Thesis draft:
Online processing of the large area SiPM detector signals for the DarkSide20k experiment

Giacomo Petrillo
University of Pisa

April 21, 2021

Contents

Introduction

Explain the chapter structure and how to reach the code for each figure.

Chapter 1

Dark matter

Random notes:

- Say why the dark matter can not form sort of a parallel universe with planets systems life etc. (no autointeraction).
- Find a reference for the dark matter not autointeracting, some convincing reasoning, the fact that it can not even form black holes by random fluctuation.

  The main point I think is the fact that the dark matter halo has the shape of a sphere (how is this determined?), it would be a disk if it had friction.
- I consider it a great victory for physicists that most of the matter in the universe is like an ideal fluid from a physics textbook.

Chapter 2

The DarkSide experiment

Chronology of DarkSide, explanation of why the detectors are done in such and such way (look at Stracka's slides).

Chapter 3

Data

Explain the Proto0 and LNGS datasets. For Proto0 look at Luzzi's thesis. For LNGS I should probably link some slides and explain what I know.

Put 2D histograms of all datasets I used, apart from when there are multiple files with the same configuration. Plot an histogram of all the LNGS triggers together to reassure the reader that I can assume the trigger position for files without trigger.

Chapter 4

Signal to noise ratio of filtered photodetection signals

The DarkSide20k detectors will use photodetector modules (PDMs) made up from many silicon photomultipliers (SiPMs). For what concerns us, the output is similar to a single SiPM output. When a photon hits the photomultiplier, the electrical output is a sudden current spike, with a rise time on the order of nanoseconds, which decays slowly in some microseconds.

Each SiPM is made up of many single photodiodes (called "cells"), so when different photons hit simultaneously different cells, the output is a signal with an amplitude proportional to the number of photons. See figure 4.1 for an example.

Bibliography for the SiPM.

Our goal is to study the performance of some filters in finding and measuring the amplitude of the signals amidst electrical noise. We'll now introduce the dataset, list the filters tested, define a performance measure and show the results. Finally we'll compute and comment the noise spectrum. The code for this work is online at `https://bitbucket.org/Gattocrucco/sipmfilter`.

4.1 Data

For this study we used test data taken at liquid nitrogen temperature from a setup in the Gran Sasso National Laboratories (LNGS). A laser pulse is shot at regular time intervals on the PDM. Both the laser trigger and the detector output are sampled at 1 GSa/s with a 10 bit ADC and saved separating the data in "events" where each event correspond to a single laser pulse. See figure 4.2.

Bibliography for the LNGS data. Maybe add a 2D histogram of the data.

We used the PDM slot 8 data, which as per figure 4.3 corresponds to tile 57 which means the data is in the directory `http://ds50tb.lngs.infn.it:2180/SiPM/Tiles/FBK/NUV/MB2-LF-3x/NUV-LF_3x_57/`. We used the file `nuvhd_lf_3x_tile57_77K_64V_6VoV_1.wav`.

Slot 8 probably is just a Proto0 name, should mention it later. And what are these "tiles"?

In the dataset there are a couple of problems. The first is that signals with many photoelectrons saturate, however this won't trouble us since we'll need only single photoelectron signals. The second is the presence of some spurious signals which do not correspond to the laser pulse. I filtered these out by putting a threshold in the part of each event *before* the laser trigger, which should be flat apart from the noise; there were 72 of them out of 10 005 events.

In principle a spurious signal arriving *after* the "official" laser-induced signal matters too, however I'm ignoring them out of this logic: spurious signals hitting earlier raise the official signal in a somewhat uniform way with their slowly decaying tail, so the detected amplitude will have a bias which is significant, but possibly small and as such not identifiable. Spurious signals hitting later will add a large spike in the tail of the official signal, so the amplitude will be noticeably higher, such that a single photoelectron pulse gets confused as a double or higher one, and we'll automatically ignore it as we'll consider signals detected as single.

This reasoning may fail depending on the details of filtering and the specific relative timing of the signals, however the final most important consideration is that I expect less than 100 spurious late pulses since there are 72 early ones and the laser pulse is in the middle of the event, so less than 1 %. See figure 4.4 for some examples of spurious/saturated signals.

The afterpulse stuff I had not understood properly may change all this.

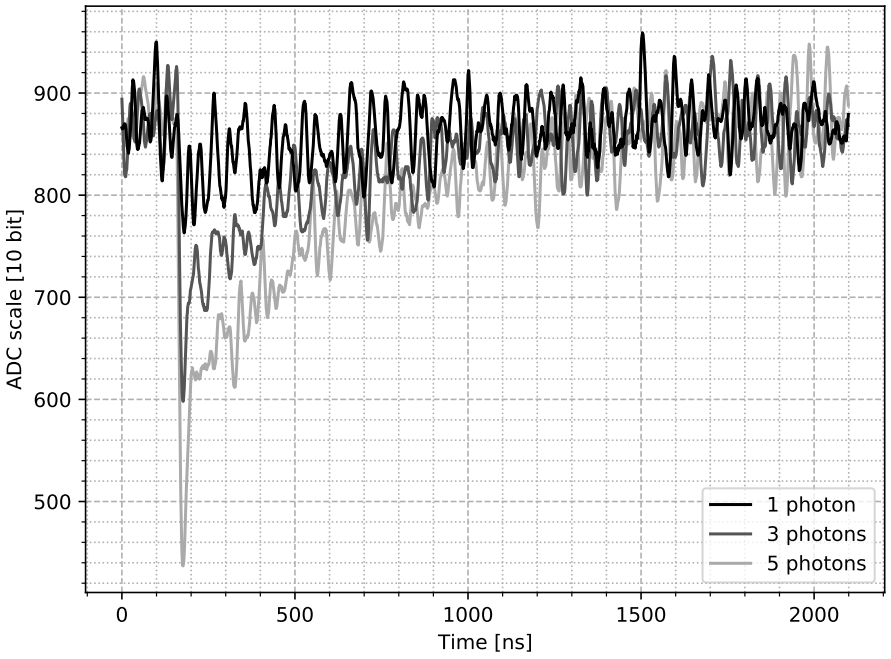

Figure 4.1: Example PDM signals taken from the LNGS test data (section 4.1). The different curves correspond to an increasing number of photons being detected simultaneously, with the resulting signal being the sum of single-photon signals. Notice that the noise amplitude is the same in all curves, which means that it is produced mostly outside of the PDM. (`figsignals.py`)

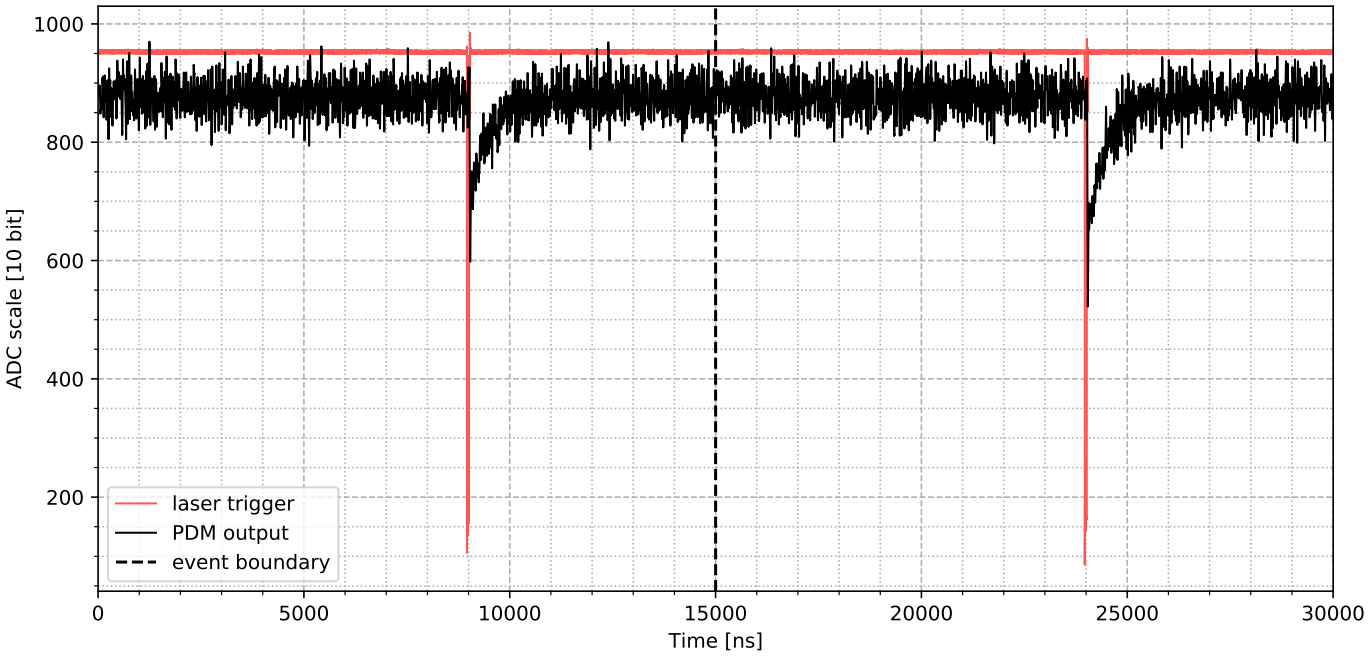

Figure 4.2: A pair of events from the LNGS test data (section 4.1). (`figlngs.py`)

PDM 1		PDM 2		PDM 3		PDM 4		PDM 5	
	31		32		39		64		55
126	57	132	44	136	42	142	52	149	53
0	0	0	2	0	4	0	6	0	8
PDM 6		PDM 7		PDM 8		PDM 9		PDM 10	
	30		59		57		37		29
127	33	133	41	138	37	144	31	150	43
0	10	0	12	1	0	1	2	1	4
PDM 11		PDM 12		PDM 13		PDM 14		PDM 15	
	38		36		58		62		60
129	46	134	48	139	60	145	59	151	50
1	6	1	8	1	10	2	0	2	2
PDM 16		PDM 17		PDM 18		PDM 19		PDM 20	
	41		61		66		63		52
130	47	135	32	140	40	146	56	152	35
2	4	2	6	2	8	2	10	3	0
PDM 21		PDM 22		PDM 23		PDM 24		PDM 25	
	34		53		54		65		42
131	38	137	58	141	34	148	51	153	45
3	2	3	4	3	6	3	8	3	10

PDM slot	
	tile (run2)
PDM	feb
V1725[0..3]	Ch[0..15] (top)

Figure 4.3: PDM matrix of the Proto0 detector.

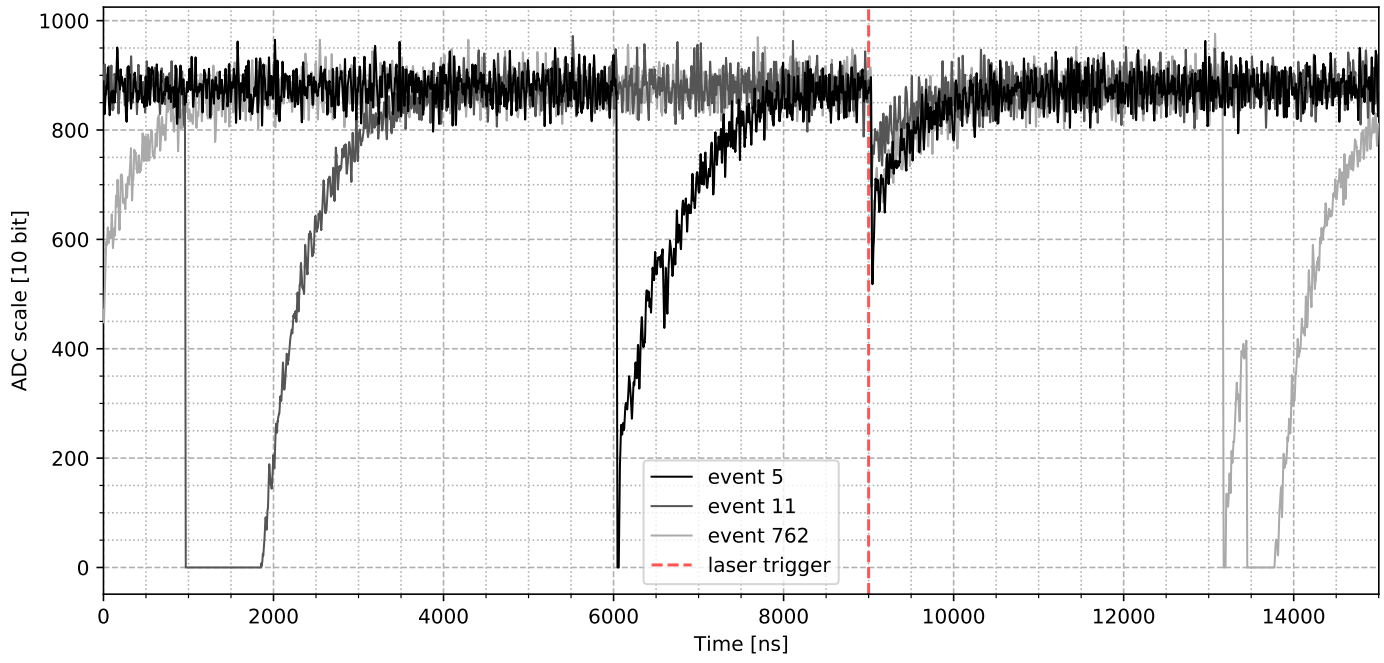

Figure 4.4: Examples of spurious signals in the LNGS test data (section 4.1). (`figspurious.py`)

4.2 Filters

A filter operates by converting the original sequence of ADC samples $(x_1, x_2, \ldots)$ to a new "filtered" sequence $(y_1, y_2, \ldots)$. The filters are causal, i.e. the filtered sample y_n can be computed only using the original samples up to x_n. This limitation is because we are interested in using the filters online, i.e. produce the filter output continuously as samples are read.

We tested three filters: the moving average, the exponential moving average or autoregressive filter, and the cross-correlation filter.

The moving average consists in taking the average of the last N samples:

$$y_n = \frac{1}{N} \sum_{i=1}^{N} x_{n-N+i}. \tag{4.1}$$

The exponential moving average weighs past samples with an exponentially decaying coefficient, and can be written recursively as

$$y_n = a y_{n-1} + (1-a) x_n, \quad a \in (0,1). \tag{4.2}$$

The scale of the exponential decay is given by

$$\tau = -\frac{1}{\log a}, \tag{4.3}$$

$$\approx \frac{1}{1-a} \text{ for } a \text{ close to 1.} \tag{4.4}$$

The cross-correlation filter is the most sophisticated we considered. Let $\mathbf{h} = (h_1, h_2, \ldots, h_N)$ be a *template* of the signal waveform we want to detect. This means $\mathbf{h}$ should ideally match the shape of the signal waveform we want to find in the noisy data. The filter is then the cross-correlation of $\mathbf{x}$ with $\mathbf{h}$:

$$y_n = \sum_{i=1}^{N} h_i x_{n-N+i}. \tag{4.5}$$

Under the assumption that the data is white noise plus a signal that perfectly matches the template apart from amplitude, this filter is optimal in the sense that in the filter output there will be a peak corresponding to the signal and this peak will have the maximum possible height relative to the standard deviation of the filtered noise.

The differences we have from the ideal case are:

1. the shape of the signal probably changes a bit each time;
2. the signal is not aligned always in the same way to the ADC timebase;
3. the noise is not white.

The variation of the actual signal shape is difficult to address directly. The noise spectrum can be corrected by appropriately transforming $\mathbf{h}$, in this case the filter is called *matched filter*. Let $\mathbf{s}$ and $\mathbf{w}$ be the signal and noise, such that the waveform to be filtered is $\mathbf{x} = \mathbf{s} + \mathbf{w}$. Let R be the noise covariance matrix, i.e. $R_{ij} = \mathrm{Cov}[w_i, w_j]$. Then the template for the matched filter is

$$\mathbf{h}_\mathrm{m} = R^{-1}\mathbf{h}. \tag{4.6}$$

Say that the matched filter is equivalent to rank 1 least squares.

We tried implementing the matched filter with mixed results we will not report in detail. We computed the noise covariance matrix on the event samples before the signal. Since the noise is stationary, R is a Toeplitz matrix, i.e. the covariance depends only on the lag between two samples. Figure 4.5 shows the noise covariance obtained.

Add the autocorrelation of the Proto0 noise to the plot.

The matched filter worked slightly better than the cross-correlation filter, as expected, but only for N sufficiently small, getting worse as N increased. This could be due to numerical accuracy problems in solving the linear system R in equation 4.6, or in computing R. We didn't work on this further since the gain is probably small.

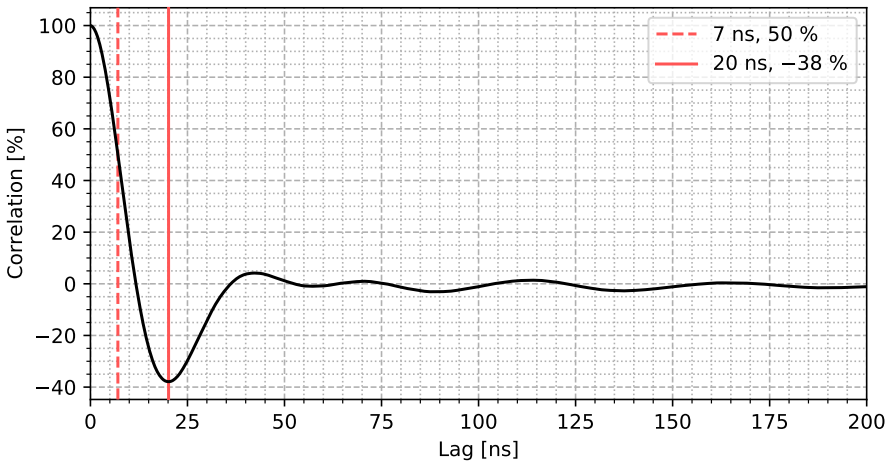

Figure 4.5: Autocorrelation of the LNGS test data noise, computed on the part of the events before the trigger. The autocorrelation at lag t is the correlation between a sample and another sample occurring a time t after the first. The standard deviation of the noise is 26.7 ADC units. (`figautocorrlngs.py`)

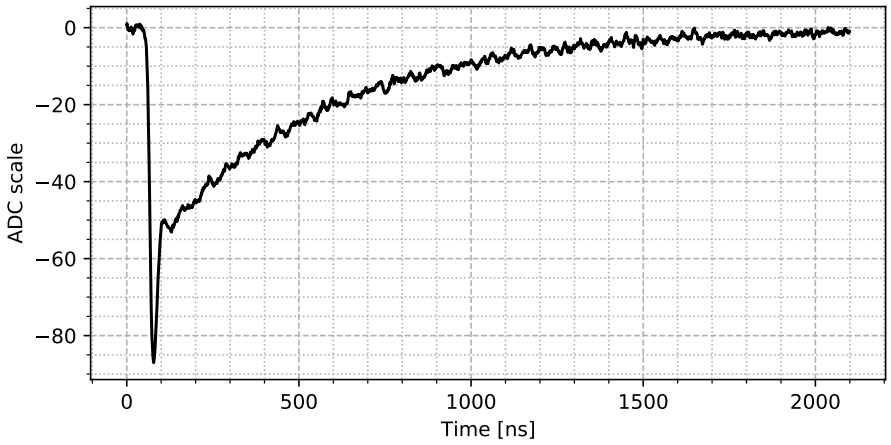

Figure 4.6: The template used for the cross-correlation filter. It is the median of unaligned single photon events from the LNGS test data. (`figtemplate.py`)

We computed the template for the cross-correlation filter by taking the median of single photoelectron signals. We did not align the signals, we operated as if they occurred always with the same alignment relative to the event time window. The template obtained is shown in figure 4.6. When using the template, we normalize it to unit sum such that the output from the cross-correlation filter is comparable to the output from the moving averages, i.e. if we send a flat waveform into the filter, the output has the same value as the input.

Since we will try various lengths of the filter template, we have to decide how to truncate the full template. When truncating to N samples, we pick N contiguous samples from the template such that their sum is the minimum possible (recall the template is negative). Of course normalizing the template to unit sum is done after truncation.

See figure 4.7 for an example of filter output.

Use the template from the second chapter; plot the template without alignment, with trigger, with filtering; explain that we use 1 pe pulses to avoid afterpulses; cite Luzzi to say it should not matter much how we compute the template in detail.

Explain that we fix the mean-like behaviour for all filters because we have to subtract the baseline.

4.3 The fingerplot

To measure the performance of filters, we define a signal to noise ratio (SNR) as follows: the SNR is the ratio of the average peak filter output value for single photon signals over the standard deviation of the filtered noise.

We could consider other similar measures, for example we could include the

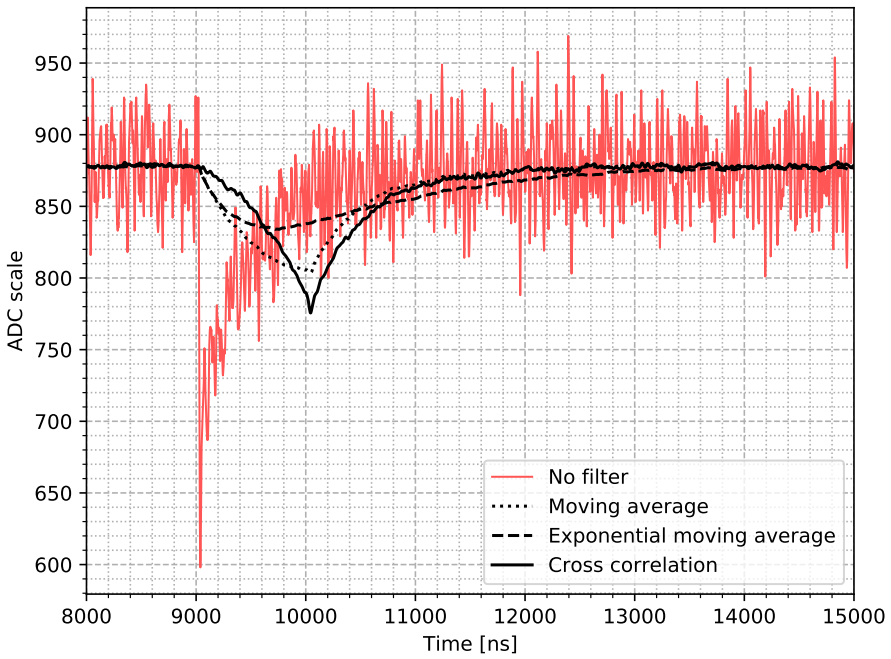

Figure 4.7: An event from the LNGS test data filtered with the three filters used. The number of averaged samples in the moving average filter, the scale of the autoregressive filter, and the length of the template of the cross-correlation filter are all 1024 samples (at 1 GSa/s). (`figfilters.py`)

standard deviation of the peak filter output value since that influences where we should place a threshold to discriminate signals, however it is sufficient to use any reasonable definition for the purpose of comparing different filters. Effectively we computed the SNR without exactly respecting the definition above, we'll see in detail.

For each event we compute the filter output at a fixed temporal delay from the leading edge of the laser trigger pulse (see figure 4.8). Then we compute the average of the samples before the trigger pulse and take that value as "baseline". We subtract the baseline from the filter output, and finally we change the sign to obtain positive values since the signals are negative.

We take the list of these baseline and sign corrected filter outputs and compute an histogram. One of these histograms is shown in figure 4.9. It is called "fingerplot" due to the descending peaks reminding of fingers.

The first peak is centered on zero and thus corresponds to cases where the laser pulse produces no photoelectrons. Since the instant where we are evaluating the filter output in each event is independent of the output itself (instead of e.g. being determined by peak finding), this peak is the distribution of the noise after passing through the filter.

The various other peaks correspond to an increasing number of photons. The second peak is the distribution of the filter output at a fixed instant for single photon signals. Assuming for the moment that the instant where we evaluate the filter yields the highest signal response, this means that the SNR is the mean of the second peak divided by the standard deviation of the first.

Since the peaks are often overlapping, and that we will have to repeat the calculations for many fingerplots automatically without checking each one, we use robust measures of location and scale instead of the average and standard deviation. We run a peak finding algorithm on the histogram and divide the data by putting boundaries midway between peaks, so that we assign each datapoint to a peak. For the second peak, we take the median instead of the average. For the first peak, we take the half symmetrized 68 % interquantile range instead of the standard deviation, i.e. half the difference between the 0.84 and 0.16 quantiles. On a gaussian distribution these are equivalent to the mean and standard deviation, however they are less sensible to messing up the tails of the

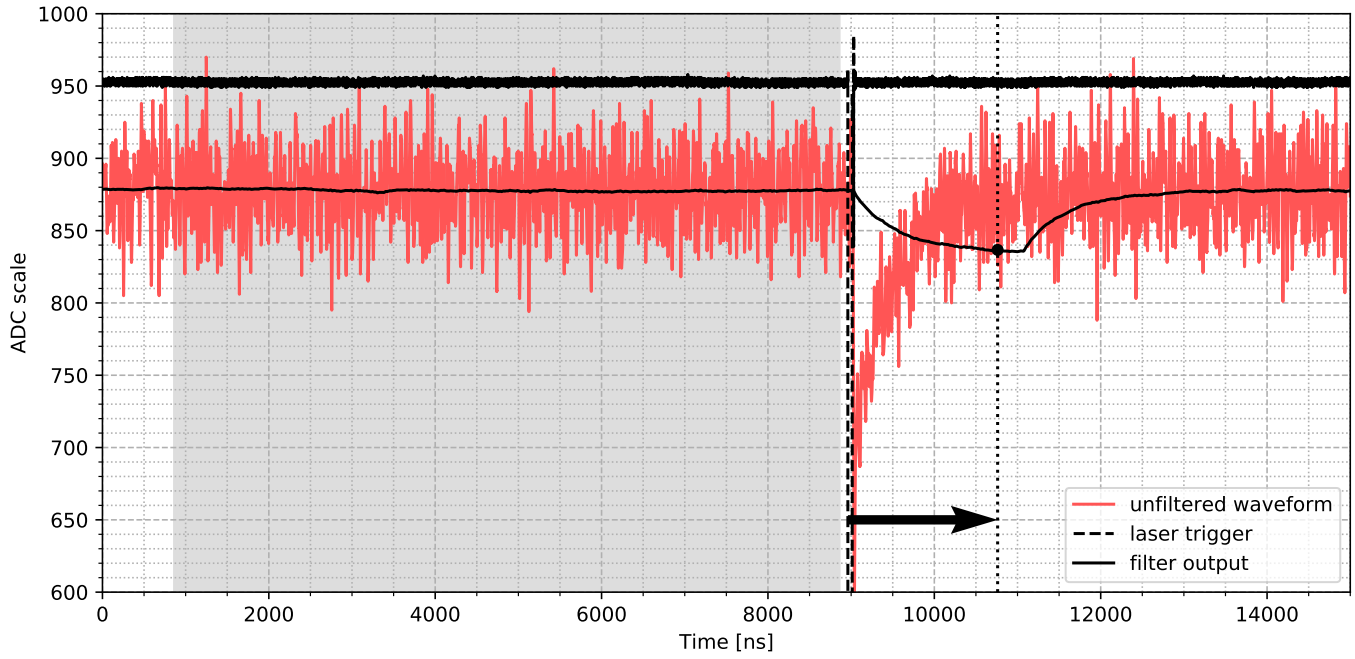

Figure 4.8: Example of how a signal amplitude is computed in an event for the purpose of computing the SNR. The samples in the shaded region are averaged to compute the baseline. The filter is evaluated at a fixed offset (indicated by the arrow) from the leading trigger edge. The amplitude then is the difference between the baseline and the filter value. The shaded region ends 100 samples before the trigger. (`figfiltersample.py`)

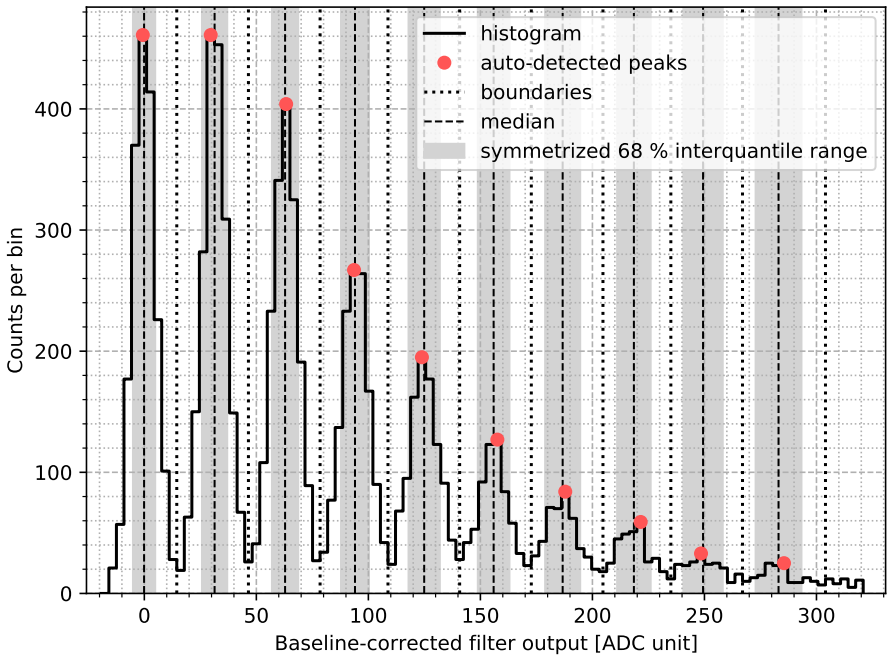

Figure 4.9: The fingerplot for the moving average filter with 128 samples evaluated 128 samples after the trigger. (`figfingerplot.py`)

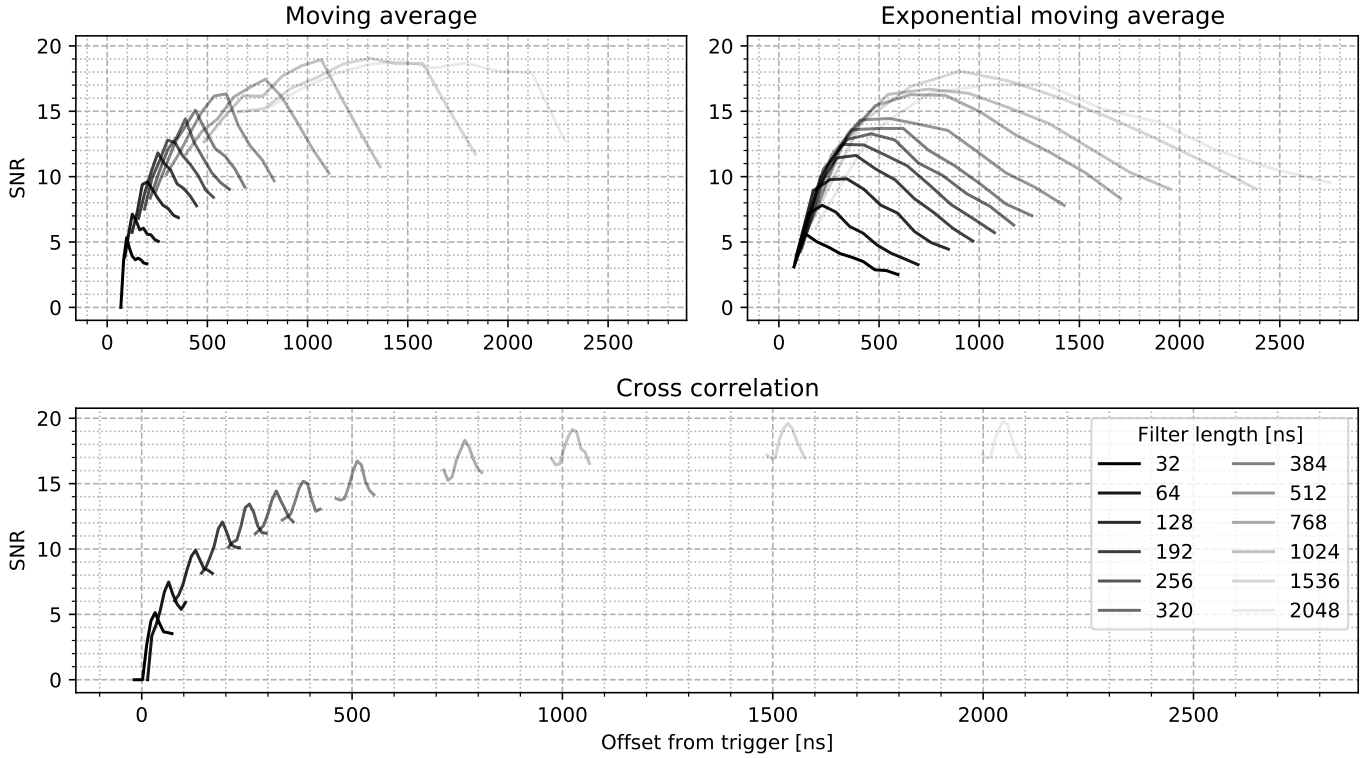

Figure 4.10: The SNR as a function of delay from trigger (x-axis) and filter length (shade of gray) for the three filters. (`figsnrplot.py`)

distribution, which happens since the boundaries cuts away the tails and there is contamination from the tails of the neighbouring peaks.

Mention that I'm always checking that the median of the 0 pe peak is zero within its error.

4.4 SNR versus filter length

When determining how to compute the SNR from the fingerplot we assumed that we were evaluating the filter output at the optimal instant. Since we do not know it a priori, we repeat the computation for a range of values of the filter evaluation point. A simpler solution that comes to mind is taking the minimum (the signals are negative) of the filter output in each event, however this would bias upward the SNR measure because the minimum can yield lower values due to noise peaks. Instead, by fixing the point independently from the data, the random oscillation due to noise is symmetrical and the averaging recovers the actual amplitude of the filter output for the signal.

The resulting SNR curves are shown in figure 4.10, repeated for a range of values of the filters length parameter. For the moving average and cross-correlation filter, the length parameter is the number of samples, N. For the exponential moving average, it is the scale of the exponential decay τ.

The maximum of each curve gives the actual SNR figure we are interested in. We expect by intuition that the width of the maximum is approximately proportional to the temporal resolution we could achieve if we used the filter output to locate temporally the signal. So we do another plot (figure 4.11) where we show the maximum SNR value and the width of the peak versus the filter length parameter. We measure the width as the distance between the two points where the SNR is 1 less than its maximum value.

I can drop the snrmaxplot because it's already in the baseline effect plot.

4.5 Effect of the baseline computation

In constructing the fingerplot, we subtract from the filter output value the baseline, i.e. the average value of the waveform in absence of signals. We compute the baseline for each event as the average of the samples before the signal. More

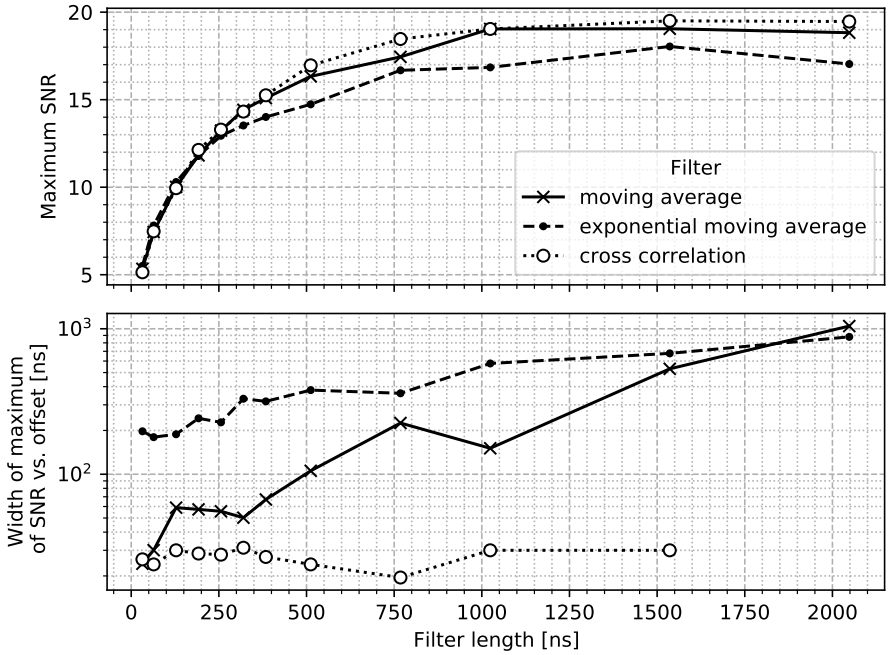

Figure 4.11: Top plot: maximum SNR achieved with each filter (different curves) and filter length (x-axis). Bottom plot: the width of the maxima, computed as the length of the interval of offset from trigger with endpoints where the SNR is 1 less than the maximum. The cross-correlation filter mantains almost constant width with increasing SNR, while the other two filters have increasing width. (`figsnrmaxplot.py`)

precisely, we averaged 8000 samples. The value obtained varies randomly due to the noise; its standard deviation is that of the noise divided by $\sqrt{8000}$. The maximum filter length we use is 2000, so in that case the width of the peaks gets an additional contribution (summed in quadrature) which is $\sqrt{2000/8000} = 1/2$ of the width we would have with a noiseless baseline, i.e. the width of the first peak is $\sqrt{1 + 1/2} - 1 = 22\,\%$ larger, lowering the SNR by the same percentual.

This is not just a problem of this test that can be worked around in the real application, because how the baseline is computed is a relevant part of the signal finding, since it requires either a fast on-digitizer algorithm estimating it reliably or sending enough samples to subsequent stages in the data processing chain.

To get an idea of the effect of the baseline, we repeat everything computing the baseline with just 1000 or 200 samples. The result is in figure 4.12. We see that, for example, going from 8000 to 200 baseline samples the maximum SNR drops from about 20 to about 9.

Explain why the SNR of the cross correlation filter drops with increasing filter length when the baseline is short. The filter behaves like a mean, so its peak value drops as it gets longer; the noise rms decreases too but gets a constant contribution from the baseline variance.

4.6 Noise spectrum

We take the region of each event before the trigger pulse, compute its periodogram, and take the median across events for each frequency. We use the median instead of the average in case there were spurious signals or irregularities we missed. We do the same for data obtained with the same sensor but when it was in the Proto0 setup, biased below breakdown voltage and sampled at 125 MSa/s. We thus obtain two plausible noise spectra, shown in figure 4.13.

Source and details of the Proto0 noise data.

We said the cross-correlation filter is optimal if the noise is white. Actually, it is sufficient that the noise spectrum is flat in the support of the signal spectrum, because if we filtered away frequencies outside of it, all the signal would still be in the waveform. In figure 4.14 we show the power spectrum of the signal and its integral. We see that 90 % of it is below XXX MHz, which corresponds to an approximately flat region in the noise spectra. This means that the cross-correlation filter is close to optimal.

False: most of the signal spectrum is in the rapidly ascending part of the noise spectra.

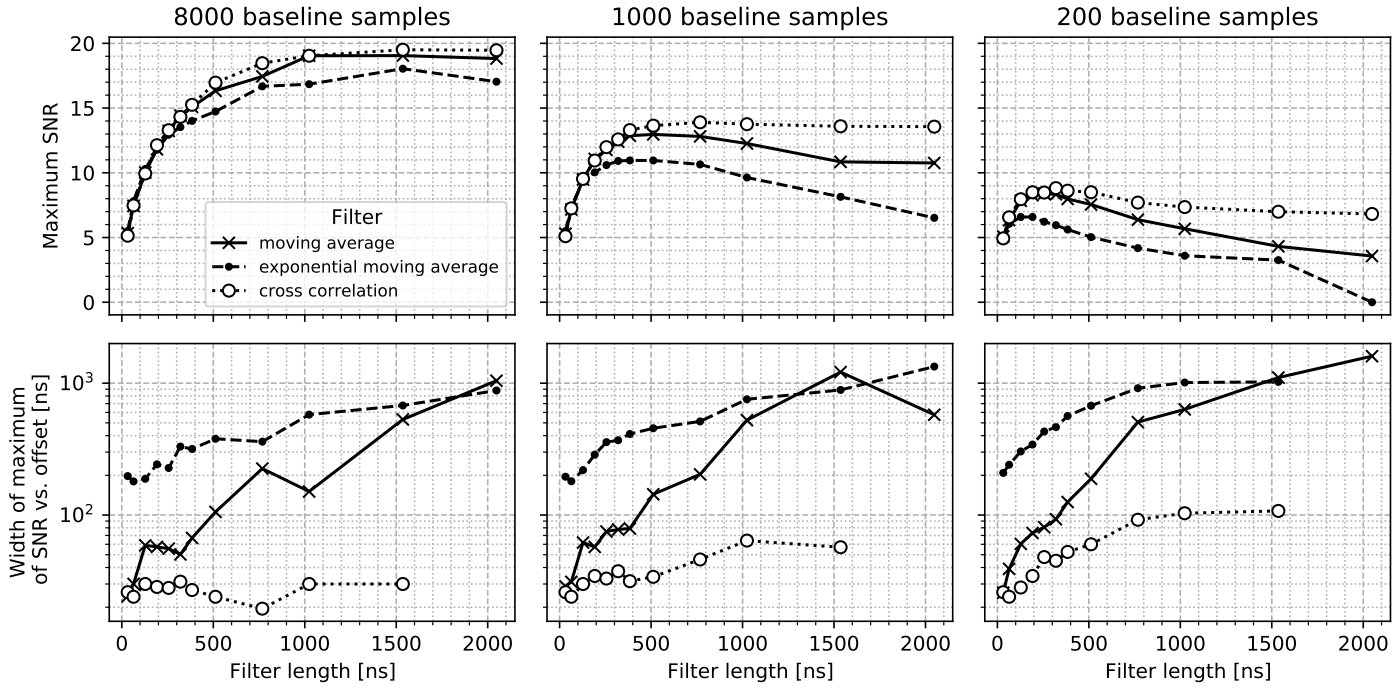

Figure 4.12: The same plot of figure 4.11 repeated changing the number of pre-signal samples used to compute the baseline. The left column is the same data from figure 4.11, with 8000 baseline samples. The performance decreases with a lower number of samples because the standard deviation of the baseline increases and the baseline is compared to the filtered signal amplitude to discriminate signals. (`figchangebs.py`)

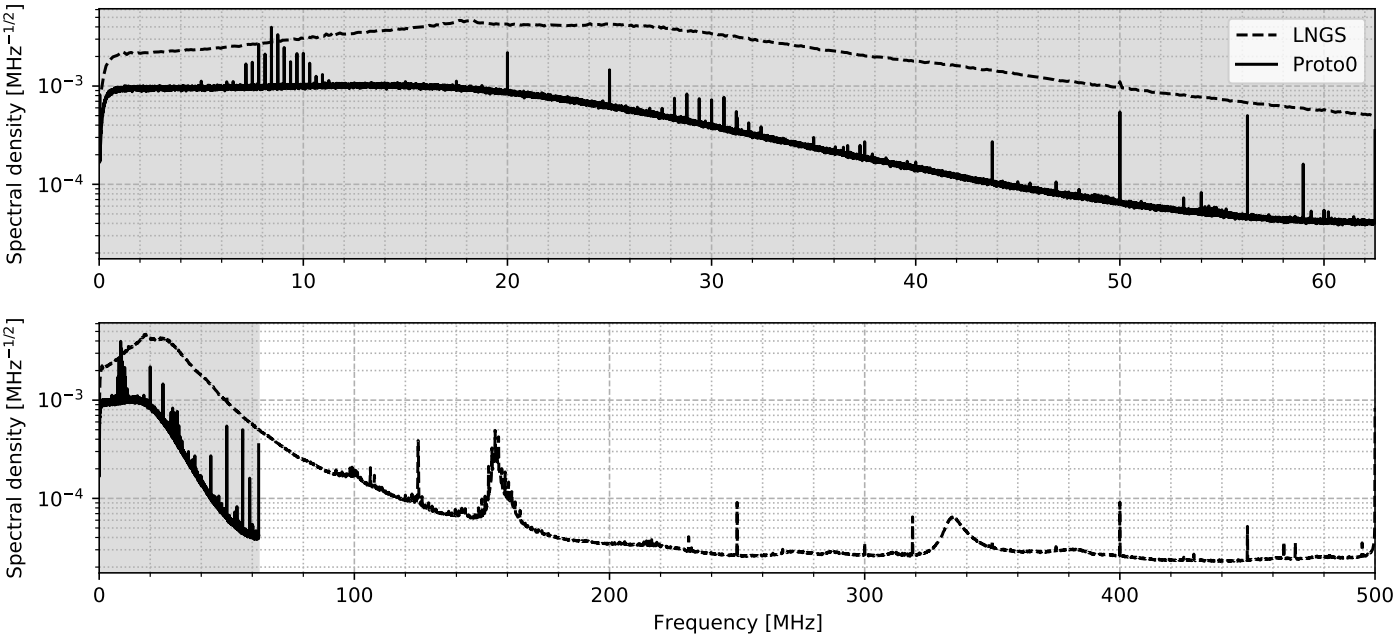

Figure 4.13: Spectrum for the PDM we used in the LNGS test data, both in the LNGS test data setup and in the Proto0 detector setup. While the LNGS data is in working conditions at liquid nitrogen temperature, the Proto0 data we used was taken at room temperature with sensors biased below breakdown voltage for the purpose of measuring noise. (`figspectra.py`)

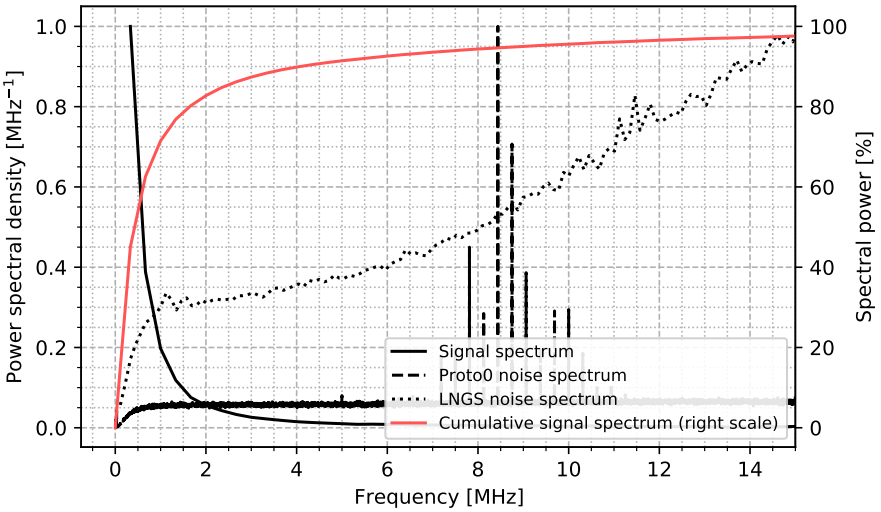

Figure 4.14: Comparison of the spectra of signal and noise. The red curve shows the percentual of signal power below a given frequency. The signal spectrum is computed with the discrete Fourier transform of the cross-correlation filter template (figure 4.6) without windowing. (`figtemplsp.py`)

Chapter 5

Temporal resolution of photon detection

In the previous chapter we studied the performance of filters, but we did not define an algorithm to search for signals. Neither we will do it here, we'll skip it and measure the temporal localization precision once we know there's a signal.

To this end, we make a simulation where each "event" contains only one signal at a known position. In principle we could use the LNGS data (section 4.1), but we don't know the jitter of the trigger pulse and we may reach a temporal resolution below the sampling period, while in the simulation we have the exact actual temporal location of signals.

The code is in the same repository of the previous chapter, `https://github.com/Gattocrucco/sipmfilter`.

5.1 Event generation

Each event is the sum of a signal and a noise waveform. We don't add a baseline, so the noise has mean zero and the signals taper down to zero. The signals are negative. We use the same scale of the LNGS data; it actually does not matter because we are not simulating digitalization.

5.1.1 Signal

We obtain the signal waveform by averaging single photon pulses from the LNGS data (see section 4.1 for a description of the dataset). We do not try to align the signals, assuming that they are aligned relative to the beginning of each acquisition event. We take 3584 1 GSa/s samples. Figure 5.1 shows the obtained signal template.

After I explained the template in the first chapter, refer to that. Redo everything using the filter-aligned template instead of the unaligned or trigger-aligned.

(A study not reported here shows that better alignment is achievable but makes a small difference, and worse alignment means the peak of the signal is smeared thus yielding lower performance in signal localization, and so our choice is irrelevant at best, conservative at worst.)

The template is at 1 GSa/s, but the simulation is at 125 MSa/s. We have to downsample the template and shift its temporal position continuously instead of by $(1\,\mathrm{GHz})^{-1} = 1\,\mathrm{ns}$ steps. Given the continuous temporal position where we want to place the template, we round it by excess and defect to the 1 ns timebase. Then we downsample the template by averaging samples in groups of 8, once with the groups aligned to the floor rounded position, once with the ceiling rounded one. Finally we interpolate linearly between the two downsampled templates. Figure 5.2 shows a series of waveforms obtained in this way.

(Downsampling with an average is not the best antialiasing filter doable, but it should be reasonably fine since the higher spectral part is already suppressed in the signal template.)

I've not said I'm varying the signal amplitude. Maybe I should drop it from the simulation altogether.

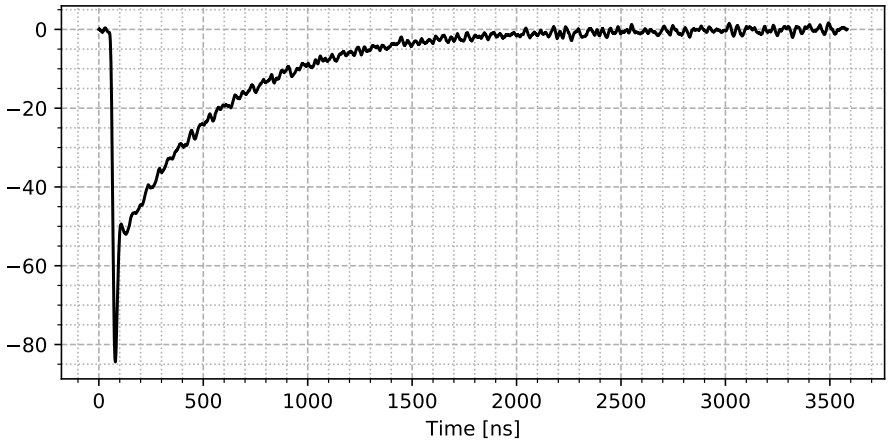

Figure 5.1: The source 1 GSa/s signal template used in the simulation. (`figtoytempl.py`)

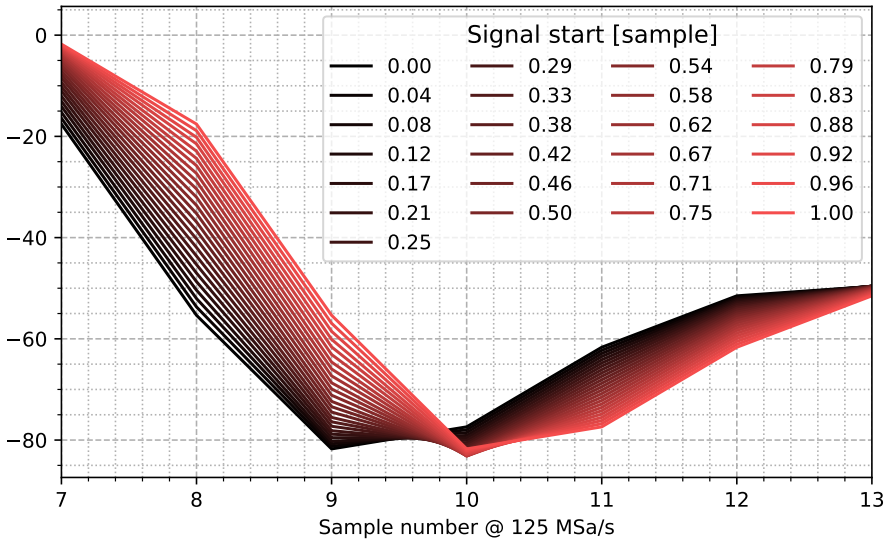

Figure 5.2: The signal template downsampled from 1 GSa/s to 125 MSa/s and translated continuously instead of by discrete steps with linear interpolation. (`figinterptempl.py`)

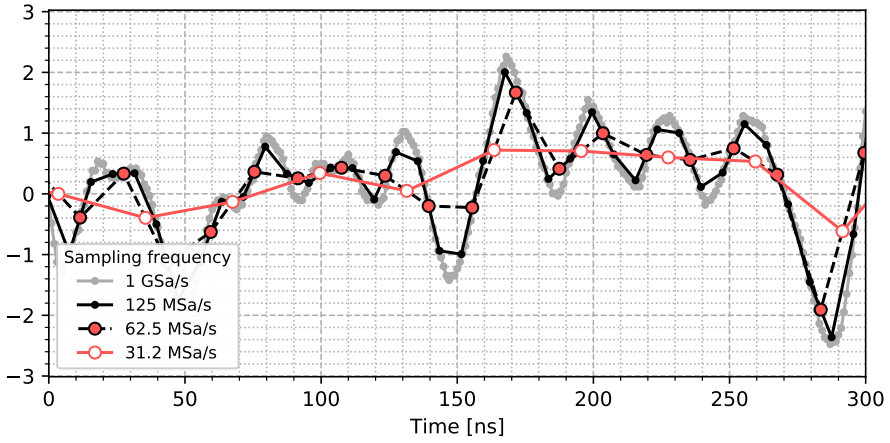

Figure 5.3: The LNGS noise at the original sampling frequency (normalized to unit variance) and downsampled. (`fignoise.py`)

5.1.2 Noise

We used three different kinds of noise: gaussian white noise; noise sampled from the LNGS data; noise sampled from Proto0 data.

The white noise is generated in the simulation. The LNGS noise is copied from the same data we used to make the signal template by taking the part of the events before the signals and filtering out a few events that contained spurious signals in that location. The Proto0 noise is copied from an acquisition made on the same PDM when it was used in the Proto0 setup with the SiPMs under breakdown voltage, thus inactive.

Say precisely how I'm filtering out spurious signals, because maybe I'm not filtering them away completely.

The noise obtained from data is normalized to the variance required by the simulation. For both LNGS and Proto0 noise the data comes divided in events and we normalized separately for each event in case the variance changed.

We downsample the noise in the same way as the signal, by averaging nearby samples. Both noises have spectra that go down with frequency (see section 4.6), so this crappy antialiasing should be sufficient. The normalization to the desired variance is done after downsampling. The order matters because downsampling with averaging reduces the variance of the noise. See figure 5.3. The Proto0 noise data is already at 125 MSa/s and so did not require downsampling for most of the simulations.

5.1.3 Event layout

Each event is the sum of a noise waveform and a shorter signal waveform. Before the beginning of the signal there's a noise-only region long enough for the filters to be in a stationary state when they reach the signal; in particular its length is the highest filter length parameter used in the simulation (2048 ns) plus 256 ns.

The simulation is repeated for various signal to noise ratios (SNR). We define the SNR as follows: the peak height relative to the baseline of the original 1 GSa/s signal template over the noise standard deviation.

I should use the peak height, averaged over continuous positioning, at the simulation sampling frequency, instead of at 1 GSa/s. Then the comparisons at different sampling frequencies would need to adjust both for the reduction of the noise variance and of the peak height.

Simulations with different SNR differ only in the multiplicative constant of the noise, so we used exactly the same noise and signal arrays for each SNR to speed up the code. This means that there's no random variation between results obtained at different SNR (or with different filters), keep this in mind when the smoothness of some plots would seem to suggest that the Monte Carlo error is negligible.

Figure 5.4 shows a complete example event.

5.2 Temporal localization

We run the three filters described in section 4.2 (moving average, exponential moving average, cross correlation), then take the minimum (the signals are neg-

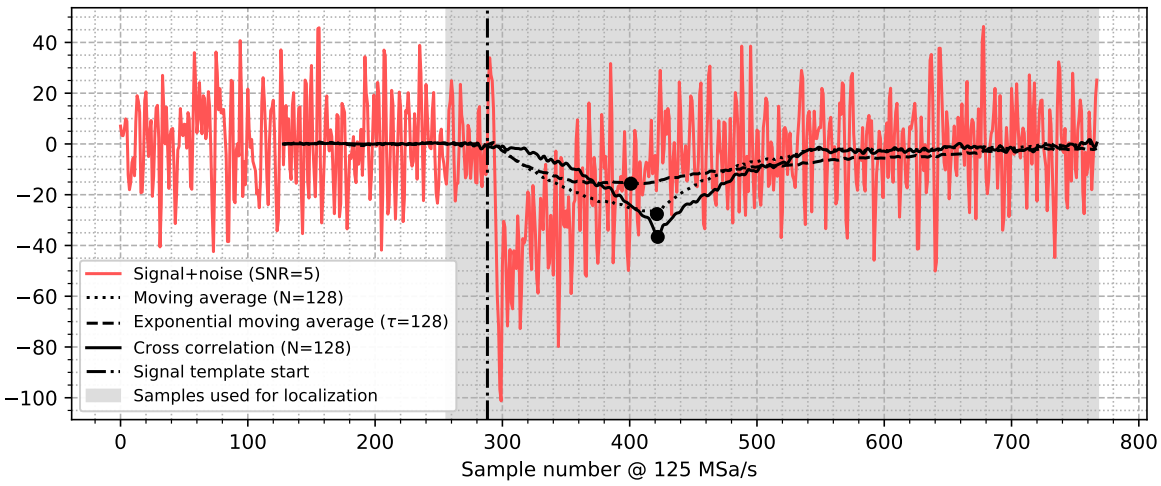

Figure 5.4: A simulation event. The dots are the minima of the filters output. The minima are searched in the shaded region only; this makes no difference with high enough SNR like in this example, but in the limit SNR = 0 the minimum fluctuates around uniformly: the search range sets the endpoints of this distribution. `(figtoyevent.py)`

ative) of the filtered waveform as the location of the signal. We also take the minimum of the unfiltered waveform as baseline comparison.

The minimum of the filter output occurs in some sense later than the signal location, this is not a problem since the choice of the point of the signal to be taken as reference is already arbitrary.

To make the template for the cross correlation filter, we first cut the signal template (section 5.1.1) to the required filter length, keeping the part of the template with maximum euclidean norm, then we downsample it by averaging nearby samples. See figure 5.5.

To allow for a localization eventually more precise than the sampling timebase, we interpolate the minimum sample and its first neighbours with a parabola. We also try upsampling the waveform to 1 GSa/s (with sample repetition) prior to filtering to check if it improves performance.

5.2.1 Resolution

We repeat the simulation for 1000 events for each filter, filter length parameter, and SNR in some range, using the Proto0 noise. Figure 5.6 shows the histograms of the temporal localization for all filters for a choice of SNR and filter length.

Specify how the signal template position varies.

We see that the distribution of localizations can be non-gaussian, so to quantify the resolution we use, instead of the standard deviation, half the distance between the 16 % and 84 % quantiles, which is equivalent to a standard deviation for a gaussian, but gives a meaningful measure for the width of the distribution even when it's highly skewed or with heavy tails.

Figure 5.7 shows the temporal resolution thus defined for each filter, filter length, and SNR. The exponential moving average has a consistently poor performance compared to the other filters. The cross-correlation filter is the best one, with performance improving with length, and at a length of 96 samples (512 ns) is already practically optimal. The moving average can get close to the cross-correlation filter by choosing appropriately the number of samples.

The online processing of the PDM output in the experiment will be done in two steps: the digitizers must find the signals, then send them to the front end processors (FEPs) for further analysis. The computational resources of the digitizers are limited compared to the FEPs.

A schematic of the DAQ would be appropriate. Maybe in a previous introductory chapter.

The exponential moving average can be surely implemented on the digitizers. The cross-correlation with 64 samples could probably be done on the digitizers since a similar computation was implemented in another study.

The FEPs can and should probably use the best filter, so they would run a

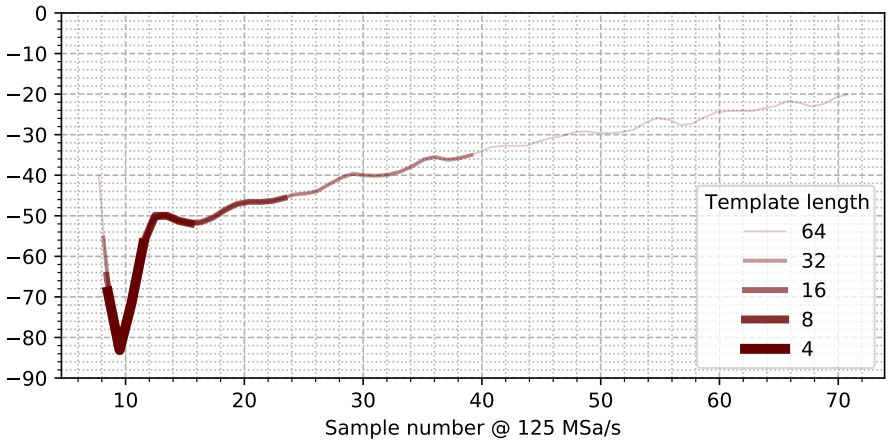

Figure 5.5: Some cross correlation filter templates for different lengths. It may appear strange that the endpoint on the left has a different height than the endpoint on the right for a given template, since we choose the truncation to maximize the norm; it happens because we downsample *after* truncation. (`figtoyfilttempl.py`)

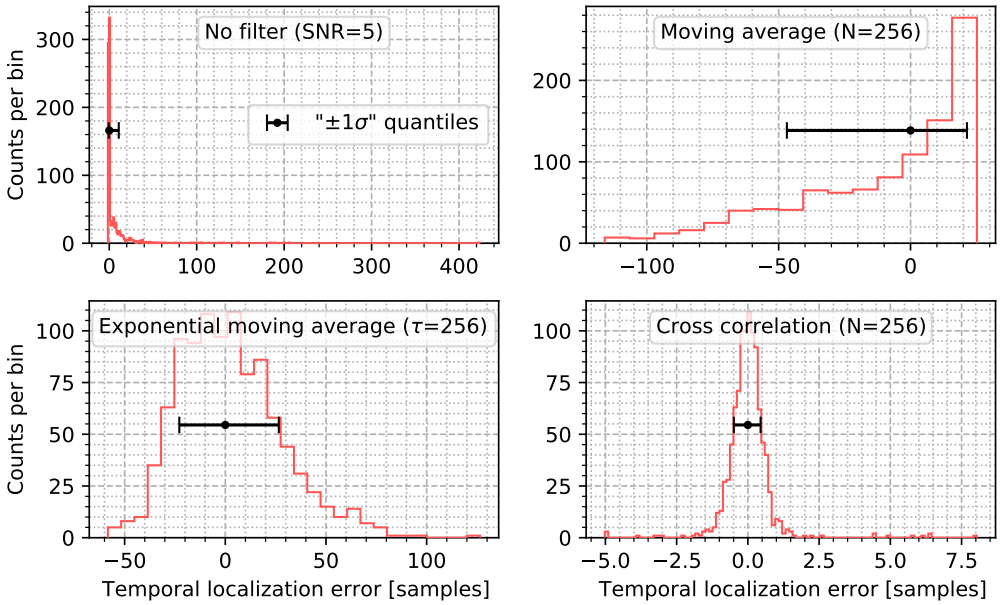

Figure 5.6: Histograms of the temporal localization error, i.e. the difference between the filter output minimum and the signal template start, translated to have zero median, for a choice of SNR and filters length. The error bars mark the 16 % and the 84 % quantiles. As definition of temporal resolution we take half the distance between those quantiles. The sampling step is 8 ns. (`figlochist.py`)

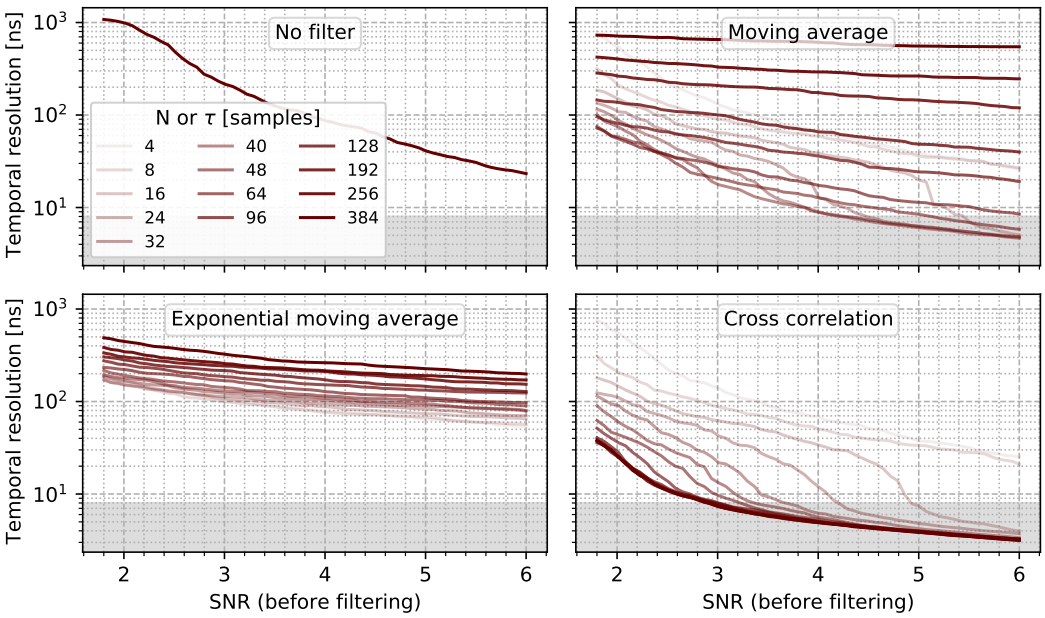

Figure 5.7: Photodetection temporal resolution for a range of SNR and filter lengths. The shaded region marks the sampling step (8 ns). The right endpoint of the cross correlation filter curves is at 3 ns. (`figrescurve.py`)

long cross-correlation filter. The temporal resolution matters in the FEPs but probably not in the digitizers, in the latter case it is just a generic measure of perfomance.

Thus out of all the temporal resolution curves the ones that matter are:

- the best we can do with the exponential moving average and moving average;
- the long cross-correlation filters;
- the 64 samples cross-correlation filter.

We plotted these curves together in figure 5.8, adding some curves done with the LNGS and white noises. Note that a different noise spectrum makes a large difference at low SNR. We also plot a curve computed with upsampling; it does not improve significantly the performance.

If at the end the comparison plot still uses up one page, make it higher so it's more legible at lower temporal resolutions.

5.3 Data reduction

We said that the digitizers must find signals in the waveform stream and send them to the FEPs for further processing. The bandwidth of the connection between the digitizers and the FEPs happens to be a bottleneck. Two possible ways of reducing the amount of transmitted data are keeping only the minimum number of samples for each signals, and reducing the sampling frequency. Both have an effect on the temporal resolution, which we assess here.

5.3.1 Waveform truncation

We repeat the simulation, but this time we use only a fixed smaller number of samples in each event to compute the filter output. We call this selection of samples "window". On the window we run only a long cross-correlation filter since that's what would be done on the FEPs. As past and future boundary condition we use zero. We evaluate the filter even after the sample window end because the window can be shorter than the filter.

While the length of the window is fixed, the placement is not fixed relative to the true signal location. Instead we use the temporal localization with another filter feasible on the digitizers, calibrated to have the median aligned to the

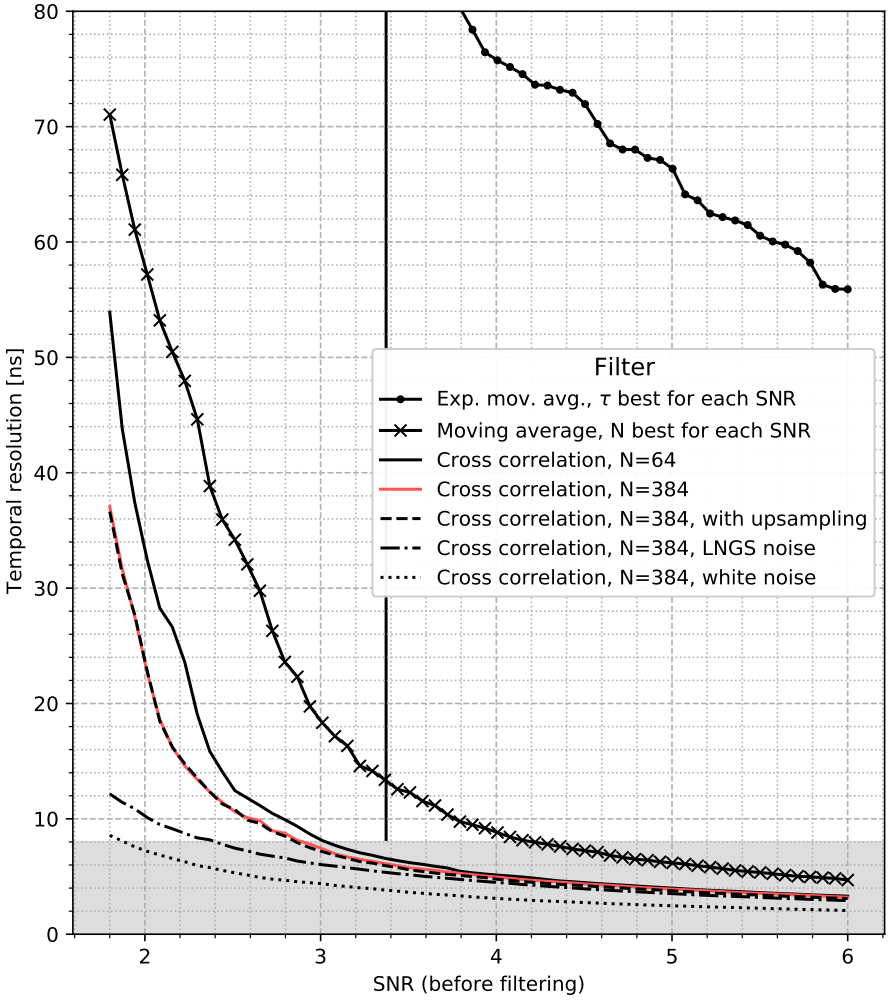

Figure 5.8: Photodetection temporal resolution vs. SNR for various filters. The shaded region marks the sampling step (8 ns). The hatched band is the interval of SNR observed in Proto0; the vertical line is the SNR in the LNGS data (after downsampling to 125 MSa/s). Where not specified, the noise is from Proto0. (`figrescomp.py`)

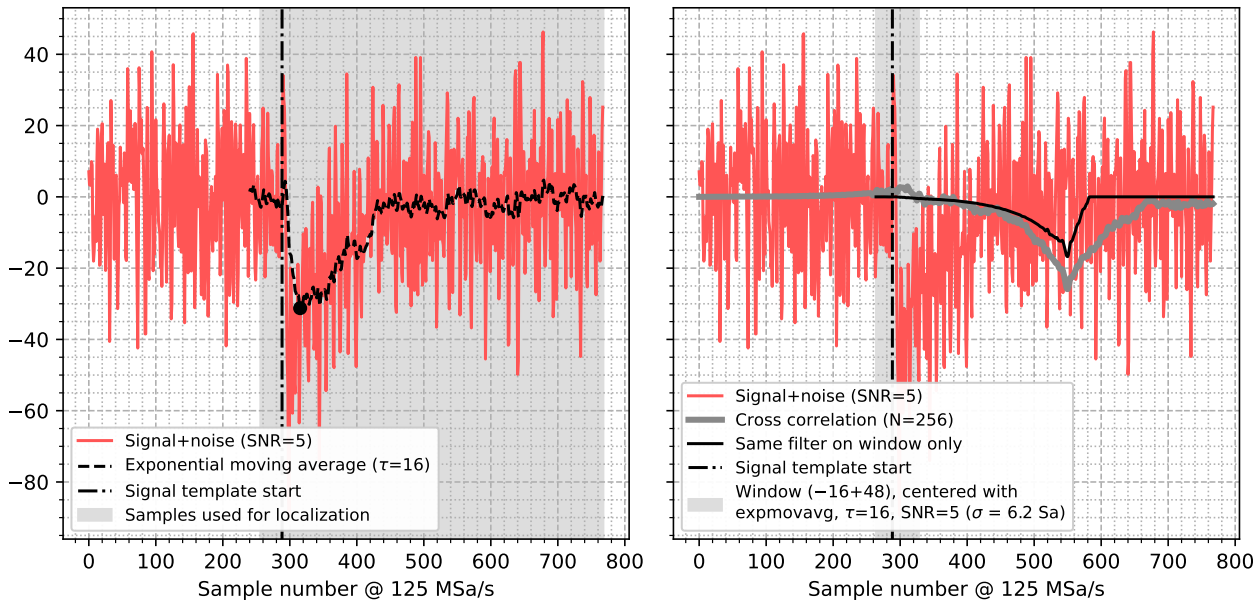

Figure 5.9: Left panel: a simulation event filtered with the exponential moving average. Right panel: the same event filtered with a long cross correlation filter, both using the whole waveform and using only the samples in the shaded window, which is centered using the localization from the filter in the left panel. `(figwindowevent.py)`

beginning of the signal template. The window then extends a given number of samples to the left and to the right of this localization.

Figure 5.9 shows this procedure graphically for a single event. Figure 5.10 shows the temporal resolution versus unfiltered SNR curves for various choices of window length, noise, and filter used to align the window, where for reasons of computation time the latter was computed at a fixed SNR that does not follow the value on the x-axis.

By looking at figure 5.10, we conclude that probably it is sufficient to save 1 µs of waveform to obtain practically the same temporal resolution achievable without windowing.

What's missing in this study is that we did not try to optimize the left/right balance of the window, and that as said above the unwindowed localization used to align the windows is done at a mismatched SNR. The first problem can only worsen the temporal resolution obtained, so it is conservative; the second is conservative assuming that our choice of SNR (2.4) is lower that what we expect in the detector.

Maybe with the channel summing this SNR will become realistic.

5.3.2 Downsampling

Another way of reducing the data throughput is downsampling. In figure 5.11 we show the temporal resolution achieved with a long cross correlation filter, for white and LNGS noise, at different sampling frequencies. We can observe that downsampling by a factor of 2 from 125 MSa/s to 62 MSa/s maintains almost the same temporal resolution, while going to 31 MSa/s lowers it visibly.

Explain the SNR rescaling.

Since we are downsampling we also need to check if we lose signal to noise ratio in the filter output. In table 5.1 we report the ratio between SNR after and before filtering. It does not change significantly with downsampling.

I'm measuring the post-filter SNR with the peak detection. This biases upward the result. I should compute the peak height on the filtered signal before adding the noise.

Add digitalization to the simulation and make a plot like the one for the sampling frequency but varying the number of bits at 125 MSa/s. Don't use the number of bits, use the ratio signal peak over digit which is well defined.

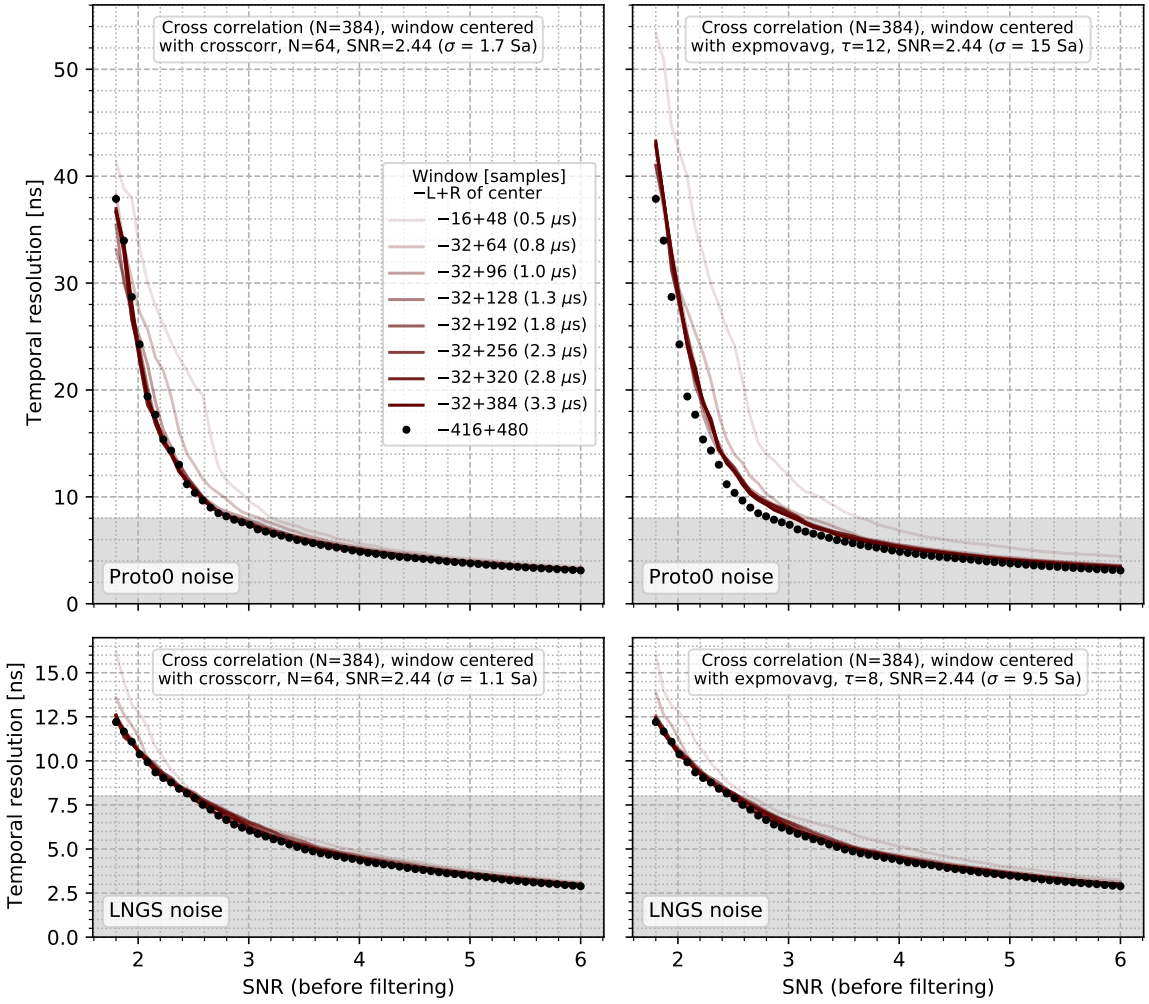

Figure 5.10: Photodetection temporal resolution with a long cross correlation filter applied only on a short window of samples centered using a shorter cross correlation filter (left panels) or an exponential moving average (right panels). The various curves correspond to different window lengths, while the black dots are the resolution without windowing. (`figwindowtempres.py`)

	SNR after over before filtering			
Noise	1 GSa/s	125 MSa/s	62.5 MSa/s	31.2 MSa/s
Proto0		3.6	3.6	3.6
LNGS	6.3	6.3	6.5	6.6
White	4.7	4.7	4.7	4.7

Table 5.1: Ratio of SNR after over before filtering with a cross correlation filter with template length 2048 ns. The 125 MSa/s column contains the actual SNR ratios of the simulations, while the values for the other sampling frequencies are divided by the noise standard deviation reduction with downsampling relative to 125 MSa/s to make them comparable.

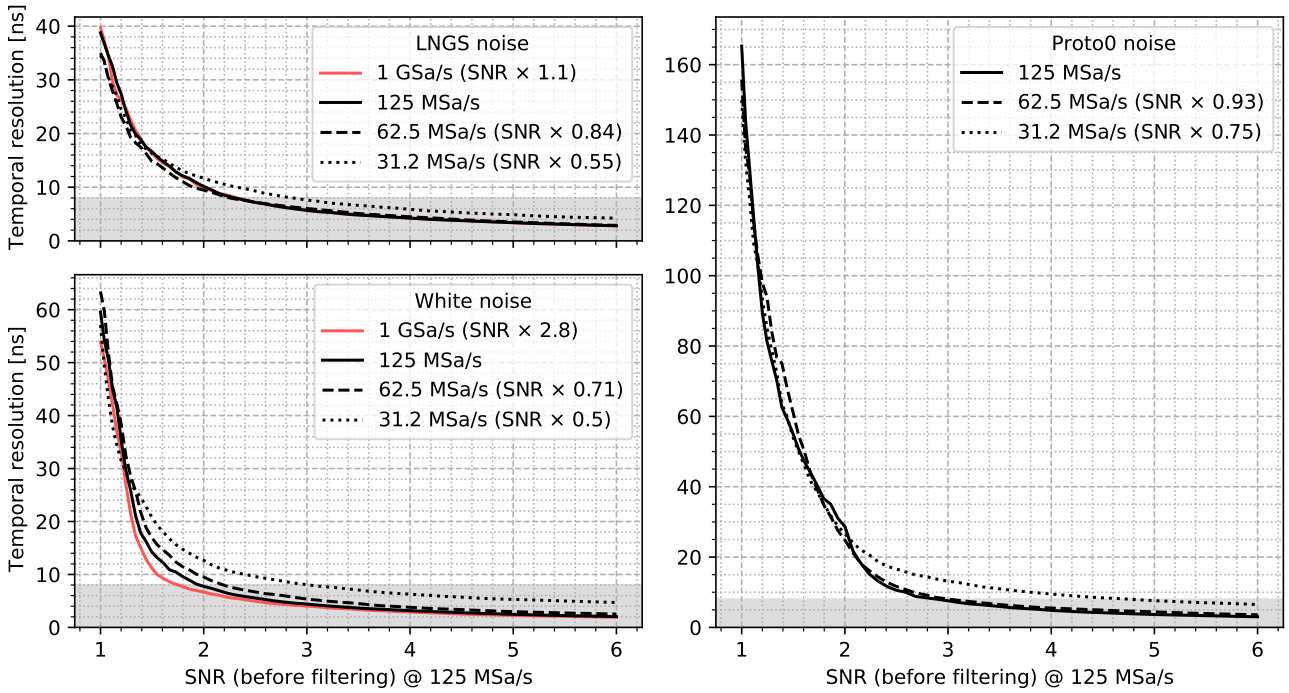

Figure 5.11: Photodetection temporal resolution at different sampling frequencies with a cross correlation filter with template length 2048 ns. The SNR scale is at 125 MSa/s; curves for different sampling frequencies are rescaled horizontally by the factor written in the legend to account for the noise variance reduction with downsampling. (`figtempresdowns.py`)

Chapter 6

Fake rate of photon detection

In chapter 4 we measured the signal to noise ratio after filtering. The point of using the SNR is expressing the signal height relative to the width of the noise distribution, because the threshold required to reject noise with a fixed probability is proportional to that scale. So it is convenient to express the threshold relative to the same scale, and check it is less than the SNR so that signals are not selected away.

Even assuming that the noise is gaussian, the probability that a random fluctuation gets over the threshold is not given simply by computing the survival function (i.e. the integral to $+\infty$) of the gaussian distribution at the threshold.

More precisely, the probability that any given sample is above the threshold is given by such integral. What we need, however, is the *rate* of threshold *crossings*. The noise is not white, but even if it was, after applying the filter, which combines linearly many input samples for each output sample, the waveform is autocorrelated at least up to the length of the filter. Intuitively, if a smooth function crosses a threshold, it takes some time to go down before it can cross the threshold again.

Since the first stage filtering procedure that will be used in DarkSide20k is not decided at this point, for brevity we will just see how to compute the threshold crossing rate of the noise, called fake rate, for a specific example filter, and check that it works. The method can then be applied to any filter of choice.

6.1 Theory

We expect the noise to be gaussian. Even if it wasn't, when filtering many samples are linearly combined, and the sum of random variables tends to have a gaussian distribution independently of the initial one, so we assume gaussianity.

6.1.1 From the continuous case

We have a discrete sequence of samples. The continuous equivalent is a Gaussian process. We can expect the discrete case to be equivalent to the continuous case if the autocorrelation time is larger enough than the sampling step, which should hold from the consideration above.

Also, even though the values are initially discrete too, after filtering the possible non integer values between two consecutive integers are at least the length of the filter (think about an average). So we take the formula for the continuous case and adapt it.

The mean number of threshold upcrossings r in the interval $(0, 1)$ by a zero-mean stationary and appropriately smooth Gaussian process is given by [RK06, p. 81]

$$r = \sqrt{-\frac{k''(0)}{2\pi}}\,\mathrm{gauss}(u; 0, \sigma) = \tag{6.1}$$

$$= \frac{1}{2\pi}\frac{\sqrt{-k''(0)}}{\sigma}\exp\left(-\frac{1}{2}(u/\sigma)^2\right), \tag{6.2}$$

where u is the threshold, σ the noise standard deviation (the RMS), $\text{gauss}(x;\mu,\sigma)$ a Gaussian probability density on x with mean μ and standard deviation σ, and k the autocovariance function, i.e. $k(x) = \text{Cov}[f(t), f(t+x)]$ for any t (for example, $k(0) = \sigma^2$), where $f(t)$ is the continuous waveform.

We have to map the second derivative of the autocovariance function to a discrete equivalent. We first do a manipulation in the continuous realm. Since the covariance operator is an integral, it commutes with derivation:

$$k''(x) = \frac{\partial^2}{\partial x^2}\,\text{Cov}[f(t), f(t+x)] = \tag{6.3}$$

$$= \text{Cov}[f(t), f''(t+x)], \tag{6.4}$$

thus $k''(0) = \text{Cov}[f(t), f''(t)]$. We estimate the second derivative with a finite difference:

$$f(t \pm \Delta t) = f(t) \pm f'(t)\Delta t + \frac{1}{2}f''(t)\Delta t^2 + O(\Delta t^3) \to \tag{6.5}$$

$$\to f''(t)\Delta t^2 = f(t+\Delta t) + f(t-\Delta t) - 2f(t) + O(\Delta t^3). \tag{6.6}$$

Choosing $\Delta t = 1/f_s$, where f_s is the sampling frequency, and calling $y_i = f(t_0 + i\Delta t)$ the samples, we have:

$$k''(0) \mapsto f_s^2 k_2, \tag{6.7}$$

$$k_2 \equiv \text{Cov}[y_i, y_{i+1} + y_{i-1} - 2y_i], \tag{6.8}$$

$$r = f_s \frac{1}{2\pi}\frac{\sqrt{-k_2}}{\sigma}\exp\left(-\frac{1}{2}(u/\sigma)^2\right). \tag{6.9}$$

The reader can check that discretizing directly $k''(0)$ yields the same result.

The covariance in equation 6.8 can be estimated with the sample covariance on a filtered waveform array $\mathbf{y}$.

6.1.2 Direct discrete derivation

Since the formula we derived is approximate, as a cross check we derive another approximate one following a different path.

A threshold crossing happens when a sample is below the threshold and the next one is above: $y_i \le u$, $y_{i+1} > u$. Fix $i = 0$ and let $p(y_0, y_1)$ be the joint distribution of the two samples. The probability of crossing at any given point then is

$$P = \int_{-\infty}^{u} \mathrm{d}y_0 \int_{u}^{\infty} \mathrm{d}y_1\, p(y_0, y_1). \tag{6.10}$$

In general we can not obtain the crossing rate just by multiplying P by the sampling frequency because of correlations. However in practice we are interested in low crossing rates, on the order of 10 cps, to be compared to the sampling frequency 125 MSa/s. If the typical time between crossings is much longer than the autocorrelation time, then we can ignore correlations. Thus the crossing rate is $r = f_s P$.

The integrand in equation 6.10 is a bivariate gaussian distribution, which explicitly is

$$p(y_0, y_1) = \frac{1}{2\pi\sqrt{\sigma^4 - c^2}}\exp\left(\frac{1}{2}\begin{pmatrix} y_0 & y_1 \end{pmatrix}\begin{pmatrix} \sigma^2 & c \\ c & \sigma^2 \end{pmatrix}^{-1}\begin{pmatrix} y_0 \\ y_1 \end{pmatrix}\right), \tag{6.11}$$

where $c = \text{Cov}[y_0, y_1]$.

We don't know how to the integral analytically, so we break down the joint distribution as $p(y_0, y_1) = p(y_1|y_0)p(y_0)$ and discretize the integral over y_0:

$$p(y_0) = \text{gauss}(y_0; 0, \sigma), \tag{6.12}$$

$$p(y_1|y_0) = \frac{p(y_0, y_1)}{p(y_0)} = \text{gauss}\left(y_1; \frac{c}{\sigma^2}y_0, \sqrt{\sigma^2 - \frac{c^2}{\sigma^2}}\right), \tag{6.13}$$

$$P \approx \sum_{k=0}^{N-1} \Delta u\, p(y_0(k, u, \Delta u)) \int_{u}^{\infty} \mathrm{d}y_1\, p(y_1|y_0(k, u, \Delta u)), \tag{6.14}$$

$$y_0(k, u, \Delta u) \equiv u - k\Delta u. \tag{6.15}$$

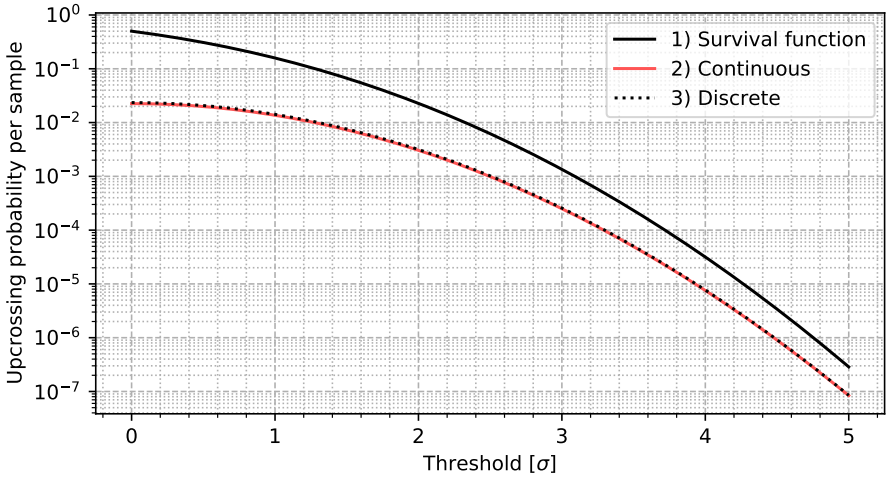

Figure 6.1: The threshold upcrossing rate expressed as per-sample crossing probability (i.e. the rate if the sampling frequency is 1) for an autocorrelated Gaussian waveform, estimated using three formulae: 1) the probability for a single sample to be higher than the threshold, 2) a formula for continuous processes (equation 6.9), 3) an approximation of the discrete case (equation 6.14).

The integral on y_1 can be computed using the error function. Δu should be chosen small compared to σ, while N large relative to $\sigma/\Delta u$.

In figure 6.1 we compare formula (6.9) (with $f_s = 1$) with P and with the Gaussian survival function. To make the comparison we have to use a k_2 that matches c:

$$\begin{aligned} k_2 &= \operatorname{Cov}[y_i, y_{i+1} + y_{i-1} - 2y_i] = \\ &= \operatorname{Cov}[y_i, y_{i+1}] + \operatorname{Cov}[y_i, y_{i-1}] - 2\operatorname{Cov}[y_i, y_i] = \\ &= 2(c - \sigma^2). \end{aligned} \tag{6.16}$$

We use $\sigma = 1$, $c = 99\,\%$, $\Delta u = 1/100$, $N = 500$ (we decreased Δu until convergence). We see that our derivation and the formula for Gaussian processes agree very well, while differing visibly from the survival function. We will henceforth use the continuous formula for its simplicity.

6.1.3 Dead time

The part of the data acquisition system (DAQ) that will look for threshold crossings are the digitizers. A digitizer can not do complicated processing, so when the threshold is crossed, a fixed slice of waveform is sent to the front end processing (FEP) for further analysis (identify multiple signals, locate them precisely, use a better filter, etc.). So we are not interested in threshold crossings that happen close to a previous crossing.

This means that we have a dead time T. We arbitrarily decide to use a non-restartable dead time, i.e. a crossing that happens within T of a previous one is ignored only if the latter has not been ignored itself.

Assuming that the crossings are a Poisson process, the formula to correct a rate R for the effect of the dead time is [Kno10, p. 120]:

$$R \mapsto \frac{R}{1 - RT}. \tag{6.17}$$

The crossings of a Gaussian process are not in general a Poisson process. We just need one counterexample to show this. Consider the process with autocovariance function $k(x) = \cos(x)$. This is positive definite because it is a linear combination of external products: $\cos(x - y) = \cos x \cos y + \sin x \sin y$. Since cos is orthogonal to sin, they are the eigenfunctions, so a realization of the process is a random linear combination of harmonic functions, which means that it is a shifted cosine, so the crossings are exactly periodic.

However, in practice we expect that there will just be a "repulsion" or "attraction" of crossings within the scale of the autocorrelation time, so for low crossing rate the formula should work.

6.2 Test

We want to test formula (6.9) on actual electrical noise.

6.2.1 Data

We will use the Proto0 run 886, a baseline acquisition. Tiles 53, 57 and 59 (used in Proto0) will be also studied with LNGS data. Tile 15 just with LNGS. The LNGS data files are:

```
FBK/NUV/MB2-LF-3x/NUV-LF_3x_53/nuvhd_lf_3x_tile53_77K_64V_6VoV_1.wav
FBK/NUV/MB2-LF-3x/NUV-LF_3x_53/nuvhd_lf_3x_tile53_77K_66V_7VoV_1.wav
FBK/NUV/MB2-LF-3x/NUV-LF_3x_57/nuvhd_lf_3x_tile57_77K_64V_6VoV_1.wav
FBK/NUV/MB2-LF-3x/NUV-LF_3x_59/nuvhd_lf_3x_tile59_77K_64V_6VoV_1.wav
LFOUNDRY/pre-production-test/TILE_15/LF_TILE15_77K_55V_0VoV_1.wav
LFOUNDRY/pre-production-test/TILE_15/LF_TILE15_77K_59V_2VoV_1.wav
LFOUNDRY/pre-production-test/TILE_15/LF_TILE15_77K_63V_4VoV_1.wav
LFOUNDRY/pre-production-test/TILE_15/LF_TILE15_77K_67V_6VoV_1.wav
LFOUNDRY/pre-production-test/TILE_15/LF_TILE15_77K_71V_8VoV_1.wav
LFOUNDRY/pre-production-test/TILE_15/LF_TILE15_77K_73V_9VoV_1.wav
```

Expand and adapt this paragraph after chapter 3 is done.

The Proto0 data is without signals so we use it as it is. For the LNGS data we take the pre-trigger part of the events, and ignore events where any pre-trigger sample is less than 750 (860 for tile 15).

We plot the time-value histogram of the data for tile 53 in Proto0 and LNGS (figure 6.2), for tile 15 at maximum overvoltage, and for tiles 57 and 59 (figure 6.3).

No need for more explanations of the time-value histograms since there will be some of them in chapter 3.

6.2.2 Filter

We filter using a 1 µs moving average with 1 µs of baseline and 1 µs of dead time, without delay between the baseline and the signal averages. In figure 6.4 we show an example filtered waveform and the filter shape.

From chapters 4 and 5 we know this is not the optimal filter by SNR neither by temporal resolution. However since we don't know which filter will be used it is appropriate to make a conservative choice.

Since on LNGS data the events are short compared to the filter length (9 µs vs. 2 µs), we need to take into account boundary effects. The output length of the filtered waveform is (initial length) − 2 µs + 1 sample; to compute the rate we have to divide by this quantity instead of the initial waveform length.

The dead time does not play nicely with borders, because a hypothetical unseen crossing within 1 µs before the event start could kill a crossing in the event. Moreover, at low thresholds, the first crossing will happen almost immediately, and again immediately after the dead time ends, thus the number of crossings per event is quantized.

A complete solution to these problems would be to avoid counting the crossings which happen within 1 µs after the event start (although they are detected and do project a dead time on the following ones), and in the remaining region further select a subregion which is 1 µs shorter but has a uniformly random starting position.

However we care only about low rates, and at low rate the dead time boundary effects are negligible, so we will keep the whole filtered region.

6.2.3 Algorithm

The simplest way to count the threshold crossings as a function of the threshold is to repeat the calculation varying the threshold. The computational complexity is $O(nN)$ where n is the number of thresholds and N the number of samples.

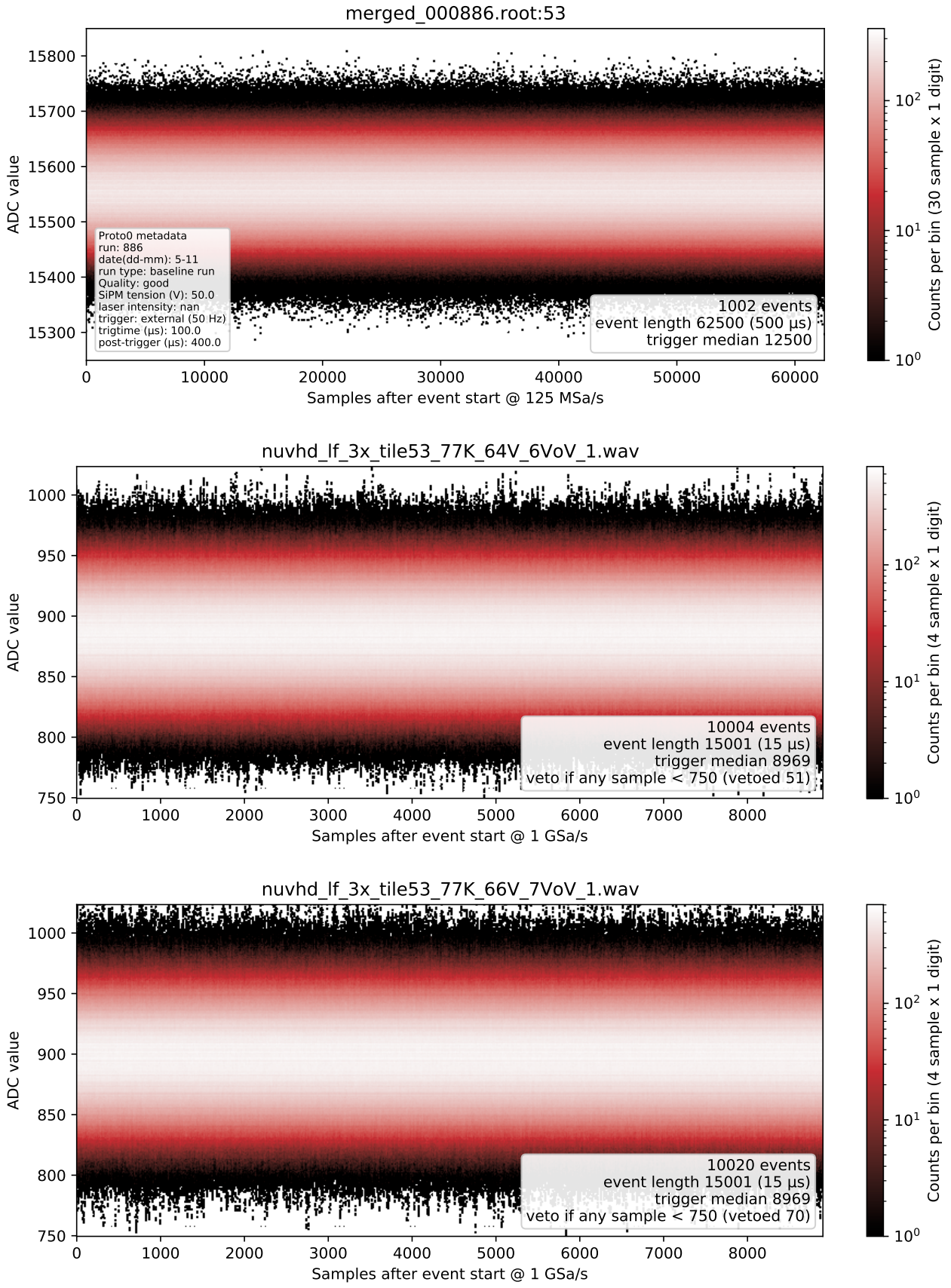

Figure 6.2: Time-value histograms of tile 53 noise in Proto0 with a baseline acquisition, and in LNGS pre-trigger at overvoltage 6 V and 7 V. (`fighist2dtile53.py`)

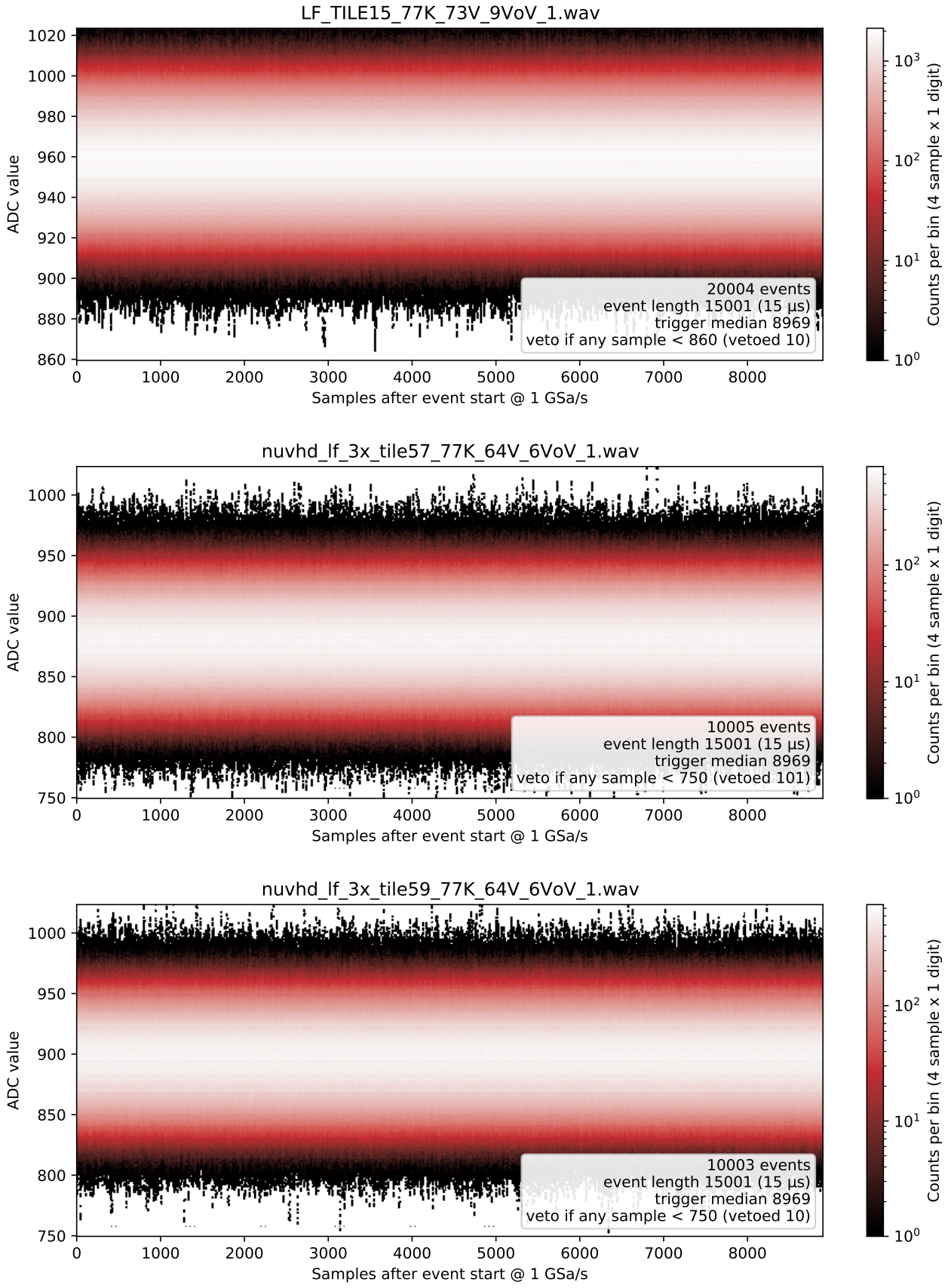

Figure 6.3: Time-value histograms of tiles 15, 57 and 59 noise in LNGS data. (`fighist2dtile155759.py`)

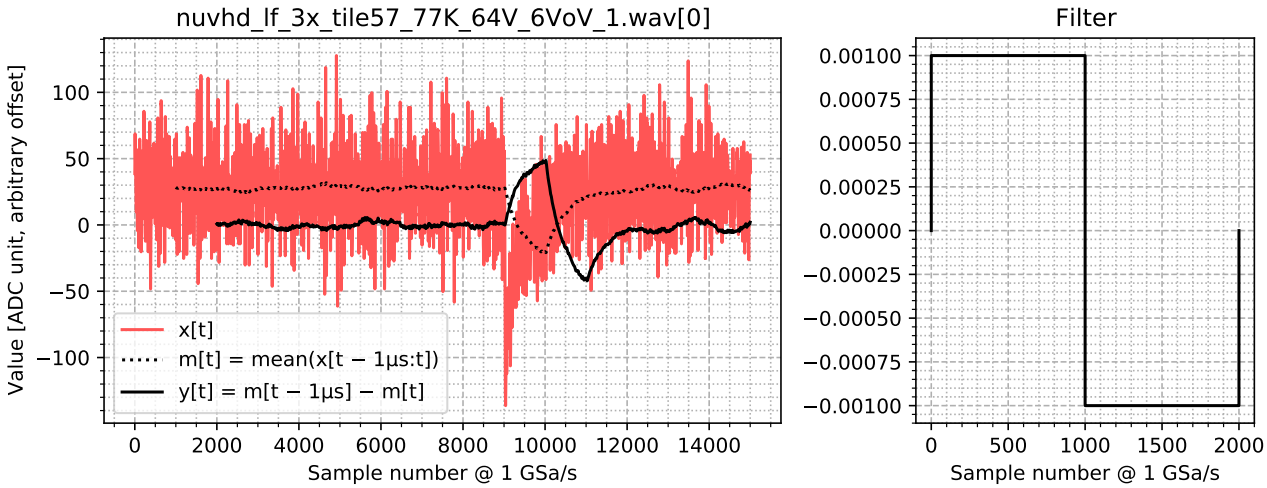

Figure 6.4: A filtered waveform. The computation is split into a moving average m (dotted line) and its finite difference y (solid black line). The right plot shows the overall filter shape. (We use an event with a signal just to see how the filter behaves, the analysis is done on noise only.) (`figsqfilt.py`)

To produce a smooth curve (large n) we use instead a reverse histogram. We choose an evenly spaced range of thresholds. For each pair of consecutive samples, if the second sample is higher than the first, we determine the subrange of thresholds that falls between the samples, and increment their counts. To apply the dead time, we keep a per-threshold last occurence time.

Since the range of thresholds is evenly spaced, the subrange can be found arithmetically, so the complexity is just $O(N)$, it does not depend on the number of thresholds.

6.2.4 Results

In figure 6.5 we compare the measured threshold crossings with the continuous-derived formula (6.9) for a single tile over the entire threshold range. The coefficients σ and k_2 for the formula are computed on each filtered event and then averaged. Finally, the dead time is accounted for with (6.17). For the data, the conversion from count to rate is done dividing by the length of the filtered waveform (so 2 µs less than the initial length per event).

The agreement is decent at low and high rates, less so in an intermediate region. The agreement at high rate is probably a sort of saturation due to dead time, since the maximum rate allowed by dead time is $1/(1\,\mu s) = 1\,\text{Mcps}$. Keep into account that the scale is logarithmic, so even where the lines are closer they still differ by a factor 1.3, while the Poisson error is approx. 3 % since the count is 1000.

In figure 6.6 we show the same comparison together for all datasets, divided in three groups (tile 15 LNGS data, Proto0 data, LNGS other tiles). The parameters for the formula, and the rates at threshold $4\,\sigma$, are listed in table 6.1.

In the second group above some threshold the measured rate stops decreasing and remains constant at approximately 10 counts. This is probably due to pulses which our very simple veto can not filter away, we guess 1 pe dark/random pulses.

In figure 6.6 we highlight the measured rates for tile 53 because they are evident outliers. The rate remains higher than the theory predicts and than the other tiles as the threshold increases. This is visibile in Proto0, and in LNGS at 6 VoV, but not at 7 VoV.

Analogously, from table 6.1 we see that the quantity $f_s\sqrt{-k_2}/(2\pi\sigma)$ that multiplies the exponential in (6.9) is different from the others for tile 53, in the same cases as the data, but with the opposite trend. This variation seems to depend only on an increase in σ and not on k_2.

Even though it's weird that this problem disappears at 7 VoV, it can't be a coincidence that it arises for the same tile in different setups, showing up both

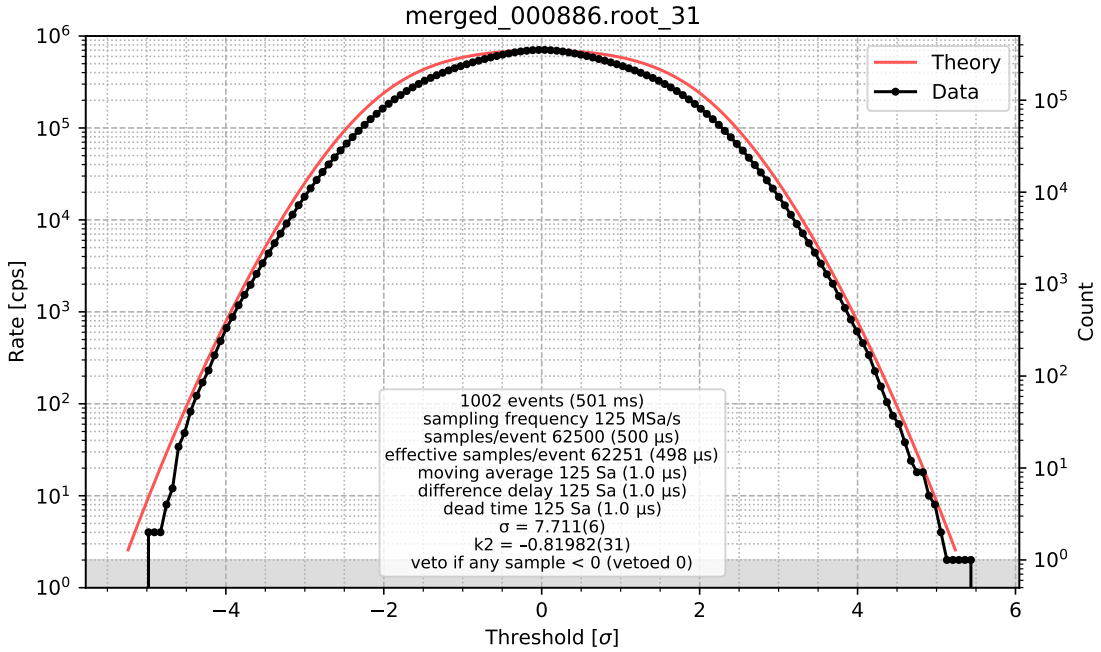

Figure 6.5: Measured and predicted fake rate for tile 31 in Proto0 with a noise-only acquisition. The right scale shows the actual count of threshold crossings for the data, with a gray band marking one crossing. (`figfakerate1.py`)

as an increased noise RMS and an higher threshold crossing rate, so it must be some property of the tile itself.

Comparing the 2D histograms for tile 53 (figure 6.2) to the others (figure 6.3) there is no apparent difference. Three possible explanations come to mind: 1) a violation of Gaussianity, 2) stray pulses, 3) electrical noise.

Regarding Gaussianity, we checked the distribution of the samples before filtering and it agrees very well with a Gaussian. If they are stray pulses, they do not have the same height, otherwise they would show up as a flat rate curve. It could be electrical noise with a frequency of approx. $1/(2\,\mu\text{s}) = 500\,\text{kHz}$, since our filter would be very good at picking that up.

We didn't investigate further the discrepancy for tile 53. Anyway, for all other tiles we looked at it works well enough.

We now want to estimate the minimum amount of data required for the procedure. From table 6.1 we see that the relative error on k_2 times the square root of the time, $C \equiv \text{Std}[\bar{k_2}]\sqrt{T}/|k_2|$, is approximately $8\,\text{ns}^{1/2}$ in all cases. The error should be proportional to $T^{-1/2}$, so to have an 1 % error we need $T_{1\,\%} = (100C)^2 = 0.7\,\text{ms}$.

For convenience we summarize all the steps to compute the fake rate:

1. Acquire 1 ms of noise data.
2. Filter the data (including baseline subtraction) producing a filtered waveform $\mathbf{y}$.
3. Compute the standard deviation σ and $k_2 = \text{Cov}[y_i, y_{i+1} + y_{i-1} - 2y_i]$.
4. Compute the fake rate for threshold u using $r = f_s\sqrt{k_2/(2\pi)}\,\text{gauss}(u; 0, \sigma)$ where f_s is the sampling frequency.

The probability of the formula not working due to mysterious reasons is 4 %. If it works, the uncertainty for low rate is ±50 % and the result is an overestimate with probability 90 %. (These assessments are based on table 6.1 but are somewhat subjective.)

Alternatively, if one has a noise spectrum available but not the noise waveform, obtain the autocovariance by doing the discrete Fourier transform of the power spectrum [Fer15, p. 84] and then normalizing it to be σ^2 in 0. Then k_2 is

Data								Rate @ 4σ	
Setup	Tile	Overvoltage [V]	T [ms]	σ [u]	k_2 [u^2]	$\mathrm{Std}[\bar{k_2}]\sqrt{T}$ [u^2 ns$^{1/2}$]	$f_s\sqrt{-k_2}/(2\pi\sigma)$ [Mcps]	Theory [kcps]	Data [kcps]
LNGS	15	0	138	1.6	−0.0017	0.012	4.2	1.4	1.1
LNGS	15	2	138	1.5	−0.0017	0.012	4.3	1.4	1.2
LNGS	15	4	138	1.5	−0.0017	0.012	4.3	1.4	1.3
LNGS	15	6	138	1.5	−0.0017	0.012	4.4	1.5	1.1
LNGS	15	8	138	1.5	−0.0017	0.012	4.5	1.5	1.4
LNGS	15	9	138	1.4	−0.0017	0.012	4.6	1.5	1.3
LNGS	53	6	69	3.8	−0.0044	0.030	2.7	0.92	2.6
LNGS	53	7	69	2.1	−0.0043	0.029	5.0	1.7	1.0
LNGS	57	6	68	2.2	−0.0043	0.031	4.8	1.6	1.2
LNGS	59	6	69	2.2	−0.0040	0.028	4.6	1.5	1.0
Proto0	29	<0	499	8.3	−0.86	10.0	2.2	0.74	0.77
Proto0	30	<0	499	6.9	−0.75	6.0	2.5	0.84	0.67
Proto0	31	<0	499	7.7	−0.82	7.0	2.3	0.78	0.60
Proto0	32	<0	499	7.0	−0.76	6.6	2.5	0.83	0.67
Proto0	34	<0	499	6.5	−0.78	6.2	2.7	0.90	0.62
Proto0	36	<0	499	7.7	−0.84	7.1	2.4	0.79	0.63
Proto0	37	<0	499	6.5	−0.69	5.4	2.5	0.85	0.67
Proto0	38	<0	499	7.7	−0.84	7.1	2.4	0.80	0.59
Proto0	39	<0	499	7.6	−0.91	7.5	2.5	0.84	0.68
Proto0	41	<0	499	7.5	−0.87	7.1	2.5	0.83	0.71
Proto0	42	<0	499	7.0	−0.82	6.8	2.6	0.87	0.63
Proto0	52	<0	499	7.8	−0.82	6.5	2.3	0.77	0.61
Proto0	53	<0	499	10.1	−0.90	8.0	1.9	0.63	2.3
Proto0	54	<0	499	8.0	−0.89	7.4	2.4	0.79	0.59
Proto0	55	<0	499	8.0	−0.85	7.1	2.3	0.77	0.63
Proto0	57	<0	499	7.8	−0.88	7.4	2.4	0.80	0.69
Proto0	58	<0	499	7.6	−0.82	6.4	2.4	0.79	0.65
Proto0	59	<0	499	7.6	−0.79	6.6	2.3	0.77	0.55
Proto0	60	<0	499	7.5	−0.77	6.3	2.3	0.78	0.61
Proto0	61	<0	499	8.0	−0.89	7.4	2.4	0.79	0.60
Proto0	62	<0	499	7.6	−0.80	6.6	2.3	0.78	0.62
Proto0	63	<0	499	8.1	−0.86	7.1	2.3	0.76	0.61
Proto0	64	<0	499	7.9	−0.81	6.9	2.3	0.76	0.66
Proto0	65	<0	499	7.5	−0.80	6.4	2.4	0.80	0.62
Proto0	66	<0	499	8.2	−0.91	7.5	2.3	0.77	0.61

Table 6.1: The coefficients measured on the filtered waveforms needed to evaluate the formula for the threshold upcrossing rate. T is the total duration, σ the standard deviation, k_2 the covariance of the waveform with its second derivative (equation 6.8), $\mathrm{Std}[\bar{k_2}]$ the uncertainty on the value of k_2 determined as the standard deviation of the sample mean of k_2 values across events. The unit "u" is the ADC digit. (`figfakerate.py`)

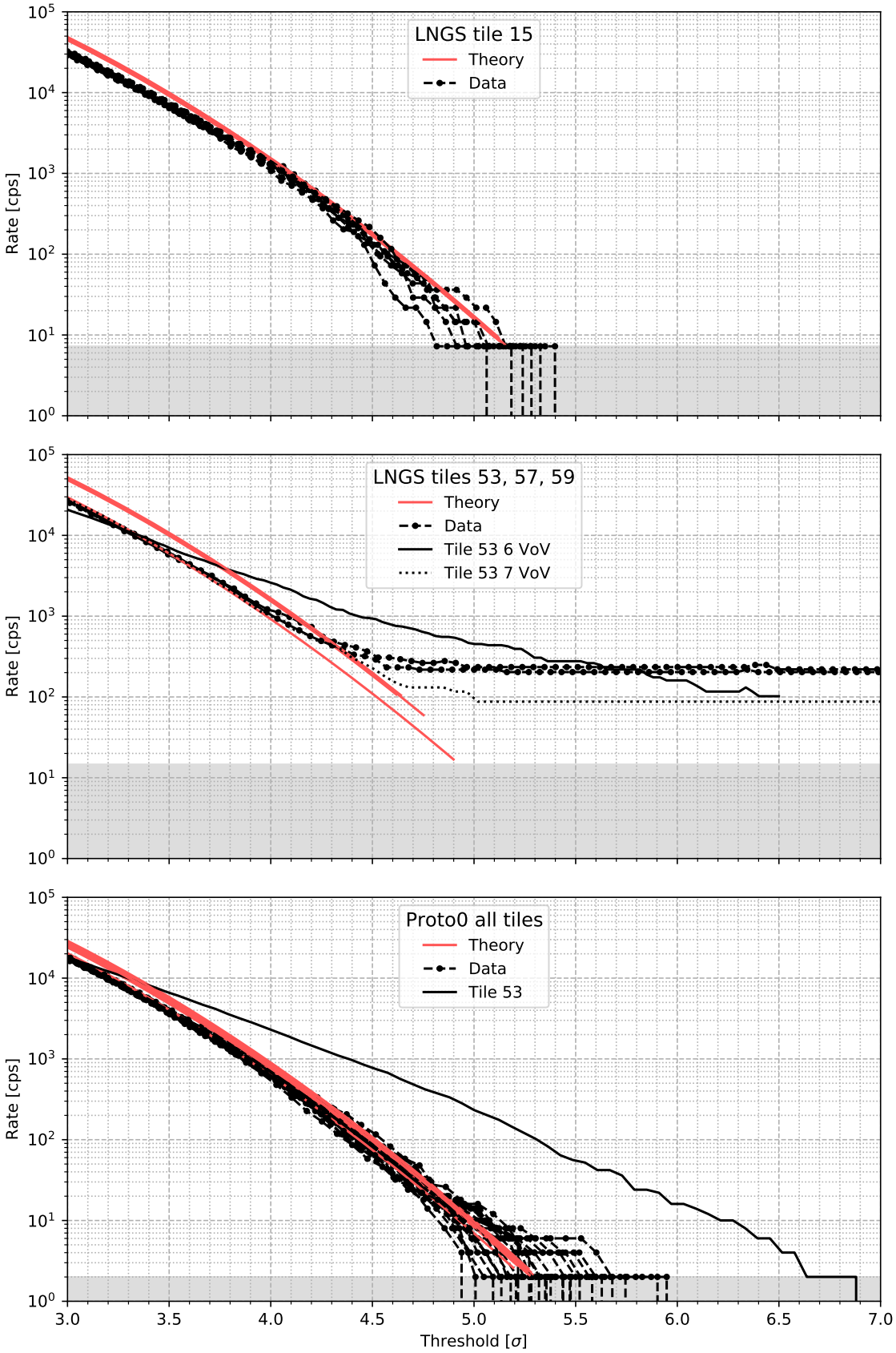

Figure 6.6: The threshold upcrossing rate both counting directly the crossings on data and computing it with formula (6.9). (`figfakerate.py`)

obtained by the covariance at lag 1 c with (6.16). (If the spectrum was obtained from the discrete Fourier transform of a noise waveform, c is the first coefficient after the central one in the autocovariance.)

Chapter 7

Noise analysis

The analysis of DCR, cross talk and afterpulses on LNGS data.

Chapter 8

Conclusions

Summarize the chapters. Maybe copy some results from the report on channel summing.

Bibliography

[Fer15] Isidoro Ferrante. *Elaborazione dei segnali per la fisica*. Pisa University Press, 2015. ISBN: 978-88-6741-548-9.

[Kno10] Glenn F. Knoll. *Radiation detection and measurement*. Third edition. John Wiley & Sons, 2010. ISBN: 0-471-07338-5.

[RK06] Carl Edward Rasmussen and Christopher K.I. Williams. *Gaussian Processes for Machine Learning*. 2006. ISBN: 0-262-18253-X. URL: `http://www.gaussianprocess.org/gpml/`.