All Trapezoidal Filters Are Created Equal

Jaewon Lee JWONLEE@BERKELEY.EDU
Department of Nuclear Engineering

Abstract

Implementing a working trapezoidal filter can be a confusing task due mainly to the recursive formulae given by Jordanov *et al.,* [1]. Furthermore, the confusion can be compounded by the fact that some literature introduce different trapezoidal filtering algorithms. As such, our group has spent an unhealthy amount of time debating which trapezoidal filter actually works. So in this short document, to put an end to our fixation, it is demonstrated that three popular trapezoidal filtering algorithms, namely, the Jordanov [1], Bogovac [2], and Cooper algorithms [3], are mathematically equivalent, up to a constant factor. Also, the three algorithms were implemented in Python and the resulting trapezoids were compared. The results show that these three algorithms indeed produce the same trapezoids, given the same peaking time and gap time parameters. In addition, Python implementation of each algorithm is given.

1. Introduction

1.1 Jordanov formula

Jordanov *et al.,* first reported the following recursive formula, which converts an exponential input signal into a trapezoidal output signal. [1]:

$$v(n) = \begin{cases} 0 & n < 0 \\ e^{-T/\tau} & n \geq 0 \end{cases}, \tag{1}$$

$$d^{k,l}(n) = v(n) - v(n-k) - v(n-l) + v(n-k-l), \tag{2}$$

$$p(n) = p(n-1) + d^{k,l}(n), \tag{3}$$

$$r(n) = p(n) + Md^{k,l}(n), \tag{4}$$

$$s(n) = s(n-1) + r(n), \tag{5}$$

where $v(n)$ is a digitized raw waveform (preamplifier output signal), and $s(n)$ is the resulting trapezoidal output. k, l, and M are trapezoid peaking time, gap time+peaking time (shaping time), and the raw waveform decay time, respectively. T is the digitizer sampling interval and τ is the time constant of the exponential falloff. In Jordanov's first paper [1], M is simply defined to be the decay time constant of the sampled exponential signal. However, it should be noted that in their subsequent paper [4], M is more rigorously defined as $M = \frac{1}{e^{T/\tau}-1}$. Using the first order Taylor expansion, when $T << \tau$, it can be shown that M reduces to τ/T. However, the more rigorous definition of M will play an important role when we show later in the document that all different trapezoidal filter implementations can be expressed in terms of Jordanov's formula.

Jordanov derived the algorithm by observing the output signals obtained from the convolution of the exponential input signal with a simple boxcar filter and a truncated ramp

filter. Although his approach is correct, the derivation lacks an intuitive explanation as the output trapezoidal signal is constructed from a seemingly arbitrary linear combination of the simple filter outputs.

1.2 Bogovac formula

Bogovac, *et al.,* reported their own implementation of the trapezoidal filter in [2]:

$$v(n) = \begin{cases} 0 & n < 0 \\ e^{-T/\tau} & n \geq 0 \end{cases}, \tag{6}$$

$$d^{k,l}(n) = v(n) - v(n-k) - v(n-l) + v(n-k-l), \tag{7}$$

$$r(n) = r(n-1) + d^{k,l}(n) - e^{-\frac{T}{\tau}} d^{k,l}(n-1), \tag{8}$$

$$s(n) = s(n-1) + r(n), \tag{9}$$

where all variables, except $r(n)$, are the same as those in the Jordanov formula.

Although, at first glance, the Bogovac formula seems to differ only little from the Jordanov formula, it offers a much more intuitive understanding of the trapezoidal filtering. As illustrated in Fig 1, the trapezoidal filtering can be understood as a convolution of a unit step signal with two boxcar signals of opposite signs. Hence, Bogovac's approach aims to first convert the exponential input signal into the two boxcar signals, $r(n)$, which is subsequently convolved with a unit step function. The convolution of $r(n)$ with a unit step function simply reduces down to the the integration of $r(n)$, and hence the recursive summation formula is given for the output signal, $s(n)$.

1.3 Cooper formula (Blog formula)

Although the origin of this approach is not clearly known, the following algorithm appears in the document written by Cooper [3] and it was also introduced in the lecture. The same formula is introduced in the blog post by Tang [5], which seems to be circulating among the students. However, the author of the blog post does not cite any sources. The algorithm is given as follows:

$$v(n) = \begin{cases} 0 & n < 0 \\ e^{-T/\tau} & n \geq 0 \end{cases}, \tag{10}$$

$$f(n) = f(n-1) + v(n) - v(n-1)e^{-\frac{T}{\tau}} \tag{11}$$

$$g^{l,m}(n) = f(n) - f(n-l-1) - f(n-l-m) + f(n-2l-m-1), \tag{12}$$

$$s(n) = s(n-1) + g^{k,m}(n), \tag{13}$$

Note that we are following the notation from the blog post [5]. The parameters l and m are defined to be the width of the boxcars and the gap between the boxcars (see Fig 1). In fact, this approach is almost the same as the Bogovac approach in spirit, but the signal is processed in the opposite way; instead of first transforming the input exponential signal into the boxcar signal, it is transformed into the unit step function, $f(n)$, and the boxcar filter is applied subsequently to shape a trapezoid. As the necessity of the pole-zero

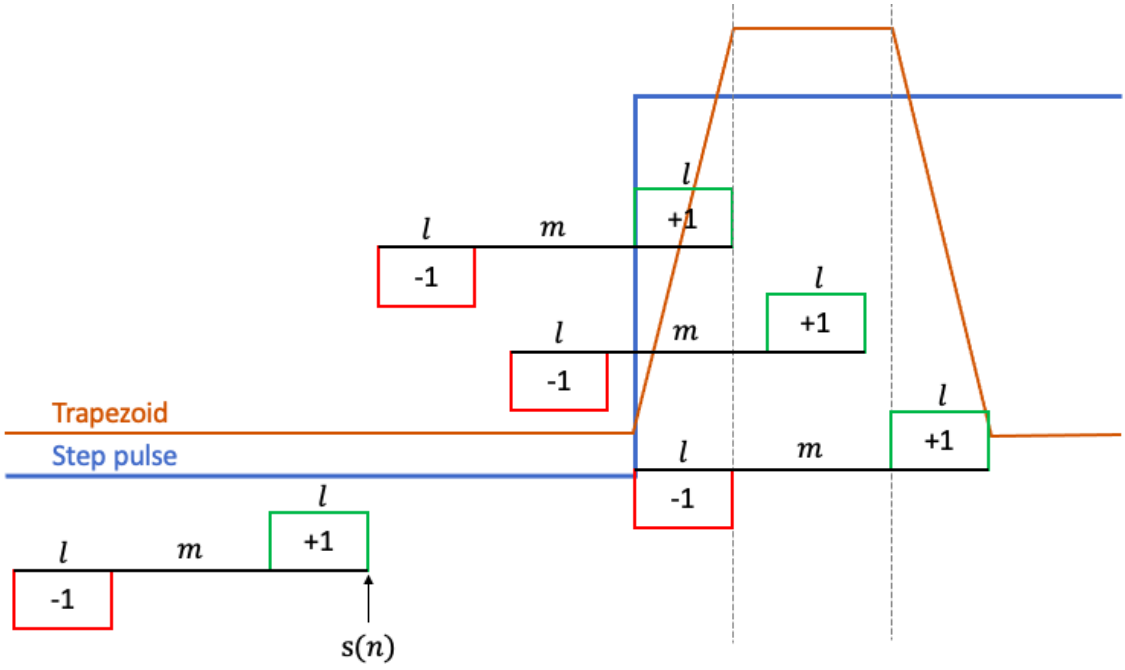

Figure 1: Convolution of a unit step signal with two boxcar filters of opposite signs yields a trapezoidal output. Image from [5].

correction arises from the exponential fall-off of the preamplifier output signal, $f(n)$ can be understood as a "pole-zero corrected" input signal. Subsequently, the convolution of $f(n)$ with the two-boxcar filter is expressed with the recursive formulae for $g^{l,m}(n)$ and $s(n)$.

2. Methods

In this chapter, we show that the Bogovac and Cooper approaches are indeed equivalent to the Jordanov approach, up to a normalization constant.

2.1 From the Bogovac algorithm to the Jordanov algorithm

First, roll out the recursion for $p(n)$ and $r(n)$ from the Jordanov algorithm as folloiws:

$$\begin{aligned} p(n) &= p(n-1) + d^{k,l}(n) \\ &= \sum_{i=0}^{n} d^{k,l}(i) \end{aligned}$$

and,

$$\begin{aligned} r(n) &= p(n) + Md^{k,l}(n) \\ &= \sum_{i=0}^{n} d^{k,l}(i) + Md^{k,l}(n) \end{aligned}$$

Next, we show that $r(n)$ from eq.(8) is the same as the $r(n)$ from the Jordanov approach.

$$\begin{aligned} r(n) &= r(n-1) + d^{k,l}(n) - e^{-\frac{T}{\tau}} d^{k,l}(n-1) \\ &= \sum_{i=0}^{n} d^{k,l}(i) - e^{-\frac{T}{\tau}} \sum_{i=0}^{n-1} d^{k,l}(i) \\ &= (1 - e^{-\frac{T}{\tau}}) \sum_{i=0}^{n} d^{k,l}(i) + e^{-\frac{T}{\tau}} d^{k,l}(n). \end{aligned}$$

Dividing the both sides of the equation by $1 - e^{-\frac{T}{\tau}}$ yields,

$$\begin{aligned} \frac{r(n)}{1 - e^{-\frac{T}{\tau}}} &= \sum_{i=0}^{n} d^{k,l}(i) + \frac{e^{-\frac{T}{\tau}}}{1 - e^{-\frac{T}{\tau}}} d^{k,l}(n) \\ &= \sum_{i=0}^{n} d^{k,l}(i) + \frac{1}{e^{\frac{T}{\tau}} - 1} d^{k,l}(n) \\ &= \sum_{i=0}^{n} d^{k,l}(i) + Md^{k,l}(n) \qquad \left(\because M = \frac{1}{e^{\frac{T}{\tau}} - 1} \quad \text{by Jordanov} \right). \end{aligned}$$

Therefore, the Bogovac $r(n)$ is the same as the Joradanov $r(n)$ up to a constant factor, $1 - e^{-\frac{T}{\tau}}$. The rest of the algorithm (recursion for $s(n)$) is the same for both algorithms, and therefore we can conclude that the Jordanov and Bogovac algorithms are equivalent.

2.2 From the Cooper algorithm to the Jordanov algorithm

To show that the Cooper approach is the same as the Jordanov approach, we first roll out the recursion for $f(n)$,

$$\begin{aligned} f(n) &= f(n-1) + v(n) - v(n-1)e^{-\frac{T}{\tau}} \\ &= \sum_{i=0}^{n} v(i) - e^{-\frac{T}{\tau}} \sum_{i=0}^{n-1} \\ &= (1 - e^{-\frac{T}{\tau}}) \sum_{i=0}^{n} v(i) + e^{-\frac{T}{\tau}} v(n) \\ &= (1 - e^{-\frac{T}{\tau}}) \sum_{i=0}^{n} v(i) + e^{-\frac{T}{\tau}} v(n) \end{aligned}$$

Dividing the equation by $(1 - e^{-\frac{T}{\tau}})$ yields,

$$\begin{aligned} \frac{f(n)}{1 - e^{-\frac{T}{\tau}}} &= \sum_{i=0}^{n} v(i) + \frac{e^{-\frac{T}{\tau}}}{1 - e^{-\frac{T}{\tau}}} v(n) \\ &= \sum_{i=0}^{n} v(i) + \frac{1}{e^{\frac{T}{\tau}} - 1} v(n) \\ &= \sum_{i=0}^{n} v(i) + Mv(n) \qquad \left(\because M = \frac{1}{e^{\frac{T}{\tau}} - 1} \quad \text{by Jordanov} \right). \end{aligned}$$

Therefore, $f(n)$ from the Cooper algorithm is of a similar form to $r(n)$ from the Jordanov approach, except that $d^{k,l}$ in Jordanov's $r(n)$ is replaced with $v(n)$. Next, $g^{l,m}(n)$ can be written as follows:

$$\begin{aligned} \frac{g^{l,m}(n)}{1 - e^{-\frac{T}{\tau}}} &= (f(n) - f(n-l-1) - f(n-l-m) + f(n-2l-m-1))/(1 - e^{-\frac{T}{\tau}}) \\ &= \sum_{i=0}^{n} v(i) - \sum_{i=0}^{n-l-1} v(i) - \sum_{i=0}^{n-l-m} v(i) + \sum_{i=0}^{n-2l-m-1} v(i) \\ &\quad + Mv(n) - Mv(n-l-1) - Mv(n-l-m) + Mv(n-2l-m-1). \end{aligned}$$

At this point, we have to keep in mind that the blog notation uses l as the length of the boxcar filter, where as Jordanov defines l as the rising time of the filter. Note that when l is the length of the boxcar filter, the peaking time of the trapezoids becomes $l + 1$ after convolution. Hence, we can define $l' = l + 1$ as the peaking time of the trapezoid. Similarly,

we can re-define peaking time + gap time to be $k' = m + l$.

$$\begin{aligned}
\frac{g^{k',l'}(n)}{1-e^{-\frac{T}{\tau}}} &= \sum_{i=0}^{n} v(i) - \sum_{i=0}^{n-l'} v(i) - \sum_{i=0}^{n-k'} v(i) + \sum_{i=0}^{n-k'-l'} v(i) \\
&\quad + Mv(n) - Mv(n-l') - Mv(n-k') + Mv(n-k'-l'). \\
&= \sum_{i=0}^{n} d^{k',l'}(i) + Md^{k',l'}(n) \quad (\because d^{k,l}(n) = v(n) - v(n-k) - v(n-l) + v(n-k-l)) \\
&= r(n) \qquad \text{(from the definition of 4)}
\end{aligned}$$

Hence, we conclude that $g^{l,m}(n)$ from the Cooper approach is the same as the $r(n)$ from the Jordanov approach. In this derivation, note that $l' = l+1$ and $k' = m+l$ correspond to k and l in the Jordanov algorithm, respectively. Since the recursion for $s(n)$ is the same for both algorithms, we conclude that the Cooper method is again, equivalent to the Jordanov method wiht the normalization factor $1 - e^{-\frac{T}{\tau}}$

3. Results

In this section, we compared the implemented Jordanov, Bogovac, and Cooper approaches with an example raw waveform. Python codes used for implementation are given as well.

3.1 Comparison of trapezoidal output signals from different algorithms

A raw waveform was used to compare three different trapezoid filtering algorithms. The waveform can be seen in Fig. 2.

Fig. 3 shows three trapezoidal output signals after applying three different trapezoidal filters (Jordanov, Bogovac, and Cooper). Note that the normalization constant $1-e^(-T/\tau)$ was applied to the the trapezoid from the Jordanov algorithm. As can be seen from Fig. 3, all algorithms produce the same trapezoids.

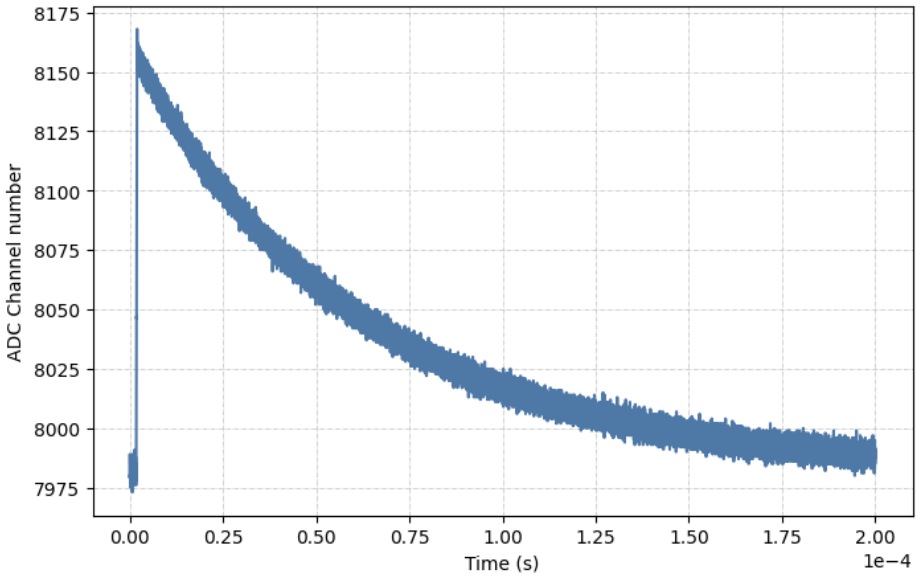

Figure 2: A raw waveform used for the trapezoidal filtering.

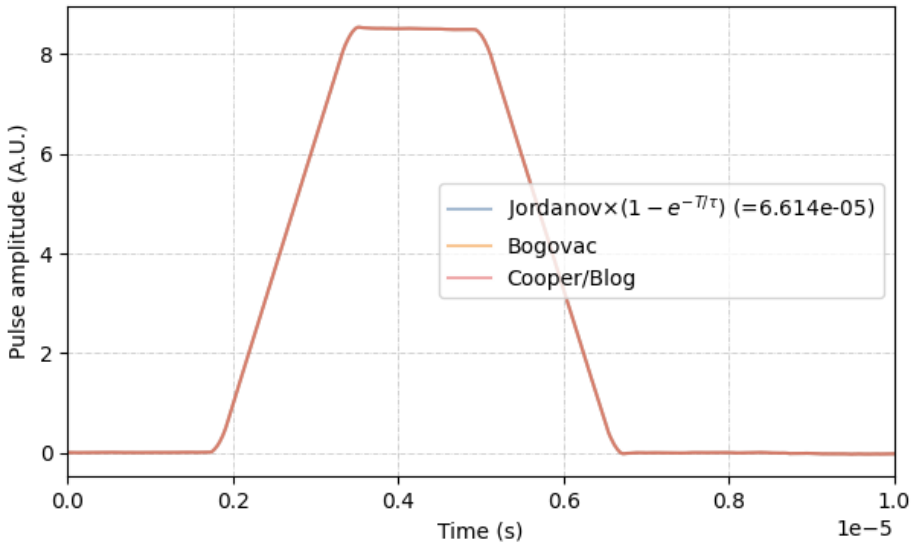

Figure 3: Filtered trapezoidal output signals using three different algorithms. $k = 400$, $l = 800$, and $\tau = 60.5\,\mu s$ are used. All three trapezoidal output signals are essentially identical.

3.2 Python implementation of Jordanov trapezoidal filtering algorithms

Below is the python code for Jordanov trapezoidal filtering algorithm implementation:

```
class TrapezoidalFilter(Filter):
    # We use the notation used in
    #V.T. Jordanoo, G.F. Knoll/Nucl. Instr. and Meth . in Phys. Res. A 345 (1994) 337-345
    def __init__(self, peaking_time, gap_time, decay_time):
        self.peaking_time = peaking_time
        self.gap_time = gap_time
        self.decay_time = decay_time
        self.k = None
        self.l = None
        self.M = None


    def filter_waveform(self, raw_data, sampling_interval=4e-9, normalize=True):
        self.k = int(self.peaking_time / self.sampling_interval)
        self.l = int(self.k + (self.gap_time / self.sampling_interval))
        # self.M = int(self.decay_time / self.sampling_interval)
        self.M = 1 / (np.exp(self.sampling_interval / self.decay_time) - 1)
        d_kl = self.get_d_kl_batch(raw_data, normalize=normalize)
        d_kl[:, 0] = 0  # This makes s(0)=0
        return np.cumsum(np.cumsum(d_kl, axis=1) + d_kl * self.M, axis=1)


    def get_d_kl_batch(self, v, pretrigger_idx=200, normalize=True):
        # Batch normalization
        # TODO it would be good to experiment with different normalization scheme
        if normalize == True:
            # baseline = v[:, :pretrigger_idx].mean()
            # v = (v - baseline) / (v.max() - baseline)
            baseline = v[:, :pretrigger_idx].mean(axis=1, keepdims=True)
            v = (v - baseline) / v.max()

        # padding v by k+l with the noise mean
        noise_mean = v[:, :pretrigger_idx].mean()
        v_padded = np.pad(
            v,
            ((0, 0), (self.k + self.l, 0)),
            mode="constant",
            constant_values=(noise_mean),
        )

        # v_padded = np.pad(v, ((0, 0), (self.k + self.l, 0)), mode="symmetric")

        d_kl = (
            v_padded[:, self.k + self.l :]
            - v_padded[:, self.l : -self.k]
```

```
            - v_padded[:, self.k : -self.l]
            + v_padded[:, : -(self.k + self.l)]
        )
        assert v.shape == d_kl.shape
        return d_kl
```

3.3 Python implementation of Jordanov trapezoidal filtering algorithms

Below is the python code for Bogovac trapezoidal filtering algorithm implementation:

```
class TrapezoidalFilter2(Filter):
    # [1] M. Bogovac, M. Jakšić, D. Wegrzynek, and A. Markowicz,\
    # \Digital pulse processor for ion beam microprobe imaging,"\
    # Nuclear Instruments and Methods in Physics Research Section B:\
    # Beam Interactions with Materials and Atoms, vol. 267, no. 12, \
    # pp. 2073{2076, Jun. 2009, doi: 10.1016/j.nimb.2009.03.033.
    def __init__(self, peaking_time, gap_time, decay_time):
        self.peaking_time = peaking_time
        self.gap_time = gap_time
        self.decay_time = decay_time
        self.k = None
        self.l = None

    def filter_waveform(self, raw_data, sampling_interval=4e-9, normalize=True):
        self.k = int(self.peaking_time / sampling_interval)
        self.l = int(self.k + (self.gap_time / sampling_interval))
        self.M = int(self.decay_time / sampling_interval)
        d_kl = self.get_d_kl_batch(raw_data, normalize=normalize)
        d_kl_cumsum = np.cumsum(d_kl, axis=1)
        r_n = d_kl_cumsum
        r_n[:, 1:] = r_n[:, 1:] - np.exp(-1 / self.M) * (d_kl_cumsum[:, :-1])
        return np.cumsum(r_n, axis=1)

    def get_d_kl_batch(self, v, pretrigger_idx=200, normalize=True):
        if normalize == True:
            baseline = v[:, :pretrigger_idx].mean(axis=1, keepdims=True)
            v = (v - baseline) / v.max()

        # padding v by k+l with the noise mean
        noise_mean = v[:, :pretrigger_idx].mean()
        v_padded = np.pad(
            v,
            ((0, 0), (self.k + self.l, 0)),
            mode="constant",
            constant_values=(noise_mean),
```

```
        )
        d_kl = (
            v_padded[:, self.k + self.l :]
            - v_padded[:, self.l : -self.k]
            - v_padded[:, self.k : -self.l]
            + v_padded[:, : -(self.k + self.l)]
        )
        assert v.shape == d_kl.shape
        return d_kl
```

3.4 Python implementation of Cooper/Blog trapezoidal filtering algorithms

Below is the python code for Bogovac trapezoidal filtering algorithm implementation:

```
class TrapezoidalFilter3(Filter):
    # From Ren Cooper's document and also the online blog post https://nukephysik101.wordpr
    def __init__(self, peaking_time, gap_time, decay_time):
        self.peaking_time = peaking_time
        self.gap_time = gap_time
        self.decay_time = decay_time
        self.k = None
        self.l = None


    def filter_waveform(self, raw_data, sampling_interval=4e-9,  normalize=True):
        self.k = int(self.peaking_time / sampling_interval)
        self.l = int(self.k + (self.gap_time / sampling_interval))
        self.M = int(self.decay_time / sampling_interval)

        Tr_prime = self.get_Tr_prime_batch(raw_data, normalize=True)

        d_kl = self.get_d_kl_batch(Tr_prime, normalize=False)
        return np.cumsum(d_kl, axis=1))

    def get_Tr_prime_batch(self, v, pretrigger_idx=200, normalize=True):
        # Batch normalization
        # TODO it would be good to experiment with different normalization scheme
        if normalize == True:
            baseline = v[:, :pretrigger_idx].mean(axis=1, keepdims=True)
            v = (v - baseline) / v.max()

        # padding v by k+l with the noise mean
        noise_mean = v[:, :pretrigger_idx].mean()
        v_padded = np.pad(
```

```
            v,
            ((0, 0), (1, 0)),
            mode="constant",
            constant_values=(noise_mean),
        )
        Tr_prime = np.cumsum(
            v_padded[:, 1:] - (1 - 1 / self.M) * v_padded[:, :-1], axis=1
        )
        assert v.shape == Tr_prime.shape
        return Tr_prime

    def get_d_kl_batch(self, tr_prime, pretrigger_idx=200, normalize=True):
        v = tr_prime
        if normalize == True:
            baseline = v[:, :pretrigger_idx].mean()
            v = (v - baseline) / (v.max() - baseline)

        # padding v by k+l with the noise mean
        noise_mean = v[:, :pretrigger_idx].mean()
        v_padded = np.pad(
            v,
            ((0, 0), (self.k + self.l, 0)),
            mode="constant",
            constant_values=(noise_mean),
        )
        d_kl = (
            v_padded[:, self.k + self.l :]
            - v_padded[:, self.l : -self.k]
            - v_padded[:, self.k : -self.l]
            + v_padded[:, : -(self.k + self.l)]
        )
        assert v.shape == d_kl.shape
        return d_kl
```

4. Conclusions

All trapezoidal filters are created equal.

5. Reference

1. Jordanov, V. T. & Knoll, G. F. Digital synthesis of pulse shapes in real time for high resolution radiation spectroscopy. en. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **345,** 337–345. ISSN: 0168-9002. https://www.sciencedirect.com/science/article/pii/0168900294910111 (2022) (June 1994).

2. Bogovac, M., Jakšić, M., Wegrzynek, D. & Markowicz, A. Digital pulse processor for ion beam microprobe imaging. en. *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms. Proceedings of the 11th International Conference on Nuclear Microprobe Technology and Applications and the 3rd International Workshop on Proton Beam Writing* **267,** 2073–2076. ISSN: 0168-583X. https://www.sciencedirect.com/science/article/pii/S0168583X09003462 (2022) (June 2009).

3. Cooper, R. J. *Digital Gamma-Ray Spectroscopy: Trapezoidal Filtering* May 2013. https://indico.ictp.it/event/a12178/session/24/contribution/16/material/1/0.pdf.

4. Jordanov, V. T., Knoll, G. F., Huber, A. C. & Pantazis, J. A. Digital techniques for real-time pulse shaping in radiation measurements. en. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **353,** 261–264. ISSN: 0168-9002. https://www.sciencedirect.com/science/article/pii/0168900294916527 (2022) (Dec. 1994).

5. Tang, T. L. *Trapezoid filter revisit* en. May 2022. https://nukephysik101.wordpress.com/2022/05/11/trapezoid-filter-revisit/ (2022).