DFT & Digital Signal Processing Experiment

2nd Year Electronics Lab

Department of Electrical and Electronic Engineering, Imperial College London, UK

Introduction

This lab will teach you some fundamentals of Digital Signal Processing (DSP), and introduce you to MATLAB, a mathematical tool that integrates numerical analysis, matrix computation and graphics in an easy-to-use environment. MATLAB is highly interactive; its interpretative nature allows you to explore mathematical concepts in signal processing without tedious programming. Once grasped, the same tool can be used for other subjects such as circuit analysis, communications, and control engineering.

Learning Outcomes

This experiment guides you through the theory of the Discrete Fourier Transform (DFT) and other discrete time signal processing concepts. There are many similarities to the continuous time domain and the Continuous Time Fourier Transform (CTFT), covered in Year 1, but there are also many differences.

At the end of this lab the student will be able to:

- Generate and analyse discrete time signals using MATLAB
- Analyse signals by applying Fourier Transforms and window functions
- Analyse digital filters and their responses
- Demonstrate conceptual understanding of discrete signal processing
- Evaluate of their work and their results

Recommended Textbooks

This experiment is designed to support the second year course Signals and Linear Systems (EE2-5). Since the laboratory and the lectures may not be synchronised, you may need to learn certain aspects of signal processing ahead of the lectures. While the notes provided in this experiment attempt to explain certain basic concepts, it is far from complete.

You are recommended to refer to the excellent textbook **An Introduction to the Analysis and Processing of Signals** by Paul A. Lynn. A more advanced reference is **'Discrete Time Signal Processing'**, by A.V. Oppenheim & R.W. Schafer, as well as the 'Signals and Linear Systems' course notes. Another good recommendation is **'Signals and Systems (2nd Edition)'** by A.V. Oppenheim, Alan S. Willsky with S. Hamid Nawab.

Timetable

There are 9 exercises in this handout; you should aim to complete all of them in the 4 timetabled lab sessions.

Exercise 1 - Learning MATLAB

Objectives

- To become familiar with MATLAB
- To be able to generate discrete signals in MATLAB
- To be able to identify when periodicity is lost due to sampling

MATLAB Tips

- The online help is available within MATLAB by simply typing help at the cursor input `'>'`
- You do not need to declare variables
- Anything between `'%'` and newline is a comment
- The result of a statement is echoed unless the statement terminates with a semicolon `';'`
- The colon `':'` has special significance, for example, `x=1:5` creates a row vector containing five elements, vector $x = [1\ 2\ 3\ 4\ 5]$.

So let's start by launching MATLAB on your computer.

Now consider a sampled sine wave. In general, it takes three parameters to describe a real sinusoidal signal completely: amplitude (A), normalised angular frequency (ω) and phase (ϕ).

$$x[n] = A\sin(\omega n + \phi)$$

Note: It is common to use F for 'real' frequencies and Ω for 'real' angular frequencies. The scaled versions being f for normalised frequency and ω for normalised angular frequency where the units of ω are 'radians per sample' and f are 'cycles per sample'. Therefore:

$$\omega = \frac{\Omega}{F_s} = 2\pi\frac{F}{F_s} = 2\pi f, \quad \text{where } f = \frac{F}{F_s} \text{ and } \Omega = 2\pi F$$

Now create a plot of a sampled sine wave with a frequency, F, of 1KHz and a sampling frequency, F_s, of 25.6kHz where $N = 128$ meaning that 128 data points should be generated.

Question 1: Before plotting this sine wave in MATLAB try to calculate how many cycles of the sine wave will be displayed in the plot.

To check your answer lets now plot the sine wave in MATLAB. In MATLAB you can create a vector, `n`, with 128 elements by entering the following:

```
% Define constants
fsamp= 25600;
fsig = 1000;
nsamp = 128;
n = 0 : nsamp-1;
```

Now try the command `'whos'` to check which variables are defined at this stage and also how large they are (matrices are reported as row by column). You can also check this using the 'Workspace' sidebar which is on the right-hand side of the MATLAB window by default.

To create the sine wave $x[n] = \sin(2\pi\frac{F}{F_s}n)$ simply enter:

```
x = sin(2*pi*(fsig/fsamp)*n);
figure;
plot (x)
```

This plots x against the sample index (from 0 to 127).

Now we can create sine waves lets generate some more plots of $x[n] = \sin(\omega n)$ where:

(i) $\omega = \frac{\pi}{4}$ and $N = 20$

(ii) $\omega = 4$ and $N = 20$

Question 2: What does (i) & (ii) tell you about the periodicity of $x[n] = \sin(\omega n))$? Is it always periodic? (Hint: a periodic sequence is one where $x[n] = x[n + N_p]$, therefore, when does $\sin(\omega n) = \sin(\omega(n + N_p))$?)

Now let's go back to our original sine wave where $F = 1$KHz, $F_s = 25.6$kHz and $N = 128$. Try to see if you can plot x against time by creating a vector `t` with the first element being 0 and the last element being $127 \times$ `tsamp`, where elements in between are at increments of `tsamp`. Hint '`tsamp = 1/ fsamp;`'.

Now you can add titles and labels to your plot:

```
grid on
title ('Simple Sine Wave')
xlabel ('Time')
ylabel ('Amplitude')
```

You can also plot the sine wave using asterisk markers by entering the command `plot(t, x, '*')` and change the colour of the curve to red with `plot(t, x, 'r')`. The output can be given as a discrete signal using the command `stem(x)`. The plot window can also be divided with the `subplot` command and the `zoom` command allows examination of a particular region of the plot (go to 'https://uk.mathworks.com' to discover more).

Sine waves are such useful waveforms that it is worthwhile writing a function to generate them for any signal frequency, F, sampling frequency, F_s, and number of samples, N. In MATLAB, functions are individually stored as a file with extension **'.m'**. To define a function `sinegen`, create a file **sinegen.m** using the MATLAB editor by selecting **New Function** from the **File** menu.

```
function y = sinegen(fsamp, fsig, nsamp)
    % body of the function
end
```

Save the file in your home directory with the **Save As** option from the **File** menu. To test `sinegen`, enter:

```
y = sinegen(fsamp, fsig, nsamp);
```

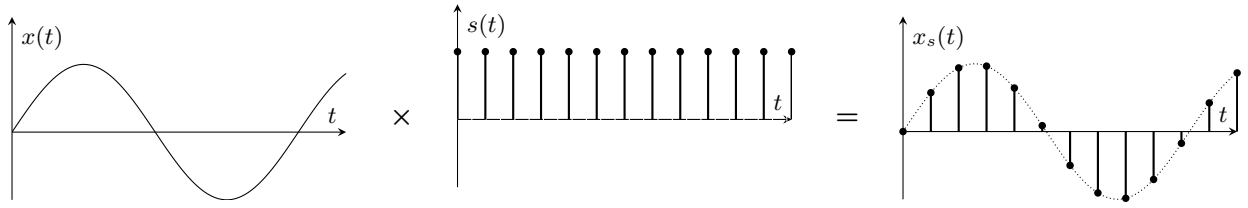
If MATLAB gives the error message **undefined function**, use the 'Current Folder' browser *(on the left-hand side of the MATLAB window by default)* to add the folder, containing the function file you have just created, to the search path. To do this, from the 'Current Folder' browser right-click the folder you want to add. Then from the context menu, select **Add to Path**, and then select **Selected Folders and Subfolders**.

Exercise 2 - Effects of sampling

Objectives

- To understand the effects of sampling especially in the frequency domain
- To understand why sampling makes aliasing possible

Let's start by thinking about sampling in the time domain. Which can be thought of as multiplying a continuous time domain signal by a train of impulses. To get a better conceptual understanding of this sampling process it has been visualised in Fig. 1.

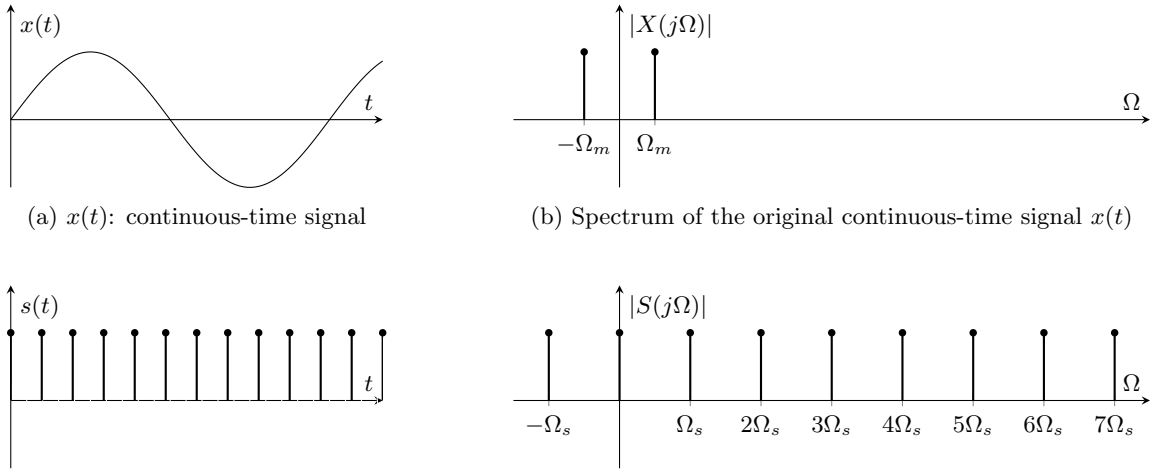


(a) $x(t)$: continuous-time signal (b) $s(t)$: impulse train (c) $x_s(t)$: sampled signal

Figure 1: The sampling process of an continuous-time signal

Note that here $x_s(t) = x(t) \times s(t) = \sum_n x(nT)\delta(t - nT)$, where $-\infty < n < \infty$ and $T = 1/F_s$ and is, therefore, still a continuous time signal which is equal to 0 for $t \neq nT$. $x_s(t)$. This should not be confused with $x[n]$ which is a discrete time signal and only exists for $n = 0, 1, 2, \cdots$.

Now think about $x(t)$ and $s(t)$ and their frequency domain representations which are shown in Fig. 2.



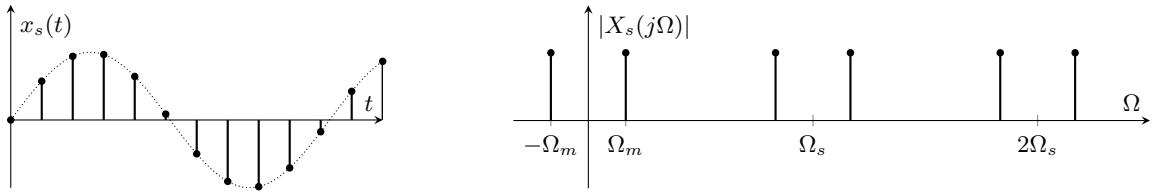
(a) $x(t)$: continuous-time signal (b) Spectrum of the original continuous-time signal $x(t)$

(c) $s(t)$: impulse train (d) Spectrum of the original continuous-time signal $s(t)$

Figure 2: Frequency domain representation of $x(t)$ and $s(t)$

You should be able to recall from Year 1 that multiplication in the time domain is equivalent to convolution in the frequency domain. Therefore $x_s(t) = x(t) \times s(t)$ is the same as $X_s(j\Omega) = X(j\Omega) * S(j\Omega)$ where '$*$' denotes the linear convolution.

Thus, $X_s(j\Omega)$ will be a periodic function of frequency Ω, consisting of the sum of shifted and scaled replicas of $X(j\Omega)$, shifted by integer multiples of Ω_s and scaled by $\frac{1}{T}$.



(a) $x_s(t)$: sampled signal (b) Spectrum of the sampled signal $x_s(t)$

Figure 3: A sample sine wave and its spectrum

Sampling Theorem

Sampling has the effect, therefore, of creating spectral images every Ω_s and thus to avoid information loss due to overlapping spectral images (aliasing) the following condition must be met:

Definition:

$$|\Omega_m| \leq \frac{\Omega_s}{2}, \quad \text{where } \Omega_s = 2\pi F_s = 2\pi/T \quad \Longrightarrow \quad |\omega| \leq \pi$$

The frequency $2\Omega_m$ is called the Nyquist rate. Sampling above this frequency is called *oversampling*, conversely, sampling below this frequency is called *undersampling*. Lastly sampling at a frequency exactly equal to the Nyquist rate is called *critical sampling*.

To see aliasing in action lets generate some more plots using `sinegen` function setting $N = 100$, $F_s = 8000$Hz and varying:

(i) $F = 150$Hz, (ii) $F = 300$Hz and (iii) $F = 600$Hz

(iv) $F = 7400$Hz, (v) $F = 7700$Hz and (vi) $F = 7850$Hz

Question 3: Discuss briefly the results of (i)-(iii) and (iv)-(vi) stating what you have observed.

Question 4: Predict the result for frequencies $F = 24150$Hz, $F = 24300$Hz and $F = 24600$Hz and confirm your prediction with MATLAB.

Question 5: Consider $x(t) = \cos(100\pi t)$

(a) Determine the minimum sampling rate to avoid aliasing

(b) Write down an expression for $x[n]$ if a sampling frequency of 200 Hz is used.

(c) Write down an expression for $x[n]$ if a sampling frequency of 75 Hz is used.

Question 6: If the sampling frequency is 48 kHz, what is the normalised angular frequency of the discrete time signal corresponding to a sinusoid at 1.2 kHz?

Question 7: Given a signal whose normalised angular frequency is $\frac{\pi}{4}$ what is its frequency in terms of the sampling frequency, F_s?

Exercise 3 - Discrete Fourier Transform

Objectives

- To see that any periodic signal can be created from a linear combination of sinusoidal waves
- To introduce the Discrete Fourier Transform (DFT)
- To understand the difference between the amplitude and phase in the frequency domain

We have seen how MATLAB can manipulate data as a 1×128 vector. Now let us create a matrix `S` containing 4 rows of sine waves such that the first row is frequency F the second row is the second harmonic $2 \times F$ and so on. In this case `nsamp` = 128, `fsig` = 1000 and `fsamp` = 25600.

```
S(1,:) = sinegen(fsamp, fsig, nsamp);
S(2,:) = sinegen(fsamp, 2*fsig, nsamp);
S(3,:) = sinegen(fsamp, 3*fsig, nsamp);
S(4,:) = sinegen(fsamp, 4*fsig, nsamp);
```

Note the use of `':'` as the second index of `S` to represent all columns. Examine `S` and then try plotting it by entering `plot(S)`.

Question 8: Is this plot correct? Why or why not?

Now plot the transpose of `S` by using `plot(S')`.

Note that in MATLAB, `''` denotes a complex conjugate transpose (Hermitian), which does not effect us here since `S` is real.

We can do the same thing to the above 4 statements by using a for-loop. The following statements create `S` with 10 rows of sine waves:

```
for i = 1:10
    S(i,:) = sinegen(fsamp, i*fsig, nsamp);
end
```

Next let us explore what happens when we add all the harmonics together. This can be done by first creating a row vector `p` containing 'ones', and then multiplying this with the matrix `S`:

```
p = ones(1,10);
f = p*S;
plot(f)
```

Note: you can also use the following command `sum(S,1)` to achieve the same outcome.

This is equivalent to calculating:

$$x_1[n] = \sum_{k=1}^{10} \sin(k\omega n)$$

Where $\omega = 2\pi \frac{F}{F_s}$ is the normalised angular fundamental frequency and $n = \{0,\ 1,\ 2, \cdots,\ 127\}$.

Question 9: Explain the result $x_1[n]$.

Instead of using a unity row vector, we could choose a different weighting $b[k]$ for each harmonic component in the summation:

$$x_2[n] = \sum_{k=1}^{10} b[k] \sin(k\omega n)$$

Try using $b[k] = \{1,\ 0,\ 1/3,\ 0,\ 1/5,\ 0,\ 1/7,\ 0,\ 1/9,\ 0\}$.

```
bk = [1,0,1/3,0,1/5,0,1/7,0,1/9,0];
f = bk*S;
plot(f)
```

Question 10: What do you get for $x_2[n]$ using $b[k] = \{1,\ -1/2,\ 1/3,\ -1/4,\ 1/5,\ -1/6,\ 1/7,\ -1/8,\ 1/9,\ 0\}$? *(You may want to try to derive these results from first principles.)*

So far we have been using sine waves as our basis functions. Let us now try using cosine signals.

First, create a `cosgen` function and use it to generate a 10x128 matrix `C` that contain 10 harmonics of cosine waveforms. Now, use the weighting vector $a[k] = \{1,\ 0,\ -1/3,\ 0,\ 1/5,\ 0,\ -1/7,\ 0,\ 1/9,\ 0\}$, to compute `g=ak*C`. Plot the result.

Question 11: How does `g` differ from `f` obtained earlier from using sine waves? What general conclusions on the even and odd symmetry of the signal can you draw?

We are now starting to see a really important property of periodic signals using Fourier series analysis. Fourier series analysis states that any periodic function can be constructed from a weighted sum of harmonically related sinusoids. This leads us to the definition of the Discrete Fourier Series (DFS):

DFS (Discrete Fourier Series):

$$\text{Forward Transform: } X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}, \quad k = 0, \pm 1, \pm 2, \cdots$$

$$\text{Inverse Transform: } x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j\frac{2\pi}{N}kn}, \quad n = 0, \pm 1, \pm 2, \cdots$$

Question 12: Prove that $x[n] = a[0] + \sum_{k=1}^{\infty} a[k] \cos(\frac{2\pi}{N}kn) + \sum_{k=1}^{\infty} b[k] \sin(\frac{2\pi}{N}kn)$ is equivalent to $x[n] = \sum_{k=-\infty}^{\infty} D[k] e^{j\frac{2\pi}{N}kn}$. How can the coefficients $D[k]$ be determined?

Hint: Remember that $\sin(\theta) = -\frac{1}{2}je^{j\theta} + \frac{1}{2}je^{-j\theta}$ and $\cos(\theta) = \frac{1}{2}e^{j\theta} + \frac{1}{2}e^{-j\theta}$.

You should now be able to see that it is possible to construct $x[n]$ from a finite number of harmonics; the frequencies being $\omega = \frac{2\pi}{N}k$, where $k = 0, 1, \cdots, N-1$.

Question 13: Derive the DFS of the periodic sequence $x[n] = \{\cdots, \underset{\uparrow}{0}, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, \cdots\}$, where $N = 4$ and the arrow denotes the 0^{th} sample. What do you notice about the periodicity of $X[k]$?

It should be evident that $X[k]$ is itself a (complex-valued) periodic sequence with fundamental period equal to N, that is,

$$X[k + N] = X[k]$$

Where k is the harmonic or frequency bin number, n is the sample number.

To recap, if the signal is periodic we consider one period and perform the DFS for frequency analysis. So what about non-periodic signals? If an infinitely long continuous time domain signal is sampled, the frequency

domain representation is given by what is called the Discrete-Time Fourier Transform (DTFT). (This is defined as $X(e^{j\omega}) = \sum_{-\infty}^{\infty} x[n]e^{-j\omega n}$ and gives a continuous frequency domain representation). It is not possible in practice, however, to get an infinite number of samples. To overcome this problem we, therefore, only take a finite number of samples N and look at the frequency domain representation of these N samples. This leads to what is called the Discrete Fourier Transform (DFT) and is just the sampled frequency domain of the DTFT spectrum $X(e^{j\omega})$.

What is surprising is the equation for the Discrete Fourier Series (DFS) and the Discrete Fourier Transform (DFT) are actually the same but the reasoning behind the periodicity is not. The periodicity of the DFS is the natural extension due to the time sequence being periodic. Whereas the periodicity of the DFT is a forced by product of sampling the frequency spectrum $X(e^{j\omega})$ *(recall exercise 2 and the effects of sampling).*

The full mathematical definition of the DFT is as follows:

DFT (Discrete Fourier Transform): $x[n] \rightarrow X[k]$

$$\text{Forward Transform: } X[k] = \sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi}{N}kn}, \quad k = 0, \pm 1, \pm 2, \cdots$$

$$\text{Inverse Transform: } x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{j\frac{2\pi}{N}kn}, \quad n = 0, \pm 1, \pm 2, \cdots$$

Question 14: Can the Fourier transform modify the energy of a signal?

Question 15: When applying Fourier analysis to a signal, under which circumstances should a DFS be employed and under which circumstances should a DFT be employed?

Important: In MATLAB we use the `fft` function to calculate the DFT. The Fast Fourier Transform (FFT) is just a fast algorithm that is used to calculate the DFT and **not a separate transform**.

Question 16: For an N-point signal, the DFT requires N^2 multiplications while the FFT only requires $\frac{1}{2}N\log_2 N$ multiplications. What is the computational complexity reduction for $N = 16$, 1024, 4096?

Now let us find the FFT spectrum of the sine wave produced with `sinegen`:

```
x = sinegen(8000,1000,8);
A = fft(x)
```

Question 17: Explain the numerical result in `A`. Make sure that you know the significant of each number you get back. If in doubt, ask a demonstrator. You might also like to evaluate the FFT of a cosine signal, is it what you would expect?

Question 18: What is the frequency resolution of a 256-point DFT when the sampling frequency is 1000 Hz?

Instead of working with real and imaginary numbers, it is often more convenient to work with the magnitude and phase of the components, as defined by:

```
mag=sqrt(A.*conj(A));
phase=atan2(imag(A), real(A));
```

If you precede the `'*'` or `'/'` operators with a period `'.'`, MATLAB will perform an element-by-element multiplication or division of the two vectors instead of the normal matrix operation. `conj(A)` returns the conjugate of each elements in `A`.

Write a function `plotspec(A)` which plots the magnitude and phase spectra of `A` (the magnitude spectrum should be plotted as a bar graph). This function will be useful later.

Now, let's create a pulse signal containing 8 samples of ones and 8 samples of zeros. Obtain its magnitude spectrum and check that it is what you expect.

```
x = [ones(1,8), zeros(1,8)];
A = fft(x)
plotspec(A)
```

Gradually reduce the width of the pulse until it becomes an impulse, i.e. contains only a single one and 15 zeros. Note how the spectrum changes.

Question 19: What does this result (plot) tell us about the frequency components that make up an impulse signal?

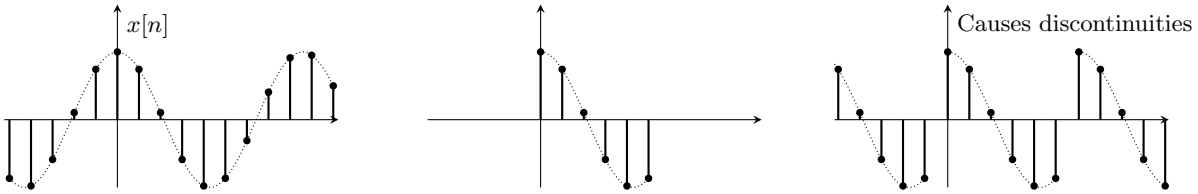
Next delay this impulse signal (a single one and 15 zeros) by 1 sample and find its spectrum again. Examine the real and imaginary spectra.

Question 20: What happens to the spectrum when the impulse signal is delayed? What do you expect to get if you delay the impulse by 2 samples instead of 1? Investigate the phase of the delay (you might find the `unwrap` function useful).

Exercise 4 - The effects of using windows

Objectives

- To understand why spectral leakage occurs due to discontinuities in the time domain
- To see why windows are helpful in reducing spectral leakage



(a) $x[n]$: sampled signal (b) $x[n] \times w[n]$: Windowed Signal (c) Repeated to calculate DFT

Figure 4: Rectangular Window

So far we have only dealt with sine waves with an integer number of cycles. Generate a sine wave with an incomplete last cycle using:

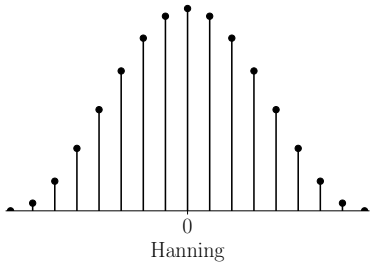
```
x = sinegen(20000,1000,128);
```

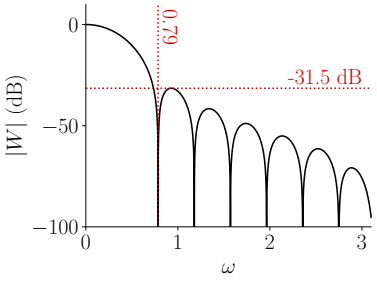
Obtain its magnitude spectrum. Instead of obtaining just two spikes for the positive and negative frequencies, you will see many more components around the main signal component. Since we are not using an integer number of cycles, we introduce a discontinuity after the last sample as shown in Fig. 4. This is equivalent to multiplying a continuous sine wave with a rectangular window as shown in Fig. 4. It is this discontinuity that generates the extra frequency components. To reduce this effect, we need to reduce or remove the discontinuities. Choosing a window function that gradually goes to zero at both ends can do this. Two common window functions are defined mathematically as:

Hanning:

$w[n] = 0.5 + 0.5\cos\frac{2\pi n}{M+1}$

Main lobe width: $8\pi/(M+1)$
Relative sidelobe level: 31.5dB
Rapid sidelobe decay

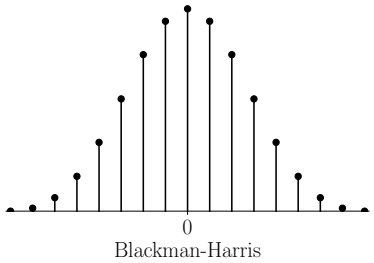


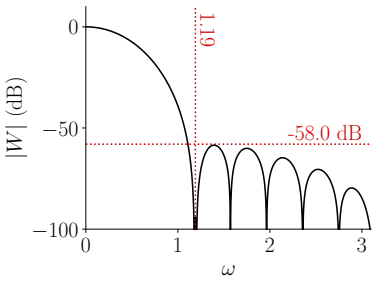


Blackman-Harris (3-term):

$w[n] = 0.42 + 0.5\cos\frac{2\pi n}{M+1} + 0.08\cos\frac{4\pi n}{M+1}$

Main lobe width: $12\pi/(M+1)$
Relative sidelobe level: 58.0dB
Best peak sidelobe





Let us now see how you would plot one of these windows in MATLAB.

```
y = hanning(128);
plot(y);
plotspec(fft(y));
```

Now try plotting the result of multiplying your sine wave with this window function and find it's magnitude spectrum of the sine wave without windowing to those weighted by the Hanning window function.

Now try to repeat this for the Blackman-Harris window using the `blackman` function in MATLAB.

Question 21: We have now seen the effect of windowing by two different windows. Compare the signals with and without windowing discuss the differences. So why might we pick a specific window?

Question 22: Recall that multiplication in the time domain is the same as convolution in the frequency domain. Try to use this fact to explain the effects of windowing.

Question 23: Is it ever a bad idea to window a signal before taking the DFT? *(Hint: Try windowing the first sine wave we generated in exercise 1)*

Exercise 5 - The analysis of an unknown signal

Objectives

- To be able to analyse the frequency domain of a real world signal

To test how much you have understood so far, you are given an unknown signal stored in a disk file. The file contains over 1000 data samples. It is also known that the signal was sampled at 1 kHz and contains one or more significant sine waves.

You can load the data into a vector using the MATLAB command:

```
load unknown;
```

Question 24: Take at least two 256-sample segments from this data and compare their spectra. Are they essentially the same? Make sure that you can interpret the frequency axis (ask a demonstrator if you are not sure). Deduce the frequency and magnitude of its constituent sine waves.

Exercise 6 - Filtering in the Frequency Domain

Objectives

- To be able to filter a signal in the frequency domain

What is a Digital Filter?

The purpose of a digital filter, as for any other filter, is to enhance the wanted aspects of a signal, while suppressing the unwanted aspects. In this experiment, we will restrict ourselves to filters that are linear and time-invariant. A filter is linear if its response to the sum of the two inputs is merely the sum of its responses of the two signals taken separately, while a time-invariant filter is one whose characteristics do not alter with time. We make these two restrictions because they enormously simplify the mathematics involved. Linear time-invariant filters may also be made from analogue components and it is often cheaper to do so.

An obvious way to filter a signal is to remove the unwanted spectral components in the frequency domain. The steps are:

1. Take the FFT of the signal to convert it into its frequency domain equivalent.
2. Set unwanted components in the spectrum to zero.
3. Take the inverse FFT of the resulting spectrum.

Filter the noise-corrupted signal by removing all components above 300 Hz. Compare the filtered waveform with the original signal.

Question 25: What are the disadvantages of filtering in the frequency domain?

Important: Please note that the filtered waveform should still be real. If your result is not real it is because you have destroyed a property in the frequency domain that makes the signal real in the time domain.

It is useful to know that if a signal $x[n]$ has a special property in the time domain then there will be a corresponding property in the frequency domain, $X(e^{j\omega})$ and $X[k]$.

One Domain	**Other Domain**
Discrete	Periodic
Symmetric	Symmetric
Antisymmetric	Antisymmetric
Real	Conjugate Symmetric
Imaginary	Conjugate Antisymmetric
Real & Symmetric	Real & Symmetric
Real & Antisymmetric	Imaginary & Antisymmetric

Symmetric: $x[n] = x[-n]$

$X(e^{j\omega}) = X(e^{-j\omega})$

$X[k] = X[(-k)_{\mathrm{mod}N}] = X[N-k]$ for $k > 0$

Conjugate Symmetric: $x[n] = x^*[-n]$

Conjugate Antisymmetric: $x[n] = -x^*[-n]$

Exercise 7 - Impulse response and convolution

Objectives

- To introduce linear convolution for discrete time signals
- To show that circular convolution in the time domain is the same as multiplication in the frequency domain for discrete time signals

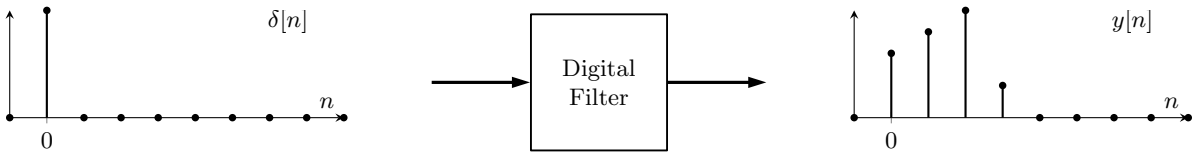


Figure 5: Impulse response of a filter

Suppose we apply an impulse (Kronecker delta), $\delta[n]$, as an input into a filter. The output, $y[n]$, denotes the impulse response of the digital filter, $h[n]$, shown in Figure 5. Since the discrete-time system is time-invariant, the filter response to $\delta[n-k]$ is $h[n-k]$.

In general, any input signal, $x[n]$, can be decomposed into a sum of impulses with different delays $\{\delta[n], \delta[n-1], \cdots, \delta[n-N]\}$. Due to linearity, the response of the digital filter will be the sum of the outputs i.e. $y[n] = h[n] + h[n-1] + \cdots + h[n-N]$. Note that the impulses should be scaled by the signal value at the respective time samples i.e. $\{x[0]\delta[n], x[1]\delta[n-1], \cdots, x[N]\delta[n-N]\}$. Consequently, the filter output is, $y[n] = x[0]h[n] + x[1]h[n-1] + \cdots + x[N]h[n-N]$ which is defined as linear convolution.

Decompose $x[n]$ into $\delta[n-k] + \delta[n-(k+1)] + \delta[n-(k+2)]$.

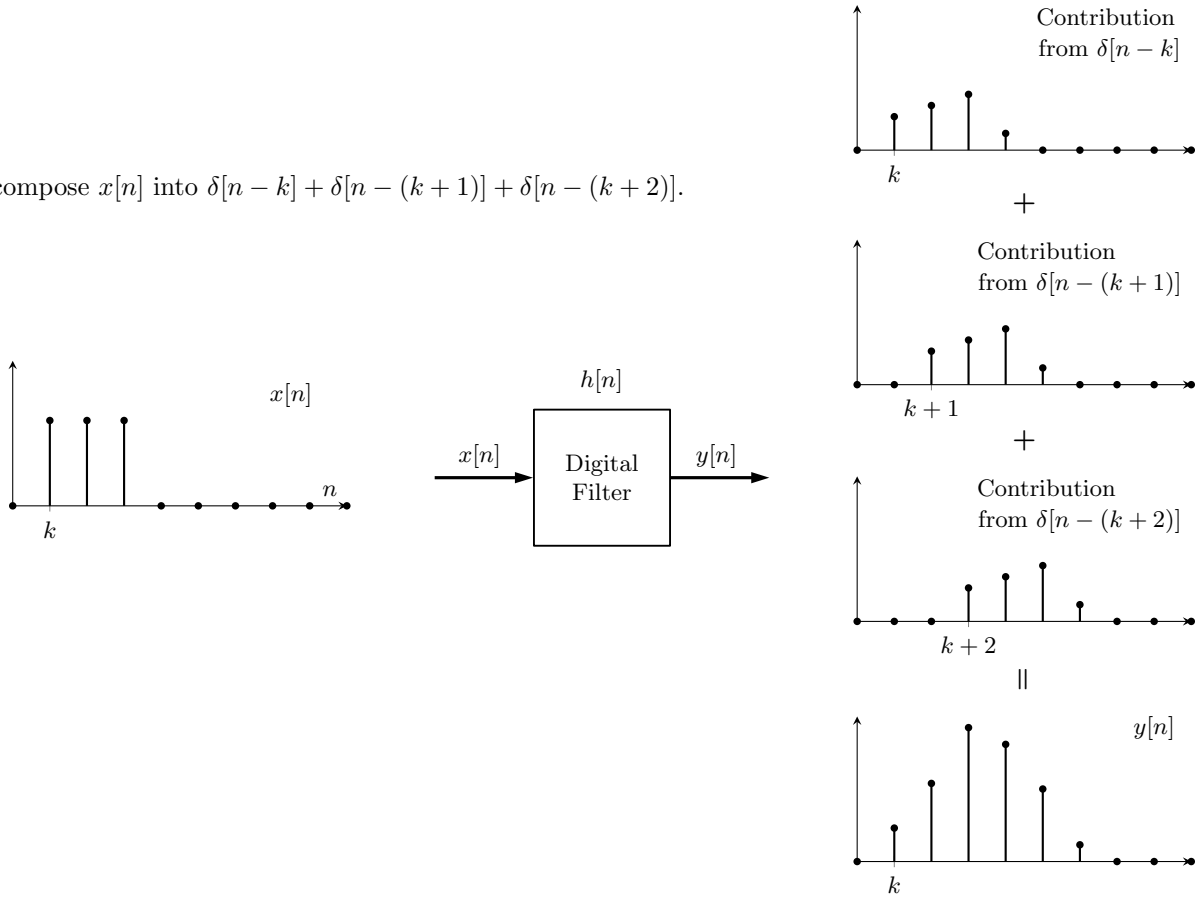


Figure 6: Linear Convolution

Definition:

$$\text{Linear Convolution: } y[n] = x[n] * h[n] \triangleq \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

If we substitute r for $n-k$ this becomes:

$$y[n] = \sum_{r=-\infty}^{\infty} h[r]x[n-r]$$

Thus, for any linear time-invariant (LTI) filter, the output values consist of the sum of the past input values, weighted by the elements of the impulse response $h[n]$. This has been visualised in Fig. 6.

Question 26: What information is needed in order to compute the output of a discrete-time LTI system?

Question 27: We now know that $h[n]$ is the impulse response of a filter. How can you obtain its frequency response?

You can perform linear convolution in MATLAB using the `conv` function. So you can replicated the result of Fig. 6 using:

```
u = [0 0 1 1 ];
v = [0.6 0.8 1 0.3];
w = conv(u,v);
stem(w);
```

We have already said that convolution in the continuous time domain is equivalent to multiplication in the frequency domain. Therefore, calculate the result of 'w' using only the `fft` and `ifft` functions.

Question 28: Explain why 'ω' computed using the two methods, linear convolution and the FFT, are different.

As explained earlier sampling the spectrum makes the time-domain periodic. This means that multiplication in the frequency domain is no longer linear convolution in the time domain. In fact multiplication in the frequency domain for discrete signals is equivalent to a different type of convolution, called **Circular Convolution** and is defined as

Definition:

$$\text{Circular Convolution: } y[n] = x[n] \circledast_N h[n] \triangleq \sum_{k=0}^{N-1} x[k]h[(n-k)_{\text{mod } N}].$$

Question 29: Knowing this fact can you think of anyway to make circular convolution the same as the linear convolution result? If so can you now calculate the values of 'w' using only the `fft` and `ifft` functions correctly?

Question 30: What differences are there, if any, between linear convolution of a periodic signal and circular convolution?

Exercise 8 - A simple Finite Impulse Response (FIR) filter

Objectives

- To introduce Finite Impulse Response (FIR) filters

The filter impulse response is the output when the input is $\delta[n]$. Recall that an impulse signal contains equal power in all frequency components, i.e. its magnitude spectrum is flat. Therefore, the frequency response of a filter is simply the discrete Fourier transform of the filter impulse response.

We are now ready to try out a simple digital filter. Let us assume that the impulse response of the filter $h[n]$ is a sequence of 4 ones.

Question 31: Describe the frequency response of this filter using `plotspec`.

This filter keeps the low frequency components intact and reduces the high frequency components. It is therefore known as a low-pass filter.

Question 32: Suppose the input signal to the filter is $x[n]$, show that this filter produces the output $y[n] = x[n] + x[n-1] + x[n-2] + x[n-3]$

Let us apply this filter to the noise corrupted signal used in exercise 5. This can be done using the `conv(x, h)` function in MATLAB to perform a convolution between the signal and $h[n]$.

Question 33: Compare the waveform before and after filtering. Do you see what you would have expected?

Now try to implement a moving average filter. (Hint: $h[n] = \frac{1}{4}\{1,\ 1,\ 1,\ 1\}$)

This averaging action tends to smooth out fast varying frequency components, but has little effects on the low frequency components.

Exercise 9 - Infinite Impulse Response (IIR) filter

Objectives

- To introduce Infinite Impulse Response (IIR) filters

The filter described in the last exercise belongs to a class known as **Finite Impulse Response (FIR)** filters. It has the property that all output samples are dependent on input samples alone. For an impulse response that contains many non-zero terms, the amount of computation required to implement the filter may become prohibitively large if the output sequence, $y[n]$, is computed directly as a weighted sum of the past input values. Instead, $y[n]$ is often computed as a sum of both past input values and past output values.

This type of filter has an impulse response of infinite length, therefore it is known as an **Infinite Impulse Response (IIR)** filter. It is also called a recursive filter because the outputs are being fed back to calculate the new output.

Derive the impulse response of the simple filter described by the following equation:

$$y[n] = \sum_{r=0}^{M} b[r]x[n-r] - \sum_{r=1}^{N} a[r]y[n-r]$$

Note that MATLAB provides a function called filter to perform general FIR and IIR filtering.

```
y = filter(b, a, x)
```

Where `b` $= \{b[0], b[1], \cdots, b[1-M]\}$ and `a` $= \{1, a[1], a[2], \cdots, a[N-1]\}$. Note that the FIR filter in exercise 8 can be implemented using `filter` where `a` $= \{1\}$.

Filter the noise corrupted signal, from exercise 5, with an IIR filter containing the following filter coefficients:

$$b = [0.0039,\ 0.0193,\ 0.0386,\ 0.0386,\ 0.0193,\ 0.0039]$$
$$a = [1.0000,\ -2.3745,\ 2.6360,\ -1.5664,\ 0.4929,\ -0.0646]$$

Find the impulse response of this filter and its frequency response. Hence explain the effect of filtering the noise corrupted signal with this filter.

This filter (known as a **Butterworth Filter**) was designed using the MATLAB functions `buttord` and `butter`. If you have time, you might explore and design your own filters.

Question 34: Derive the impulse response of: y[n]=0.25x[n]+y[n-1]?

Question 35: Why does an IIR filter have to be implemented recursively (i.e. it needs to use past outputs)?

Question 36: Compare the advantages and disadvantages of FIR and IIR filters.

If you have time why not see exercises 6, 7 & 8 in action by filtering a short bit of your own music. You can read and write '.wav' files in MATLAB in the following way:

```
[data,fs] = audioread('input.wav');
x=data(:,1); % make stereo recordings mono
X=fft(x);
% do processing
y =  ifft(X)
audiowrite('output.wav',y,fs)
```

In this case 'input.wav' can be a song of your choice *(why not use a online 'youtube to wav' converter to obtain a '.wav' file)*. Can you hear the difference that filtering makes?