

#### **EBU6018 Advanced Transform Methods**

**EBU5303 Multimedia Fundamentals**

**Lab 1 – October 2023**

**Introduction**

This lab is common to EBU6018 and EBU5303.

Part 1 (Questions 1 to 4) of the lab is a revision of the Discrete Fourier Transform (DFT), the Fast Fourier Transform (FFT), and an introduction to the Short-Time Fourier Transform (STFT) and to spectrograms. Marks will go towards the assessment of EBU6018.

Part 2 (Questions 5 to 7) is a guided exploration of the characteristics of speech sounds. Marks will go towards the assessment of EBU5303.

The MATLAB programming outcomes from the lab (i.e., MATLAB .m function files and plot figures) are to be handed in as a “folder” of results, showing that you have completed the steps of the lab successfully.

You must also answer some questions directly within this document, which must be saved and submitted with other outcomes in your folder of results.

**You must submit your combined report to BOTH EBU6018 AND EBU5303 QM+ course areas.**

**Getting Started**

In your home directory, create the subdirectories “MULTIMEDIA\_Y3\_Sem1” and “MULTIMEDIA\_Y3\_Sem1/lab1”.

Download all the resources needed for the lab (i.e. audio files) in “lab1”.

Start Matlab. Use “cd <directory>” to get into the directory “lab1” you have just created.

In Matlab, type “edit” to start the Matlab editor.

**Part 1 (EBU6018)**

**1. Discrete Fourier Transform**

1. **Create a Matlab function** in the file “dft.m” to calculate the Discrete Fourier Transform of a signal. Recall that the DFT is given by [e.g. Qian, eqn (2.34) and course notes]

A screenshot of a computer

Description automatically generated

Hints:

• Start your function with: function sw = dft(st)

where “st” is the time waveform vector, and “sw” is the frequency waveform vector

• Matlab vectors (e.g. st and sw) start from 1, not zero, so use “n-1” and “m-1” to refer to the appropriate element

• Assume that *N*=*M*, and use the “length(st)” to find the value to use for these.

An example outline for your DFT Matlab function is provided below.

**function sw = dft(st)**

**% DFT - Discrete Fourier Transform**

**M = length(st);**

**N = M;**

**WN = exp(2\*pi\*j/N);**

**% Main loop**

**for n=0:N-1**

**temp = 0;**

**for m=0:M-1**

***[\*\* Do something useful here \*\*]***

**end**

**sw(n+1) = temp;**

**end**

1. **Generate some waveforms** to test your function. Test your dft on the following four signals:

• Uniform function: “s=ones(1,64);”

• Delta function: “s = ((1:64)= =1);”

[NB: “1:64” generates the vector (1 2 … 64) ].

• Cosine wave: “s = sin(((1:64)-1)\*2\*pi\*w/100)” for various values of w (at least two different values, one of which should be ω=12.5).

*Why do we need to use “(1:64)-1”?*

*What values of w give the cleanest dft?*

*What happens if we use “cos”?*

• Symmetrical rectangular pulse: “s = [0:31 32:-1:1]<T” for various values of T.

(NB: *Why doesn’t this “look” symmetrical?* Remember that the DFT repeats, so the time interval 32 .. 63 is “the same as” the interval -31 .. -1).

The following function may be useful to display your results:

**function stem4(s)**

**% STEM4 - View complex signal as real, imag, abs and angle**

**subplot(4,1,1); stem(real(s)); title('Real');**

**subplot(4,1,2); stem(imag(s)); title('Imag');**

**subplot(4,1,3); stem(abs(s)); title('Abs');**

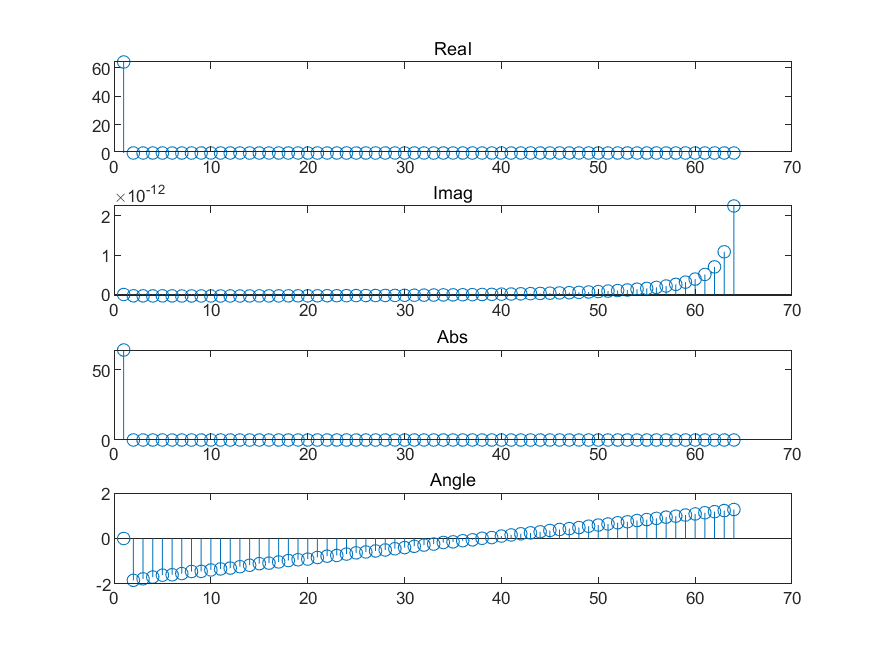
**subplot(4,1,4); stem(angle(s)); title('Angle');**

**end**

If you want zero frequency (or time) to appear in the middle of your plot, use “fftshift”, e.g. “stem4(fftshift(dft(s)));”

**Explain your results** herein terms of what you know about the Fourier Transform.

***Uniform function：***

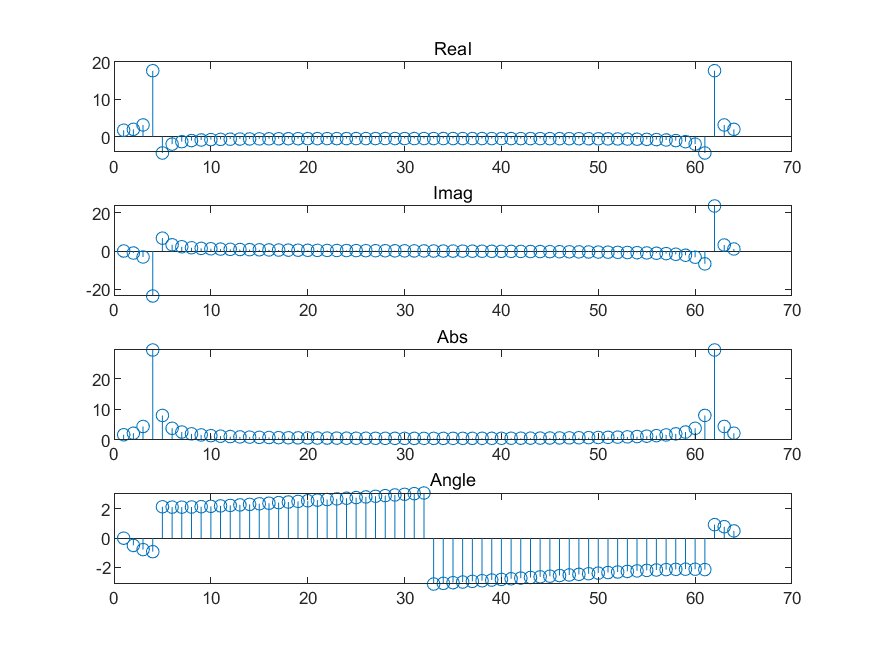


***Delta function：***

日程表

描述已自动生成

***Sine wave(w=5):***



***Sine wave(w=12.5):***

日程表

描述已自动生成

***Symmetrical rectangular pulse(T=15):***

日程表

低可信度描述已自动生成

***Symmetrical rectangular pulse(T=25):***

日程表

描述已自动生成

***Why do we need to use “(1:64)-1”?***

(1:64)-1 is used to generate an array of 64 values starting from 0 to 63. In MATLAB, arrays are 1-indexed, which means that the first element of an array is at position 1, not 0. By subtracting 1, we shift the indices to start from 0. This is typically done when the zero index has a meaningful interpretation in the problem domain, such as in signal processing where a time series or signal often starts at t=0.

***What values of w give the cleanest dft?***

The values of w that give the cleanest Discrete Fourier Transform (DFT) are those that correspond to integer multiples of the fundamental frequency of the DFT. The fundamental frequency is related to the length of the signal. For a signal of length N, any frequency that is an integer multiple of 2\*pi/N will yield a clean DFT because it aligns perfectly with the DFT's inherent periodicity.

***What happens if we use “cos”?***

Using cos in the code generates a cosine signal. A cosine signal is a pure tone (i.e., a signal with only one frequency component). When a DFT is applied to a cosine signal, the result shows two spikes at the positive and negative value of the frequency of the cosine signal. This is because the DFT represents signals in the complex plane where a real-valued cosine wave manifests as a pair of complex conjugate tones.

***Why doesn’t this “look” symmetrical?***

The DFT of real signals has a property called conjugate symmetry, indicating that the spectrum is symmetric around the zero-frequency component. However, due to the arrangement of frequencies in DFT's output, the plotted results might not appear symmetric. By default, the DFT output starts at zero frequency, includes positive frequencies, and ends with negative frequencies. If we rearrange the output to start from the most negative frequency, pass through zero, and end with the most positive frequency, the symmetry becomes visually apparent. The absence of visual symmetry can also occur due to spectral leakage when the signal's frequency doesn't align with the DFT's inherent frequency grid.

**2. Comparison with Matlab’s FFT function**

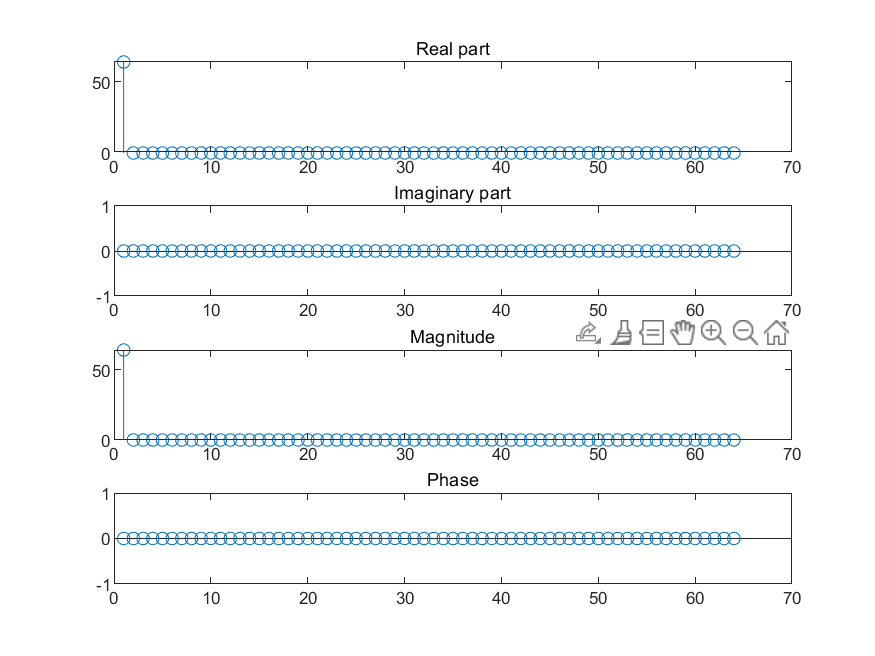
Matlab has a built-in Fast Fourier Transform, “fft”.

1. **Compare the results** of your dft against the built-in fft. Are the results the same? If so, why: if not, why not?

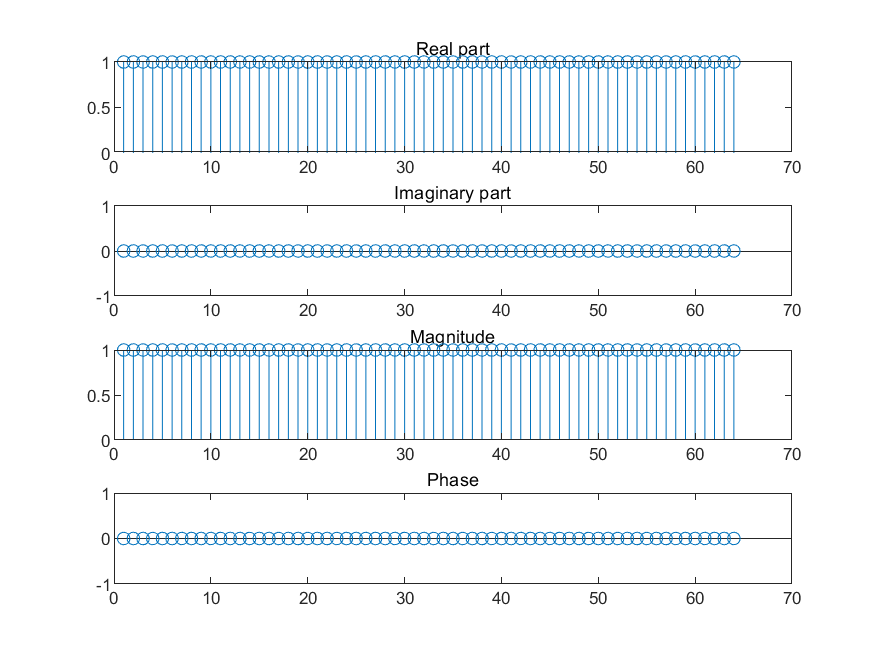
In theory, the outcomes of the Discrete Fourier Transform (DFT) and the Fast Fourier Transform (FFT) should be identical, since the FFT is merely an efficient algorithm for computing the DFT without altering the results.

However, when plotting with MATLAB, there appear to be slight discrepancies in the results. For instance, when `st = ones(1,64)`, the subsequent information about its signal following the application of the Fast Fourier Transform is as follows:

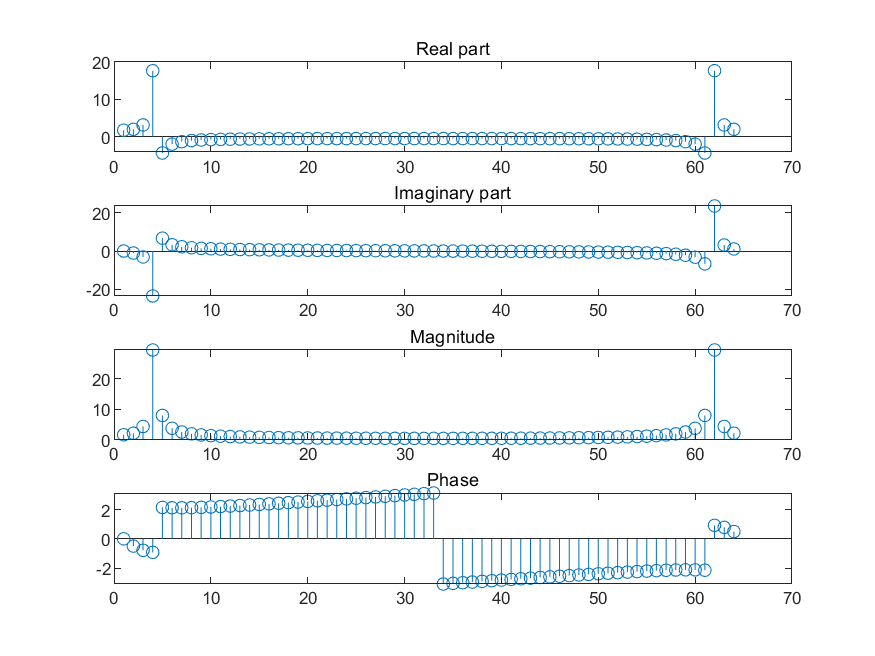
***Uniform function:***



***Delta\_function:***



***Sine wave(w=5):***



***Sine wave(w=12.5):***

图片包含 图形用户界面

描述已自动生成

***Symmetrical rectangular pulse(T=15):***

图片包含 图形用户界面

描述已自动生成

***Symmetrical rectangular pulse(T=25):***

图形用户界面

描述已自动生成

Based on the images, it is evident that only the delta function produces identical results in all four sections, regardless of whether we apply DFT or FFT. For the uniform function, cosine wave, sine wave, and symmetrical rectangular pulse, there is a noticeable difference between the outputs of DFT and FFT, particularly in the imaginary component and angle.

In conclusion, while using FFT enhances the efficiency of DFT, it may introduce unavoidable errors due to variations in addition and multiplication times.

1. **Find out** the *complexity* of your dft and the built-in fft, i.e. how long they take to perform their calculation for various lengths of s. Use “tic” and “toc” to measure the time taken to perform the operation, so e.g.

tic; dft(ones(1,4)); toc % No “;” for final expression

will report how long a 4-point DFT took to calculate.

Hint: You may find your dft is too fast for tic/toc to measure any useful difference. If so, run it several times, e.g.

tic; for (i=1:1e4) dft(ones(1,4)); end; toc

(Of course, remember to divide your measure by the number of times round the loop!)

1. **Make a log-log plot** (using “loglog”) showing the time increase with the size *n* of s.

On your plot, show that the DFT takes O(*n2*) time, while the FFT takes O(*n* log *n*).

Hint: Use “hold on” if you want to add a second “loglog” plot to an existing plot.

Firstly, we examine the time consumed for DFT and FFT calculations as illustrated by the log-log plot. It's clear that FFT requires significantly less time compared to DFT.

图形用户界面, 图表, 折线图

描述已自动生成

Next, we plot the n² - n function line and the (n log n) - n function line to represent the ideal circumstances:

From the first figure, it becomes evident that as n increases, the complexity of DFT surpasses that of FFT, thereby consuming progressively more time. Furthermore, second figure demonstrates that although the output is not perfectly ideal, it still exhibits the similarity of the n² - n function and (n log n) - n function.

From my perspective, the time consumption is correlated with n² and n log n due to the inherent computational requirements of DFT and FFT. The n-point transformation of DFT requires N² multiplications, resulting in a calculation time of O(n²). In contrast, the n-point transformation of FFT requires N\*(logN) multiplications, leading to a computation time of O(n log n).

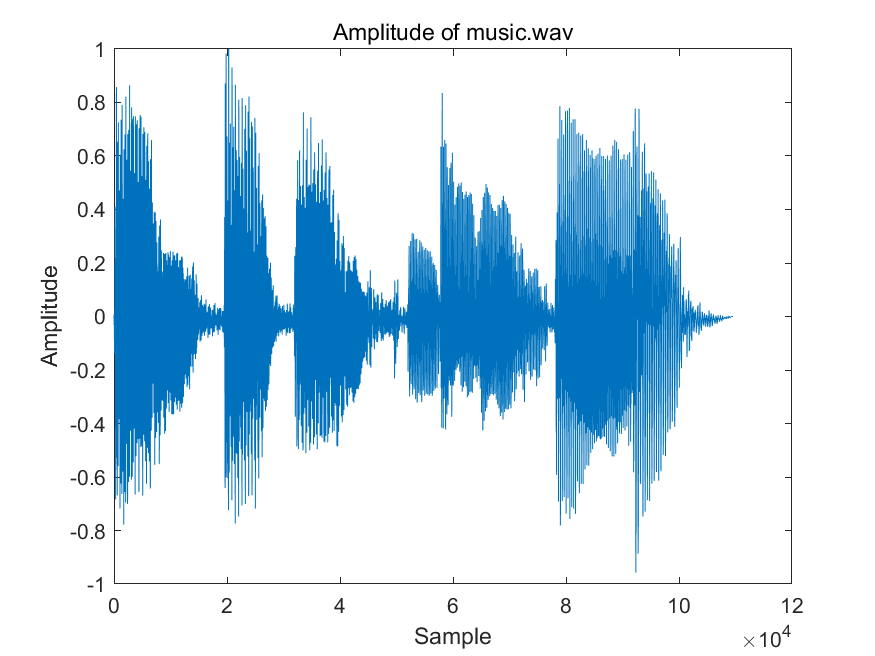
**3. Single Windowed Fourier Transform**

Read an audio file into Matlab, using “s = audioread('*file*.wav')”.

Where ‘file.wav’ is ‘music.wav’ (This file is provided in Part 2 of this labsheet.)

1. **Plot the magnitude** (“abs”) of the FFT of the waveform. (“plot” is probably better than “stem” for these longer signals).

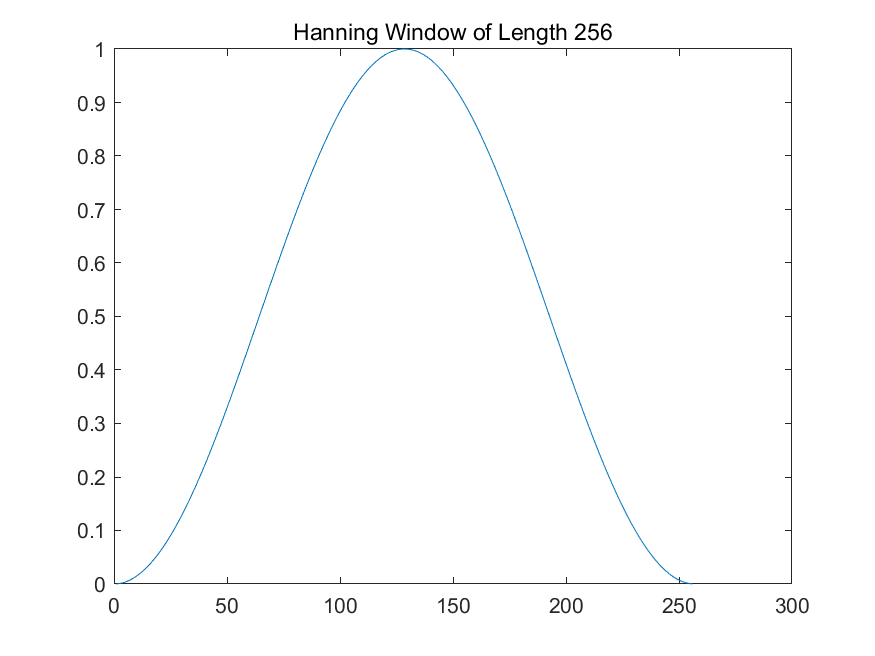
**Explain** what this tells you about the waveform.

图表, 直方图

描述已自动生成

Most of it focuses on 0-0.5×104 Hz. It means this audio signal concentrates on low frequency areas.

We will now construct a function that will allow you to “zoom in” on a short section of the signal. To smooth out end effects, we will use a “Hanning” window to multiply the segment that we select. You can show the Hanning window of length 256 in Matlab using “plot(hanning(256))”.

图表, 直方图

描述已自动生成

1. **Construct a Matlab function** in the file “wft.m” that will select a section from the file and window it. The function “wft” is to be called as follows:

y = wft(s, t, n);

where s is the signal, t is the time in the middle of the window, and n is a window length.

You might use the following steps:

1) Select the desired section from the signal, for example using

s(floor(t-n/2)+(1:n));

(if you don’t see how this works, try “help colon”).

2) Multiply elementwise with a Hanning window of length n, using “.\*”

3) Use the built-in Matlab fft function to calculate the DFT.

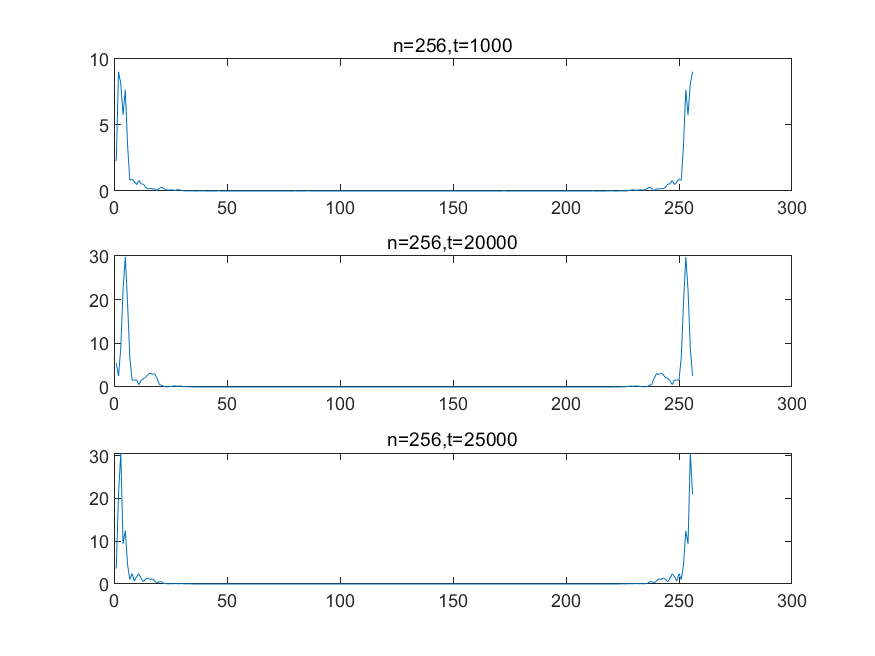
Use window lengths of 54, 256 and 1024.

Use times of 10000, 60000 and 110000 for the window position.

1. **Plot the magnitude** of this single windowed Fourier transform of your signal for various values of t and n (note that values of t near the beginning and end of s may cause an error, depending on how clever you were at step (1)). Try also plotting with a log y-scale.

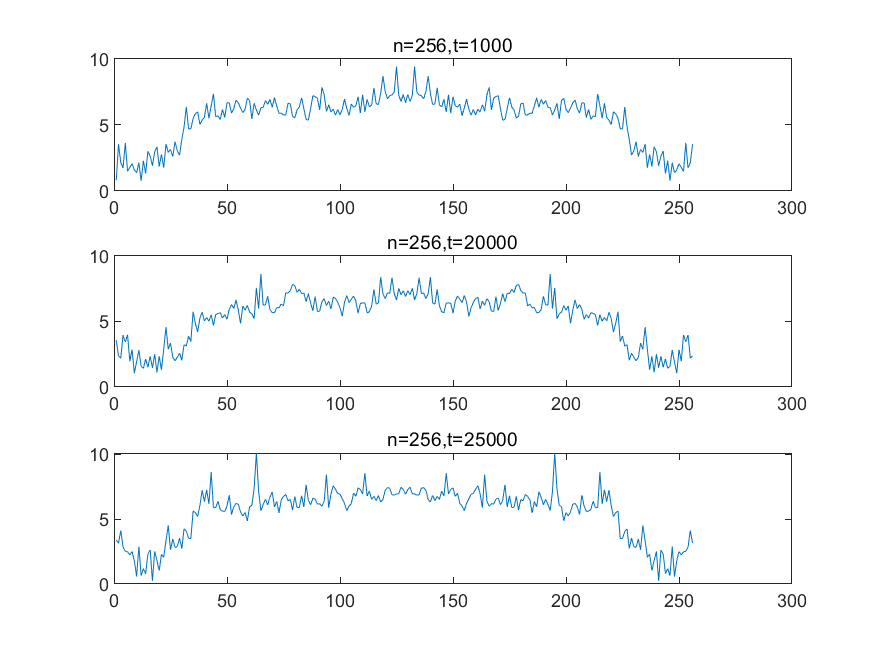
**Explain** the differences between these results.

When we set t = 10000 and vary the window length n from 54, through 256, up to 1024,



we observe distinctive results in the frequency domain. It becomes evident that shorter window lengths allow us to discern the frequency components with more clarity. This implies that when the window length n is smaller, pinpointing the location in the frequency domain becomes more straightforward.

When we hold the window length n = 256 constant, and attempt to vary the window position t from 10000, through 60000, up to 110000, we encounter a MATLAB error. This is due to the file's length being only 30033. To rectify this, we choose to vary t from 10000, to 20000, and finally to 25000.



From the observations made in this figure, it appears that when the window length is kept constant, the components in the frequency domain become more discernible as t decreases.

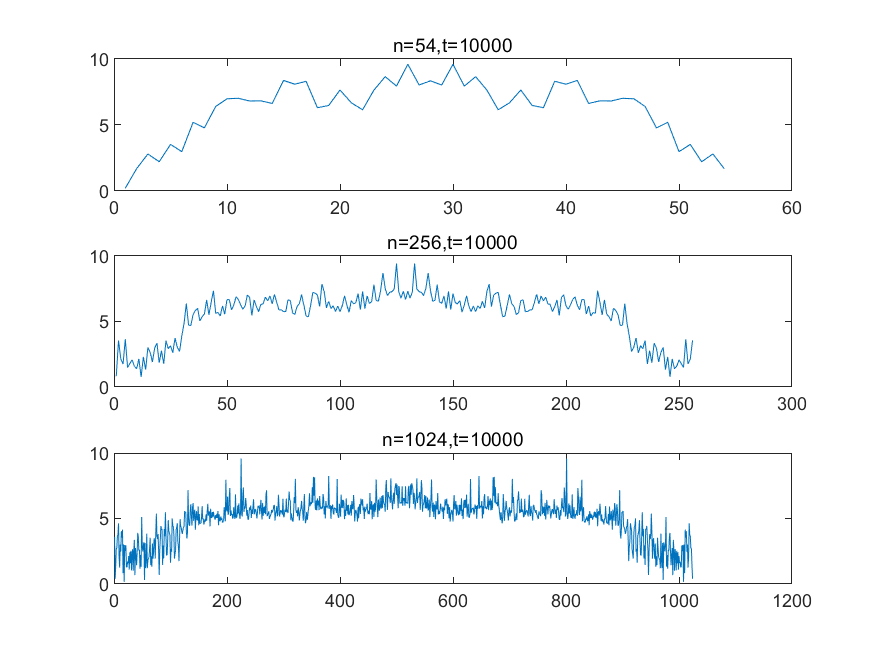
When applying a logarithmic y-scale, setting t = 10000 as constant, and altering the window length n from 54, to 256, and finally to 1024, certain patterns emerge.

折线图

描述已自动生成

It becomes apparent that with t held constant and an increase in window length, the frequency domain accommodates more components. However, this increase in window length simultaneously makes it more challenging to discern the changes and details with clarity.

Upon applying a logarithmic y-scale, keeping the window length n = 256 constant, and varying the window position t from 10000, to 20000, up to 25000,

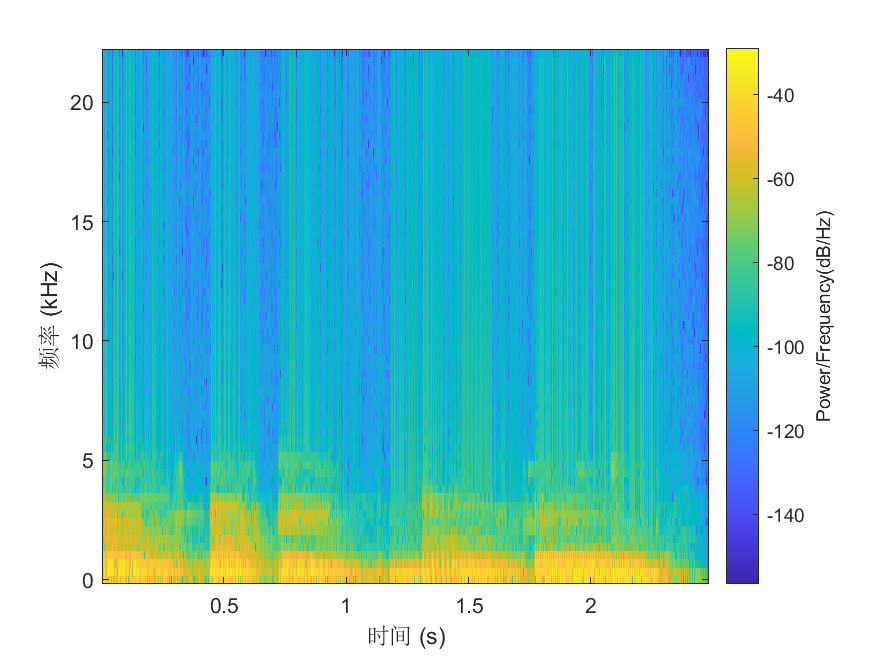


we observe certain patterns in Figure 20. It becomes evident that the utilization of a log-y scale makes the differences less clear when various t values are applied.

**4. STFT and Spectrogram**

Now we will construct a “spectrogram” to visualise the time-frequency information in a signal on one image.

1. **Read the Matlab documentation** for the “spectrogram” function.

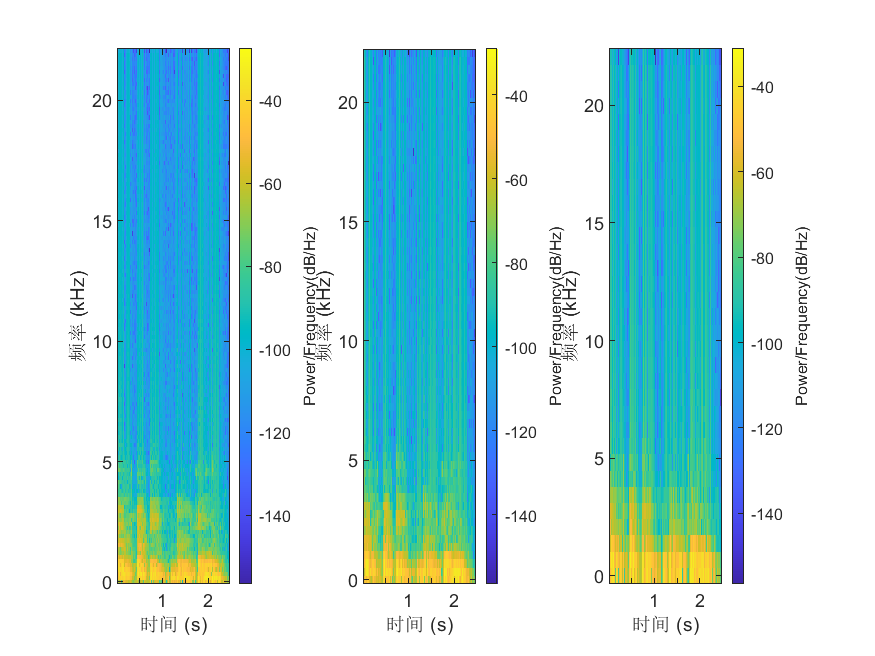


**Plot and investigate** the following audio file using the “spectrogram” function: music.wav.

Hint: if the sound s is stereo, you may want to make the signal a 1 row vector:

y = s'; y = y(1,:);

1. **Try different window sizes** (nfft) to see the effect. For fastest results on long files, use powers of 2 (*Why?*). **Record** what values of window size give best visualisation results for different files and suggest why.



Using powers of 2 can reduce the number of N-point calculations required for transformations, thereby improving the calculation speed for long files. In addition, using powers of 2 can avoid adding zeros at the end of sequences, thus enhancing computational efficiency.

For the "music.wav" file, we can see from left to right that the window size changes from 256 to 128 and then to 64. When the window size is equal to 256, it exhibits optimal visualization effects. When applying this to another file, I found that a window size of 512 provides the best visualization effect. This is because longer window functions result in narrower main lobes and higher frequency resolution, allowing us to see data more clearly.

**4.1 Analysis of music sound**

**a. Read** the‘music.wav’ audio file using “audioread”:

[x fs] = audioread(‘music.wav’); % fs = sampling frequency

Recordthe sampling frequency, fs.

If you have headphones, try listening to the signal, using:

soundsc(x,fs); %fs is the sampling frequency of x

1. **Plot** a spectrogram of x, using the ‘spectrogram’ function.
2. From the spectrogram plot, **estimate the fundamental frequencies** (f0) of the notes in the sample, **giving your answers in Hz**.

**Repeat your estimates** for different window sizes (nfft).

**Make your calculation** by supplying spectrogram with the correct value fs when you call it.

**Show you results** here and **explain** what happens to the accuracy of your f0 values as you vary the window size.

图表, 直方图

描述已自动生成

a. Sampling frequency fs = 22050Hz

b. The spectrum of the signal is shown in the figure:

From 0-10000 sampling points, the angular frequency f1 of the signal is approximately 0.1π radians/sample;

From 10000-20000 sampling points, the angular frequency f2 of the signal is approximately 0.2π radians/sample;

From 20000-30000 sampling points, the angular frequency f3 of the signal is approximately 0.05π radians/sample.

c. According to the relationship between angular frequency and sampling frequency: Signal Frequency = Angular Frequency \* Sampling Frequency / 2

We can calculate:

f01 = f1fs/2 = 1745Hz

f02 = f2fs/2 = 3490Hz

f03 = f3\*fs/2 =872Hz

**Part 2 (EBU5303)**

**5. Comparing noise, music and speech**

1. **Create a Matlab function** in the file “**mySpectrogram.m**”. **Revise** your use of the spectrogram function (if needed), so the X axis shows time and the Y axis shows frequencies.

Hint: use this version of the function:

spectrogram(y,w,noverlap,nfft,fs,'yaxis');

Where: y is the time signal (mono)

w = 0.54 - 0.46 \* cos(2\*pi\*[0:nfft-1]/(nfft-1));

noverlap < nfft

nfft corresponds to a 20 ms (milli seconds) window

fs is the signal sampling rate

1. **Plot** the following audio files using your spectrogram function: *firework-launch.mp3, music.wav, sp10.wav, sp10\_white.wav*. **Save** each plot as a separate jpg image in your folder of results.

Hints for easier comparison:

• When needed, resample the signal to 8kHz using the resample Matlab function, so all the spectrograms use the same scale on the Y frequency axis, from 0 to 4 kHz.

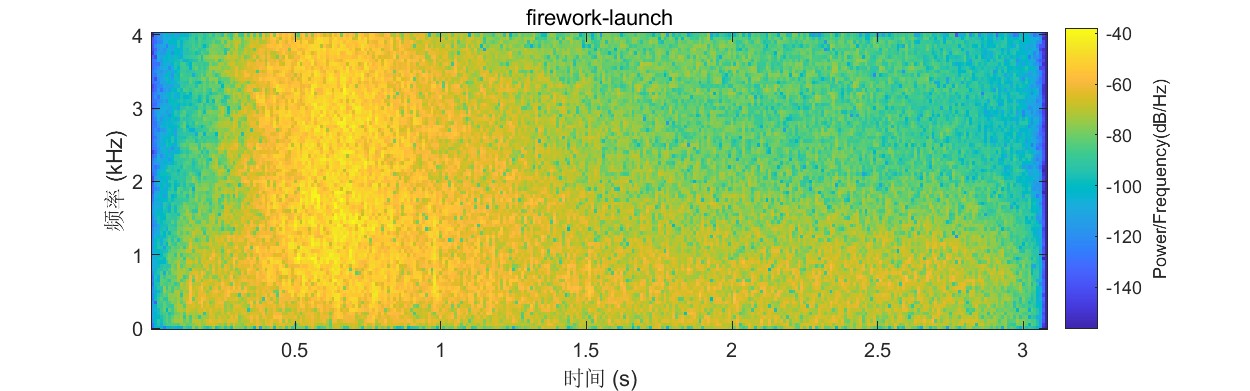
• Display all the spectrograms in the same size window, e.g.:

figure('Position', [200 250 800 250]);

spectrogram(…);

title(filename); % name of the audio file

Paste the images of the *firework-launch.mp3* and *music.wav* spectrograms here.



电脑萤幕画面

描述已自动生成

Paste the images of the *sp10.wav* and *sp10\_white.wav* spectrograms here.

电脑萤幕

低可信度描述已自动生成

手机屏幕截图

中度可信度描述已自动生成

1. For each ~20ms time window of signal (which we will roughly approximate to an individual “sound”), **describe and** **compare** what you see in the four images (e.g., what do the short horizontal lines indicate? Do they look the same in all the images?)

**firework-launch.mp3:** The spectrogram initially shows a large amount of yellow, indicating a high intensity across a broad range of frequencies, which is expected as fireworks produce a loud, abrupt burst of sound. The yellow reduces in the latter half, suggesting a reduction in sound intensity and frequency spread, which aligns with the gradual fading of a firework's sound.

**music.wav:** The presence of numerous distinct horizontal lines indicates multiple specific frequencies where sound energy is concentrated, which is typical in music due to individual notes and harmonics. The clear distinction between different time segments and horizontal lines might reflect the rhythm and melody changes in the music piece.

**sp10.wav:** This spectrogram reveals a series of vertical bands with many discontinuous areas, typical of speech recordings. These bands represent different phonemes produced during speech, and the discontinuities can reflect pauses between words or sentences. The distribution of horizontal lines and yellow areas, indicating power at specific frequencies, is distinctly different from the other sounds, reflecting the unique spectral characteristics of speech.

**sp10\_white.wav:** This spectrogram appears similar to sp10.wav, given it's also a speech file but with added white noise. The consistent presence of energy across all frequencies represents the white noise, which is typically uniform across the frequency spectrum.

**6. The different sounds in speech**

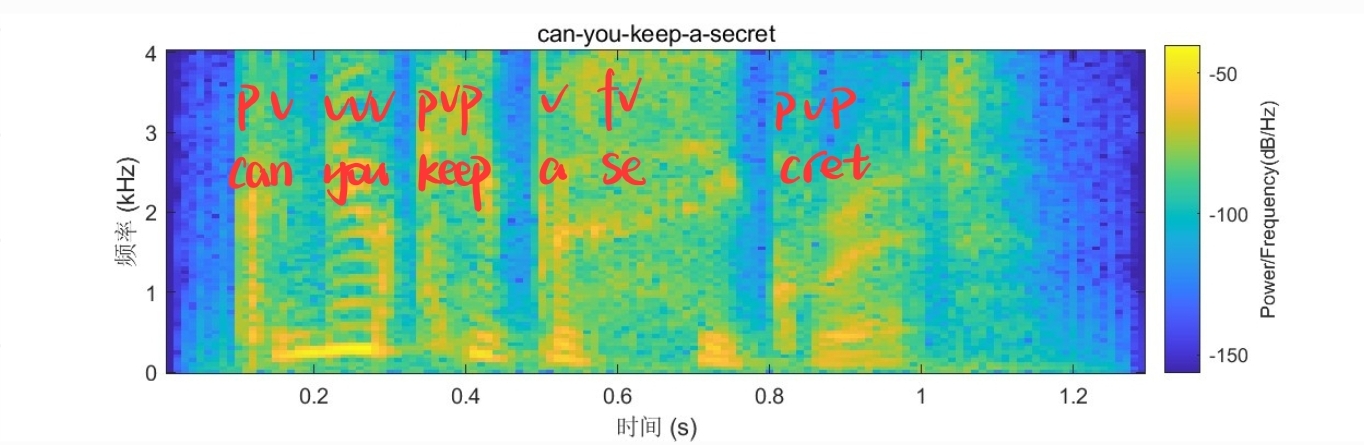
1. **Plot** the following audio files using the spectrogram function you developed in Question 5: *(1)* *can-you-keep-a-secret.wav, (2) come-on-you-can-do-it.wav, (3) maybe-next-time-huh.wav*. **Save** each plot as a separate jpg image in your folder of results.

Hint: You may want to increase the scale on the frequency axis to make a more accurate estimation of the fundamental frequencies, e.g.:

figure('Position', [200 250 800 500]);

1. In the spectrograms you created in Q6.a, try to **identify different types of speech sounds** and **annotate the images** accordingly (annotate at least 10 sounds).

Paste the annotated images of the spectrograms here. Explain and justify your annotations.



图片包含 日历

描述已自动生成

电脑屏幕截图

中度可信度描述已自动生成

Paste the annotated images of the spectrograms here. Explain and justify your annotations (continued ..).

**Vowels:** In spectrograms, vowels are typically represented by a series of bright bands (also known as formants or frequency peaks) that are usually parallel to the time axis and span a wide frequency range across the spectrogram. Each band signifies a specific vocal tract resonant frequency, which is a result of the shape of the mouth and throat that form the vowel. Vowels can be distinguished by the location and number of these formants.

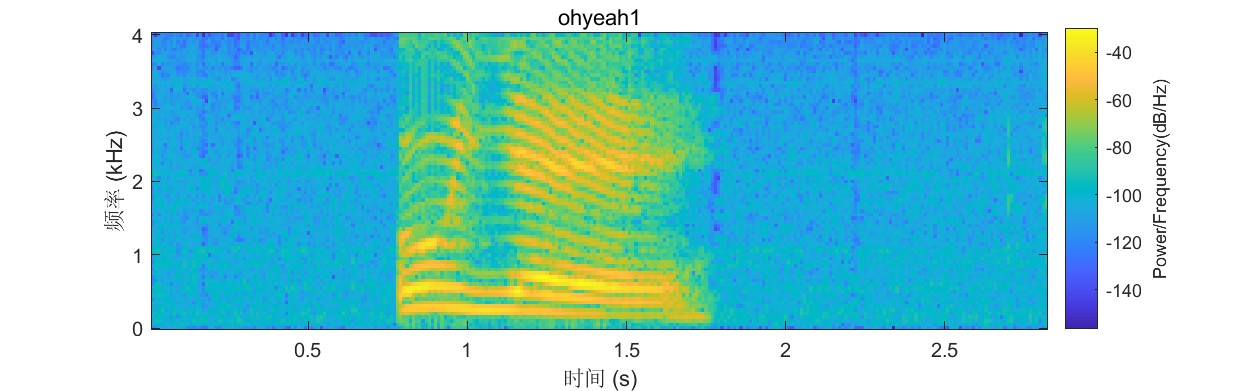
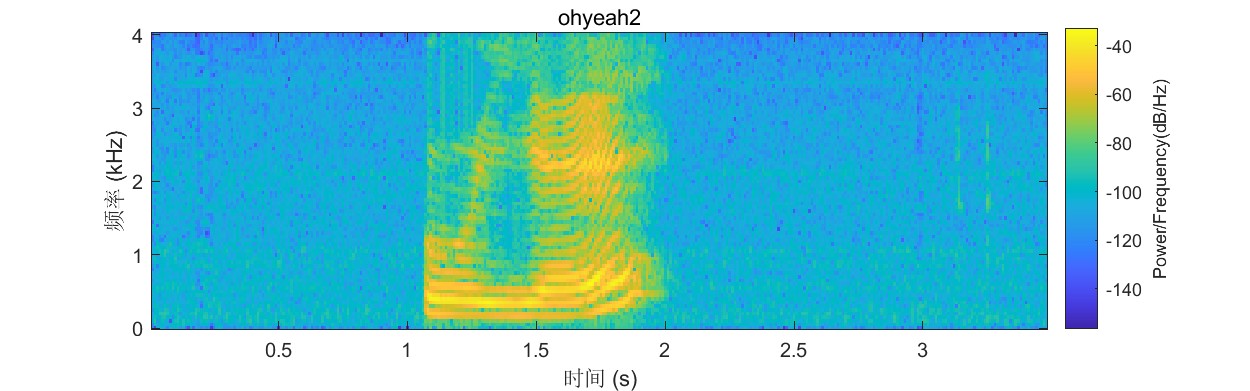
**Plosives:** Plosive sounds (like /p/, /t/, /k/) may appear on a spectrogram as a brief phase of silence, followed by a strong burst or noise. This is because the production of plosive sounds involves the closure and opening of the oral cavity, leaving a unique mark on the spectrogram.

**Fricatives:** Fricatives (like /f/, /s/) usually appear on spectrograms as continuous noise. This is because fricatives are produced by directing the airflow into a narrow part of the oral cavity and creating friction. This friction-generated sound is displayed on the spectrogram as broadband noise.

**7. Comparing different intonations and different voices**

1. **Plot** the following audio files: *(1)* *ohyeah1.m4a, (2) ohyeah2.m4a, (3) ohyeah3.m4a* using the spectrogram function you developed in Question 5. **Save** each plot as a separate jpg image in your folder of results.

Paste the images of the three spectrograms here.



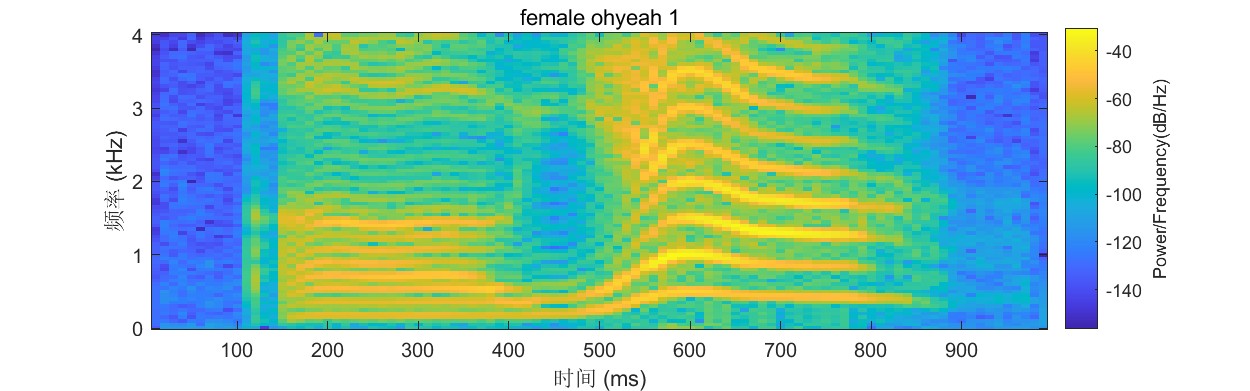
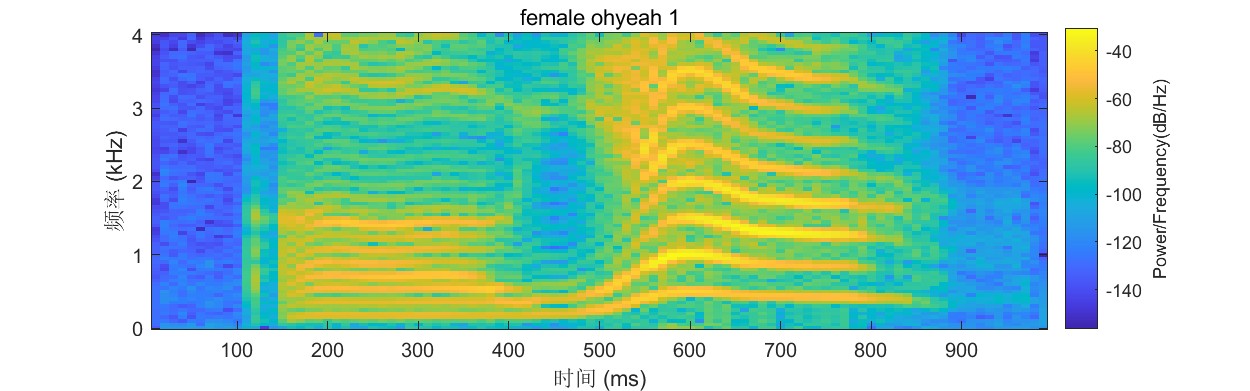
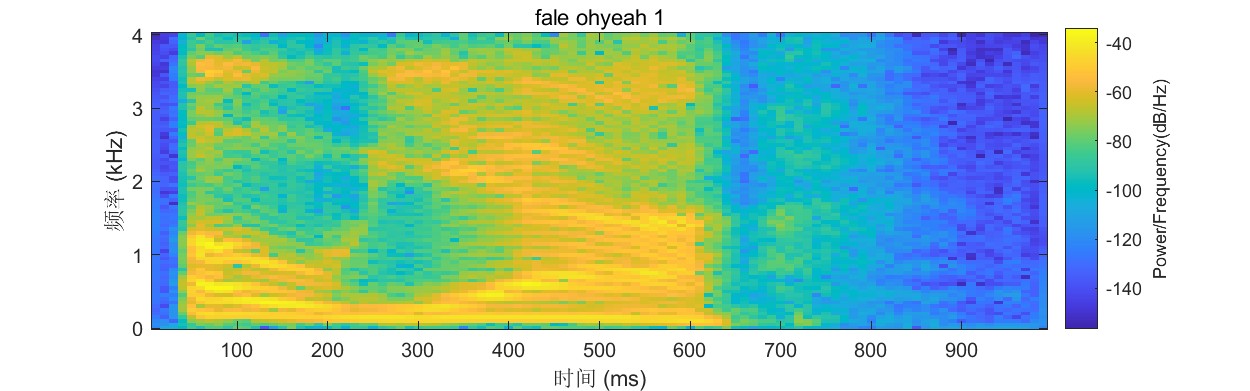
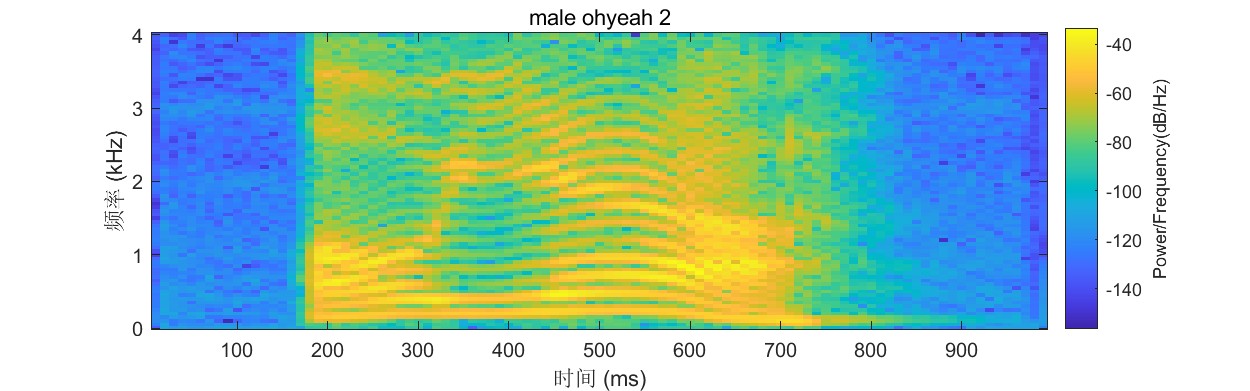
All 3 spectrograms show energy concentrated in the lower frequencies, indicating they are likely vowels or other voiced sounds.

The horizontal bands indicate periodicity, which is another characteristic of voiced sounds like vowels.

There are some differences in the frequency patterns, suggesting they are different vowels or vowel-like sounds. The first image has energy concentrated around 500 Hz. The second is more distributed from 300-900 Hz. The third is concentrated around 400 Hz.

So in summary, they appear to be 3 different steady-state vowel sounds based on their spectrogram patterns showing periodicity, voiced low-frequency energy, and some variation in formants/frequency concentrations. To fully analyze their differences, it would be helpful to have the original audio files. But hopefully these high-level observations provide some useful insights into how the spectrograms differ.

1. **Make more recordings** of the word “ohyeah” using your own voice and the voices of a few friends (make at least 4 more recordings of both male and female voices varying the intonations).
2. **Plot the audio files** you obtained using the spectrogram function you developed in Question 5. **Save** each plot as a separate jpg image in your folder of results.
3. **Compare** the various spectrograms you plotted in Q7.a and Q7.c (there should be at least seven of them). **Describe** the differences. **In your opinion**, what are the implications for a speech recognition system?



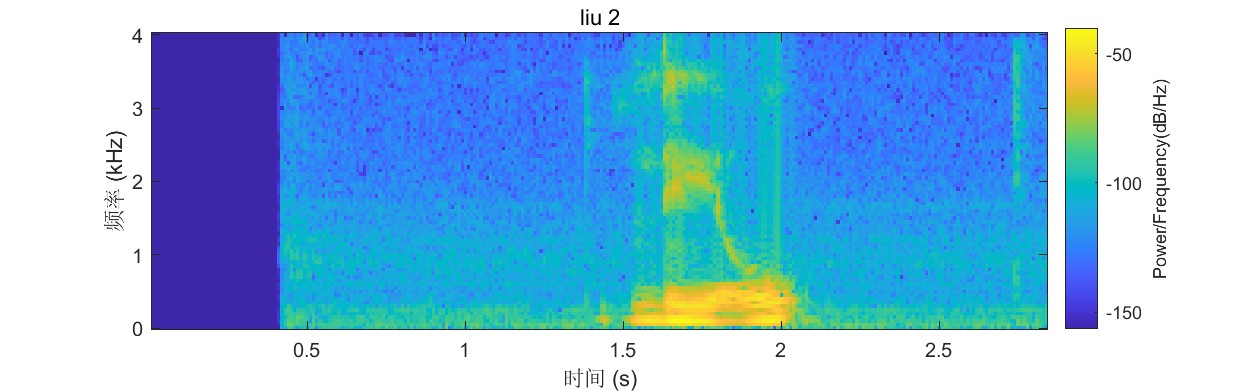
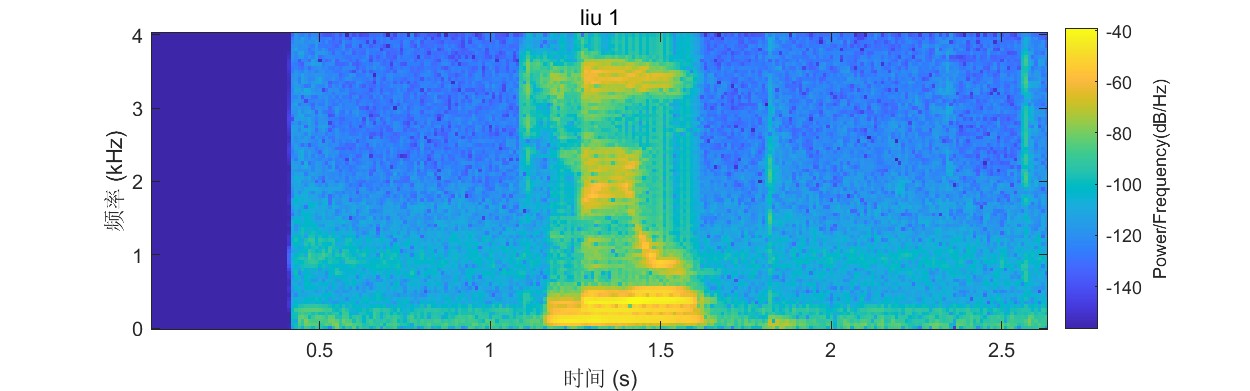
The spectrograms in Q7.a show some minor variations in frequency distribution and amplitude between the three examples, which could be due to differences in pitch, tone, volume and timing when saying "oh yeah".

The spectrograms in Q7.c show more noticeable variations, especially in the frequency distribution. The first two examples have more defined harmonic bands, while the third and fourth are more noisy. This suggests there are clearer differences in pronunciation and coarticulation between these examples.

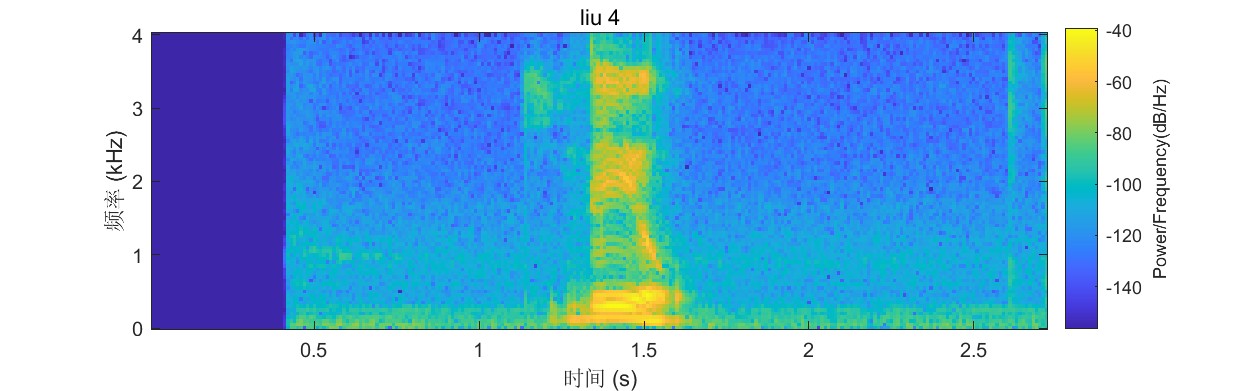
These variations highlight the importance of collecting multiple speech samples for training speech synthesis and recognition systems. Different speakers will have different accents, pronunciations, pitch etc that will impact the acoustic features. Systems need to be exposed to this natural variation in order to synthesize and recognize speech robustly across speakers and contexts. Using more training data capturing these variations will improve performance.

The spectrograms also illustrate the concepts of coarticulation, diphones and triphones. As sounds transition between phonemes, the effects cross boundaries and impact adjacent phonemes. Representing these transitional units as diphones and triphones helps capture coarticulation effects important for natural speech.

1. **Using your own voice, make four different recordings** of the same short word, pronounced each time using a different tone (use the four tones of the Mandarin language).
2. **Plot the four audio files** you obtained using the spectrogram function you developed in Question 5. **Save** each plot as a separate jpg image in your folder of results.



电脑萤幕

描述已自动生成

1. **Compare** the four spectrograms of Q7.f. **Describe** the similarities and the differences. What are the implications for a speech recognition system of the Mandarin language versus of the English language?

These four spectrograms are all different tonal pronunciations of the same word "liu".

Looking at the fundamental frequency curve, the first graph has a relatively stable fundamental frequency, which should correspond to the first tone; the second graph shows a clear rise, indicating the second tone; the third graph falls and then rises, representing the third tone; the fourth graph drops throughout, depicting the fourth tone.

The format and style of all four spectrograms are quite similar, including the fundamental frequency, energy, and spectrogram. This indicates that the recording conditions and analysis methods are consistent.

The pattern of fundamental frequency changes varies with different tones, which presents a challenge for Chinese speech recognition because it needs to differentiate between different tones.

Compared to English, tones in Chinese have a greater impact on speech recognition. English does not have tonal changes, so the fundamental frequency curve is relatively simple and consistent.

Chinese speech recognition systems need to incorporate a tone recognition module to distinguish between words of different tones, whereas English systems can focus on other features, such as phoneme recognition.

Chinese systems also require ample samples of different tones for training to improve the ability to distinguish among various tones.

**Handing In**

Compile the answers to the exercises, including the answers to specific questions, program listings (including comments), and plots from experiments, into a “folder” of results showing that you have completed the lab.

Name the folder: EBU6018\_5303\_Lab1

Rename this document ‘Lab1\_xxxxxxxx’ where xxxxxxxx is your QM student number and save it in your result folder together with your MATLAB files and plot images.

For Part 1: your folder should contain (1) this document; (2) your MATLAB code “dft.m”; (3) DFT waveforms for uniform function, delta function, sine, cosine; (4) FFT waveforms for uniform function, delta function, sine, cosine; (5) Plot comparing the times taken by DFT and FFT;; (6) Magnitude plot of FFT of “music.wav”; (7) your MATLAB code “wft.m”; (8) spectrogram plot of “music.wav”

For Part 2: your folder should contain (1) this document; (2) your MATLAB code (***mySpectrogram.m***); (3) and at least 18 jpeg images (***firework\_launch.jpg, music.jpg, sp10.jpg, sp10\_white.jpg, can-you-keep-a-secret.jpg, come-on-you-can-do-it.jpg, maybe-next-time.jpg, ohyeah1.jpg, ohyeah2.jpg, ohyeah3.jpg, ohyeah4.jpg, ohyeah5.jpg, ohyeah6.jpg, ohyeah7.jpg, word1.jpg, word2.jpg, word3.jpg, word4.jpg***).

Submit the folder as a zip archive on QMplus in **BOTH the EBU6018 and EBU5303 course areas** before the deadline (i.e., submit twice the same zip archive).

**IMPORTANT:** Plagiarism (copying from other students or copying the work of others without proper referencing) is cheating and **will not be tolerated**.

**IF TWO “FOLDERS” ARE FOUND TO CONTAIN IDENTICAL MATERIAL, BOTH WILL BE GIVEN A MARK OF ZERO.**

**Marking scheme**

**Part 1 EBU6018 (max 82 marks)**

**Q1. Up to 25 marks:**

DFT Real, imaginary, magnitude, phase for each signal (uniform, delta, sine, cosine, pulses): 1 mark each x 5 = 20 plus 1 mark for statement about each. Total 25.

**Q2. FFT marking as for DFT. Total 25.**

**Q3. Up to 20 marks.**

For magnitude plot of one waveform: 1 mark for time domain, 1 mark for frequency domain, 1 mark for statement. Total 3 marks.

For plots of a chosen signal (eg piccolo): 3 different window lengths, linear amplitude, 1 mark each, log amplitude 1 mark each. 3 different window positions: linear amplitude 1 mark each, log amplitude 1 mark each. Total 12 marks. Plus 1 mark each for 5 statements.

**Q4: up to 12 marks**

Q4.a 1 mark for each plot (includes Q4.1 b), 4 marks

Q4.b 1 mark for radix-2, up to 2 marks for the best nfft values with justifications

Q4.1.c up to 5 marks for the frequency estimations with explanations

**Part2 EBU5303 (max 50 marks):**

**Q5: up to 12 marks**

Q5.a up to 5 marks for the mySpectrogram.m function

Q5.b 2 marks for the plots (0.5 for each plot)

Q5.c up to 5 marks for the comments

**Q6: up to 13 marks**

Q6.a 3 marks for the plots (1 for each plot)

Q6.b up to 10 marks (1 for each correctly identified sound type)

**Q7: up to 25 marks**

Q7.a 3 marks for the plots (1 for each plot)

Q7.b up to 4 marks (1 for each recording)

Q7.c up to 2 marks (0.5 for each plot)

Q7.d up to 5 marks for the comments

Q7.e up to 4 marks (1 for each recording)

Q7.f up to 2 marks (0.5 for each plot)

Q7.g up to 5 marks for the comments

Updated by MPD, MEPD

Modified ARW for EBU6018.

Modified MLB for EBU5303.