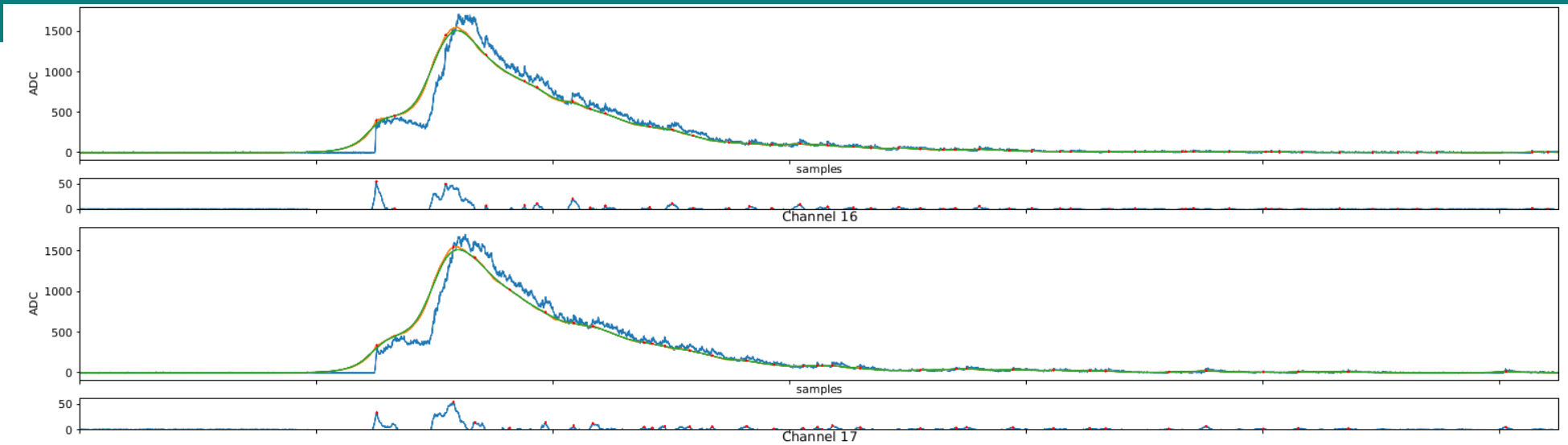
Pulse finder algorithm

Emmanuel Le Guirriec

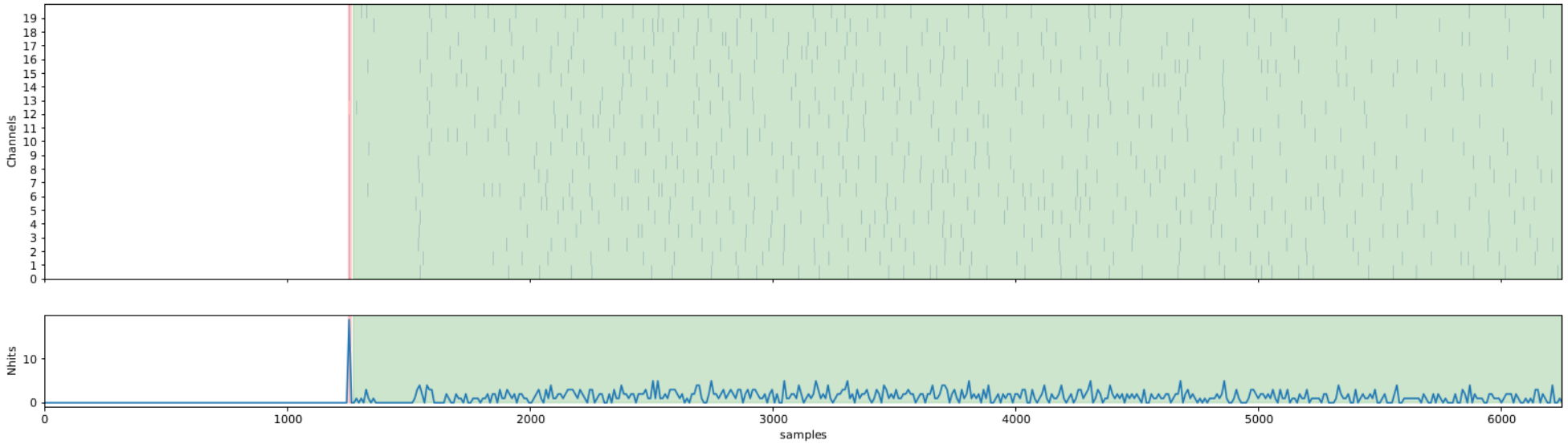
DarkSide CPPM Meeting

20 May 2021

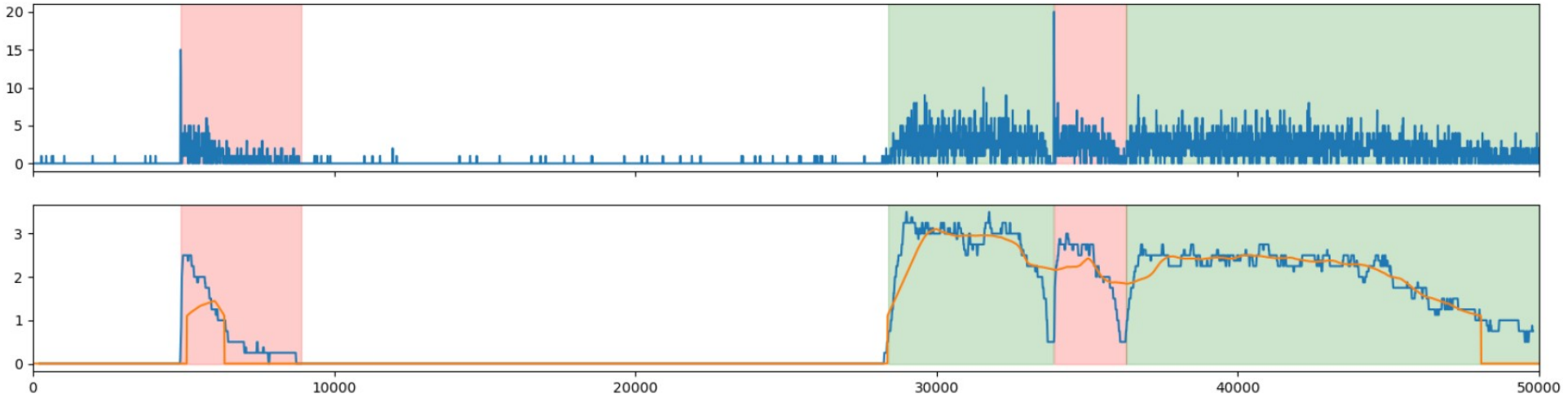
Histogram form hits



- 2D histogram
- Create hits histogram (width parameter for bining)



Julie´s code



- Many steps in pulse_finder code
- Many not documented cases to characterize S1 and S2

Find S1 candidate

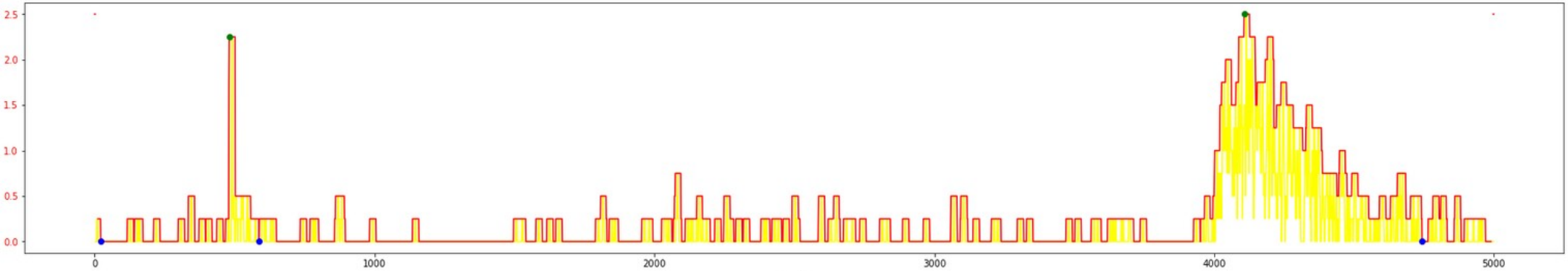
- **Find all the bins where there is more then s1_min hits (7 with pyred)**
  - Should be a proportional parameter of the number of channels
- **Compute the number of hits over X neighboring bins around the S1 candidate**
  - X is hard coded (10 in pyred test)
- **If the number of hits in the S1 bin is higher than the sum of hits in neighboring bins of the hits, it is a S1 candidate**

Prepare signal for mountains detection

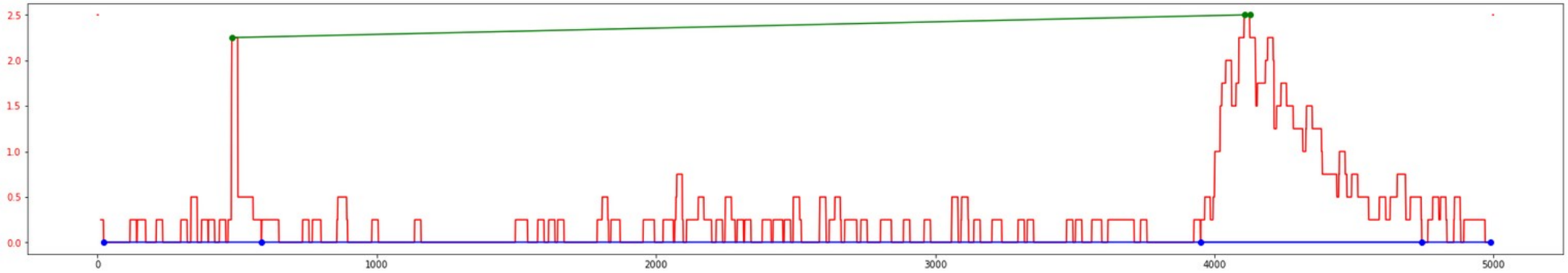
- Remove very small isolated peaks of the hits histogram
- A moving median average is applied to the resulted signal
  - rolling parameter is used fro windows
- Package peakdetect to find the local maxima and minima in some noisy signal

Compute peaks and valleys with peakdetect

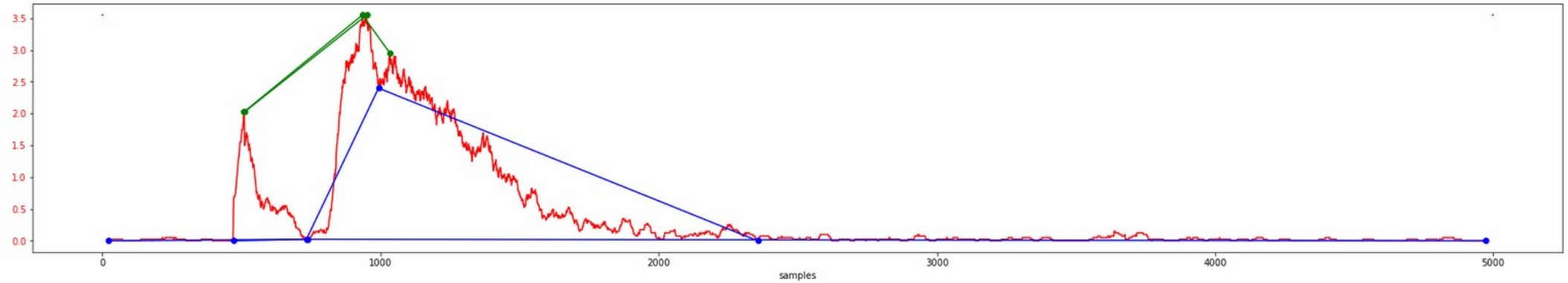
- at the beginning and end of the signal a fake peak is added to force valleys finding (not plotted)
  - pre_gate parameter is used



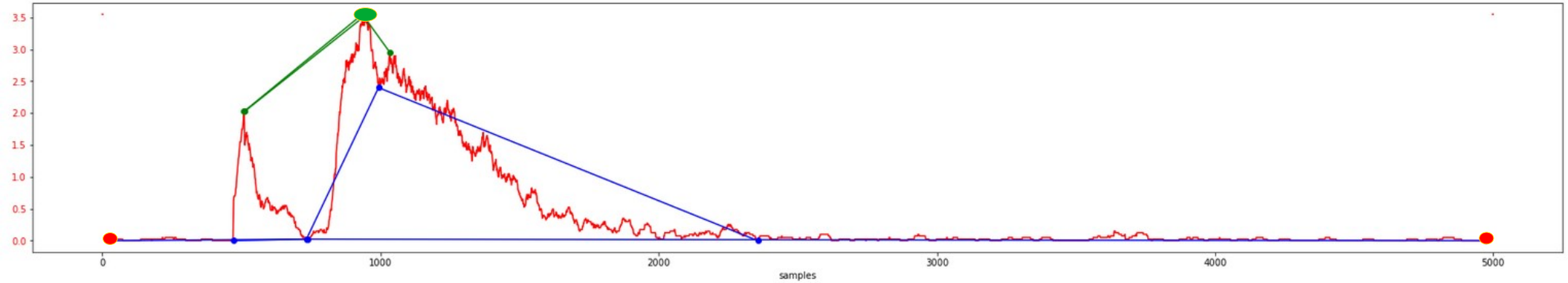
- the signal is time reversed and same peakdetect is applied.



Define mountains

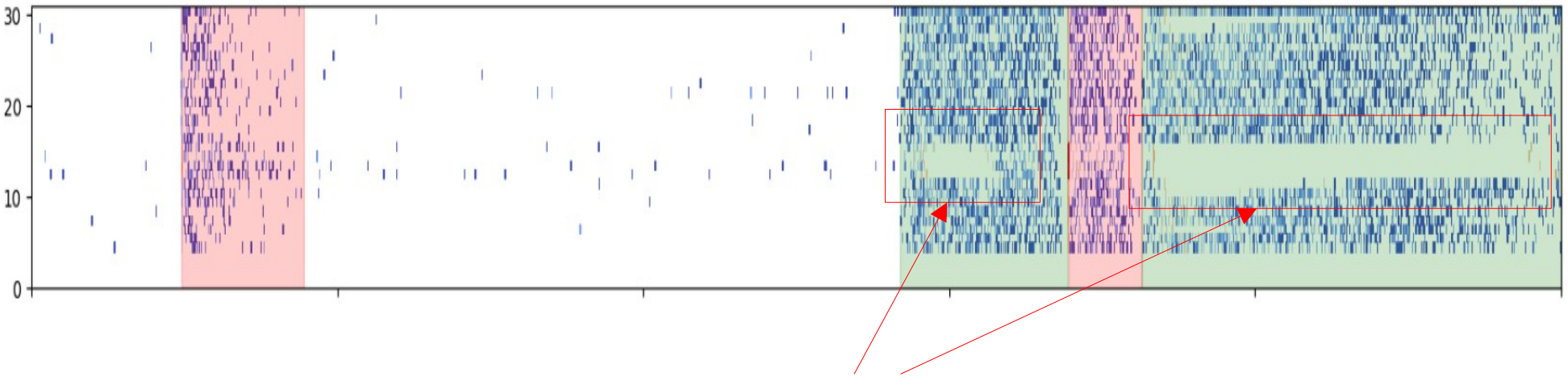


- With this two sets of peaks and valleys we can define mountains: one peak with the closest base.



- We obtain 3 mountains

Saturated area



- Find large area where no peaks has been found
- Here we use the 2D histogram (per channel)
- Remove some peaks and valleys if long saturated area in many channels
  - saturated_window = self.s1_window / 2 # Maybe a dedicated var needed
  - Number of channels = 3
    - Should be a proportional parameter of the number of channels
- Saturated area: sign of S2?

Characterize the peaks

- **Loop on mountains to characterize S1 and S2**
  - Check that S1 candidate is in mountain
  - Compute the fraction prompt for S1
    - fp_s1 = (number of hits[ind_peak:ind_peak+2])/(total number of hits)
  - Find peak of the mountain→ Good S1 if
    - fp_s1 > 0.013 (hard coded (from pyred Julie))
  - Other peaks are S2 candidates
    - fp_s2 = (number of hits within [ind_peak:ind_peak+10])/(total number of hits)
    - remove S2 if fp_s2 > 0.2 (hard coded)
    - remove S2 if gate < 100 (hard coded)

Merge S2 and resize S1 S2

- **Step to merge close S2s**
  - Here also hard coded parameter to check if S2 are close
  - If previous S2 is ending less than 200 bins before current S2 is starting and 5 times more hits in current S2 than previous S2
  - If previous S2 is ending less than 3 bins before current S2 is starting
- **Step to resize S1 and S2**
  - Use s1_window and s2_window (from pyred)

Main method

```
def get_clusters(self, table):

    hits_perch = table.reshape((-1,self.width,table.shape[1] ))
    hits_perch = np.sum(hits_perch,  axis=1)
    hits = np.sum(hits_perch,  axis=1)

    #Find S1 candidates
    S1 = self.S1Candidate(hits)

    #Remove very small isolated peaks
    ma = np.convolve(hits, np.ones(self.rolling), 'same')
    sip = ((hits== 1) & (ma== 1)).nonzero()[0]
    hits[sip] = 0

    #Find peaks and valleys
    y = pd.Series(hits).rolling(self.rolling, center=True).mean()
    (Peaks, Valleys) = self.DetectPeaksAndValleys(y)

    #Find if too many channels saturated during an identical long period
    #More than 3 channels by default
    #Period is 200 by default
    Sat_Area = self.FindSaturatedSignal(hits_perch)

    #Join peaks and valleys when a saturated area is found
    self.JoinPeaksAndValleys(Peaks, Valleys, Sat_Area)

    #Characterise the peaks
    pulses = self.CharacterizeS1S2(hits, Peaks, Valleys, S1)

    #S2 merging step
    self.MergeS2(pulses, hits, prox = 3)
    pulses = self.remove(pulses)

    if len(pulses) > 0:
        nbins = hits_perch.shape[0]
        self.ResizeS1S2(pulses, hits, nbins)

        pulses['start'] = pulses['start']*self.width
        pulses['stop']  = pulses['stop']*self.width
        pulses['gate']  = pulses['gate']*self.width

    clusters = self.sort(pulses[['start','stop','type','nhits']])
```

Backup slides

Parameters in config.ini

- [pulse_finding]
  - width = 80 #(ns)
  - s1_min = 7
  - rolling = 3200 #(ns)
  - s1_window = 32000 #(ns)
  - s2_window = 120000 #(ns)
  - pre_gate = 240 #(ns)

Parameters not in config.ini

- Number of bins to calculate the number of hits around the the S1 candidate: S1_neighbor_bins (5 on each side with pyred tests)
- Number of channels with long saturated period: nb_sat_channels (3 with pyred tests)
- Number of bins to compute S1 fp: s1_bins_fp
- Minimal fraction prompt of S1: min_fp_s1
- Number of bins to compute S2 fp: s2_bins_fp
- Maximal fraction prompt of S2: max_fp_s2
- Minimal S2 gate (100): min_S2_gate
- Distance between end of S2 and next one combined with the ratio of number of hits: s2_dist_hits_ratio(200, 5)
- Distance between end of S2 and next one: s2_dist (3)