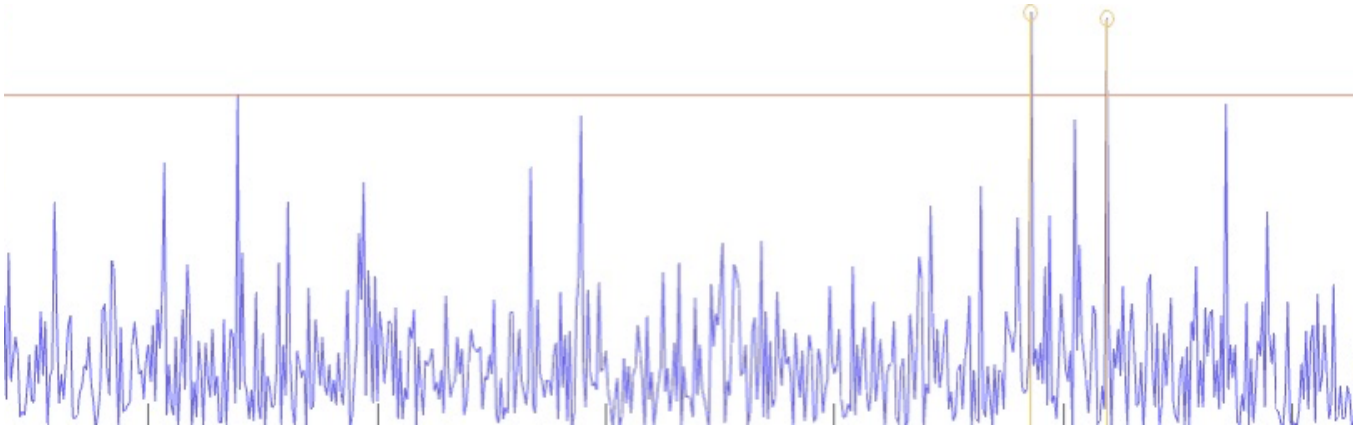
Matched Correlator



Jake Garrison and Jisoo Jung

E E 416, 12/15/2015

Introduction

Matched correlator or matched filter is a useful filter in signal acquisition because it is used to detect an unknown signal by correlating it with a known signal to maximize the signal to noise ratio (SNR). In this project, a matched correlator was implemented using autocorrelation to detect the original signal that was buried in the Gaussian white noise. This project is divided into four tasks. In Task 1, we ran a simulation on a known signal to compare theoretical and observed probability density function (pdf) of a filtered data to check for its normality. In Task 2, we calculated a threshold value for each of the given false alarm probabilities. In addition, for each of the false alarm probability, SNR vs. P_d (probability of detection) was plotted on the same figure. In task 3, three Barker codes were auto-correlated and the sidelobes were analyzed. In task 3.5, longer pseudo-Barker codes were generated using the tensor product. For each generated code, code Identification, number Peak (dB), peak sidelobe (dB) and mean sidelobe (dB) were recorded. In task 4, we extracted multiple signals buried in a White Gaussian Noise using a matched filter and statistical analysis. The simulation was run using Matlab, and the included code is saved in a separate file for each task number.

Task 1

The goal in Task 1 is to derive random variable V that will represent filtered signal in Task 4 and simulate distribution of a random variable V and a distribution of normalized V.

The variables are introduced below:

s = original input signal that will be added with the noise w. Represented as a column vector. Here, 's' is defined as barker 13.

w = Additive Gaussian White Noise (AGWN). It is randomly distributed and its pdf is shows a normal distribution. Represented as a column vector.

$$r = s + w \qquad (1)$$

k = non-zero constant

$$h = ks \qquad (2)$$

V = random variable defined as h and r dot-product

$$V = h^T r \qquad (3)$$

which can be expanded out in terms of s and w:

$$V = (ks)^T (s + w) \qquad (4)$$

v = normalized V by its standard deviation

$$v = V/\sigma_V \qquad (5)$$

For `Nt =1000` trials the random variable `V` is generated with randomly generated Gaussian random noise with `randn` function in Matlab. The constant k is set as 1 to simplify the calculation. Each `V` is stored in a vector `V_RV`, which then is divided by its standard deviation to generate v as defined in eq.5.

```
s = barker(13);
k = 1; % arbitrary scalar
h = k * s;

Nt = 1e3;
V_RV = zeros(Nt, 1);
for i=1:Nt
    w = randn(length(s),1);
    r = s + w;
    V = h' * r;
    V_RV(i) = V;
end
v = V_RV / std(V_RV);
```

v's mean and standard deviation was computed to generate theoretical pdf using `normpdf`. The argument `bin` was computed from the `hist` function.

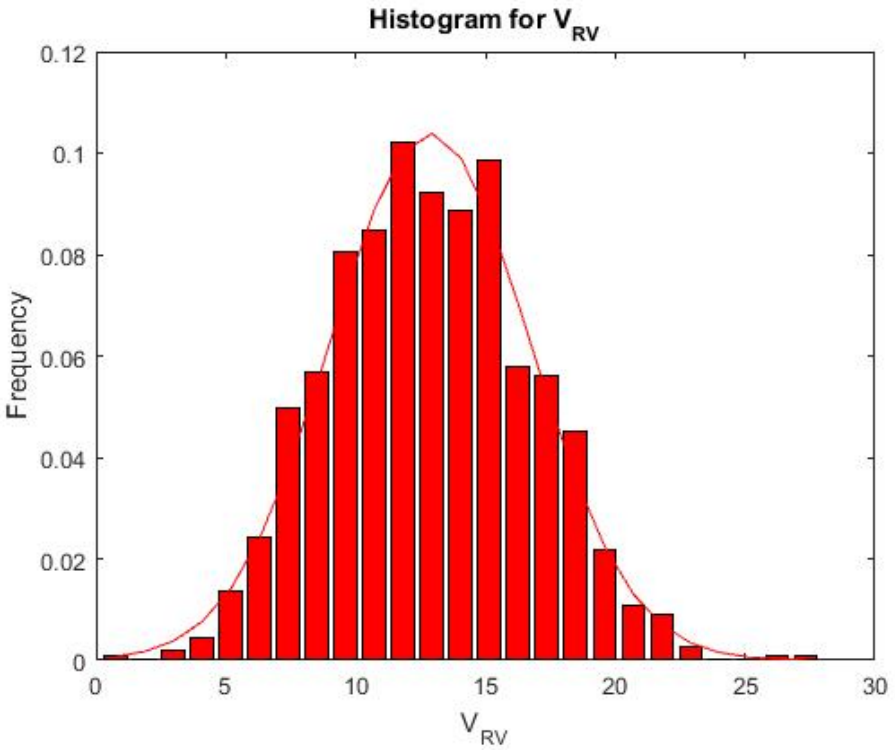


Figure 1: Distribution without standard deviation

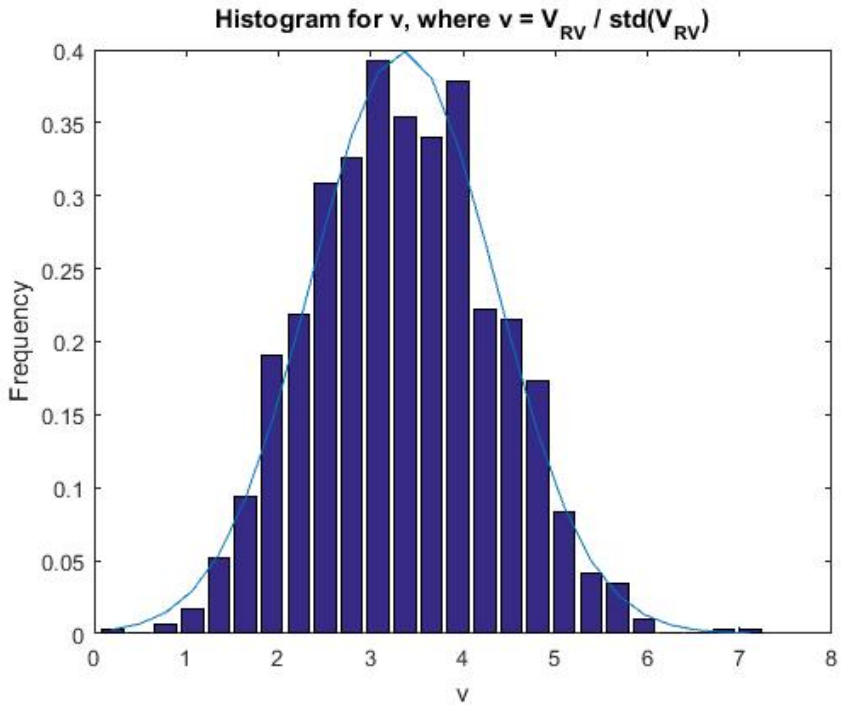


Figure 2: Normalized distribution

The Result of Chi-Squared Test:

```
h = 0
p = 0.2362
chi2stat: 9.2356
df: 7
edges: [1x11 double]
O: [10 24 51 142 216 220 196 93 35 13]
E: [1x10 double]
```

The result from `chi2gof` function shows `h = 0`, which means that it failed we reject the null hypothesis. Therefore, v is normally distributed.

Task 2

In Task 2, we explored how modulating the threshold affects the false alarm probability and subsequently, the probability of detection. As shown in the table below, the threshold increases as the set false alarm probability increases. In order to eliminate false positives, the threshold must be set high. When the threshold is too high, it may fail to detect legitimate signals. The analysis in Task 2 aids the setting of the periodic threshold in Task 4.

Threshold at P_{fa}	
P_{fa}	v_0
0.1	1.281552
0.01	2.326348
0.001	3.090232
0.0001	3.719016

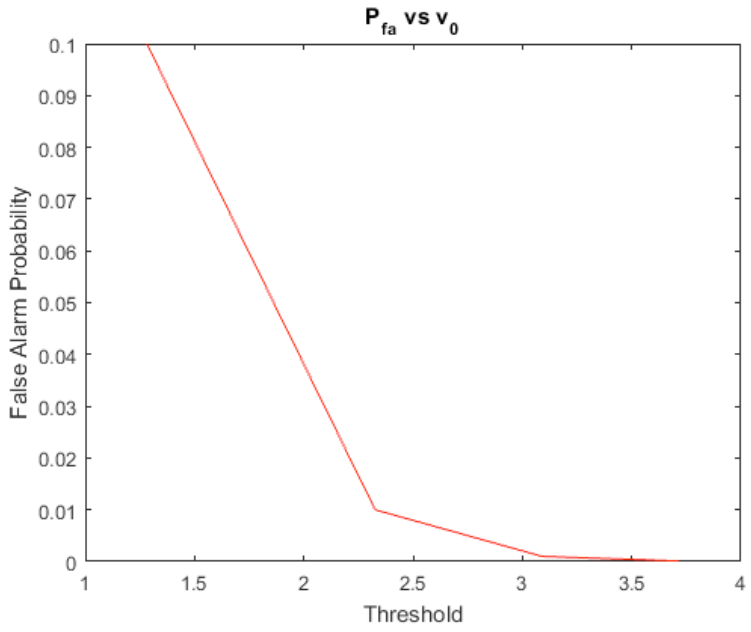


Figure 3: Probability of False Alarm vs Threshold, v0

$$P_{fa} = P\{V > v_0 | H_0\}$$

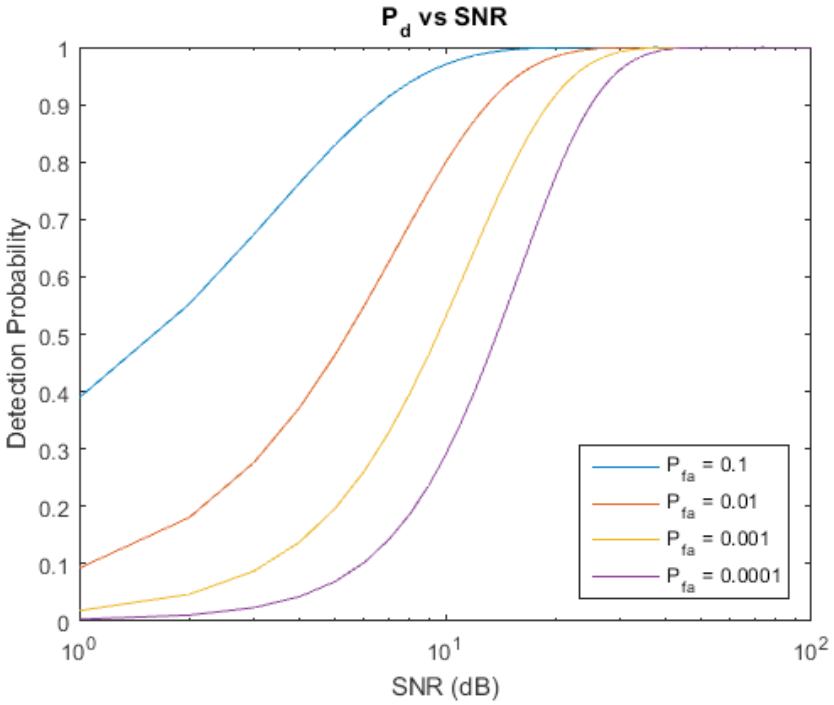


Figure 4: Probability of Detection vs SNR in dB

$$P_d = P\{V > v_0 | H_1\}$$

Figure 4 highlights how P_d is optimized by setting the P_{fa} metrics and desired SNR. Each line shifts right as the P_{fa} is set lower. The reason for this shift is that a larger SNR would be required to differentiate between false alarms and true detections. This is because an outlier noise value could achieve a similar amplitude level to the true signal. With a large SNR, this is more unlikely. To conclude, achieving high detection and low false alarm, is possible when the SNR is relatively large.

Task 3

In Task 3, different codes were analyzed as potential correlator codes. Auto-correlation is used to correlate the signal with itself and the sidelobes were analyzed to judge the effectiveness of the code. The results are presented in the table, and graphically in Figure 5. Note the `autocorr.m` function and `barker.m` function were utilized in the code to make the plotting and correlation process repeatable.

Sidelobe Peak	
Code	**PSL**
1	10
2	1
3	1

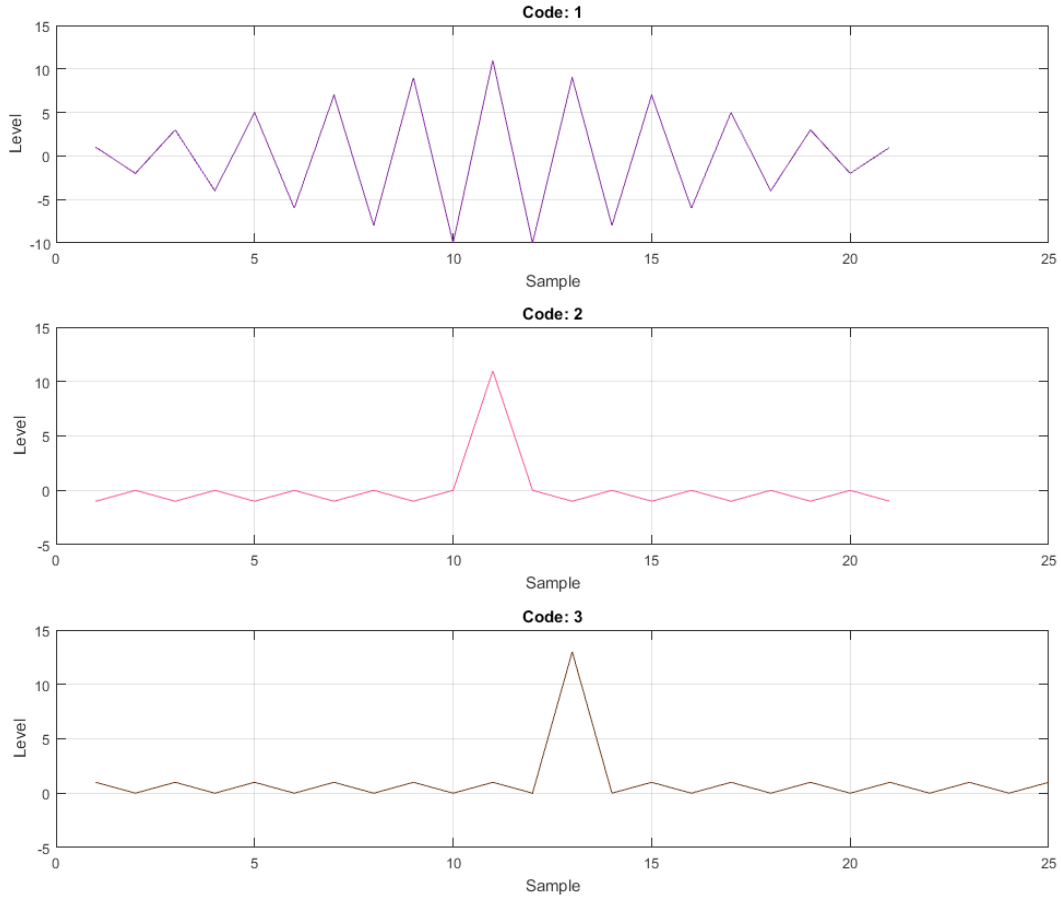


Figure 5: Auto-Correlation for given codes

This analysis proves that Barker codes are ideal for correlation since the peak is significantly higher than the other sidelobes. The distinct peak exposes the signal of interest, even when significant noise is present. Code 1, on the other hand, has sidelobes that are only slightly less than the peak, making it difficult to detect desired signal when noise is present.

Task 3.5

In Task 3.5, we performed a more in depth analysis on autocorrelation codes. Using the tensor product, we created longer pseudo barker codes made up of two combined Barker codes and demonstrating similar characteristics. In certain applications, a code longer than length 13 is necessary and this can be achieved through the tensor product, or `kron` in Matlab. The properties of the generated codes are shown in the table below. All codes were combined with a Barker 13, and have slightly different outcomes. The output of the combination is simply the product of the two lengths, and longer codes outputted larger peaks. There was no observed link between the mean sidelobe versus code length.

Code Properties Table			
Code	**Peak (dB)**	**Peak Sidelobe (dB)**	**Mean Sidelobe (dB)**
2	14.150	11.139	1.703
3	15.911	11.139	0.537
4	17.160	11.139	1.617
5	18.129	11.139	0.969
7	19.590	11.139	1.139
11	21.553	11.139	1.287
13	22.279	11.139	1.326

In many applications, longer barker codes are ideal as they have smaller peak sidelobes and make the desired signal easier to detect. The downside is they can be slower to transmit and process in real time due to the larger size. In addition, issues with longer codes may arise when signals are overlapped, as the longer code may not be able to differentiate between the overlapped signals. Shorter codes, while having larger sidelobes, can be much faster and preferred in real time applications. For most high precision applications, such as radar, longer codes, for example, Barker 11 or 13 are used mainly due to the ideal autocorrelation properties (sidelobe levels).

Task 4

In Task 4, we applied the theoretical information from the previous tasks to our custom dataset with the goal of extracting the time indices of the signal buried in the noise. The block diagram in Figure 6 summarizes the process. The signal of interest interested in detecting was Barker 13, thus it was used in the autocorrelation function.

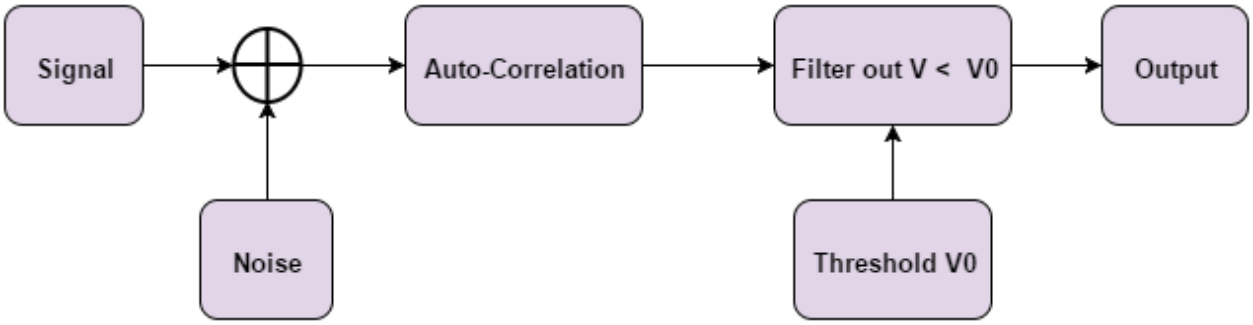


Figure 6: Block Diagram of system

We experimented with two different thresholds to perform the final detection. One was set with the expectation of 10 signals embedded in the noise. After applying the correlation to the received signal , we used the following Matlab code to return a table of the top 10 maximums. These are most likely the epochs of the desired signals.

```
corr_sort = sort(abs(corr));     % set thresh at 10th max
thresh_m = corr_sort(end - 10)
hit_m = find((corr > thresh_m));
```

Note, the `corr` variable is the result of the autocorreclation implemented with the `filter` function, and the `hit_m` is the vector of epochs. While this method is great when the number of signals is known, and in memory, it fails in real time applications. For this reason, we employed a theoretical, dynamic threshold that uses the result from Task 2 to estimate an ideal threshold with minimal P_{fa}. This relies on a relatively high SNR. This theoretical threshold ended up being slightly lower than the optimized one, which introduces some possible false alarms. The code is shown below.

```
thresh_th = v0(4)* sqrt(13*var(w));
hit_th = find((corr > thresh_th));
```

We are essentially scaling the noise variance by the length of the signal (13) and multiplying it by the threshold determined in Task 2. Offsetting based on the mean was also attempted but less successful.

The comparison of the two thresholds is shown in the table below. Notice the theoretical approach detected some additional signals, but can also be employed in real time. In an actual application, this threshold can be much more rigorously tuned.

Correlation Result														
Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Epochs (Theoretical)	27	52	61	193	274	281	287	330	439	589	786	805	819	871
Epochs (Top Ten)	27	52	61	193	274	281	287	330	-	-	786	-	819	-

The plot in Figure 7 shows the effectiveness of the different thresholds graphically, as well as the original received signal for reference. The threshold and detected signals are also outlined in yellow. We initially

found the theoretical threshold derivation difficult, mainly due to lack of direction and insight. After trying different methods and derivations, we settled on the one shown above.

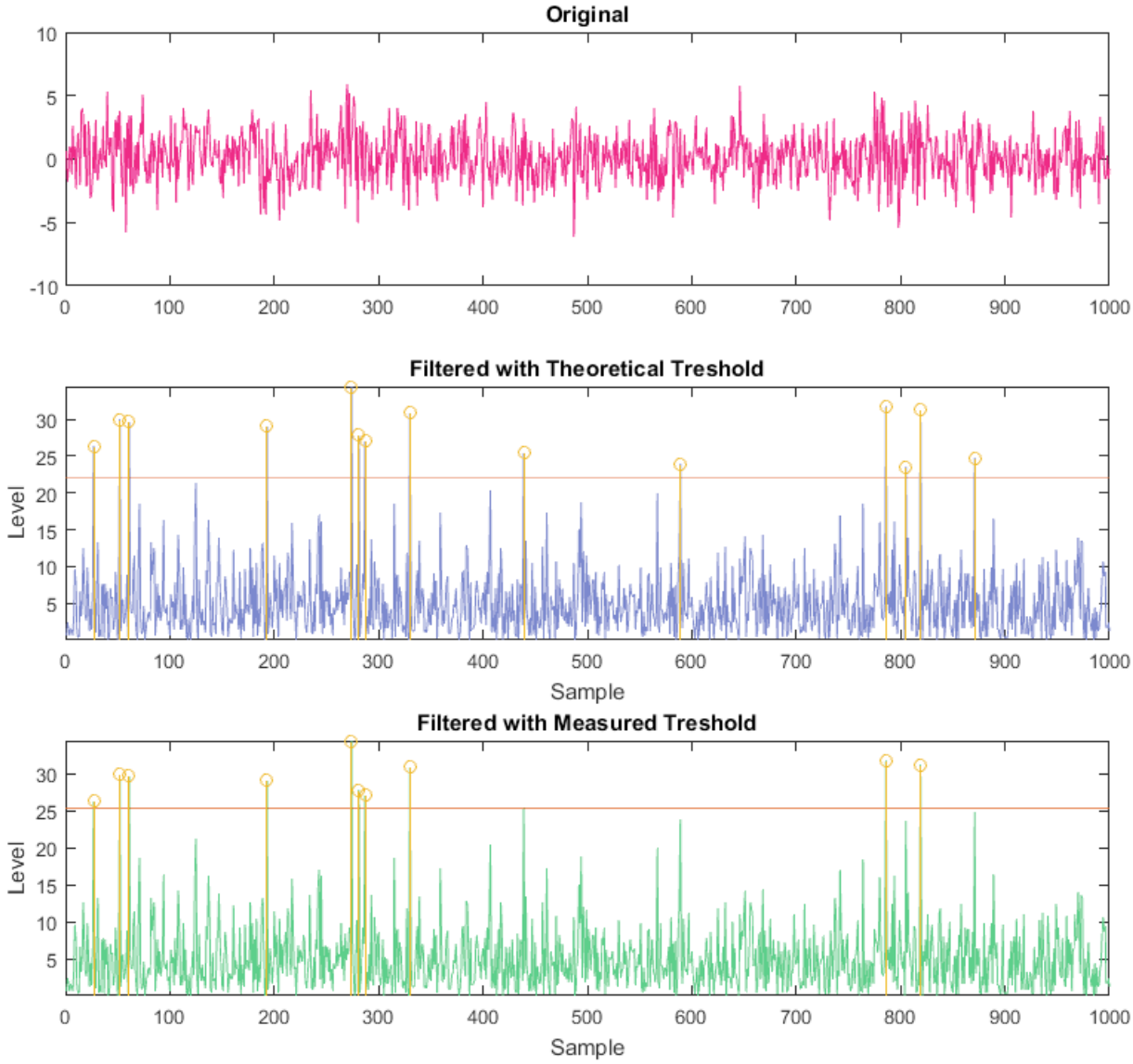


Figure 7: Detected signals at threshold

Conclusion

To conclude, we completed four tasks that led up to the detection of a known signal in Gaussian noise. The first two tasks set up the theoretical framework necessary to derive and optimize the detection method. The third task was aimed to outline the functionality of Barker codes and use cases in practical applications. Finally, in task four, we applied the results from previous tasks to a personalized dataset and identified the desired signal epochs from within the noise. The full Matlab code is attached in the submission, and the key lines of code are briefly discussed in the report.