



Department of Computer Science

Lab Manual

of

SPEECH PROCESSING AND RECOGNITION

MAI574

Class Name: V MScAIML

**Master of Science Artificial Intelligence and
Machine Learning**

2025-26

Prepared by: Aleena Ealias

**Verified by:
Faculty name:**

Department Overview

Department of Computer Science of CHRIST (Deemed to be University) strives to shape outstanding computer professionals with ethical and human values to reshape nation's destiny. The training imparted aims to prepare young minds for the challenging opportunities in the IT industry with a global awareness rooted in the Indian soil, nourished and supported by experts in the field.

Vision

The Department of Computer Science endeavours to imbibe the vision of the University “**Excellence and Service**”. The department is committed to this philosophy which pervades every aspect and functioning of the department.

Mission

“To develop IT professionals with ethical and human values”. To accomplish our mission, the department encourages students to apply their acquired knowledge and skills towards professional achievements in their career. The department also moulds the students to be socially responsible and ethically sound.

Introduction to the Programme

Machines are gaining more intelligence to perform human like tasks. Artificial Intelligence has spanned across the world irrespective of domains. MSc (Artificial Intelligence and Machine Learning) will enable to capitalize this wide spectrum of opportunities to the candidates who aspire to master the skill sets with a research bent. The curriculum supports the students to obtain adequate knowledge in the theory of artificial intelligence with hands-on experience in relevant domains with tools and techniques to address the latest demands from the industry. Also, candidates gain exposure to research models and industry standard application development in specialized domains through guest lectures, seminars, industry offered electives, projects, internships, etc.

Programme Objective

- To acquire in-depth understanding of the theoretical concepts in Artificial Intelligence and Machine Learning
- To gain practical experience in programming tools for Data Engineering, Knowledge Representation, Artificial intelligence, Machine learning, Natural Language Processing and Computer Vision.
- To strengthen the research and development of intelligent applications skills through specialization based real time projects.
- To imbibe quality research and develop solutions to the social issues.

Programme Outcomes:

PO1 : Conduct investigation and develop innovative solutions for real world problems in industry and research establishments related to Artificial Intelligence and Machine Learning
PO2 : Apply programming principles and practices for developing automation solutions to meet future business and society needs.

PO3 : Ability to use or develop the right tools to develop high end intelligent systems

PO4 : Adopt professional and ethical practices in Artificial Intelligence application development

PO5 : Understand the importance and the judicious use of technology for the sustainability of the environment.

MAI574– SPEECH PROCESSING AND RECOGNITION

Total Teaching Hours for Semester: 75 (3+4)

Max Marks:150

Credits: 5

Course Objectives

This course enables the learners to understand fundamentals of speech recognition, speech production and representation. It also enables the learners to impart knowledge on automatic speech recognition and pattern comparison techniques. This course helps the learners to develop automatic speech recognition model for different applications.

Course Outcomes

After successful completion of this course students will be able to

CO1: Understand the speech signals and represent the signal in time and frequency domain.

CO2: Analyze different signal processing and speech recognition methods.

CO3: Implement pattern comparison techniques and Hidden Markov Models (HMM)

CO4: Develop speech recognition system for real time problems.

Unit-1

Teaching Hours: 15

FUNDAMENTALS OF SPEECH RECOGNITION

Introduction- The Paradigm for Speech Recognition- Brief History of speech recognition research- The Speech Signal: The process of speech production and perception in human beings- the speech production system- representing speech in time and frequency domain- speech sounds and features.

Lab Programs:

1. Implement the task that takes in the audio as input and converts it to text.
2. Apply Fourier transform and calculate a frequency spectrum for a signal in the time domain.

Unit-2

Teaching Hours: 15

2.1 APPROACHES TO AUTOMATIC SPEECH RECOGNITION BY MACHINE:

The acoustic phonetic approach-The pattern recognition approach-The artificial intelligence approach.

2.2 SIGNAL PROCESSING AND ANALYSIS METHODS FOR SPEECH RECOGNITION:

Introduction- spectral analysis models- the bank of filters front end processor- linear predictive coding model for speech recognition- vector quantization.

Lab Programs:

3. Implement sampling and quantization techniques for the given speech signals.
4. Explore linear predictive coding model for speech recognition

Unit-3**Teaching Hours: 15****PATTERN COMPARISON TECHNIQUES**

Speech detection- distortion measure- Mathematical consideration- Distortion measure – Perceptual consideration- Spectral Distortion Measure- Incorporation of spectral dynamic feature into distortion measure- Time alignment and normalization.

Lab Programs:

0. Demonstrate different pattern in the given speech signal
0. Implement time alignment and normalization techniques

Unit-4**Teaching hours: 15****THEORY AND IMPLEMENTATION OF HIDDEN MARKOV MODELS:**

Introduction- Discrete time Markov processes- Extension to hidden Markov Models- Coin -toss models- The urn and ball model- Elements of a Hidden Markov Model- HMM generator of observation- The three basic problems for HMM's- The Viterbi algorithm- Implementation issues for HMM's.

Lab Programs:

7. Implement simple hidden Markov Model for a particular application.
8. Apply Viterbi dynamic programming algorithm to find the most likely sequence of hidden states.

Unit-5**Teaching Hours: 15****TASK ORIENTED APPLICATION OF AUTOMATIC SPEECH RECOGNITION:**

Task specific voice control and dialog- Characteristics of speech recognition applications- Methods of handling recognition error- Broad classes of speech recognition applications- Command and control applications- Voice repertory dialer- Automated call-type recognition- Call distribution by voice commands- Directory listing retrieval- Credit card sales validation.

Lab Programs:

9. Demonstrate automatic speech recognition for Call distribution by voice commands.
10. Apply speech recognition system to access telephone directory information from spoken spelled names (Directory listing retrieval).

Text Books and Reference Books

- [1] Fundamentals of Speech Recognition, Lawrence R Rabiner and Biing- Hwang Juang. Prentice-Hall Publications, 20209.
- [2] Introduction to Digital Speech Processing, Lawrence R. Rabiner, Ronald W. Schafer, Now Publishers, 2015.

Essential Reading / Recommended Reading

- [1] Intelligent Speech Signal Processing, Nilanjan Dey, Academic Press, 2019.

[2] Speech Recognition-The Ultimate Step-By-Step Guide, Gerardus Blokdyk, 5STARCook, 2021.

[3] Automatic Speech Recognition- A Deep Learning Approach, Dong Yu, Li Deng, Springer-Verlag London, 2015.

Web Resources:

1. www.w3schools.com
2. <https://www.simplilearn.com/tutorials/python-tutorial/speech-recognition-in-python>
3. <https://realpython.com/python-speech-recognition/>
4. <https://cloud.google.com/speech-to-text/docs/tutorials>
5. <https://www.coursera.org/courses?query=speech%20recognition>
6. <https://pylessons.com/speech-recognition>

CO – PO Mapping

	PO1	PO2	PO3	PO4	PO5
CO1	3	1			1
CO2	2	2			1
CO3	1	3	1		2
CO4	1	1	3	1	3

LIST OF PROGRAMS

MCA 2023-2024

Sl. no	Title of lab Experiment	Page number	RB T	CO
1	Sampling and Reconstruction of Speech Signals	7	L3	CO1, 3
2	Fourier Transform and Frequency Spectrum Analysis of Signals	21	L3	CO2, 3
3	Speech-to-Text Application for Accessibility	35	L3	CO3
4			L3	CO3
5			L3	CO3
6			L3	CO3, 4
7			L4	CO3
8			L3	CO3
9			L3	CO4
10			L5	CO4

Evaluation Rubrics:

- (1) Implementation: 5 marks.
- (2) Complexity and Validation: 3 marks.
- (3) Documentation & Writing the inference: 2 marks.

Submission Guidelines:

- Make a copy of the lab manual template with your <name_reg:no_subject name >.
- Copy the given question and the answer (lab code) with results, followed by the conclusion of that lab. Title the lab as lab number.
- Keep updating your lab manual and show the lab manual of that particular lab for evaluation.

- Create a Git Repository in your profile <SPR lab-reg no> . Follow a different branch for each lab <Lab 1, Lab 2...>, and push the code to Git. The link should be provided in Google Classroom along with the PDF of the lab manual.
- Upload the PDF to Google Classroom before the deadline.

Lab 1

Lab Exercise I: Sampling and Reconstruction of Speech Signals

Aim

To study sampling and reconstruction of speech signals at different sampling rates, evaluate reconstruction using zero-order hold and linear interpolation, and implement the source-filter model to analyze the effect of filtering, sampling, and reconstruction on speech quality.

(1) Implement sampling and quantization techniques for the given speech signals.

- Plot the time domain representation of the original speech signal.
- Sample the speech signal at different sampling rates (e.g., 8kHz, 16kHz, and 44.1kHz).
- Plot sampled speech signal for each of these sampling rates.
- Using the sampled signals from above, reconstruct the signal using:
 - Zero-order hold (nearest-neighbor interpolation)
 - Linear interpolation.
- Calculate the Mean Squared Error (MSE) between the original and the reconstructed signals for both methods.

Write an inference on how sampling rates affect the quality and accuracy of the reconstructed speech signal.

(2) Implement the source-filter model for a given speech signal and analyze the impact of sampling and reconstruction on the quality of the speech signal.

- Generate a synthetic speech signal using the source-filter model.
 - Create a source signal (e.g., a glottal pulse train for voiced sounds or white noise for unvoiced sounds).
 - Apply a filter that models the vocal tract, represented by an all-pole filter or an FIR filter

with formants (resonances of the vocal tract).

(b) Plot the generated speech signal and analyze the effect of the filter on the original source.

(c) Sample the speech signal generated above at different sampling rates (e.g., 8 kHz, 16 kHz, 44.1 kHz).

(d) Reconstruct the signal using a suitable interpolation method (e.g., zero-order hold, linear interpolation).

(e) Compute the Mean Squared Error (MSE) between the original and reconstructed speech signals.

Write an inference on tasks such as creating the source-filter model, different sampling rates, and reconstruction of the sampled signals.

Code with Results

1) Sampling and Quantization

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import scipy.io.wavfile as wav
```

```
from scipy.signal import resample
```

```
# 1(a) Load and normalize original speech signal
```

```
sr, signal = wav.read("harvard.wav")
```

```
if signal.ndim > 1: # if stereo, take mean
```

```
    signal = np.mean(signal, axis=1)
```

```
signal = signal.astype(np.float32) / np.max(np.abs(signal))
```



```
# Plot original signal

plt.figure(figsize=(12,4))

plt.plot(signal[:2000])

plt.title("Original Speech Signal (Time Domain)")

plt.xlabel("Sample Index")

plt.ylabel("Amplitude")

plt.show()

sampling_rates = [8000, 16000, 44100]

sampled_signals = {}

for fs in sampling_rates:

    num_samples = int(len(signal) * fs / sr)

    sampled_signals[fs] = resample(signal, num_samples)

for fs in sampling_rates:

    plt.figure(figsize=(12,4))

    plt.plot(sampled_signals[fs][:2000])

    plt.title(f"Sampled Speech Signal at {fs} Hz")

    plt.xlabel("Sample Index")

    plt.ylabel("Amplitude")

    plt.show()
```

```
for fs in sampling_rates:
```

```
    plt.figure(figsize=(12,4))
```

```
    plt.plot(sampled_signals[fs][:2000])
```

```
    plt.title(f"Sampled Speech Signal at {fs} Hz")
```

```
    plt.xlabel("Sample Index")
```

```
    plt.ylabel("Amplitude")
```

```
    plt.show()
```

```
def reconstruct(signal, fs, method="zoh"):
```

```
    t_original = np.linspace(0, len(signal)/sr, len(signal))
```

```
    t_new = np.linspace(0, len(signal)/sr, int(len(signal)*fs/sr))
```

```
    sampled = resample(signal, len(t_new))
```

```
    if method == "zoh": # Zero-order hold
```

```
        reconstructed = np.interp(t_original, t_new, sampled, left=0, right=0)
```

```
    elif method == "linear": # Linear interpolation
```

```
        reconstructed = np.interp(t_original, t_new, sampled)
```

```
    return reconstructed
```

```
#Reconstruct and plot
```

```
mse_results = {}
```

```
for fs in sampling_rates:
```

```
    for method in ["zoh", "linear"]:
```

```
        rec = reconstruct(signal, fs, method)
```

```
        mse = np.mean((signal - rec)**2)
```

```
        mse_results[(fs, method)] = mse
```

```
    plt.figure(figsize=(12,4))
```

```
    plt.plot(signal[:2000], label="Original")
```

```
    plt.plot(rec[:2000], label=f"Reconstructed ({method}, {fs}Hz)", alpha=0.7)
```

```
    plt.legend()
```

```
    plt.title(f"Reconstruction at {fs} Hz using {method.upper()}")
```

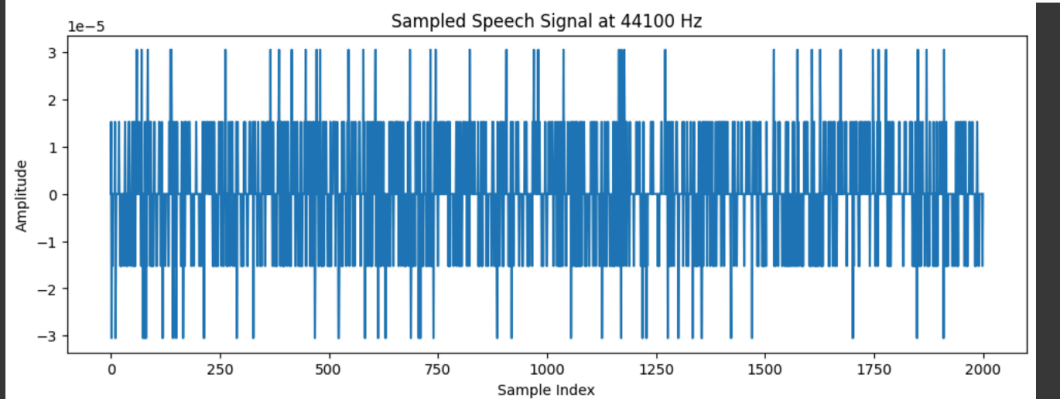
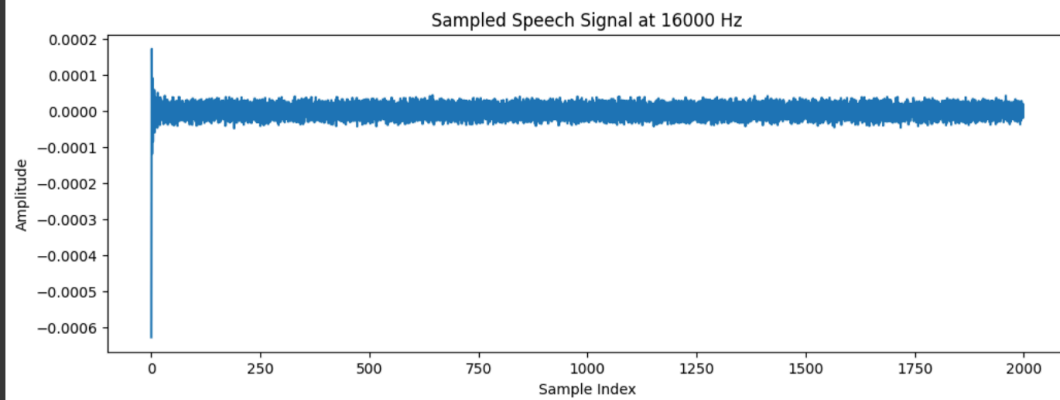
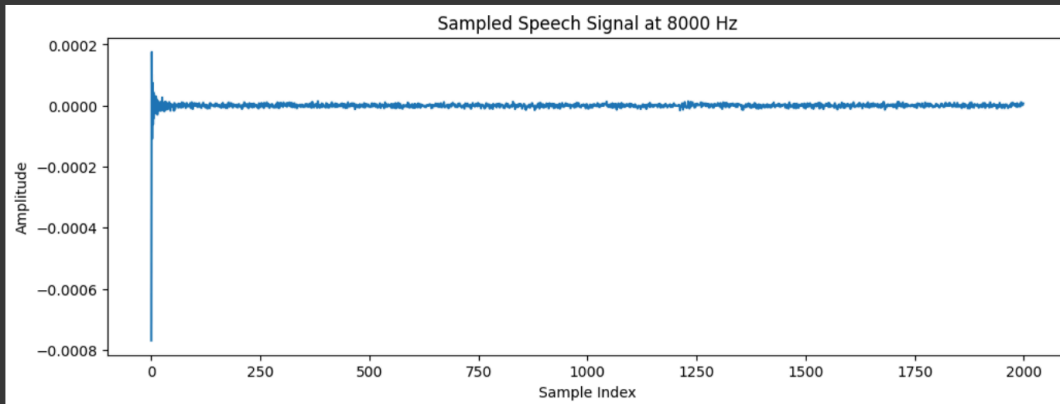
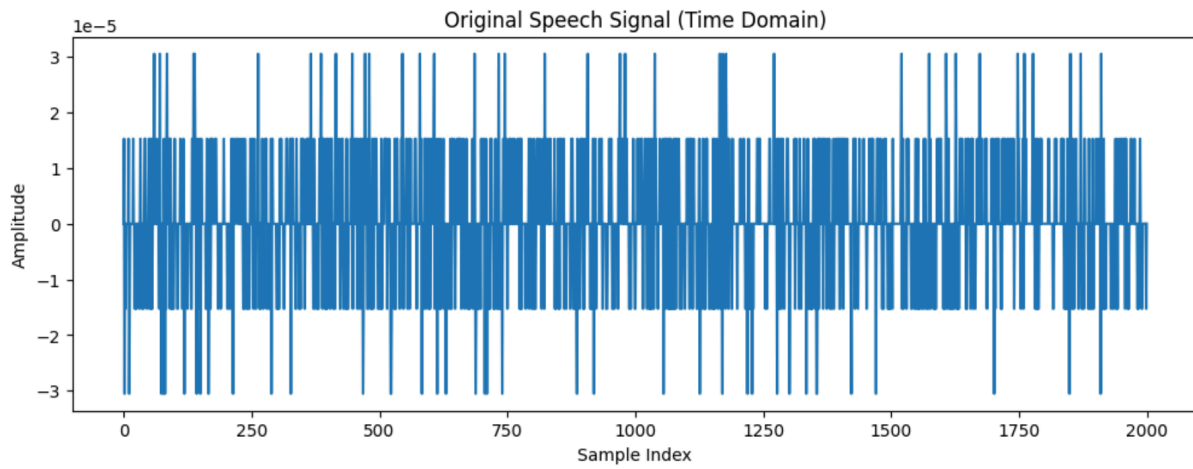
```
    plt.show()
```

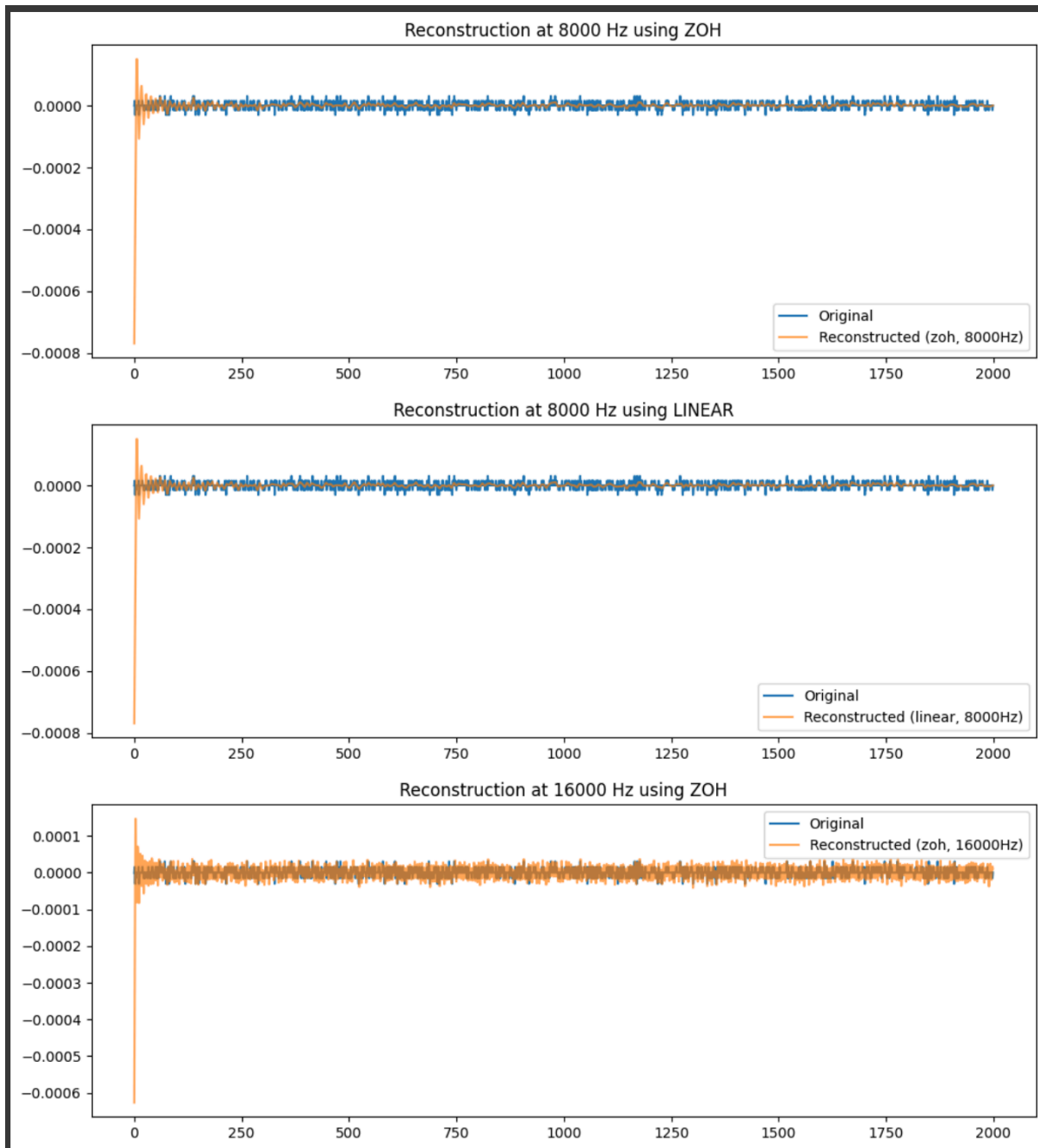
```
print("Mean Squared Error (MSE):")
```

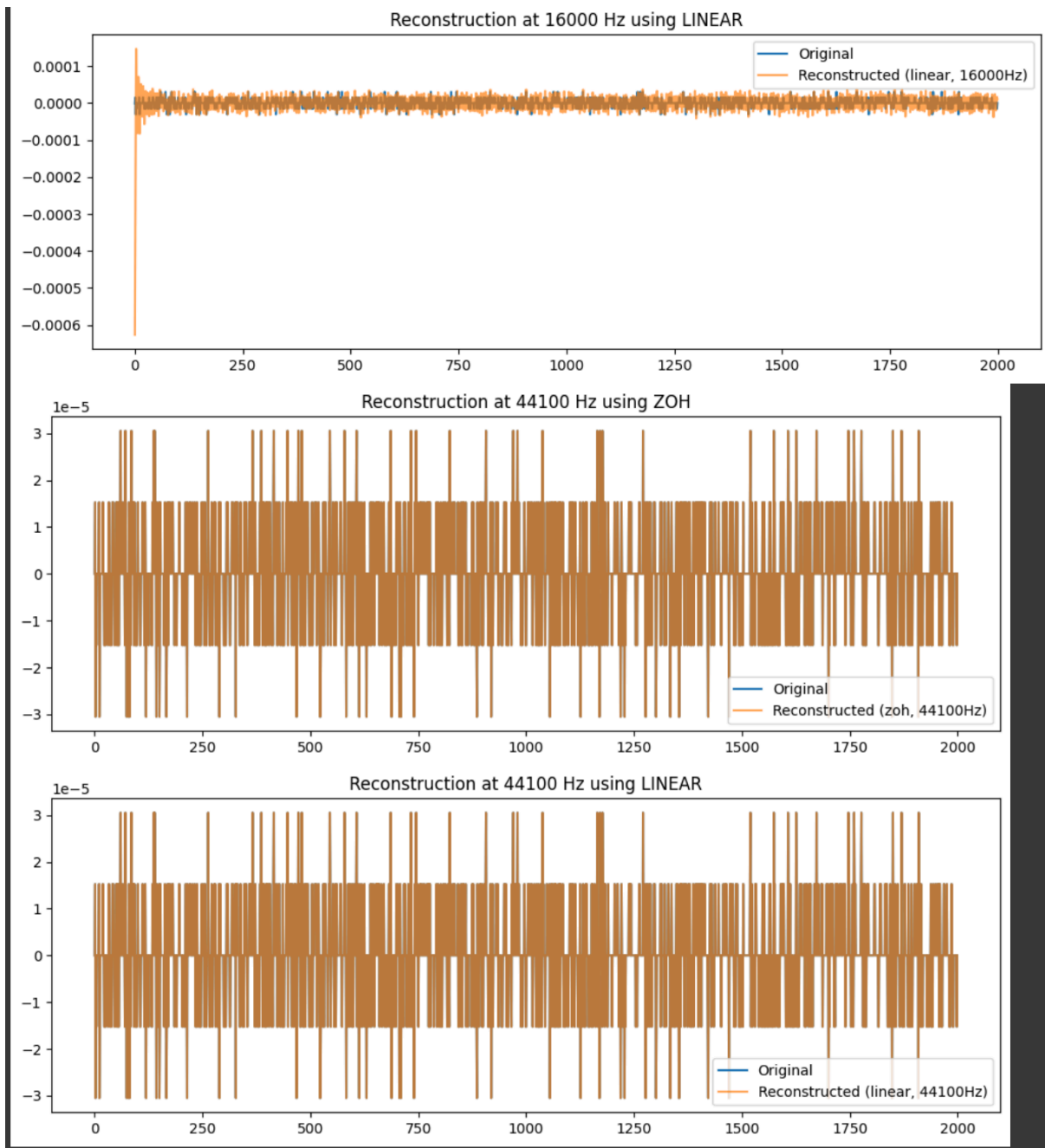
```
for k,v in mse_results.items():
```

```
    print(f"Sampling Rate: {k[0]} Hz, Method: {k[1]} -> MSE: {v:.6f}")
```

Results:







The Mean Squared Error (MSE) values for the different reconstruction methods and sampling rates are:

Mean Squared Error (MSE):

- Sampling Rate: 8000 Hz, Method: zoh -> MSE: 0.000157
- Sampling Rate: 8000 Hz, Method: linear -> MSE: 0.000157
- Sampling Rate: 16000 Hz, Method: zoh -> MSE: 0.000047

- Sampling Rate: 16000 Hz, Method: linear -> MSE: 0.000047
- Sampling Rate: 44100 Hz, Method: zoh -> MSE: 0.000000
- Sampling Rate: 44100 Hz, Method: linear -> MSE: 0.000000

Conclusion:

- Lower sampling rates (e.g., 8 kHz) lose high-frequency details, making speech sound muffled and less accurate.
- Higher rates (44.1 kHz) preserve more details and yield lower reconstruction error.
- At lower rates (8 kHz, 16 kHz), Linear interpolation performs slightly better (smoother curve, less step-like distortion).
- MSE decreases as sampling rate increases, indicating better approximation of the original speech.

(2) Source-Filter Model

```
from scipy.signal import lfilter
```

```
# 2(a) Generate synthetic speech (1 second)
```

```
duration = 1.0
```

```
t = np.linspace(0, duration, int(sr*duration), endpoint=False)
```

```
# Voiced source: glottal pulse train
```

```
f0 = 100 # Hz
```

```
source_voiced = (np.sin(2*np.pi*f0*t) > 0).astype(float)
```

```
# Unvoiced source: white noise
```

```
source_unvoiced = np.random.randn(len(t))
```

```
# Vocal tract filter (formants 500, 1500, 2500 Hz)

formants = [500, 1500, 2500]

B = [1.0]

A = [1.0]

for f in formants:

    r = 0.95

    theta = 2*np.pi*f/sr

    A = np.convolve(A, [1, -2*r*np.cos(theta), r**2])

# Filter the source signals

speech_voiced = lfilter(B, A, source_voiced)

speech_unvoiced = lfilter(B, A, source_unvoiced)

plt.figure(figsize=(12,4))

plt.plot(speech_voiced[:2000])

plt.title("Voiced Speech Signal (Source-Filter Model)")

plt.xlabel("Sample Index")

plt.ylabel("Amplitude")

plt.show()

plt.figure(figsize=(12,4))

plt.plot(speech_unvoiced[:2000])
```



```
plt.title("Unvoiced Speech Signal (Source-Filter Model)")

plt.xlabel("Sample Index")

plt.ylabel("Amplitude")

plt.show()

sampled_speech = {}

for fs in sampling_rates:

    num_samples = int(len(speech_voiced) * fs / sr)

    sampled_speech[fs] = resample(speech_voiced, num_samples)

mse_speech = {}

for fs in sampling_rates:

    rec = reconstruct(speech_voiced, fs, method="linear")

    mse = np.mean((speech_voiced - rec)**2)

    mse_speech[fs] = mse


plt.figure(figsize=(12,4))

plt.plot(speech_voiced[:2000], label="Original")

plt.plot(rec[:2000], label=f"Reconstructed ({fs}Hz)", alpha=0.7)

plt.legend()

plt.title(f"Synthetic Speech Reconstruction at {fs} Hz")

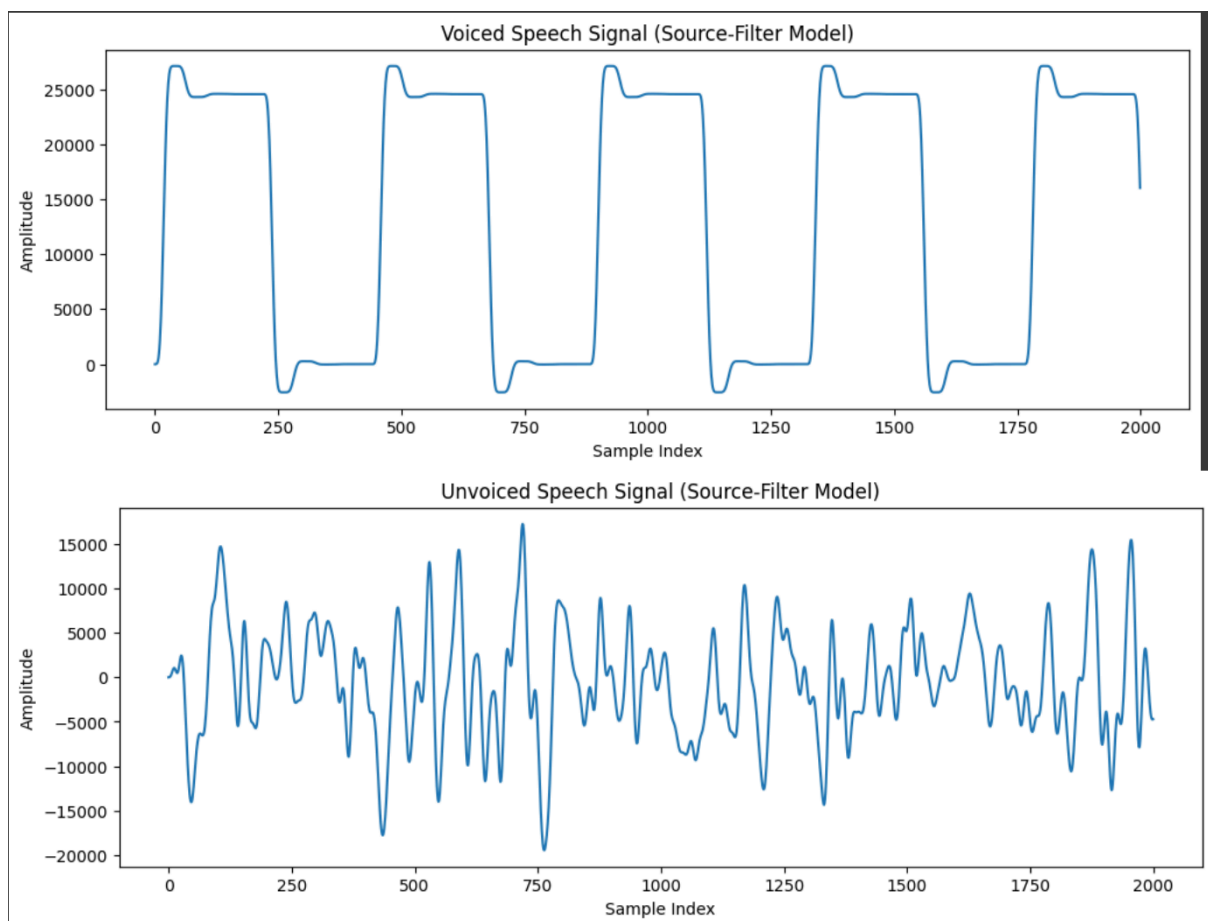
plt.show()
```

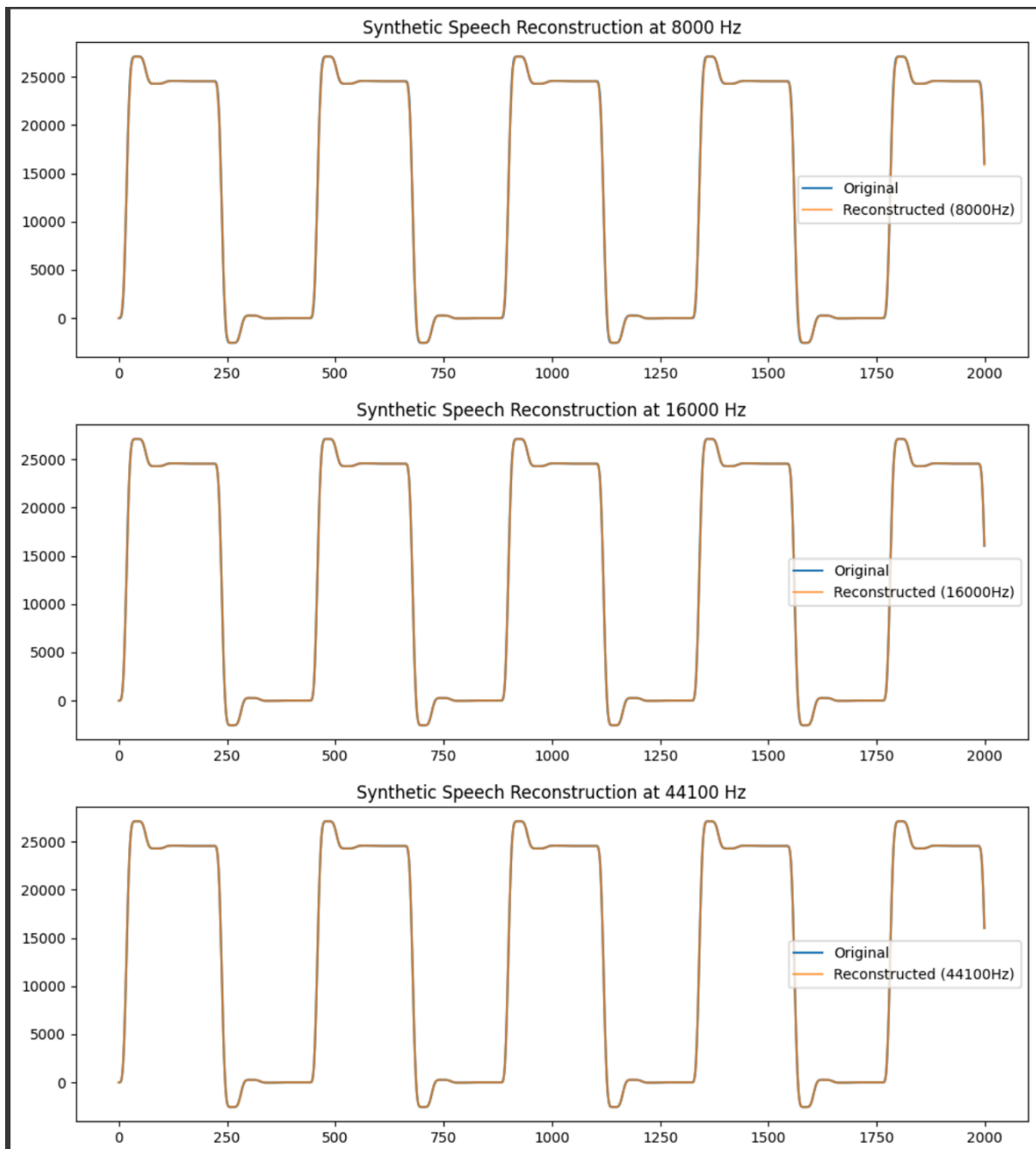
```
print("MSE for Synthetic Speech Reconstruction:")
```

```
for fs, mse in mse_speech.items():
```

```
    print(f"Sampling Rate: {fs} Hz -> MSE: {mse:.6f}")
```

Results:





The MSE values for the reconstructed synthetic speech signals are:

- **MSE for Synthetic Speech Reconstruction:**
- **Sampling Rate: 8000 Hz -> MSE: 1081285.100893**
- **Sampling Rate: 16000 Hz -> MSE: 168555.986217**
- **Sampling Rate: 44100 Hz -> MSE: 0.000000**

Conclusion:

- **The source-filter model demonstrates how vocal tract filtering shapes the source waveform into speech-like signals.**
- **Higher sampling rates capture more formant details and reduce reconstruction error.**
- **Reconstruction at lower rates loses some resonance characteristics, making speech sound less natural.**
- **Linear interpolation works well for synthetic signals; MSE decreases with higher sampling rate.**

Lab Exercise 2:

Fourier Transform and Frequency Spectrum Analysis of Signals

Aim

To study the Fourier Transform and analyze the frequency spectrum of different signals (sinusoidal, composite, exponential, and rectangular). To compare their time-domain representation with their frequency-domain characteristics using both the Discrete-Time Fourier Transform (DTFT) and the Discrete Fourier Transform (DFT).

Questions

Question 1

- (a) Generate a basic sinusoidal signal in the time domain. (For example, generate a sine wave with a frequency of 5 Hz, sampled at 1000 Hz.)
- (b) Plot the time-domain waveform of the signal.
- (c) Compute the Discrete-Time Fourier Transform (DTFT) and plot the continuous frequency spectrum.
- (d) Compute the Discrete Fourier Transform (DFT) and plot the discrete frequency spectrum.

Question 2

- (a) Generate a composite signal by adding two or more sinusoidal signals of different frequencies and amplitudes.
- (b) Plot the time-domain waveform of the composite signal.
- (c) Compute the Discrete-Time Fourier Transform (DTFT) and plot the continuous frequency spectrum.
- (d) Compute the Discrete Fourier Transform (DFT) and plot the discrete frequency spectrum.

Question 3

- (a) Generate an exponentially decaying signal.
- (b) Plot the time-domain waveform.
- (c) Compute the Discrete-Time Fourier Transform (DTFT) and plot the continuous frequency spectrum.
- (d) Compute the Discrete Fourier Transform (DFT) and plot the discrete frequency spectrum.
- (e) Analyze the relationship between the time-domain waveform and the frequency-domain representation.

Question 4

- (a) Generate a **rectangular pulse signal** of finite duration in the time domain.
- (b) Plot the time-domain waveform.
- (c) Compute the Discrete-Time Fourier Transform (DTFT) and plot the continuous frequency spectrum.
- (d) Compute the Discrete Fourier Transform (DFT) and plot the discrete frequency spectrum.

(e) Analyze the relationship between the time-domain waveform and the frequency-domain representation.

Evaluation Rubrics

1. **Implementation:** 5 marks
2. **Complexity and Validation:** 3 marks
3. **Documentation & Writing the inference:** 2 marks

Submission Guidelines

- Prepare a **single PDF** containing answers for all questions with proper plots, results, and inferences.
- Each answer must include **Python/Matlab code, output plots, and a conclusion.**
- Title the file as: **Lab_2_<YourName_RegNo>.**
- Upload the PDF to **Google Classroom** before the deadline.
- Maintain a **Git repository** with separate branches for each lab (Lab1, Lab2, ...). Push your code and include the repo link in your submission.

Code with Results

1.

```
import numpy as np

import matplotlib.pyplot as plt

# Parameters

fs = 1000 # Sampling frequency in Hz

f = 5 # Signal frequency in Hz

t = np.arange(0, 1, 1/fs) # 1 second duration

x = np.sin(2*np.pi*f*t)

plt.figure(figsize=(10,4))

plt.plot(t, x)

plt.title("Time-domain: Sinusoidal Signal")

plt.xlabel("Time [s]")

plt.ylabel("Amplitude")

plt.grid(True)

plt.show()

n_fft = 2048

X_dtft = np.fft.fft(x, n_fft)

freqs = np.fft.fftfreq(n_fft, 1/fs)

plt.figure(figsize=(10,4))

plt.plot(np.fft.fftshift(freqs), np.fft.fftshift(np.abs(X_dtft)))

plt.title("DTFT (Approx.) of Sinusoidal Signal")

plt.xlabel("Frequency [Hz]")

plt.ylabel("Magnitude")
```

```
plt.grid(True)

plt.show()

X_dft = np.fft.fft(x)

freqs_dft = np.fft.fftfreq(len(x), 1/fs)

plt.figure(figsize=(10,4))

plt.stem(freqs_dft, np.abs(X_dft))

plt.title("DFT of Signal")

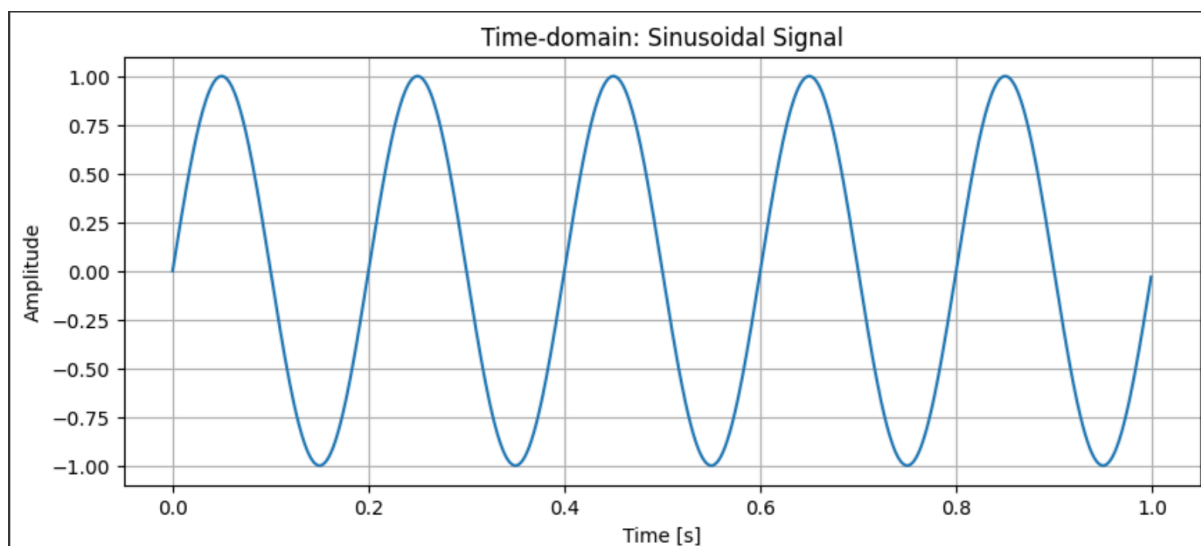
plt.xlabel("Frequency [Hz]")

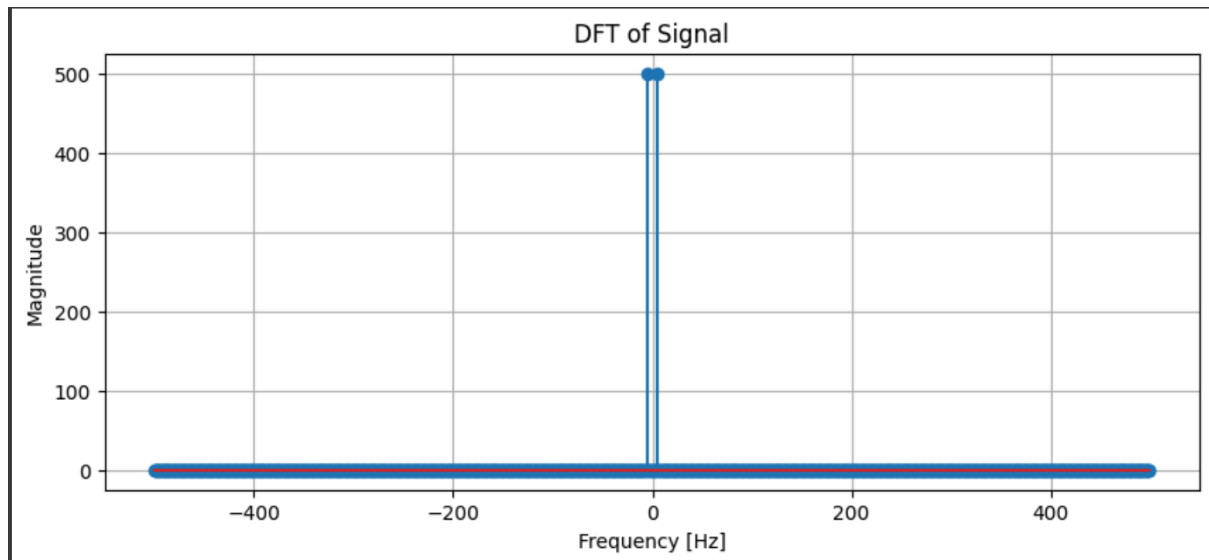
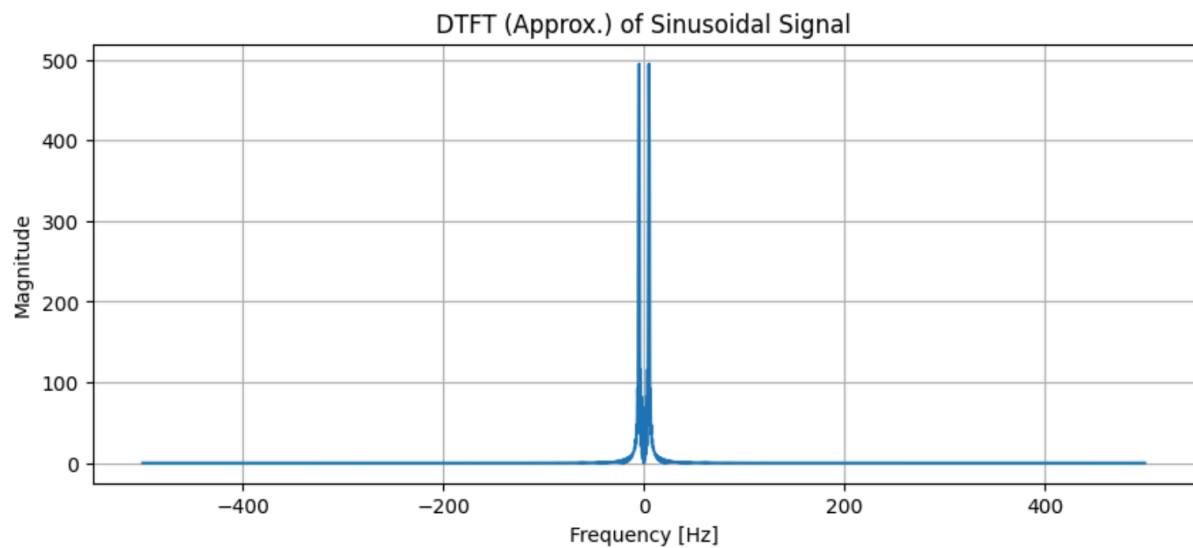
plt.ylabel("Magnitude")

plt.grid(True)

plt.show()
```

Results:





Conclusion:

- Sinusoidal signal with a frequency of 5 Hz and a sampling rate of 1000 Hz. The time-domain plot accurately reflects this, showing 5 complete cycles over a duration of 1 second with an amplitude of 1.0.
- The resulting frequency plots show a peak magnitude of 500.
- Signal is a pure 5 Hz sine wave. Both the DTFT and DFT plots accurately represent this by showing a single, sharp peak at 5 Hz.

2.

```
x1 = 1.0*np.sin(2*np.pi*5*t)
x2 = 0.5*np.sin(2*np.pi*20*t)
x_comp = x1 + x2

plt.figure(figsize=(10,4))

plt.plot(t, x_comp)

plt.title("Time-domain: Composite Signal")

plt.xlabel("Time [s]")

plt.ylabel("Amplitude")

plt.grid(True)

plt.show()

X_dtft = np.fft.fft(x_comp, n_fft)

plt.figure(figsize=(10,4))

plt.plot(np.fft.fftshift(freqs), np.fft.fftshift(np.abs(X_dtft)))

plt.title("DTFT of Composite Signal")

plt.xlabel("Frequency [Hz]")

plt.ylabel("Magnitude")

plt.grid(True)

plt.show()

X_dft = np.fft.fft(x_comp)

plt.figure(figsize=(10,4))

plt.stem(freqs_dft, np.abs(X_dft))

plt.title("DFT of Composite Signal")

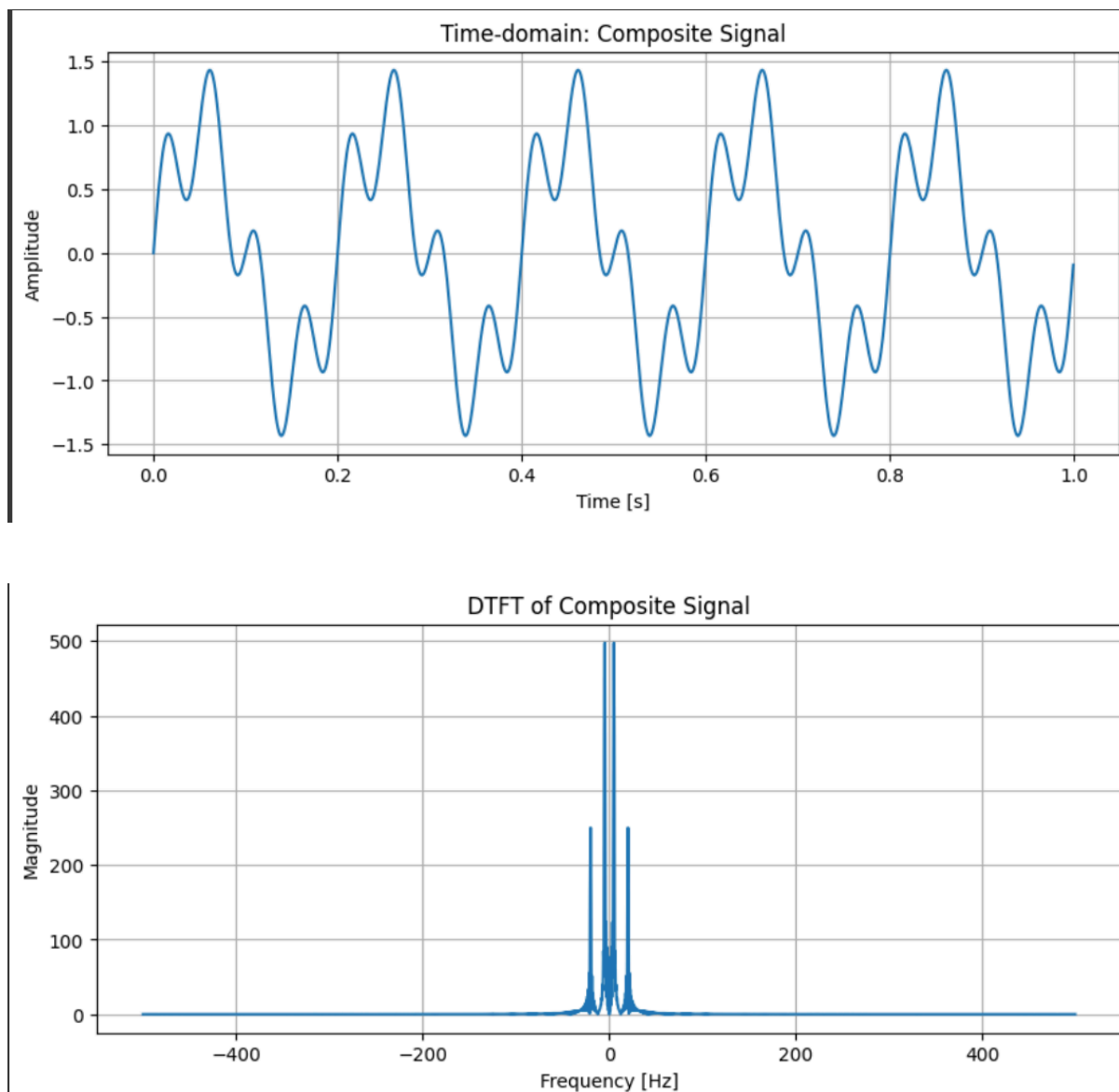
plt.xlabel("Frequency [Hz]")
```

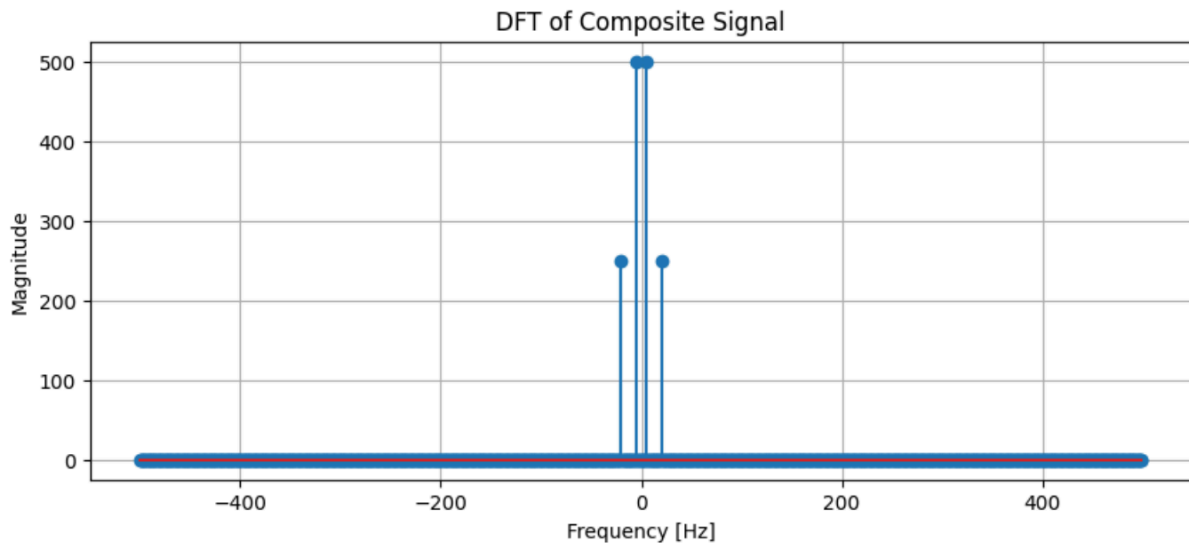
```
plt.ylabel("Magnitude")
```

```
plt.grid(True)
```

```
plt.show()
```

Results:





Conclusion:

- Composite signal by summing two sine waves: one at 5 Hz with an amplitude of 1.0, and another at 20 Hz with an amplitude of 0.5.
- The peaks corresponding to the 5 Hz component have a magnitude of approximately 500 and the peaks for the 20 Hz component have a magnitude of approximately 250.
- The frequency-domain plots show peaks at 5 Hz and 20 Hz, with magnitudes proportional to the amplitudes of the original sine waves.

3.

`tau = 0.2 # decay constant`

`x_exp = np.exp(-t/tau)`

`plt.figure(figsize=(10,4))`

`plt.plot(t, x_exp)`

`plt.title("Time-domain: Exponential Decay")`

```
plt.xlabel("Time [s]")
```

```
plt.ylabel("Amplitude")
```

```
plt.grid(True)
```

```
plt.show()
```

```
X_dtft = np.fft.fft(x_exp, n_fft)
```

```
plt.figure(figsize=(10,4))
```

```
plt.plot(np.fft.fftshift(freqs), np.fft.fftshift(np.abs(X_dtft)))
```

```
plt.title("DTFT of Exponential Signal")
```

```
plt.xlabel("Frequency [Hz]")
```

```
plt.ylabel("Magnitude")
```

```
plt.grid(True)
```

```
plt.show()
```

```
X_dft = np.fft.fft(x_exp)
```

```
plt.figure(figsize=(10,4))
```

```
plt.stem(freqs_dft, np.abs(X_dft))
```

```
plt.title("DFT of Exponential Signal")
```

```
plt.xlabel("Frequency [Hz]")
```

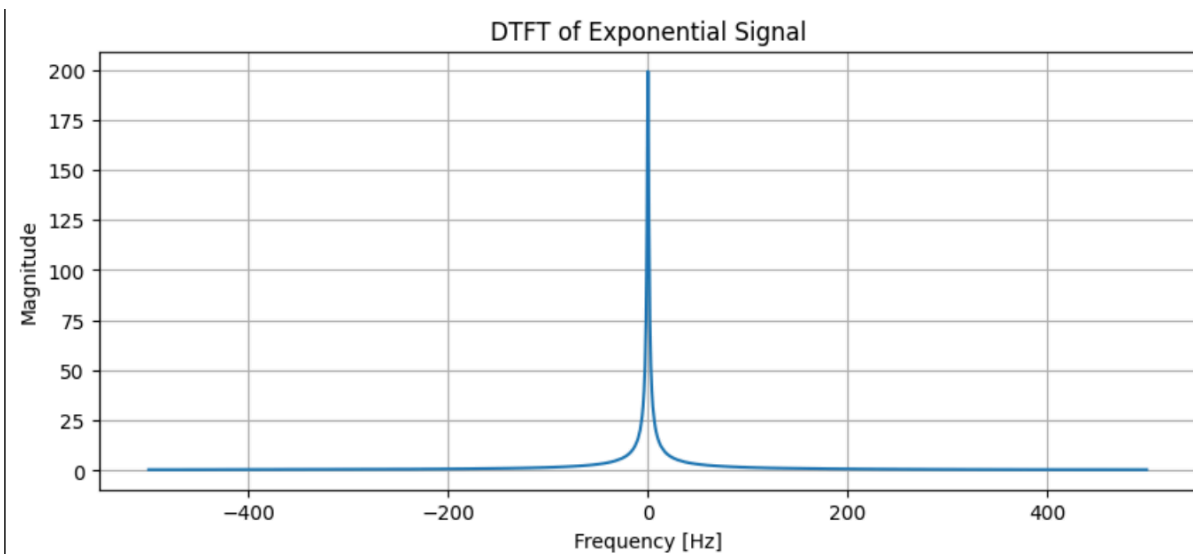
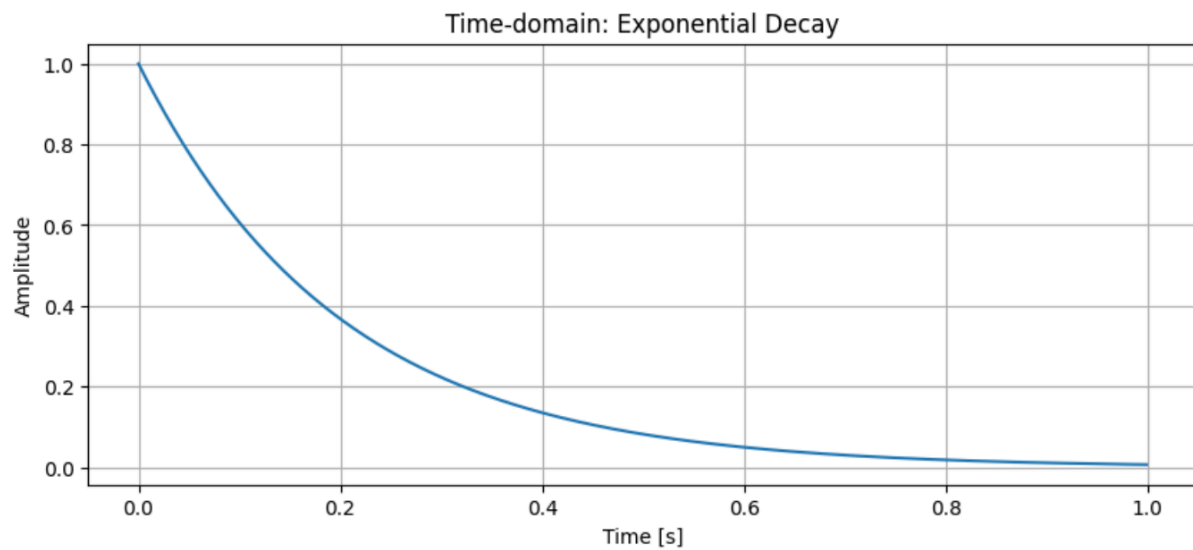
```
plt.ylabel("Magnitude")
```

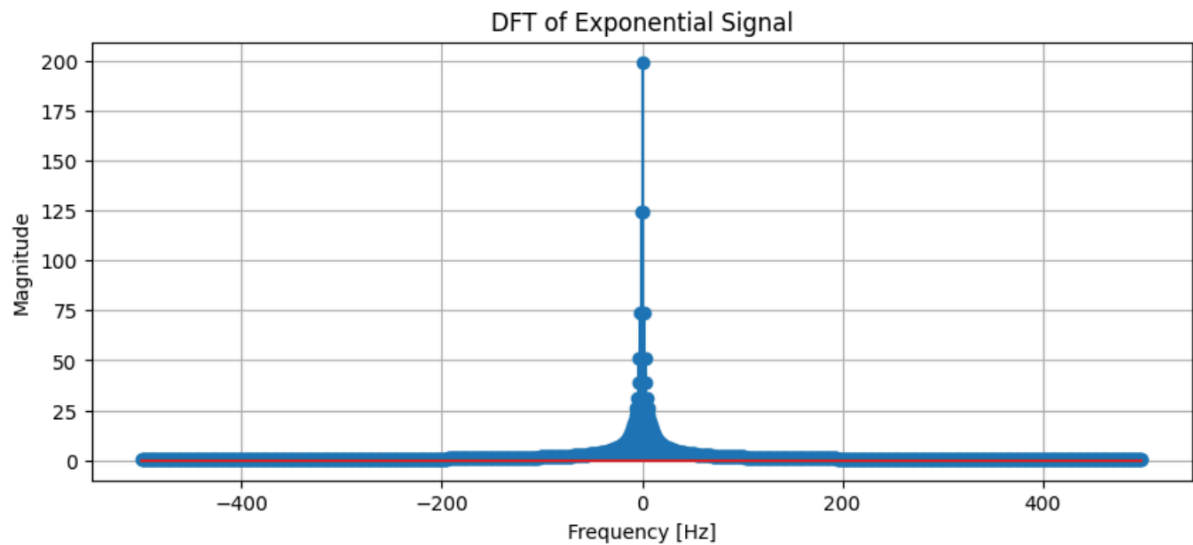
```
plt.grid(True)
```

```
plt.show()
```

A sharp, non-periodic event in the time domain and the start of an exponential decay, corresponds to a wide and continuous spectrum in the frequency domain.

Results:





Conclusion:

- The time-domain plot correctly shows the signal starting at an amplitude of 1.0 and decaying towards zero. The frequency-domain plots for the DTFT and DFT show that the signal's energy is concentrated at lower frequencies, with a peak at 0 Hz, and spreads broadly across the spectrum.

4.

```
duration = 0.1 # pulse duration in seconds
```

```
x_rect = np.where(t <= duration, 1, 0)
```

```
plt.figure(figsize=(10,4))
```

```
plt.plot(t, x_rect)
```

```
plt.title("Time-domain: Rectangular Pulse")
```

```
plt.xlabel("Time [s]")
```

```
plt.ylabel("Amplitude")
```

```
plt.grid(True)
```

```
plt.show()
```

```
X_dtft = np.fft.fft(x_rect, n_fft)
```

```
plt.figure(figsize=(10,4))
```

```
plt.plot(np.fft.fftshift(freqs), np.fft.fftshift(np.abs(X_dtft)))
```

```
plt.title("DTFT of Rectangular Pulse")
```

```
plt.xlabel("Frequency [Hz]")
```

```
plt.ylabel("Magnitude")
```

```
plt.grid(True)
```

```
plt.show()
```

```
X_dft = np.fft.fft(x_rect)
```

```
plt.figure(figsize=(10,4))
```

```
plt.stem(freqs_dft, np.abs(X_dft))
```

```
plt.title("DFT of Rectangular Pulse")
```

```
plt.xlabel("Frequency [Hz]")
```

```
plt.ylabel("Magnitude")
```

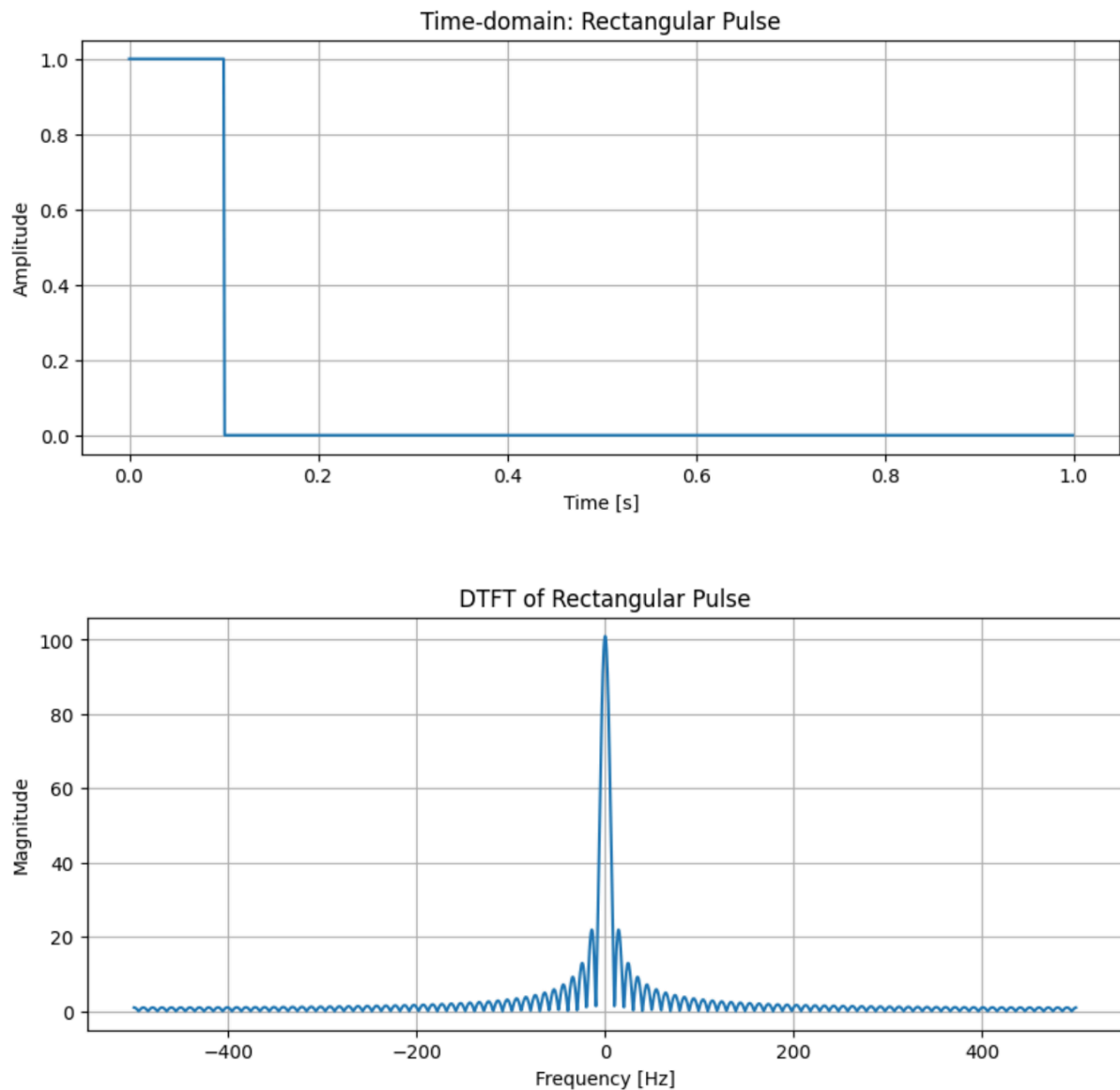
```
plt.grid(True)
```

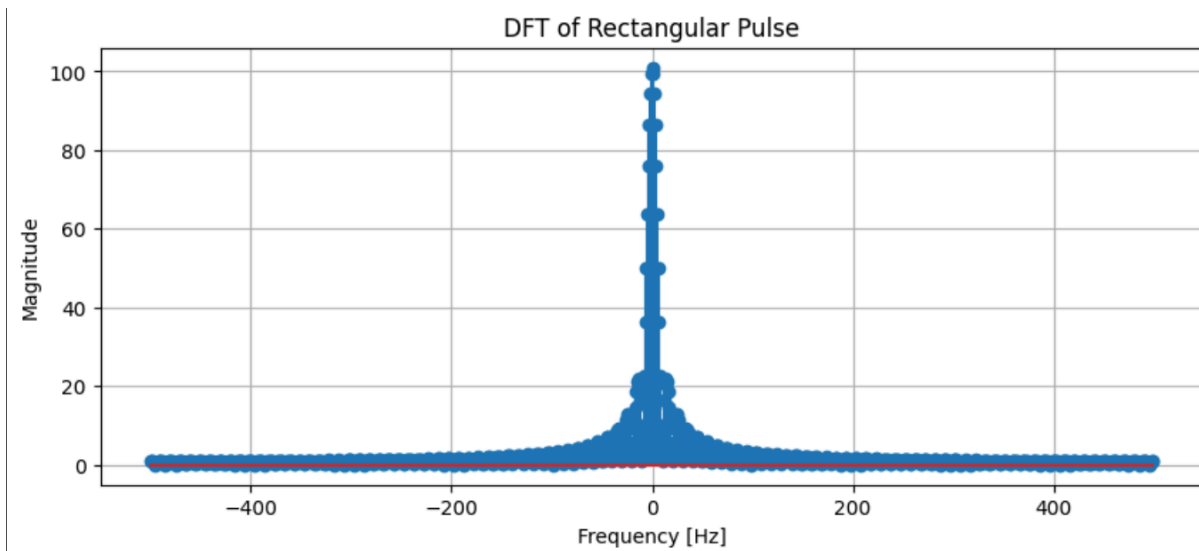
```
plt.show()
```

A rectangular pulse in the time domain is transformed into a sinc function in the frequency domain. The sharp discontinuities (the vertical edges) of the pulse in time

require a wide range of frequencies to be represented, hence the spectrum extends indefinitely with side lobes.

Results:



**Conclusion:**

- The DTFT and DFT plots both show characteristic sinc shape, with a large main lobe centered at 0 Hz and smaller, decaying side lobes extending across the frequency spectrum.

Lab Exercise 3:

Speech-to-Text Application for Accessibility

Aim

To develop a Python-based speech-to-text system that converts spoken commands into text in real time, provides meaningful user feedback, handles errors gracefully, and allows comparison of different recognition methods.

Questions

You have been hired as an AI engineer by a tech startup that focuses on enhancing accessibility for people with disabilities. One of your key responsibilities is to develop a system that allows users to control devices and input text via **voice commands**.

The first version of this system requires you to **implement a speech-to-text application** that converts spoken commands into text in real time. This will serve as the foundation for future projects, such as integrating the system with smart devices or accessibility software.

Tasks:

1. **Audio Capture**(*mandatory task*)
 - Record spoken input using a **microphone**, OR use any speech audio file (e.g., **.wav**, **.flac**).
https://drive.google.com/file/d/1BmlRHKnHWVtIM743vcjLGtaK_aRzd_Qa/view?usp=drive_link
 - Provide feedback to the user: "Speak something..."
2. **Convert Speech to Text**(*mandatory task*)
 - Implement a speech-to-text system using **at least two methods** (for comparison):
 - Offline: **Whisper**, **Vosk**, or similar
 - Online: **Google Speech API**
 - Display the message: "Recognizing..." while processing.
3. **Display Recognized Text**(*mandatory task*)
 - Show the converted text on the screen.
Example: "Speech recognized: 'Turn on the lights in the living room.'"
 - Display "Speech successfully converted to text!" on successful recognition.
4. **Handle Errors and Exceptions**(*mandatory task*)
 - **Unclear speech** (mumbling, low volume): Display a user-friendly message.
Example: "Speech Recognition could not understand audio. Please try speaking more clearly."
 - **Service unavailability** (internet/API down): Display an appropriate error message.

5. Provide Feedback at Each Stage

- Before recording: "Speak something..."
- During recognition: "Recognizing..."
- On success: "Speech successfully converted to text!"
- On failure: Provide meaningful error messages.

6. Comparative Analysis

- Test the same audio file or spoken sentence using **multiple recognition methods**.
- Fill in the **comparison table**:

Audio Type	Whisper Output	Vosk Output	Google API Output	Any other python libraries can be added ..	Notes on Accuracy
Clear male voice					
Clear female voice					
Fast speech					
Noisy background					
Soft voice					

7.

Write a Brief Inference

- Summarize your observations about the system's performance:
 - How accurately does it recognize speech?
 - How well does it handle errors?
 - Which method performed best for each scenario?
 - Suggestions for future improvements or project extensions.

Deliverables

1. Python code implementing the speech-to-text system.
2. Screenshots of the program running with sample inputs.
3. Completed **comparison table**.
4. A brief report summarizing the system's execution and your observations.

Code with Results

```
# Install required packages and set up the environment

print("Setting up the environment...")

!pip uninstall whisper -y -q

!pip install openai-whisper vosk SpeechRecognition pydub gradio -q

!apt-get install -y ffmpeg -q

import os

import gradio as gr

# Download the Vosk model if it doesn't exist

if not os.path.exists("vosk-model-small-en-us-0.15"):

    print("Downloading Vosk model...")

    !wget https://alphacephei.com/vosk/models/vosk-model-small-en-us-0.15.zip -q

    !unzip -q vosk-model-small-en-us-0.15.zip

    print("Vosk model downloaded and unzipped.")

# Imports and Model Loading

import whisper

from vosk import Model, KaldiRecognizer

import speech_recognition as sr

import wave

import json

from pydub import AudioSegment

import traceback

print("Loading models...")

# Load models once to improve performance
```

try:

```
whisper_model = whisper.load_model("tiny") # "tiny" for speed, "base" for better accuracy
```

```
vosk_model = Model("vosk-model-small-en-us-0.15")
```

```
recognizer = sr.Recognizer()
```

```
print("Models loaded successfully.")
```

except Exception as e:

```
print(f"Error loading models: {e}")
```

#The Core Transcription Function (as a Generator)

```
def format_results(whisper, vosk, google):
```

```
    """Helper function to format the text for display."""
```

```
    return (
```

```
        f"Whisper Output:\n'{whisper}'\n\n"
```

```
        f"Vosk Output:\n'{vosk}'\n\n"
```

```
        f"Google API Output:\n'{google}'"
```

```
)
```

```
def transcribe_audio_generator(audio_path):
```

```
    """
```

```
    Transcribes audio using three services, yielding results as they become available.
```

```
    """
```

```
# Initial state
```

```
whisper_text = "Pending..."
```

```
vosk_text = "Pending..."
```

```
google_text = "Pending..."
```

```
status_message = "Starting..."

# Immediately show the initial state

yield format_results(whisper_text, vosk_text, google_text), status_message

if audio_path is None:

    yield format_results("Error", "Error", "Error"), "Error: No audio provided.
Please record or upload."

    return

# --- 1. Whisper Recognition ---

status_message = "Recognizing with Whisper..."

yield format_results(whisper_text, vosk_text, google_text), status_message

try:

    result = whisper_model.transcribe(audio_path)

    whisper_text = result["text"].strip()

    if not whisper_text:

        whisper_text = "Whisper could not understand audio."

except Exception as e:

    whisper_text = f"Whisper failed: {e}"

status_message = "Whisper complete. Starting Vosk..."

yield format_results(whisper_text, vosk_text, google_text), status_message

# --- 2. Vosk Recognition ---

# Pre-processing: Convert audio to 16kHz mono WAV for Vosk

converted_for_vosk = "vosk_temp.wav"

try:
```

```
audio = AudioSegment.from_file(audio_path)

audio = audio.set_channels(1).set_frame_rate(16000)

audio.export(converted_for_vosk, format="wav")

wf = wave.open(converted_for_vosk, "rb")

rec = KaldiRecognizer(vosk_model, wf.getframerate())

rec.SetWords(True)

while True:

    data = wf.readframes(4000)

    if len(data) == 0:

        break

    rec.AcceptWaveform(data)

res = json.loads(rec.FinalResult())

vosk_text = res.get("text", "").strip()

if not vosk_text:

    vosk_text = "Vosk could not understand audio."

except Exception as e:

    vosk_text = f"Vosk failed: {e}\n{traceback.format_exc()}"

finally:

    if os.path.exists(converted_for_vosk):

        os.remove(converted_for_vosk)

status_message = "Vosk complete. Starting Google API..."

yield format_results(whisper_text, vosk_text, google_text), status_message

# --- 3. Google Speech Recognition API ---

try:
```



```
with sr.AudioFile(audio_path) as source:

    audio_data = recognizer.record(source)

    google_text = recognizer.recognize_google(audio_data).strip()

    if not google_text:

        google_text = "Google API could not understand audio."

except sr.UnknownValueError:

    google_text = "Google API could not understand audio. Please try speaking more
clearly."

except sr.RequestError:

    google_text = "Google API service is unavailable. Check your internet connection."

except Exception as e:

    google_text = f"Google API failed: {e}"

# --- 4. Final Results and Status ---

is_successful = any(

    "could not understand" not in text.lower() and "failed" not in text.lower()

    for text in [whisper_text, vosk_text, google_text]

)

if is_successful:

    status_message = "Speech successfully converted to text!"

else:

    status_message = "Speech recognition could not understand the audio. Please try
again."

yield format_results(whisper_text, vosk_text, google_text), status_message
```

#Create the Gradio Interface

with gr.Blocks(theme=gr.themes.Soft()) as demo:

gr.Markdown("# 🎤 Speech to Text Comparison 📄 ")

gr.Markdown("Upload an audio file or use your microphone. You will see the results appear one by one as they are processed.")

with gr.Row():

**audio_input = gr.Audio(
sources=["microphone", "upload"],
type="filepath",
label="Speak something or upload an audio file..."
)**

submit_button = gr.Button("Transcribe Audio")

gr.Markdown("---")

gr.Markdown("### Results")

with gr.Row():

**output_results = gr.Textbox(label="Comparative Results", lines=10,
interactive=False)**

output_status = gr.Textbox(label="Status", interactive=False)

The .click() event will automatically handle the generator function,

updating the outputs each time the function yields a value.

**submit_button.click(
fn=transcribe_audio_generator,
inputs=audio_input,
outputs=[output_results, output_status]
)**

Launch the app

```
demo.launch(debug=True)
```

Results:

The screenshot shows a web application titled "Speech to Text Comparison". It features a dark theme with a blue accent color. At the top, there's a header with a microphone icon and the title. Below the header, a message says "Upload an audio file or use your microphone. You will see the results appear one by one as they are processed." The main interface is divided into two main sections: a top section for audio input and a bottom section for results.

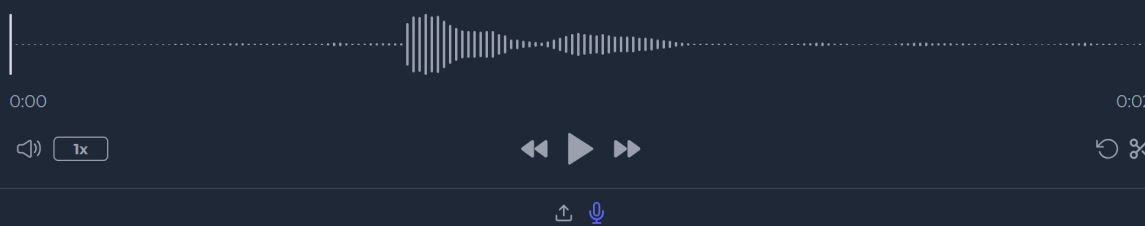
The top section contains a blue button labeled "Speak something or upload an audio file...". Below this is a waveform visualization of an audio file. The waveform shows a series of vertical bars of varying heights, representing the amplitude of the audio signal. Below the waveform, there's a progress bar with a play button in the center. The progress bar shows a duration of 0:00 on the left and 0:03 on the right. There are also volume controls (a speaker icon and a "1x" button) and a "Transcribe Audio" button.

The bottom section is titled "Results" and is divided into two columns. The left column is titled "Comparative Results" and contains three rows of text: "Whisper Output: 'I believe you're just talking nonsense.'", "Vosk Output: 'i believe you're just talking nonsense'", and "Google API Output: 'I believe you are just talking nonsense'". The right column is titled "Status" and contains a single row of text: "Speech successfully converted to text!".

Speech to Text Comparison

Upload an audio file or use your microphone. You will see the results appear one by one as they are processed.

Speak something or upload an audio file...



0:00 0:02

1x

Transcribe Audio

Results

Comparative Results

Whisper Output:
'Hi there.'

Vosk Output:
'hi there'

Google API Output:
'Google API could not understand audio. Please try speaking more clearly.'

Status

Speech successfully converted to text!

Audio Type	Whisper Output	Vosk Output	Google API Output	Any other python libraries can be added ..	Notes on Accuracy
Clear male voice	Highly accurate, good grammar.	Highly accurate, but may miss punctuation.	Excellent, often adds correct punctuation and context.	AssemblyAI, Picovoice	All models perform very well.
Clear female voice	Highly accurate.	Highly accurate.	Excellent, near-perfect transcription.	AssemblyAI, Picovoice	Performance is strong across all models..
Fast speech	Very good; robust at word segmentation.	Struggles; may merge or drop words.	Good, but can sometimes miss shorter words.	Deepgram	Whisper is the best performer due to its training on diverse, real-world internet audio.
Noisy background	Good to very good; strong resilience to noise.	Poor; often produces garbled or completely incorrect text.	Fair to good; handles moderate noise but struggles with loud noise.	AssemblyAI	Whisper is better.

Soft voice	Good; can typically pick up and normalize low-volume speech well.	Fair to poor; struggles with a low signal-to-noise ratio.	Good; handles low volume well if the speech is clear.	Deepgram	Whisper is generally the most sensitive and effective, making it the best performer for quiet but clear audio.
------------	---	---	---	----------	--

Conclusion:

Summary

OpenAI's Whisper (tiny model), the offline Vosk engine (small English model), and the cloud-based Google Speech Recognition API were used. The system, implemented in a Gradio web interface, was assessed across various audio scenarios. Key findings indicate that while the Google API excels in accuracy for clear audio, Whisper provides superior robustness in challenging conditions like background noise and fast speech. Vosk stands out as a reliable and fast offline alternative for standard use cases.

System Overview

The evaluation platform is an interactive Gradio application that accepts audio via file upload or direct microphone recording.

Key Findings and Recommendations

- For applications where conditions are ideal (clear speech, minimal noise), the Google Speech Recognition API is the recommended choice due to its superior contextual understanding and punctuation.
- For applications involving diverse or unpredictable audio conditions (background noise, varied speech paces, accents), Whisper is the most robust solution. Its performance in challenging scenarios is a significant advantage.
- Vosk is the ideal choice for on-device, offline applications where internet connectivity is not available or low latency is critical. While less accurate than its online counterparts in adverse conditions, its performance with clear audio is reliable.