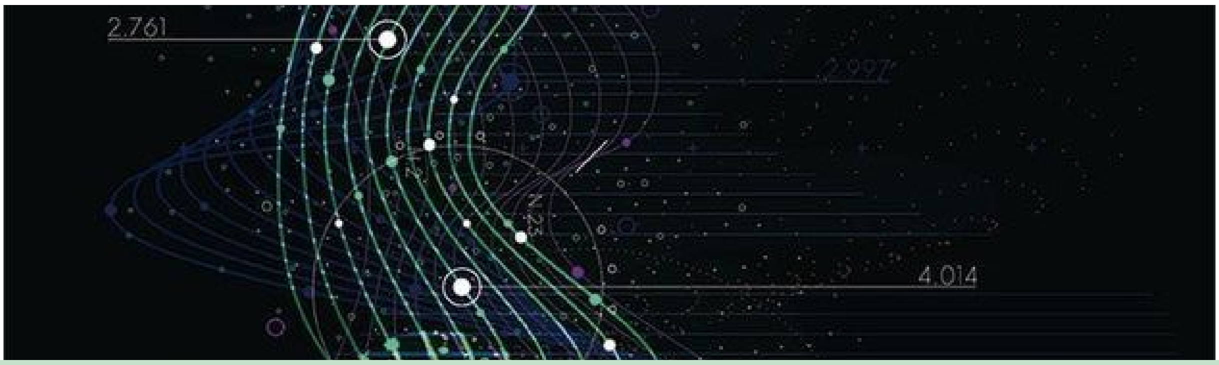
DIGITAL SIGNAL PROCESSING



MATLAB PROJECT

ECE 255

APRIL 2023

Submitted by

Nouran Hamdy Mohamed	20P4106
Mai Sherif Fekry Youssef	20P6381
Nahed Emad El-Dien Mostafa	21P0127

Submitted to

Dr. Michael Naiem Abdelmassih

Eng. Ahmed Mohamed Abd El Khalik

INTRODUCTION

The dual-tone multi-frequency (DTMF) signaling scheme is a method of transmitting information over a telephone line using a combination of two audible tones. Each tone corresponds to a row and a column of a 4x4 keypad, and each pair of tones represents a digit or a symbol. For example, the digit 1 is encoded by the tones 697 Hz and 1209 Hz.

The notion of time-frequency analysis and spectrograms is useful for studying the DTMF signals and detecting the transmitted information. Time-frequency analysis is a technique that decomposes a signal into its frequency components as a function of time using spectrograms, which are graphical representations of the time-frequency analysis, where the intensity of each pixel indicates the magnitude of a frequency component at a given time.

The report starts by generating a DTMF sequence for a phone number and adding AWGN to the signal. The signal is then saved as an audio file and loaded back into MATLAB. The report then presents spectrograms with different time-domain window sizes for the signal, and discusses the trade-off between time and frequency resolution. Finally, the Goertzel algorithm is used to decode the DTMF symbols in the signal, and the symbols detected are reported.

BACKGROUND

01 SPECTROGRAMS

Introduction:

A spectrogram is a visual representation of the frequency content of a signal as it changes over time. It is a powerful tool for analyzing signals that change in frequency and amplitude over time, such as speech, music, or other sounds.

The spectrogram is created by dividing the signal into short segments, typically using a time-domain window function such as a Hamming or Blackman window. These windows effectively "chop" the signal into short, overlapping sections, with the length of each section determined by the window size. The window size is typically chosen to balance time resolution and frequency resolution, with longer windows providing better frequency resolution and shorter windows providing better time resolution.

Once the signal is divided into segments, a Fourier transform is performed on each segment to obtain the frequency content. The resulting spectrum for each segment is then displayed as a color-coded image, with the frequency content represented along the vertical axis and time represented along the horizontal axis. Brighter colors indicate higher spectral energy at that particular frequency and time.

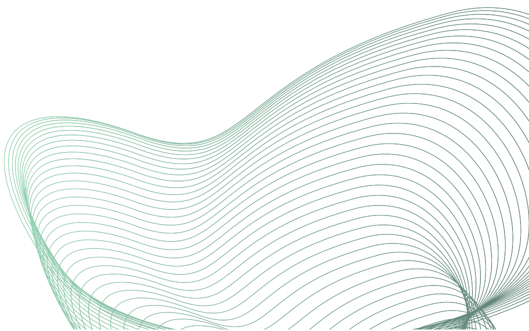
BACKGROUND

01 SPECTROGRAMS

Detailed Explanation:

Example Code:

```
% Read in an audio file
[y,fs] = audioread('example_audio.wav');

% Define the parameters
window_size = 256; % Time-domain window size
overlap = 0.5; % 50% Overlap
fft_size = 512; % FFT size

% Create a spectrogram using a Hamming window
window = hamming(window_size); % create a Hamming window
[S,F,T] = spectrogram(y,window,overlap*window_size,fft_size,fs,'yaxis');
% Compute spectrogram
imagesc(T,F,20*log10(abs(S))) % Plot the spectrogram as an image
set(gca,'YDir','normal') % Flip the y-axis to have low frequencies at the
bottom
xlabel('Time (s)') % Add x-axis label
ylabel('Frequency (Hz)') % Add y-axis label
colorbar % Add a colorbar to show the magnitude scale
title('Spectrogram of Example Audio Signal Using Hamming Window') %
Add a title to the plot
```



BACKGROUND

01 SPECTROGRAMS

Detailed Explanation of the Code:

- **y** and **fs** are the audio signal and its corresponding sampling frequency, respectively, read in from a WAV file using the **audioread** function.
- **window_size**, **overlap**, and **fft_size** are parameters that define the time-domain window size, the percentage of overlap between adjacent windows, and the FFT size used for the spectrogram computation, respectively.
- **window** is the Hamming window used as the time-domain window function for the spectrogram computation.
- **spectrogram** function computes the spectrogram of the audio signal **y** using the Hamming window **window**, with an overlap of **overlap*window_size** samples between adjacent windows, an FFT size of **fft_size**, and a sampling frequency of **fs**. It returns the spectrogram **S**, the frequency vector **F**, and the time vector **T**.
- **imagesc** function plots the spectrogram **S** as an image with the x-axis representing time, the y-axis representing frequency, and the color intensity representing the magnitude of the corresponding frequency component at that point in time.
- **set(gca,'YDir','normal')** flips the y-axis to have low frequencies at the bottom.
- **colorbar** adds a colorbar to the plot to show the magnitude scale.

BACKGROUND

02 “GOERTZEL” ALGORITHM

Introduction:

The Goertzel algorithm is a digital signal processing technique used to determine the strength of a particular frequency component in a signal. It is similar to the Discrete Fourier Transform (DFT) but is optimized for single-frequency detection rather than spectral analysis of a wide frequency range. The algorithm computes the magnitude of a specific frequency component by filtering the signal using a resonator filter with a frequency equal to the component of interest.

The Goertzel algorithm is commonly used in digital signal processing applications where real-time processing is required, such as audio processing, speech recognition, and telecommunications. It is particularly useful when the frequency of interest is known in advance, as it can quickly and accurately determine the presence and strength of that frequency in the input signal.

Compared to other DFT algorithms, such as the Fast Fourier Transform (FFT), the Goertzel algorithm has lower computational complexity and requires less memory, making it a good choice for embedded systems or other applications with limited computational resources.

Overall, the Goertzel algorithm is a valuable tool in digital signal processing that allows for efficient and accurate analysis of specific frequencies in a signal.

BACKGROUND

02 "GOERTZEL" ALGORITHM

Detailed Explanation:

The basic idea behind the algorithm is to take advantage of the fact that a single frequency in a signal can be represented by a complex sinusoid. The Goertzel algorithm computes the complex output of a digital filter for a specific frequency, using a fixed set of coefficients that depend only on the desired frequency.

Example Code:

```
function power = goertzel(samples, freq, sample_rate)
    N = length(samples);
    k = round(0.5 + N * freq / sample_rate);
    omega = 2 * pi * k / N;
    s_prev = 0;
    s_prev2 = 0;

    for n = 1:N
        s = samples(n) + 2 * cos(omega) * s_prev - s_prev2;
        s_prev2 = s_prev;
        s_prev = s;
    end

    power = s_prev2 * s_prev2 + s_prev * s_prev - cos(omega) * s_prev *
s_prev2;
end
```

BACKGROUND

02 "GOERTZEL" ALGORITHM

Detailed Explanation of the Code:

This is a function definition with two parameters: **samples**, which is a list of audio samples, and **freq**, which is the frequency of the tone we want to detect.

```
function power = goertzel(samples, freq, sample_rate)
```

This line calculates the number of samples in the input signal and assigns it to the variable **N**.

```
N = length(samples);
```

The Goertzel algorithm works by calculating the power at a specific frequency bin in the discrete Fourier transform (DFT) of a signal. The index of this frequency bin is given by: **k = round(0.5 + freq / (sample_rate / N))**, the 0.5 is added to ensure that we round to the nearest integer.

freq is the frequency we want to extract, **sample_rate** is the sampling rate of the signal, **N** is the length of the signal, and **k** is the index of the frequency component in the DFT.

So, the resulting value of **k** is the index of the frequency component in the DFT that corresponds to the specified frequency **freq**. This is the frequency bin that the Goertzel algorithm will compute the power of.

```
k = round(0.5 + N * freq / sample_rate);
```

This line calculates the value of the constant **ω** that is used in the Goertzel algorithm. **ω** is calculated using the formula **ω = 2πk / N**.

```
omega = 2 * pi * k / N;
```

BACKGROUND

02 “GOERTZEL” ALGORITHM

Detailed Explanation of the Code:

These two lines initialize two variables **s_prev** and **s_prev2** to zero. These variables are used to store the two previous samples of the signal.

```
s_prev = 0;
s_prev2 = 0;
```

This loop, which iterates over all the samples in the input signal, implements the Goertzel algorithm. They calculate the value of **s**, which represents the frequency component at index **k**. The variable **s_prev2** is used to store the value of **s_prev** from the previous iteration, and **s_prev** is used to store the current value of **s**. This way, we can calculate **s** for each sample in the input signal.

```
for n = 1:N
        s = samples(n) + 2 * cos(omega) * s_prev - s_prev2;
        s_prev2 = s_prev;
        s_prev = s;
end
```

s is a variable that represents the state of the filter at each iteration of the loop. It is updated using a recurrence relation that depends on the current input sample **samples[n]**, the previous state **s_prev**, and the state before that **s_prev2**.
This formula can be derived from the difference equation of a second-order digital filter, which is the basis of the Goertzel algorithm. The purpose of this update equation is to filter out all frequency components except the one at the index **k**.

BACKGROUND

02 “GOERTZEL” ALGORITHM

Detailed Explanation of the Code:

This line calculates the power of the frequency component at index k. The power is calculated using the formula power = s_prev2^2 + s_prev^2 - 2cos(ω)s_prevs_prev2. This formula represents the magnitude squared of the frequency component at index k, which is the power of that component. This value represents the strength of the tone at the given frequency in the input signal.

```
power = s_prev2 * s_prev2 + s_prev * s_prev - cos(omega) * s_prev * s_prev2;
```

The first two terms of the equation represent the magnitude squared of the two recursive values s_prev and s_prev2, which are the two intermediate values obtained during the Goertzel algorithm calculation. The last term of the equation represents the product of s_prev and s_prev2, multiplied by the cosine of the frequency component's angle in the complex plane (i.e., the frequency component's phase angle). This term is subtracted from the sum of the first two terms to remove any phase-related components from the final power calculation. Therefore, the resulting value of power is the power of the frequency component at index k.

The Code

BY USING MATLAB TO PERFORM THE FOLLOWING STEPS:

1) (10%) Write a Matlab function x = sym2TT(S) that generates a 100 milliseconds time-domain signal corresponding to a given DTMF symbol S. The set of all DTMF symbols and their corresponding frequencies are indicated in Figure 1.

```
function x = sym2TT(S)
    % Define the DTMF frequency pairs
    F = [697, 770, 852, 941; 1209, 1336, 1477, 1633];

    % Define the DTMF symbol table
    DTMF = ['1', '2', '3', 'A'; '4', '5', '6', 'B'; '7', '8', '9', 'C'; '*', '0', '#', 'D'];

    % Find the row and column indices of the input symbol in the DTMF table
    [r, c] = find(DTMF == S);

    % Generate the corresponding DTMF signal
    f1 = F(1, r);
    f2 = F(2, c);
    fs = 8000;
    Ts = 1/fs;
    t = 0:Ts:0.1;
    x = sin(2*pi*f1*t) + sin(2*pi*f2*t);
end
```

2) (10%) Generate a signal x(t) which contains a sequence of DTMF symbols corresponding to your cell phone number. Add a 20 milliseconds guard interval between consecutive DTMF symbols.

```
% Generate the DTMF sequence for phone number 01014227773
% Followed by a silence (guard) interval of 20 ms
number = '01014227773';
fs = 8000; Ts = 1/fs;
x = [];
for i = 1:length(number)
        symbol = number(i);
        dtmf_signal = sym2TT(symbol);
        x = [x, dtmf_signal, zeros(1, fs*0.02)];
end
```

3) (10%) Create a signal y(t) which is the signal x(t) contaminated with additive white Gaussian noise with variance 0.1.

```
% Add additive white Gaussian noise with variance 0.1
SNR = 10*log10(var(x)/0.1); % Calculate the required SNR for a variance of 0.1
y = awgn(x, SNR, 'measured'); % Add noise to the signal x
```

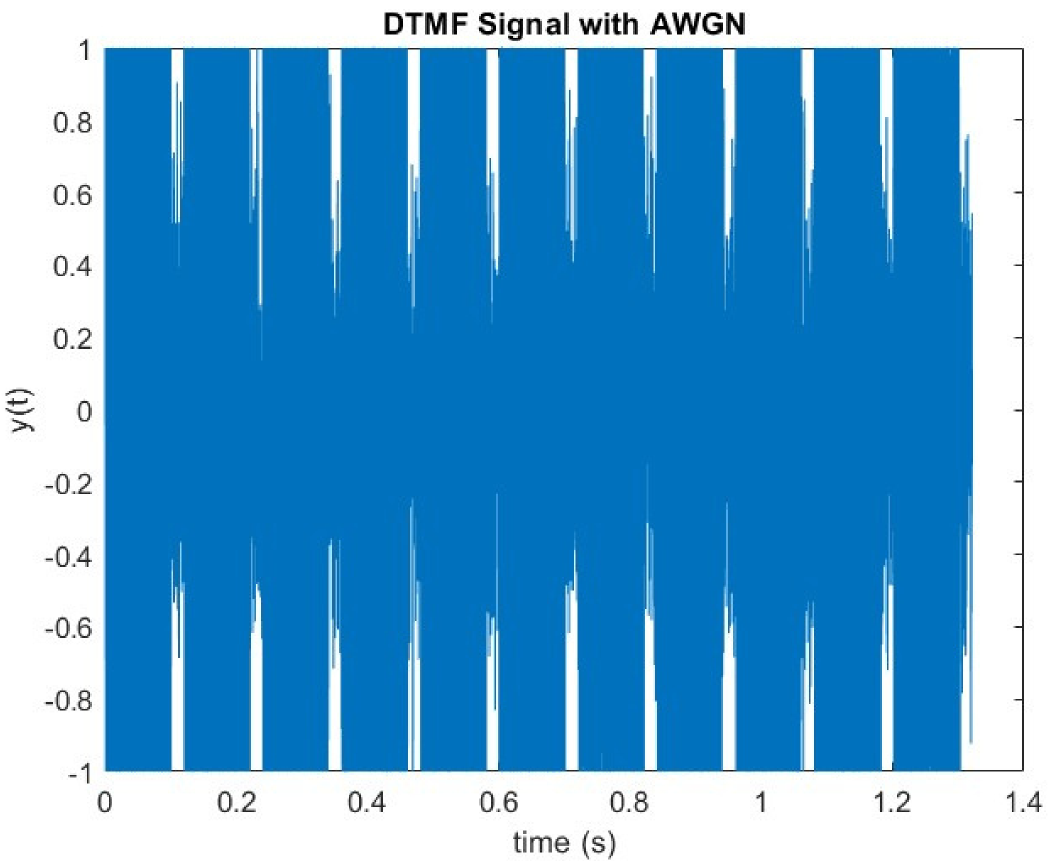
4) (10%) Store the generated signal y(t) as an audio file with extension (*.wav)

```
% Write audio file
filename = 'y(t).wav';
myaudiowrite(filename, y, fs, "BitsPerSample", 16);

% Load audio file
[y,fs] = audioread('y(t).wav');
```

So that the audio does not get clipped to 1s, a modified function **myaudiowrite** is used instead of the default MATLAB function **audiowrite**. In the modified function, the signal does not get clipped to 1s.

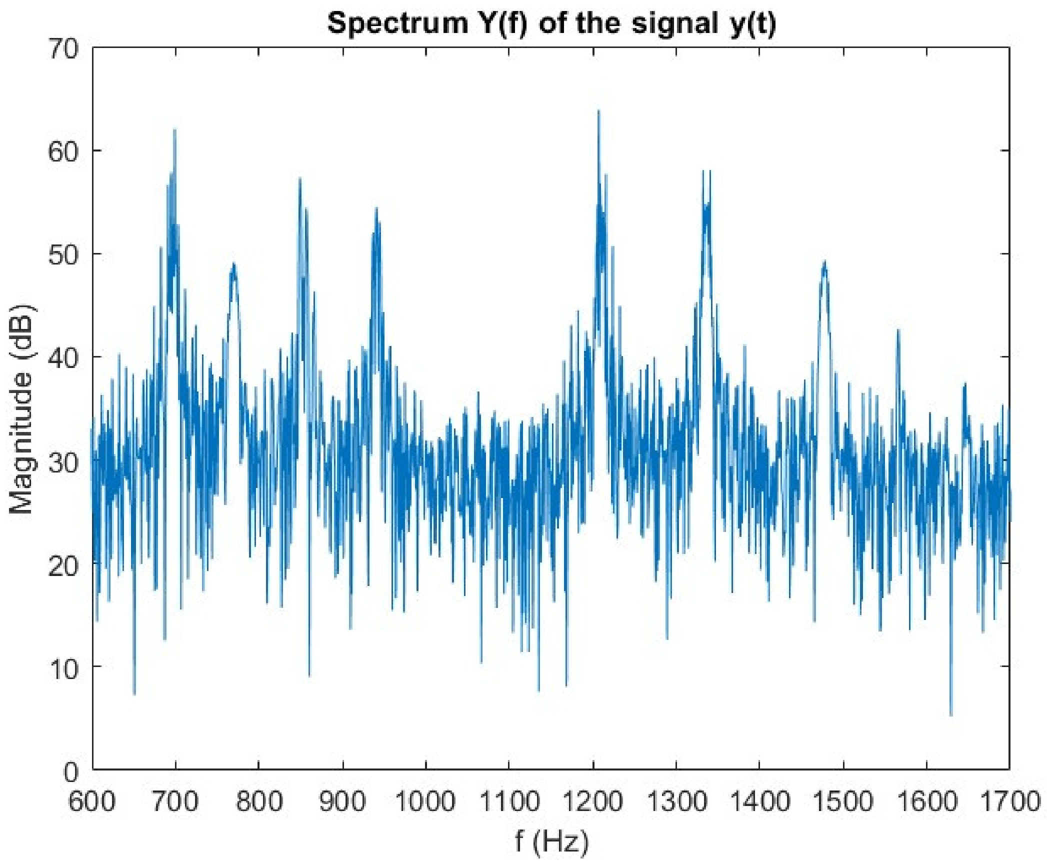
5) (10%) Plot the signal y(t). Note that, this plot does not provide any frequency-domain information about y(t).

```
% Plot the signal y(t)
t = (0:1:length(y)-1)/fs;
figure;
plot(t, y);
xlabel('time (s)');
ylabel('y(t)');
title('DTMF Signal with AWGN');
```



6) (10%) Use the FFT command to obtain the spectrum Y(f) of the signal y(t), hence plot the magnitude spectrum (in dB) in the frequency range 600Hz <= f <= 1700Hz. Note that, this plot does not provide any time-domain information about y(t).

```
% Compute the signal in the frequency domain
Y = fft(y);
f = linspace(-fs/2,fs/2,length(Y));
Y_Shifted = fftshift(Y);

% Plot the signal in the frequency domain
figure;
plot(f,20*log10(abs(Y_Shifted)))
xlim([600,1700])
title('Spectrum Y(f) of the signal y(t)')
xlabel('f (Hz)')
ylabel('Magnitude (dB)')
```



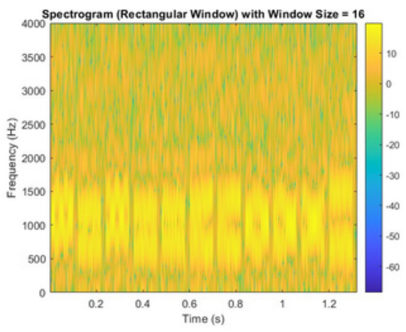
7) Write your own Matlab code to create a set of spectrograms of the signal y(t) using the following parameters:

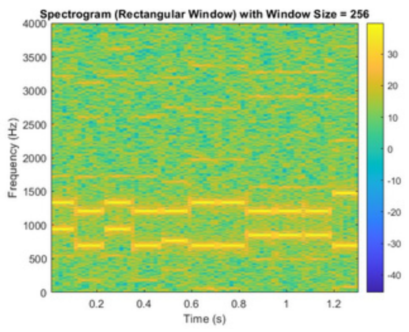
- Overlap between the sliding windows: 50%
- FFT size: 2^14, this is the total number of samples in time domain and in the frequency domain as well.
- Time-domain window size: use the following values {16, 64, 256, 1024, 4096}.
- (10%) Create a spectrogram for each window size ({16, 64, 256, 1024, 4096}) using a rectangular time-domain window.

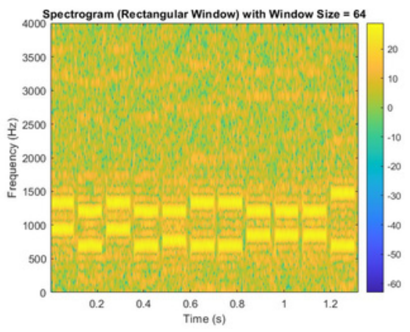
```
% Define the parameters
overlap = 0.5; % 50% overlap
fft_size = 2^14; % FFT size
window_sizes = [16, 64, 256, 1024, 4096]; % Time-domain window sizes

% Create spectrograms using rectangular time-domain window
for i = 1:length(window_sizes)
    window = rectwin(window_sizes(I));
    [S,F,T] = spectrogram(y,window,overlap*length(window),fft_size,fs,'yaxis');
    figure;
    imagesc(T,F,20*log10(abs(S)))
    set(gca,'YDir','normal')
    title(sprintf('Spectrogram (Rectangular Window) with Window Size = %d',window_sizes(i)))
    xlabel('Time (s)')
    ylabel('Frequency (Hz)')
    colorbar
end
```

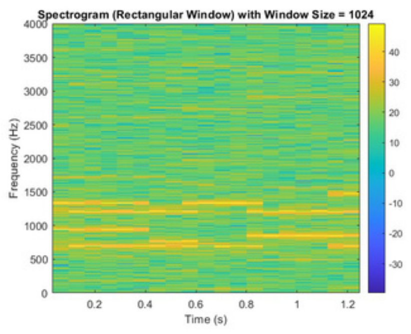
7) Write your own Matlab code to create a set of spectrograms of the signal y(t) using the following parameters:

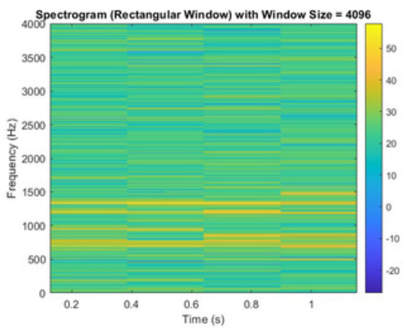
- Overlap between the sliding windows: 50%
- FFT size: 2^14, this is the total number of samples in time domain and in the frequency domain as well.
- Time-domain window size: use the following values {16, 64, 256, 1024, 4096}.
- (10%) Create a spectrogram for each window size ({16, 64, 256, 1024, 4096}) using a rectangular time-domain window.











- (2%) Which window size provides the worst time-domain resolution?

  Worst time-domain resolution: t4096

- (2%) Which window size provides the worst frequency-domain resolution?

  Worst frequency-domain resolution: t16

- (2%) Which window size provides a ”kind-of” optimal trade-off between time and frequency resolutions?

  Optimal trade-off: t256

- (4%) Change the time-domain window to a Blackman window and create a spectrogram for each window size (t16, 64, 256, 1024, 4096u). For a given window size, which window type (rectangular or Blackman) provides a better frequency resolution?

  Blackman window provides better frequency resolution than a rectangular window.

```
% Create spectrograms using blackman time-domain window
for i = 1:length(window_sizes)
    window = blackman(window_sizes(i));
    [S,F,T] = spectrogram(y,window,overlap*length(window),fft_size,fs,'yaxis');
    figure;
    imagesc(T,F,20*log10(abs(S)))
    set(gca,'YDir','normal')
    title(sprintf('Spectrogram (Blackman Window) with Window Size = %d',window_sizes(i)))
    xlabel('Time (s)')
    ylabel('Frequency (Hz)')
    colorbar
end
```

A rectangular window has a sharp transition between the signal and zero, causing spectral leakage and reducing accuracy in the frequency content of the signal. A Blackman window reduces spectral leakage by smoothly tapering the signal to zero at the edges of the window, resulting in a more accurate representation of the frequency content of the signal in the spectrogram.

- (2%) Which window size provides the worst time-domain resolution?

  Worst time-domain resolution: t4096

- (2%) Which window size provides the worst frequency-domain resolution?

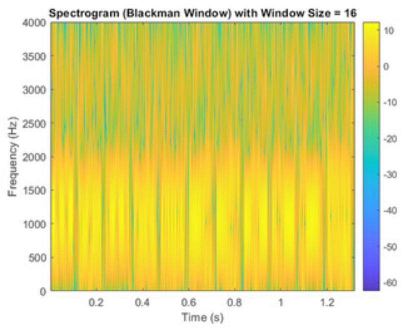
  Worst frequency-domain resolution: t16

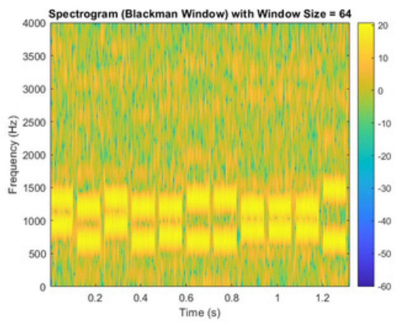
- (2%) Which window size provides a ”kind-of” optimal trade-off between time and frequency resolutions?

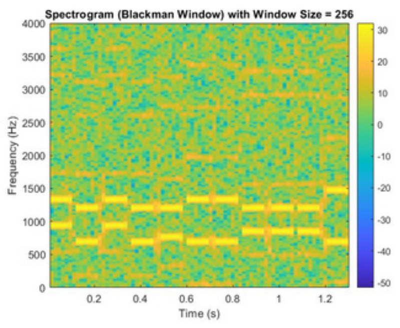
  Optimal trade-off: t256

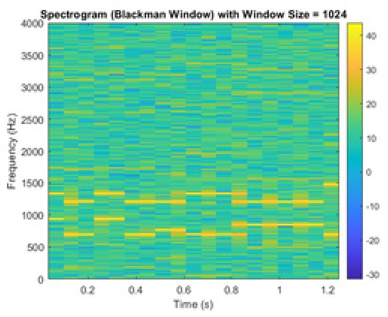
- (4%) Change the time-domain window to a Blackman window and create a spectrogram for each window size (t16, 64, 256, 1024, 4096u). For a given window size, which window type (rectangular or Blackman) provides a better frequency resolution?

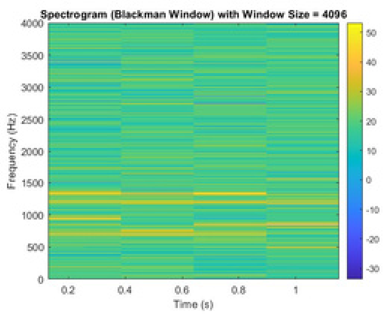
  Blackman window provides better frequency resolution than a rectangular window







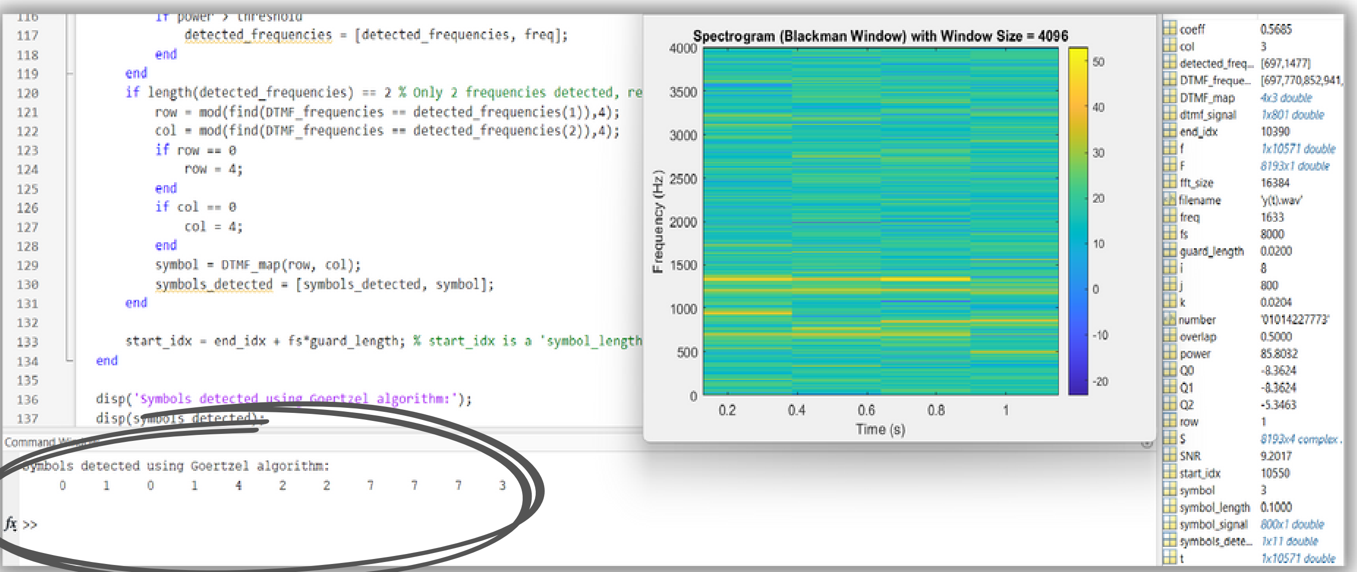




- (10%) Do your research on how to use the Matlab function “Goertzel” to decode DTMF symbols.
- (10%) Write your own Matlab code to decode the signal y(t), verify that you will obtain your cell phone number.

```
% Decode DTMF symbols using Goertzel algorithm
symbol_length = 0.1; % Duration of each DTMF symbol
guard_length = 0.02; % Guard interval
start_idx = 1;
end_idx = 0;
symbols_detected = [];

while start_idx <= length(y) - fs*symbol_length
    % Extract the symbol signal
    end_idx = start_idx + fs*symbol_length - 1;
    symbol_signal = y(start_idx:end_idx); % The segment extracted that corresponds to a single DTMF symbol

    % Decode the symbol using the Goertzel algorithm
    DTMF_frequencies = [697, 770, 852, 941, 1209, 1336, 1477, 1633];
    DTMF_map = [1, 2, 3; 4, 5, 6; 7, 8, 9; -1, 0, -2];
    threshold = 10e3;
    detected_frequencies = [];
    for i = 1:length(DTMF_frequencies)
        freq = DTMF_frequencies(i);
        k = freq*symbol_length/fs; % Compute the index of the desired frequency bin
        w = 2*pi*k/symbol_length;
        coeff = 2*cos(w);
        Q2 = 0;
        Q1 = 0;
        for j = 1:length(symbol_signal)
            Q0 = symbol_signal(j) + coeff*Q1 - Q2;
            Q2 = Q1;
            Q1 = Q0;
        end
        power = abs(Q1)^2 + abs(Q2)^2 - 0.5*coeff*Q1*Q2;
        % Power calculated for the frequency bin with index (k) of the desired frequency (freq)
        if power > threshold
            detected_frequencies = [detected_frequencies, freq];
        end
    end
    if  length(detected_frequencies) == 2 % Only 2 frequencies detected, referring to a specific DTMF symbol
        row = mod(find(DTMF_frequencies == detected_frequencies(1)),4);
        col = mod(find(DTMF_frequencies == detected_frequencies(2)),4);
        if row == 0
            row = 4;
        end
        if col  ==  0
            col = 4;
        end
        symbol = DTMF_map(row, col);
        symbols_detected = [symbols_detected, symbol];
    end

    start_idx = end_idx + fs*guard_length; % start_idx is a 'symbol_length + guard_length' over the current one
end

disp('Symbols detected using Goertzel algorithm:');
disp(symbols_detected);
```

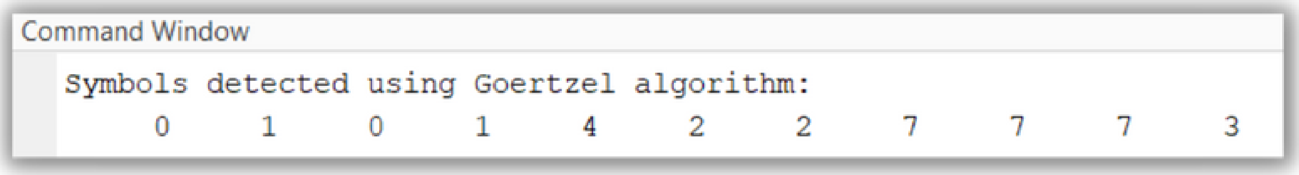
- (10%) Do your research on how to use the Matlab function “Goertzel” to decode DTMF symbols.
- (10%) Write your own Matlab code to decode the signal y(t), verify that you will obtain your cell phone number.



```
            if power > threshold
                detected_frequencies = [detected_frequencies, freq];
            end
        end
        if length(detected_frequencies) == 2 % Only 2 frequencies detected, re
            row = mod(find(DTMF_frequencies == detected_frequencies(1)),4);
            col = mod(find(DTMF_frequencies == detected_frequencies(2)),4);
            if row == 0
                row = 4;
            end
            if col == 0
                col = 4;
            end
            symbol = DTMF_map(row, col);
            symbols_detected = [symbols_detected, symbol];
        end

        start_idx = end_idx + fs*guard_length; % start_idx is a 'symbol_length
    end

    disp('Symbols detected using Goertzel algorithm:');
    disp(symbols_detected);
```

```
Command Window
Symbols detected using Goertzel algorithm:
     0     1     0     1     4     2     2     7     7     7     3

fx >>
```

coeff	0.5685
col	3
detected_freq...	[697,1477]
DTMF_freque...	[697,770,852,941,
DTMF_map	4x3 double
dtmf_signal	1x801 double
end_idx	10390
f	1x10571 double
F	8193x1 double
fft_size	16384
filename	'y(t).wav'
freq	1633
fs	8000
guard_length	0.0200
i	8
j	800
k	0.0204
number	'01014227773'
overlap	0.5000
power	85.8032
Q0	-8.3624
Q1	-8.3624
Q2	-5.3463
row	1
S	8193x4 complex .
SNR	9.2017
start_idx	10550
symbol	3
symbol_length	0.1000
symbol_signal	800x1 double
symbols_dete...	1x11 double
t	1x10571 double



```
Command Window
Symbols detected using Goertzel algorithm:
     0     1     0     1     4     2     2     7     7     7     3
```

Contributions	Team Member	Load
Created the MATLAB function x = sym2TT(S) that generates a 100 milliseconds time-domain signal corresponding to a given DTMF symbol S. Generated the signal x(t) which contains a sequence of DTMF symbols corresponding to the cell phone number, with a 20 milliseconds guard interval between consecutive DTMF symbols.	Nouran Hamdy	20%
Created the signal y(t) which is the signal x(t) contaminated with additive white Gaussian noise with a variance of 0.1. Stored the generated signal y(t) as an audio file with extension (*.wav), created the function 'myaudiowrite' which is an edited version of the MATLAB function 'audiowrite' to avoid audio clipping. Plotted the signal y(t), computed its spectrum Y(f), and plotted the magnitude spectrum (in dB) in the frequency range 600Hz < f < 1700Hz.	Nahed Emad	30%
(Background Section) Researched the idea behind the spectrogram, why it is used, and how. Created a set of spectrograms with the given parameters for the frequency-time analysis of the generated audio signal y(t). Studied the effect of different windows (rectangular and blackman), and different window sizes on the time-domain & frequency-domain resolution.	Mai Sherif	30%
(Background Section) Researched Goertzel Algorithm in decoding DTMF symbols, why it is preferred over other DFT algorithms, its dependance on the idea of digital filters, and how it calculates the power of a specific frequency component in a given number of samples to identify which tones where pressed. Wrote the MATLAB code which decodes the DTMF symbols pressed in the audio signal using Goertzel Algorithm, thus, obtaining the cell phone number provided.	Nouran Hamdy	20%