Queen’s University

ELEC 324

Lab 3 Report

Daniil Nistribenko

Date:
13/11/2025

ELEC 324 Lab 3 – Fourier Series, Fourier Transform and Signal Spectra

1 Introduction

The purpose of this lab is to familiarize yourself with properties of the discrete Fourier transform and discrete-time Fourier series. You will use them to study signals in the frequency domain; you will also develop tools to construct a *spectrogram*, which is a very useful way of presenting both time- and frequency-domain information of a signal, together.

1.1 Preparation

Background material on DTFT, DTFS and DFT can be found in lecture notes and the textbook (consult the posted course syllabus).

Questions marked "Pre-lab" need to be answered *before* the lab. Bring your pre-lab answers to the lab with you.

2 Periodic Signals

In this lab you will work with discrete-time periodic signals (DTPS). For reasons which will be clear in the following sections, a very important property of these signals is their *periodicity*. Recall that some discrete-time signals which appear to be periodic are not, and that a DT signal obtained from sampling a CT periodic signal may not be periodic. Can you construct an example where an *aperiodic* CT signal is sampled and the resultant DT signal is periodic?

Questions

2.0.1 Pre-lab: Periodic and Non-Periodic Signals

Which of the following discrete-time signals are periodic? For the periodic ones, what is the period? For the aperiodic ones, provide a justification.

1. $x[n] = \cos(n)\,, n = 0, 1, 2, 3, ...$

2. $y[n] = \sin(\frac{\pi}{5}n)\,, n = 0, 1, 2, 3, ...$

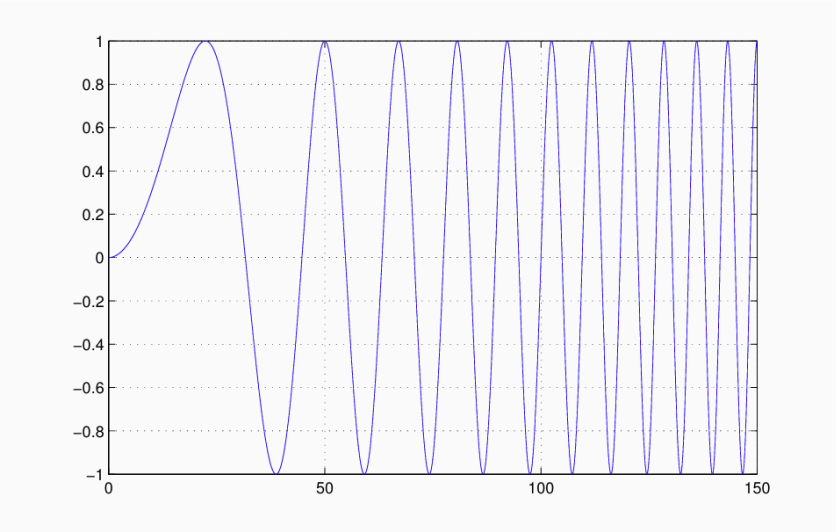
3. $z[n] = e^{j\pi(n+3)}\,, n = 0, 1, 2, 3, ...$

Plot each of the above in Matlab using the *stem* command. Select a domain which highlights their periodicity (or lack thereof).

Answer:
Provided in Prelab section at the back of this document

2.0.2 Prelab: Creating a DTPS

Consider the aperiodic continuous time signal $x(t) = \sin(\frac{\pi}{1000}t^2)$, shown below.



This is clearly not a periodic signal. For reasons which will become apparent later, we are interested in creating a DTPS $y[n]$ from this CT signal. Using Matlab, sample $x(t)$ at integer-valued time intervals from $t = 0$ to $t = 99$. Take the 100 samples as a period and repeat it to form 3 periods and assign them to $y[n]$. Plot $y[n]$.

Answer:
Provided in Prelab section at the back of this document

3 Discrete-Time Fourier Transform and Series

The infinite series that defines the discrete-time Fourier transform (DTFT) does not converge when applied to a DTPS $x_p[n]$. However, one can multiply $x_p[n]$ with a rectangular window $u[n] - u[n - N_F]$, where N_F equals one or integer-multiples of the fundamental period of the DTPS. Let the windowed signal be $x[n] = x_p[n](u[n] - u[n - N_F])$. The DTFT of $x[n]$ is given by

$$X(\mathrm{j}\Omega) = \sum_{n=0}^{N_F-1} x[n]e^{-j\Omega n}.$$

The discrete-time Fourier series (DTFS) of a DTPS $x_p[n]$ using a representational period of N_F samples is expressed as

$$x_p[n] = \sum_{k=\langle N_F\rangle} X[k]e^{jk\omega_0 n},$$

with

$$X[k] = \frac{1}{N}\sum_{n=\langle N_F\rangle} x_p[n]e^{-jk\omega_0 n},$$

where $\omega_0 = \frac{2\pi}{N_F}$. Note that $x[n] = x_p[n]$ for $0 \le n \le N_F - 1$. If one is interested in representing sample values only over this time interval, one may be willing to accept a representation that implies that the samples are periodically replicated or have zero values outside of the interval.

3.0.1 Pre-lab: DTFS and Sampling DTFT

Suppose $X(\mathrm{j}\Omega)$ is sampled at $\Omega_k = \omega_0 k$, k an integer. How are the samples $X(\mathrm{j}\Omega_k)$ related to the harmonic function $X[k]$? For the purpose of representing $x[n]$ in the frequency domain, is it necessary to know $X(\mathrm{j}\Omega)$ at every possible value of Ω?

Answer:

Provided in Prelab section at the back of this document

3.1 Discrete Fourier Transform (DFT)

The N-point DFT $X_{DFT}[k], k = 0, \ldots, N-1$ of a signal $x[n]$ that is defined only over the interval $0 \leq n \leq N-1$ can be viewed as either

1. the N samples of the DTFT of a signal $x_a[n]$, where $x_a[n]$ is equal to $x[n]$ for $0 \leq n \leq N-1$ and to zero outside of this interval. (The subscript "a" stands for "aperiodic".)

2. the DTFS coefficients of $x_p[n]$ multipled by N, where $x_p[n]$ is the periodically extended version of $x[n]$, i.e., $x_p[n] = \sum_{k=-\infty}^{+\infty} x[n+kN]$. Thus, we have $X_{DFT}[k] = NX_{DTFS}[k], k = 0, \ldots, N-1$. In general, $X_{DFT}[k] = NX_{DTFS}[k]$ for all integer k since both $X_{DFT}[k]$ and $X_{DTFS}[k]$ are periodic in k. Usually, we are only interested in one fundamental period $k = 0, \ldots, N-1$. The correspondence between k and physical (analog) frequency is $F = F_s k/N$, where F_s denotes the sampling frequency.

Either view has its limitations. In particular, if one were interested in analysing a signal $x'[n]$, and $x[n]$ were obtained from $x'[n]$ through windowing, the effect of the windowing process must be considered. The operation of *windowing* a DT signal to time interval $[N_1, N_2]$ refers to multiplying $x'[n]$ by the sequence $(u[n-N_1] - u[n-\mathrm{N}_2-1])$.

3.2 DFT of Sinusoidal Signals

In this section we study the DFT of sinusoidal signals.

Questions

3.2.1 Basic DFT

In Matlab, create a discrete signal x representing a sine wave of period 20 and length $L = 100$ (*i.e.* 5 full periods). Perform an N-point DFT, where $N = L$, on the signal using the *fft* command. Perform inverse DFT (IDFT) on the DFT coefficients using the *ifft* command. Plot the original signal, the magnitude of its DFT coefficients, and the reconstructed signal from IDFT. Use *stem* instead of *plot* to be better aware of the discrete values of each signal. Comment on the results. At which points are the DFT coefficients non-zero? Why? [Note: It is instructive for you to scrutinize the values of the IDFT reconstructed signal samples that are submitted to *stem*.]

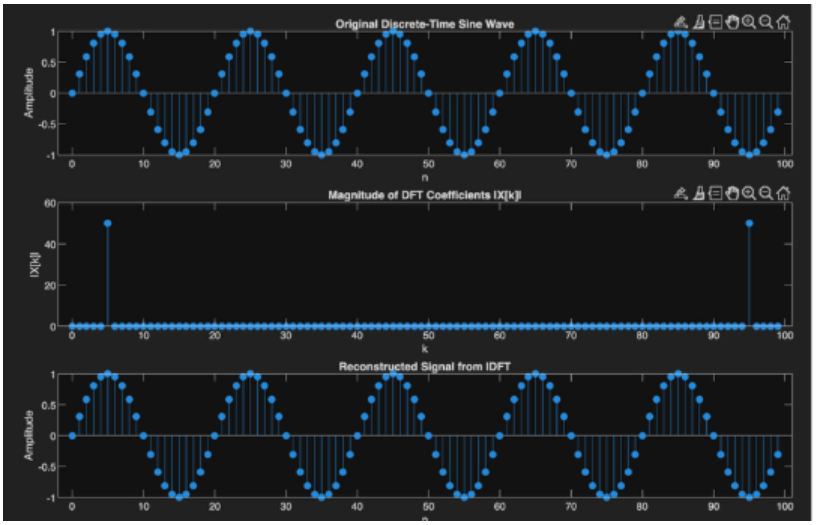
Pitfall to look out for: IDFT must be performed on the complex-valued DFT coefficients. Moreover, the signal produced by *ifft* may be complex-valued: due to finite-numerical precision effects, the imaginary part may have extremely small values around zero. Older version of Matlab may not plot the correct reconstructed signal as *stem* might have dropped the imaginary part. The real part has to be taken before submitting to *stem*. In practice, if the signal is not known a priori to be real, one can quantize the imaginary part to zero when its magnitude is smaller than a threshold, i.e., apply a "deadzone" to the imaginary part.

Answer:

The discrete-time sine wave with a period of 20 and total length L = 100 (5 full periods) produced a clean and periodic signal in the time domain. The magnitude spectrum of its DFT shows two distinct non-zero coefficients at indices k = 5 and k = 95, corresponding to the positive and negative frequency components of the sine wave. All other frequency bins are effectively zero, confirming that the signal contains a single frequency component.

The IDFT accurately reconstructed the original time-domain signal, matching the amplitude and phase of the original waveform. Any small imaginary parts observed were due to finite-precision numerical effects, which are negligible.

This demonstrates that the DFT correctly represents a discrete-time sinusoid in the frequency domain and that the inverse DFT perfectly recovers the original signal when N = L and the signal is periodic within the observation window.



Code:

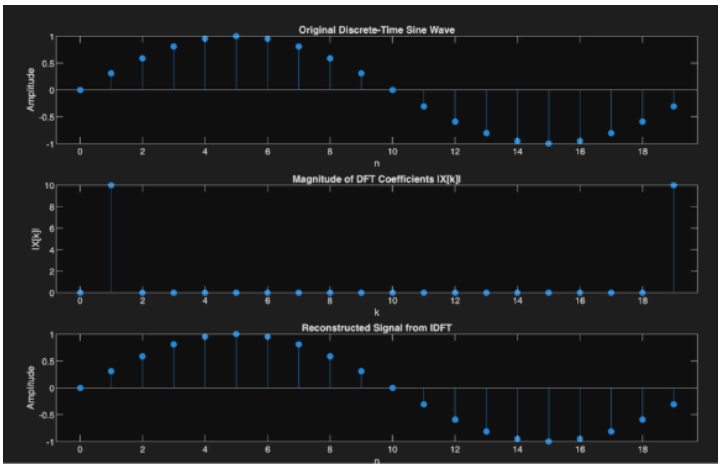
```
% Lab 3 - Section 3.2.1 Basic DFT
clear; close all; clc;

% Parameters
L = 100;
period = 20;
n = 0:L-1;

% Create sine wave with period 20 and 5 full periods in 100 samples
x = sin(2*pi*n/period);

% Compute N-point DFT
X = fft(x, L);

% Compute inverse DFT
x_reconstructed = ifft(X, L);

% Plot results
figure;

subplot(3,1,1);
stem(n, x, 'filled');
title('Original Discrete-Time Sine Wave');
xlabel('n'); ylabel('Amplitude');

subplot(3,1,2);
stem(0:L-1, abs(X), 'filled');
title('Magnitude of DFT Coefficients |X[k]|');
xlabel('k'); ylabel('|X[k]|');

subplot(3,1,3);
stem(n, real(x_reconstructed), 'filled');
title('Reconstructed Signal from IDFT');
xlabel('n'); ylabel('Amplitude');
```

3.2.2 Changing the Transform Size

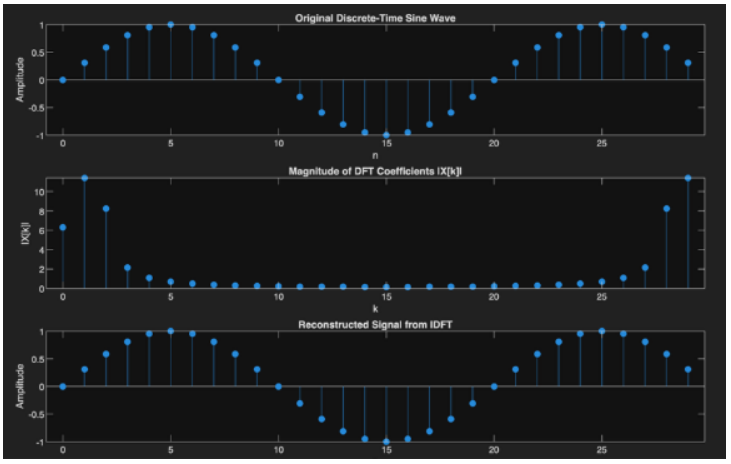
Repeat the plots from Question 3.2.1 with $L = 20$, and then with $L = 30$. Explain any observed differences between the earlier case (which was for $L = 100$) and both of these cases.

Answer:

When the signal length was reduced from L = 100 to L = 20, the DFT still produced two sharp peaks at k = 1 and k = 19, representing the positive and negative frequency components of the sine wave. Because L = 20 exactly matches one period of the signal, the waveform repeats perfectly within the analysis window. As a result, there is no spectral leakage, and the reconstructed signal from the IDFT matches the original waveform precisely.



For L = 30, the signal contains a non-integer number of periods (1.5 periods within the window). This causes the waveform to start and end at different zphases, introducing discontinuities at the window boundaries. In the frequency domain, this appears as energy spread across multiple DFT bins rather than two sharp peaks, an effect known as spectral leakage. The reconstructed signal therefore shows small distortions relative to the ideal sine wave.

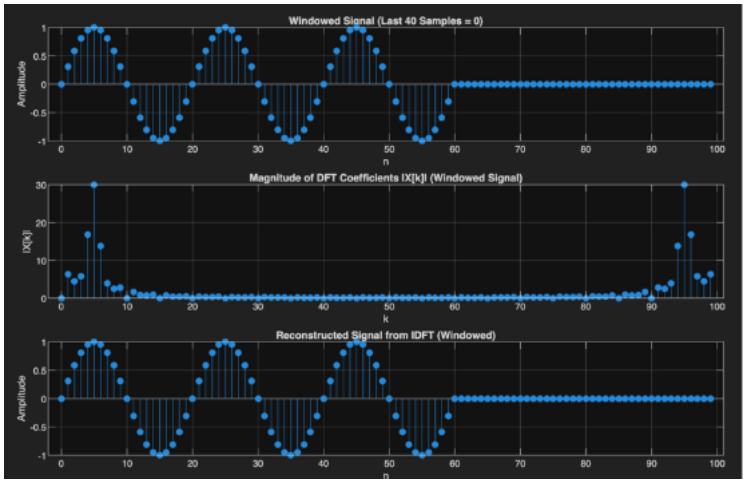


3.2.3 Windowing the Signal

Modify our x signal with $L = 100$ so that the last 40 of its 100 samples are zero. Repeat the plots in Question 3.2.1. Explain any differences in the DFT coefficients, between this and the original results from Question 3.2.1. [Hint: Two points of view may be considered: DTFS and sampling the DTFT.]

Answer:

When the last 40 samples of the signal are set to zero, the DFT shows spreading of energy around the main frequency components instead of sharp peaks. This occurs because multiplying the signal by a rectangular window in the time domain corresponds to convolution with a sinc function in the frequency domain. The resulting spectrum displays leakage into adjacent bins, and the reconstructed signal is non-periodic and attenuated after the zeroed region.



Code:

```
% Section 3.2.3: Windowing the Signal
clear; close all; clc;

% Parameters
L = 100;
period = 20;
n = 0:L-1;

% Original full-length sine wave (same as 3.2.1)
x = sin(2*pi*n/period);

% Apply rectangular window: zero out the last 40 samples
x_windowed = x;
x_windowed(end-39:end) = 0;     % last 40 samples = 0

% Compute DFT and IDFT
X_full = fft(x, L);             % original signal spectrum
X_windowed = fft(x_windowed, L); % windowed signal spectrum
x_recon = ifft(X_windowed, L);

% --- Plot results ---
figure('Name','3.2.3 Windowing the Signal','NumberTitle','off');

subplot(3,1,1);
stem(n, x_windowed, 'filled');
title('Windowed Signal (Last 40 Samples = 0)');
xlabel('n'); ylabel('Amplitude'); grid on;

subplot(3,1,2);
stem(0:L-1, abs(X_windowed), 'filled');
title('Magnitude of DFT Coefficients |X[k]| (Windowed Signal)');
xlabel('k'); ylabel('|X[k]|'); grid on;

subplot(3,1,3);
stem(n, real(x_recon), 'filled');
title('Reconstructed Signal from IDFT (Windowed)');
xlabel('n'); ylabel('Amplitude'); grid on;
```

3.3 DFT of an Aperiodic Signal – Linear-Frequency-Modulation

We will now examine how the signal length L and transform size N interact and affect transform results when working with an aperiodic signal. Run the code below:

```
% aperiodic.m
clf;
L = 100;
L_factor = 1;
N_factor = 1;
L = round(L*L_factor);
n = [0:1:L-1];
x = sin(pi/1000*(n.*n));
subplot(3,1,1);
stem(n,x);
title ('x[n]');
subplot(3,1,2);
y = fft(x,L*N_factor);
plot(abs(y));
title ('DFT');
subplot(3,1,3);
xr = ifft(y);
stem(n,xr(1:L));
title ('IDFT');
```

Questions:

3.3.1 Pre-Lab

Describe the role of the L_factor and N_factor variables in the above code.

Answer:

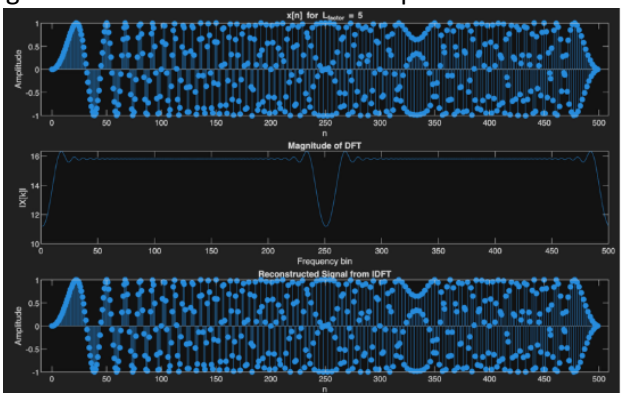
Provided in Prelab section at the back of this document

3.3.2 Effect of L

Change L_factor from 1 to 5 in steps of 1; observe and comment on differences on all three plots.

Answer:

As L_{factor} increases from 1 to 5, the signal x[n] becomes longer, and the DFT shows a denser spectrum with improved frequency resolution. The spacing between frequency samples decreases as L increases. The IDFT reconstruction remains consistent with the original signal for all cases, confirming correct transform and inverse operations.



Code:

```
clf;

L_base = 100;
N_factor = 1;

for L_factor = 1:5
   % --- Compute signal ---
            L = round(L_base * L_factor);
            n = 0:(L-1);
            x = sin(pi/1000 * (n.^2)); % aperiodic chirp-like signal

   % --- Compute DFT and IDFT ---
            y = fft(x, L * N_factor);
   xr = ifft(y);

   % --- Plot results ---
            figure(L_factor)
   subplot(3,1,1);
   stem(n, x, 'filled');
            title(['x[n] for L_{factor} = ', num2str(L_factor)]);
   xlabel('n'); ylabel('Amplitude');

   subplot(3,1,2);
            plot(abs(y));
   title('Magnitude of DFT');
   xlabel('Frequency bin'); ylabel('|X[k]|');

   subplot(3,1,3);
   stem(n, real(xr(1:L)), 'filled');
   title('Reconstructed Signal from IDFT');
   xlabel('n'); ylabel('Amplitude');

   % --- Optional: link figure number for clarity ---
   sgtitle(['Effect of L_{factor} = ', num2str(L_factor)]);
end
```

3.3.3 Effect of N

For a fixed L_factor of 1, vary N_factor and describe its effect.

Answer:

With L_{factor} = 1 fixed and increasing N_{factor}, the time-domain signal x[n] remains unchanged, but the DFT length increases. The frequency-domain plot becomes smoother because more frequency samples are taken, resulting in finer interpolation of the same spectrum. The overall spectral shape does not change, only its resolution in frequency increases. The IDFT reconstruction remains identical to x[n], confirming that changing N_{factor} affects frequency sampling density but not the underlying signal content.

3.3.4 A Limit

What happens to the magnitude spectrum (the magnitudes of the DFT coefficients) when L_factor is 6 or more? Describe the underlying cause of the observed phenomenon.

Answer:

When L_{factor} is 6 or greater, the magnitude spectrum becomes distorted and irregular. The DFT coefficients appear to lose a clear structure, and the spectral pattern no longer represents the expected smooth frequency variation. This occurs because the signal x[n] grows too long relative to the quadratic phase term in $\sin(\pi / [100n_2])$. As L increases, the instantaneous frequency exceeds the Nyquist limit, causing frequency components to alias. The resulting overlap of aliased components produces the observed distortion in the magnitude spectrum.

3.4 Summarizing DTFS, DFT, and DTFT

This section enabled you to visualize the relationships between three Fourier transformations: DTFS, DFT, and DTFT. The three transformations are intimately related. In fact, for a limited duration signal, given the values of one transform, one can always compute the values of the other two transforms. With care, these transforms can be usefully applied to analyze time-varying real-world signals, as we see next.

4 Spectrum Analysis

Real-world signals that one may wish to analyze spectrally are not usually simple periodic signals. Think of speech, music, video, or the output of a seismometer. While it would be possible to perform a Fourier transform on an entire 5-minute audio clip, this is unlikely to be very useful. Since the characteristics of audio signals change considerably over time (as one would expect for a 5-minute audio clip), taking the Fourier transform of the entire signal will effectively mask those variations: it will give you a picture of the overall average spectral content, but it will not convey any dynamic information about the signal. The music notes are all averaged together and therefore it would be hard to differentiate between entirely different genres of music.

Suppose we seek a spectral representation that also conveys some time-domain information. How can this be done? If the Fourier transform changes the domain from time to frequency, how can we re-introduce time into the picture? The answer is the spectrogram, and we have already started the foundation for building one in Section 3. While Matlab does have a spectrogram command (*specgram*), in this section we will build one from scratch.

In this section we will be working with signals based on the following signal frequency modulated (FM) carrier signal:
$x[n] = \sin(\frac{\pi}{100}n^{1.5})$, $n\,\epsilon\,[0, 1499]$.

Questions:

4.0.1 Pre-lab: Frequency Range

Suppose the signal $x[n] = \sin(\frac{\pi}{100}n^{1.5})$, $n\,\epsilon\,[0, 1499]$ has been obtained by sampling a continuous-time (CT) signal at $F_s = 1000$ Hz. Write the expression for the CT FM signal. The instantaneous frequency of a CT sinusoidal signal is defined as the derivative of the sinusoid function argument. Find the instantaneous frequency as a function of time. What is the range of frequencies in Hz swept by the signal?

Answer:
Provided in Prelab section at the back of this document

4.0.2 Time Analysis

Using either the *audiowrite* or *sound* command, listen to the above signal. Note that you will need to specify the sampling rate and the number of bits. The sampling frequency could be anything, but in the interest of keeping the frequencies in the range of human hearing, something around 2000 Hz should be used. To avoid data clipping in *audiowrite* you should multiply the sine function with a number slightly less than 1, say 0.999.

Create a time-reversed version of that signal and listen to that too. Then create a third signal by concatenating the first two signals and listen to that too.

Answer:

To code the described signal:

1. The discrete signal x x[n] = sin([π/100]$n^{1.5}$ is defined for n = 0 to 1499.
2. The signal is scaled by 0.999 to prevent amplitude clipping in audiowrite.
3. The time-reversed signal is produced using fliplr(x).
4. The concatenated signal is formed by joining x[n] and its reverse.
5. Each signal is played and optionally written to an audio file for verification.

Code:

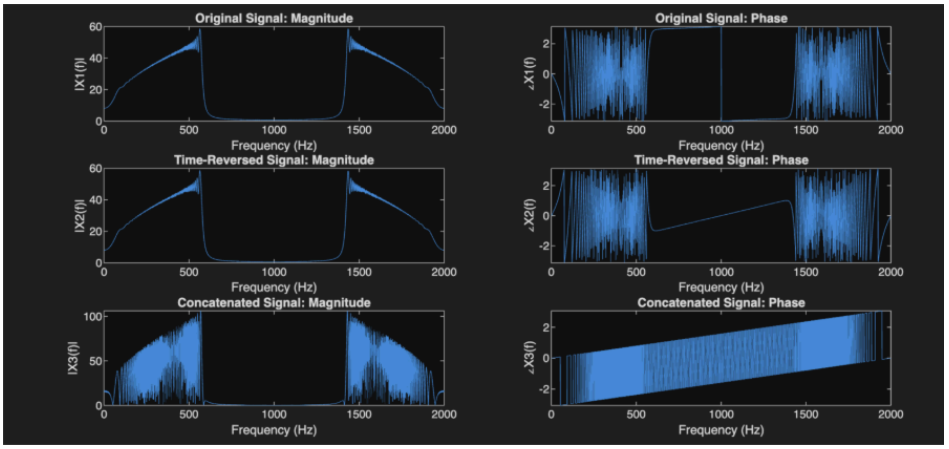
```
% Define signal parameters
Fs = 2000;
n = 0:1499;
x = sin((pi/100) * n.^1.5);  % Discrete-time FM signal
x = 0.999 * x;

% (a) Original signal
sound(x, Fs);
audiowrite('signal1.wav', x, Fs, 'BitsPerSample', 16);

% (b) Time-reversed version
x_rev = fliplr(x);
sound(x_rev, Fs);
audiowrite('signal2_reversed.wav', x_rev, Fs, 'BitsPerSample', 16);

% (c) Concatenated signal
x_combined = [x, x_rev];
sound(x_combined, Fs);
audiowrite('signal3_combined.wav', x_combined, Fs, 'BitsPerSample', 16);
```

4.0.3 Frequency Analysis

Plot the magnitude and phase of the DFT for each of the three signals of the previous question. Comment on their differences. Is this a good way to analyze the combined time and frequency domain characteristics of a signal?

Answer:

The magnitude spectra of the original and time-reversed signals are the same, confirming that time reversal does not change the distribution of frequency components. Their phase spectra, however, are opposite in sign, consistent with the phase conjugation property of time reversal. The concatenated signal shows a wider and more irregular magnitude spectrum because it combines the frequency content of both the forward and reversed segments, and its phase varies more rapidly due to the increased duration and discontinuity between sections. Although the DFT provides insight into the overall spectral composition, it represents only the average frequency content over the entire time window. It does not reveal when specific frequencies occur within the signal. Therefore, the DFT is not an effective method for analyzing combined time and frequency characteristics; a spectrogram or short-time Fourier transform should be used instead.



Code:

```
% Parameters
Fs = 2000;
n = 0:1499;

% Signals from 4.0.2
x = 0.999 * sin((pi/100) * n.^1.5);
x_rev = fliplr(x);
x_combined = [x, x_rev];

% Compute DFTs
X1 = fft(x);
X2 = fft(x_rev);
X3 = fft(x_combined);

% Frequency axis for plotting
f1 = (0:length(X1)-1)*(Fs/length(X1));
f2 = (0:length(X2)-1)*(Fs/length(X2));
f3 = (0:length(X3)-1)*(Fs/length(X3));

% Plot magnitude and phase for each
figure;

subplot(3,2,1);
plot(f1, abs(X1));
title('Original Signal: Magnitude');
xlabel('Frequency (Hz)'); ylabel('|X1(f)|');

subplot(3,2,2);
plot(f1, angle(X1));
title('Original Signal: Phase');
xlabel('Frequency (Hz)'); ylabel('∠X1(f)');

subplot(3,2,3);
plot(f2, abs(X2));
title('Time-Reversed Signal: Magnitude');
xlabel('Frequency (Hz)'); ylabel('|X2(f)|');

subplot(3,2,4);
plot(f2, angle(X2));
title('Time-Reversed Signal: Phase');
xlabel('Frequency (Hz)'); ylabel('∠X2(f)');

subplot(3,2,5);
plot(f3, abs(X3));
title('Concatenated Signal: Magnitude');
xlabel('Frequency (Hz)'); ylabel('|X3(f)|');

subplot(3,2,6);
plot(f3, angle(X3));
title('Concatenated Signal: Phase');
xlabel('Frequency (Hz)'); ylabel('∠X3(f)');
```

4.1 A Waterfall Spectrogram

Some insight in frequency domain analysis can be obtained from the fact that the human ear does not hear frequencies below 30 Hz. Signals below this frequency are not interpreted as sine waves with specific frequencies but rather as changes in the overall signal characteristics. For this reason, our spectrogram will analyze the input signal in "chunks" of $\frac{1}{30}$ seconds.

Our goal is to perform a DFT on each chunk and somehow present this data in a useful, easy-to-understand format. After performing a DFT on each chunk, one way to present the data would be to simply create a separate two-dimensional plot for each chunk. However, if our source signal has a lot of chunks, this quickly becomes unmanageable, both in terms of space required and ease of interpretation. A way to address both of these issues is to introduce a third dimension to our graphical representation of the results. It should then be possible to convey the spectral information much more efficiently.

Questions:

4.1.1 Pre-lab: Transform Size

If one unit of time, n, represents $1/2000$ second, how many samples does a single "chunk" contain for our x signal defined above? How many chunks will have to be processed?

Answer:
Provided in Prelab section at the back of this document

4.1.2 Prelab: The Waterfall Spectrogram

The function below can be used to create what is called a "waterfall spectrogram". Based on your answers from the previous sections, call this function with the appropriate parameters to produce a spectrogram of our x signal defined above. Does this spectrogram do a good job of conveying the time- and frequency-domain information of x?

```
function waterfallspect(s, fs, sizeofspectra, numofspectra)
% Simplified version of Lee and Varaiya's waterfallSpectrogram
% function; modified to use fft.
% Displays a 3-D plot of a spectrogram of the signal s.
%
% Arguments:
%   s - The signal.
%   fs - The sampling frequency (in samples per second).
%   sizeofspectra - The number of samples to use to calculate each
%          spectrum.
%   numofspectra - The number of spectra to calculate.
frequencies = [0:fs/sizeofspectra:fs/2];
offset = floor((length(s)-sizeofspectra)/numofspectra);
for i=0:(numofspectra-1)
    start = i*offset;
    A = abs(fft(s((1+start):(start+sizeofspectra))));
    magnitude(:,(i+1)) = A(1:sizeofspectra/2+1);
end
waterfall(frequencies, 0:(numofspectra-1), magnitude');
xlabel('frequency');
ylabel('time');
zlabel('magnitude');
```

Answer:
Provided in Prelab section at the back of this document

4.2 A Two-Dimensional Spectrogram

A three-dimensional graph is pretty and impressive, but as engineers it's always a good idea to remember that sometimes, "less is more". Is it possible to represent the same information with a two-dimensional graph? The answer is yes. We will use colour to represent the third dimension. Recall from the first lab that the *image* command can be used to display a color image. For example, try the following code:

```
colormap(jet(256));
image([1:256]);
```

You should see a nice rainbow of colours (256 of them to be exact). Matlab automatically stretches images to make them square, which is why the above results in a big color square rather than a simple 1-pixel wide line. Now, try this:

```
image([1:256]');
```

Can you explain what changed and why?

Questions:

4.2.1 A One-Dimensional Fourier Transform

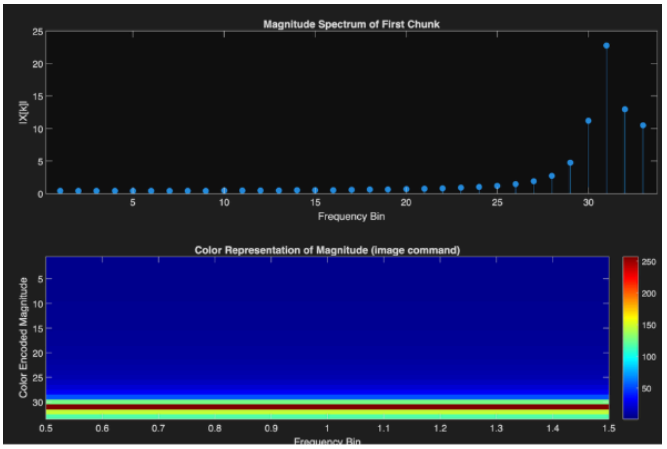
Our first step is to create a 1-D plot of a single DFT. Perform a DFT on the first chunk of x (defined above), but instead of plotting it as you usually would with *plot*, use the *image* command. Here are some points to consider:

- you only need to display the magnitude of the DFT (ignore the phase here)
- to convey the maximum amount of information, the size of the colormap must match the range of values of your DFT; use *colormap(jet(x))*, where x is the desired size

The matlab code for this can be found in the included file oneDimDFT.m. How do you interpret the resulting graph? What do the colours mean? *Hint:* refer back to the introduction of Section 4.2.

Answer:

The image plot uses color to represent the magnitude of each frequency component in the DFT. In the jet colormap, blue corresponds to low magnitude values and red/yellow to high magnitudes. This provides a compact 2D representation of frequency-domain information, where the third dimension (magnitude) is encoded as color. Thus, the same spectral information that would be plotted in 3D (frequency vs. magnitude vs. color) is represented in a simple 2D color strip.



Code:

```
clc; clear; close all;

n = 0:1499;
x = sin(pi/100*(n.*sqrt(n)));

% Take first 66 samples (first chunk)
y = abs(fft(x(1:66)));

% Consider only half of the symmetric spectrum
y_half = y(34:66);

% Normalize for color mapping
y_norm = y_half / max(y_half);

% Use a 256-color jet colormap
colormap(jet(256));

% --- Plot 1: Traditional DFT magnitude ---
subplot(2,1,1);
stem(y_half, 'filled');
title('Magnitude Spectrum of First Chunk');
xlabel('Frequency Bin');
ylabel('|X[k]|');

% --- Plot 2: Color Image Representation ---
subplot(2,1,2);
image(y_norm' * 256);     % Transpose to show vertical color strip
title('Color Representation of Magnitude (image command)');
xlabel('Frequency Bin');
ylabel('Color Encoded Magnitude');
colorbar;
```

4.2.2 A Spectrogram

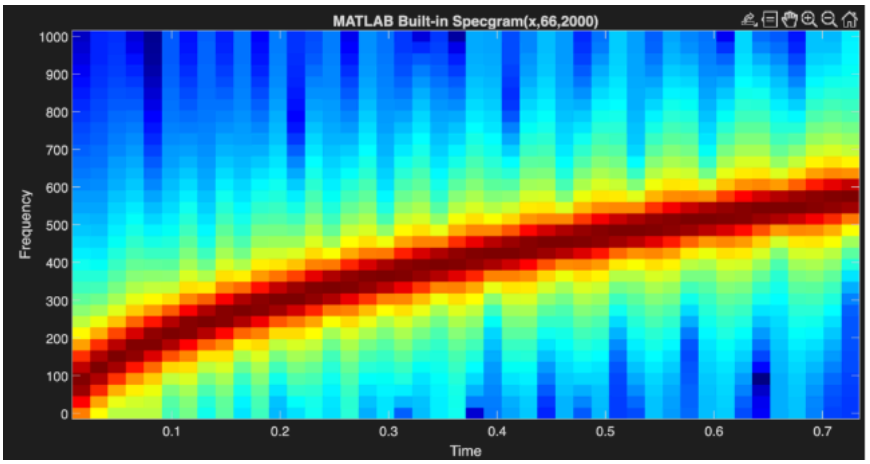
Write a Matlab program which will plot a two-dimensional colour spectrogram by extending the method used in the previous section to put the DFT of each chunk on the same image, in sequential order, with the x-axis representing time and the y-axis representing frequency. Here are some points to consider:

- you only need to display the magnitude of the DFTs (ignore the phase here)
- your spectrogram should only show half of the spectrum (*i.e.* from zero to half the sampling frequency)
- as in the previous question, match the size of the colormap to the range of values present in the transforms

Validate your results by comparing them with Matlab's built-in *specgram* command, used as follows: `specgram(x,66,2000)`. Does your spectrogram convey the same information as the waterfall spectrogram? What are the differences, if any? Describe what changes you would need to make for the x- and y-axis of your spectrogram to take on the same range of values as *specgram*'s.

Answer:

The custom spectrogram shows the same time – frequency information as the waterfall and MATLAB's specgram(x,66,2000). The main differences are that the custom plot uses linear magnitude, chunk/time indices, and no overlap. The spectrogram, on the other hand, uses logarithmic scale, seconds, and hertz axes. To match spectrogram, scale the x-axis as time = (chunk_size/Fs)*index, the y-axis as freq = (Fs/2)*(bin/half_spectrum), and use imagesc(time,freq,data) with axis xy.

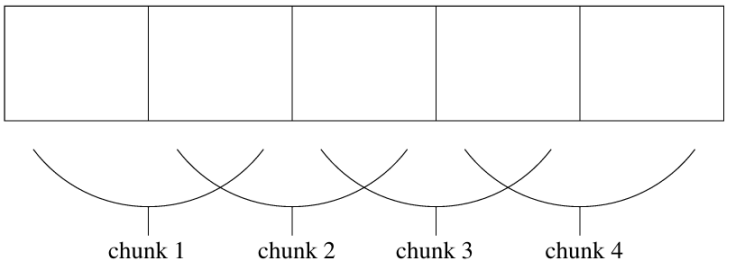


Code:

```
clc; clear; close all;
% Signal definition
Fs = 2000; % Sampling frequency [Hz]
n = 0:1499; % Discrete-time index
x = sin(pi/100*(n.*sqrt(n))); % Frequency modulated signal

% Chunk parameters
chunk_size = 66; % Samples per DFT
num_chunks = floor(length(x)/chunk_size);
half_spectrum = chunk_size/2; % Half spectrum (0 to Fs/2)

% Preallocate spectrogram matrix
spectrogram_matrix = zeros(half_spectrum, num_chunks);

% Compute DFT magnitude for each chunk
for i = 1:num_chunks
    start_idx = (i-1)*chunk_size + 1;
    end_idx   = start_idx + chunk_size - 1;
              chunk      = x(start_idx:end_idx);

    % DFT magnitude
              Y = abs(fft(chunk));

    % Keep only first half (0–Fs/2)
    spectrogram_matrix(:, i) = Y(1:half_spectrum);
end

% Normalize for color scaling
spectrogram_matrix = spectrogram_matrix / max(spectrogram_matrix(:));

% Create frequency and time axes
freq_axis = linspace(0, Fs/2, half_spectrum); % Hz
time_axis = ((1:num_chunks) * chunk_size) / Fs; % seconds

% Display spectrogram
figure('Name','Custom 2D Colour Spectrogram (with Frequency & Time Axes)');
colormap(jet(256));
imagesc(time_axis, freq_axis, spectrogram_matrix); % imagesc uses real-world axes
axis xy; % Flip Y so low freq at bottom
xlabel('Time (seconds)');
ylabel('Frequency (Hz)');
title('Custom Spectrogram (DFT Magnitude in Colour)');
colorbar;

% Comparison with MATLAB's specgram()
figure('Name','MATLAB Built-in Specgram for Comparison');
specgram(x, chunk_size, Fs);
title('MATLAB Built-in Specgram(x,66,2000)');
colormap(jet(256));
```

4.2.3 An Improvement

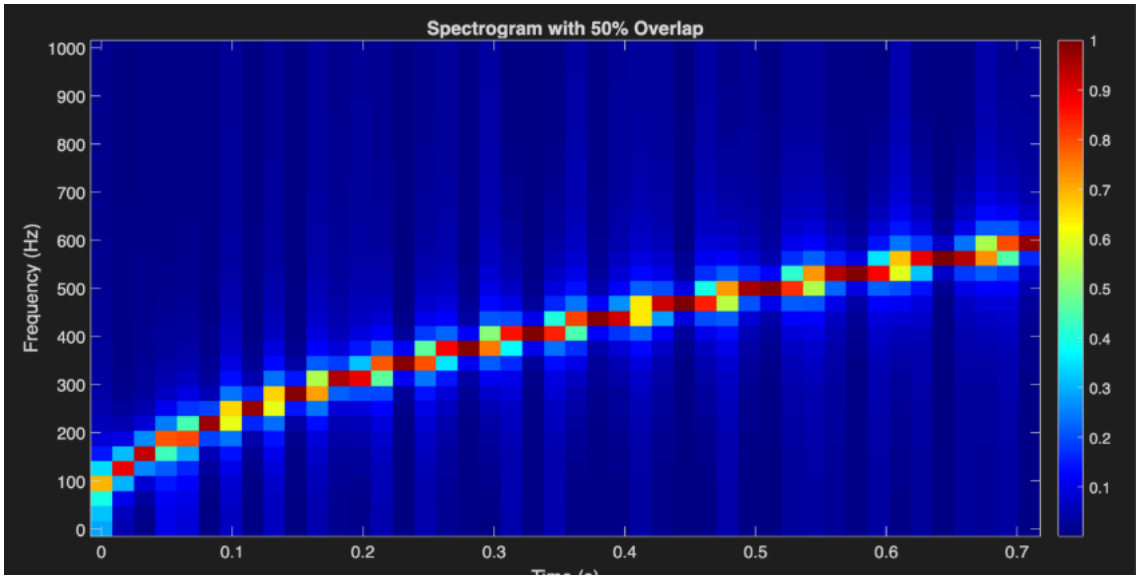
To reduce coarseness in your spectrogram, you can perform your DFTs on *overlapping* chunks, as illustrated below:



Every chunk contains half of the previous chunk's samples. So in the above diagram, the first two squares make up the first chunk, squares 2 and 3 make up the second chunk, squares 3 and 4 make up the third chunk, and so on. Modify your program to implement this idea; comment on the results.

Answer:

The use of overlapping chunks increases time resolution and produces a smoother spectrogram. Frequency transitions appear more continuous, and short-term variations are better represented. The overall frequency content remains the same, but computation time increases due to the additional DFTs.



Code:

```
clc; clear; close all;
% Signal setup
Fs = 2000; % Sampling frequency (Hz)
n = 0:1499; % Sample index
x = sin(pi/100*(n.*sqrt(n))); % Test signal

% Spectrogram parameters
chunk_size = 66; % Number of samples per DFT
overlap = 0.5; % 50 percent overlap
step = round(chunk_size * (1 - overlap)); % Step size between chunks

num_chunks = floor((length(x) - chunk_size) / step) + 1;
half_spectrum = chunk_size / 2; % Only use the first half of FFT
spectrogram_matrix = zeros(half_spectrum, num_chunks);
```

```
% Compute FFT magnitude for each overlapping chunk
for i = 1:num_chunks
   start_idx = (i-1)*step + 1;
   end_idx = start_idx + chunk_size - 1;
            chunk = x(start_idx:end_idx);
            Y = abs(fft(chunk));
   spectrogram_matrix(:, i) = Y(1:half_spectrum);
end

% Normalize for color display
spectrogram_matrix = spectrogram_matrix / max(spectrogram_matrix(:));

% Define frequency and time axes
freq_axis = linspace(0, Fs/2, half_spectrum);
time_axis = ((0:num_chunks-1) * step) / Fs;

% Plot spectrogram
figure('Name','Spectrogram with Overlapping Chunks');
colormap(jet(256));
imagesc(time_axis, freq_axis, spectrogram_matrix);
axis xy;
xlabel('Time (s)');
ylabel('Frequency (Hz)');
title('Spectrogram with 50% Overlap');
colorbar;
```

4.2.4 Something More Interesting

Now use your spectrogram program from the previous section on the supplied *newvoice.wav* file. This file has a 8 KHz sampling rate, so modify your transform size accordingly (as per 4.1.1). It's important to notice that the simple signals we were working with before were row vectors, but here using *audioread* will create a column vector – you may need to account for this in your code.

You may be disappointed with your initial result, but there are two things you can do to obtain a more interesting result:

1. If you look at the numerical values of your blocks of DFT transforms, you will notice that many values lie between 0 and 1. The *image* command will round these up or down to 0 or 1 respectively, and in doing so, much information which was contained in the original transform vector will be lost (this effect is called *quantization* and is studied in greater detail in another lab). You will also notice that the range of (rounded) values is quite small. This means that we are not utilizing the colour spectrum very efficiently; if only we could assign more colours to represent those low values between 0 and 1, our spectrogram would contain more information. There is an easy way to do this: multiply the original time-domain waveform by a large constant, like 1000. Comment on the effect of this on your spectrogram.

2. Another way to bring out more information is to actually reduce the size of your colour map. The easiest way to understand the effect of this is to repeat the *colormap* and *image* command you tried at the start of this section on page 8. Reduce the size of the colour map by half – how is the `[1:256]` vector mapped to colours? Now apply this colour map reduction to your spectrogram. What happens when you reduce it by a factor of 10? Why? Is there a limit to how much you can reduce it?

After making these improvements, compare your spectrogram to the original time-domain waveform by plotting them one above the other. Can you see the correlation between the two? *Hint:* there is a good reason why we chose to put time on the x-axis.

Finally, run the waterfall spectrogram on the same time-domain signal (be patient, it could take a while!). For the purpose of gleaning as much information as possible from the signal, do you think the waterfall is better than your spectrogram? Why or why not?

Answer:

1. Effect of multiplying the signal:

Scaling the waveform by 1000 increases the magnitude of the DFT, expanding the color range. The spectrogram displays more visible frequency content and low-magnitude components become distinguishable.

2. Effect of reducing the colormap size:

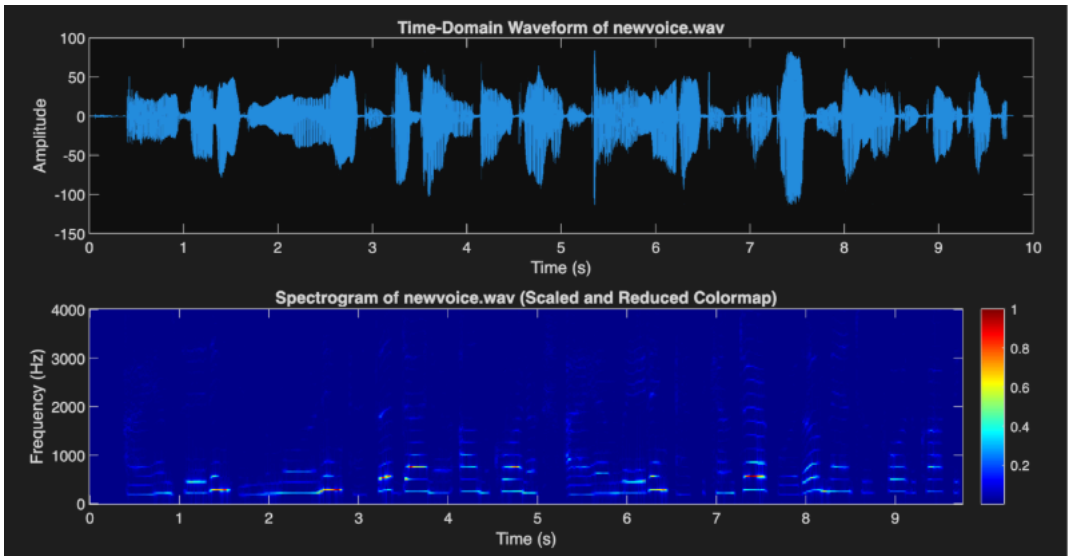
Reducing the colormap compresses the range of colors used to represent magnitudes, increasing contrast between frequency components. Excessive reduction causes color saturation and loss of detail. There is a lower limit where further reduction no longer reveals additional information.

3. Comparison with time-domain waveform:

Peaks in the waveform correspond to high-intensity regions in the spectrogram. The time alignment between the two plots shows how frequency content changes during voiced segments.

4. Comparison with waterfall spectrogram:

Both contain the same information. The 2D color spectrogram conveys time-frequency relationships more clearly and efficiently, while the waterfall plot provides a 3D view but is harder to interpret quantitatively.



Code:

```
clc; clear; close all;

% Read the voice file
[x, Fs] = audioread('newvoice.wav');
x = x(:)';

% Multiply signal to expand color range
x = x * 1000;

% Parameters
chunk_size = 256; % FFT size for 8 kHz sampling rate
overlap = 0.5; % 50% overlap
step = round(chunk_size * (1 - overlap));
num_chunks = floor((length(x) - chunk_size) / step) + 1;
half_spectrum = chunk_size / 2;
spectrogram_matrix = zeros(half_spectrum, num_chunks);

% Compute overlapping DFTs
for i = 1:num_chunks
    start_idx = (i-1)*step + 1;
```

```
    end_idx = start_idx + chunk_size - 1;
            chunk = x(start_idx:end_idx);
            Y = abs(fft(chunk));
    spectrogram_matrix(:, i) = Y(1:half_spectrum);
end

% Normalize for display
spectrogram_matrix = spectrogram_matrix / max(spectrogram_matrix(:));

% Frequency and time axes
freq_axis = linspace(0, Fs/2, half_spectrum);
time_axis = ((0:num_chunks-1) * step) / Fs;

% Plot time-domain waveform and spectrogram
figure;
subplot(2,1,1);
plot((0:length(x)-1)/Fs, x);
xlabel('Time (s)');
ylabel('Amplitude');
title('Time-Domain Waveform of newvoice.wav');

subplot(2,1,2);
colormap(jet(128));    % Reduced color map size
imagesc(time_axis, freq_axis, spectrogram_matrix);
axis xy;
xlabel('Time (s)');
ylabel('Frequency (Hz)');
title('Spectrogram of newvoice.wav (Scaled and Reduced Colormap)');
colorbar;
```