Department of Electrical and Computer Systems Engineering
Monash University

Information and Networks, ECE3141

Lab 6: Rectangular Pulses

Authors: Dr. Mike Biggar
Dr. Gayathri Kongara
(updated 5 May 2022)

1. Introduction

Over the next three laboratory classes (6-8), we will explore the types of pulses that are transmitted over telecommunication systems and the methods by which they can be used to modulate a carrier signal. The obvious thing to do if thinking about transmitting digital data is just to send one fixed level (say, +1 V) during the time interval of a digital "1", and a different one (say, -1 V) during the time interval of a digital "0". In this lab, we will explore the characteristics of such a rectangular pulse signal, and in particular how it behaves in the frequency domain. We will then be prepared to investigate other types of pulses with characteristics better suited to band-limited transmission systems in Lab 7.

The core Matlab m-file *Rectangular_bit_sequence.m* contains the basic functionality you will need for this lab. While you do not have to write the code from scratch, you will need to edit this m-file and change parameters in it, so it is important that you understand what it is doing and how it does it. This m-file will perform the following operations:

- Allow a sequence of 1 or 0 bits to be generated in different ways
- Copy those bits to a larger array, with an integer number of sample points (determined by the variable `SampPerSymbol`) representing each input bit. (That is, it generates rectangular pulses.)
- Takes the FFT of the array, allows it to be restricted by a "brick-wall" filter, and plots the spectral magnitude. This simulates the band-limiting filter that would be applied at a transmitter before transmission.
- Allows noise to be added to the resultant signal, simulating noise added in transmitter or receiver circuitry or over the transmission path by the time the signal is to be decoded.
- Applies a second filter (identical to the first), simulating that which would be applied at the input to the receiver, to reject signals other than those in the bandwidth of the required signal.
- Applies a simple threshold level detection in the middle of each pulse, so we can see how bit errors can result from the combination of pulse smoothing, inter-symbol interference and noise.

You will progressively turn these functions on as you work through this laboratory investigating the effect of passing a sequence of rectangular pulses through a band-limited, noisy system and, as a result:

- Understand the frequency domain characteristics of a rectangular pulse and a sequence of rectangular pulses
- Observe the distortion that results from band-limiting such a sequence of pulses

- See what the resultant signal would be at a receiver if this sequence of pulses was subject to both band-limiting filters and AWGN (Additive White Gaussian Noise).
- Investigate the effect of the above processes on a simple detecting system that will decide whether a 0 or a 1 was received, and how often errors occur.
- Develop your skills in Matlab coding and processing

You should copy selected Matlab graphs into this document to help you recall what is happening and to help with later revision.

1.1 Background

You are doing this lab just at the time we are commencing to look at the Physical Layer, so you are not drawing on lecture theory to complete it. In fact, we are using this lab to establish some insights that we will use in the lectures to develop concepts of pulse shaping. Though we're not assuming much theoretical background, the following is important:

- **Frequency domain representations.** (See pre-lab below.) You need to be comfortable about the concepts of representing signals in either the time or frequency domain. We use a "normalised" definition of frequency here, which is explained in the box in Section 4.
- **Sampling rates.** You need to be familiar with Nyquist sampling rates, as we discussed early in the semester.
- **AWGN.** The concept of "Additive White Gaussian Noise" will be discussed in lectures in the coming weeks, but we use it in this lab. It is a model for random noise that can distort a signal transmission. The important thing to be aware of for this lab is that, viewed in the frequency domain, such random noise has energy at ALL frequencies. So, even if we filter some frequency components out (high pass filter, low pass filter, etc.) we cannot eliminate the effect of such noise completely. We will look at the theoretical interpretation of random noise in lectures.

2. Pre-lab

Prior to the laboratory class, you must read through this entire laboratory description and complete the pre-lab online quiz.

You should also brush up or familiarise yourself with the concept of frequency domain representation of signals. If you have previously covered Fourier Transforms and frequency domain signal representations more thoroughly in other units, this would be a good time to review that material. If you have not studied this topic before, then as a minimum make sure you read through the brief "Intro to Frequency Domain" pdf document on Moodle (under "Week 1").

You should also look through the Matlab code of the m-file that you are given for this Lab, make sure you understand the operations that are carried out and be ready to modify the code as necessary.

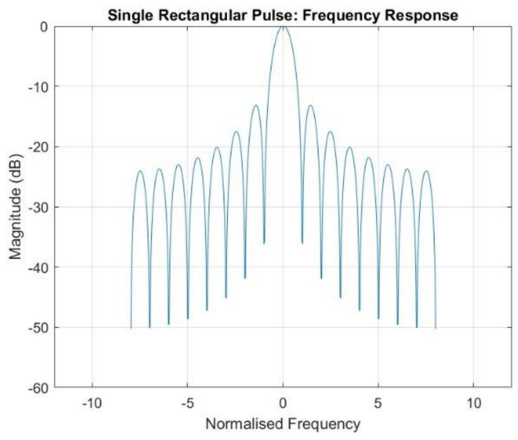
3. Frequency domain representation of a rectangular pulse

The Matlab m-file *Rectangular_bit_sequence.m* contains the basic functionality you will need for this lab. When you download it, it will be set up to carry out the tasks below[1]. Much of the code is initially commented out, but we will use those components later on. As you start running the code and generating graphs, it would be a good idea to size and arrange them as you want at the side of the screen; then, every time you run the code, they will simply be updated; there is no need to keep closing them all between runs.

In this Section, don't worry too much about the magnitudes and graph scales. We'll get to them later. Here, we're only concerned with the basic shapes and behaviours of the signals.

a) The bit pattern generated by the code is determined by the value of the `BitPattern` variable at line 24. (It should initially be set to 3.) When you run the code, you will see a plot of a rectangular pulse (set by parameters to be 16 samples wide in an array of 1024)[2]. There is also a frequency domain plot of this array, where the frequency domain magnitude is plotted in dB.

We know that the Fourier Transform of a rectangular pulse is a sinc function[3], but the frequency domain plot does not look like a "sinc". Why? (Hint: check what the code is doing at line 61.)



This is because the frequency domain plot is plotted on a logarithmic scale. Thus, the frequency domain plot does not look like a "sinc" function. We can also say that the frequency in dB is obtained by normalizing the maximum magnitude in the frequency domain of the pulse. That's why we obtain a graph that looks inverted (negative values of the frequency domain).

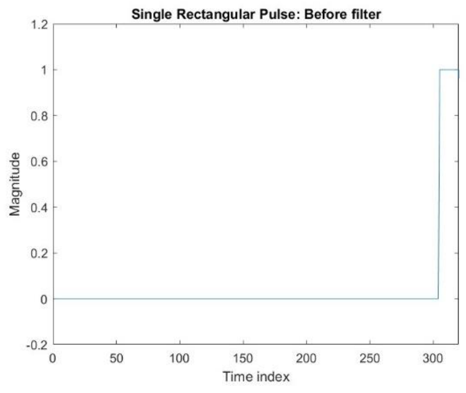
[1] It would be a good idea to make a copy of the downloaded m-file. Open both in Matlab, but edit and run only one of them while you keep the other unchanged. This will ensure that, even if you change some of the code you are running, you know which bit of code is being referred to when we direct you to specific code line numbers in the sections to follow.

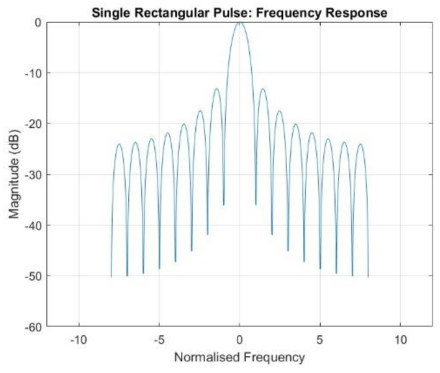
[2] As the code is, the whole array is not plotted. To make detail more visible, only the first few pulse-widths are shown (determined by the parameter `DisplayBits`). Of course, you can change the code if you want.

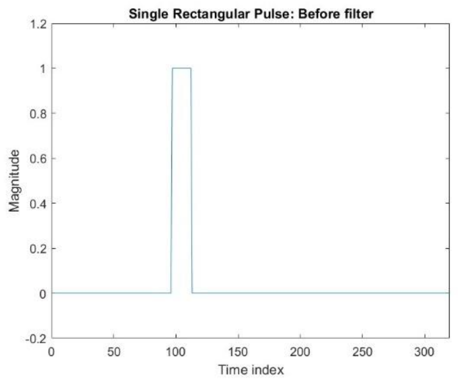
[3] If this is new to you, have a look at http://www.thefouriertransform.com/pairs/box.php

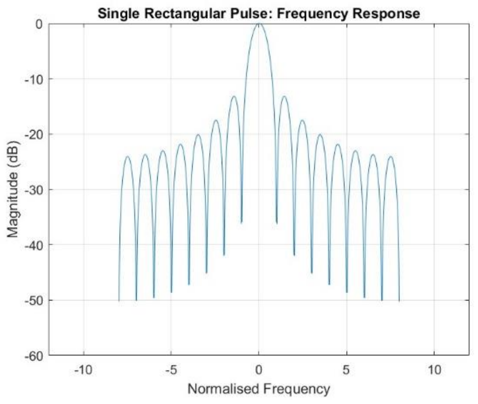
b) Lines 25-35 in the m-file include three different ways of defining the input pulses. Edit line 33 to place the rectangular pulse (the "1") to a different position in time. Run the code again to confirm that the pulse has been moved, and look at the new spectral plot. (Remember that, in the state you download it, the script only plots the first `DisplayBits` pulse positions.)

If you change the position of the single pulse in the array, does it change the frequency domain plot? Why?





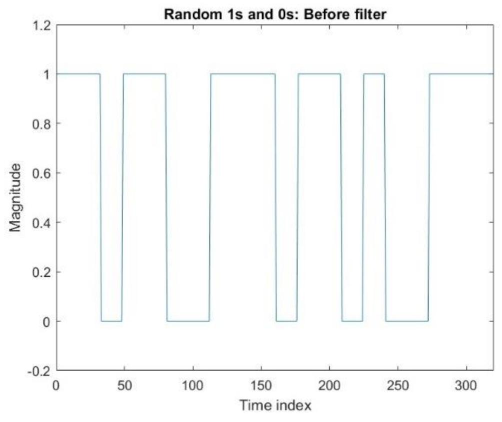


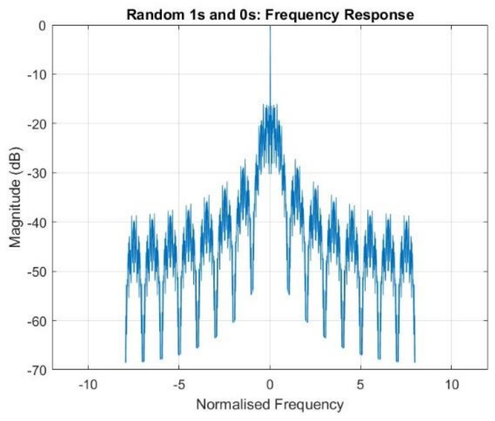


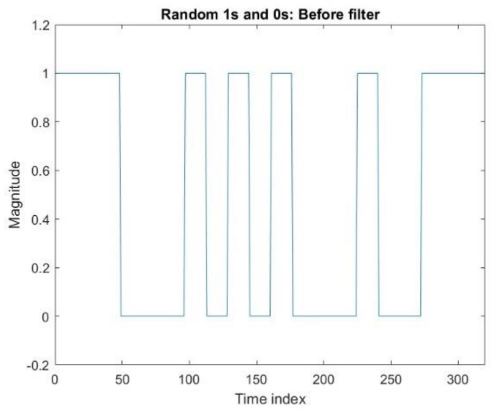
When I change the position of the single pulse in the array, it does not change the frequency domain plot. This is because the frequency will not be affected by the location or position of the pulse but will only be affected by the period of the cycle. We can also that we have normalized the magnitude of the frequency domain to the centre frequency, causing the plot of the frequency domain to remain the same.

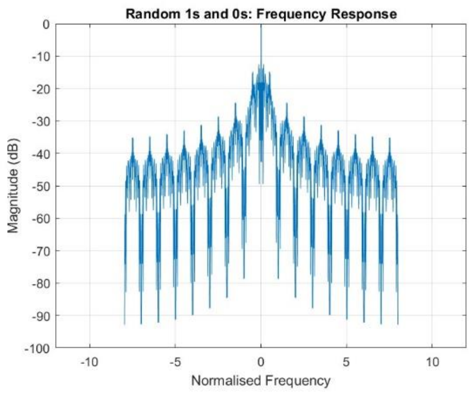
c) Set BitPattern = 1 so we are generating a random sequence of bits instead of just an isolated “1”. Run the code and have a look at the resulting spectral magnitude.

The spectrum looks a bit messier than it did for a single pulse. Why? What happens if you run the code again? Is the spectrum identical?









The spectrum will not be identical and looks messier when I run the code again. However, we can see that the overall structural shape of the graph looks similar.

A different time domain signal will be produced every time we run the code due to the randomised sequence of bits. The period of the impulse is also changing based on the sequence of bits for each case. So, the frequency domain spectrum will be different every time depending on the period of the impulse.

4. Applying a "brick wall" filter[4]

The code that you ran above actually applies a low pass filter to the transformed data array, but the cut-off frequency was initially set high, so it includes all frequencies[5]. As a result, the time domain pulse shape is not (yet) changed by the filter. You can confirm this by uncommenting lines 71-79 of the code and running it again. Now we see the time domain representation of the (first few) input pulses after transformation into the frequency domain and then back again. We see that they are identical to those at the input to the fft.

Normalised frequency

As you are aware, signals in the time domain have features and durations measured in seconds, while in the frequency domain our units are cycles/sec (or Hz), or perhaps radians/sec[6]. Thus, a sine wave of period 5 ms has a frequency of $\frac{1}{5x10^{-3}} = 200$ Hz, and a sine wave of frequency 2.7 GHz has a period of $\frac{1}{2.7x10^{9}} \approx 3.7x10^{-10}$ s or 0.37 ns.

In signal processing, we often want to illustrate concepts and draw conclusions in a generic way, independent of the specific frequency involved. We don't want to redo the calculations and replot the graphs for every possible scale of time or frequency. So, we try to "normalise" relationships and plots, which we can easily scale appropriately if we want to apply them to a specific time scale or frequency.

Therefore, when doing pulse width and frequency analysis we commonly make the pulse width T=1 unit and we use a normalised frequency scale of "cycles/sample".

Thus, the spectrum of a pulse that is considered to be 1 time unit wide (even if we're representing it with 16 sample values in the default Matlab code) will have the first zero at frequency $\frac{1}{T} = 1$ in the frequency domain. (You will have observed this in the first spectrum plot you made.) In terms of array index locations, if T = 1 corresponds to 16 samples, then the first zero in the frequency domain is at 1/16 x 1024 = 64. That is, if we use an array of 1024 for our time samples and Fourier transform, then the first zero in the array that contains the frequency domain data will be at index number 64, but it represents f=1 in normalised frequency.

What happens if we make our T=1 pulse 4 samples wide instead of 16? In the transform domain, the first zero is still at f=1, but now that point will be at index number ¼ x 1024 = 256.

And if we want to convert from "normalised" measures to real measures of time and frequency? Let's use an example. Let's make T = 2 ms. Then the zero points in the spectrum will occur at multiples of 1/T = 500 Hz. That is, at 500 Hz, 1000 Hz, 1500 Hz,….

If you're still confused by this, maybe you would like to experiment a bit in Section 6 below. For instance, you could change the pulse duration to be an actual time in seconds and change the frequency scale to Hz.

[4] A "brick wall" filter is one that has a gain (output/input) of exactly 1 within the passband and exactly zero outside it. Thus, there is an instantaneous drop-off at the cut-off frequency (i.e. the edge is like a brick wall). In reality, such an ideal filter function cannot be realised; it can only ever be approximated, but it is a very simple function that can help us understand important principles.

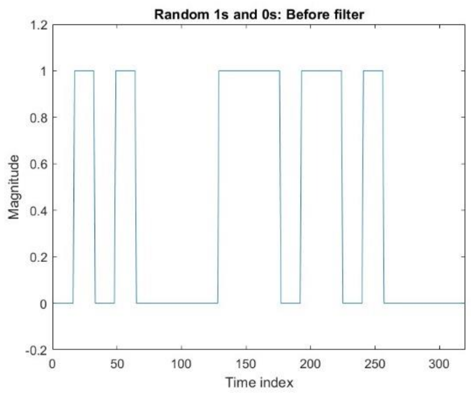
[5] If you look at the value of the variable `LPFcutoff` in the Workspace (right side of your Matlab Window), you'll see that it currently has a value of 512. The filter would remove (or zero) any frequencies more than 512 samples away from the zero frequency (array index 1) but, because of the two-sided frequency representation, that ±512 range includes the entire frequency array, so the LPF has no effect.

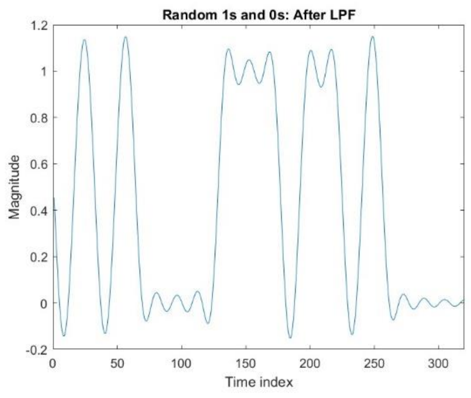
[6] Commonly we use f to represent frequencies in Hz, and ω for radians/sec, where $\omega = 2\pi f$.

So, let’s apply a filter to see what would happen if our pulse sequence was passed through a low pass filter. This is not a random choice of something to do, but it reflects the reality that very many signals must be restricted to a limited frequency range. This might be because the channel will not pass certain frequencies (a channel cannot pass an infinite range of frequencies, but also it might not be able to pass very low frequencies down to zero, for example, because some circuit components might not operate at those frequencies). It is also very common that we need to share our transmission channel by dividing the frequency spectrum up and allocating it to different users[7] (and possibly only when they need to use it). So, we need to restrict the frequency range that we generate to make sure that our signal doesn’t interfere with others sharing the transmission medium. “Leakage” of our signal beyond the bounds of our allocated frequency range is called “Out Of Band” (OOB) emission, and should always be minimised.

a) Select a smaller value for the cut off frequency (set `FilterWidth=1` at line 14 of the m-file) and run it again. You will see that with this setting we preserve only the central lobe of the spectral plot.

How does the time domain plot after filtering compare with the input? Can you explain why it has the appearance it has?





We can see a sinusoid graph in the time domain plot after filtering. It has the appearance it has as the low pass filter will only pass signals with the frequency that is lower than the cut-off frequency. This means that it will filter out the signals with high frequencies. We can see that there will be less sharpness of the graph.

If this was the signal received at the far end of a transmission link, do you think it would be any more difficult to detect at the receiver whether a 1 or a 0 was transmitted during each pulse period?

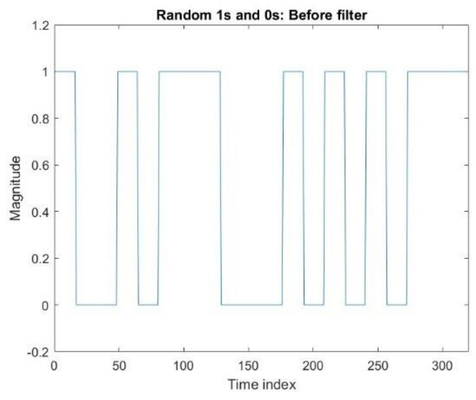
Yes, it would be more difficult to detect at the receiver whether a 1 or 0 was transmitted during each pulse period as the higher frequencies are filtered out by the low pass filter. The higher frequency would very likely contain the details of the original signal.

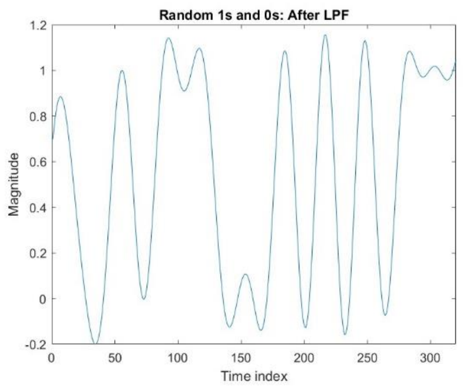
[7] This is very common and applies in a wide range of applications. Obvious examples are the frequencies allocated to radio channels (e.g. the AM radio station “ABC Radio Melbourne” is allocated 765 – 783 kHz, an 18 kHz range of frequencies centred on 774 kHz) or mobile phones for the duration of each cellular telephone call. On DSL broadband internet connections over telephone wire pairs, the low frequencies are reserved for analogue telephony (where it still exists) while the remainder must be split between upstream and downstream data transmission. Even with optical fibres, different wavelengths (which correspond to different frequencies) of light are used to support separate data transmissions in parallel over the one fibre.

b) Now let's filter the signal even more[8]. Repeat the above with `FilterWidth=0.6` and then `FilterWidth=0.4`.

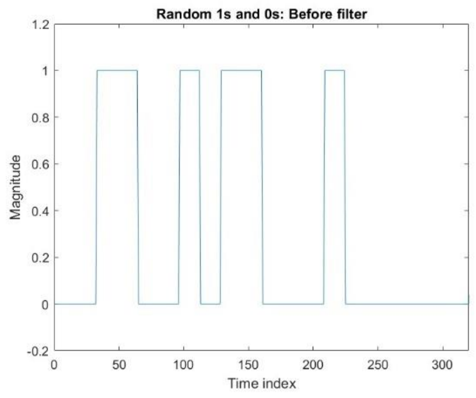
How do the filtered pulses appear now? Could we still reliably decode a received signal if it had been filtered in this way?

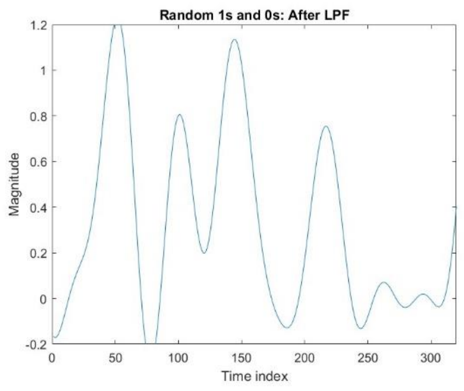
FilterWidth = 0.6





FilterWidth = 0.4





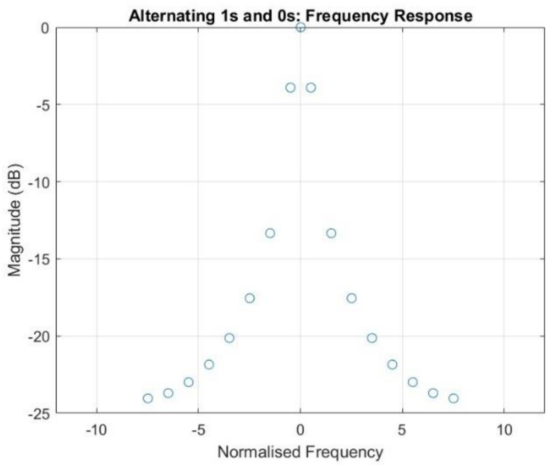
The filtered pulses look less similar with the input pulses. We could still reliably decode the received filtered signal, but it would be very likely that we would **not** obtain an accurate detail of the signal (due to the loss of information) and it would be too tedious to decode the signal.

[8] The way the code is written, a new random bit sequence is generated each time you run it. If you're trying to compare behaviour for the same type of bit sequence using different filters, you might have to run the code a few times to see a similar combination of bits again. Alternatively, you could modify the code to use the same bit sequence each time. If you do this, just be aware that later in these lab instructions, we will be referring to specific line numbers in the original code.

c) Now edit the bit generation part of the code. Set BitPattern = 2. Now we are generating bits that are simply alternating 1 0 1 0 1 0…. Before we filter this signal, let's see what it looks like in the frequency domain without filtering. Set `FilterWidth=8` and run the script. The inverse transformed signal looks exactly the same as the input, but the spectrum looks empty! This is because we only have isolated non-zero values, so we do not get a plotted line. We can mark the plotted points by including '-o' in the plot command, so that line 64 should read:

```
plot(((frequencies)/(2*pi*Sample_rate)),Frequency_response_in_dB1,'-o')
```

Why is this spectrum so different from those we met earlier? (Hint: think about the frequency components present in a regular 10101010… pattern compared with those in a sequence of random bits.)

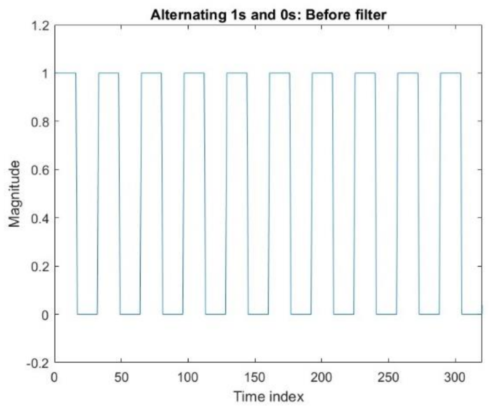


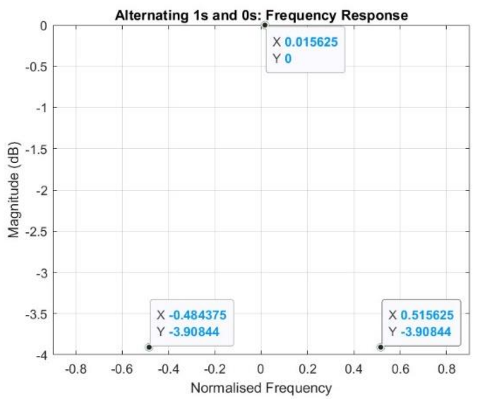
The spectrum is different from those we met earlier as the input pulse is periodic. The fundamental frequencies will be captured in the frequency domain plot. There will also be many different frequencies to be captured.

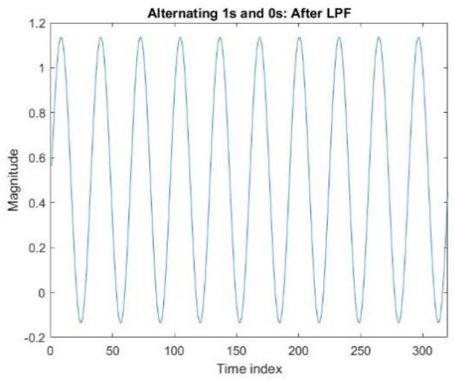
d) Repeat the test with the two filter widths you used in part b) above.

For the ideal square wave signal we are generating (10101010…), what is the significance of a (normalised) frequency of 0.5? What do you expect to happen if we filter out all frequencies above 0.5?

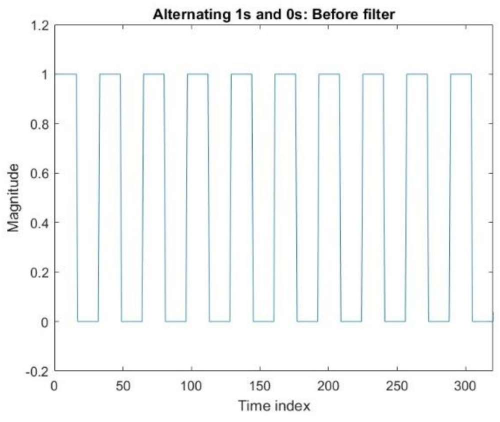
Filter Width = 0.6

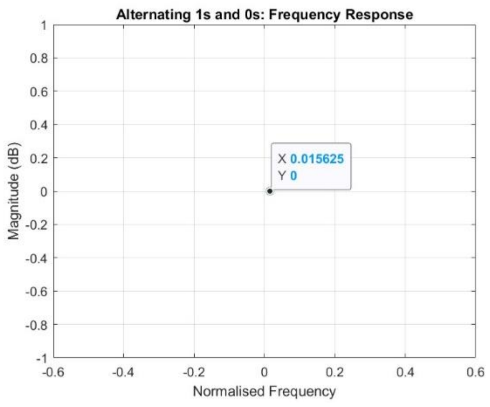


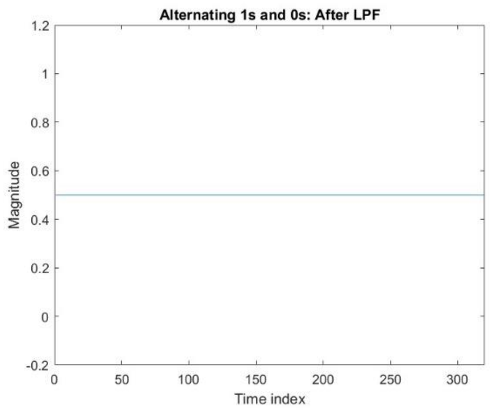




Filter Width = 0.4







When I filter out all frequencies above 0.5, we would obtain a straight line.

What do you observe and how do you explain it?

When the value of the filter width is changed to 0.6, we would obtain a sinusoidal plot (one oscillating waveform). When the filter width is 0.4, we would obtain just a straight line, meaning that there is no output signal.

This observation could be due to the fact that some or most of the fundamental frequencies has been filtered out by the low pass filter. This means that there will be fewer values that can be used to reconstruct the signal.

Considering the case for `FilterWidth=0.4`, why is the result here so dramatically different compared with that for randomly generated bits that you looked at in part b) even though the width of the pulses is the same?

When the value of the variable FilterWidth is 0.4, most of the fundamental frequencies have been filtered out (only 1 left). Hence, we would only obtain a straight line as the signal is reconstructed based on that one value of fundamental frequency.

5. Adding noise and decoding our signal

We will now complete our simple model of transmission over a bandlimited channel (see Figure 1). In a real system, we have to expect that our received signal will contain some random noise. We would also apply a second filter at the receiver. This is because the noise that has been added to the signal includes frequencies outside the bandwidth limits of our signal. (Recall the information from Section 1.1; AWGN includes energy at ALL frequencies.) We certainly don't want to have more signal distortion than necessary when we try to decide whether we received a 0 or a 1, so we filter out those frequencies that we know we didn't transmit, giving us the best chance of making a correct decision[9].

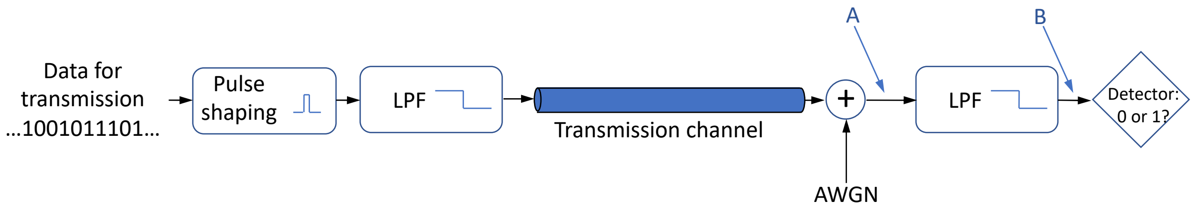
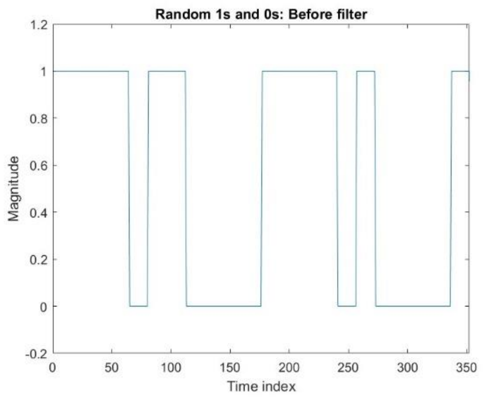
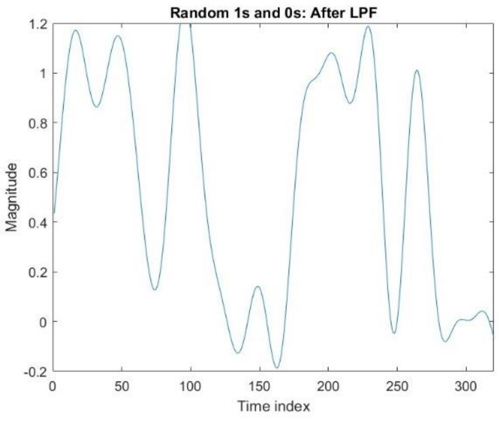


Figure 1. Simple transmission system being modelled.

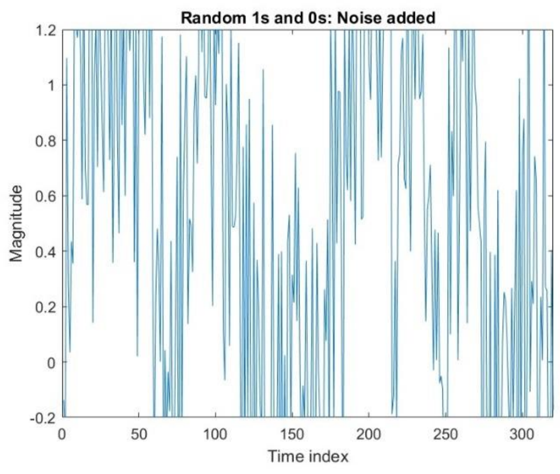
a) We can see the value of the receiver filter if we uncomment and run a bit more code in the m-file. First, return the code to the state where it was generating random data (`BitPattern`=1) and we are filtering just above the Nyquist limit (set `FilterWidth=0.6`). Uncomment lines 81-89. This extra code adds some noise and plots yet another graph; that of the signal at point A in Figure 1. The level of noise is determined by the SNR parameter at line 15. You can leave it set at 5 dB for now. Run the code and have a look at the additional plot.

What features do you see in the plot with AWGN included? Apart from the noise, this plot should be similar to that you produced in Section 3b, but how easy do you think it would be to detect whether the original data was a 0 or a 1?





[9] In fact we will see in lectures that there is an optimum filter (called a "matched" filter), which is dependent on the pulse shape we are expecting. This is not, in general, just a duplicate of the filter that is at the transmitting end.



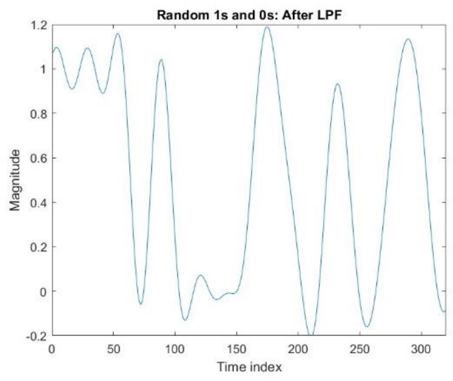
We can see that a messy graph is produced. It would not be possible to detect whether the original data was a 0 or 1.

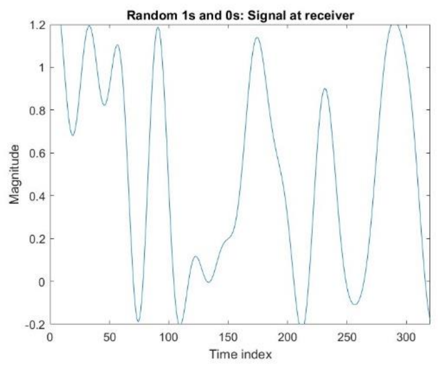
Can you make any observations about the likely frequency content of the added noise, and whether it might be different to that of the signal we want?

There will be many high frequencies as the plot consists or the original signal and the noise signal input. Due to the addition of higher frequency noise, the signal will have a large number of high frequency oscillations. There will be a difference between the signal with noise and signal without noise. There will be problems in decoding the signal with the added noise.

b) Let's see if we can eliminate some of that noise by applying our LPF a second time. This should not affect our signal (we have filtered it once, so we know it contains only frequencies below f=0.6), but it will eliminate the noise that is outside our band limit, and therefore improve our chances of reliably detecting the original data value. Uncomment lines 92-106 of the m-file and run it again. We get yet another plot, this time of the signal at point B in Figure 1.

How does matlab figure 5 (after noise addition and the second LPF) compare with matlab figure 3 (output of first LPF)?





The MATLAB figure 5 and MATLAB figure 3 is very similar to each other. The noise which lies outside the bandwidth is filtered out, so both graphs are similar to each other. There could still be inaccuracies caused by the lower frequency noise at the AWGN, leading to decoding error at the receiver.

a) Finally, we will attempt to decode our bit sequence and check for errors. That's what the last bit of code (lines 108-119) does[10], so uncomment this now and run the script again. With the parameters we have set, and such a small number of bits in the sequence (64), it is very unlikely that you will see any bit errors. However, if you decrease the SNR to 2 dB, you should see some appear.

Can you relate the errors to the appearance of the received signal after noise addition and filtering? What has happened to the signal that has caused the threshold detection to give errors?

Yes, we can relate the errors to the appearance of the received signal after noise addition and filtering. The errors are more likely to occur as the threshold detection is too low. Whenever there is a spike in the noise, it will be counted as a '1' instead of an error. When the SNR is high, the ratio of the signal to noise will be higher. The noise magnitude will not have much effect on the signal power. Thus, there will be fewer bit errors. When SNR is 2, the ratio of the signal to noise will be lower. There will be more distortions in the signal , resulting in inaccuracies in the signal.

6. Now it's your turn!

With the time remaining in this lab exercise, devise at least one of your own experiments to investigate the behaviour of either a part of this system (e.g. filtering of rectangular pulses) or the whole system (that is, the ultimate effect on errored bits). Do not just randomly change parameters, but consider what you want to learn, what should be controlled/fixed and what you should vary and observe or measure. Describe your experiment, observations and conclusion below.

We can investigate how SNR affects the transmission system.
The controlled variable would be SNR. The fixed variable would be the filter width. In this case, the fixed variable will be fixed to have a value of 0.6.

When the SNR is set to have a value of 10, the MATLAB figure 3 and MATLAB figure 5 are quite similar to each other. If we set the SNR to have a value of 1, MATLAB figure 3 and MATLAB figure 5 will be different from each other.

From here, we can conclude that the SNR must be maintained at a high level for us to decode the signal more accurately.

[10] This is a very simple bit level detector. Using just one sample value near the middle of each pulse, it applies a threshold of 0.5. If above this value, a 1 is assumed to have been transmitted and, if below, a 0. The output bit sequence is then compared with the input and the number of errors is counted and displayed.

7. Conclusion

We have introduced some important concepts about telecommunication systems in this lab. Signals must be shaped or filtered to ensure that they pass through band-limited channels. The signal will be distorted by the time it arrives at the receiver and it will have noise added to it. While we can filter again to remove some of the noise, we will still have a non-zero probability of error.

As well as this, though, we have seen that a simple, rectangular pulse shape (in which each data value is mapped to a constant output voltage throughout the pulse period) is not well-suited to band-limited systems. The sharp edges are lost after passing through a band-limited channel, and the smoothed pulses overlap and interfere with one another, making detection difficult at the receiver and increasingly vulnerable to noise.

These ideas of pulse shapes, filtering and noise addition will provide a good basis for lectures on pulse transmission. In Lab 7, we will explore this further and learn about pulse shapes that give us the maximum bit rate and reliability of detection in band-limited systems.

Before finishing, could each student please click on the "Feedback" icon for this laboratory on Moodle and provide some brief feedback on this laboratory exercise (all inputs are anonymous).

Finally, please submit this completed report via Moodle by the stated deadline. In so doing, please be aware of the following:

- Even if you have had a mark assigned to you during the lab session, this mark will not be registered unless you have also submitted the report.
- Your mark may also not be accepted or may be modified if your report is incomplete or is identical to that of another student.
- By uploading the report, you are agreeing with the following student statement on collusion and plagiarism, and must be aware of the possible consequences.

Student statement:

I have read the University's statement on cheating and plagiarism, as described in the *Student Resource Guide*. This work is original and has not previously been submitted as part of another unit/subject/course. I have taken proper care safeguarding this work and made all reasonable effort to make sure it could not be copied. I understand the consequences for engaging in plagiarism as described in *Statue 4.1 Part III – Academic Misconduct*. **I certify that I have not plagiarised the work of others or engaged in collusion when preparing this submission.**

– END –