this again for the highest threshold (for understanding), I first needed to understand how the 'high time - low time' threshold was set, leading to *fig 4.a* showing the threshold cutting out the bulk of the data.



(a) Low Threshold Cut

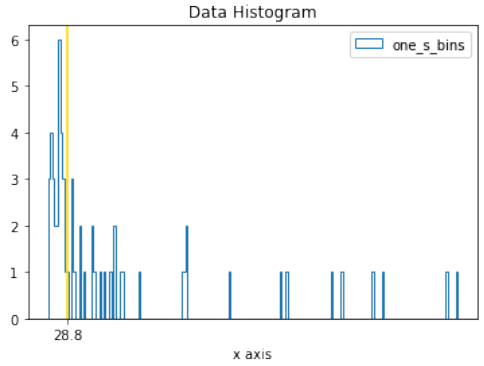


(b) One Minute

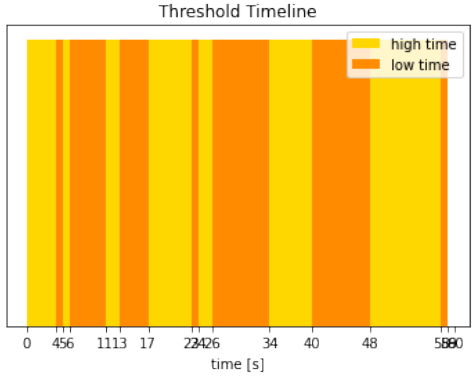


(c) Normalized Consecutive hits

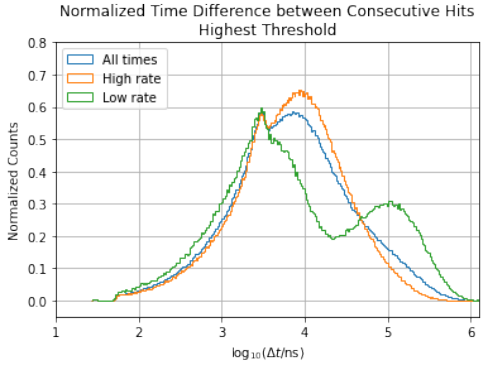
Figure 4: Low threshold hits with cut



(a) High Threshold Cut



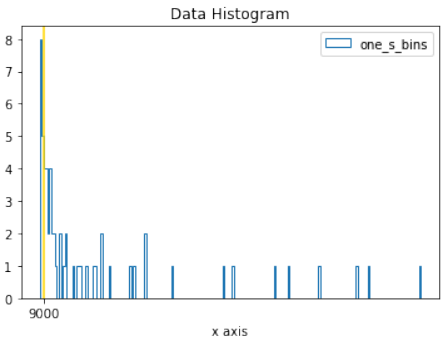
(b) One Minute



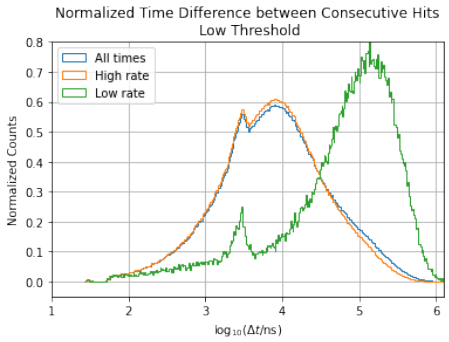
(c) Normalized Consecutive hits

Figure 5: High threshold hits with cut

Here is a comparison of all four thresholds to how the features change.

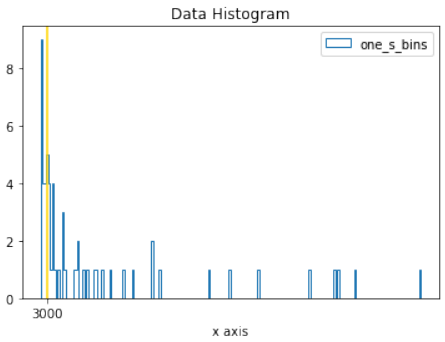


(a) Threshold Cut

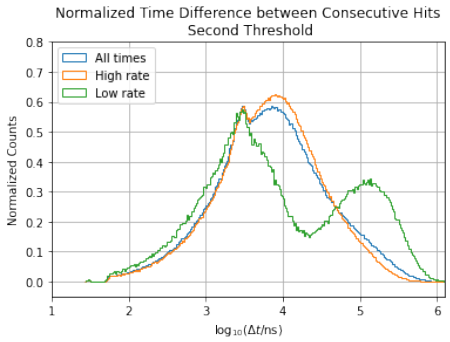


(b) Time Difference

Figure 6: First Threshold

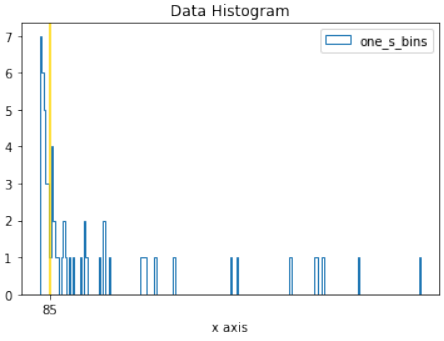


(a) Threshold Cut

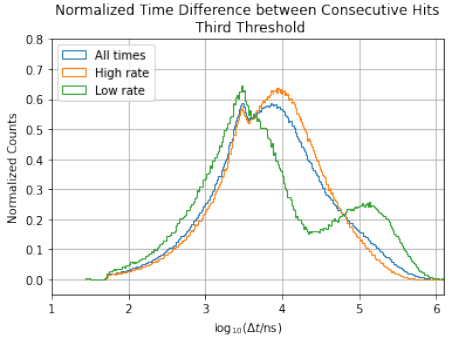


(b) Time Difference

Figure 7: Second Threshold

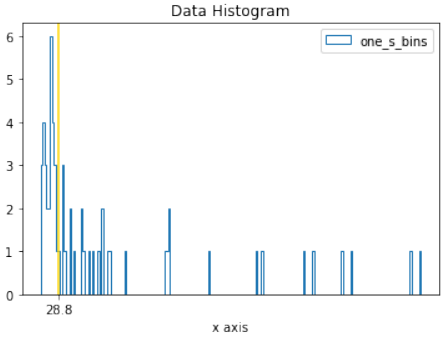


(a) Threshold Cut

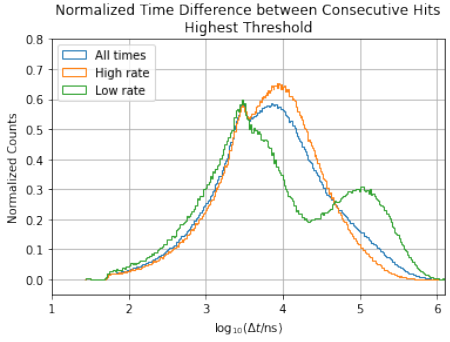


(b) Time Difference

Figure 8: Third Threshold



(a) Threshold Cut



(b) Time Difference

Figure 9: Fourth Threshold

There is a possibility that the cut off threshold between high and low rates needs to be adjusted, but assuming they are close enough we can see some changes and the thresholds get higher. Namely the long time noise seems to drop drastically, and the hits in low rate increase by the sharp zone to meet (overpass in threshold 3) the high rate hits.

I suspect this has more to do with my placement of the cut off value than the four thresholds themselves. I ran the last three thresholds again with more strict cut offs, however this caused the amount of low rate hits around and before the spike to increase again.

Now I am understanding the code to look for coincidences which I have only run once as it takes about 3 hours (down from 18 by reinstalling python through miniconda). Here is the code (typo when defining tmax as 1+tcoin instead of t+tcoin. I am now running this fix with half of the data set.

[4]:
```
upData =   np.sort(time[np.where( np.logical_and(channel==1, edge==0))])
downData = np.sort(time[np.where( np.logical_and(channel==5, edge==0))])
```

[5]:
```
len(upData)
```

[5]:
```
2399223
```

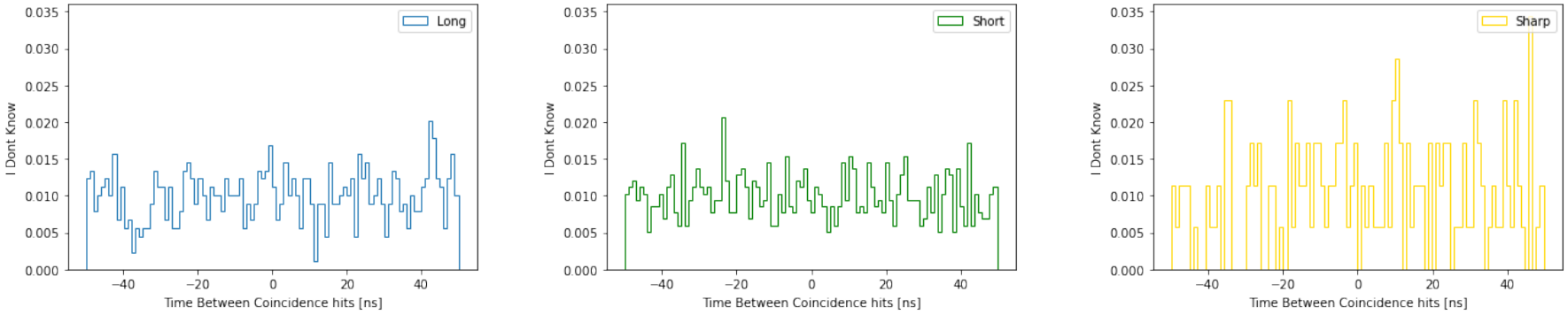
[6]:
```
len(downData)
```

[6]:
```
1985677
```

[7]:
```
def coincidences(up, down):
    longDiffs = []
    shortDiffs = []
    sharpDiffs = []
    masterList = []

    if len(up) > len(down): # decide which (up or down) is a longer list
        short = down; long = up
    else:
        short = up; long = down

    for i in range(len(short)): # parse over the shorter list to save time
        t = short[i]            # value at that index
        tcoin = 500             # coincidence time in 'ns'
        tmin = t-tcoin; tmax = 1+tcoin
        potentialList = long[np.where(np.logical_and(long > tmin, long < tmax))]

        if len(potentialList) > 0:
            if len(potentialList) > 1:
                # there are multiple values
                continue

            nextEvent = short[ i + 1 ] - t
            if nextEvent > 10**4.2:
                longDiffs.append(t - potentialList[0]) # low noise time?
            else:
                shortDiffs.append(t - potentialList[0]) # high noise time?

            if nextEvent > 10**3.4 and nextEvent < 10**3.6:
                sharpDiffs.append(t - potentialList[0])

            masterList.append(np.array([t, potentialList[0]]))
        else:
            masterList.append(np.array([0,0]))

    longDiffs = np.array(longDiffs)
    shortDiffs = np.array(shortDiffs)
    sharpDiffs = np.array(sharpDiffs)
    masterList = np.array(masterList)

    return longDiffs, shortDiffs, sharpDiffs, masterList
```

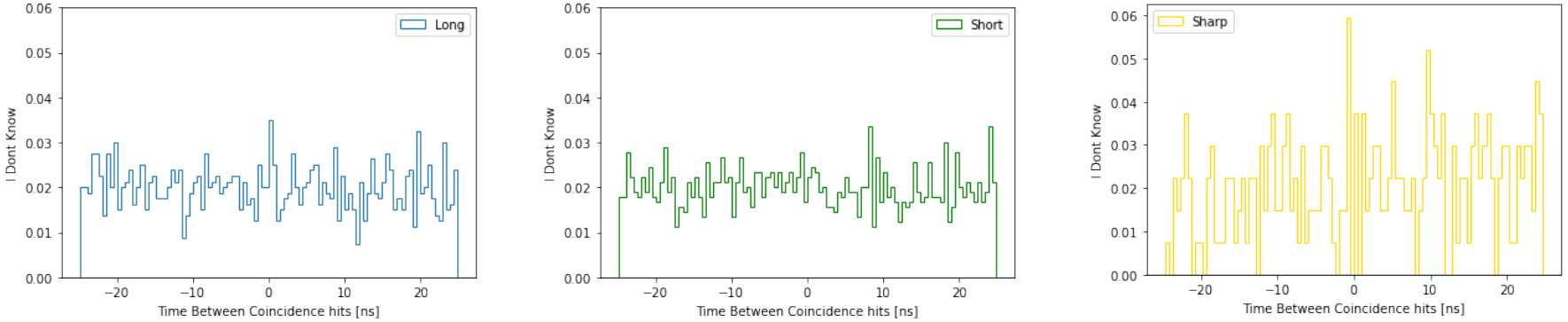
Figure 10: Code with error

Here I started visualizing the coincidence data with tcoin as 50 [ns] on a smaller data set. You can see three plots of the different categories all set to the same y-axis (of which I don't yet understand the units). After which you can see the same but run on with full data set with tcoin = 25 [ns].



(a) Long (Long Time) (b) Short (Short Time) (c) Sharp (Time in spike)

Figure 11: Time between coincidental hits Short Set



(a) Long (Long Time) (b) Short (Short Time) (c) Sharp (In the spike)

Figure 12: Time between coincidental hits Full Set

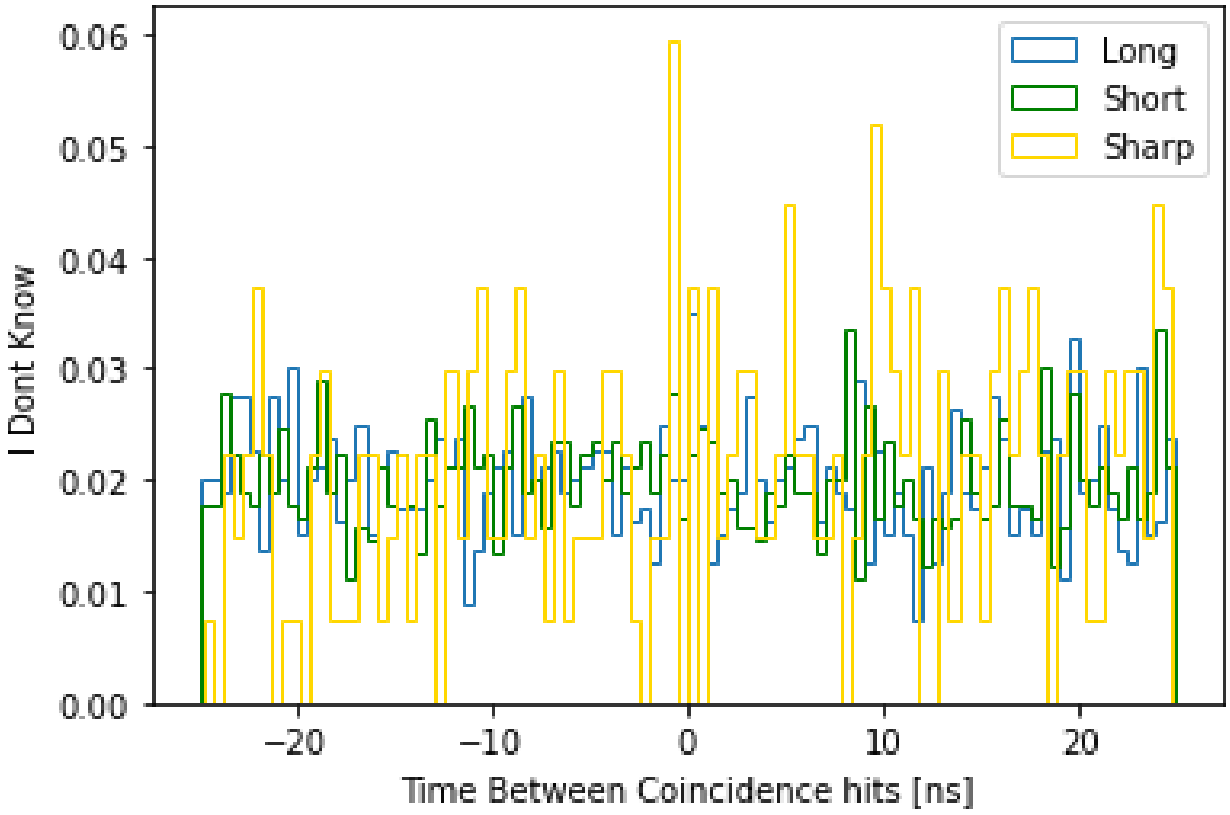


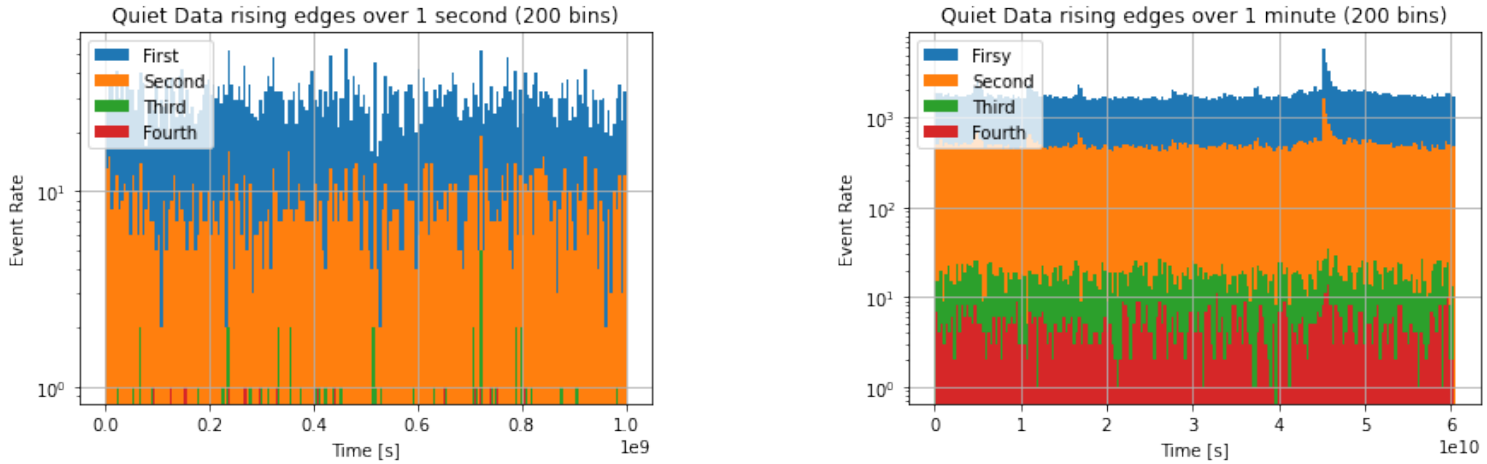
Figure 13: All sets from the full file

2.2 Quiet K40 Data Set

In this section we are dealing with a quieter data file to more easily notice the effects of K40. I will be doing the following analysis on all 4 thresholds again, but I am starting with the lowest threshold until I know it is working.

1. Bin the entire minute of data into bins of ms length (instead of s length)
2. Once you've done this, partition the run into ms that have between 1-20 hits and others (This is the hard step where you'll have to do the most thinking)
3. Using only these times make the 'time difference between consecutive hits' plots that you have already made for the other data file (Figure 4c and 5c). Since you already chose 'low' noise times in step 2 you will only be making the green curve, you could make the other two curves (orange and blue) by just looking at the time difference for all the data (blue) or by only taking the times that were filtered out in step 2 and not the ones you kept.
4. Do steps 2&3 for all thresholds, it would be interesting to see both the plots when you partition the threshold you are analyzing (ie take channel 2, take times between 1-20 hits ON CHANNEL 2 and then make plots) and the plots when you partition the threshold you are analyzing with the times from the lowest threshold (ie take channel 2 and use the times where there are between 1-20 hits per ms ON CHANNEL 1 and make the plots from those times)

Just to contrast with the data set, here is how the new is displayed the same way as *fig 3*.

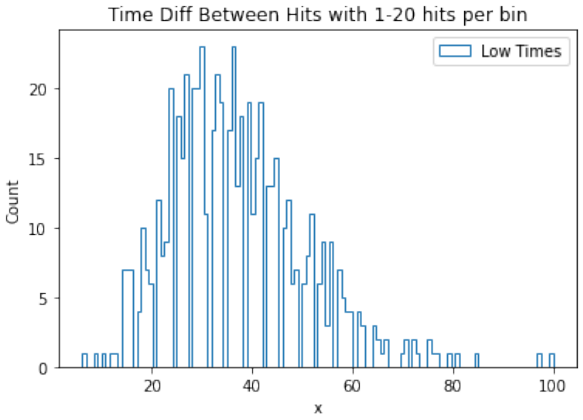


(a) One Second

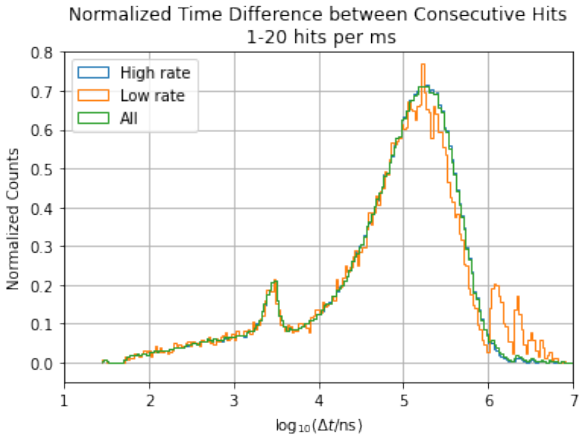
(b) One Minute

Figure 14: Rising edge hits coloured by threshold - Quiet

At first I ran my code with larger bins than 1 per ms so save on time, however I did not properly record my process. And only set up with git after I had changed my code. I believe it was run on a much smaller sample of the data file (between 10% and 50%). and I am working on re-creating it.



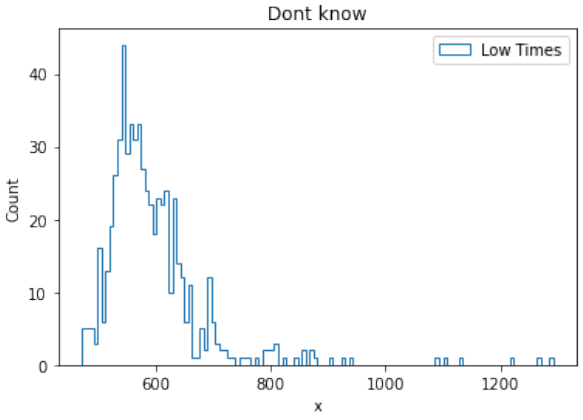
(a) Data Histogram



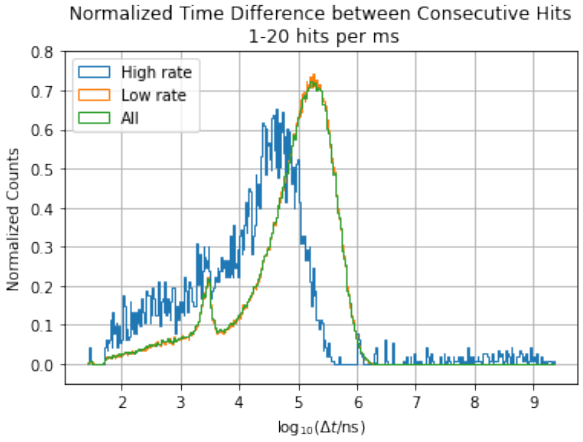
(b) Time Difference

Figure 15: Low threshold partitioned data - **Few Bins**

The high rate and all rates match up very precisely indicating very little data is in low rate compared to high rate, and this time you can see how the Low rate deviates in time differences over 6 seconds. After 16 hours I ran the algorithm binning per 1 ms and it gave either vastly different or incorrect results as shown below.



(a) Data Histogram

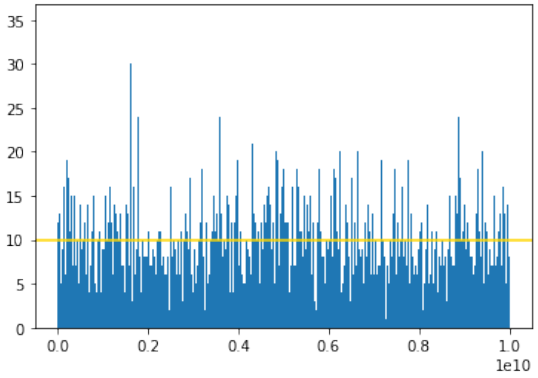


(b) Time Difference

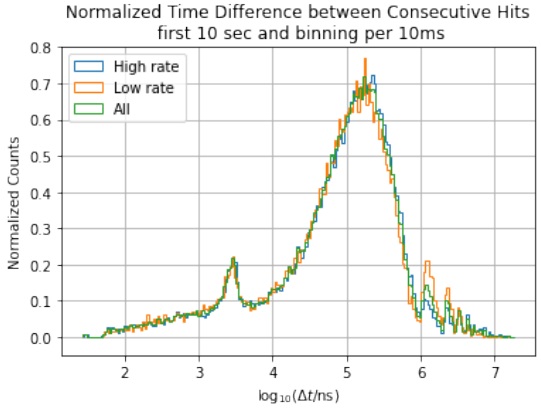
Figure 16: Low threshold partitioned data - **1 ms Bins** and **10 hit/bin** cutoff

Most of the differences between fig 15.b and fig 16.b may come from the fact that I accidentally set low rate as 1-10 hits per bin for fig 15 and made it the correct 1-20 hits for fig 16. For this reason fig 15 may not have much significance.

I ran another copy of the notebook to re-create the larger bin file. It originally ran with larger bins as well as only a section of the data. I ran it again with large bins for 10 second sections from 0-10 sec and 3-4 sec. They both showed the saw-tooth pattern indicating it persists through the file. Either due to a smaller sample size or larger bins. **Correction: bin per 1.666 ms.**

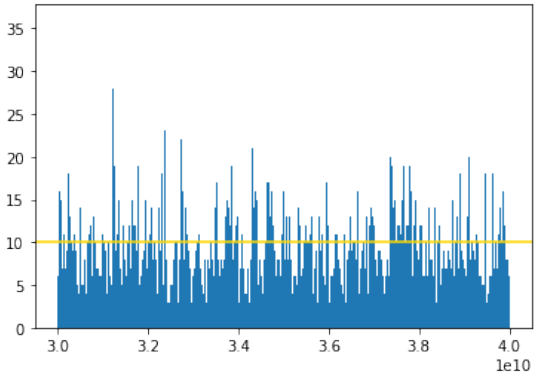


(a) Bin Histogram with 10 hits/bin cut off

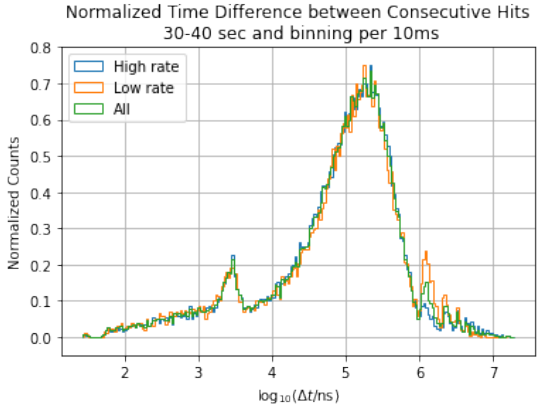


(b) Time Difference

Figure 17: Partitioned **first 10 seconds** *binning per 10 ms*



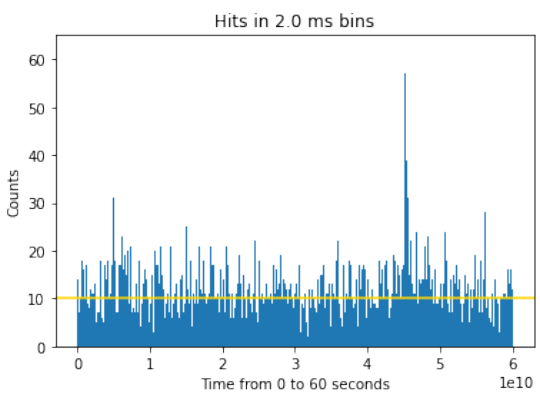
(a) Bin Histogram with 10 hits/bin cut off



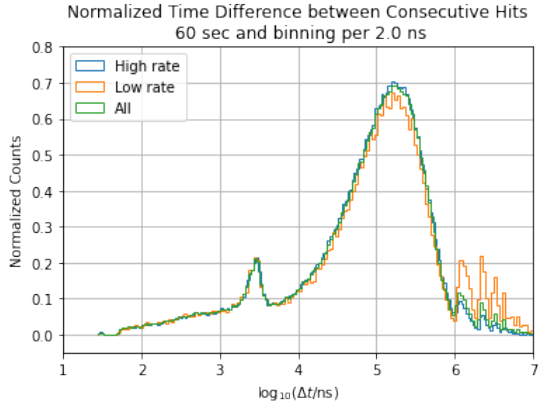
(b) Time Difference

Figure 18: Partitioned **fourth 10 seconds** *binning per 10 ms*

I ran it again with the entire 60 seconds, bins per 2ms and partitioned at 10 hits per bin giving the same over all shape in *fig 19* and the again with the cut off at 20 hits per bin in *fig 20*. You can see how the saw-tooth pattern has to do more within the noise.

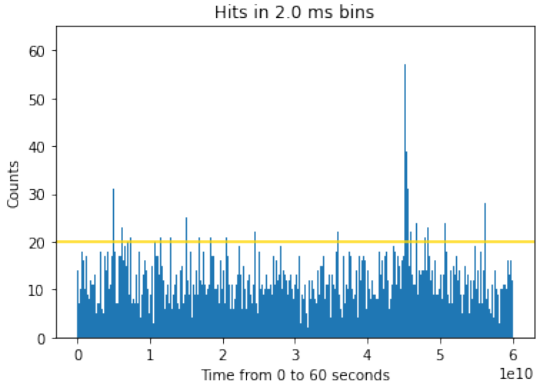


(a) Bin Histogram with 10 hits/bin cut off

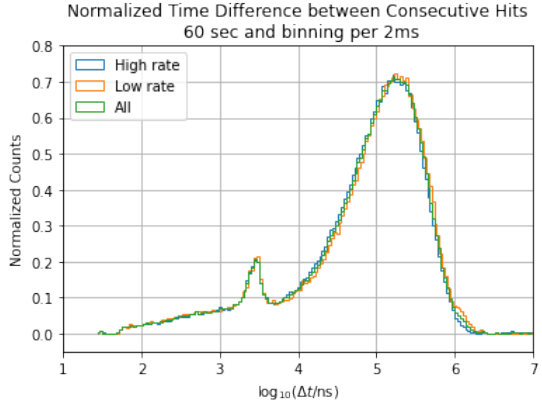


(b) Time Difference

Figure 19: Partitioned **full 60 sec** *binning per 2 ms* - **10 hit** partition



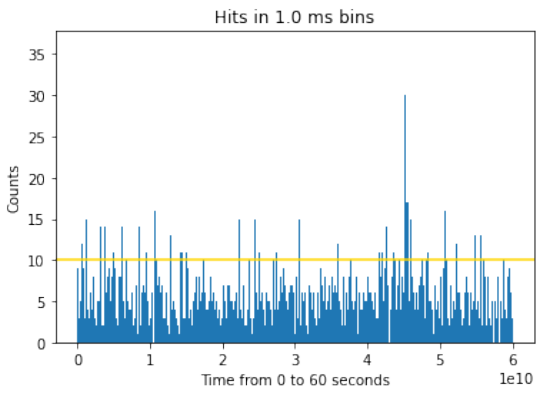
(a) Bin Histogram with 20 hits/bin cut off



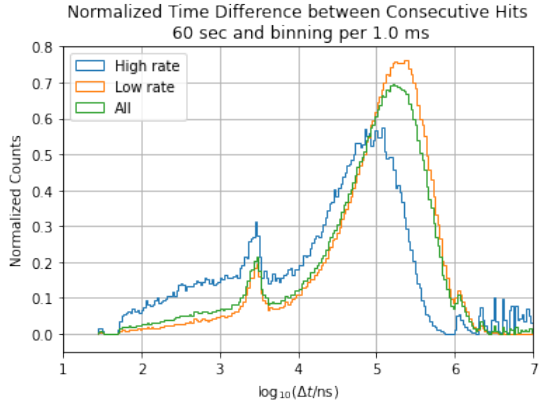
(b) Time Difference

Figure 20: Partitioned **full 60 sec** *binning per 2 ms* - **20 hit** partition

Now we have 1 ms bins with a 10 hit cutoff over the fill one minute, followed by 1 ms bins with a 20 hit cut off over the full one minute.

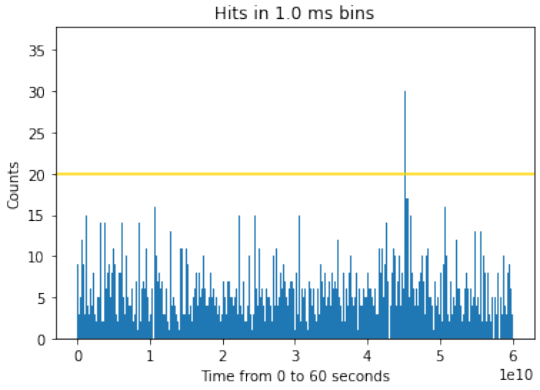


(a) Bin Histogram with 10 hits/bin cut off

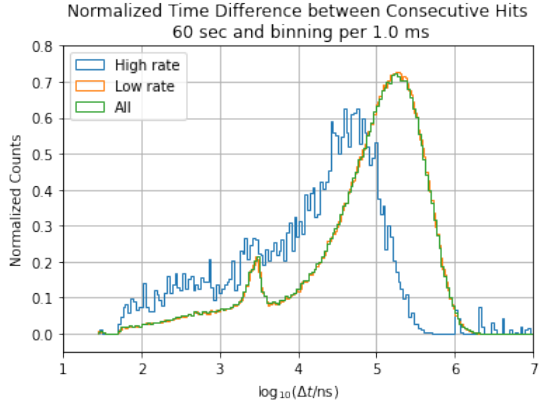


(b) Time Difference

Figure 21: Partitioned **full 60 sec** *binning per 1 ms* - **10 hit** partition



(a) Bin Histogram with 20 hits/bin cut off



(b) Time Difference

Figure 22: Partitioned **full 60 sec** *binning per 1 ms* - **20 hit** partition

Until I can confirm that I am on the right path I will start running the code that made *fig 22* on each threshold to see if it leads to anything.

3 Questions

1. Why do we see the saw-tooth shape in Low rate after 6 seconds in Figure 15b?
2. Sort out if fir 15.a means something.
3. My current code is a set of bodged functions to sort the data based on the amount of hits in each bin and is pretty much the opposite of efficient.
   (a) Is my code doing what its meant to be doing?
      - Compare with working results.
      - Compare with working code.
   (b) Fix some of the unnecessarily memory intensive areas:
      - Squish down lowTimeRanges to turn this: '[0,1],[1,2],[4,5],[5,6]' into this: '[1,2],[4,6]'.
      - Once the code is working, break file into 15 second chunks so I can run it on multiple cores at once (will cut times down by 10x but heat up my computer by 5x).