Signal Processing Practical 2

140012021

November 2018

1 Introduction

The concept of the practical was to use the programming language MATLAB to solve different signal processing tasks, then analyse the results. Through completing this practical, experience would be gained in the area of processing signals in the domains of time and frequency. Familiarity of MATLAB would also be gained, a very common programming environment for signal processing. The practical was broken down into different questions, which involved using different sounds files.

2 Background Reading

2.1 Fast Fourier Transform

An important concept for this practical would be Fast Fourier Transform (FFT). FFT was derived from the Discrete Fourier Transform (DFT). The DFT is obtained by decomposing a sequence of values into components of different frequencies. In reference to digital signal processing the function is any type of signal over time, from waves to temperature. The complexity of the DFT algorithm is O(n^2), which grows exponentially as the data size increases and with modern data sets, containing millions of values this could be an issue. A faster algorithm was looked into during the early 19th century, but never completed. In 1965 Cooley and Tukey published the FFT method. FFT complexity was lower when compared to DFT, with it being O(nlog n). FFT does this by quickly calculating transformations by factorising DFT matrices into a product of sparse factors.

2.2 Frequency Mirroring

This is phenomenon that occurs when the FFT is preformed on real signals. A Fourier transform is defined in figure 1

$$H(f) = \int h(t)e^{-j2\pi ft}\,dt$$

Figure 1: Fourier transform Equation.

The equation correlates the signal with complex sinusoids, which each have their own frequencies.

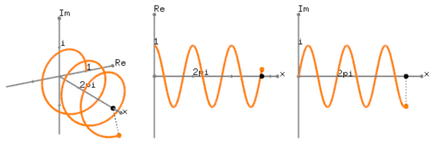


Figure 2: Frequency Mirroring with Imaginary and Real graphs.

In figure 2 we see what the complex sinusoids look like when graphed. The corkscrew is the sinusoid in time, with the two graphs to the side being the imaginary and complex components of it. The imaginary components that allows things to be viewed on an extra plane, which is easier than complex differential equations. However we cannot preform anything real on the imaginary parts. For example, you cannot have a negative frequency of a signal and because of this it needs to cancel itself out. So in a FFT of a time based signal, if we add the positive and negative components of the signal, the imaginary parts cancel out. When we graph a FFT these conjugate signals exist, with the real values part of each 'sharing the magnitude, half in the positive domain, half in the negative, so in effect adding the conjugates together removes the imaginary content and provides the real content only.'[4]

3 Implementation

The following sections represent the different sub-questions of the practical.

3.1 Question 1 - Middle C

The first piece of code to be produced was used to create a middle C tone for a period of 10 seconds. The frequency of middle C is 261.626[6]. A variable was created to store this value. To double check that this was correct, a tone of a middle C was played from a tone website[3]. This was compared with the audioplayer() built in function in MATLAB.

Next the note had to have a period of 10 seconds. When recording or playing any piece of sound, there needs to be a sampling frequency, which is how many samples of the sounds is stored every second. For CD quality this is around

44100Hz. If the sampling frequency is 44100Hz and the recording has a duration of 60 seconds this equates to 2646000 samples. A sampling frequency of half that will lose some of the detail of the track and will mostly keep the speech, as the frequency of speech is around 11025Hz. As the sampling frequency goes down you lose detail but in this question, there is just a single tone, therefore there is not a huge difference in using a sampling frequency of 44100 and 10000. So after defining a sampling frequency, this could be used for creating a time array that is same length as the data vector, which would allow us to create a ten second recording of a middle C note. The array was defined using a range with steps in the size of 1/sampling frequency. By going from 0 to 10 in these steps we can use this along with the frequency to create an array of data to graph with.

The data would be graphed in two domains, time and frequency. When graphing in these domains, we use sinusoidal curves to keep all the values in our new array between the values of -1 and 1.

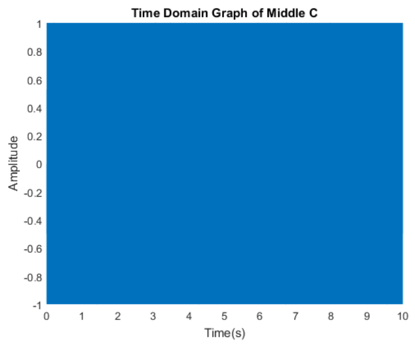


Figure 3: Time Domain graph of middle C from task 1.

In figure 3 above, due to the size of sampling and the time duration it is hard to see any detail in the output. In MATLAB you can use a zoom tool to see extra detail in the graph. Once zoomed in using the 'zoom in' tool on the figure output window more detail could be seen in the diagram.

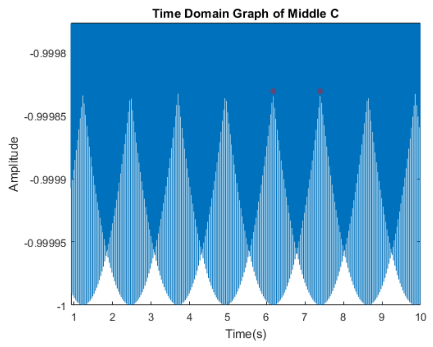


Figure 4: Time Domain graph of middle C from task 1 with zoom.

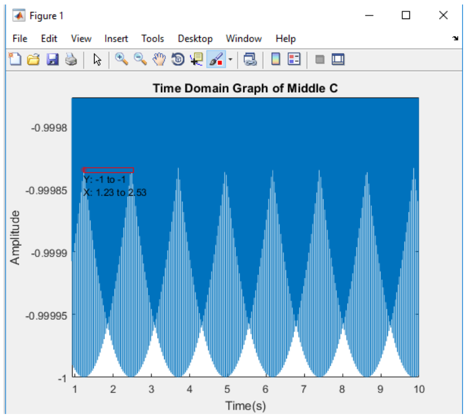


Figure 5: Time Domain graph of middle C from task 1 showing incorrect peaks.

In figure 5 the 'Brush/Select Data' was used to draw the distance between peaks. This was calculated to be around 1.3. However the value that was expected should be able to produce back the frequency that was coded for the Middle C. When placed into the $T = 1/f$ equation it did not line up. This suggests that the peaks that are in the picture are not all peaks and that they don't represent the period of the audio. In order to calculate the actual period, it was then decided to do it mathematically inside MATLAB. Inside MATLAB there is a built in function for identifying peaks on waves, called findpeaks(). The function returns the indices that the peaks appear at, which we then use to calculate the period of the signal. This returns 0.0038 which when used in the $T = 1/f$ equation equals the frequency of middle C. So even though it appeared as a solid sound wave when listening to it, little oscillations are occurring. When listening back to the noise I feel like these are heard and it does not sound like a solid tone. However this could be placebo affect after discovering the results of the period.

In order to plot the data in the frequency domain, the previously mentioned FFT would be used. There is a built in FFT function in MATLAB which was used called FFT().

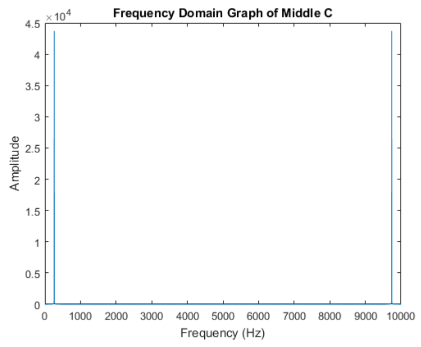


Figure 6: Frequency Domain graph from task 1, showing frequency mirroring.

In Figure 6 above the overall frequency graph that was produced is shown. In this graph there is the previously mentioned mirroring effect. Again similar to the time domain in the first question, once zoomed in we can see more detail.

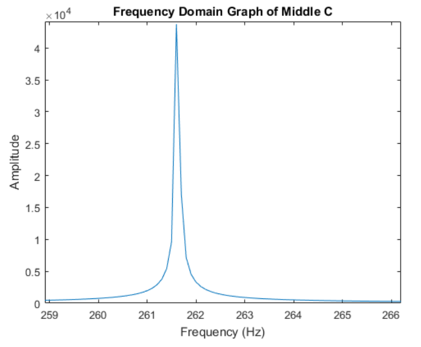


Figure 7: Frequency Domain graph from task 1 with zoom.

As the task was to create a tone of middle C and we know the exact frequency of it, there is no surprise that the peak of the spike appears right above the 261.626 value. We also confirmed while inspecting the time domain that the frequency must be 261.626Hz as the period of the signal relates to this number. Therefore you don't need to frequency domain to get a single value, but when looking at multiple frequencies, such as the audio files in the later sections it can be incredibly useful.

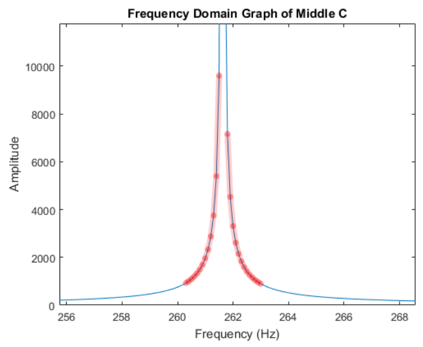


Figure 8: Frequency Domain graph from task 1, showing data points.

Finally the reason the frequency domain is not just a single line, but a curve is due to the fact that multiple points are used to create the curve, which can be seen in figure 8.

3.2 Question 2 - Voice and Music Time Domain

For this question the voice and music files needed to be plotted in the time domain. The clips had to be around ten seconds in length. The voice clip was recorded using a laptop and the music clip was Mr Blue Sky by ELO an MP3 taken from a personal music library. After reading in the music file, it had to be cut to ten seconds in length, which was done by manipulating the array the sound file's data had been read into. For every second of the track there the same number of entries as the value of the sampling frequency. Therefore the new array uses the FS value to cut it to ten seconds.

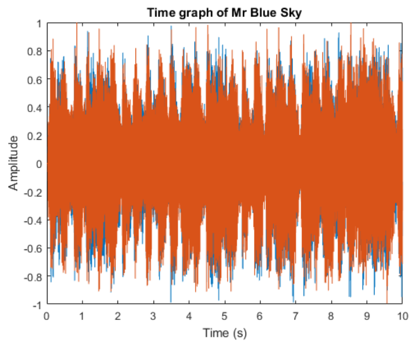


Figure 9: Time Domain graph from task 2 of Mr Blue Sky.

After preforming the preprocessing, figure 9 was created. The two colours on the graph, is due to the audio having two channels. In speaker devices there are normally two settings that it can have, Mono or Stereo.

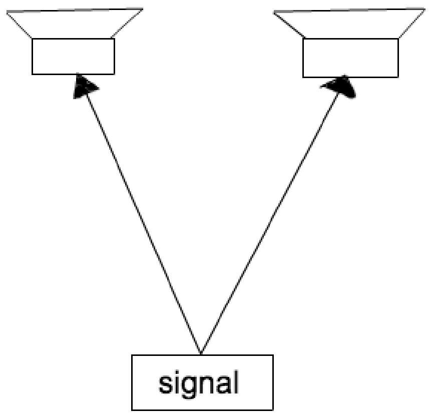


Figure 10: Mono Speaker set up.

Mono is where there is one single channel used. It can be played through multiple speakers but it sounds the same through all the speakers.

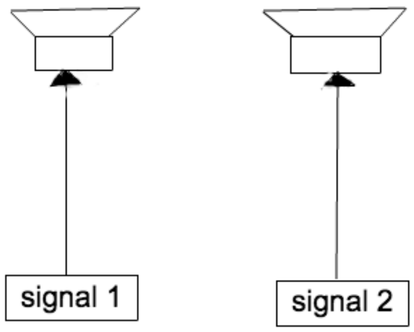


Figure 11: Stereo Speaker set up

In stereo multiple channels are used, but normally two. Each channel goes to a speaker, which makes the music directional to the user.

The peaks of the graph are the various words that are sung and sharp beats of the music. This differs from the time domain graph the middle C graph created. That was a constant noise so the graph was completely solid from normal. Because the music is created using various items that make noise (synthesizer, voice, instruments etc) there are gaps and peaks on the graph.

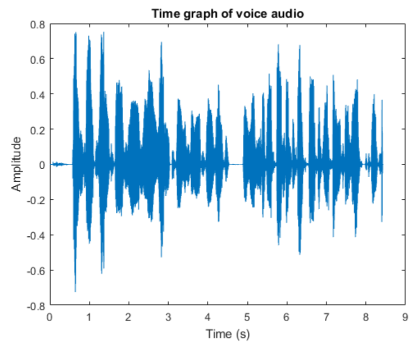


Figure 12: Time Domain graph from task 2 of voice recording.

Figure 12 is the time domain graph for the voice recording, is a lot smaller than the graph for the music recording, which is as expected. The voice recording amplitude goes towards zero after every word is spoken. Each spike is a word that is spoken in the recording, which you can follow as the recording plays. In contrast to the previous graph where it is hard to pick out individual sounds.

3.3 Question 3 - Voice and Music Frequency Domain

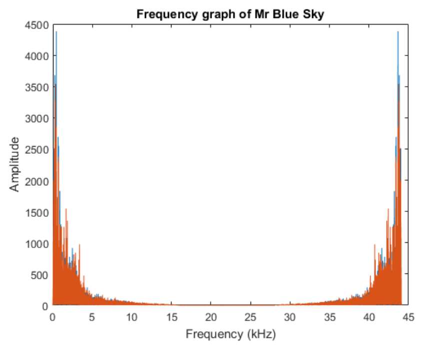


Figure 13: Frequency Domain graph from task 2 of Mr Blue Sky.

Again similar to the previous questions a FFT was preformed in order to transform it from the time to the frequency domain. The standard audio CD sample rate of 44.1kHz, allows up to around 20kHz of response frequency. "Audio that's encoded at 22.05kHz has a high-frequency limit of about 10kHz, and audio encoded at 8kHz only up to about 4kHz."[5] After inspecting the frequency graph of the music we can see that the majority of the spikes in the frequency domain occur around and before 5kHz. This suggests that a common lossy audio compression has been used around the 8kHz range.

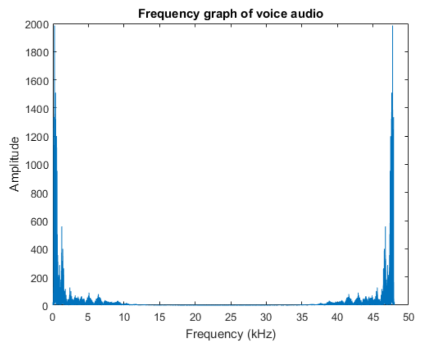


Figure 14: Frequency Domain graph from task 2 of voice recording.

In the figure 14 we see that the frequency spikes are further down the axis. When zoomed in on the graph, you can see the range of frequencies clearer.

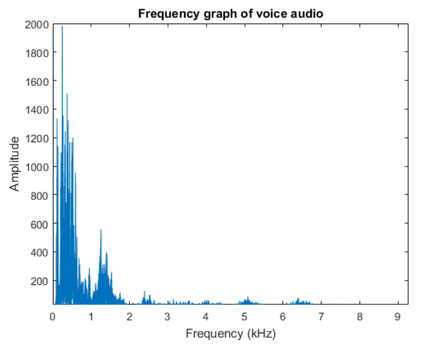


Figure 15: Frequency Domain graph from task 2 of voice recording with zoom.

The reason the frequencies appear further down the axis, is due the range of the human voice being much lower than that of a musical piece, as well as having a smaller range. While in conversation the human voice will not go through massive frequency changes that a operatic singer might do in comparison.[8]

3.4 Question 4 - Filtering the Audio Files

3.4.1 Music File

The audio file contained the song 'Mr Blue Sky', which involves noises form different instruments. There is a symbol drum being used in the background as well as a Piano playing on the lower end of its scale. To see if certain instruments could be removed or not from the music clip different filters were used. A high-pass filter was used, which cuts off frequencies that are lower that the cutoff value that is given. High-pass filters are often used to get rid of the rumbling in a soundtrack, such as noise that has accidentally been recorded in the background. The filter that was created cuts off all frequencies below the value of 2kHz. It is not practical to have a square frequency band filter, it needs time to get back down to allow frequencies again, so this value was 2.5kHz, which is where the signal returned to normal. When this filter is applied to the audio file of the music, the song sounds different. The vocal of the song can be heard very faintly, but the symbol drum sounds very loud. The low piano sounds are gone as well as other bass noises.

Another filter was made in a similar fashion to the high-pass filter but this time it was low-pass filter, which cuts off sounds above a certain frequency. The values used for the filter was were 800Hz with the frequency for the signal to return to normal being 1200Hz. When this filter was applied to the audio piece, the high symbol drum cannot be heard at all and was successfully removed.

A band-pass filter was then used to try and limit the voice, using a combination of the values already used. When this filter was applied to the signal, the voice appears quieter than before the filtering, the other instruments also appear louder, due to the comparison with the other recording.

3.4.2 Voice file

The human voice is over a range of frequencies, but it was decided to see if all that range would be required to understand the message. The first filter that was used was a low-pass filter. This would leave the signal with the lowest points of the human voice. A low-pass filter was created using the filterDesigner, setting the Fpass at 100 and the Fstop at 300.

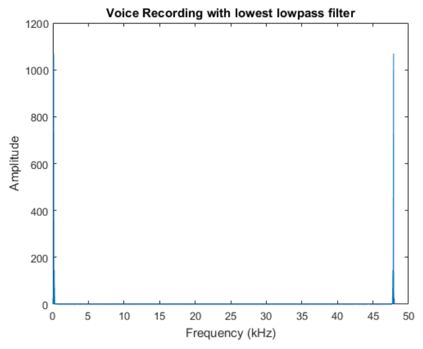


Figure 16: Frequency Domain graph of voice recording with most extreme low-pass filter.

The result of using a filter that low, means that the voice now does not have detail to it. It is a low mumbling tone, where you can still perceive that the pattern of the words is still the same, but you cannot tell words that are being spoken.

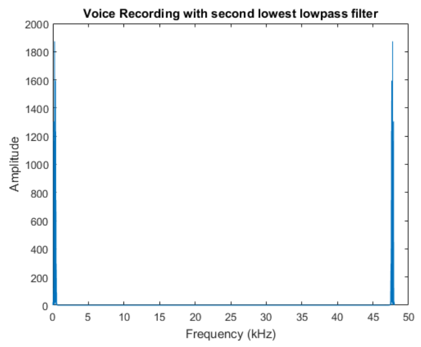


Figure 17: Frequency Domain graph of voice recording with other low-pass filter.

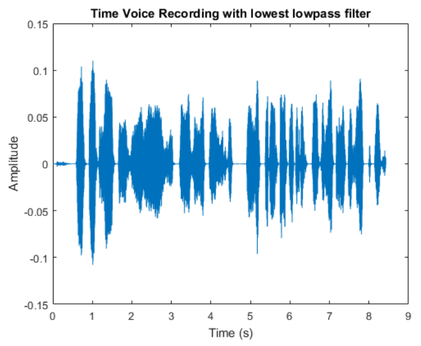


Figure 18: Time Domain graph of voice recording with other low-pass filter.

Another low-pass filter was then used as can be seen in figure 18 which used an Fpass of 400 and Fstop of 600. The voice recording with this filter applied is clearer than the first one. Not all words are clearly visible, but there are words visible. The words in the original recording that were spoken with more clarity, and have more differentiating syllables are the ones that can be understood more, but not enough to be used as a way to play the audio. We can see this in the time graph where the peaks of each word are not as high, showing that some of the information to know the word being spoken, is missing.

Next high-pass filters were applied to the audio of the voice, removing the noises we previously kept in using the low-pass filter. The high-pass filter should also remove background noise, as in recordings this is normally lower frequency noises. The first high pass filter that was used, had an Fstop of 4000 and a Fpass of 4200.

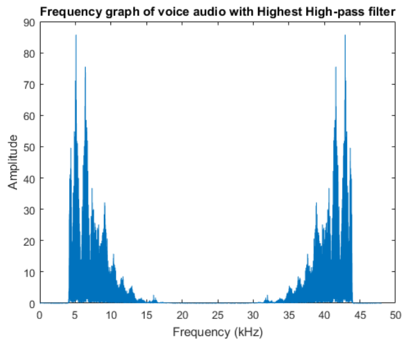


Figure 19: Frequency Domain graph of voice recording with most extreme high-pass filter.

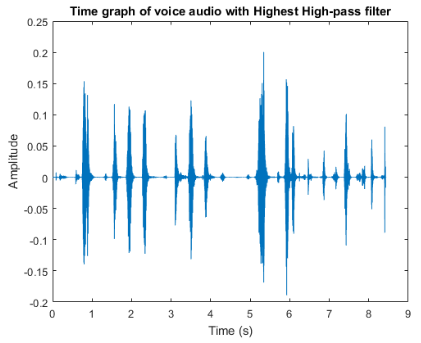


Figure 20: Time Domain graph of voice recording with most extreme high-pass filter.

In figure 20 the time graph for the voice audio after the filter is applied we see that the peaks of the signal where the words have been spoken still have their height, but are thinner. They still contain the information for each word but not all of it. This is why you can just understand the words, but the quality of the recording sounds lower, due to losing some background detail from the lower frequency values that were removed.

The next high-pass filter that was applied attempted to include some of the important information that was lost in the previous high-pass filter was included again. The Fstop was set at 2000, with the Fpass set at 2200.

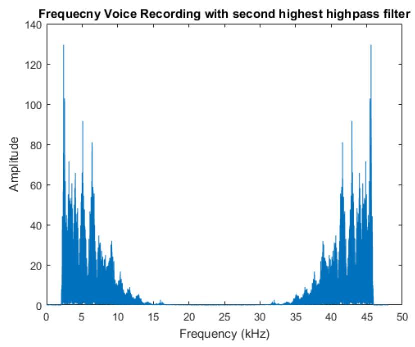


Figure 21: Frequency Domain graph of voice recording with other high-pass filter.

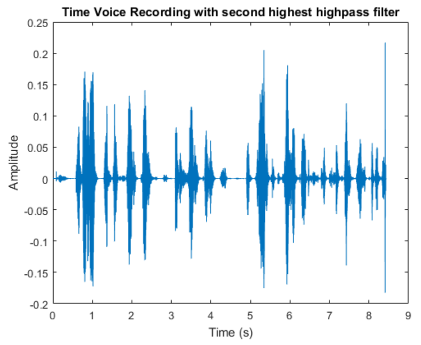


Figure 22: Time Domain graph of voice recording with other high-pass filter.

In figure 22 we see more width to the Time bands and from listening to the newly filtered audio we understand the words being said more clearly and the quality of the voice recording is better. This shows when humans listen we need a range of frequencies for the audio to be pleasing to the recipient.

We've seen both a low-pass and a high-pass filter, where we can remove from the extremes of the frequency bands. In order to remove frequency information, that is not from the edge, we can use a band-pass filter. This filter allows frequencies within a certain range to pass through. It can be seen as a combination of a low-pass and high-pass filter. Using the band-pass filter, we can show that the range from Fstop to Fpass cannot be too small or it can ruin the quality of the audio.

The first band-pass filter made the signal stop and start again so quickly that it would struggle to show the content of the file. For a band-pass filter there needed to be two sets of the Fstop and Fpass. The higher end one would not change and would have the values of 1600 and 2000 respectively. The lower band in the first set up would have an Fstop of 400 and Fpass of 600.

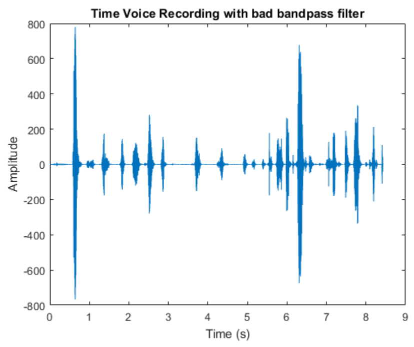


Figure 23: Time Domain graph of voice recording with first band-pass filter.

In figure 23 we know we can see on the Time graph that there is a spike in the amplitude that is higher than previous graphs. The audio that can be heard is also very unpleasant, piercing the ear with high pitch screeches. This is due the filter, trying to quickly allow frequencies back into the system without enough time to do so. This does not mean band-pass filters do not work, by simply changing the lower band's Fpass to 800, giving the system more time to reduce, you can here all the information in the voice recording successfully. The quality is better than the high-pass filter by not losing information and is clear as it does not exclude the higher tones.

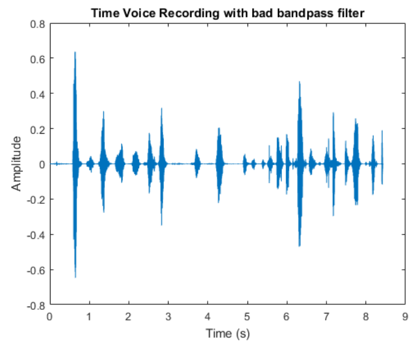


Figure 24: Time Domain graph of voice recording with improved band-pass filter..

Figure 24 shows the time graph for the correctly scaled band-pass filter does not have the spikes, that figure 23 had, meaning that it is easier to listen too.

Band-pass filters can adapt to the sound around them, using automatic gain control, in different products such as hearing aids, allowing the user to here the most data they can despite now being able to hear all frequencies.[7]

3.5 Question 5 - Identifying the Noise

For this question of the practical we needed to identify the loud noise that was interfering the the rest of the information in the sound piece. As the information surrounding the pitch of a piece of noise is not visible in the time domain. A FFT was used to place the audio clip into the frequency domain.

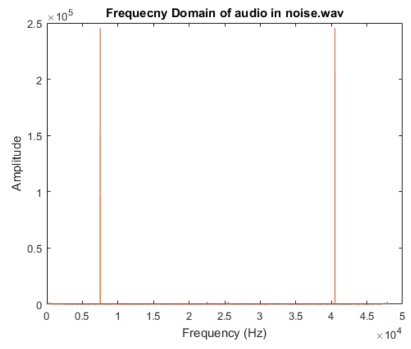


Figure 25: Frequency Domain graph from audio_in_noise.m

The frequency graph shows a single spike around one frequency. However on the audio clip there is clearly someone speaking alongside the noise and there is no oscillation in the frequency graph showing this. The high pitched noise is by far the dominate frequency, so is the only one that appears on the graph. The next question in the practical involves filtering out the noise, therefore first the noise needs to be identified. A few methods were used to make sure the frequency was identified.

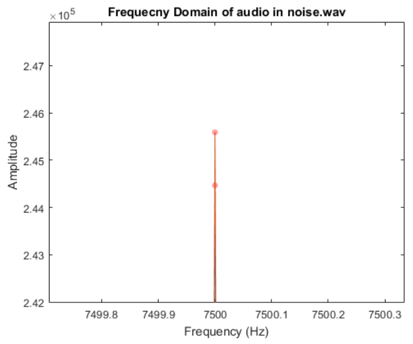


Figure 26: Frequency Domain graph from audio_in_noise.m showing value of spike.

The figure command was used in MATLAB in order to get the graph to show on the screen in a pop out window. The 'Brush/Select Data' tool was used to identify the points on the graph. In figure 26 you can see that two points were highlighted with this tool. The next tool that was used to identify the frequency was the 'Data Cursor' tool. This gives the exact value of a point.

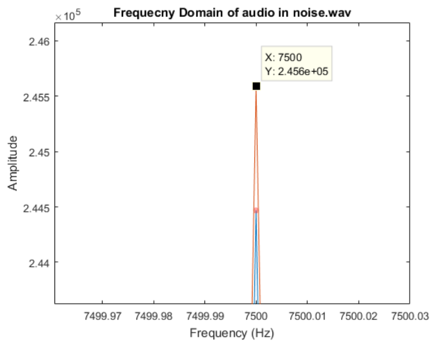


Figure 27: Frequency Domain graph from audio_in_noise.m showing value with data tool.

Figure 27 shows the look of the graph after the Data Cursor tool had been used. This gave the value of 7500Hz, which seems appropriate, when looking at the axis and the spike.

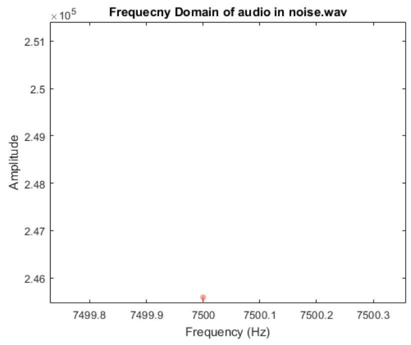


Figure 28: Frequency Domain graph from audio_in_noise.m showing value with data tool and zoom

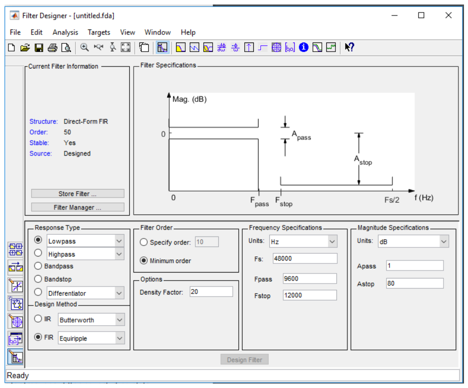
Using the interactive tool that allows you to move the graph around the pop out window. By dragging the graph you can put the data point onto the axis to see where the value lines up too and again it lines up to the value of 7500. Another test was done by using hearing to see if the recording was the same as an online simulator of the same noise. This gave a constant noise the same as the noise in the audio file.[3]

3.6 Question 6 - Filtering out the Noise

In order to remove the filter, some research was done. There are several different tools that come with MATLAB that can be used to create filters that can be added to remove unwanted frequencies.

The first tool that was investigated was filterBuilder, there tutorials that could be followed to create a filter using this tool. A warning appeared in MATLAB when this command was entered into the command window, saying that it would be depreciated shortly. After further research was done another tool called filterDesigner was found. An online tutorial was watched explaining how to use the tool and the different benefits the tool brings to signal processing.[2].

To active the tool, 'filterDesigner' needs to be entered into the command window. Once this has been entered, figure 29 appears.

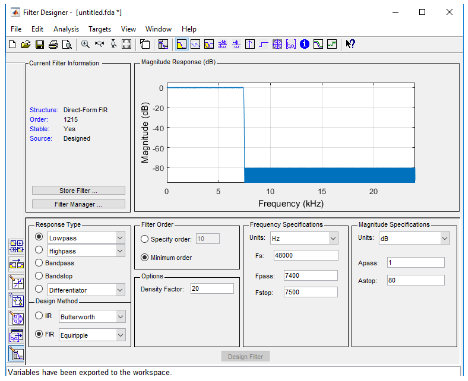


Current Filter Information
Structure: Direct-Form FIR
Order: 50
Stable: Yes
Source: Designed

Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator
Design Method: IIR Butterworth [✓ FIR Equiripple]
Filter Order: Specify order: 10 [✓ Minimum order]
Options: Density Factor: 20
Frequency Specifications: Units: Hz; Fs: 48000; Fpass: 9600; Fstop: 12000
Magnitude Specifications: Units: dB; Apass: 1; Astop: 80

Ready

Figure 29: FilterDesigner main menu.



Current Filter Information
Structure: Direct-Form FIR
Order: 1215
Stable: Yes
Source: Designed

Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator
Design Method: IIR Butterworth [✓ FIR Equiripple]
Filter Order: Specify order: 10 [✓ Minimum order]
Options: Density Factor: 20
Frequency Specifications: Units: Hz; Fs: 48000; Fpass: 7400; Fstop: 7500
Magnitude Specifications: Units: dB; Apass: 1; Astop: 80

Variables have been exported to the workspace.

Figure 30: FilterDesigner menu and settings used for Q6 filter.

Figure 30 shows the settings that were used in creating the filter for the noise. In the Design Method box in the bottom left of the screenshot there are two types of design filters. They are Infinite Impulse Response (IIR) and Finite Impulse Response. IIR's equation uses some of the output in the equation, therefore it is a recursive method, while FIR does not contain recursion. IIR is computationally faster so is ideal on small mobile signal processing devices, but it does not have constant phase and is not always stable. However FIR is slower, but has a constant phase and is always stable. Due to the filter being run on a computer where time is no issue, a FIR filter was chosen. The 'equiripple' filter design minimizes the maximum ripple in the bands of the filter.

A low-pass filter was chosen as it attenuates or removes frequencies above a specified frequency, which is ideal for removing high pitched frequencies from audio files. This was ideal for the question as it was a high pitched noise disturbing the speech in the audio recording.

After the filter was created, the Design Filter button was clicked, then the filter was exported as an object. This was done as it would be the simplest way of implementing it. The built in filter function was used in MATLAB that took both the data and the newly created object.

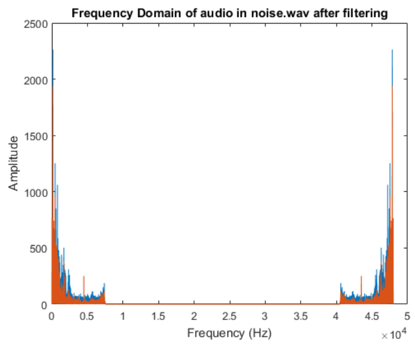


Figure 31: Frequency Domain graph of audio_in_noise.m after filter. Frequency

In figure 31 we now see a frequency graph similar to the music and voice audio files. After the dominated frequency had been removed the lower and less represented frequencies are visible.

The message that is left after the noise is removed from the audio clip is; "Hey CS4302, what a fun practical this is! I hope you are having fun, I am". However the noise is not completely gone. Faint parts of the previous noise can be heard when the speaker ends a word. This proved hard to remove, and I believe that is because as it was recorded as part of the clip it resonates in the frequency of the speaker's voice clip. Even though it is a higher frequency and should be removed by the low pass filter, remnants of it are not taken out, as the filter needs time to scale up and down around the speaker's voice.

4 Extension

There are four types of noise that can be found in audio recordings.[9]

- Hiss
- Hum
- Rumble
- Crackle

4.1 Hiss

Hiss often occurs when electrons do not follow their intended path in an electronic device. These electrons change the output voltage of the signal, creating a noise. A change in heat can cause the electrons to change direction, devices such as an air condition unit or radiator for example. Better recording equipment do not suffer as much as lower equality equipment. If the signal is recorded onto a magnetic tape then the issue can be due to larger magnetic particles, allowing the audio noise to appear.

4.2 Hum

Can be composed of a single frequency or a low frequency sound, similar with oscillation. It is caused due to audio equipment containing or using faulty circuits or not being correctly grounded. Also a hum can occur if a power cable is in close proximity to the audio equipment's cables used for data.

4.3 Rumble

A rumble is a very low frequency noise occurring below 40Hz. A classic example of the rumble noise was the turntable turning on a record player. This can cause issues when playing it, but in good turntables they avoid this by using slide bearings over regular bearings.

4.4 Crackle

Crackles are discontinuous noises that are noting to do with the music. They are irregular and can happen at any point. They are further defined into two categories; fine and coarse. Fine crackles are usually higher pitched and less intense noises, while coarse and low pitch and last for long periods of time.

4.5 Solutions

There are many ways to fix the above issues. The main way to fix them is to sort them before they happen in the first place. Better quality equipment and care when recording can create good audio files. However this is not always possible and we can fix the issues after the recording has occurred.

To remove hiss and hum and rumble, there are various tools. Adaptive Noise Reduction is done live and filters out these noises as they are played. This is not as effective at removing hisses or hums. A tool that could be used is Audition CC by Adobe[1].

They can also be removed using MATLAB. Using FFT we can identify the noises. Hums and Rumbles can be removed using a high-pass filter which we have already seen be used for getting rid of low frequency noises. Low-pass filters can do the same with high pitched hisses that can occur in the audio recordings.

Crackles which are loud will create spikes in the time graph. One way to filter all of these out of a file or song, would be to take a mean of all the data of the signal, then use this to set a threshold that would allow 99 percent of the data to remain in the signal, removing the random peaks that could occur. A small band-pass filter could also be used to remove small spikes, this is called a notch filter.

5 Conclusion

Overall this practical was challenging and great amount of time was spent getting used to the MATLAB environment. I feel I have achieved and answered all the questions correctly, while learning about both MATLAB and the area of processing signals in the domains of time and frequency. I also completed an extension looking into the different types of noises that can appear in an audio file and different techniques that can be used to remove them.

6 How to Run Different Questions

Task 1 is found inside 'task1.m' and has comments describing the file. When run it shows the time and frequency graphs for a middle C note.

For task 2,3,4 all the code can be found inside 'task2Voice.m' and 'task2Music.m' The graphs that are produced for the two audio files are found under the comments 'Time graph' and 'Frequency graph'. The labels and titles would be changed to fit the filter that has been implemented. All the filters for both the files are in comments at the top of the files. To run one of the filters uncomment that line only. There is also a line that can be uncommented to run it without filters. All the objects for the filters are contained in the file 'p1.mat' which is imported at the start of each file.

For task 5 and 6 all the code is found inside 'task5.' Similar to before 'p1.m' is loaded in at the start as it contains the filter objects. There are commented lines that indicated how to run the code with and without the filter.

References

[1] *Adobe Noise Reduction.* `https://helpx.adobe.com/uk/audition/using/noise-reduction-restoration-effects.html`. Accessed on 5/11/18.

[2] *filterDesigner tutorial.* `https://www.youtube.com/watch?v=VFt3UVw7VrE`. Accessed on 1/11/18.

[3] *Frequency Generator.* `http://www.szynalski.com/tone-generator/`. Accessed on 1/11/18.

[4] *Frequency Mirroring.* `https://dsp.stackexchange.com/questions/4825/why-is-the-fft-mirrored/4827#4827`. Accessed on 1/11/18.

[5] *Music Frequency Compression.* `https://www.soundonsound.com/techniques/what-data-compression-does-your-music`. Accessed on 1/11/18.

[6] *Note frequencies.* `http://www.sengpielaudio.com/calculator-notenames.htm`. Accessed on 1/11/18.

[7] *Odame, K. M., Anderson, D. V., Hasler, P. (2008). A bandpass filter with inherent gain adaptation for hearing applications. IEEE Transactions on Circuits and Systems I: Regular Papers, 55(3), 786-795.*

[8] *Re, D. E., O'Connor, J. J., Bennett, P. J., Feinberg, D. R. (2012). Preferences for very low and very high voice pitch in humans. PLoS One, 7(3), e32719.*

[9] *Types of Noise.* `http://www.audioshapers.com/blog/audio-noise.html`. Accessed on 5/11/18.