B1 Engineering Computation Project

Spectral Shaping for Communication Systems

Michaelmas 2014 Prof. Justin Coon
Rev: October 24, 2014 justin.coon@eng.ox.ac.uk

What to do first

1. You will need a copy of these notes.
2. Once logged into a machine using your single-sign-on username and password visit the B1 Mini Projects directory on Weblearn. Navigate to the directory associated with this project, then download and unzip the file contained within.
3. Unzipping will create a B1 directory with various subdirectories in it.
    - Notes: containing pdfs of these notes and a published paper, which you will refer to throughout the project.
    - Code: containing subdirectories
        - Preliminaries: contains three .m files
        - Optimisation: contains one .m file
4. Start up Matlab and point it to the Preliminaries directory ...

Take care to make backups

As this is an extended project, and you may wish to work on your own machine as well as on those in the Department, it is vital that you maintain a backup of your work, and keep track of the latest version. I suggest treating the Departmental copy as your master copy (as it is backed up properly). Also never rely on a memory stick for backup. Sticks are great for transporting stuff from A to B, but flash memory can die suddenly.

1 Introduction

1.1 Signals and Systems

Many engineered systems can be modelled mathematically using input signals, output signals, and a set of mappings that describe the relationship between the them. The signals, themselves, are typically represented as mathematical functions or data sets, which are frequently obtained through electrical measurement processes. Such measurements might be used to monitor stresses in mechanical systems, flows in fluidic media, or potentials in electrical circuits.

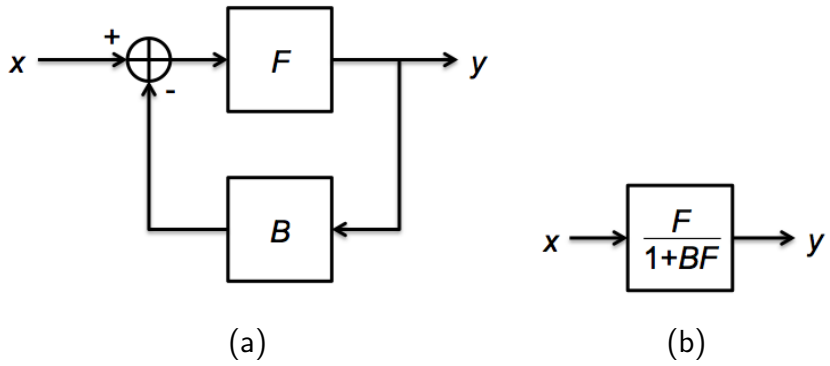


Figure 1: A block diagram of a control system with feedback (a) and its equivalent simplified representation (b).

As an example of a system with one (possibly multidimensional) input and one output, consider the simple feedback controller depicted in Fig. 1. (You will have seen this crop up in P1 and various other places.) The input is x, the output is y, and the mapping from input to output is given by the transfer function $F/(1 + BF)$, with F denoting the feedforward part of the function and B signifying the feedback part. We have spoken a little about the nature of x and y; but what does the transfer function do? Well, this entirely depends on the application and the nature of the system. We might consider the case where the input is a vector that represents the desired position of a copper electrode on the head of a spot welder in three dimensional space (think about the production line in an automobile factory - see Fig. 2). The output might be the current position of the electrode. Thus, the signal after the summing junction in Fig. 1(a) would represent the error in the desired position. The feedforward transfer function F could then signify an error analysis process that is fed into the robot positioning

software, which in turn adjusts the position of the electrode. When the feedback element B is unity, this example relates to the classical method of *proportional-integral-derivative* (PID) control. A considerable amount of information on these systems can be found on the web.

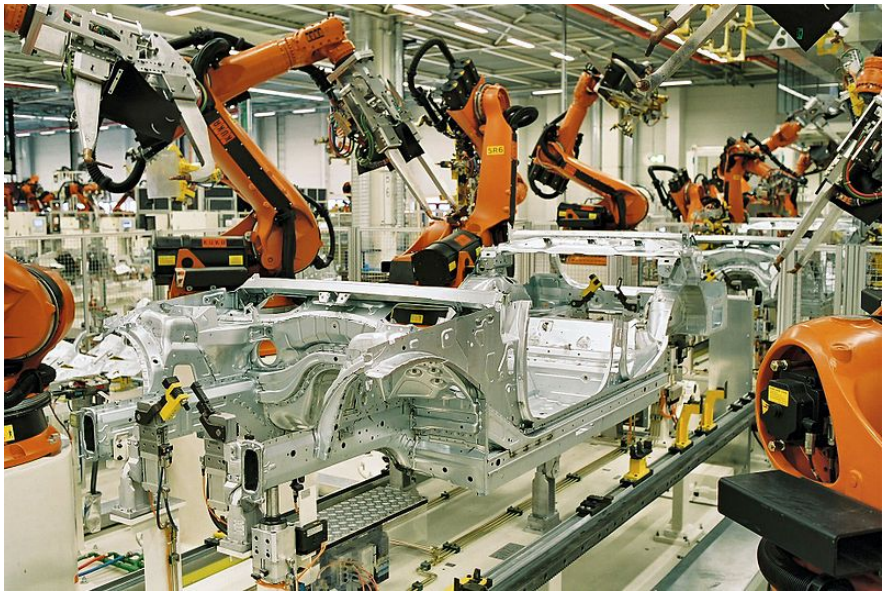


Figure 2: Robotic spot welding. (BMW Werk Leipzig / CC-BY-SA-2.0-DE)

Alternatively, let us consider the scenario where x represents an electrical signal with a certain bandwidth, and y is a modified signal, which is confined to a bandwidth that is a subset of that occupied by x. In this case, the transfer function would define a process that creates a spectral notch in the desired way. In fact, this second example will be the main consideration of the rest of this project. In particular, the input x will be defined as a *communication signal*, which must be processed with a time-domain *window* that induces the desired modification in the shape of the signal spectrum (see Fig. 3) prior to transmission. But before getting into the details of the project, let us spend a bit more time on the basics of communication system design.

1.2 Communication Systems

The theory underpinning communication systems is very straightforward: at a source, information can be *encoded*, resulting in some physical signal, which can then be conveyed through a *channel* to a destination, where a *decoding*

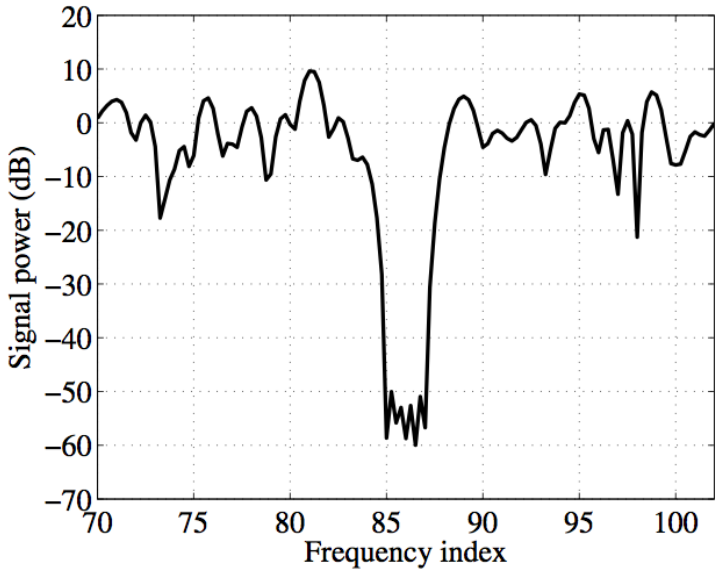


Figure 3: The spectrum of a communication signal that has shaped to yield a notch between the 85th and 87th frequency indices inclusive.

process can be used to recover the transmitted message. In reality, communication systems are much more complicated, encompassing many stages of filtering, up/down-conversion (i.e., changing the nominal frequency of the signal), amplification, etc. For the purposes of this project, we will consider a *baseband equivalent* model of a communication system, whereby many of the standard signal conditioning operations are abstracted in favour of a simple, discrete-time, vector-based model[1]. In this case, we can assume that the information message takes the form of a string of binary symbols ('bits'), which can then be mapped to *constellation symbols* in the complex plane. We will say that each group of m bits map to a single M-ary constellation symbol, where $M = 2^m$ (i.e., each constellation symbol represents $m = \log_2 M$ bits).

For example, a 16-ary rectangular constellation in the complex plane might look something like the one shown in Fig. 4. A standard bit-to-constellation-symbol mapping scheme is also depicted in the figure[2]. The 'I' and 'Q' labels

[1]The rigorous justification for using this simplified model of a communication system lies with the so-called *sampling theorem*, which was discovered by Harry Nyquist and Claude Shannon, two of the founders of modern communication theory. The theorem basically states that continuous signals with bandwidth limited to some finite region can be *sampled* at discrete points in such a way as to ensure the original continuous signal can be reproduced perfectly from the sampled measurements.

[2]This mapping is known as Gray coding, named after Frank Gray, an American physicist and researcher at Bell Labs in the early 20th century.

on the axes stand for 'in phase' and 'quadrature', terms that are used in communication engineering for historical reasons. They basically just mean 'real' and 'imaginary' in this context. But, in a physical system, the 'in phase' component of a signal is modulated with a cosine wave at a desired carrier frequency and the 'quadrature' component is modulated with a sine wave, and the resulting electrical waveforms are added together before being transmitted over the communication channel to the receiver. The constellation shown in Fig. 4 is known as 16-QAM, where 'QAM' stands for *quadrature amplitude modulation*. If you own a wireless router or a smart phone, then you probably use this constellation every day; it's a rather popular one.

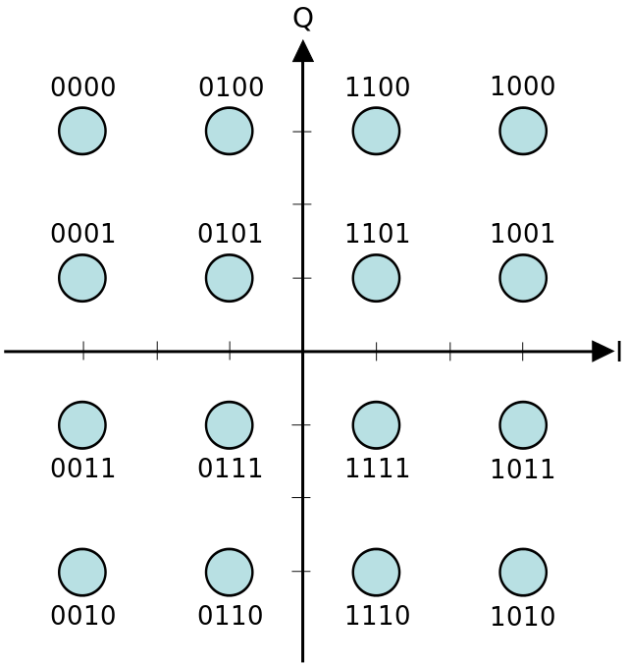


Figure 4: 16-QAM constellation and corresponding bit mapping.

Suppose we map a string of bits to constellation symbols. Let us denote the ith such symbol as $d[i]$. The notation 'd' is used here to signify that we are talking about 'data'. We could parse a group of these symbols into a vector of, say, N symbols if we wish. Naturally, we may have a lot of data, and hence a lot of vectors. Mathematically, we can write the nth such block as the vector

$$\mathbf{d}_n = (d[nN], \ldots, d[N-1+nN])^T \tag{1}$$

where the superscript $(\cdot)^T$ denotes the transpose operator (i.e., rows become

columns and vice versa). Grouping data symbols in this manner is typical of modern communication systems, and carries many theoretical and practical advantages. Perhaps the most important advantage that we will make use of in this project is the fact that block-based windowing techniques can be employed. Indeed, if we now assume that $\mathbf{d}_n$ is the input to a window function, with $\mathbf{s}_n$ denoting the output, then we have defined a simple system in the manner discussed above (see Fig. 5). This system will be the focus of the rest of the project.

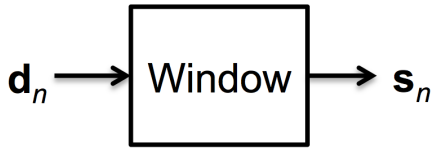


Figure 5: Block diagram representing a part of a block-based communication system with a window function.

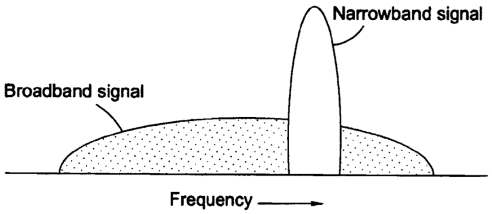
1.3 This Project

Task 1:
Read section I of the paper.

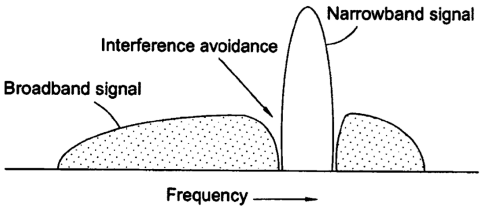
Imagine you are a communications engineer working on a signalling scheme called *ultra-wideband communication*. You know that this interesting communication paradigm possesses many useful advantages, such as the potential to achieve very high communication rates over a short range. However, you also know that these benefits are a result of the very wide bandwidth that the technology utilises, which means that ultra-wideband signals will inevitably interfere with other concurrent transmissions with narrower bandwidth, such as Wi-Fi. Fig. 6a depicts an example of this sort of interference scenario.

Your group leader has charged you with designing a method that can be used to condition an ultra-wideband signal such that interference to other concurrent narrowband transmissions is avoided (see Fig. 6b). You are told that the design must adhere to the following constraints:

1. The conditioning method must take the form of a scalar multiplication of each symbol in the baseband data vector $\mathbf{d}$ by an optimised coefficient.



(a) Illustration of interference between an ultra-wideband signal and a narrowband signal such as Wi-Fi.



(b) Depiction of the interference avoidance result that you hope to achieve with your design.

Figure 6

Mathematically, this operation can be written as

$$\mathbf{s} = \mathbf{D}\mathbf{x} \tag{2}$$

where $\mathbf{D}$ is a diagonal matrix with the elements of $\mathbf{d}$ on the diagonal and zeros elsewhere, $\mathbf{x}$ is a length-N column vector of optimised coefficients and $\mathbf{s}$ is the conditioned signal.

2. The so-called *window vector* $\mathbf{x}$ must not dramatically alter the phase and amplitude characteristics of the data symbols in $\mathbf{d}$ lest the information encoded in $\mathbf{d}$ be lost.

3. The total energy of the signal vector $\mathbf{s}$, represented by the squared norm of the vector, must be limited (i.e., the elements of $\mathbf{x}$ must not be allowed to get too big).

We will return to these constraints later. Additionally, your group leader would like to see a design that

1. yields a deep notch in the desired location of the spectrum of $\mathbf{s}$, and

2. is not too complicated to compute in real time since it will potentially have to be done with each new block of data symbols $\mathbf{d}$.

During this project, you will be guided through the world of Matlab and convex optimisation theory as you strive to complete the aforementioned assignment. In fact, you will come close to solving the problem early on in the project, but will see that the result is unsatisfying for several reasons, and thus you will then spend

time revising your method and searching for new and better ways of achieving the goal that has been set for you. The pace will start slowly, ramping up as we go along. You will complete a number of assigned tasks along the way. These will form building blocks, which you will use to construct more and more intricate, optimal and suitable designs.

1.4 Your Report

Your group leader is keen to understand what you learn through your assignment, particularly as it applies to the optimal design of the so-called *window vector*, and has thus requested that you write a report giving details of your findings. Your report must be no more than 10 pages. It must read well in itself, and should be regarded as a mathematical and computational technical annexe to a larger ultra-wideband communication research and design project. It must begin with a suitable introduction that describes the scope of your work, and it must end with a section giving full details of your conclusions. You are *not* expected to discuss ultra-wideband communication at any length in your report. The goal of the report is to inform your group leader of design techniques, and in particular various aspects of optimisation theory and application. Guidance on specifically what to include in your report is given with the task descriptions in this set of project notes.

2 Preliminaries

2.1 Matlab Revision and Test Signal Construction

Before delving into the main task of designing window vectors according to the specifications outlined above, it would be wise to revise a bit of Matlab coding. In this section, we will get to grips with the basic Matlab programming structures whilst exploring some of the constituent components of the system that we will focus upon in this project.

2.1.1 Matlab

We will assume a basic knowledge of Matlab syntax; see the following URL for a refresher.

http://www.mathworks.co.uk/help/matlab/index.html

It is worth spending a few moments (re-)familiarising yourself with the Matlab user interface and programming environment. Here is a basic explanation of some of the different windows and functions you may find useful in Matlab:

Command window: The *command window* is where run-time commands can be entered. This environment is useful for probing workspace variables, executing quick calculations and running programmes.

Workspace: The *workspace* window displays all of the variables that are currently defined in the global environment. Variables defined within functions and not passed outside are not shown here (unless you are running a debugging session - see below).

Command history: The *command history* window does what it says on the tin: it shows you what you have recently typed into the command prompt.

Editor: The *editor* is where you will do most of your more intricate programming, including writing scripts and functions that you may wish to call from the command window later. You can save code written in the editor as a file with the extension '.m'.

Debugging: You need to familiarise yourself with Matlab's debugging functionality. This is very simple. The basic process is to set breakpoints in your code in the editor window (this can be done using shortcut buttons or the text menus), then run the code. You will then enter a run-time environment, but where a single line of code will be executed at a time. You can step through code in this manner, view the workspace, plot intermediate results, reassign values to variables, pretty much anything you want. This functionality is incredibly useful for quickly and efficiently identifying and correcting errors in the code.

Help: If you need more information about an in-built Matlab function or process, use the online help tools. Help can be accessed in the usual way through the text menus at the top of the screen (or sometimes by hitting the F1 key). Alternatively, you can quickly get some basic information about a function by typing 'help' at the command prompt (e.g., `help plot`).

2.1.2 Generating Random Data Symbols

To begin with, let's generate some QPSK data and store this as a length-N vector. We can jump straight to constellation symbols here since the window we wish to design will operate at this level; it does not operate at the bit level, so we can abstract the bit-to-symbol mapping function accordingly[3]. What steps do we need to take? First, we need a vector of constellation symbols. Run the following code at the Matlab command prompt and observe the variable `d`.

```
% QPSK constellation
const_sym = [1+1i 1-1i -1+1i -1-1i];
M = length(const_sym);  % Get the number of constellation points

% Set the number of constellation symbols
N = 10;

% Initialise vector for storing symbols
d = zeros(N,1);

% Randomly assign constellation symbols to vector
```

[3]Of course, the bit-to-symbol mapping naturally has to be one-to-one in a practical system. Otherwise, the communication receiver would not be able to decode the message without ambiguity.

```
for i = 1:N
    % 1st argument of randi: maximum integer in range
    % 2nd argument: 1 specifies a scalar output
    sym_index = randi(M,1);
    d(i) = const_sym(sym_index);
end
```

What does the code do? Let's go through it in detail.

- In the first section of code, we manually define the constellation symbol set and use the built-in Matlab `length` function to get the number of elements in the set of constellation points.
- In the second section, we define the number of symbols we want our vector to have.
- The third section creates space in memory for `N` symbols. This is good coding practice, but not strictly necessary in Matlab. However, if it isn't done, there is a risk of the code running slowly due to dynamic memory allocation.
- The last section of code uses a `for` loop to sequentially assign a random constellation symbol to each element of the initialised column vector `d`. For each iteration of the `for` loop, the built-in `randi` function is called, which returns an integer randomly (and uniformly) chosen from the set $\{1, 2, ..., M\}$. This integer is directly assigned to the variable called `sym_index` through the equality. In the next line, the integer just generated is used to index the set of constellation symbols; a symbol is selected and assigned to the `i`th element of the vector `d`.
- Finally, we note the good practice of using comments throughout this code, which are always prefaced with a `%` symbol.

This simple example serves many purposes. It illustrates how variable assignments are made, how `for` loops work, and it introduces the `randi` function. But we don't necessarily want to type everything into the command prompt each time we run the code. Furthermore, as a rule, `for` loops are slow; there must be a way to improve the execution time of the code, which would be noticeably long

if N were large (on the order of tens of thousands). We can deal with the first issue by writing this code as a *function* and storing it in a .m file. The second issue is alleviated by exploiting the inherently vector-based nature of Matlab and its built-in functions. Have a look at the code below, which you will find in the file named get_sym.m located in ./Code/Preliminaries.

```
function [d,const_sym] = get_sym(N,const_str)
%GET_SYM
%
%Usage: [d,const_sym] = get_sym(N,const_str)
%
%Summary: The get_sym function generates a length N
%         row vector of constellation symbols.
%
% inputs:
%   N           Number of constellation symbols
%   const_str   A string indicating the constellation
%
% outputs:
%   d           N-tuple of random constellation symbols
%   const_sym   set of possible constellation symbols
%
% (c) jpc 13.06.14

% Check to see if the requested constellation is QPSK
if ~strcmpi(const_str,'qpsk')
    error('Only QPSK is supported.');
end

% QPSK constellation
const_sym = [1+1i 1-1i -1+1i -1-1i];
M = length(const_sym);  % Get the number of constellation points

% Randomly assign constellation symbols to vector
sym_indices = randi(M,[1 N]);  % Note the additional argument
d = const_sym(sym_indices);    % Directly construct the N-tuple

end
```

Notice the differences between this code and the previous code we ran. Apart from a bit of error checking and more extensive comments, the code is much

simpler. You should familiarise yourself with this code, particularly the `function` syntax and `N`-fold assignment to `d` in the last executable line (without initialisation of `d` - why is this OK?). Use Matlab's online help to fill in the gaps in your understanding.

2.1.3 Plotting

It is often useful to be able to visualise the data we generate. We can plot the constellation returned from the function `get_sym` with the code given below. Try it. The result is shown in Fig. 7. You should see something similar when you run the code.

```
% Plot the constellation points
figure                                          % New fig
plot(real(const_sym),imag(const_sym),'ko')      % Plot points
xlabel('Real part')                             % x-axis label
ylabel('Imaginary part')                        % y-axis label
title('QPSK Constellation Diagram')             % Title
axis([-1.5 1.5 -1.5 1.5])                       % Set axes
axis square                                     % Square axes
grid on                                         % Turn grid lines on

% Save the figure
saveas(gcf,'qpsk_plot','fig')
```

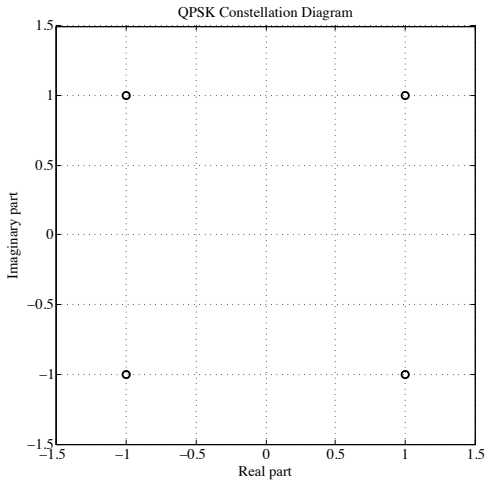


Figure 7: QPSK constellation generated from Matlab code.

Notice the labelling, axis customisation and saving features of this code. By writing scripts and functions to include this information, we can automate many mundane operations and save a lot of time.

2.1.4 Test Signal Function

Now, what if we want to generate symbols from another constellation? Do we need to write an entirely new function? No, of course not. We can just assign a different string to the input `const_str`. Of course, there is more to it than this, as you will find out when you work through the following task.

Task 2:
Modify the `get_sym` Matlab function so that it supports the following:

- Either QPSK or 16-QAM can be specified as the `const_str` input.
- The output `d` will be a *column* vector of randomly generated QPSK or 16-QAM symbols, according to the specified input.
- The complete set of constellation points is plotted to the screen and saved in a file called `const_plot.fig`.

The nominal 16-QAM constellation is given by the set of complex points with real values $\{\pm 1, \pm 3\}$ and imaginary values $\{\pm i, \pm 3i\}$, i.e., the constellation is made up of the 16 complex points generated by all possible combinations of these values.

Hints and tips: Try using an `if` or `switch` statement when making a comparison with the `const_str` string. Also, the task asks for a *column* vector output. You may benefit from brushing up on Matlab's built in array manipulation methods. Try searching for 'Sorting and Reshaping Arrays' in Matlab's online help.

For your report: Document major innovative sections of code and interesting observations related to this task in your report.

2.2 Signal Analysis

Now that we have a function that generates a test signal, either in the form of QPSK symbols or 16-QAM symbols, we can perform some basic signal analysis. Recall from section I of the paper that our ultimate task is to design a function that can be applied to a time-domain signal to produce a deep notch in the frequency domain at a prescribed location. It stands to reason that we will need to develop some tools that enable us to observe the spectral properties of signals.

2.2.1 Fourier Transforms

Fourier transform theory seems like a good place to start. But Fourier theory is particularly rich, offering techniques that can be employed to study continuous-time signals, discrete-time signals with continuous frequency-domain representations, periodic signals...which should we use? One clue arises from the fact that we have so far discussed a baseband communication system model, whereby signals are represented as arrays of discrete complex numbers. Thus, we will require a transform that operates on a discrete sequence. To gain the remaining clues, we need to know a bit more about the system we are working with.

Task 3:
Read section II of the paper, paying particularly close attention to the passages related to frequency-domain representations of signals. Don't worry if you are confused by some of the communication related jargon.

Many of the details contained in section II of the paper are extraneous for the purposes of this project. However, the text contains some important information. In particular, we note that each precoded data vector **s** will be appended with zeros. Although, as the paper points out, we don't need to model these in our system, the zeros are key since they create the illusion of periodicity in the communication signal. The result is that we can employ the *discrete Fourier transform* (DFT) – which inherently operates on a finite or periodic sequence – to study the spectral properties of the signal. The *normalised* DFT of a length-N

sequence $x_0, \ldots, x_{N-1}$ is mathematically given by

$$\tilde{x}_k = \frac{1}{\sqrt{N}} \sum_{m=0}^{N-1} x_m e^{-i2\pi \frac{mk}{N}} \tag{3}$$

with the normalisation coming from the $1/\sqrt{N}$ prefactor. In matrix form, the transform can be written as

$$\tilde{\mathbf{x}} = \mathbf{F}\mathbf{x} \tag{4}$$

where the (k, m)th element of the $N \times N$ matrix $\mathbf{F}$ is given by

$$F_{k,m} = \frac{1}{\sqrt{N}} e^{-i2\pi \frac{mk}{N}} \tag{5}$$

and $\mathbf{x} = (x_0, \ldots, x_{N-1})^T$ with $\tilde{\mathbf{x}}$ defined similarly.

It is important to understand that the DFT is invertible. The normalised inverse DFT (IDFT) of a length-N frequence-domain sequence $\tilde{x}_0, \ldots, \tilde{x}_{N-1}$ is given by

$$x_m = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \tilde{x}_k e^{i2\pi \frac{mk}{N}}. \tag{6}$$

In matrix form, the inverse operation is just given by

$$\mathbf{x} = \mathbf{F}^{-1}\tilde{\mathbf{x}} = \mathbf{F}^H\tilde{\mathbf{x}} \tag{7}$$

where the notation $(\cdot)^H$ denotes the *conjugate* transpose operation. The elegant equality $\mathbf{F}^{-1} = \mathbf{F}^H$ further justifies the normalisation of the DFT operators. Without normalisation, this equality would not hold.

2.2.2 Spectral Analysis

Now we can perform some basic spectral analysis on our test signal. To start with, let's generate a test signal and use the DFT to view its spectrum. You can use the code given below to achieve this task. You will find the code in the file `fd_analysis.m`, which is located in the ./Code/Preliminaries directory[4].

[4]The full usage comments have been omitted from the code printed here for brevity, but you will find them included in the function supplied in this directory.

```
function [df,const_sym] = fd_analysis(N,const_str)
%FD_ANALYSIS
%
%Usage: ...

% Generate test signal
[d,const_sym] = get_sym(N,const_str);

% Take the normalised DFT
df = 1/sqrt(N)*fft(d);

% Plot the freuqency-domain envelope (sample-spaced)
figure                              % New fig
plot(1:N,abs(df),'k')               % Plot interpolated points
xlabel('Sample number')             % x-axis label
ylabel('Magnitude of DFT(d)')       % y-axis label
title('Magnitude of DFT(d)')        % Title
a = axis;                           % Get axis limits
axis([1 N a(3) a(4)])               % Set axis limits
grid on                             % Turn grid lines on

end
```

It is worth noting a few things about the code itself before discussing the output. First, we see that there are three main sections to the code, which correspond to the generation of a test signal, execution of the DFT operation on that signal, then graphical illustration of the results. The code has been built on our previous work, i.e., it is *modular*. This is good programming practice since it helps with debugging and scaling of the code to larger programming tasks. For example, if we suspect our modular programme contains an error, we can usually pinpoint it to a single function (also known as a *subroutine* in programming parlance). This localises the problem, and it can often be fixed easily without having knock-on effects on other subroutines or sections of code.

Notice that the DFT is performed easily in this function using Matlab's built-in `fft` function[5]. You should view the online help file for this function to understand the different input and output arguments. This is a very useful and efficient way of performing DFTs, but we will see in the next section that it cannot always be used.

[5] 'FFT' is short for 'fast Fourier transform'. It is an efficient digital implementation of the DFT.

Another built-in function that has been introduced in the code detailed above is the `abs` function. In short, this is the *modulus* or *absolute value* function. Also, a new use of the `axis` function is shown. Have a look at the online help for details.

A typical plot that might be generated by this code is illustrated in Fig. 8. The inputs that were used to generate this plot were `N=64` and `const_str='qpsk'`. Notice that only 64 data points were generated for the frequency response, but we have opted to graphically interpolate between these data points in the plot since the real world is not sampled at discrete points, i.e., the actual signal used in the communication system will be continuous valued.

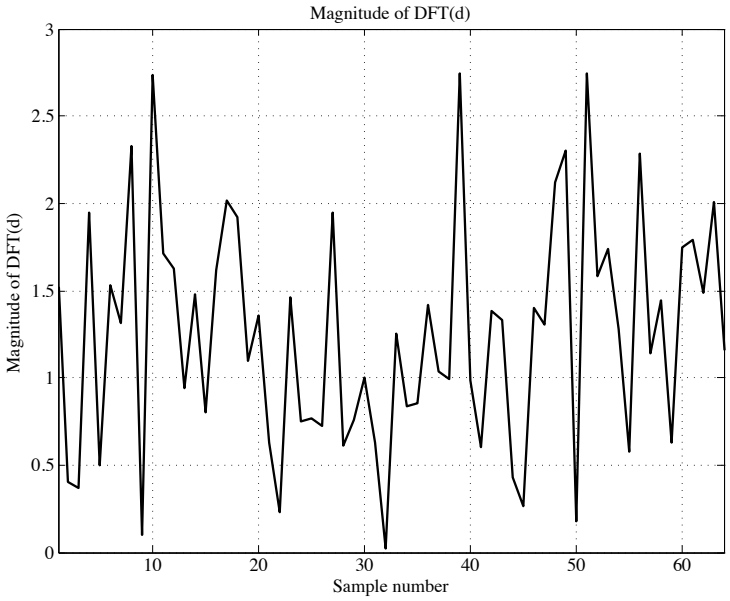


Figure 8: Typical graphical output of the `fd_analysis.m` function with inputs `N=64` and `const_str='qpsk'`.

2.2.3 Representation in Decibels

Communications engineers like to observe real-valued signal powers in units of *decibels*[6]. This is a simple logarithmic unit that provides a measure of the ratio of two physical quantities, each denoting a *power* signal. One of the quantities

[6]One *decibel* is equal to a tenth of a *bel*, which was named in honour of Alexander Graham Bell, who is credited as being one of the founders of modern communications.

is often taken to be a reference value (for example, one watt), in which case a signal measurement expressed in decibels represents the absolute value of the measurement.

The simple logarithmic relationship between signals expressed in standard 'linear' units and decibels is given by

$$x_{db} = 10 \log_{10}(x_{lin}/x_{ref}) \tag{8}$$

where x_{lin} represents a real-valued signal expressed as a power measurement, x_{ref} denotes the reference and x_{db} is the relative measurement expressed in decibels. Often, when normalising signals (as was done above using the normalised DFT operation), the reference is taken to be one unit of power. Thus, eq. (8) becomes

$$x_{db} = 10 \log_{10}(x_{lin}). \tag{9}$$

Students frequently get confused between *magnitude* and *power* when expressing signal levels in decibels. A typical mistake made in practice is to express a voltage level in decibels using eq. (9). However, a voltage signal is not measured in units of power. Remember that

$$\text{power} = \frac{\text{voltage}^2}{\text{resistance}}$$

and thus the voltage signal must be squared before eq. (9) can be used in this context. Alternatively, the prefactor of 10 in eq. (9) can be replaced by a prefactor of 20.

Armed with this knowledge, we can replace a few lines of the plotting code in the `fd_analysis.m` function with the code given in the box below to illustrate the signal power in decibels. Try this. You should see something that resembles Fig. 9. Since converting signal measurements between linear and logithmic (decibel) units is a common task in communications engineering, you may find it prudent to write a small function that does this for you.

```
...
plot(1:N,20*log10(abs(df)),'k')    % Plot interpolated points
...
ylabel('Power of DFT(d) (dB)')     % y-axis label
title('Power of DFT(d) (dB)')      % Title
...
```

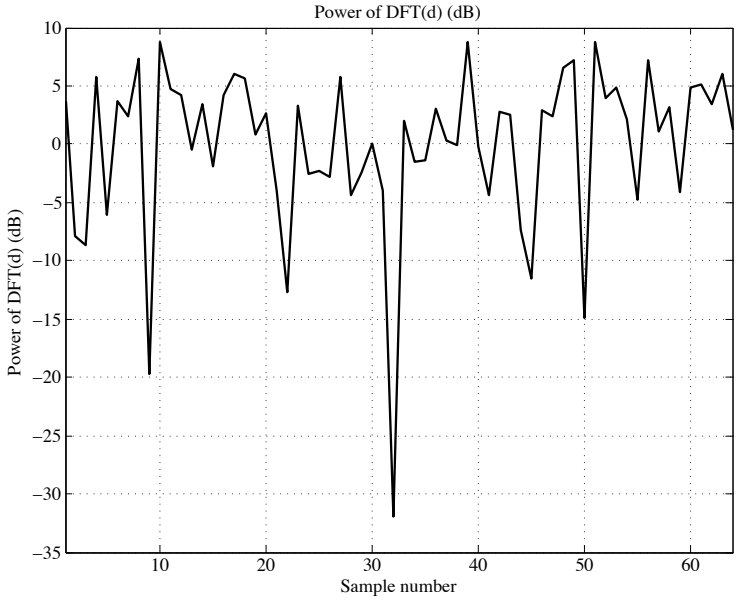


Figure 9: The same output as shown in Fig. 8 plotted as a power in units of decibels (with a reference value of one).

2.3 A Naive Approach

It seems that we are in a position to construct a transfer function, or *window* as it is called in section II of the paper, that could achieve our goal of creating a deep notch in the data signal. Indeed, now that we are able to transform the input data vector **d** into the frequency domain, it stands to reason that we can simply set the appropriate frequency components of the DFT of **d** to zero and subsequently transform this modified signal back into the time domain with an IDFT. This crude and potentially inefficient way of filtering the signal is exactly what is accomplished by the function `notch_data_v1.m`, which is included in the ./Code/Preliminaries directory. The main sections of this code are included below for convenience (some comments that can be found in the file have been omitted). The final task of this section will explore the functionality and output of the `notch_data_v1.m` function.

```
function [s,sf,const_sym] = notch_data_v1(N,const_str,indices)
%...

% Error checking - make sure indices are within range
if min(indices) < 1 || max(indices) > N
```

```
    error('Indices are outside of range.')
end

% Generate test signal
[d,const_sym] = get_sym(N,const_str);

% Take the normalised DFT
df = 1/sqrt(N)*fft(d);

% Apply a notch at the appropriate indices (frequencies)
% Assign to output variable sf
df(indices) = 0;
sf = df;

% Transform back into the time domain
s = sqrt(N)*ifft(sf);

% Plot the magnitude of DFT(s) (linear)
figure                                      % New fig
plot(1:N,abs(sf),'k')                       % Plot
xlabel('Sample number')                     % x-axis label
ylabel('Magnitude of DFT(s)')               % y-axis label
title('Notched signal frequency response')  % Title
a = axis;                                   % Get axis limits
axis([1 N a(3) a(4)])                       % Set axis limits
grid on                                     % Turn grid lines on

% Plot real/imag parts of time-domain signal s
figure                                      % New fig
plot(real(s),imag(s),'ko')                  % Plot
xlabel('Real part')                         % x-axis label
ylabel('Imaginary part')                    % y-axis label
title('Transformed signal constellation')   % Title
grid on                                     % Turn grid lines on

end
```

Task 4:
Review and understand the code in the function `notch_data_v1.m`. Run the function with the input `const_str='qpsk'` and different combinations of the following arguments:

N	indices
16	[9]
64	[9:12]
	[9:15]

You may wish to try other combinations. You should notice that some combinations yield transformed data that would be unsuitable for transmission through a communication system. Why?

For your report: Comment on your observations. Try to propose an intuitive explanation for what this small, numerical, exploratory exercise shows. Clearly, your report should not contain all of the figures that you generate in this task, but you may wish to include one or two if it will aid your explanation.

2.4 Summary

We have come to the end of the initial phase of the project. In this part, we

- (re-)introduced the Matlab programming environment and some basic functionality;
- explored coding techniques that can be used to generate a random set of data symbols, which are drawn from a QPSK constellation;
- learned how to plot data in Matlab and save the resulting figures to the current working directory;
- constructed a Matlab function that generates either QPSK or 16-QAM data symbols according to a specified input;
- revisited Fourier transforms, paying particular attention to DFTs and IDFTs;

- investigated the spectrum of a test signal and plotted its magnitude in 'linear' units and decibels;
- made a first attempt at designing a function that creates a notch in the data spectrum at a desired location, but found that the output data is unsuitable for transmission in a communication system.

This part of the project was very much a tutorial. As we venture into the more intricate (and interesting!) parts of the project, the helping hand that was available during this first phase will largely be removed. By the end of the project, you will be on your own.

3 Optimisation

Recall that in the last task, you found that a simple, naive nulling approach cannot be used to create a notch in the data spectrum without significantly altering the information content of the message. In this section, we will start to explore ways that we can mitigate this problem by imposing constraints on the transfer function such that it does not catastrophically compromise the amplitude and phase information of the original message vector **d**.

Task 5:
Read section III.A of the paper.

3.1 Upsampling

The first thing that we notice from our perusal of section III.A is that, contrary to our investigations thus far, we will need to *upsample* the discrete data so that we can obtain a reasonably accurate estimate of the depth of the notch when the signal is continuous, as is the case in practice. The process of upsampling a digital signal is akin to interpolating between the original discrete points. In the time domain, this is done by inserting $\rho - 1$ zeros after each element of the original sequence, with ρ denoting the upsampling factor, then filtering the result. Alternatively, to upsample the spectrum of a sequence, DFT interpolation/upsampling is typically used. In our case, this is done by appending $(\rho - 1)N$ zeros to the end of the length-N vector **d**, then taking the ρN-point DFT of the resulting sequence. (See Fig. 10.)

Why do we append zeros in this manner? This is best explained through an example. Suppose we have a length-4 ($N = 4$) sequence $\mathbf{d} = (d_0, d_1, d_2, d_3)^T$, which we wish to upsample by a factor of 2 ($\rho = 2$). First, let us consider the first two elements of the length-4 DFT of **d**. Using eq. (3), we find that these are given by

$$\begin{aligned} \tilde{d}_0 &= d_0 + d_1 + d_2 + d_3 \\ \tilde{d}_1 &= d_0 + d_1 e^{-i\frac{\pi}{2}} + d_2 e^{-i\pi} + d_3 e^{-i\frac{3\pi}{2}}. \end{aligned} \tag{10}$$

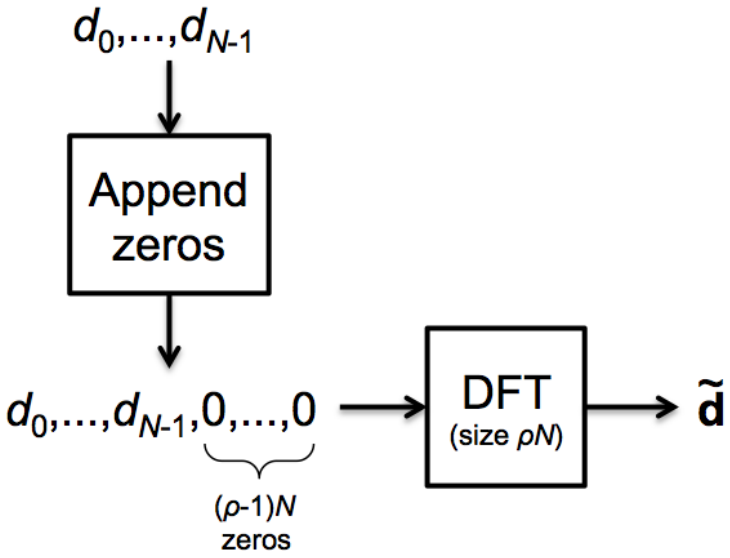


Figure 10: Illustration of process used to upsample spectrum.

Now let's apply the upsampling procedure detailed above. We create the intermediate sequence $\mathbf{d}' = (d_0, d_1, d_2, d_3, 0, 0, 0, 0)^T$, then take the 8-point DFT of this sequence. Consider the first *three* terms of this transform:

$$\begin{aligned}
\tilde{d}'_0 &= d_0 + d_1 + d_2 + d_3 \\
\tilde{d}'_1 &= d_0 + d_1 e^{-i\frac{\pi}{4}} + d_2 e^{-i\frac{\pi}{2}} + d_3 e^{-i\frac{3\pi}{4}} \\
\tilde{d}'_2 &= d_0 + d_1 e^{-i\frac{\pi}{2}} + d_2 e^{-i\pi} + d_3 e^{-i\frac{3\pi}{2}}.
\end{aligned} \tag{11}$$

We see that

$$\tilde{d}'_0 = \tilde{d}_0 \qquad \text{and} \qquad \tilde{d}'_2 = \tilde{d}_1$$

and $\tilde{d}'_1$ interpolates between these points[7]. It is easy to show that this pattern repeats, and also that we obtain a better interpolation for larger ρ.

Let us return to our naive approach to solving the notching problem in order to see the effect that interpolation has on the measurement of the data spectrum. Have a look at the code below. Notice that the Matlab `fft` function supports upsampling. However, in order to view so-called *symbol spaced* and *fractionally*

[7]It is also possible to visualise the interpolation process using the unit circle in the complex plane. In this case, cycling through the exponentials that make up the DFT kernel resembles a clock hand running backwards. For small DFT sizes, the jumps of the hand are large owing to a small value of N in the denominator of the exponential. For larger DFT sizes, the jumps are small. In the limit, the movement of the clock hand is continuous, i.e., interpolation is perfect.

spaced data on the same plot, we have to be careful with indexing with respect to the horizontal. This can be observed in the arguments of the `plot` function. The logic behind the `1:1/r:(N+1)-1/r` x-axis location vector should be clear from the simple example we worked through above.

```
% Script for plotting upsampled spectrum of s

% Define length of sequence and sampling factor
N = 16;
r = 8;

% Run naive notching function
[s,sf,const_sym] = notch_data_v1(N,'qpsk',[9:12]);

% Upsampled DFT (see help for usage)
% Notice the prefactor is still 1/sqrt(N)
sfup = 1/sqrt(N)*fft(s,r*N);

% Plot the magnitude of DFT(s) (dB)
figure                                          % New fig
plot(1:N,abs(sf),'ko',...
    1:1/r:(N+1)-1/r,abs(sfup),'k-')             % Plot
xlabel('Sample number')                         % x-axis label
ylabel('Magnitude of DFT(s)')                   % y-axis label
title('Notched signal frequency response')      % Title
a = axis;                                       % Get axis limits
axis([1 N a(3) a(4)])                           % Set axis limits
grid on                                         % Turn grid lines on
```

Task 6:
Run the code given above. You should see several graphs, the third one resembling Fig. 11. Modify the code to give this result in decibels and rerun the code. There will be discontinuities in the resulting plot owing to the fact that $\log(0) = -\infty$.

For your report: Comment on your observations. Why do these results show that we must use upsampled data when designing a suitable notching function? Include a figure with units in decibels in the report and use this to support your arguments.

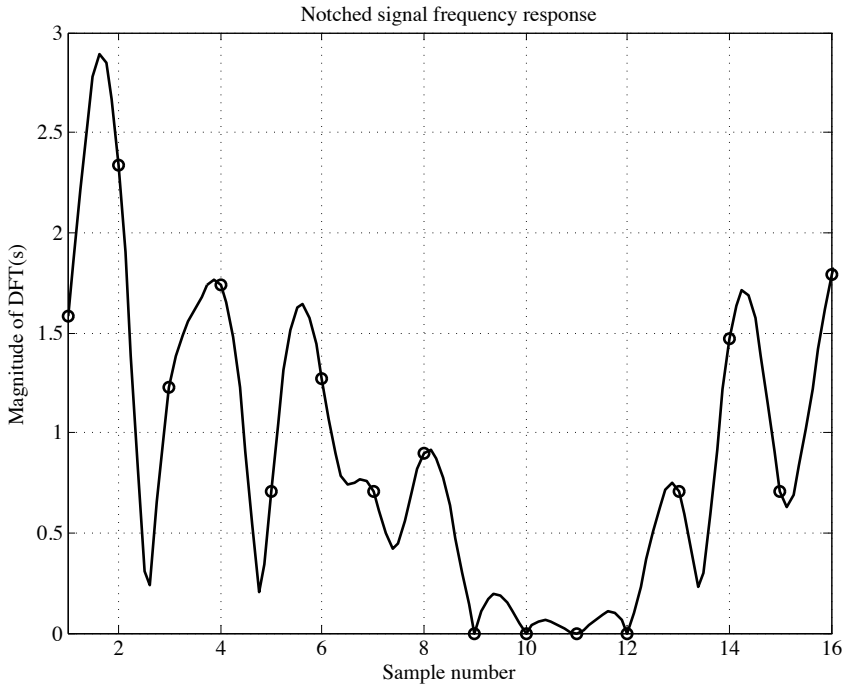


Figure 11: Upsampled spectrum corresponding to a signal processed with the naive notching function described in the previous section. Circle markers relate to *symbol spaced* data.

3.2 A Better Approach: 'Eigen-Window'

Now that we have convinced ourselves of the need for upsampling, we can move on to design a better window function. The approach we will take in this section

is outlined in eq. (4) of the paper and the surrounding text. To solve the problem given in eq. (4), the text alludes to a tool known as *Lagrange multipliers*. We will not dwell on this tool here[8]. Suffice to say that these so-called multipliers can be used to incorporate constraint equations into cost functions, which we wish to minimise. Here, we will briefly go through the calculations omitted from the paper in order to paint a better picture of what is happening with this window design.

After applying the method of Lagrange multipliers to eq. (4) in the paper, we obtain the augmented cost function, or *Lagrangian*

$$L(\mathbf{x}, \lambda) = \|\mathbf{W}_{\mathcal{I}}\mathbf{D}\mathbf{x}\|_2^2 - \lambda(\|\mathbf{D}\mathbf{x}\|_2^2 - N) \tag{12}$$

where we have used the same notation as in the paper: $\mathbf{W}_{\mathcal{I}}$ is a set of rows of a $\rho N \times N$ upsampled DFT matrix that correspond to the indices of the notch, $\mathbf{D}$ is a matrix with the elements of the data vector $\mathbf{d}$ on the main diagonal and zeros elsewhere, $\mathbf{x}$ is a length-N vector of real-valued parameters that define our window (which we want to optimise), and λ is the real-valued Lagrange multiplier. The norm notation $\|\cdot\|_2$ indicates that we are concerned with ℓ^2-norms. To find the minimum of L, differentiate eq. (12) with respect to $\mathbf{x}$ to obtain

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{x}} &= \frac{\partial}{\partial \mathbf{x}} \left\{ \mathbf{x}^T \mathbf{D}^H \mathbf{W}_{\mathcal{I}}^H \mathbf{W}_{\mathcal{I}} \mathbf{D}\mathbf{x} - \lambda \mathbf{x}^T \mathbf{D}^H \mathbf{D}\mathbf{x} + \lambda N \right\} \\
&= \left[\mathbf{D}^H \mathbf{W}_{\mathcal{I}}^H \mathbf{W}_{\mathcal{I}} \mathbf{D} + \left(\mathbf{D}^H \mathbf{W}_{\mathcal{I}}^H \mathbf{W}_{\mathcal{I}} \mathbf{D} \right)^T \right] \mathbf{x} - \lambda \left[\mathbf{D}^H \mathbf{D} + \left(\mathbf{D}^H \mathbf{D} \right)^T \right] \mathbf{x} \\
&= 2\,\mathrm{Re}\left\{ \mathbf{D}^H \mathbf{W}_{\mathcal{I}}^H \mathbf{W}_{\mathcal{I}} \mathbf{D} \right\} \mathbf{x} - 2\lambda \mathbf{D}^H \mathbf{D}\mathbf{x}.
\end{aligned} \tag{13}$$

By setting this equal to zero, we can rearrange the expression to yield

$$\left(\mathbf{D}^H \mathbf{D} \right)^{-1} \mathrm{Re}\left\{ \mathbf{D}^H \mathbf{W}_{\mathcal{I}}^H \mathbf{W}_{\mathcal{I}} \mathbf{D} \right\} \mathbf{x} = \lambda \mathbf{x} \tag{14}$$

which shows us that the multiplier λ is an eigenvalue of the matrix on the left hand side of the equation. Now, using a little trickery, we can rewrite the original

[8] See http://en.wikipedia.org/wiki/Lagrange_multiplier for an overview.

cost function as

$$\begin{aligned}\|\mathbf{W}_{\mathcal{I}}\mathbf{D}\mathbf{x}\|_2^2 &= \frac{1}{2}\left(\|\mathbf{W}_{\mathcal{I}}\mathbf{D}\mathbf{x}\|_2^2 + \|(\mathbf{W}_{\mathcal{I}}\mathbf{D}\mathbf{x})^T\|_2^2\right) \\ &= \frac{1}{2}\left(\mathbf{x}^T\mathbf{D}^H\mathbf{W}_{\mathcal{I}}^H\mathbf{W}_{\mathcal{I}}\mathbf{D}\mathbf{x} + \mathbf{x}^T\left(\mathbf{D}^H\mathbf{W}_{\mathcal{I}}^H\mathbf{W}_{\mathcal{I}}\mathbf{D}\right)^T\mathbf{x}\right) \\ &= \mathbf{x}^T\,\mathrm{Re}\{\mathbf{D}^H\mathbf{W}_{\mathcal{I}}^H\mathbf{W}_{\mathcal{I}}\mathbf{D}\}\,\mathbf{x} \\ &= \lambda\mathbf{x}^T\mathbf{D}^H\mathbf{D}\mathbf{x} \\ &= \lambda N \end{aligned} \tag{15}$$

where the second to last line follows from eq. (14) and the last line is imposed by the constraint in eq. (4) of the paper. Thus, we see that the cost function is minimised, subject to the constraint, when λ is the minimum eigenvalue of the matrix

$$\left(\mathbf{D}^H\mathbf{D}\right)^{-1}\mathrm{Re}\{\mathbf{D}^H\mathbf{W}_{\mathcal{I}}^H\mathbf{W}_{\mathcal{I}}\mathbf{D}\}$$

and the vector $\mathbf{x}$ is the corresponding eigenvector (scaled such that the constraint $\|\mathbf{D}\mathbf{x}\|_2^2 = N$ is met).

For the next task, you will need to navigate to the ./Code/Optimisation directory and locate the file called `eigen_window.m`. Look at the code in this function. You will see that it is incomplete. Your task will be to complete it.

Task 7:
Complete the code in `eigen_window.m` such that the function computes a window according to the eigenvector approach discussed above. As specified in the code comments, the function takes the following as inputs: a data vector $\mathbf{d}$; an upsampling factor ρ; and a vector of indices defining the location of the notch (e.g., `[9:11]`). The function should return as output two plots:

1. the constellation diagram of the windowed data points corresponding to the equation $\mathbf{s} = \mathbf{D}\mathbf{x}$ in the paper;

2. a plot depicting the upsampled spectrum of $\mathbf{s}$ in decibels.

For your report: Include two example plots obtained from your function (clearly state the chosen inputs); comment on your observations.

This is a slightly bigger task than those set before. So here are a few thoughts to help you on your way.

- You will need to define some matrices as discussed in the text surrounding eq. (4) of the paper. Build these up in stages, and look for efficient methods of doing this. It is possible to write the main computational instructions for this task in less than twenty lines of code!
- The length of the vector **d** is not specified. Try different powers of two (e.g., 16, 32, 64, 128, etc).
- Matlab's `eig.m` function might come in handy.
- Remember to make sure the constraint given in eq. (4) of the paper is satisfied when you finally have your window vector!
- Plotting the results should be very similar to what you've already done.

3.3 Reformulating the Problem Once Again

You should have seen in the previous section, and in particular the last task, that although the 'eigen-window' approach is easy to implement and provides a very deep notch in the desired spectral location, it is not suitable for use in our application of interest. (Why? Make sure you discuss this in the report.) The paper suggests a reformulation of the problem stated in eq. (4) therein. The reformulation is given in eq. (5) of the paper, and is repeated below in a slightly different form for convenience.

$$\begin{aligned} \text{minimise} \quad & \|\mathbf{W}_{\mathcal{I}}\mathbf{D}\mathbf{x}\|_2^2 \\ \text{subject to} \quad & \|\mathbf{x}\|_2^2 \leq N \\ & x_i \geq \delta, \;\; i = 0, \ldots, N-1 \\ & \delta \geq 0. \end{aligned} \tag{16}$$

This formulation of the problem has a couple of advantages. By relaxing the equality constraints to inequalities, we are now in the realms of convex optimisation theory[9]. This means we can bring to bear a number of tools to solve the

[9]For the interested reader, the book by Stephen Boyd and Lieven Vandenberghe on the subject is excellent. See http://www.stanford.edu/∼boyd/books.html.

problem, such as interior point methods, which you saw in the B1 lectures on optimisation. Also, we have explicitly limited the attenuation of the window function so that amplitude information is not completely lost from the data signal after the window is applied. Furthermore, as with the 'eigen-window' approach, we have implicitly specified that the window coefficients $x_0, x_1, \ldots$ are real-valued through the second line of constraints. This causes the phase transfer function to be unity, i.e., the output phase is equal to the input phase. Thus, the phase information of the input signal is not altered. The main disadvantage of the formulation given in eq. (16) is that this problem is only useful for constant-modulus constellations such as QPSK. If we used this exact formulation for 16-QAM signals, we might run into practical difficulties (why?). Remember this; we'll come back to it later.

The process of reformulating a problem is a recurring theme in optimisation theory. In fact, the fields of convex and integer optimisation are almost completely based on the premise that once a problem is formulated in a certain way, it is, in effect, solved. This is because standard numerical algorithms can be used to solve well formulated problems efficiently. The purpose of this project, of course, is to delve into a couple of those numerical techniques in the practical setting of communication system design. This is exactly what you will do during the remainder of the project.

3.3.1 The Barrier Method

Task 8:
Read section III.B of the paper.

This is a key passage in the paper. In this section, all of the details that will be required to solve the problem stated in eq. (16) are given. It is noted at the beginning of the section that an interior point method known as the *barrier method* is well suited to this problem. Recall that you were introduced to the barrier method in the B1 lectures on optimisation. We will briefly review the technique here.

The barrier method is based on the idea that one can augment the cost function with indicator functions of constraints, which diverge when constraints are broken and are approximately zero otherwise. These functions act as *barriers* to the set of solutions to the optimisation problem. If $f_0(\mathbf{x})$ is a cost function that

we wish to minimise and $\{f_i(\mathbf{x})\}$ are a set of constraints that must be nonpositive for $i = 1, \ldots, p$, then this formulation would look like

$$\text{minimise} \quad f_0(\mathbf{x}) + \sum_{i=1}^{p} I(f_i(\mathbf{x})) \tag{17}$$

with

$$I(u) = \begin{cases} 0, & u \leq 0 \\ \infty, & u > 0 \end{cases} \tag{18}$$

denoting the indicator function.

By reformulating the problem in this way, the constrained optimisation problem becomes an unconstrained one. However, a problem remains. To solve the new problem, one would typically differentiate the cost function and set the result equal to zero to find the stationary point(s). Alternatively, a numerical algorithm such as Newton's method would be employed to find a local minimum (you saw Newton's method in the B1 lectures as well!). But indicator functions are not differentiable everywhere in their domain; hence, we need to come up with an approximation to such a function that *is* differentiable. It turns out the logarithm makes a decent approximation:

$$I(u) \approx -\frac{1}{t} \log(-u).$$

The parameter $t > 0$ sets the accuracy of the approximation; a larger value of t gives a better approximation. Note that the base of the logarithm doesn't matter.

A very nice outline of the barrier method is provided in Table I of the paper (this table almost looks like code!), which is reproduced in Fig. 12 for convenience. We see from this table that the idea is to set the value of t, then find the vector $\mathbf{x}$ that minimises the unconstrained problem given in eq. (17) before increasing t a bit and doing this again. We stop when t passes a predetermined threshold, which can largely be chosen heuristically to give a good result (measured rigorously, of course). Why solve these subproblems sequentially? Why not set t to be large from the outset and solve one problem? Basically, the answer lies in the way we find the optimal vector for the unconstrained problem, which is typically done using Newton's method. If t is very large, rapid variations in the second derivative of the augmented cost function near the boundaries of the admissible, or *feasible*, set of solutions for $\mathbf{x}$ cause Newton's method to converge very slowly, thus rendering the entire approach inefficient.

given strictly feasible $\mathbf{x}$, $t > 0$, $\mu > 1$, tolerance $\epsilon_o > 0$, and p constraints
repeat
1. compute the optimal vector $\mathbf{x}^\star(t)$ for the current value of t beginning at $\mathbf{x}$
2. $\mathbf{x} := \mathbf{x}^\star(t)$
3. quit if $p/t < \epsilon_o$
4. $t := \mu t$

Figure 12: The barrier method.

3.3.2 Newton's Method

In order to implement the barrier method, we must back-pedal a bit to explore ways to solve the unconstrained problem (see the first line of Table I in the paper). Newton's method is an obvious choice, but you have come across others in the B1 lectures. Newton's method is outlined in Table II of the paper, which is reproduced in Fig. 13 for convenience.

You should be familiar with the algorithm from your lectures, so we will not re-derive it here. However, a few things should be noted about the algorithm as it is written in Table II:

- $\nabla f(\mathbf{x})$ denotes the gradient of the augmented cost function;
- $\nabla^2 f(\mathbf{x})$ is the Hessian of the augmented cost function;
- the stopping criterion in line 2 of the algorithm is chosen such that the algorithm quits when the slope of the cost function becomes small, which indicates that we might be close to a stationary point.

We also note that, in line 3 of the algorithm, a *line search* is performed, but the specific method is left to implementation. The main purpose of performing a line search is to specify the magnitude of the update on our estimate of the optimal point $\mathbf{x}^\star$. Indeed, we could set $\theta = 1$ in the algorithm (or any other nonnegative constant for that matter), but we might frequently overshoot or undershoot the optimum as we iterate the algorithm. Solving for $\Delta\mathbf{x}$ in line 1 of the algorithm gives us the magnitude and direction we should travel to find a stationary point of the *local quadratic approximation*, but the line search refines our guess a bit by taking into account the *actual* function we have locally approximated.

given a feasible starting point $\mathbf{x}$, tolerance $\epsilon_i > 0$
repeat
1. $\Delta\mathbf{x} = -\nabla^2 f(\mathbf{x})^{-1} \nabla f(\mathbf{x})$
   $\omega^2 = -\nabla f(\mathbf{x})^T \Delta\mathbf{x}$
2. quit if $\omega^2/2 < \epsilon_i$
   return $\mathbf{x}^\star := \mathbf{x}$
3. line search (determine θ)
4. $\mathbf{x} := \mathbf{x} + \theta\Delta\mathbf{x}$

Figure 13: Newton's method.

Different line search methods exist. Here, we will focus on the exact line search and the backtracking line search in order to get across the main idea. You will later be asked to implement a line search technique; the specific method is left up to you to research and choose. The exact line search seeks the parameter θ that follows the cost function f along the ray $\mathbf{x} + \theta\Delta\mathbf{x}$ to its minimum, i.e.,

$$\theta = \arg\min_{s \geq 0} f(\mathbf{x} + s\Delta\mathbf{x}). \tag{19}$$

One problem with this approach is that it involves an optimisation problem in itself. Sometimes, this can be solved explicitly and/or efficiently, but this is not always the case.

The backtracking line search gets around this problem by iteratively trying different parameters θ in a structured way until a 'good' value is found. We will refrain from going into the underlying theory of this method, but have included the algorithm below for convenience.

Table 1: Backtracking line search.

given descent direction $\Delta\mathbf{x}$ for f and parameters $\alpha \in (0, 0.5)$, $\beta \in (0, 1)$: $\theta := 1$
while $f(\mathbf{x} + \theta\Delta\mathbf{x}) > f(\mathbf{x}) + \alpha\theta\nabla f(\mathbf{x})^T\Delta\mathbf{x}$, $\quad \theta := \beta\theta$.

3.3.3 Complete Algorithm: Major Task

Task 9:
Write a programme called `barrier_window.m` that uses the barrier method to numerically calculate the optimal vector **x** according to the problem formulation given in eq. (16). Use Newton's method in each iteration of the barrier method. As with Task 7, the function should take the following as inputs:

1. a data vector **d**;
2. an upsampling factor ρ;
3. a vector of indices defining the location of the notch (e.g., `[9:11]`).

The function should return as output two plots:

1. the constellation diagram of the windowed data points corresponding to the equation $\mathbf{s} = \mathbf{D}\mathbf{x}$ in the paper;
2. a plot depicting the upsampled spectrum of **s** in decibels.

For your report: Include key sections of your code and two example plots obtained from your function (clearly state the chosen inputs); comment on your methodology and observations.

This is a major task. You may wish to consider the following as you work to complete it:

- As intimated by the tables outlining the Newton and barrier methods, you may wish to design your code in a modular fashion. For example, `barrier_window.m` could be a 'wrapper function', which calls a function that executes Newton's method for given inputs with each iteration of the barrier method.
- The choice of the line search technique that is used within Newton's method is completely up to you. Your lecture notes, relevant texts and online resources may be useful. But note that iterative methods may require

careful selection of the initial scaling parameter θ so as not to violate the constraints $x_i \geq \delta \ \forall i$. In other words, $\theta \ll 1$ may be required from the outset.

- It is stated in Fig. 12 that you will need a feasible starting vector $\mathbf{x}$. This just means you must initiate the algorithm with a vector that satisfies the constraints of the problem stated in eq. (16).
- The length of $\mathbf{d}$ has not been specified. Try different powers of two.
- Loops will be your friends during this task. Read the Matlab help files on different loop structures, such as `for` and `while` loops.
- The values of several parameters in the barrier and Newton algorithms described above are not specified (e.g., μ, ϵ_o, ϵ_i). You may consider making these inputs to your function so that you can easily explore the effect that choosing different values has on the output. In fact, we will return to this in the next section.
- You will need to calculate the gradients and Hessian matrices for the cost function f_0 and the constraint functions g_k for $k = 1, 2, \ldots, N+1$ (see section III.B in the paper). For example,

$$f_0(\mathbf{x}) = \mathbf{x}^T\mathbf{D}^H\mathbf{W}_{\mathcal{I}}^H\mathbf{W}_{\mathcal{I}}\mathbf{D}\mathbf{x} \quad \Rightarrow \quad \nabla f_0(\mathbf{x}) = 2\,\mathrm{Re}\{\mathbf{D}^H\mathbf{W}_{\mathcal{I}}^H\mathbf{W}_{\mathcal{I}}\mathbf{D}\}\mathbf{x}$$

and

$$g_1(\mathbf{x}) = -\frac{1}{t}\ln(N - \mathbf{x}^T\mathbf{x}) \quad \Rightarrow \quad \nabla g_1(\mathbf{x}) = \frac{2}{t(N - \mathbf{x}^T\mathbf{x})}\mathbf{x}.$$

Once you have done this, it may be worth writing different `.m` functions for each mathematical function so that you can simply call these at the appropriate points in the Newton-barrier programme.

3.4 Summary

In this section, we

- introduced the notion of upsampling and explored the motivation for doing so in practice;

- solved the windowing problem using the method of Lagrange multipliers for a particular problem formulation and studied the benefits and drawbacks of this approach;
- reformulated the optimisation problem to be convex, and wrote code that solved the problem using the barrier and Newton methods.

The code that you have written and the results that you have generated will play key roles later in the project, where we will extend the current ideas to a slightly more complicated scenario.

4 Convergence and Refinements

In this section, you will study the convergence of the basic algorithm that you implemented in the last task. You will also use the Matlab time keeping functions to study the computational time required to execute your algorithm. You will then use this analysis to propose improvements to the method, either based on what is reported in the paper or through the use of novel or otherwise unexplored techniques (at least, unexplored thus far in this project).

4.1 Convergence and Run-Time

Task 10:
Read section III.C of the paper.

This section in the paper presents a very loose analysis of the convergence of the Newton-barrier algorithm implemented in the previous section. A tool known as *self-concordance* is used. You do not need to worry about the specifics of the underlying self-concordance analysis here. Suffice to say, the main result of this part of the paper is that the Newton-barrier algorithm *does* converge. This is of practical importance since, as stated in the paper, such an algorithm would have to be run with each transmitted message. Thus, if a transmitter had a lot to send, then the algorithm would be executed many times over. Without performing the self-concordance analysis, it would not be possible to predict whether a situation might arise whereby convergence does not occur, which would lead to significant problems in the communication system through the delay that would be incurred.

Here, we will study the convergence of the Newton-barrier algorithm heuristically. Indeed, the paper states that the self-concordance analysis provides a very loose bound on the number of iterations required to attain a specified error ϵ_l. Thus, it makes sense to test convergence, in terms of the number of iterations that are needed to obtain a sufficiently deep notch in the windowed signal, by running a few numerical experiments.

It is also of interest to analyse the 'run-time' of the algorithm. Indeed, it is not much use to design an algorithm that converges in only a few iterations if each iteration requires a large amount of time to execute. Matlab conveniently

supports such an analysis through several built-in functions. If you are interested in CPU time that has elapsed since you started an operation, you can use the `cputime` function. If you want to record *real* time, you can try the `tic` and `toc` functions. These are command line functions, but a more powerful graphical interface is included with the `profile` command. You can use the online help documentation to learn about these functions.

Task 11:
Construct a set of numerical experiments that explore the relationship between the convergence of the Newton-barrier algorithm and the following parameters: μ, ϵ_o, ϵ_i and the initial value of t. Both the number of iterations and the run-time of the algorithm should be studied.

For your report: A large amount of data can be generated in this task; do not attempt to show all of it. Instead, present the most interesting results in a logical and well-organised manner. The exact approach you take is left up to you, but you may wish to show one or two graphs or tables giving details of your findings, and use these to support any conclusions you might glean from the study.

4.2 Refinements

We have rigorously designed and analysed an optimisation procedure for obtaining a window for a given data vector **d** based on the problem formulation given in eq. (16). We have tested this procedure and (hopefully) found it to provide a good depth of notch with reasonable convergence and run-time properties. What is left to do? The paper provides some thoughts.

Task 12:
Read the rest of the paper.

From the paper, we see that there is a lot more that can be done to the Newton-barrier algorithm that we have implemented in order to obtain more favourable properties with regard to real-time implementation. The paper points

out that the direct use of the Newton-barrier method has $O(N^3)$ complexity, and several approximations and simplifications are proposed, which are shown to reduce this complexity to $O(N^2)$. Various parameters are also studied in more depth in order to provide some usage guidelines. Finally, a few communication-related performance metrics are investigated (e.g., dynamic range of the signal, packet-error rate). Although not relevant to your Newton-barrier implementation, you may find these to be of interest as they place the current work back into the context of the overarching communication system.

Based on your heuristic convergence and run-time analysis, you should think about alternative or additional refinements that you might implement in order to improve the computational properties of the numerical optimisation algorithm. It is not necessary to include this information in your report, but it is instructive to reflect upon the possibilities nonetheless.

4.3 Summary

In this penultimate part of the project, we

- studied the convergence and the run-time of the Newton-barrier algorithm through heuristic, numerical experiments;
- explored possible refinements to the algorithm that can be made to, for example, reduce the computational complexity of each Newton iteration;
- thought about alternative/additional refinements as well as numerical tests that could be carried out to quantify their benefits and drawbacks.

The final, short section of the project will focus on extending what you have learned and implemented to a slightly different scenario.

5 Extensions

So far, we have focused on a communication system that encodes information using a QPSK symbol constellation and constructs a window that, when applied to the time-domain vector of QPSK symbols $\mathbf{d}$, yields a notch at a desired spectral location. The window is designed such that it does not distort the phase of the QPSK symbols, nor does it *significantly* distort the amplitude. Thus, the information content of the QPSK message vector largely remains intact.

But what if the data is encoded using a multilevel constellation like 16-QAM? How can we ensure the information encoded in the amplitudes and phases of the 16-QAM symbols is not lost? What if we wish to create multiple notches in the spectrum of the transmitted message vector, as depicted in Fig. 14? Your final task will revolve around devising a method to overcome the issues raised by these questions.

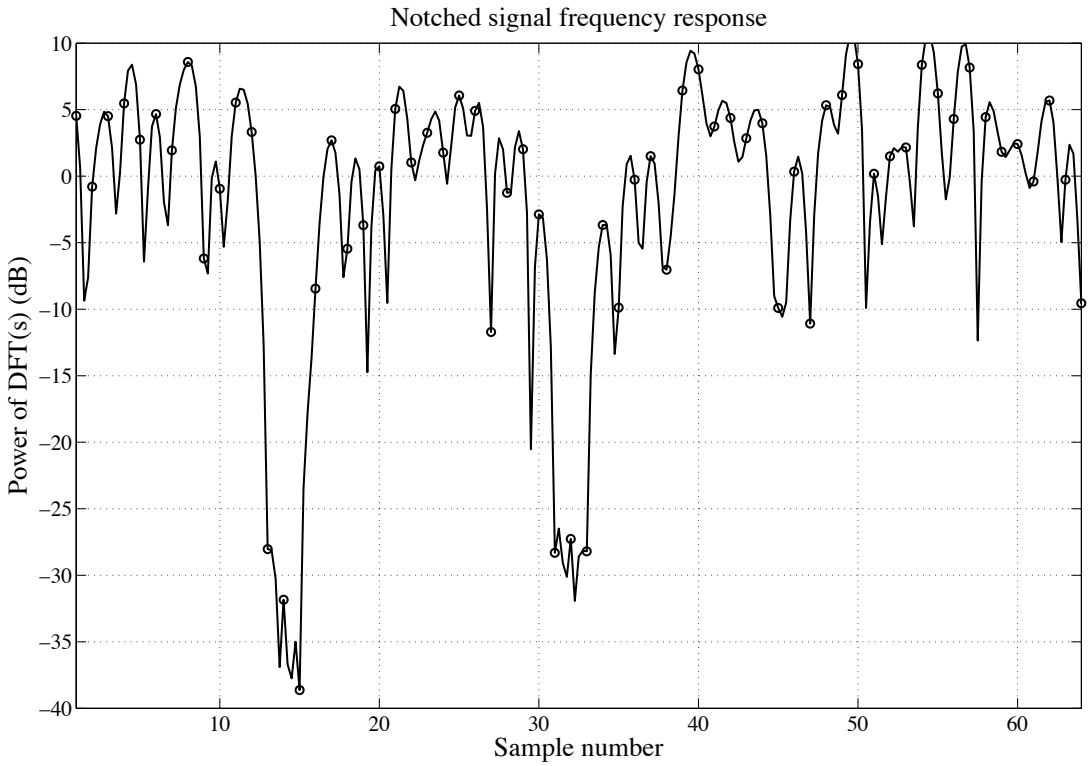


Figure 14: Illustration of data spectrum with two notches.

Task 13:
Reformulate the problem stated in eq. (16) to support window vector optimisation for possibly multiple spectral notches where the elements of the input data vector **d** are taken from a 16-QAM constellation. Write a programme called `barrier_window_16qam.m` that uses the barrier method to numerically calculate the optimal vector **x** according to this new problem formulation. You may use Newton's method or another suitable numerical optimisation algorithm in each iteration of the barrier method. The function should take the following as inputs:

1. a data vector **d**;
2. an upsampling factor ρ;
3. a variable called `indices` that defines the location(s) of the notch(es).

The function should return as output two plots:

1. the constellation diagram of the windowed data points corresponding to the equation $\mathbf{s} = \mathbf{D}\mathbf{x}$ in the paper;
2. a plot depicting the upsampled spectrum of **s** in decibels.

For your report: Describe the new optimisation problem formulation, both mathematically and qualitatively. Include key sections of your code and two example plots obtained from your function (clearly state the chosen inputs); comment on your methodology and observations.

6 Conclusion

You have come to the end of the project. As you write your report, pay close attention to its structure. You should provide an adequate introduction and conclusions at the beginning and end of the report, respectively. You must carefully balance the amount of detail that you give such that the report flows in a logical and easy-to-read fashion. Do not be too concise, but equally, you must refrain from being overly verbose. Be sure to include what was asked of you in the tasks. And, finally, remember to proofread your work before you submit!