Lab_1

December 24, 2024

Manish Mangrati

1 Theory

Sound processing is a field that involves manipulating sound signals, typically in digital form, to create new effects or modify existing sounds. The process includes various techniques like synthesis, filtering, and effects processing. In this lab, we will focus on audio synthesis, which is the creation of sound from simple waveforms, and basic audio manipulation, such as filtering and pitch shifting.

1.1 Looking at the sampling rate of a audio

[8]:
```
import librosa

y, sr = librosa.load(r"C:\Users\manis\Documents\Jupyter Notebook\nadaniya.mp3",␣
  ↪sr=None)
print(f"Sample rate: {sr}, Duration: {len(y) / sr:.2f} seconds")
```

```
Sample rate: 48000, Duration: 169.15 seconds
```

1.2 Playing audio file

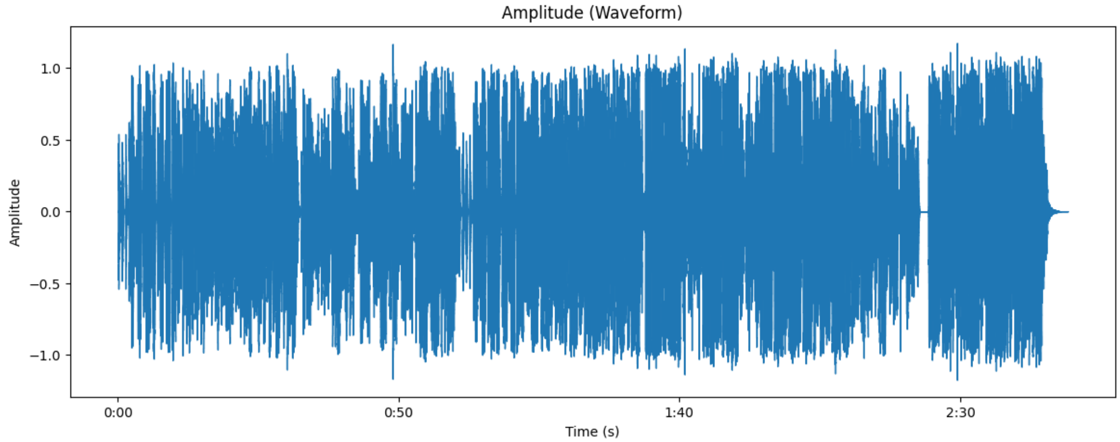
[1]:
```
import pygame
import time

# Initialize the mixer
pygame.mixer.init()

# Load the audio file
file_path = r"C:\Users\manis\Documents\Jupyter Notebook\nadaniya.mp3"
pygame.mixer.music.load(file_path)

# Start playing the audio
pygame.mixer.music.play()
print("Audio started. Press 'p' to pause, 'r' to resume, and 'q' to quit.")

while True:
    command = input("Enter command: ").strip().lower()
```

```python
    if command == 'p':
        # Pause the audio
        pygame.mixer.music.pause()
        print("Audio paused.")
    elif command == 'r':
        # Resume the audio
        pygame.mixer.music.unpause()
        print("Audio resumed.")
    elif command == 'q':
        # Stop the audio and exit
        pygame.mixer.music.stop()
        print("Audio stopped. Exiting...")
        break
    else:
        print("Invalid command. Use 'p' to pause, 'r' to resume, 'q' to quit.")
```

```
pygame 2.6.1 (SDL 2.28.4, Python 3.12.8)
Hello from the pygame community. https://www.pygame.org/contribute.html
Audio started. Press 'p' to pause, 'r' to resume, and 'q' to quit.

Enter command:  p

Audio paused.

Enter command:  r

Audio resumed.

Enter command:  q

Audio stopped. Exiting…
```

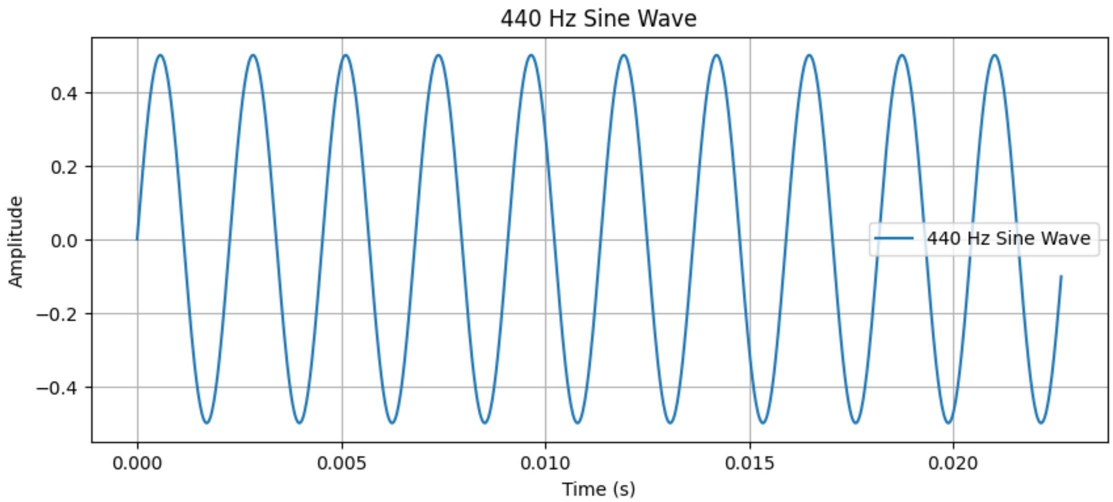
1.3 Plotting the waveform of the audio file

[2]:
```python
import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt

# Load the audio file
file_path = r"C:\Users\manis\Documents\Jupyter Notebook\nadaniya.mp3"
y, sr = librosa.load(file_path, sr=None)
plt.figure(figsize=(14, 5))
plt.title("Amplitude (Waveform)")
librosa.display.waveshow(y, sr=sr)
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.show()
```



1.4 Generating the sine wave and playing the sound

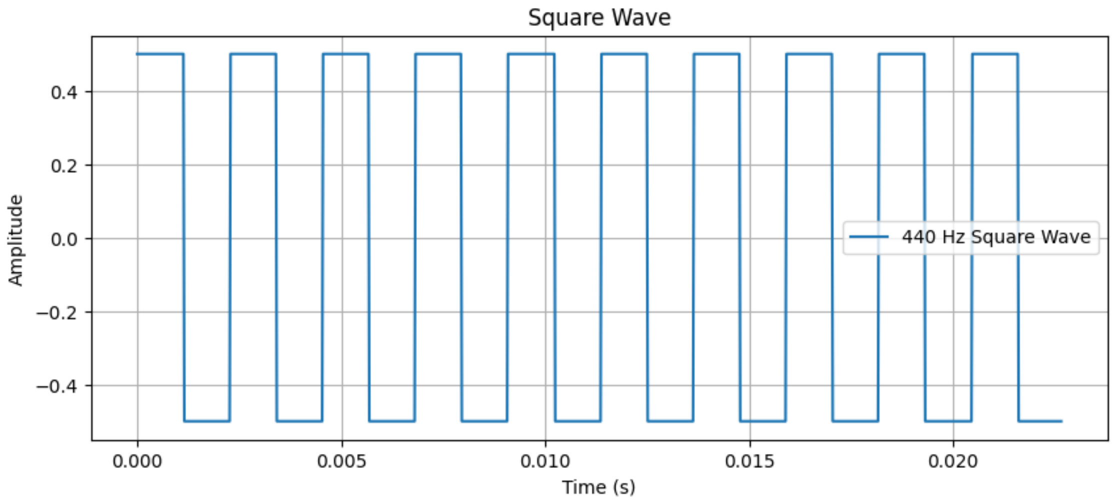
[3]:
```
import numpy as np
import sounddevice as sd
import matplotlib.pyplot as plt

# Parameters
frequency = 440  # Frequency in Hz (A4 note)
duration = 5.0   # Duration in seconds
sample_rate = 44100  # Sampling frequency in Hz

# Generate the sine wave
t = np.linspace(0, duration, int(sample_rate * duration), endpoint=False)  #␣
↪Time vector
sine_wave = 0.5 * np.sin(2 * np.pi * frequency * t)  # Amplitude scaled to 0.5

# Play the sine wave
sd.play(sine_wave, samplerate=sample_rate)
sd.wait()  # Wait until playback is complete

# Plot the sine wave
plt.figure(figsize=(10, 4))
plt.plot(t[:1000], sine_wave[:1000], label=f"{frequency} Hz Sine Wave")  # Plot␣
↪the first 1000 samples
plt.title("440 Hz Sine Wave")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.legend()
plt.grid()
plt.show()
```



1.5 Generating the square wave and playing the sound

[4]:
```
import numpy as np
import sounddevice as sd
import matplotlib.pyplot as plt
from scipy.signal import square

# Parameters
frequency = 440  # Frequency in Hz
amplitude = 0.5  # Amplitude (range 0 to 1)
duration = 2.0   # Duration in seconds
sample_rate = 44100  # Sampling frequency in Hz

# Generate the square wave
t = np.linspace(0, duration, int(sample_rate * duration), endpoint=False)  #␣
↪Time vector
square_wave = amplitude * square(2 * np.pi * frequency * t)

# Play the square wave
sd.play(square_wave, samplerate=sample_rate)
sd.wait()  # Wait until playback is complete

# Plot the square wave
plt.figure(figsize=(10, 4))
plt.plot(t[:1000], square_wave[:1000], label=f"{frequency} Hz Square Wave")  #␣
↪Plot the first 1000 samples
plt.title("Square Wave")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
```

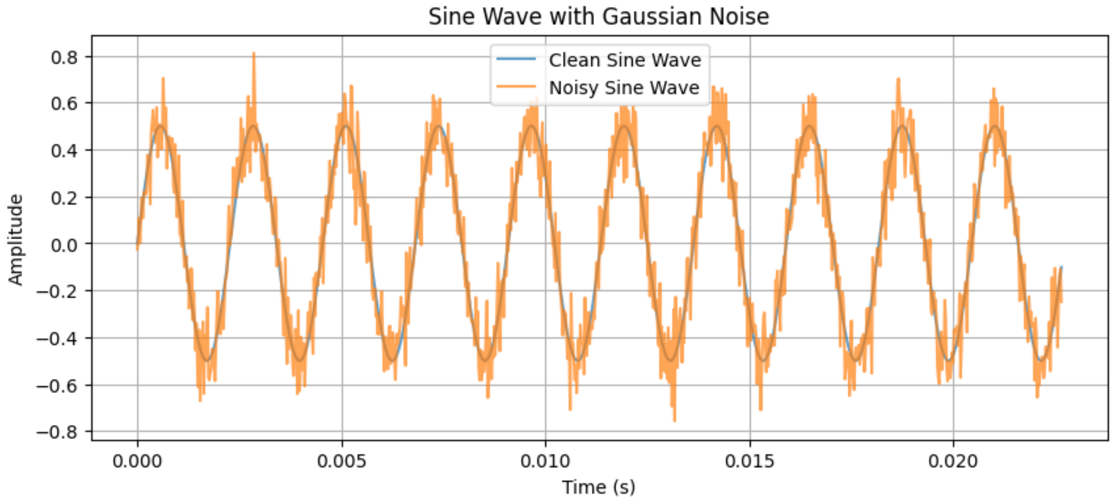
```
plt.legend()
plt.grid()
plt.show()
```



1.6 Adding the gaussian noise to the prevously poltted sine wave of 440hz

[5]:
```
import numpy as np
import sounddevice as sd
import matplotlib.pyplot as plt

# Parameters
frequency = 440  # Frequency in Hz
amplitude = 0.5  # Amplitude
duration = 2.0   # Duration in seconds
sample_rate = 44100  # Sampling frequency in Hz
noise_std_dev = 0.1  # Standard deviation of the Gaussian noise

# Generate the sine wave
t = np.linspace(0, duration, int(sample_rate * duration), endpoint=False)
sine_wave = amplitude * np.sin(2 * np.pi * frequency * t)

# Generate Gaussian noise
gaussian_noise = np.random.normal(0, noise_std_dev, sine_wave.shape)

# Add noise to the sine wave
noisy_sine_wave = sine_wave + gaussian_noise

# Play the noisy sine wave
sd.play(noisy_sine_wave, samplerate=sample_rate)
```

```
sd.wait()

# Plot the clean sine wave
plt.figure(figsize=(10, 4))
plt.plot(t[:1000], sine_wave[:1000], label="Clean Sine Wave", alpha=0.7)
plt.plot(t[:1000], noisy_sine_wave[:1000], label="Noisy Sine Wave", alpha=0.7)
plt.title("Sine Wave with Gaussian Noise")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.legend()
plt.grid()
plt.show()
```

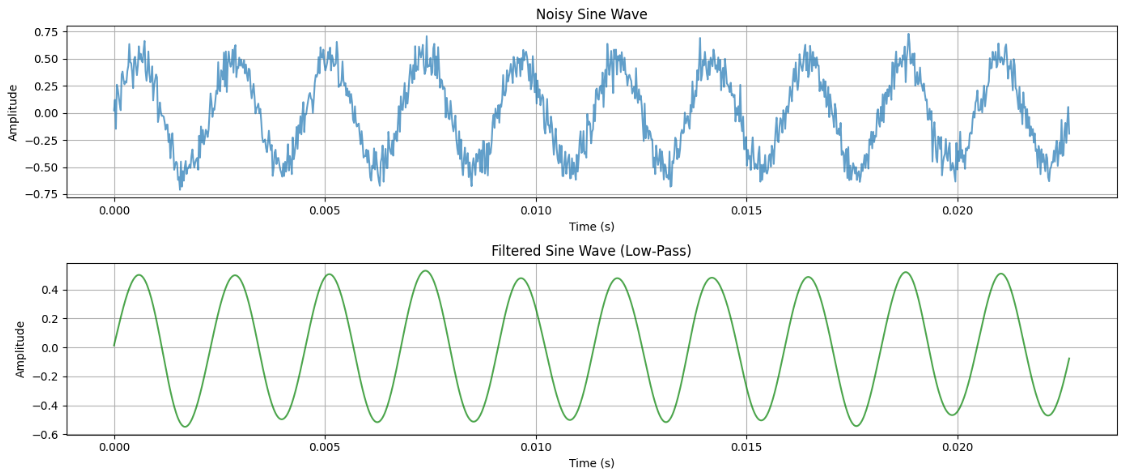


1.7 Filtering the gaussian noise and plotting the necessary diagram

[6]:

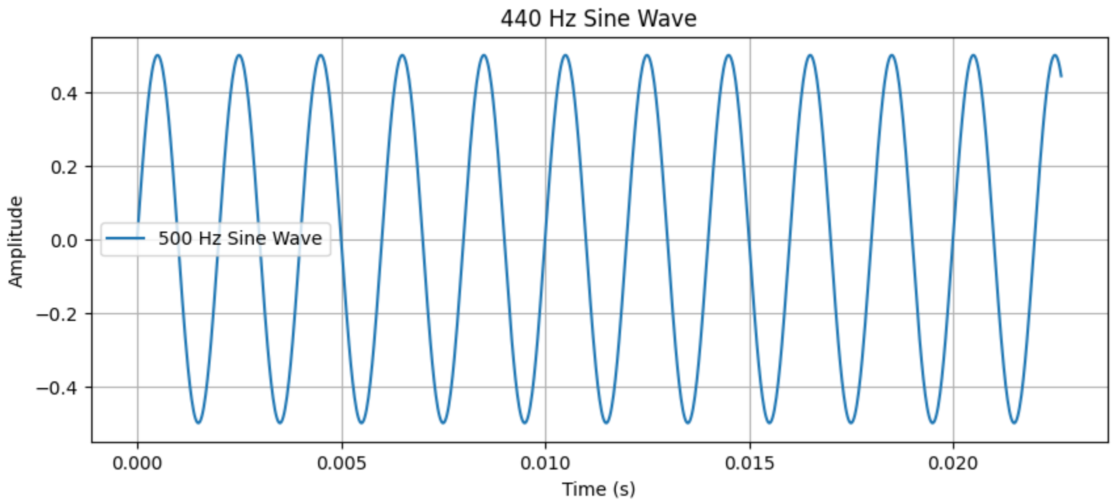
```
import numpy as np
import scipy.signal as signal
import matplotlib.pyplot as plt
import sounddevice as sd

# Parameters
frequency = 440  # Frequency of sine wave in Hz
amplitude = 0.5  # Amplitude
duration = 2.0   # Duration in seconds
sample_rate = 44100  # Sampling frequency in Hz
noise_std_dev = 0.1  # Standard deviation of Gaussian noise
cutoff_frequency = 1000  # Cutoff frequency of the low-pass filter in Hz

# Generate the sine wave
```

```
t = np.linspace(0, duration, int(sample_rate * duration), endpoint=False)
sine_wave = amplitude * np.sin(2 * np.pi * frequency * t)

# Generate Gaussian noise
gaussian_noise = np.random.normal(0, noise_std_dev, sine_wave.shape)

# Add noise to the sine wave
noisy_sine_wave = sine_wave + gaussian_noise

# Design a low-pass filter
nyquist = 0.5 * sample_rate
normal_cutoff = cutoff_frequency / nyquist
b, a = signal.butter(4, normal_cutoff, btype='low', analog=False)  # 4th-order␣
↪Butterworth filter

# Apply the filter
filtered_signal = signal.filtfilt(b, a, noisy_sine_wave)

# Play the filtered sine wave
sd.play(filtered_signal, samplerate=sample_rate)
sd.wait()

# Plot the results
plt.figure(figsize=(14, 6))

# Plot the noisy sine wave
plt.subplot(2, 1, 1)
plt.plot(t[:1000], noisy_sine_wave[:1000], label="Noisy Sine Wave", alpha=0.7)
plt.title("Noisy Sine Wave")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.grid()

# Plot the filtered sine wave
plt.subplot(2, 1, 2)
plt.plot(t[:1000], filtered_signal[:1000], label="Filtered Sine Wave",␣
↪color="green", alpha=0.7)
plt.title("Filtered Sine Wave (Low-Pass)")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.grid()
plt.tight_layout()
plt.show()
```



1.8 Generating the sine wave and playing the sound pitch shifted to 450

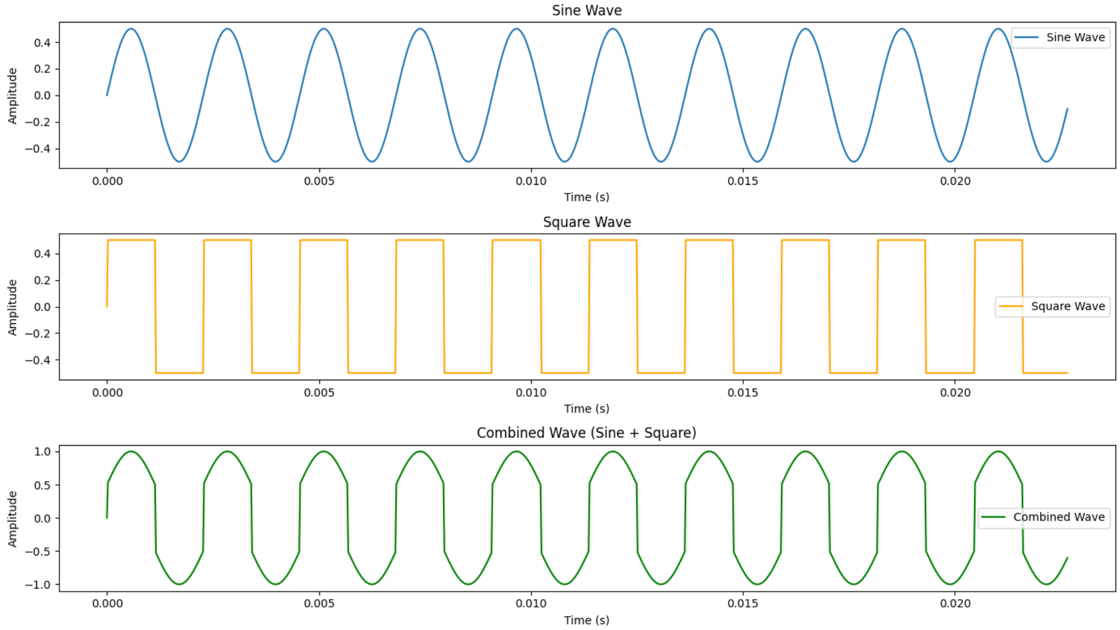
[7]:
```
import numpy as np
import sounddevice as sd
import matplotlib.pyplot as plt

# Parameters
frequency =  500 # Frequency in Hz (A4 note)
duration = 5.0   # Duration in seconds
sample_rate = 44100  # Sampling frequency in Hz

# Generate the sine wave
t = np.linspace(0, duration, int(sample_rate * duration), endpoint=False)  # Time vector
sine_wave = 0.5 * np.sin(2 * np.pi * frequency * t)  # Amplitude scaled to 0.5

# Play the sine wave
sd.play(sine_wave, samplerate=sample_rate)
sd.wait()  # Wait until playback is complete

# Plot the sine wave
plt.figure(figsize=(10, 4))
plt.plot(t[:1000], sine_wave[:1000], label=f"{frequency} Hz Sine Wave")  # Plot the first 1000 samples
plt.title("440 Hz Sine Wave")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.legend()
plt.grid()
plt.show()
```



1.9 Combining both square waveform and sin waveform

[8]:
```
import numpy as np
import matplotlib.pyplot as plt
import sounddevice as sd

# Parameters for the waveforms
fs = 44100  # Sampling frequency
duration = 2.0  # Duration in seconds
frequency = 440  # Frequency of both waveforms (Hz)

# Generate time array
t = np.linspace(0, duration, int(fs * duration), endpoint=False)

# Generate sine and square waveforms
sine_wave = 0.5 * np.sin(2 * np.pi * frequency * t)  # Sine wave (Amplitude = 0.
↪5)
square_wave = 0.5 * np.sign(np.sin(2 * np.pi * frequency * t))  # Square wave␣
↪(Amplitude = 0.5)

# Combine the waveforms
combined_wave = sine_wave + square_wave

# Normalize the combined wave to avoid clipping
combined_wave = combined_wave / np.max(np.abs(combined_wave))

# Plot the waveforms
plt.figure(figsize=(14, 8))
```

```python
# Sine wave
plt.subplot(3, 1, 1)
plt.plot(t[:1000], sine_wave[:1000], label="Sine Wave")
plt.title("Sine Wave")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.legend()

# Square wave
plt.subplot(3, 1, 2)
plt.plot(t[:1000], square_wave[:1000], label="Square Wave", color="orange")
plt.title("Square Wave")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.legend()

# Combined wave
plt.subplot(3, 1, 3)
plt.plot(t[:1000], combined_wave[:1000], label="Combined Wave", color="green")
plt.title("Combined Wave (Sine + Square)")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.legend()

plt.tight_layout()
plt.show()

# Play the combined waveform
print("Playing combined waveform...")
sd.play(combined_wave, samplerate=fs)
sd.wait()
```




```
Playing combined waveform…
```

[ ]: