ELEC6021 Research Methods
Communications Assignment I

Submission Details

This assignment forms your assessment for the "Communications 1" part of ELEC6021 Research Methods and contributes 25% of your mark for that course. Since this assignment is worth five credits, it should take you up to 50 hours to complete it. Please do not spend any more time than this since I'm sure you have better things to be doing!

You are required to work entirely on your own and produce a write-up, which only needs to include the figures, calculations and comments that are directly requested in the **bold** paragraphs below. Each figure in your write-up should have well labelled axes and a relevant title or legend that identifies which signals are shown.

When you are finished, you need to submit your write-up electronically at C-BASS
`https://handin.ecs.soton.ac.uk/handin/0910/ELEC6021/2/`
before 4pm on Thursday 12/11/2009. You also need to print out your write-up, staple or bind it together with your C-BASS receipt and submit it at the ECS reception before the same deadline.

If you notice any mistakes in this document or have any queries about it, please email me at `rm@ecs.soton.ac.uk`

Have fun, Rob Maunder.

1 Analogue Modulation

1.1 Double SideBand Suppressed Carrier Modulation

When a Radio Frequency (RF) signal is placed onto the antenna of a transmitter, it will propagate through the air and can be detected on the antenna of a receiver. The higher the frequency of this signal, the smaller the antennas that are required. However, we are often interested in communicating relatively low frequency signals, such as audio. Hence, we must modulate our low frequency signal onto a high frequency carrier, in order to transmit it.

Figure 1 shows the schematic of a transmission scheme that uses Double SideBand Suppressed Carrier (DSBSC) modulation to transmit an input signal $x(t)$ and demodulation to obtain a received signal $\hat{x}(t)$.

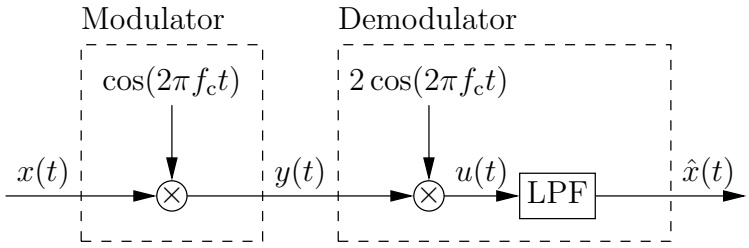

Figure 1: DSBSC modulation and demodulation.

As shown in Figure 1

$$y(t) = x(t)\cos(2\pi f_c t), \tag{1}$$
$$u(t) = 2x(t)\cos(2\pi f_c t)\cos(2\pi f_c t). \tag{2}$$

Using the trigonometric identity $2\cos(\theta)\cos(\theta) = 1 + \cos(2\theta)$, we get

$$u(t) = x(t)\left(1 + \cos(4\pi f_c t)\right), \tag{3}$$
$$= x(t) + x(t)\cos(4\pi f_c t). \tag{4}$$

a) Input Signal. For our input signal $x(t)$, we shall employ a normally distributed random signal that has been bandlimited to a maximum frequency of $f_{\max} = 10$ kHz. Let's start by generating a normally distributed random signal $x_w(t)$ and worry about bandlimiting it later. We'll simulate the analogue signal $x_w(t)$ over an interval of $T = 0.1$ s, using a high sampling rate of $f_s = 2$ MHz. Thus, the first step is to put the commands

```
f_s=2000000;
T=0.1;
samples=f_s*T;
t=(0:(samples-1))/f_s;
```

into a new Matlab .m file. You should use this .m file throughout Section 1.1. This way, you'll be able to modify it when you get to Section 1.2, making things much easier for you.

You can generate a random analogue signal $x_w(t)$ by putting the commands

```
randn('seed',1);
x_w=randn(size(t));
```

into your .m file. Here, we have *seeded* the random number generator, which will ensure that you'll get the same sequence of random numbers every time you run your .m file. This will make it easier for you to keep your results self-consistent in your write-up (and make it easier for me to mark it!)

Note that you can look up any unfamiliar Matlab commands and their parameters using Matlab's help facility. For example, in Matlab's command window type

```
help randn
```

to obtain information on how to create a Gaussian distributed random variable.

You can plot the first 1000 samples of the signal $x_{\mathrm{w}}(t)$ by putting the commands

```
figure;
plot(t(1:1000),x_w(1:1000));
```

into your .m file and running it. Use the 'insert' menu in the resultant figure to add relevant labels to the x and y axes of this plot, as well as a relevant title. You should do this for all of the plots generated in this assignment.

The signal $x_{\mathrm{w}}(t)$ is said to be *white*. This means that it contains components at all frequencies. You may check this by using the `pwelch` Matlab function, which displays the signal's *Power Spectral Density* (PSD). Simply put the commands

```
figure;
pwelch(x_w);
```

into your .m file and run it. Again, use the 'insert' menu in the resultant figure to change the title so that it indicates which signal the PSD relates to. You should do this for all of the PSDs generated in this assignment.

However, we require a signal that does not contain any components above a frequency of $f_{\max} = 10$ kHz. We can eliminate these components from $x_{\mathrm{w}}(t)$ using a *Low-Pass Filter* (LPF). Matlab's `fir1` function can be used to design an LPF. Take a look at the help that Matlab offers for this function using the command

```
help fir1
```

This tells us that the command

```
B=fir1(N,W_n);
```

designs an $\mathtt{N}^{\mathrm{th}}$ order LPF and returns the `N + 1` filter coefficients in a vector `B`. The normalised cut-off frequency `W_n` must be in the range $0 <$ `W_n` < 1, with 1 corresponding to half the sample rate f_s. A filter order of `N = 1023` is a good choice, but it's up to you to work out what value to use for `W_n`, with consideration of $f_{\max}$ and f_{s}.

The signal $x_{\mathrm{w}}(t)$ can be filtered to obtain $x(t)$ using the command

```
x=filter2(B,x_w);
```

Plot the first 1000 samples of the signal $x(t)$ and generate its PSD using the `plot` and `pwelch` functions, as before. Observe that the high frequency components of $x(t)$ have been attenuated by about 90 dB compared to the low frequency components. In other words, the PSD of the low frequency components is $10^{90/10} = 1000000000$ times higher than that of the high frequency components! Our LPF has done such a good job because its order `N = 1023` is so high.

Export the plots and PSDs of the signals $x_{\mathrm{w}}(t)$ and $x(t)$ as .jpg images and include these in your write-up. Comment on the differences between the two signals and between the two PSDs. Also, explain the calculation you used to choose the normalised cut-off frequency `W_n` for the LPF.

b) Modulation. Create a carrier $c(t)$ with frequency $f_{\mathrm{c}} = 250$ kHz using the commands

```
f_c=250000;
c=cos(2*pi*f_c*t);
```

Plot the first 1000 samples of this carrier $c(t)$ using the `plot` function, as before. You may like to use the zoom button in the resultant figure to take a closer look at the carrier wave. Also, generate the PSD of $c(t)$ using the `pwelch` function, as before.

Next, we need to multiply the carrier $c(t)$ with the input signal $x(t)$ in order to obtain the

DSBSC signal $y(t)$, as shown in Figure 1. You can do this using the command

```
y=c.*x;
```

Note that we must use the element-wise multiplication (`.*`) rather than the vector multiplication (`*`) of `c` and `x`. Again, plot the first 1000 samples of this DSBSC signal $y(t)$ and generate its PSD using the `plot` and `pwelch` functions, as before.

Include the new plots and PSDs in your write-up. Comment on how the plots of $x(t)$ and $c(t)$ may be combined to obtain that of $y(t)$. Similarly, comment on how the PSD of $y(t)$ may be obtained by combining those of $x(t)$ and $c(t)$. What is the bandwidth of $y(t)$ in Hertz? How does this compare to the maximum frequency in $x(t)$, namely $f_{\max}$?

c) Demodulation. Take a look at the demodulator in Figure 1. Notice that it doesn't use any components that we haven't already considered. Demodulation should be no problem for a student of your talents!

Start by transforming $y(t)$ into $u(t)$. You can use similar Matlab code to that which transformed $x(t)$ into $y(t)$, but don't forget to multiply by two, as shown in Figure 1. Plot the first 1000 samples of the signal $u(t)$ and generate its PSD using the `plot` and `pwelch` functions, as before. Note that an image signal has arisen in the PSD.

This image signal can be removed using the LPF shown in Figure 1. With consideration of the PSD of $u(t)$, choose a normalised cut-off frequency `W_n` for this LPF that is half-way between the desired signal and the image signal. A filter order of `N = 23` will be sufficient for this LPF; there is no need to attenuate the image signal by as much as a thousand million times in this case! Obtain the reconstructed signal $\hat{x}(t)$ and compare its first 1000 samples to those of $x(t)$ using the commands

```
figure;
plot(t(1:1000),xhat(1:1000),'-', t(1:1000),x(1:1000),'--');
```

Here, the parameter `'-'` tells Matlab to plot $\hat{x}(t)$ using a solid line, while the parameter `'--'` results in a dashed line for $x(t)$. Make sure you are happy that $\hat{x}(t)$ and $x(t)$ are very similar. Also, generate the PSD of $\hat{x}(t)$ using the `pwelch` function, as before.

Include the new plots and PSDs in your write-up. Comment on how the plots of $u(t)$ and $\hat{x}(t)$ compare to those of $x(t)$ and $y(t)$. Similarly, comment on how the PSDs of $u(t)$ and $\hat{x}(t)$ compare to those of $x(t)$ and $y(t)$. Also, comment on how the PSD of $u(t)$ relates to the equations provided below Figure 1. Explain the calculation you used to choose the normalised cut-off frequency `W_n` for the LPF.

1.2 Quadrature Amplitude Modulation

So far, we have only considered the use of a cosine carrier wave. If we had used a sine carrier wave in Section 1.1, we would have obtained very similar results. This is because the sine and cosine functions differ only by a phase shift of $\pi/2$ radians, ie $\cos(\theta) = \sin(\theta + \pi/2)$. As a result, $\sin(\theta) = 0$ if $\cos(\theta) = \pm 1$ and $\cos(\theta) = 0$ if $\sin(\theta) = \pm 1$. These results indicate that the cosine and sine functions are *orthogonal* to each other. This means that a signal $x_{\mathrm{i}}(t)$ that is amplitude modulated onto a cosine carrier wave will not interfere with another signal

$x_{\mathrm{q}}(t)$ that is modulated onto a sine carrier wave having the same frequency f_{c}. In this way, we can transmit two signals at once, which is useful for stereo audio for example. We refer to these signals as the *in-phase signal* $x_{\mathrm{i}}(t)$ and the *quadrature-phase signal* $x_{\mathrm{q}}(t)$. The additional presence of the quadrature-phase signal gives *Quadrature Amplitude Modulation* (QAM) its name. A schematic for a QAM scheme is shown in Figure 2

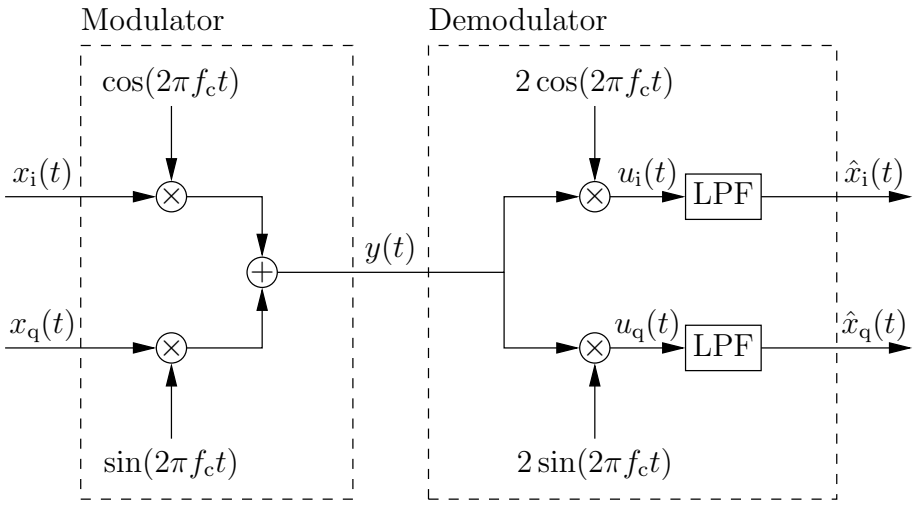

Figure 2: QAM modulation and demodulation.

As shown in Figure 2

$$y(t) = x_{\mathrm{i}}(t)\cos(2\pi f_{\mathrm{c}}t) + x_{\mathrm{q}}(t)\sin(2\pi f_{\mathrm{c}}t), \tag{5}$$
$$u_{\mathrm{i}}(t) = 2x_{\mathrm{i}}(t)\cos(2\pi f_{\mathrm{c}}t)\cos(2\pi f_{\mathrm{c}}t) + 2x_{\mathrm{q}}(t)\sin(2\pi f_{\mathrm{c}}t)\cos(2\pi f_{\mathrm{c}}t), \tag{6}$$
$$u_{\mathrm{q}}(t) = 2x_{\mathrm{q}}(t)\sin(2\pi f_{\mathrm{c}}t)\sin(2\pi f_{\mathrm{c}}t) + 2x_{\mathrm{i}}(t)\cos(2\pi f_{\mathrm{c}}t)\sin(2\pi f_{\mathrm{c}}t). \tag{7}$$

Using the trigonometric identities $2\cos(\theta)\cos(\theta) = 1+\cos(2\theta)$, $2\sin(\theta)\sin(\theta) = 1-\cos(2\theta)$ and $2\cos(\theta)\sin(\theta) = \sin(2\theta)$, we get

$$u_{\mathrm{i}}(t) = x_{\mathrm{i}}(t) + x_{\mathrm{i}}(t)\cos(4\pi f_{\mathrm{c}}t) + x_{\mathrm{q}}(t)\sin(4\pi f_{\mathrm{c}}t), \tag{8}$$
$$u_{\mathrm{q}}(t) = x_{\mathrm{q}}(t) - x_{\mathrm{q}}(t)\cos(4\pi f_{\mathrm{c}}t) + x_{\mathrm{i}}(t)\sin(4\pi f_{\mathrm{c}}t). \tag{9}$$

After the high-frequency components of $u_{\mathrm{i}}(t)$ and $u_{\mathrm{q}}(t)$ are removed by the LPFs shown in Figure 2, we obtain

$$\hat{x}_{\mathrm{i}}(t) = x_{\mathrm{i}}(t), \tag{10}$$
$$\hat{x}_{\mathrm{q}}(t) = x_{\mathrm{q}}(t), \tag{11}$$

as desired.

a) Plots and PSDs. Modify the Matlab .m file you wrote for Section 1.1 in order to simulate the scheme of Figure 2. You can generate $x_{\mathrm{i}}(t)$ and $x_{\mathrm{q}}(t)$ in a similar manner to how you generated $x(t)$ in Section 1.1a, by using the commands

```
randn('seed',1);
x_w = randn(size(t));
x_i = filter2(B,x_w);
x_w = randn(size(t));
```

```
x_q = filter2(B,x_w);
```

Note that by regenerating $x_{\mathrm{w}}(t)$ before obtaining $x_{\mathrm{q}}(t)$, we ensure that it differs to $x_{\mathrm{i}}(t)$.

You can obtain $y(t)$ by using the commands

```
c_i=cos(2*pi*f_c*t);
c_q=sin(2*pi*f_c*t);
y=c_i.*x_i+c_q.*x_q;
```

You can obtain $u_{\mathrm{i}}(t)$ and $u_{\mathrm{q}}(t)$ in the same way that you obtained $u(t)$ in Section 1.1c. These signals can be filtered using the same LPF as the one you used in Section 1.1c.

Generate the PSD of $y(t)$ in the same way that you did in Section 1.1b. Also, plot the first 1000 samples of $x_{\mathrm{i}}(t)$ and $\hat{x}_{\mathrm{i}}(t)$ in the same figure, like you did in Section 1.1c. Do the same for $x_{\mathrm{q}}(t)$ and $\hat{x}_{\mathrm{q}}(t)$. Make sure that you are convinced that QAM allows the transmission of two signals at the same time.

Include the PSD and one of the new plots in your write-up. With consideration of the PSD, explain how the bandwidth of $y(t)$ compares to that obtained for the DSBSC scheme in Section 1.1b? This may seem surprising since the DSBSC scheme only transmits one signal, whereas the QAM scheme transmits two. If not in bandwidth, how does the QAM scheme 'pay' for the additional signal?

1.3 Complex Quadrature Amplitude Modulation

Note that just like how our signal comprises the two components $x_{\mathrm{i}}(t)$ and $x_{\mathrm{q}}(t)$, there are two components to a complex number, namely the real part and the imaginary part. Complex numbers can therefore conveniently represent the two parts of our signal, according to

$$x(t) = x_{\mathrm{i}}(t) + jx_{\mathrm{q}}(t), \tag{12}$$

where $j = \sqrt{-1}$. In this case, the schematic of Figure 2 is transformed into that of Figure 3.

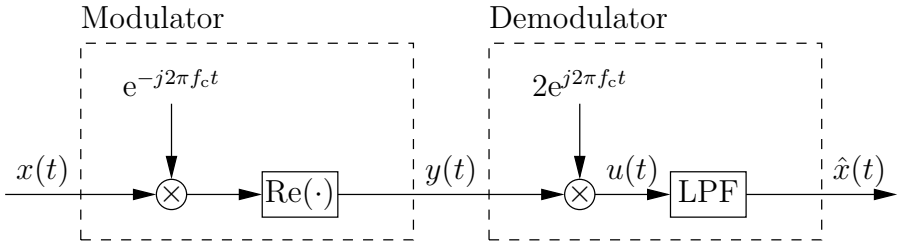

Figure 3: Complex QAM modulation and demodulation.

As shown in Figure 3

$$y(t) = \mathrm{Re}\left[x(t)\mathrm{e}^{-j2\pi f_c t}\right], \tag{13}$$

where $\mathrm{Re}[a + jb] = a$.

a) Calculations. Use the following identities to determine expressions for $y(t)$, $u(t)$ and $\hat{x}(t)$.

$$j^2 = -1, \tag{14}$$
$$\mathrm{e}^{-j\theta} = \cos(\theta) - j\sin(\theta), \tag{15}$$
$$\mathrm{e}^{j\theta} = \cos(\theta) + j\sin(\theta), \tag{16}$$
$$2\cos(\theta)\cos(\theta) = 1 + \cos(2\theta), \tag{17}$$
$$2\sin(\theta)\sin(\theta) = 1 - \cos(2\theta), \tag{18}$$
$$2\cos(\theta)\sin(\theta) = \sin(2\theta). \tag{19}$$

Provide your calculations in your write-up, showing where you have used each of the identities provided above. Comment on how your expression for $y(t)$ compares to that provided in Equation 5 for the scheme of Figure 2. Comment on how your expression for $\hat{x}(t)$ compares to that of Equation 12.

1.4 Additive White Gaussian Noise

So far, we've assumed that our channel does not adversely affect our transmitted signal $y(t)$. Let's see how our analogue modulation scheme performs when the channel introduces some *Additive White Gaussian Noise* (AWGN) $n(t)$, as shown in Figure 4.

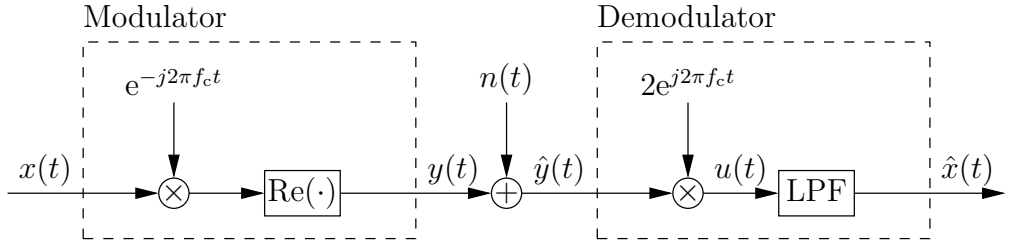

Figure 4: Complex QAM modulation and demodulation in the presence of AWGN.

This noise $n(t)$ is additive because it is added to the transmitted signal $y(t)$. It is Gaussian because it may be generated using a sequence of Gaussian distributed random variables, just like how our signal $x_{\mathrm{w}}(t)$ was generated in Section 1.1a. As described in Section 1.1a, sequences of Gaussian distributed random variables have components at all frequencies. Therefore, our noise $n(t)$ is white.

a) Plots and PSDs. Modify your .m file from Section 1.2 in order to simulate the scheme of Figure 4. Generate the noise $n(t)$ using the commands

```
randn('seed',1);
n=0.01*randn(size(t));
```

Here, the multiplier of 0.01 results in some relatively low-power noise. Here are some other commands to help you simulate the scheme of Figure 4

```
x=x_i+j*x_q;
yhat=real(x.*exp(-j*2*pi*f_c*t))+n;
u=2*yhat.*exp(j*2*pi*f_c*t);
```

Compare plots of the transmitted and received signals using the commands

```
figure;
plot(t(1:1000),real(xhat(1:1000)),'-', t(1:1000),real(x(1:1000)),'--');
figure;
plot(t(1:1000),imag(xhat(1:1000)),'-', t(1:1000),imag(x(1:1000)),'--');
```

Also, generate PSDs for the two components of the received signal using the commands

```
figure;
pwelch(real(xhat));
figure;
pwelch(imag(xhat));
```

Include one of the plots and one of the PSDs in your write-up. Comment on how the AWGN has affected the demodulated signals. Comment on how the PSD compares to that obtained for $\hat{x}(t)$ in Section 1.1c. How has this PSD been affected by your choice of the cut-off frequency for the LPF of Figure 4? Comment on how this would influence your choice of cut-off frequency in the future.

2 Digital Modulation

2.1 4-ary Quadrature Amplitude Modulation

The previous section showed that analogue modulation schemes are susceptible to noise. In this section, we'll show that digital modulation schemes can achieve reliable communications even in the presence of relatively severe noise. The difference between an analogue and a digital modulation scheme is the type of signal they are used to convey. As we showed in Section 1, analogue modulators transmit an analogue signal $x(t)$. However, the analogue demodulator can never be sure if a particular component of the demodulated signal $\hat{x}(t)$ is signal or noise. By contrast, digital modulators transmit digital signals, such as a sequence of binary digits $b[n]$. Since a bit can only have a value of '0' or '1', the demodulator just has to choose from these two values when recovering the sequence $\hat{b}[n]$. While the demodulator can never be sure that it has made the right choices, it will typically do a good job so long as the noise is not really bad.

We can construct a digital modulation scheme by converting the digital signal $b[n]$ into an analogue signal $x(t)$ and using the analogue modulation scheme of Figure 4. Once the demodulated signal $\hat{x}(t)$ has been recovered, we just need to convert it back into a digital signal $\hat{b}[n]$. Schematics for a Digital to Analogue Converter (DAC) and an Analogue to Digital Converter (ADC) are provided in Figures 5 and 6.

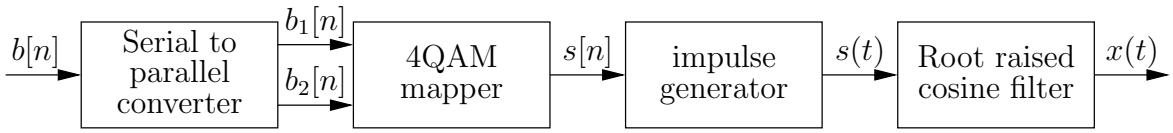

Figure 5: Digital to analogue conversion using 4QAM.

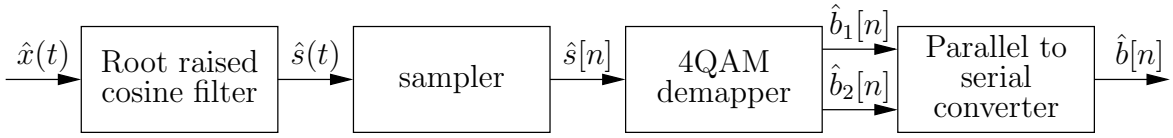

Figure 6: Analogue to digital conversion using 4QAM.

a) Bit mapping. Let's consider a scheme that transmits bits at a rate of $f_\mathrm{b} = 20$ kbit/s. Start by adding the following commands into the .m file you created in Section 1.4.

```
f_b=20000;
bits=f_b*T;
```

You can generate a random bit sequence $b[n]$ using the commands

```
rand('seed',1);
b=round(rand(1,bits));
```

You may like to look up the `rand` and `round` functions in Matlab's help facility to see how this works.

Let's use $M = 4$-ary Quadrature Amplitude Modulation (4QAM), like in Figures 5 and 6. This transmits $\log_2(M) = 2$ bits at a time by combining them into a single 4QAM symbol. You can determine the symbol rate f_symb, the total number of symbols and the number of samples per symbol using the commands

```
bits_per_symbol=2;
f_symb=f_b/bits_per_symbol;
symbols=bits/bits_per_symbol;
samples_per_symbol=f_s/f_symb;
```

As shown in Figure 5, the first step in 4QAM modulation is to convert our serial sequence of bits $b[n]$ into two parallel sequences $b_1[n]$ and $b_2[n]$. You can do this using the commands

```
b_mat=reshape(b,bits_per_symbol,symbols);
b_1=b_mat(1,:);
b_2=b_mat(2,:);
```

Verify that this works by comparing the first 10 bits in $b[n]$ (use the command `b(1:10)`) with the first 5 bits in $b_1[n]$ and $b_2[n]$ (use the commands `b_1(1:5)` and `b_2(1:5)`). Notice that the bits in $b[n]$ having even indices go into one of the parallel bit sequences, while the bits with odd indices go into the other.

Next, we need to map each pair of bits to a 4QAM symbol, in order to generate the symbol sequence $s[n]$, as shown in Figure 5. Like in the complex QAM scheme of Section 1.3, the 4QAM symbols are complex. You can do the mapping using the command

```
sn=a*(-2*(b_1-0.5)+j*-2*(b_2-0.5));
```

Here, you need to choose the value for `a` that results in an average symbol power of one. You can check that the average symbol power is normalised in this way using the command

```
sum(abs(sn).^2)/symbols
```

Output the first five values in your resultant symbol sequence $s[n]$ using the command `sn(1:5)`. Make sure that you can see how the bit values in $b_1[n]$ and $b_2[n]$ relate to the corresponding complex values in $s[n]$. The possible values of the 4QAM symbols in $s[n]$ can be visualised using the *constellation diagram* of Figure 7.

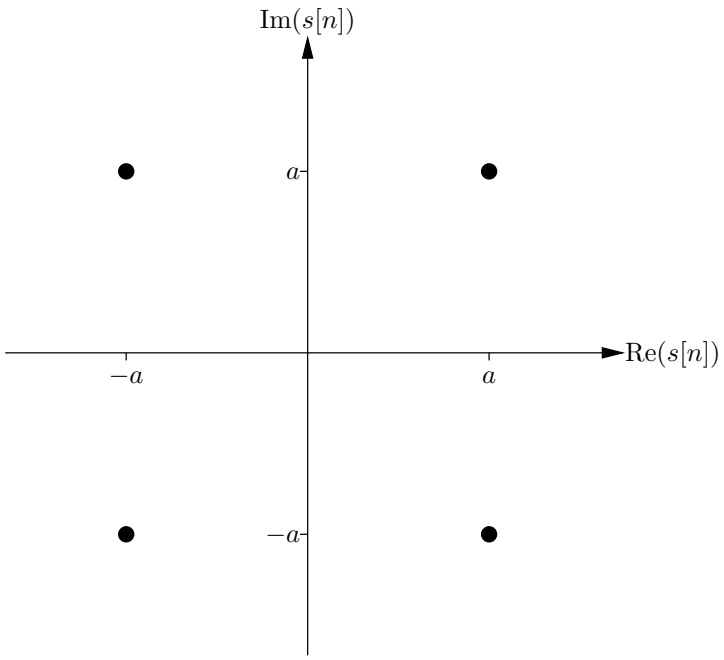

Figure 7: 4QAM constellation diagram.

The bit mapping we have used is called the *Gray bit mapping*. Table 1 would detail this bit mapping, but it is incomplete.

$b_1[n]$	$b_2[n]$	$s[n]$
0	0	
0	1	
1	0	
1	1	

Table 1: 4QAM Gray bit mapping

Include the bit and symbol values that you observed in your write-up. Include a table like Table 1 in your write-up and fill in the missing complex values of $s[n]$ by examining the provided Matlab code. Comment on how the distance between the various pairs of constellation points in Figure 7 is related to how similar their mapped bit values are. Comment on why this is beneficial. Derive an expression for the value of `a` that correctly normalises the average symbol power and include this in your write-up.

b) Impulse generation. As shown in Figure 5, the next step is to convert our sequence of discrete 4QAM symbols $s[n]$ into a continuous function of time $s(t)$. We can do this by generating impulses having the corresponding complex amplitudes, as shown in Figure 5. Since our symbols are generated at a rate of $f_{\text{symb}} = 10^4$ symbols/s, each one has a duration of 10^{-4} s. Let's generate our impulses at the midpoints of these durations. Remember that in our simulation, we are using samples to model the various continuous functions of time. Hence, you can generate $s(t)$ by up-sampling $s[n]$ by `samples_per_symbol` times. You can do this using the command

```
st=upsample(sn*samples_per_symbol,samples_per_symbol,samples_per_symbol/2);
```

Here, the third parameter ensures that the impulses occur at the midpoints of the symbol durations.

Plot the real and imaginary parts of the first 1000 samples in $s(t)$. Make sure that you can see how these plots relate to the first five bits of $b_1[n]$ and $b_2[n]$, as well as the first five complex values in $s[n]$. Also, generate the PSDs for the real and imaginary parts of $s(t)$.

Include these plots and one of the PSDs in your write-up. With reference to your PSD, comment on why it is necessary to apply a LPF to $s(t)$ before modulating it onto the channel.

c) Root raised cosine filtering. The root raised cosine filter of Figure 5 is a special type of LPF that is particularly suited for digital modulation in the presence of AWGN. You can generate a root raised cosine filter using the command

```
B_rrcos=firrcos(10*samples_per_symbol,f_symb/2,f_symb,f_s,'sqrt');
```

Like in Section 1.1a, use the `filter2` function to filter $s(t)$ and generate the analogue signal $x(t)$.

Compare the real part of the signals $s(t)$ and $x(t)$ using the commands

```
figure;
plot(t(1:1000),real(x(1:1000)),'-',
                    t(1:1000),real(st(1:1000))/samples_per_symbol,'--');
```

Do the same for the imaginary part. Also, generate PSDs for the real and imaginary parts of $x(t)$.

Include these plots and one of the PSDs in your write-up. Comment on how the signals $s(t)$ and $x(t)$ are related. What is the maximum frequency that is present within the signal $x(t)$? How does this compare to the symbol rate f_{symb}?

d) Modulation and demodulation. Use the Matlab code you wrote for Section 1.4 to simulate the transmission of your signal $x(t)$. Compare the real and imaginary parts of $x(t)$ with those of $\hat{x}(t)$ using the commands provided in Section 1.4a. Confirm that these signals are very similar and that they only differ because of the noise you introduced.

State the bandwidth of $y(t)$ in your write-up.

e) Root raised cosine filtering 2. As shown in Figure 6, the first step in obtaining $\hat{b}[n]$ from $\hat{x}(t)$ is to filter it. This will remove some of the noise that was not filtered out by the LPF of Figure 4. You should use the same root raised cosine filter that you designed in Section 2.1c and you should use the `filter2` function again. Compare the real and imaginary parts of the resultant signal $\hat{s}(t)$ with those of $s(t)$ by using commands similar to the one provided in Section 2.1c.

As shown in Figures 5 and 6, the impulses of $s(t)$ pass through two root raised cosine filters, one in the transmitter and one in the receiver. Figure 8 shows the impulse response of this pair of filters. In the absence of any channel noise, the signal $\hat{s}(t)$ can be obtained by *convolving* $s(t)$ with the response shown in Figure 8. This convolution replaces each impulse in $s(t)$

with a version of the impulse response shown in Figure 8 that has the corresponding complex amplitude.

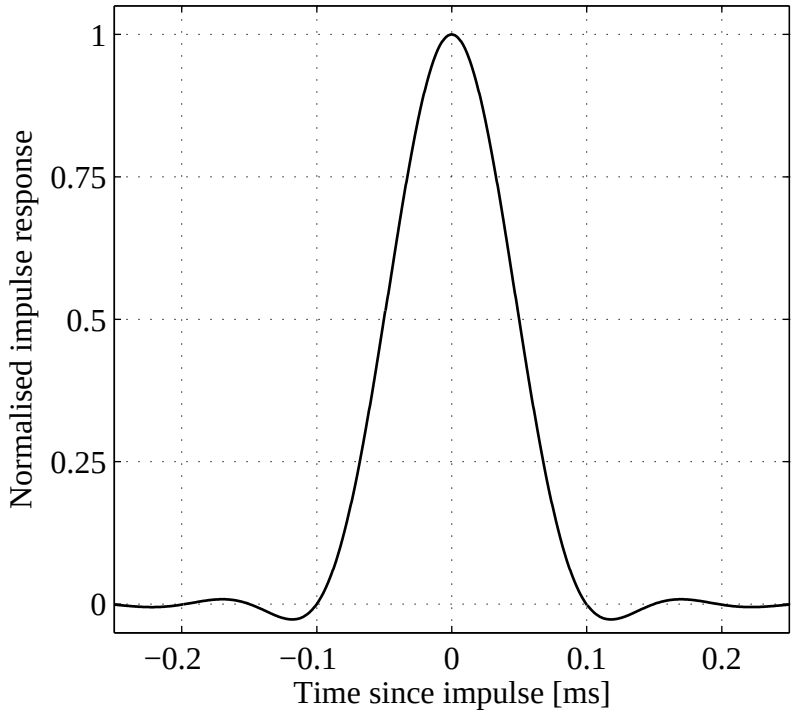

Figure 8: Impulse response of a pair of root raised cosine filters.

Include your new plots in your write-up. Explain the features of Figure 8 which show that the root raised cosine filters of Figures 5 and 6 facilitate (i) a low bandwidth, (ii) a low filter order and (iii) no inter symbol interference.

f) Sampling and 4QAM demapping. Next, we need to sample $\hat{s}(t)$ at the midpoints of the symbol durations, as shown in Figure 6. These midpoints coincide with the impulses shown in the plots you just generated. You can sample $\hat{s}(t)$ using the command

```
snhat=sthat(upsample(ones(size(sn)),
                     samples_per_symbol,samples_per_symbol/2)==1);
```

Output the first five values in the resultant sequence $\hat{s}[n]$ using a command similar to the one suggested in Section 2.1a. Make sure that you can see how these five values are obtained from $\hat{s}(t)$ by comparing them with the plots you generated in Section 2.1e.

As shown in Figure 6 the next step is 4QAM demapping. This can be performed by seeing where the samples in $\hat{s}[n]$ occur within the constellation diagram of Figure 7. More specifically, we need to decide which constellation point is the nearest to each sample. For 4QAM, we can do this by seeing which side of the axes the samples are on. This can be achieved using the commands

```
b_1hat=real(snhat)<0;
b_2hat=imag(snhat)<0;
```

Display the first five bits in the resultant sequences $\hat{b}_1[n]$ and $\hat{b}_2[n]$ using similar commands to the ones suggested in Section 2.1a. Make sure you can see why these values have been selected for these bits by considering the first five values in `snhat`.

The final step of Figure 6 is parallel to serial conversion. You can achieve this using the commands

```
b_mathat=[b_1hat;b_2hat];
b_hat=reshape(b_mathat,1,bits);
```

Display the first ten bits in the resultant sequence $\hat{b}[n]$ using a similar command to the one suggested in Section 2.1a.

Include the values of $\hat{s}[n]$, $\hat{b}_1[n]$, $\hat{b}_2[n]$ and $\hat{b}[n]$ that you observed in your write-up. Comment on how these values compare to the ones you observed in Section 2.1a.

g) Bit Error Ratio. We can assess how well our 4QAM modulation scheme has mitigated the noise $n(t)$ by comparing $b[n]$ with $\hat{b}[n]$. The similarity of these bit sequences may be quantified using the Bit Error Ratio (BER), which is obtained using the command

```
ber=sum(b_hat~=b)/bits
```

In Section 1.4a, a multiplier of 0.01 was applied to the noise signal $n(t)$. Gradually increase this multiplier until at least one of the first ten bits in $\hat{b}[n]$ differs to the corresponding bit in $b[n]$.

Using your new noise multiplier, compare the real and imaginary parts of $x(t)$ with those of $\hat{x}(t)$ by invoking the commands provided in Section 1.4a. Also, compare the real and imaginary parts of $\hat{s}(t)$ with those of $s(t)$ by using commands similar to the one provided in Section 2.1c.

Include the new plots in your write-up. Also provide the first five values of $\hat{s}[n]$, $\hat{b}_1[n]$ and $\hat{b}_2[n]$, as well as the first ten values of $\hat{b}[n]$. Comment on why the observed bit errors have occurred. State the BER that you obtained.

2.2 16-ary Quadrature Amplitude Modulation

In the 4QAM scheme of Section 2.1, we transmitted two bits at a time and achieved a bit rate of $f_\mathrm{b} = 20$ kbit/s. However, we can achieve higher bit rates by transmitting more bits at a time. In $M = 16$-ary Quadrature Amplitude Modulation (16QAM), we transmit $\log_2(M) = 4$ bits at a time, as shown in Figures 9 and 10.

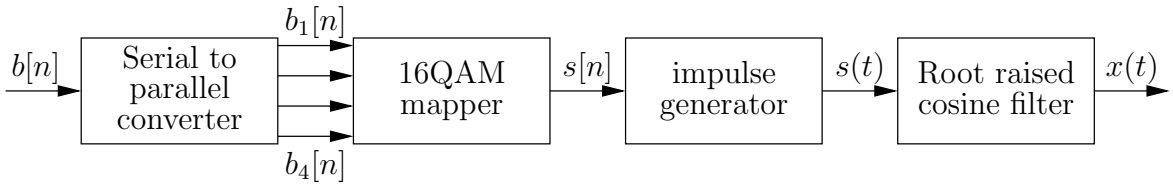

Figure 9: Digital to analogue conversion using 16QAM.

Compare Figures 9 and 10 with Figures 5 and 6. Notice that the only differences are in the design of the serial to parallel converter, the bit mapper, the bit demapper and the parallel to serial converter. In this section, you'll need to write some Matlab code that will simulate each of these components.

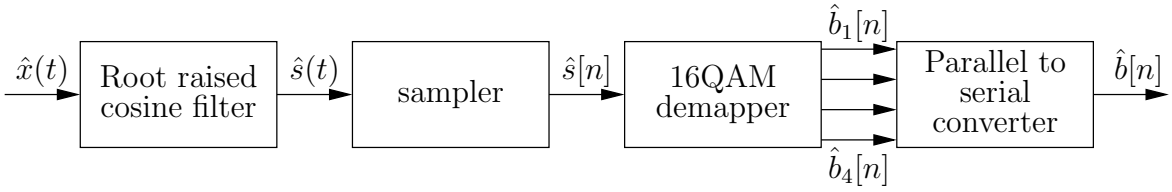

Figure 10: Analogue to digital conversion using 16QAM.

a) Serial to parallel conversion. Modify the .m file you wrote in Section 2.1 to give a bit rate of $f_\mathrm{b} = 40$ kbit/s. Also, modify this code to decompose the bit sequence $b[n]$ into four sequences $b_1[n]$, $b_2[n]$, $b_3[n]$ and $b_4[n]$, as shown in Figure 9. Output the first 20 bits in $b[n]$, as well as the first five bits in each of $b_1[n]$, $b_2[n]$, $b_3[n]$ and $b_4[n]$.

Include these values in your write-up and comment on how they are related. Also, include any lines of Matlab code that you modified.

b) Bit mapping. The constellation diagram for 16QAM is provided in Figure 11. The constellation points in this figure are labelled with the corresponding Gray-mapped bit values of $b_1[n]$, $b_2[n]$, $b_3[n]$ and $b_4[n]$.

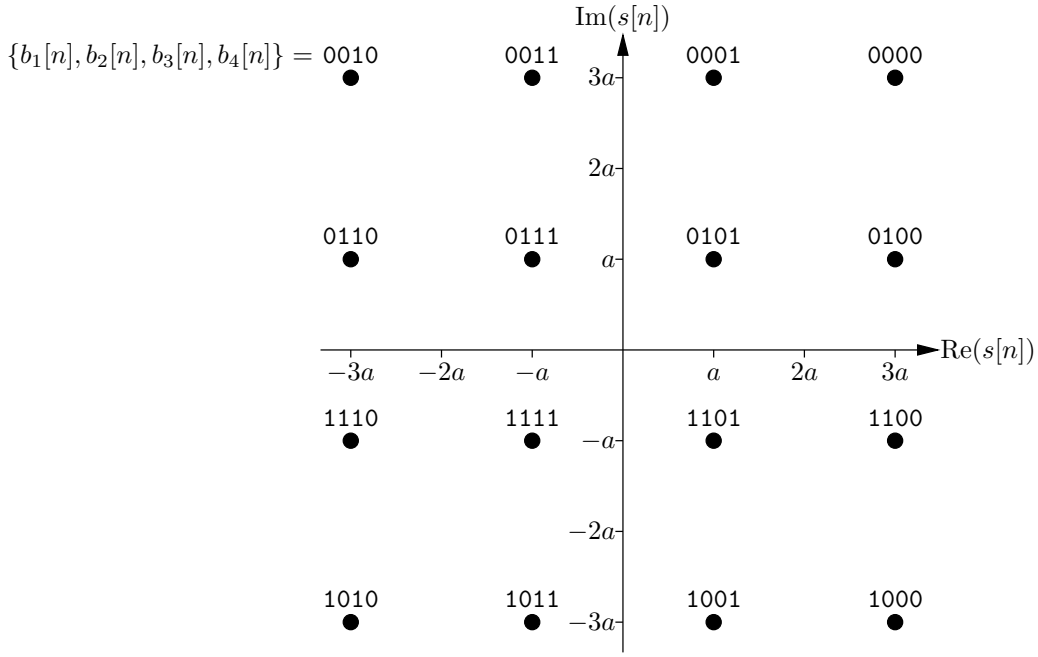

Figure 11: 16QAM constellation diagram and Gray bit mapping.

Write some Matlab code to perform 16QAM bit mapping. There are a number of ways that you could do this, so feel free to get creative! You'll need to normalise the average symbol power to one by selecting a value for the constant a of Figure 11. Note that this value will be different to the one you used in 4QAM. Because the average symbol powers of both the 4QAM and 16QAM schemes are normalised in this way, it is fair to compare them. Output the first five complex values in the resultant symbol sequence $s[n]$.

Include these values in your write-up and comment on how they are related to the bit values you obtained in Section 2.2a. Also, include your Matlab code for

performing 16QAM bit mapping. Comment on how the distance between the various pairs of constellation points in Figure 11 is related to how similar their mapped bit values are. Comment on why this is beneficial. Finally, include a derivation of the constant a.

c) Modulation and demodulation. Use the Matlab code you wrote in Section 2.1 to transmit $s[n]$ and recover $\hat{s}[n]$. Output the first five complex values in $\hat{s}[n]$.

Include these values in your write-up. Also, include plots and PSDs that correspond to the ones you generated in Section 2.1. What is the maximum frequency that is present within the signal $x(t)$? How does this compare to the symbol rate f_{symb}? What is the bandwidth of the signal $y(t)$? How does this compare to the bandwidth of the 4QAM scheme?

d) Bit demapping. Write some Matlab code to perform 16QAM bit demapping. Again, there are a number of ways that you could do this. Display the first five bits in the resultant sequences $\hat{b}_1[n]$, $\hat{b}_2[n]$, $\hat{b}_3[n]$ and $\hat{b}_4[n]$.

Include these values in your write-up and comment on how they relate to the first five values in $\hat{s}[n]$. Also, include the Matlab code you wrote to perform 16QAM demapping. Explain how your code decides what value each bit should have.

e) Parallel to serial conversion. Write some Matlab code to perform parallel to serial conversion. Display the first 20 bits in the resultant sequence $\hat{b}[n]$.

Include these values in your write-up and comment on how they relate to the sequences $\hat{b}_1[n]$, $\hat{b}_2[n]$, $\hat{b}_3[n]$ and $\hat{b}_4[n]$. Also, include the Matlab code you wrote to perform parallel to serial conversion. What BER do you obtain when you use the same noise multiplier that you found in Section 2.1g? Your 16QAM scheme has the same bandwidth and average transmit power as the 4QAM scheme, but it has the advantage of a higher bit rate. What are the disadvantages of 16QAM compared to 4QAM? Which of the bit sequences $\hat{b}_1[n]$, $\hat{b}_2[n]$, $\hat{b}_3[n]$ and $\hat{b}_4[n]$ do you think are most susceptible to bit errors? Explain your reasoning.

Sheng Chen 07/10/2005
Rob Maunder 15/10/2009