EE 242 Lab 3a – Frequency Domain Representation of Signals - Fourier Series

Mason Wheeler

This lab has 2 exercises to be completed as a team. Each should be given a separate code cell in your Notebook, followed by a markdown cell with report discussion. Your notebook should start with a markdown title and overview cell, which should be followed by an import cell that has the import statements for all assignments. For this assignment, you will need to import: numpy, the wavfile package from scipy.io, simpleaudio/librosa, and matplotlib.pyplot.

In [7]:
```
# We'll refer to this as the "import cell." Every module you import should be i
%matplotlib notebook
import numpy as np
import matplotlib
import scipy.signal as sig
import matplotlib.pyplot as plt
import simpleaudio as sa
# import whatever other modules you use in this lab -- there are more that you
```

Summary

In this lab, we will learn how to build periodic signals from component sinusoids and how to transform signals from the time domain to the frequency domain. The concepts we'll focus on include: implementation of the Fourier Series synthesis equation, using a discrete implementation of the Fourier Transform (DFT) with a digitized signal, and understanding the relationship between the discrete DFT index k and frequency ω for both the original continuous signal x(t). This is a two-week lab. You should plan on completing the first 2 assignments in the first week.

Lab 3a turn in checklist

• Lab 3a Jupyter notebook with code for the 2 exercises assignment in separate cells. Each assignment cell should contain markdown cells (same as lab overview cells) for the responses to lab report questions. Include your lab members' names at the top of the notebook.

Please submit the report as PDF (You may also use : https://www.vertopal.com/ suggested by a student)

Assignment 1 -- Generating simple periodic signals

In the first assignment, you will develop an understanding of how some periodic signals are easier to approximate than others with a truncated Fourier Series. In this lab, we'll work with real signals and use the synthesis equation:

$$x(t) = a_0 + \sum_{k=1}^{N} 2|a_k|cos(k\omega_0 t + \angle a_k)$$

In lecture, you saw that you get ripples at transition points in approximating a square wave (Gibbs phenomenon). This happens for any signals with sharp edges. This assignment will involve approximating two signals (a sawtooth and a triangle wave) that have the same fundamental frequency (20Hz).

A. Write a function for generating a real-valued periodic time signal given the Fourier series coefficients $[a_0\ a_1\ \cdots\ a_N]$, the sampling frequency, and the fundamental frequency. You may choose to have complex input coefficients or have separate magnitude and phase vectors for describing a_k.

B. Define variables for the sampling frequency (8kHz) and the fundamental frequency (20Hz). Using this sampling frequency, create a time vector for a length of 200ms.

C. The sawtooth signal has coefficients as follows:

$$a_0 = 0.5, a_k = 1/(j2k\pi)$$

Using the function from part A, create three approximations of this signal with N = 2,5,20 and plot together in a 3×1 comparison.

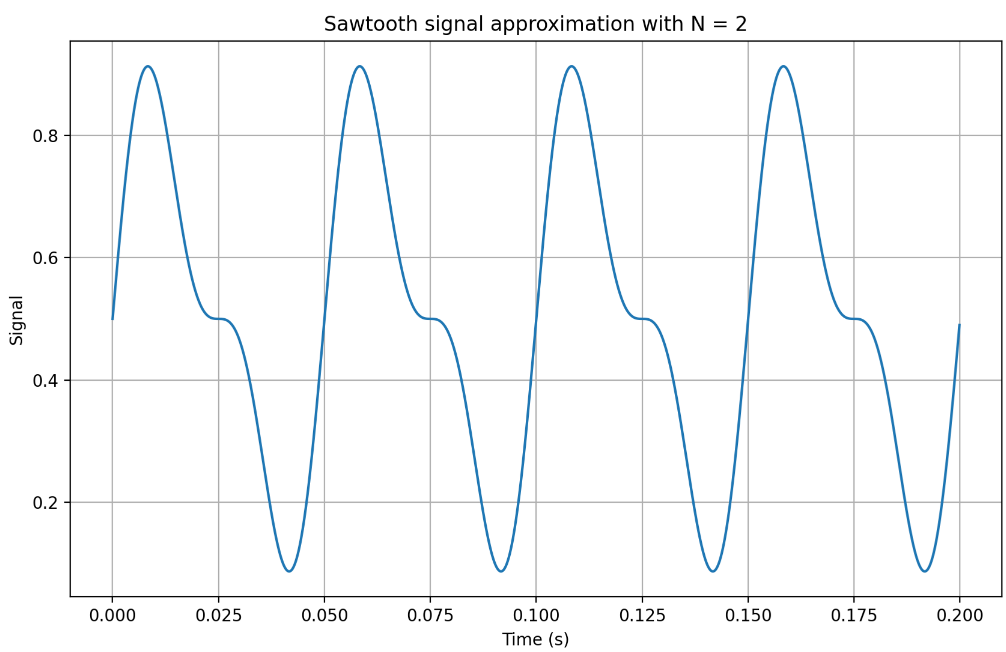
D. A triangle signal has coefficients:

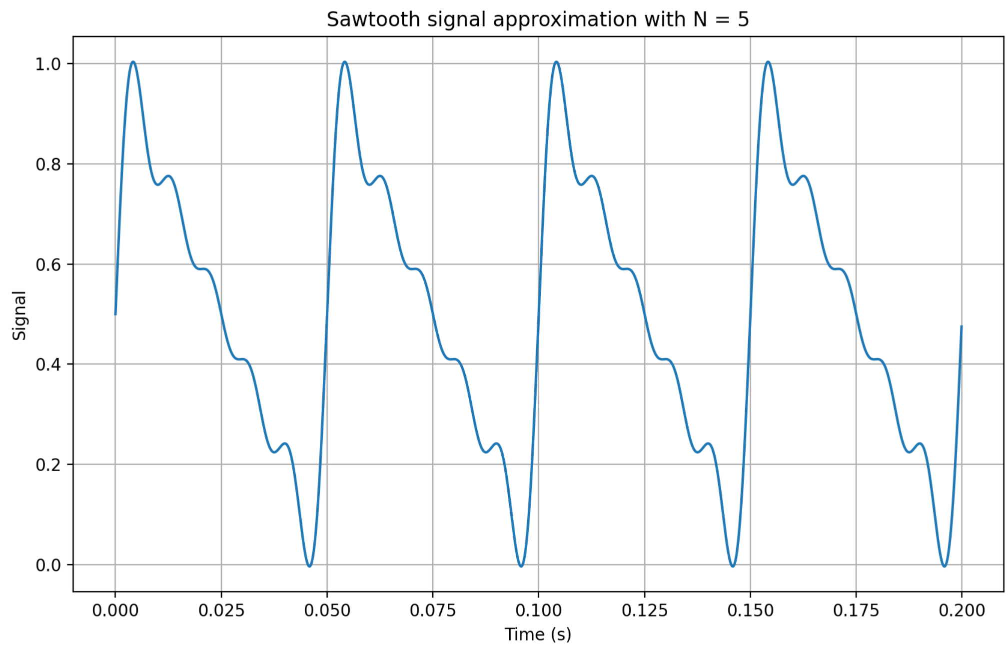
$$a_0 = 0.5, a_k = \frac{2sin(k\pi/2)}{j(k\pi)^2}e^{-j2k\pi/2}$$

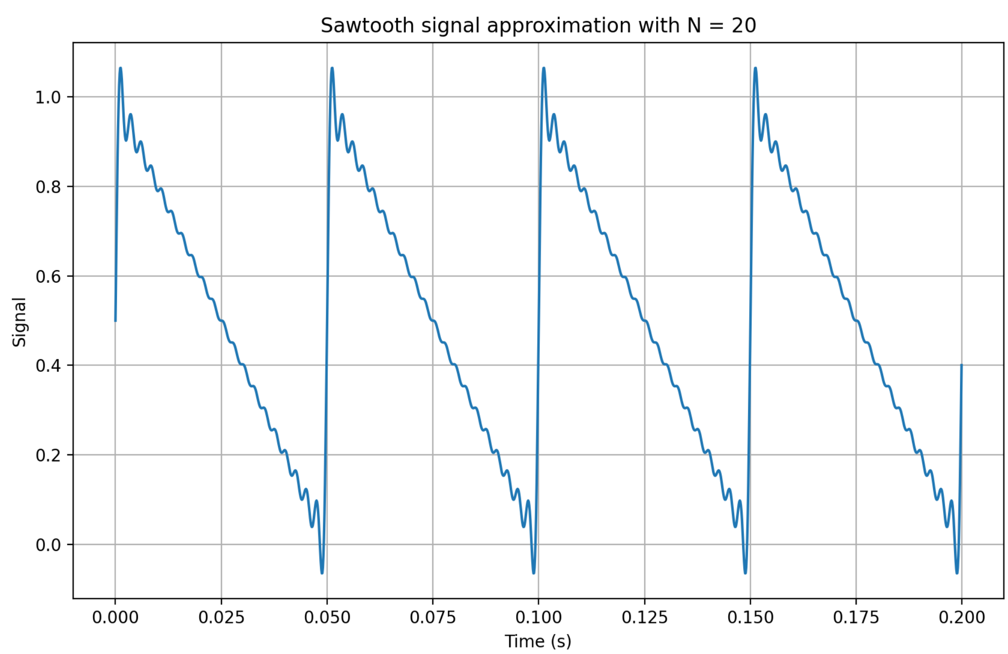
Create three approximations of this signal with N = 2,5,20 and plot together in a 3×1 comparison.

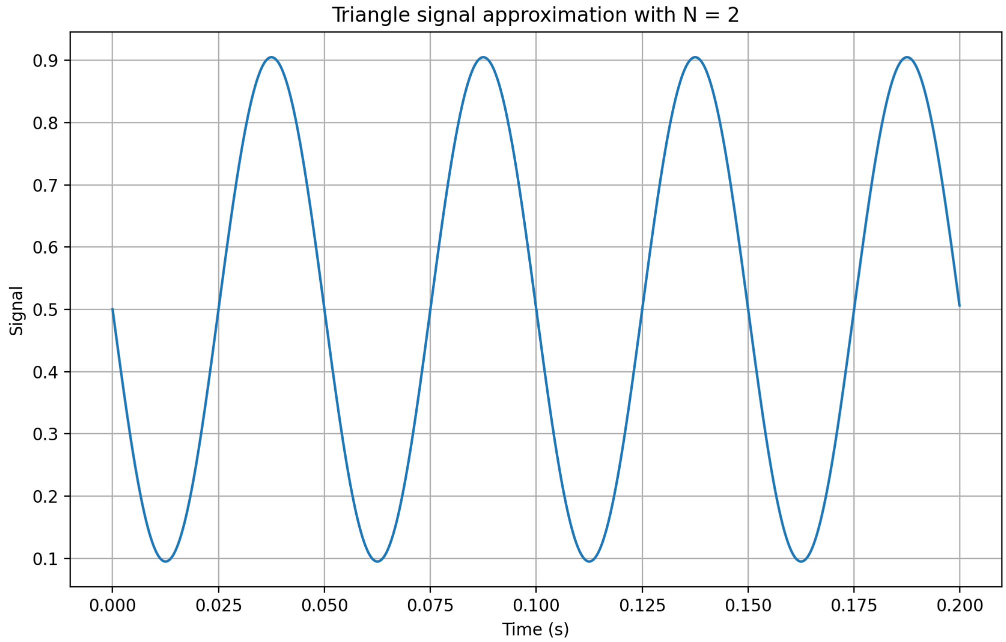
In [8]:
```
# Assignment 1 - Generating Periodic Signals

# Part A
def generate_signal(a_coefficients, sampling_frequency, fundamental_frequency,
    omega = 2 * np.pi * fundamental_frequency
    x_t = np.zeros_like(time_vector)
    a_0 = a_coefficients[0]

    for k, a_k in enumerate(a_coefficients[1:], 1):
        x_t += 2 * np.abs(a_k) * np.cos(k * omega * time_vector + np.angle(a_k)

    return a_0 + x_t

# Part B
sampling_frequency = 8000  # 8kHz
fundamental_frequency = 20  # 20Hz
time_vector = np.linspace(0, 0.2, int(0.2 * sampling_frequency), endpoint=False

# Part C
```

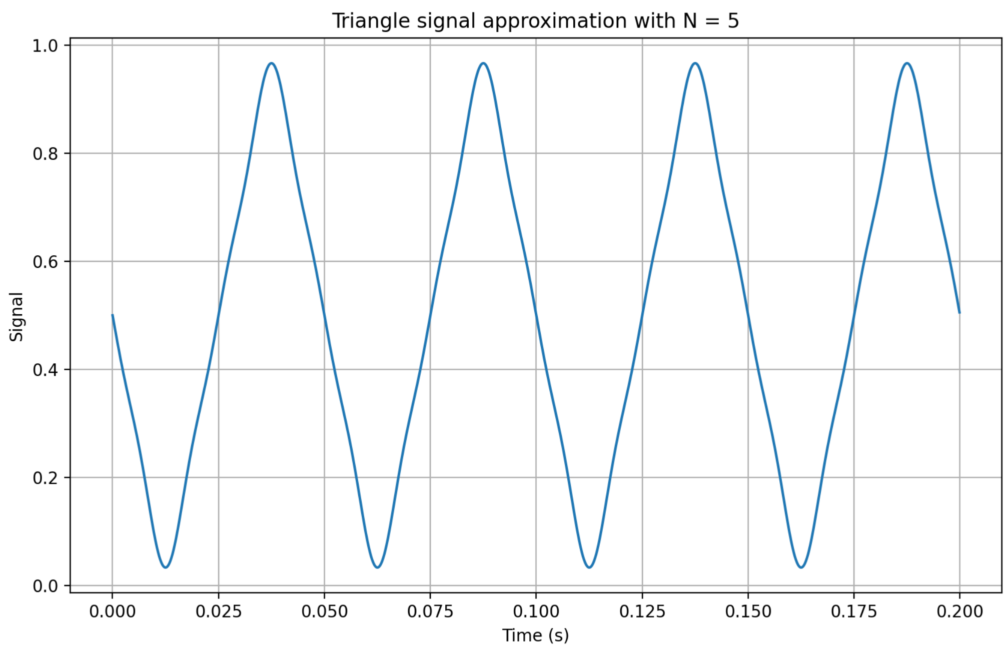
```python
def sawtooth_coefficients(N):
    a_coefficients = [0.5]
    for k in range(1, N + 1):
        a_coefficients.append(1 / (1j * 2 * np.pi * k))
    return a_coefficients

for N in [2, 5, 20]:
    a_coefficients = sawtooth_coefficients(N)
    signal = generate_signal(a_coefficients, sampling_frequency, fundamental_fr

    plt.figure(figsize=(10, 6))
    plt.plot(time_vector, signal)
    plt.title(f"Sawtooth signal approximation with N = {N}")
    plt.xlabel("Time (s)")
    plt.ylabel("Signal")
    plt.grid(True)
    plt.show()

# Part D
def triangle_coefficients(N):
    a_coefficients = [0.5]
    for k in range(1, N + 1):
        a_coefficients.append((2 * np.sin(k * np.pi / 2)) / (1j * (k * np.pi)**
    return a_coefficients

for N in [2, 5, 20]:
    a_coefficients = triangle_coefficients(N)
    signal = generate_signal(a_coefficients, sampling_frequency, fundamental_fr

    plt.figure(figsize=(10, 6))
    plt.plot(time_vector, signal)
    plt.title(f"Triangle signal approximation with N = {N}")
    plt.xlabel("Time (s)")
    plt.ylabel("Signal")
    plt.grid(True)
    plt.show()
```

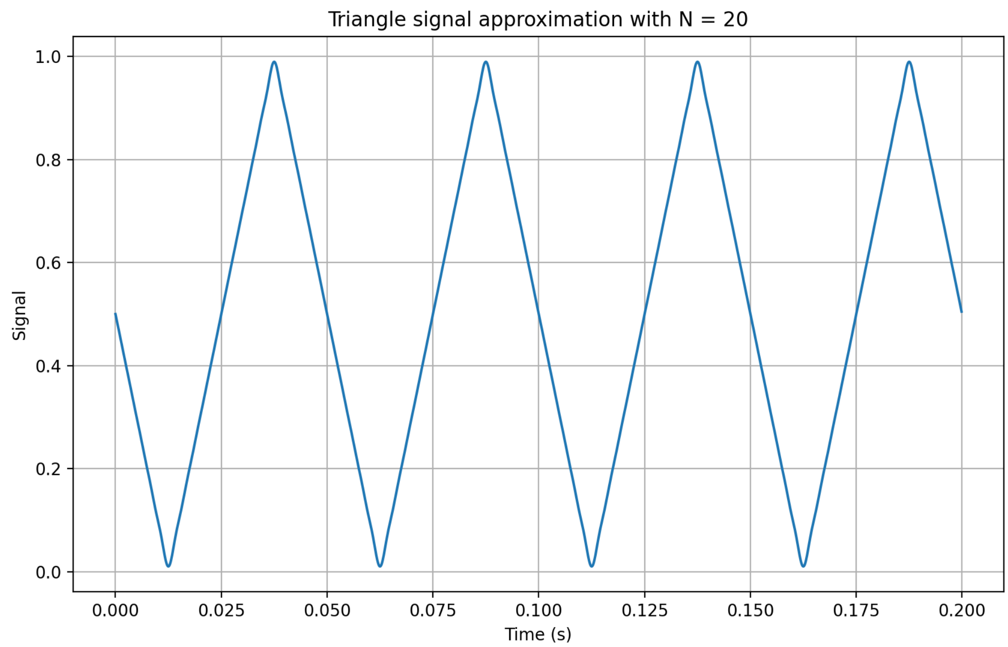



Sawtooth signal approximation with N = 5
Signal
Time (s)
0.0
0.2
0.4
0.6
0.8
1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200


Sawtooth signal approximation with N = 20
Signal
Time (s)
0.0
0.2
0.4
0.6
0.8
1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200


Triangle signal approximation with N = 2
Signal
Time (s)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200


Triangle signal approximation with N = 5
Signal
Time (s)
1.0
0.8
0.6
0.4
0.2
0.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200



Discussion

You should have noticed that the second signal converges more quickly. Discuss the two reasons for this.

Answer

The second signal, which is a triangle wave, converges more quickly than the first signal, which is a sawtooth wave, for two main reasons:

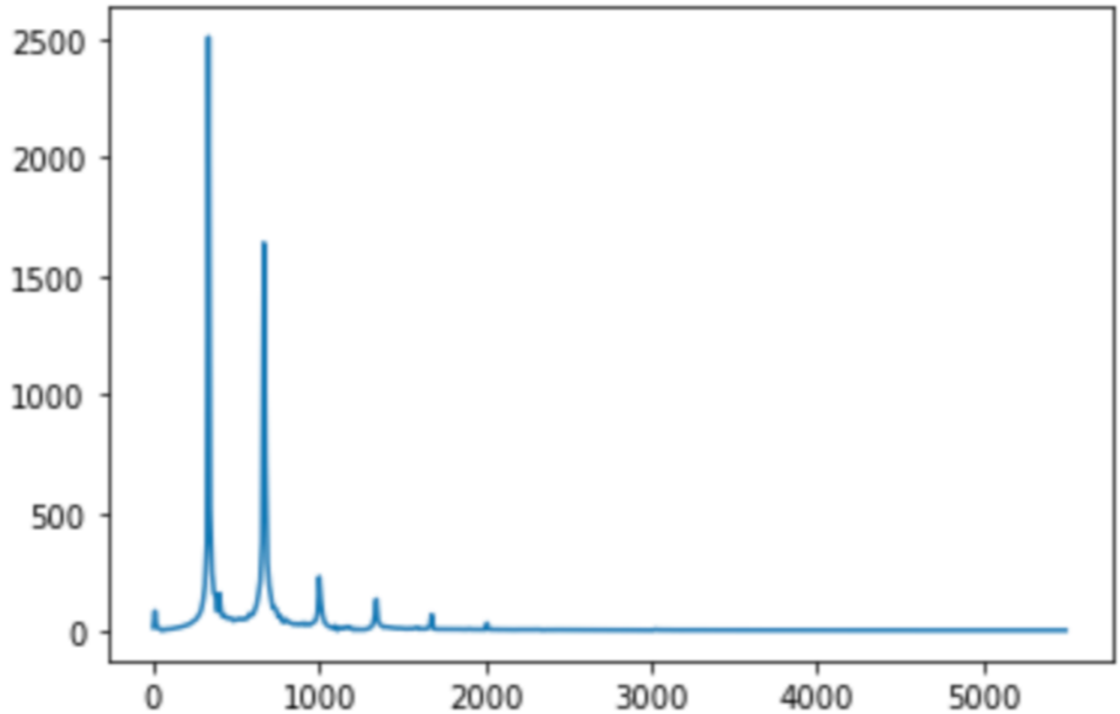
Harmonic decay: The rate at which the Fourier series coefficients (harmonics) decay plays a significant role in the rate of convergence of the series. For the triangle wave, the coefficients decay as $1/k^2$, where k is the harmonic number. This is a faster rate of decay compared to the sawtooth wave, where the coefficients decay as 1/k. The faster decay of the coefficients in the triangle wave means that fewer terms are needed to get a good approximation of the signal, leading to faster convergence.

Continuity of the derivative: The triangle wave has a continuous derivative everywhere, while the sawtooth wave has a discontinuity at the point where the signal resets. Fourier series approximations converge more quickly for functions that are smoother, i.e., functions that have continuous derivatives. The discontinuity in the derivative of the sawtooth wave leads to a slower rate of convergence compared to the triangle wave.

Assignment 2 -- Synthesizing a musical note

In this assignment, you will use the same synthesis equations to try to approximate a single note from a horn, which has the frequency characteristics illustrated below. Download the file horn11short.wav from the assignment page to compare your synthesized version to the original.

Figure below shows the frequency component of a note played by a horn.



A. Read in the horn signal, and use the sampling rate f_s that you read in to create a time vector of length 100ms. Define the fundamental frequency to be f_0 = 335Hz. Create a signal that is a sinusoid at that frequency, and save it as a wav file.

B. Create a vector (or two) to characterize a_k using:

$$|a_k| : [2688, 1900, 316, 178, 78, 38]$$

$$\angle a_k : [-1.73, -1.45, 2.36, 2.30, -2.30, 1.13]$$

assuming $a_0 = 0$ and the first element of the vectors correspond to a_1. Use the function you created in part 1 to synthesize a signal, with f_s and f_0 above, and save it as a wav file.

C. Plot the 100ms section of the original file starting at 200ms with a plot of the synthesized signal in a 2×1 plot.

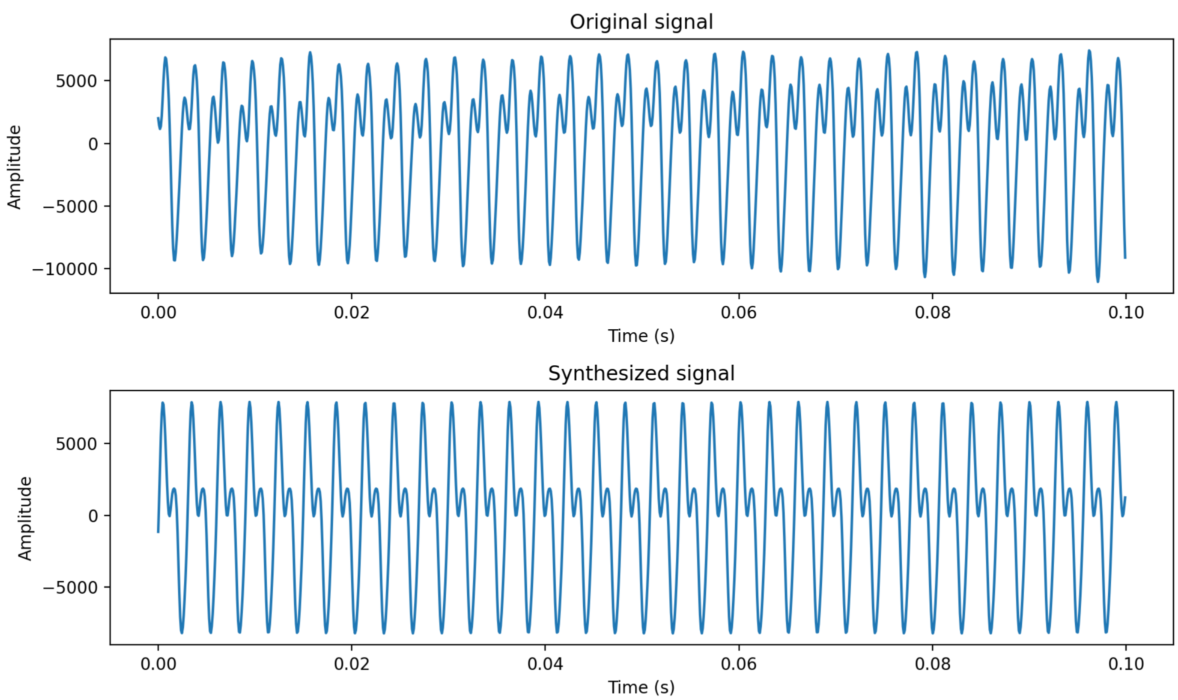
D. Play the original file, the single tone, and the 6-tone approximation in series.

In [9]:
```
# Assignment 2 - Synthesizing a musical note
from scipy.io import wavfile
from IPython.display import Audio

# Function from Assignment 1
def generate_signal(a_coefficients, sampling_frequency, fundamental_frequency,
    omega = 2 * np.pi * fundamental_frequency
```

```
    x_t = np.zeros_like(time_vector)
    a_0 = a_coefficients[0]

    for k, a_k in enumerate(a_coefficients[1:], 1):
        x_t += 2 * np.abs(a_k) * np.cos(k * omega * time_vector + np.angle(a_k)

    return a_0 + x_t

# Part A
# Read the WAV file
fs, horn_signal = wavfile.read("horn11short.wav")

# Define the fundamental frequency
f0 = 335  # Hz

# Create a time vector of length 100ms
t = np.linspace(0, 0.1, int(0.1 * fs), endpoint=False)

# Create a sinusoid signal
sinusoid_signal = np.sin(2 * np.pi * f0 * t)

# Save the sinusoid signal as a WAV file
wavfile.write("sinusoid.wav", fs, sinusoid_signal.astype(np.int16))

# Part B
# Define the magnitude and phase vectors
a_magnitude = np.array([2688, 1900, 316, 178, 78, 38])
a_phase = np.array([-1.73, -1.45, 2.36, 2.30, -2.30, 1.13])

# Combine magnitude and phase to get complex coefficients
a_coefficients = a_magnitude * np.exp(1j * a_phase)

# Add a_0 = 0 to the beginning of the coefficients
a_coefficients = np.insert(a_coefficients, 0, 0)

# Generate the signal using the previously defined function
horn_approximation = generate_signal(a_coefficients, fs, f0, t)

# Save the approximation as a WAV file
wavfile.write("horn_approximation.wav", fs, horn_approximation.astype(np.int16)

# Part C
# Define the start and end indices
start_index = int(0.2 * fs)
end_index = start_index + len(t)

# Plot the original signal
plt.figure(figsize=(10, 6))
plt.subplot(2, 1, 1)
plt.plot(t, horn_signal[start_index:end_index])
plt.title("Original signal")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")

# Plot the synthesized signal
plt.subplot(2, 1, 2)
plt.plot(t, horn_approximation)
plt.title("Synthesized signal")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
```

```
# Adjust the layout and show the plot
plt.tight_layout()
plt.show()

# Part D
# Play the original signal
short_horn = sa.WaveObject.from_wave_file("horn11short.wav")
play_obj = short_horn.play()
play_obj.wait_done()

# Play the sinusoid signal
sinusoid = sa.WaveObject.from_wave_file("sinusoid.wav")
play_obj = sinusoid.play()
play_obj.wait_done()

# Play the synthesized signal
approximation = sa.WaveObject.from_wave_file("horn_approximation.wav")
play_obj = approximation.play()
play_obj.wait_done()
```

```
/var/folders/qz/cx605xfd6hg3dyw77ss7dhc80000gn/T/ipykernel_39816/2005989900.p
y:47: ComplexWarning: Casting complex values to real discards the imaginary pa
rt
  wavfile.write("horn_approximation.wav", fs, horn_approximation.astype(np.int
16))
```



Discussion

The approximation does not sound quite like the original signal and the plot should look pretty different. The difference in sound is in part due to multiple factors, including the truncated approximation, imperfect estimate of the parameters, and the fact that the original signal is not perfectly periodic. Try adjusting some parameters and determine what you think is the main source of distortion.

Answer

The approximation does not sound quite like the original signal due to several factors:

Truncated approximation: The Fourier series approximation uses a finite number of terms, which means it can only approximate the signal to a certain degree of accuracy. The more terms we use, the closer the approximation will be to the original signal, but it will never be exactly the same.

Imperfect estimate of the parameters: The parameters (amplitudes and phases) used in the Fourier series approximation were estimated from the original signal. Any error in these estimates will result in a difference between the original signal and the approximation.

Non-periodicity of the original signal: The Fourier series approximation assumes that the signal is perfectly periodic. However, real-world signals are rarely perfectly periodic, and this discrepancy can lead to differences between the original signal and the approximation.