Signals and Systems Laboratory Courswork (v. 2021)

2nd Year MTC

Department of Mechatronics Engineering, Chandigarh University, Mohali, Punjab, India

Introduction

This lab will teach you some fundamentals of Signals and systems (SAS) analysis, 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

CO1 Apply the basics of MATLAB programming to perform basic signal processing operations.
CO2 Make use of MATLAB to generate and characterize various continuous and discrete time signals.
CO3 Design and analyze discrete time systems using convolution and correlation in MATLAB.
CO4 Determine individual/ team work while performing experiment.
CO5 Adapt professionalism with ethics and communicate effectively.

Requirements

To complete this lab you will need to download the following 7 files (available on Blackboard): `ecg.mat`, `unknown.mat`, `piano.wav`, `track_1.wav`, `coronavirus.mat`, `aircrew.wav` and `aircrew_coef.mat`.

Recommended Textbooks

This experiment is designed to support the second year Signals and Systems course. 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, they are far from complete. You will likely find the recommended textbook helpful when studying this experiment as well as the lecture course: S. Haykin and B. Van Veen, "Signals and Systems," 2nd Ed., Wiley, 2003.

Timetable

There are 8 exercises in this coursework; you should aim to complete all of them in approximately 9 hours. Drop-in sessions for this module will also be provided through MS Teams.

Assessment

This laboratory coursework will be assessed through an online midterm test.

Contact

This lab was developed and designed by Anurag Sharma and Pulkit Jain. Please email us at anurag.ece@cumail.in if you have any suggestions or comments about this document (as this will help future year groups).

Exercise 1 - Learning MATLAB

Objectives

- To become familiar with MATLAB
- To be able to generate discrete signals in MATLAB

Launch MATLAB, click **New**, select **Live Script** (https://www.mathworks.com/help/matlab/matlab_prog/create-live-scripts.html). After you create a live script, you will be able to run the code you add to it.

GTA Tip

Don't use the terminal window because all of your code will be lost as soon as you close MATLAB.

So let's start by considering a sampled sine wave: $x[n] = A\sin(\Omega n + \phi)$. In general, it takes three parameters to describe a real sinusoidal signal completely: amplitude, A, normalised angular frequency, Ω, and phase, ϕ.

Definition:

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

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'.

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 by hand how many cycles of the sine wave will be displayed in the plot.

To check your answer let's 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;
```

GTA Tip

- 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 $x = [1\,2\,3\,4\,5]$.

Now check the 'Workspace' on the right-hand side of the MATLAB window (go to 'https://uk.mathworks.com/help/matlab/ref/workspace.html' to discover more). This tab allows you to check which variables are defined at this stage and also their dimension(s) where matrices are reported as row by column.

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). 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 × `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 [s]')
ylabel('Amplitude')
```

GTA Tip

You can also plot the sine wave using asterisk markers for each discrete value point 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 `xlim` 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 let's write 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);
```

GTA Tip

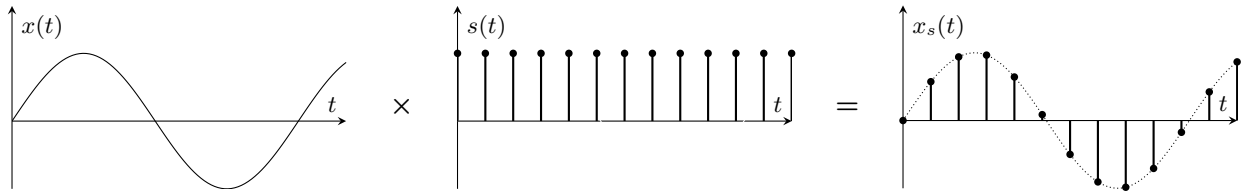
If MATLAB gives the error message **undefined function**, use the 'Current Folder' browser *(on the left-hand side of the MATLAB window by default)* to add the folder, containing the function file you have just created, to the search path. To do this, from the 'Current Folder' browser right-click the 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 might cause the problem of aliasing

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. This sampling process is shown in Fig. 1.

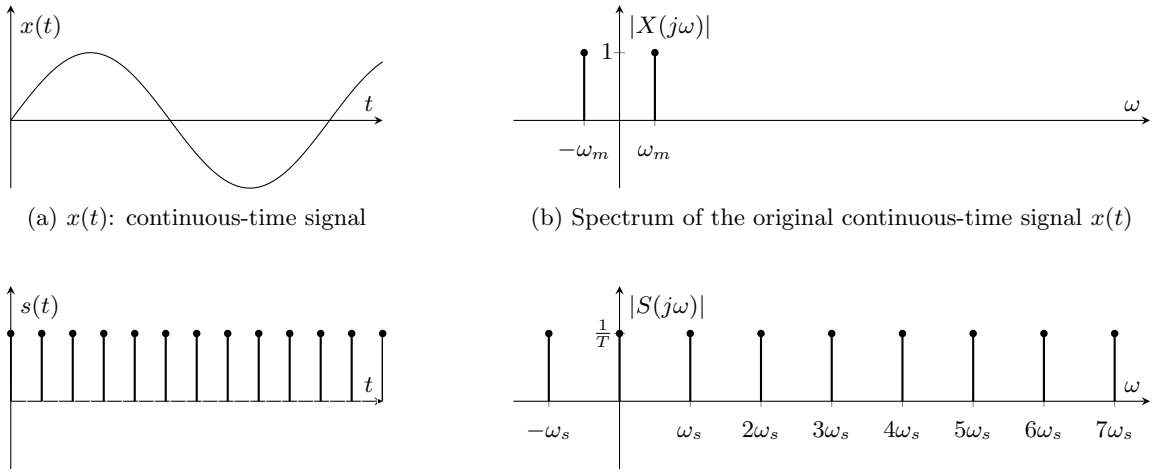


(a) $x(t)$: continuous-time signal (b) $s(t)$: impulse train (c) $x_s(t)$: sampled signal

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 $t \neq nT$. This should not be confused with $x[n]$ which is a discrete-time signal comprising of the amplitudes of the signal at the instants of sampling 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.



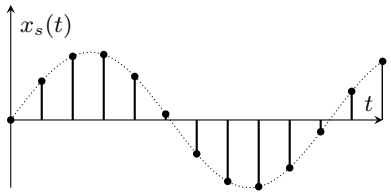
(a) $x(t)$: continuous-time signal (b) Spectrum of the original continuous-time signal $x(t)$

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

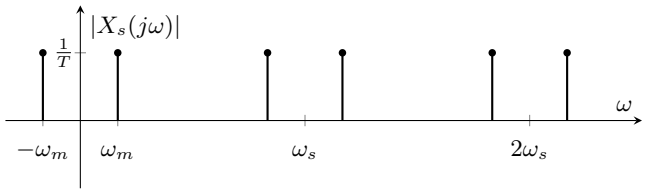
Figure 2: Frequency domain representation of $x(t)$ and $s(t)$

You may be able to recall that multiplication in the time domain is equivalent to convolution in the frequency domain. Therefore $x_s(t) = x(t) \times s(t)$ can be written in the frequency domain 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 at each and every integer multiple of ω_s and, therefore, to avoid information loss due to overlapping spectral images (this distortion is called aliasing) the following condition must be met:

Definition:

$$|\omega_m| < \frac{\omega_s}{2} \text{ (Nyquist frequency)}, \quad \text{where } \omega_s = 2\pi f_s = 2\pi/T \quad \Longrightarrow \quad |\Omega| < \pi$$

The frequency $\omega_s/2$ is called the Nyquist frequency. If ω_m is the highest frequency component in the signal then sampling above twice this frequency, $|2\omega_m| < \omega_s$, is called *oversampling*, conversely, sampling below twice this frequency, $|2\omega_m| > \omega_s$, is called *undersampling*. Lastly sampling at a frequency exactly twice this frequency, $|2\omega_m| = \omega_s$, 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 2: Discuss briefly the results of (i)-(iii) and (iv)-(vi) stating what you have observed.

Question 3: Predict the result for frequencies $f = 24150$Hz, $f = 24300$Hz and $f = 24600$Hz and confirm your prediction with MATLAB.

Question 4: 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 200Hz is used.

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

Question 5: 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 6: 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 Analysis and Synthesis of Signals

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 stored 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 7: Is this plot correct? Why or why not?

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

Note that in MATLAB, '`'`' denotes the transpose operator for real signals. Check what it does for complex signals.

We can do the same thing as the above 4 statements by using a for-loop. The following statements create `S` with 10 rows of sine waves. The frequencies of the sine waves are at integer multiples of the signal frequency and are known as harmonics.

```
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);
x1 = p*S;
plot(x1)
```

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 $n = \{0,\ 1,\ 2, \cdots,\ \text{Nsamp} - 1\}$.

Question 8: 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];
x2 = bk*S;
plot(x2)
```

Question 9: 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 contains 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 `x3=ak*C` and thus plot the result of:

$$x_3[n] = \sum_{k=1}^{10} a[k] \cos(k\Omega n).$$

Question 10: How does $x_3[n]$ differ from $x_2[n]$ obtained earlier using sine waves? What general conclusions on the even and odd symmetry of the signal can you draw?

Question 11: 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 $x[n] = \sum_{k=-\infty}^{\infty} A[k] e^{j\frac{2\pi}{N} kn}$. How can the coefficients $A[k]$ be determined?

GTA Tip

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

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) for periodic signals:

DFS (Discrete Fourier Series):

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

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

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 12: 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 a fundamental period equal to N, that is,

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

where k is the harmonic or frequency bin number and n is the sample number.

To recap, to perform the frequency analysis of a periodic signal, we take one cycle and perform the DFS. 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 process an infinite number of samples. Realistically, therefore, to be able to perform a frequency analysis, we 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). The DFT is a version of the DTFT in which the frequency domain variable Ω is sampled at frequencies $2\pi k/N$, $k = 0, 1, \cdots N-1$.

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): $x[n] \rightarrow X[k]$

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

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

Question 13: Can the Fourier transform modify the energy of a signal?

Question 14: When applying Fourier analysis to a signal, under which circumstances should a DFS be employed and under which circumstances should a DFT be employed?

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.

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

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

Question 15: Explain the numerical result in `A`. Make sure that you know the significance of each number in the MATLAB output. Evaluate the DFT of a cosine signal. Is it what you would expect?

Question 16: What is the frequency resolution of a 256-point DFT when the sampling frequency is 1000 Hz?

GTA Tip

- The first sample of the DFT represents the DC component (0 Hz).
- The last sample of the DFT represents the sampling frequency minus the frequency resolution ($F - \Delta F$).

Instead of working with real and imaginary numbers, it is often more convenient to work with the magnitude and phase of the frequency 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, respectively, 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 `stem` 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 a unit impulse function, i.e. contains only a single one and 15 zeros. Note how the spectrum changes.

Question 17: What do these plots tell us about the frequency components that make up a unit impulse function?

Next delay this unit impulse function by 1 sample and find its spectrum again. Examine the real and imaginary spectra.

Question 18: What happens to the spectrum when the unit impulse function 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 in MATLAB useful).

To test how much you have understood so far, you are given a real electrocardiogram (EGC) signal. The signal contains 500 data samples and was sampled at 200 Hz. Our aim is to detect the heart rate of the patient from which this ECG was taken.

First, download the file, `ecg.mat`, from blackboard into your working directory. Then load the data into a vector using the MATLAB command:

```
load("ecg.mat");
```

First plot the ECG signal:

```
fsamp = 200;
Nsamp = length(ecg);
Tsamp = 1/fsamp;
t = 0:Tsamp:(Nsamp-1)*Tsamp;

plot(t, ecg)
title('ECG')
xlabel('Time [s]')
ylabel('Amplitude')
```

Now let's remove the DC component from the ECG signal using the frequency domain:

```
ECG = fft(ecg);
ECG(1) = 0; % remove the 0 Hz component
ecg_no_dc = ifft(ECG);
```

Question 19: Can you think of a way of removing the DC component in the time domain without using the DFT? Why not try to implement this time domain approach in MATLAB.

Let's plot the ECG signal with no DC offset:

```
plot(t, ecg_no_dc)
title('ECG (DC offset removed)')
xlabel('Time [s]')
ylabel('Amplitude')
```

Question 20: Plot the spectra of the `ecg_no_dc` signal. Make sure that you can interpret the frequency axis. Deduce the the person's heart rate in beats per minute. Does this result make sense when compared to your plot of `ecg_no_dc` in the time domain?

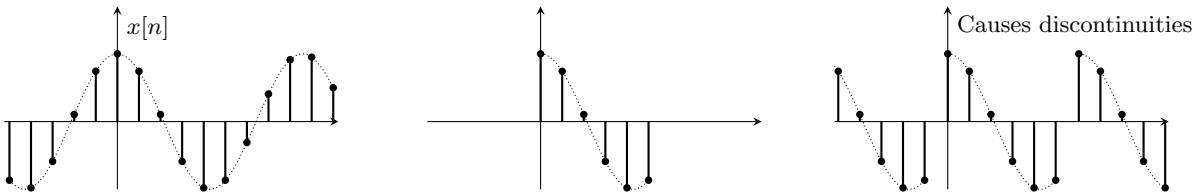
GTA Tip

The ECG signal is quasi-periodic and at each period the heart cycles through both a systole and diastole phase. In the frequency domain you will not just see the heart rate, but also a series of successively higher frequencies called 'harmonic frequencies'. Therefore, to calculate the frequency of the heart you first need to detect the multiple peaks generated in the frequency domain which will all be multiples of the fundamental frequency, F_0 (i.e. 1^{st} peak $= F_0$; 2^{nd} peak $= 2 \times F_0$; 3^{rd} peak $= 3 \times F_0$; and so on).

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 at 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:

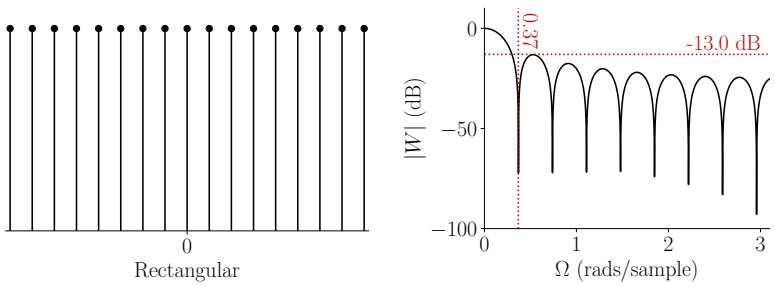
```
x = sinegen(20000,1000,128);
```

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. Four common window functions are defined mathematically as:

Rectangular:

$w[n] = \delta_{-\frac{M}{2} \leq n \leq \frac{M}{2}}$

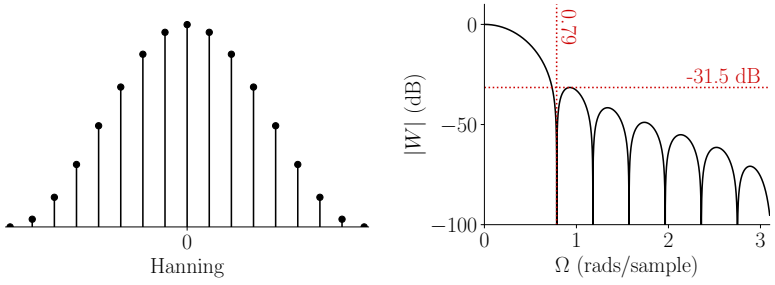
Main lobe width: $4\pi/(M+1)$
Relative sidelobe level: 13.0 dB
Never use



Hanning:

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

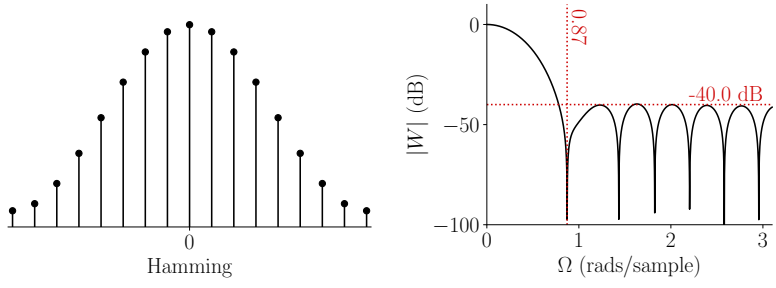
Main lobe width: $8\pi/(M+1)$
Relative sidelobe level: 31.5 dB
Rapid sidelobe decay



Hamming:

$w[n] = 0.54 + 0.46\cos\frac{2\pi n}{M+1}$

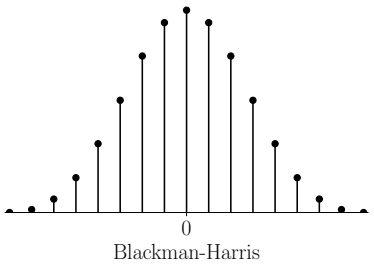
Main lobe width: $8\pi/(M+1)$
Relative sidelobe level: 40.0 dB
Best peak sidelobe

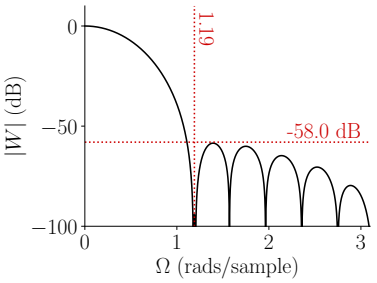


Blackman-Harris (3-term):

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

Main lobe width: $12\pi/(M+1)$
Relative sidelobe level: 58.0 dB
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 your sine wave multiplied by this Hanning window function. Compare the 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 and discuss the differences. State a reason why the Hanning window would be chosen in preference to the Blackman-Harris window. State a reason why the Blackman-Harris window would be chosen in preference to the Hanning window.

Question 22: Recall that multiplication in the time domain is equivalent to 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)*

To test how much you have understood so far, you are given an unknown signal, `unknown.mat` (which can be downloaded from blackboard into your working directory). The file contains 1048 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.mat")
```

Question 24: Take at least two different 256-sample segments from this data, starting at different chosen starting sample numbers, and compare their spectra. Are they essentially the same? Make sure that you can interpret the frequency axis. Deduce the frequency and magnitude of its constituent sine waves.

Now let's see how windowing is used in a real world example. Start by downloading the '.wav' file `piano.wav` from blackboard into your working directory.

Why not listen to the `piano.wav` file. It is a 6 second long recording that contains 3 notes played at 2 second intervals on a grand piano. *(The first note is played at 0 seconds then the second note is played starting at 2 seconds and the third note is played starting at 4 seconds.)*

We are now going to do the job of an electronic tuning device by using windowing and the frequency domain to workout which notes are being played.

First, we need to read this '.wav' file into MATLAB in the following way:

```
[data, fsamp] = audioread('piano.wav');
Tsamp = 1/fsamp;
```

Now let's set up our windowing parameters to investigate the second note that was played 2 seconds into the recording:

```
win_loc_sec = 2.5 % 0.5 seconds into second note
win_size_sec = 0.200  % 200 ms frame
win_loc = win_loc_sec/Tsamp;
win_size = win_size_sec/Tsamp;
```

We can now plot this frame in both the time and frequency domain without using a window:

```
x = data(win_loc:win_loc+win_size-1);

Nsamp = length(x);
t = 0:Tsamp:(Nsamp-1)*Tsamp;
plot(t, x);

X = fft(x);

f_step = (0:(fsamp/nsamp):(fsamp-(fsamp/nsamp)));
plot(f_step, abs(X)); % plot spectra
xlim([0 1000]) % limit spectra x-axis between [0, 1000] Hz
```

Question 25: Now try using a window to reduce the spectral leakage and try to find the frequency and the corresponding musical note value that is being played. For the musical notes, use scientific pitch notation as tabulated here: https://en.wikipedia.org/wiki/Piano_key_frequencies.

GTA Tip

Recall what we learned about harmonics from the ECG example in Exercise 3.

Question 26: See if you can work out the other 2 notes that were played.

Hint: use `winlocsec = 0.5` (0.5 seconds into the 1st note), `winlocsec = 2.5` (0.5 seconds into the 2nd note) and `winlocsec = 4.5` (0.5 seconds into the 3rd note)

Exercise 5 - 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. This is referred to as frequency selectivity. In this lab, 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 such as resistors, capacitors, inductors and op-amps.

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

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

Let's filter part of a song, `track_1.wav`, by removing all components above 300 Hz. Compare the filtered waveform and audio with the original signal. Start by downloading the `track_1.wav` file from blackboard into your working directory and run the code below:

```
[x,fsamp] = audioread('track_1.wav');
Tsamp = 1/fsamp;
Nsamp = length(x);

X = fft(x);

cutoff = 300;
n = round(cutoff/fsamp*Nsamp); % calculate the frequency index

f_step = (0:(fsamp/nsamp):(fsamp-(fsamp/nsamp)));
plot(f_step, abs(X))
title('DFT')
xlabel('Frequency [Hz]')
ylabel('Amplitude')

X(n+1:end-n+1) = 0; %+1 due to MATLAB indexing starts from 1
plot(f_step, abs(X))
title('DFT (after filtering)')
xlabel('Frequency [Hz]')
ylabel('Amplitude')

y = ifft(X);

audiowrite('track_1_lowpass.wav', y, fsamp);
```

Question 27: Now try to modify the code above to create a high-pass filter by removing all components below 300 Hz. Save the result as `track_1_highpass.wav` and listen for the differences.

GTA Tip

Note that the filtered waveform should still be real. If your result is not real it is because you have destroyed the conjugate symmetric property in the frequency domain that makes the signal real in the time domain.

Question 28: What are the disadvantages of filtering in the frequency 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 $X[k]$ (and vice versa).

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

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

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

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

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

Exercise 6 - Impulse response and convolution

Objectives

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

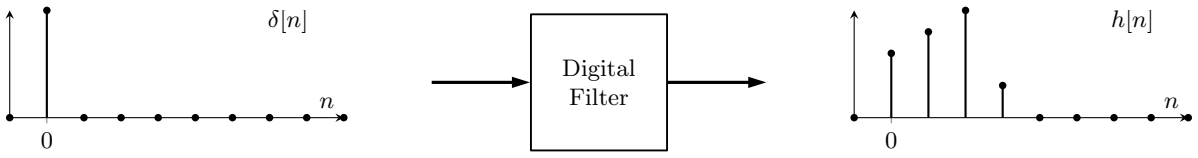


Figure 5: Impulse response of a filter

Suppose we apply a unit impulse function, $\delta[n]$, as an input into a filter. The output denotes the impulse response of the digital filter, $h[n]$, shown in Fig. 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 $\delta[n-k] + \delta[n-(k+1)] + \delta[n-(k+2)]$.

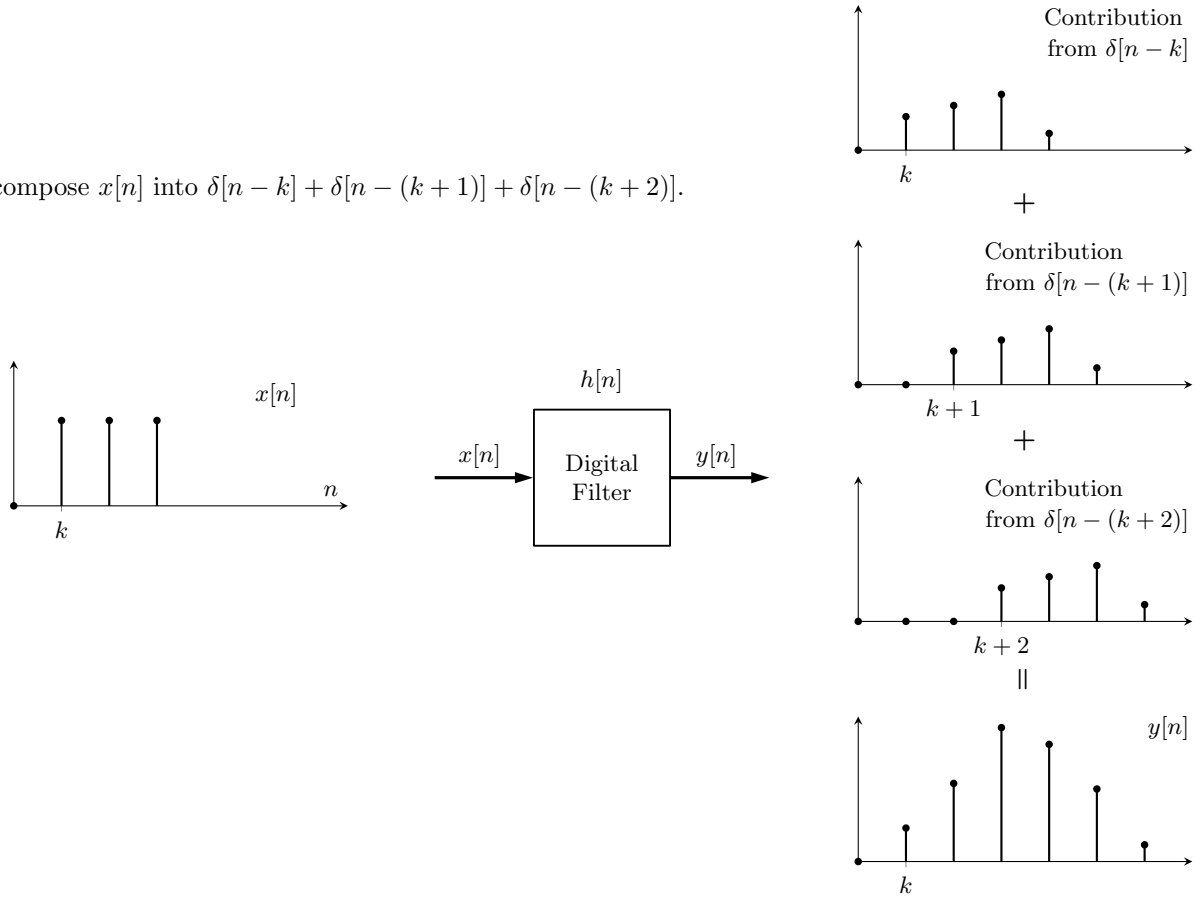


Figure 6: Linear convolution in time domain

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 shown in Fig. 6.

Question 29: What information is needed in order to compute the output of a discrete-time LTI system?

Question 30: 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 replicate the result of Fig. 6 using:

```
x = [0 1 1 1 ];
h = [0.6 0.8 1 0.3];
y = conv(x, h);
stem(y);
```

We have already said that convolution in the continuous-time domain is equivalent to multiplication in the frequency domain. This has been shown in Fig. 7, therefore, calculate the result of 'y' using only the `fft` and `ifft` functions.

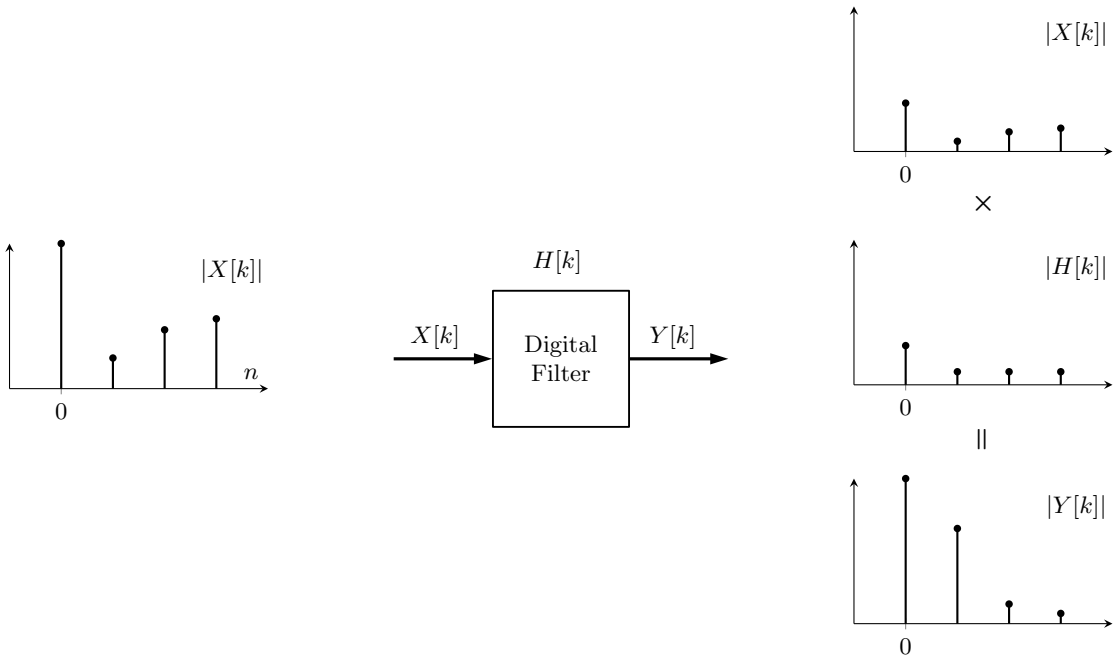


Figure 7: Circular convolution in frequency domain

Question 31: Explain why $y[n]$ computed using the two methods, linear convolution and the DFT, 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 32: 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 'y' using only the `fft` and `ifft` functions correctly?

GTA Tip

Google is your friend (https://uk.mathworks.com/help/signal/ug/linear-and-circular-convolution.html).

Question 33: What differences are there, if any, between linear convolution of a periodic signal and circular convolution?

Exercise 7 - Finite Impulse Response (FIR) filters

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 a constant at all frequencies (flat). Therefore, the frequency response of a filter is simply the discrete Fourier transform of the filter's impulse response.

We have already tried one method of filtering in Exercise 5 which involved setting all unwanted frequency components to zero. However this is not the only, nor most direct, way to filter a signal. In this exercise we are going to look at a different and often better approach using Finite Impulse Response (FIR) filters.

Definition:

$$\text{Finite Impulse Response (FIR) Filter: } y[n] = \sum_{k=0}^{M} b[k]x[n-k]$$

The impulse response of FIR filters are defined as $b[n]$ with a length of $M+1$. Let us assume that the impulse response of the filter is a sequence of 4 ones.

Question 34: Describe the frequency response, $B(e^{j\Omega})$, of this filter, $b[n] = \{1,\ 1,\ 1,\ 1\}$, using `plotspec`. What do you notice about the phase?

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

Question 35: Suppose the input signal to the filter is $x[n]$, show that this filter produces the output $y[n] = x[n] + x[n-1] + x[n-2] + x[n-3]$.

Now let's create a moving average filter and use it on some real data, namely the number of daily Coronavirus (COVID-19) cases in the UK between 2020-01-30 and 2020-06-29. Start by downloading `coronavirus.mat` from blackboard and plotting the data in MATLAB:

```
load("coronavirus.mat")

t = datetime(2020,01,30) + caldays(0:length(data)-1);
stem(t, data)
grid
title('UK Coronavirus Cases')
xlabel('Time [days]')
ylabel('Amplitude [no. cases]')
```

Implement a 4 day moving average filter (Hint: $b[n] = \frac{1}{4}\{1,\ 1,\ 1,\ 1\}$) which can be done using the `conv(x, b)` function to perform a convolution between the signal, $x[n]$, and $b[n]$.

Question 36: Compare the waveform before and after filtering. Do you see what you would have expected? Now modify $b[n]$ to implement a 7 day moving average filter.

Now let's try and create some more impressive FIR filters using MATLAB's Filter Designer. First run the code below where the FIR coefficients have already been calculated for you using MATLAB's Filter Designer and stored in `aircrew_coef.mat`. The code applies a band-stop filter to the audio file `aircrew.wav` (download both '`aircrew_coef.mat`' and '`aircrew.wav`' from Blackboard).

```
[x,fsamp] = audioread('aircrew.wav');
Tsamp = 1/fsamp;
Nsamp = length(x);
t = 0:Tsamp:(Nsamp-1)*Tsamp;

load('aircrew_coef.mat')
y_bs = conv(x, b_fir_bsf);

audiowrite('aircrew_fir_bsf.wav', y_bs, fsamp);
```

Now try to calculate these coefficients yourself using MATLAB's Filter Designer. To start type and run `filterDesigner` on the MATLAB command prompt: `>> filterDesigner`. A GUI should appear (see Fig. 8) displaying a default filter.

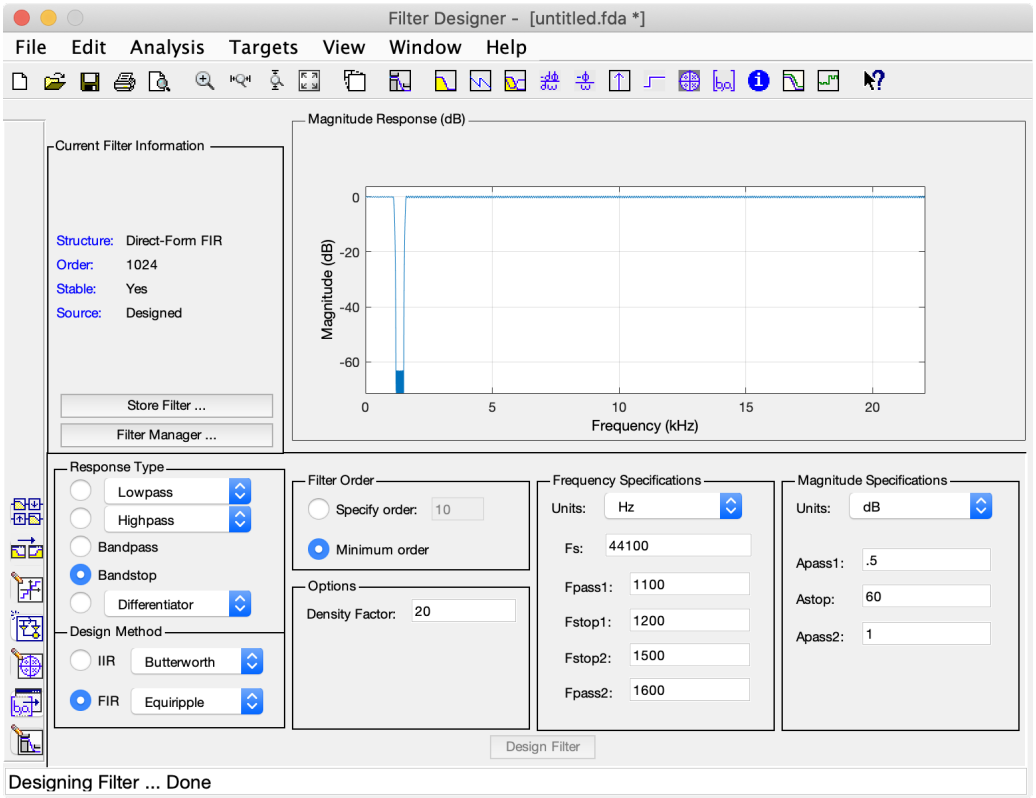


Figure 8: Bandstop Filter Design

Change the default settings to those shown in Fig. 8 and then click the **Design Filter** button. Then click **File** and select **Export...** from the drop-down list. A export GUI should appear. Select **Workplace** for 'Export To' and **Coefficients** for 'Export As'. Choose a sensible variable name ('Numerator'), e.g. `my_b_fir_bsf`, to store the coefficients. When you are happy with the selected settings click **Export**.

If you check your MATLAB 'Workspace' on the right-hand side of the MATLAB window you should now see your new variable storing the coefficients of the band-stop filter that you have just designed. To save these coefficients as a '`.mat`' file so they are not lost when you close MATLAB just right click on the variable, `my_b_fir_bsf`, in the 'Workspace' and select **Save As** from the drop-down list. Regenerate '`aircrew_fir_bsf.wav`' using the coefficients you have just exported. You should hear the same result.

You have probably notice that this recording also contains a lot of background noise at high frequencies. Let us now design a low-pass filter using Fig. 9 to create the coefficients, `b_fir_lpf`, for the low-pass filter.

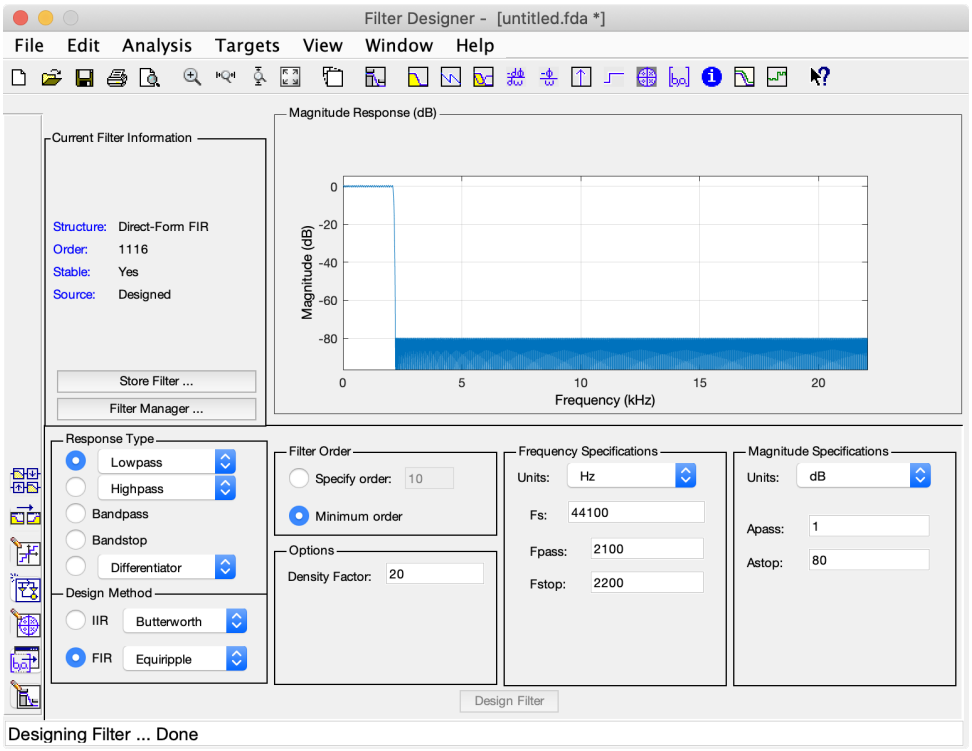


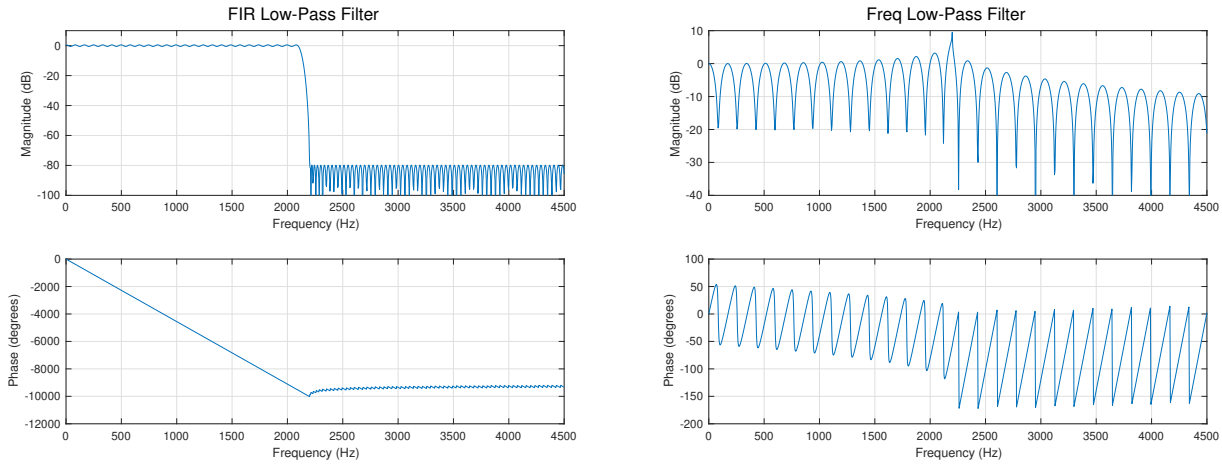
Figure 9: Lowpass Filter Design

```
y_lp = conv(y_bs, b_fir_lpf);
audiowrite('aircrew_fir_bsf_lpf.wav', y_lp, fsamp);
```

Let's compare this result with the filtering approach used in Exercise 5. To save time the complete code to implement this approach has been given to you below:

```
[x,fsamp] = audioread('aircrew.wav');
Tsamp = 1/fsamp;
Nsamp = length(x);
t = 0:Tsamp:(Nsamp-1)*Tsamp;

X = fft(x);

cutoff = 2200;
band_start = 1200;
band_end = 1500;
n = round(cutoff/fsamp*Nsamp); % calculate the frequency index
n_start = round(band_start/fsamp*Nsamp);
n_end = round(band_end/fsamp*Nsamp);

X(n+1:end-n+1) = 0; % low-pass
X(n_start+1:n_end+1) = 0; % band-stop (1)
X(end-n_end+1:end-n_start+1) = 0; % band-stop (2)

y = ifft(X);

audiowrite('aircrew_freq_bsf_lpf.wav', y, fsamp);
```

To truly understand the difference between these two approaches (Exercise 5 vs FIR) we need to examine their frequency responses which have been given to you in Fig. 10.



(a) FIR low-pass filter

(b) Freq low-pass filter (Exercise 5 appoach)

Figure 10: Performance Comparison: FIR filtering vs Exercise 5 approach

Question 37: The performance using the approach from Exercise 5 is worse but can you explain why? Are you able to hear the difference between '`aircrew_fir_bsf_lpf.wav`' and '`aircrew_freq_bsf_lpf.wav`'?

To understand how Fig. 10(b) is calculated the code used to produce it is given below:

```
fsamp = 44100;
Nsamp = 100000;

cutoff = 2200;
n = round(cutoff/fsamp*Nsamp);

H = ones(1, Nsamp); % all-pass filter in the frequency domain
H(n+1:end-n+1) = 0; % low-pass (sets unwanted frequencies to 0)
h = ifft(H); % impulse response of the filter H

limit = 4500; % only view frequency response between [0, 4500] Hz
w = linspace(0, limit, 1024);
freqz(h, 1, w); % plot frequency response
freqz(h, 1, w, fsamp); % limits range of y-axis
sgtitle('Freq Low-Pass Filter')
```

Question 38: In Fig. 10(b) the frequency response is perfect at the sample points but not between them. Can you explain why?

Exercise 8 - Infinite Impulse Response (IIR) filters

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, $y[n]$ may be computed using a recursive form, as a sum of present and past input values and past output values.

In this exercise we will explore a new type of filter which 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 old outputs are being fed back to calculate the new output. They are described by the following equation:

Definition:

$$\text{Infinite Impulse Response (IIR) Filter: } y[n] = \sum_{k=0}^{M} b[k]x[n-k] - \sum_{k=1}^{N} a[k]y[n-k]$$

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` $= \{1\}$.

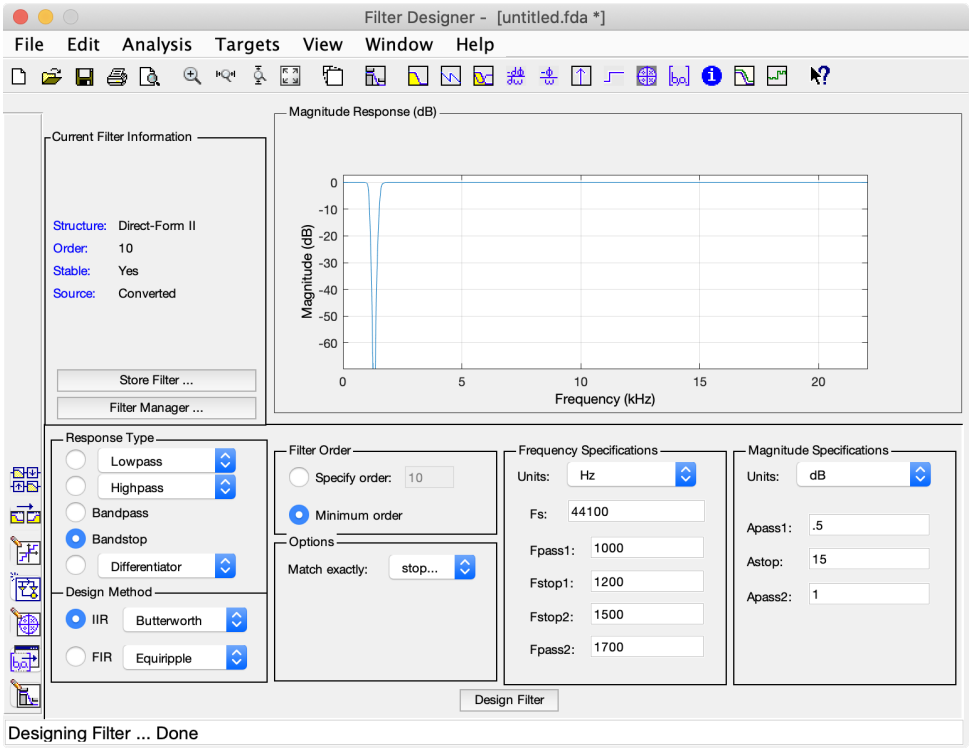


Figure 11: Bandstop Filter Design

Now let's try to filter `aircrew.wav` using IIR filters. To start with lets run `filterDesigner` and use the specification shown in Fig. 11. For IIR filters before you export the coefficients you will want to convert the filter structure from second-order sections to a single section. To do this click **Edit** and select **Convert to Single Section** from the drop-down list. Check that 'Source: Converted' is now displayed on the left-hand side.

We can now export the coefficients in the same way as in Exercise 7 (see Fig. 12)

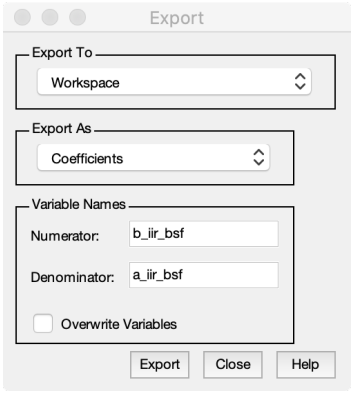


Figure 12: Export

Next use the `filterDesigner` and Fig. 13 to generate coefficients for the low-pass filter which is required to remove the high frequency background noise from the signal.

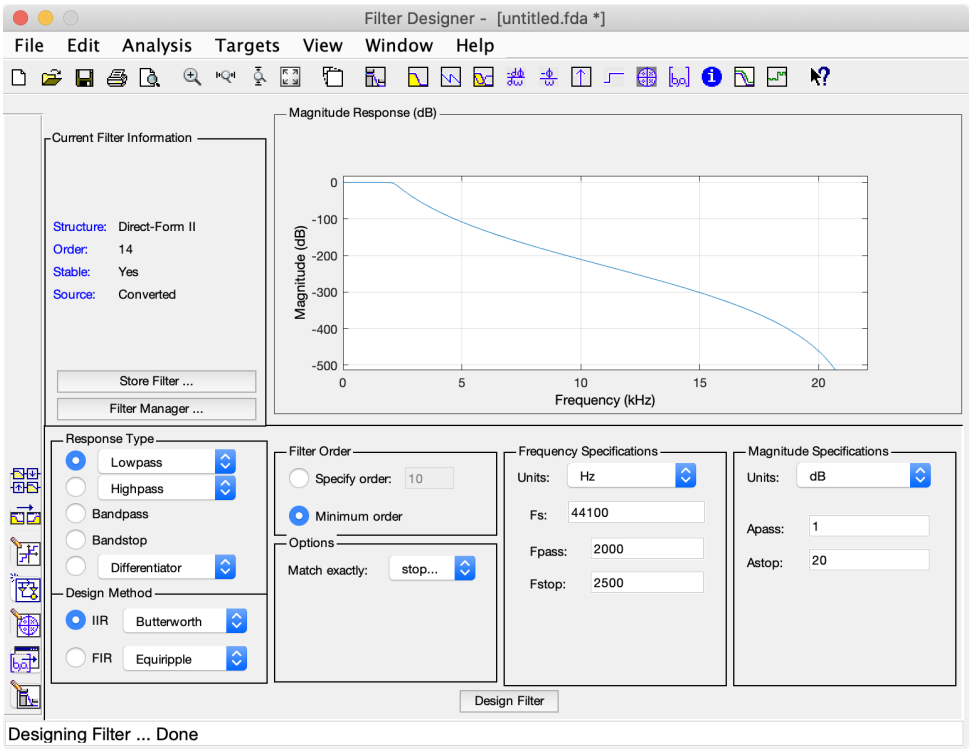


Figure 13: Lowpass Filter Design

Let's now listen to the effect of the two IIR filters that we have created. To do this copy and run the code snippet below. *(You may need to change `a_iir_bsf`, `b_iir_bsf`, `a_iir_lpf` and `b_iir_lpf` if you have named the coefficients that you just exported differently.)*

```
[x,fsamp] = audioread('aircrew.wav');
Tsamp = 1/fsamp;
Nsamp = length(x);
t = 0:Tsamp:(Nsamp-1)*Tsamp;

y_bs = filter(b_iir_bsf, a_iir_bsf, x);
y_lp = filter(b_iir_lpf, a_iir_lpf, y_bs);

audiowrite('aircrew_iir_bsf.wav', y_bs, fsamp);
audiowrite('aircrew_iir_bsf_lpf.wav', y_lp, fsamp);
```

Question 39: Compare and contrast the frequency response of the FIR and IIR low pass filters you have just created. Does the frequency response make sense of what you hear when you listen to the filtered signal?

A simple way to compute and plot the frequency response of the IIR filter you have just created is:

```
fsamp = 44100;

limit = 4500; % only view frequency response between [0, 4500] Hz
w = linspace(0, limit, 1024);
freqz(b_iir_lpf, a_iir_lpf, w, fsamp); % plot frequency response
ylim([-100 10]) % limits range of y-axis
sgtitle('IIR Low-Pass Filter')
```

Question 40: Derive the impulse response of: $y[n] = 0.5x[n] + 0.5y[n-1]$?

Question 41: Why does an IIR filter have to be implemented recursively (i.e. does it absolutely need to use past output values)?

Question 42: Compare the advantages and disadvantages of FIR and IIR filters (Hint: look at the number of coefficients required to implement both types of filters, e.g. `b_fir_bsf`, `a_iir_bsf` and `b_iir_bsf`).