Algorithm for Detecting Slope Transition Points of a Noisy Signal

Aidan Johnson
johnsj96@uw.edu

Emi Harada
harada25@uw.edu

13 December 2017

EE 416 Final Project Report: Group 4
Random Signals for Communications and Signal Processing
Department of Electrical Engineering
University of Washington

Abstract

To detect the changes in concavity of a noisy signal, we implemented an algorithm that reconstructed the original signal with a matched filtering technique. Our algorithm successfully detected transitions of the signal slope, allowing the digital information encoded in the slope of the signal to be read. When given a random signal with varying amount of noise and varying amount of slopes given, it is difficult to analyze what the original signal was that is in the interest of the receiver. This sort of signal processing is applicable in wireless systems like WiFi, cellular (LTE, CDMA), and in radar communications where it is corrupted by white Gaussian noise (WGN) . Thus, from the fundamentals of analysis, it is crucial to understand the detection of wireless signals in many aspects of signal processing.

Executive Summary

From the result of our detection algorithm, we found the estimated transition points to be at the following time points:

```
[1 177 219 275 319 425 471 549 614 672 699 795 842 963 997 1040]
```

where the total number of transition points were 15, taking into account of all the maxima and minimas taken before the ~1000 time samples. The filtered signal is provided in `filtsignal-n.txt`. Our analysis using signal processing and probability techniques learned in this and previous classes demonstrates our algorithm accurately detects the time-index where changes in noisy signal concavity (e.g. inversions of the signs of binary slopes) occur. Then the algorithm reconstructs the binary slopes of the original, transmitted signal from the received, noisy signal. To accomplish this, we implemented a matched filtering technique that requires the knowledge of the characteristic slope transition feature of a “corner” consisting of slope pairs that are the negatives of each other. The sample length of the signal and the number of segments that have a random positive or negative slope is also required to be known. From the time-indices, the signal can then be reconstructed when applying basic correlation (or convolution) principles. The resulting reconstructed signal is ensured to have randomly alternating $+1$ and -1 slopes such that the digital information sent and transmitted is restored and readable, thus countering the noise born during transmission.

Application of a Noisy Signal

This type of detection problems that involves signals over time can arise in radar, sonar, and communication. For example, transmitting binary communication using pulse amplitude modulation is common. The code of using “1” or “0” to represent the digital pulse that represents the mere existence of echoes happen in radar and sonar systems [1], [2]. By allowing bits to be detected, data streams can be extracted to represent meaningful communication. The additive of white Gaussian noise (AWGN) represents the most realistic approach that can happen in a signal. Since signals is not guaranteed to have exact signal (i.e. clutter in open air, noise in a transmission line) the white Gaussian noise (WGN) that is seen in a signal is a better simulated system than a clear received signal [3]. In communication, RF waves are signals received from the antenna and goes through a demodulator. And after a signal demodulator, the detector can configure the correct relevant parts of the signal [3]. The RF waves used in LTE, CDMA, or even in WiFi, are passed through a series of clutter during transmission and can be distorted with noise [4]. Regardless, the signal assigned to our group and others perhaps best compares to a digital signal as

the slope of the signal corresponds to digital or binary information. In our case, the binary +1 and −1 states information (duration and state) is encoded in the slope of the signal.

Explanation of Random Slope Program Operation

Since the signal assigned to our group was generated using the `randslopes1(1e+03, 20, 0.9, 20, 1, 1)` call, it is important to understand how the signal was generated. In the supporting `randslopes1()`, we debugged each step to find what the program does. See the flowchart below for the explanation of each step in generating the random slopes:

1. Initialise signal length, total percentage of duration, number of slopes, slope values according to the passed variables.
2. Start, duration, ending times are calculated (plus or minus a random number of samples) using for loops.
3. Slopes are randomly generated to be either +1 and −1 (the signs are randomised).
4. At each loop, the number of slope lines are written over in the command window
5. Once the actual original signal is generated, the normalised white Gaussian noise is added to the amplitude of the signal.
6. If the signal has too few transitions, the signal is rejected.
7. If everything is fine, then the program plots the original signal and scaled slopes, as well as the noisy signal graph.

Figure 1: Flow chart detailing the `randslopes1()` operation (numbered according to the above procedure).

The alternating slopes are randomly generated using a Markov Process. They are selected from a table of slopes between 1 and the number of slopes. In the signal assigned to our group, the number slopes is set at two (*Nslopes*). Thus, for randomly varying signs, the slopes will vary randomly between either +1 and −1. In the initial run through the loop that determines segment length and slope, the previous transition (or quasi-transition where the segment ends but the slope signs remain the same) starting slope is assumed to be +1. Given that the old or previous slope is +1, the current slope is `1+floor(2(1-0.3)rand(1,1))`. Therefore, the current slope can only be +1 when $0.7unif[0,1] \geq 0.5$ and −1 when $0.7unif[0,1] < 0.5$. Where **x** is a uniformly distributed random variable on the interval [0,1), the expression on the left of the inequality can be written as 0.7**x** given +1 for the previous slope. The probability density function for a uniform random variable is

$$f(x) = \frac{1}{b-a},\ a \leq x \leq b$$

where $a = 0$ and $b = 1$. The probability that 0.7**x** is greater than or equal to 0.5 is, or when the current slope will be −1 given the previous slope of +1:

$$P(0.7x < 0.5) = P(x < 0.71428) = \int_0^{0.71428} dx = P(+1 \mid -1) = 0.71428$$

And therefore, the probability that the slope will be +1 given the previous slope is −1:

$$P(0.7x \geq 0.5) = 1 - P(x < 0.71428) = 1 - 0.71428 = P(+1 \mid +1) = 0.28572$$

When the previous slope is −1, the current slope is `1+floor(2(0.3+0.7rand(1,1)))`, or +1 when $0.3 + 0.7unif[0, 1] \geq 0.5$ and −1 when $0.3 + 0.7unif[0, 1] < 0.5$. Similarly, solving for the conditional probabilities when the previous slope is −1 for 0.3+0.7**x** we get:

$$P(0.7x + 0.3 < 0.5) = P(x < 0.28572) = P(-1 \mid -1) = 0.28572$$
$$P(0.7x + 0.3 \geq 0.5) = 1 - P(x < 0.28572) = P(+1 \mid -1) = 0.71428$$

From these conditional probabilities, we can then derive the Markov Chain that describes this random slope generation (Figure 2).

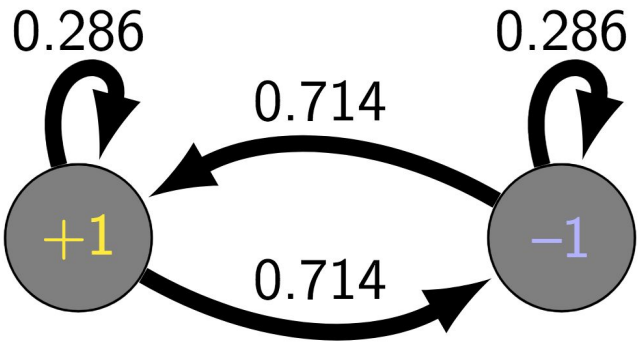

Figure 2: Markov Chain diagram for the two possible slope states their conditional probabilities.

In running a sample simulation, we used the typical call of `randslopes1(1e+03,20,0.9,20,1,1)`. It gave the following output in the command window (Figure 3) during the step 4. The list follows until 20 because that's the number of segments that is taken in from the input. Notice how the *Start* of the second segment starts at 62 and that is right after the first segment's *End*. Since the slopes of the segments will be determined with only +1 and −1, the random slopes are generated between those numbers at each iteration.

```
Slopes =

    -1     1

The duration are drawn from a uniform
 Mean and plus/minus range are here:
    50    45



    el start(el) ending(el) dur(el) slope(el)
    No Start Ending Duration Slope

     1     1    61     61     -1

     2    62   113     52      1
```

Figure 3: Sample command window output of `randslopes1()`.

The Gaussian distribution comes into play because it has a zero-mean, with a variance σ^2 [2]. You can determine this by modeling the signal with random variables and denote the hypothesis with:

$R[n] = s[n] + W[n]$, where $W[n]$ is the stochastic white Gaussian noise and $s[n]$ is the deterministic signal. Since we know that $W[n] = \sigma\mathbf{w}$, where $\mathbf{w}$ is a random variable and normally distributed with a mean of zero and a variance of 1, we can find its expected value (mean) and variance. The scaling performed by σ results in the expected value of $\sigma\mathbf{w}$ to still equal zero, or $E\{\sigma\mathbf{w}\} = \sigma E\{\mathbf{w}\} = 0$. The variance of $\sigma\mathbf{w}$ is $Var\{\sigma\mathbf{w}\} = E\{(\sigma\mathbf{w})^2\} - E\{\sigma\mathbf{w}\}^2 = \sigma E\{\mathbf{w}^2\} = \sigma^2$. This is also a model that has a linear regression that is normally distributed with Data = Fit + Residual form.

Low-Pass Filtering of the Noisy Signal

Seeing that a signal is noisy, a novice student might immediately jump to low-pass filtering the signal. However, in the case of our random-slope signal and the additive white Gaussian noise (AWGN), this technique proves to be ineffective. While we do know the approximate minimum length of a ramp segment in the signal needed for setting an appropriate window for a low-pass, moving average filter, the information contained in the slope of the signal is not adequately isolated. That is, we care to know the slope of our signal in order to extract the binary states encoded in these slopes that are either +1 or −1, in an alternating pattern.

To demonstrate why the moving average filtered, defined by the equation:

$$y[n] = (1/N)(x[n]) + (1/N)(x[n-1]) + \ldots + (1/N)(x[n-(N-1)])$$

where 1/N is the equal weighting defined by the width of the window. The moving average smooths the signal, resulting in more constant slopes, but the slope of the filtered signal still varies randomly. Nonetheless, the binary states are obscured by the remnants of the AWGN. Not only that, the low-pass filter destroys the dynamics of the signal. The transition points are "smeared" because of the inherent lag in responding to changes in state (i.e., concavity). By having a wider moving window, the filtered signal could be even smoother (i.e., removing more noise), but the transition points would be destroyed as the window becomes infinitely large. Figure 4 below shows the result of the moving average low-pass filter.

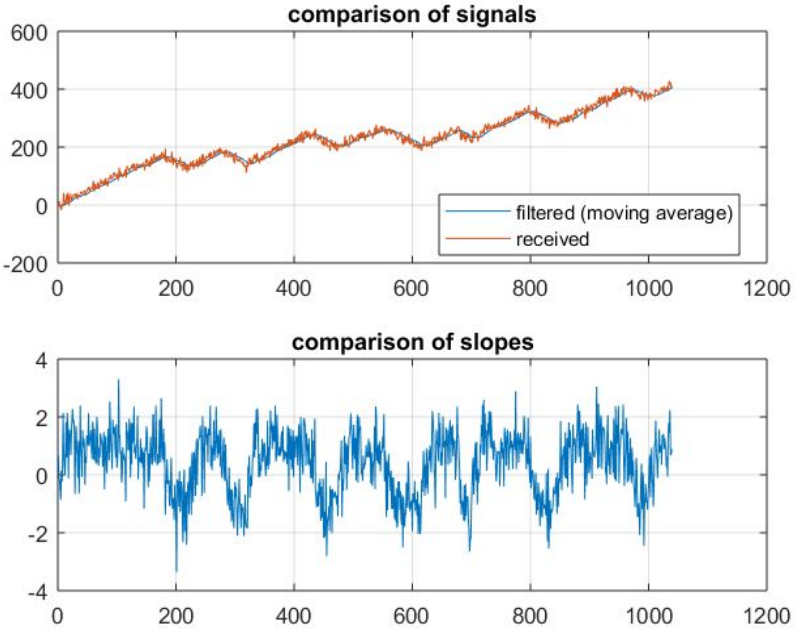

Figure 4: Comparison of received and low-pass filtered signal (amplitude and slope).

Even if, for example, a one-pole IIR autoregressive filter were used in lieu of the moving average filter, the transition points would be not as delayed but the slopes would still be randomly fluctuating. Therefore, we can see that applying a low-pass filter would not be useful. The implementation of a matched filter algorithm becomes the better approach.

Transition Point Detection Algorithm

In this section, we describe the matched filtering algorithm implemented to detect the transition points of our noise-corrupted received signal. The foundation of our algorithm is the matched filtering approach. Matching the impulse function to the transmitted signal, which is then corrupted by AWGN during transmission, maximises the signal-to-noise ratio (SNR) of the output [3].

The output $z(t)$ of a filter defined by the impulse function $h(t)$, whose input is the received signal $r(t)$ or the transmitted signal $s(t)$ plus the white Gaussian noise (WGN) $n(t)$, is:

$$z(t) = \int_0^t r(t)h(t-\tau)d\tau = \int_0^t s(t)h(t-\tau)d\tau + \int_0^t n(t)h(t-\tau)d\tau$$

when the filter's impulse function is $s(-t)$. In other words, when the received signal is correlated with itself, the SNR is maximum. This continuous-time definition can be extrapolated to discrete-time. Next, if we autocorrelate the received signal, we can detect the time of the transition points from the output. In discrete-time, given a symmetric impulse response $h[n]$ and the received signal $r[n]$ for $0 \le n \le L$ (else zero):

$$z[n] = \sum_{k=-\infty}^{\infty} r[k]h[k-n] = \sum_{k=0}^{L} r[k]s[k]$$

where $s[n]$ is the sent signal, or the matched template, if we correlate instead of convolve (note how impulse function is not "flipped" for the filtered signal $z[n]$ [2]. For an illustration of this, consider correlating a rectangular function with itself. When a rectangle is correlated (or even convoluted since the signal is symmetric across the y-axis) with itself, the result is a triangle with its peak at the time-width of the rectangle. Similarly, if we autocorrelate a different signal like our received signal with a predefined template, the maximal overlap (the peak of the correlation) corresponds to when that feature represented by the template is detected.

With our received signal we know some of its qualities. First we know that the signal randomly alternates between positive and negative slopes. Since the magnitude of these slopes are the same, regardless of concavity (concave up minima or concave down maxima), we can describe the corner features with this template. Any change in concavity is characterised by this corner and therefore we can match filter the received signal with this template. With confidence in this principle, we then implemented it in an algorithm for detecting the corners (or transition points) in MATLAB. Second, we know that the length of the signal will be ~1000 samples. Finally, we also know the number of segmented transition points (possibly with no change in concavity) in the signal. In this case, the number of transition points—or points where the random slopes program randomly decides whether to change from positive to negative and vice-versa. Therefore, we can conclude

that the period or width of a corner will be at least 50 samples (give-or-take a few samples due to the random margin caused by the Markov process for the transition decision).

If the template for the match filter is chosen to be a 50 sample wide isosceles triangle ramp with a unit slope (the amplitude is unimportant in the context of finding the corners), in principle, the peak result from the correlation will occur at 25 samples to the right of the transition point. We arrive at this time-sample shift from autocorrelation with the matched filter template. The peak of this correlation occurs at sample 50 [1]. Since the resulting signal in discrete time has a length of 2W+1, where W=50, and where the peak of the matched filter template occurs at sample 25, the location of the correlation peak must be shifted by 25 indices to determine the location of the corner in the input signal. In general, this shift and the location of the peak in the template is n=N/(2T).

Below, in Figure 5, is a plot representation of the template used in our algorithm.

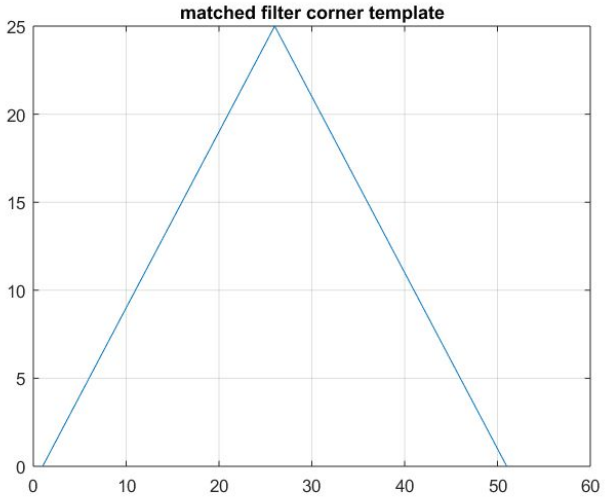

Figure 5: Triangular function used for corner detection.

Since this template is symmetrical across the y-axis, then convolution of the template with the received signal has the same result as the correlation. After the correlation, if we find corners, we can then detect the transition points after applying the index shift. The corners we are trying to detect take two forms: (1) a concave down local maxima or a peak; and (2) a concave up local minima or valley. By definition, a maxima occurs when slope changes from positive to negative, and a minima occurs when the slope changes from negative to positive. The first form corresponds to when the concave down match template correlates with a similarly concave down corner in the received signal. While, the second form corresponds to when the template correlates with an inverted corner (concave up).

In MATLAB, the maximas can be easily detected using the `findpeaks()` function. For the minimas, the output can be reflected across the x-axis to make the valleys into peaks and then the `findpeaks()` function can be again used. However, at this point, we are not in the clear. The correlation results in peaks near the start- and end-points of the signal. These occur not because there is a corner, but because the template just begins to partially overlap the signal. After trial and error, it becomes apparent that these "false positive" ripples caused by noise occur within the first and last 25/2 samples. In Figure 6, the maximas and minimas are indicated before the "weed" out step.

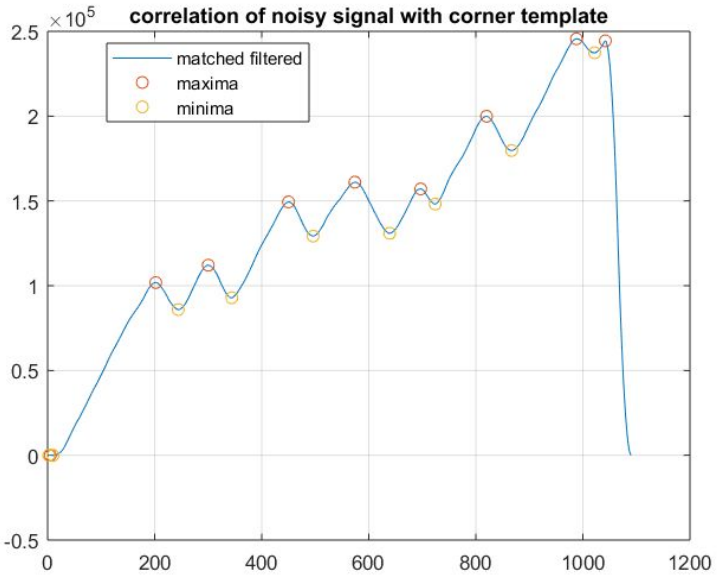

Figure 6: Result of correlation of received signal with corner template in Figure 5. The corners or transition points are marked with circles.

Once the maximas and minimas are detected, and stored in their respective grouping, the index locations can be sorted with `sort()` in increasing order. Since the "weeding" out step using the function `find()` removes indices near the endpoints, they must be manually added to this ranked list of locations at the beginning and end. With these start- and end-points, and transition points, we have all the information needed to reconstruct the transmitted signal, sans noise. We also know which sample locations correspond to concave up and concave down corners so they corners do not have to be reconstructed in an arbitrary alternating pattern. To reconstruct the signal, we loop through the ranked list starting with the first data point at index 1. We find the sample duration of the up or down ramp by computing the difference between the time indices of the i and $i+1$ location. Since we can assume the signal is linear between each transition point, we can reconstruct the signal between the corners, as shown in the comparison of Figure 7.

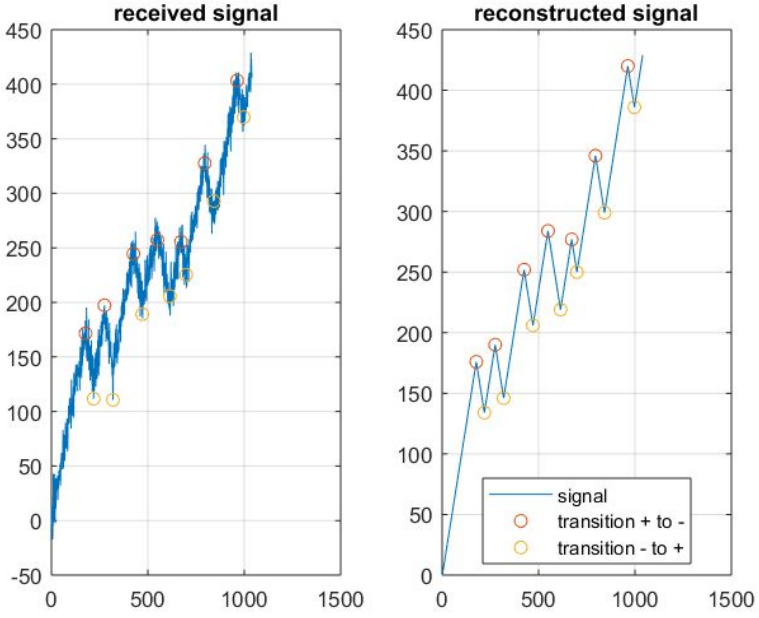

Figure 7: Comparison of the received and reconstructed signal with transition points indicated.

To determine the slope of this ramp segment, we check if the endpoint transition is a concave up or down corner. If it is a minima, then we know the slope preceding it will be −1 and vice-versa. The special case is when the we reach the penultimate time index, where we must check the concavity of the start-point transition to determine the following slope. From the linear "ramp" spaces created in between the transition points, we have effectively reconstructed the transmitted signal. The slopes of these ramp segments are ensured to be +1 and −1, thus ensuring a binary signal can be converted from the now denoised received signal. As can be seen in Figure 8, the reconstructed received signal slope is quite "clean" and digital, thus achieving our goal.

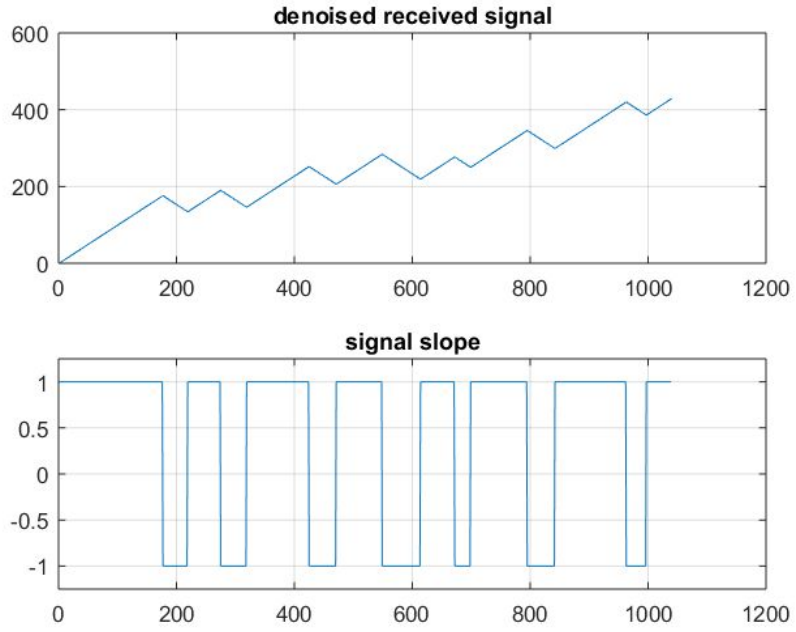

Figure 8: The denoised received (reconstructed) signal with its slope, the information it carries.

Using this method, we find the transition points of the "Group 4" .mat code to be:

```
[1 177 219 275 319 425 471 549 614 672 699 795 842 963 997 1040]
```

These transition points can be found in the ASCII text file `detect-n.txt`. The reconstructed, filtered signal can be found in `filtsignal-n.txt`. Based on our analysis, we believe this to be an accurate result for the transition points. However, there are weakness of this approach. For example, we make assumptions that the input received signal a predictable but random segment duration. While it allows us to detect the corners, it limits the robustness of the algorithm when applied to signals that behave differently. Second, even though it is convenient to discard any potential false positives peaks near the beginning and end of the signal, this could potentially discard actual peaks. This might occur if the initial sampling point is not a transition point but actually a point somewhere in a segment. The algorithm also assumes that there are only two possible slope states. Since the signal denoised by this algorithm was generated with the `randslopes1()` program, the number of slope magnitudes could be more than one. Despite these weaknesses, the algorithm performs well for the given assumptions and information characterising the transmitted and received signal. Lastly, for reference and convenience, the code for the algorithm (in the attached .m file) is included in the Appendix.

References

[1] "Matched filter," *Wikipedia*. 21-Nov-2017. [Online]. Available: https://en.wikipedia.org/wiki/Matched_filter. [Accessed: 11 Dec. 2017].

[2] Alan V. Oppenheim and George C. Verghese, "Chapter 14: Signal Detection," in *Class Notes for 6.011: Introduction to Communication, Control and Signal Processing Spring 2010*, Massachusetts Institute of Technology, 2010. [Online]. Available: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-011-introduction-to-communication-control-and-signal-processing-spring-2010/readings/MIT6_011S10_notes.pdf. [Accessed: 11 Dec. 2017].

[3] John Proakis, *Digital Communications*, 4th ed. New York: McGrawHill, 2000. [Online]. Available: https://archive.org/details/DigitalCommunicationsByJohnProakis4thEdition. [Accessed: 11 Dec. 2017].

[4] K. Vasudevan, *Digital Communications and Signal Processing*, ver. 3.1. Kanpur, India: Indian Institute of Technology, 2017. [Online]. Available: http://home.iitk.ac.in/~vasu/book0.pdf. [Accessed: 11 Dec. 2017].

Appendix: MATLAB Algorithm Code

The code for the transition point detection algorithm attached below for reference. The .m file is also included in the same directory of this report.

```
% loads given test file (receives random transmitted signal)
Rx = load('SGroup4');
T = Rx.SimData.Nsigs; % number of signal transitions/segments
N = Rx.SimData.Nsim; % total signal length
r = Rx.SimData.rcvd; % received signal
f = filter(ones(1,T)/T, 1, r); % received signal filter with moving average

% template: symmetric triangular ramp up then down (each ramp length n)
n = N/(2*T); % duration of ramp with slope of +/- 1
mf = [linspace(0,n,n+1), linspace(n-1,0,n)];
% match filters received signal with template  (convolves ~ correlate)
y_mf = conv(r,mf);

% finds maxima/minima of convolution result (concave down/up)
[maxs,t_max] = findpeaks(y_mf);
[mins,t_min] = findpeaks(-1*y_mf);

% weeds out false positive detected transitions; endpoints (initial and
% final transitions) added manually
i_max = find(t_max > n/2 & t_max < length(r)-n/2);
i_min = find(t_min > n/2 & t_min < length(r)-n/2);

% shifts concave down (maxima) and up (minima) corner locations to
% time scale of signal
dwncrnrs = t_max(i_max(:)) - n;
upcrnrs = t_min(i_min(:)) - n;

% combines and sorts transition time locations with the initial and final
% times manually assumed to be transition points
tcorners = sort([1; dwncrnrs; upcrnrs; length(r)]);
```

```
% reconstructs transmitted signal from the corners
z = zeros(length(r),1);
for i = 1:length(tcorners)-1
      % assumes linear line between each corner (maxima/minima) location
      t1 = tcorners(i);
      t2 = tcorners(i+1);

      t = t2-t1+1; % number of sample points for line connecting corners
      zi = []; % values of signal in between corners

      % final corner (penultimate with respect to the final data point)
      if i == length(tcorners)-1
      if ismember(t1,upcrnrs)
            zi = z(t1) + linspace(0,t-1,t)*1;
      elseif ismember(tcorners(i),dwncrnrs)
            zi = z(t1) + linspace(0,t-1,t)*-1;
      end
      elseif ismember(t2,upcrnrs) % a concave up transition
      zi = z(t1) + linspace(0,t-1,t)*-1;
      elseif ismember(t2,dwncrnrs) % a concave down transition
      zi = z(t1) + linspace(0,t-1,t)*1;
      end

      % reconstructs signal with corner connector
      z(t1+1:t2) = zi(2:end);
end

% outputs detection points (corners or concavity transistions)
% to ASCII text file
dlmwrite('detect-n.txt',tcorners);

% plots template matched filtered received signal with locations of corners
figure;
plot(y_mf);
hold on;
scatter(t_max,y_mf(t_max));
hold on;
scatter(t_min,y_mf(t_min));
hold off;
grid on;
title('correlation of noisy signal with corner template');
legend('matched filtered','maxima','minima','Location','best');

% plots detected corners/transitions locations overlaid with received
signal
figure;
subplot(1,2,1);
plot(r);
hold on;
scatter(dwncrnrs,r(dwncrnrs));
hold on;
scatter(upcrnrs,r(upcrnrs));
hold off;
grid on;
title('received signal');

% plots transition locations overlaid with transmitted signal
subplot(1,2,2);
```

```
plot(z);
hold on;
scatter(dwncrnrs,z(dwncrnrs));
hold on;
scatter(upcrnrs,z(upcrnrs));
hold off;
grid on;
title('reconstructed signal');
legend('signal','transition + to -','transition - to +','Location','best');

% plots reconstructed and trasmitted signal for comparison
figure;
subplot(2,1,1);
plot(z);
hold on;
plot(f);
hold off;
grid on;
title('comparison of signals');
legend('reconstructed','filtered (moving average)','Location','best');

% plots slopes (the useful information of the signal) of plot above
dzdt = diff(z);
dfdt = diff(f);
subplot(2,1,2);
plot(dzdt);
hold on;
plot(dfdt);
hold off;
grid on;
title('comparison of slopes');

% plots the reconstructed signal z (the received signal matched filtered to
% remove noise)
figure;
subplot(2,1,1);
plot(z);
grid on;
title('denoised received signal');
subplot(2,1,2);
plot(dzdt);
grid on;
title('signal slope');
ylim(1.25*[min(dzdt), max(dzdt)]);
```