EE 242 Lab 1b – Modifying Signals - Time Scaling + Time Shift

Yehoshua Luna, Aaron McBride, Ben Eisenhart, Team AE01

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, and matplotlib.pyplot.

In [98]:
```
# We'll refer to this as the "import cell." Every module you import should be imported here.
import numpy as np
import scipy.signal as sig
import matplotlib.pyplot as plt

# Import whatever other modules you use in this lab -- there are more that you need than we've included
import scipy.io.wavfile as wav
from IPython.display import Audio, display

# Shifting waveforms
from scipy.ndimage import shift
```

Summary

In this lab, you will continue to work through a series of exercises to introduce you to working with audio signals and explore the impact of different amplitude and time operations on signals. This is a 1 week lab.

Lab 1b turn in checklist

Lab 1b 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 (if your browser crashes while trying to export, download as html instead)

Assignment 3 -- Time Scaling Audio Signals

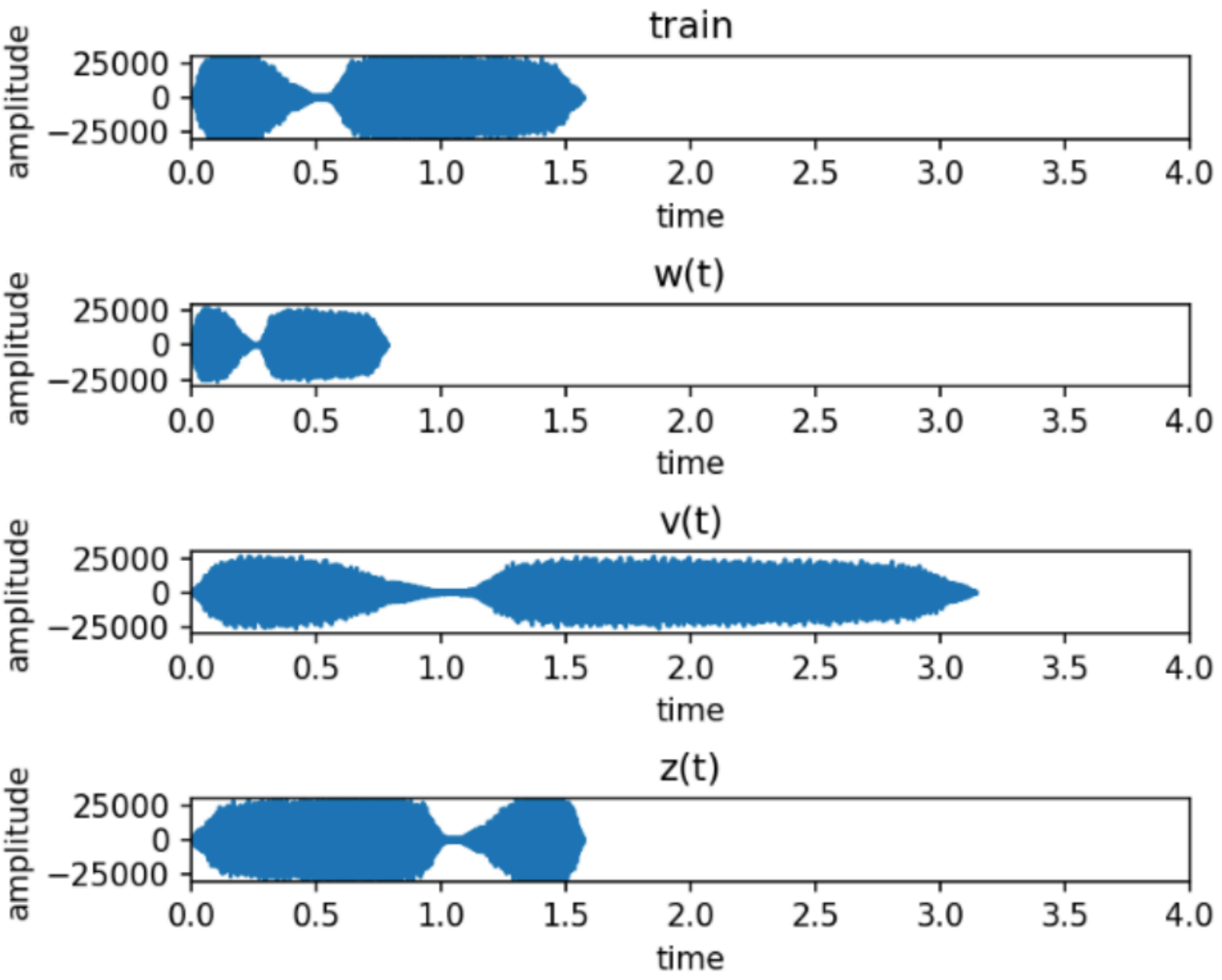
This assignment will have four parts, A-D, each of which should be indicated with comments.

A. Run the timescale function (should be in its own cell) below. Also read the train.wav file as in Lab 1A.

B. Use the timescale function to obtain w(t)=$x_1(2t)$ and v(t)=$x_1(0.5t)$
a. Create w(t) using a=2 and store the outputs of the timescale function as w and t_w.
b. Create v(t) using a=0.5 and store the outputs of the timescale function as v and t_v.
c. Create a time-reversed version of the signal z(t)=$x_1(-t)$ by reversing the order of elements in the sequence. In order to plot in parallel with other signals, use the time vector t_z = t_{x_1}.
Note: When trying to play z(t), you might get a C-contiguous error. To fix this, use z(t) = np.ascontiguousarray(z(t)).
d. Save the resulting signal to a wav file.

C. Load a figure and plot the four signals (x1, w, v, and z) using a 4x1 subplot. Adjust the x and y axis limits to have the same ranges in both plots. Be sure to title the plots and label axes appropriately. For the time axis, you should plot time in sec. Do not use grids on the subplots. Adjust the spacing to avoid overlap. If you have implemented the code correctly, the figure should look something

like this.


train
amplitude
25000
0
−25000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
time
w(t)
amplitude
25000
0
−25000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
time
v(t)
amplitude
25000
0
−25000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
time
z(t)
amplitude
25000
0
−25000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
time

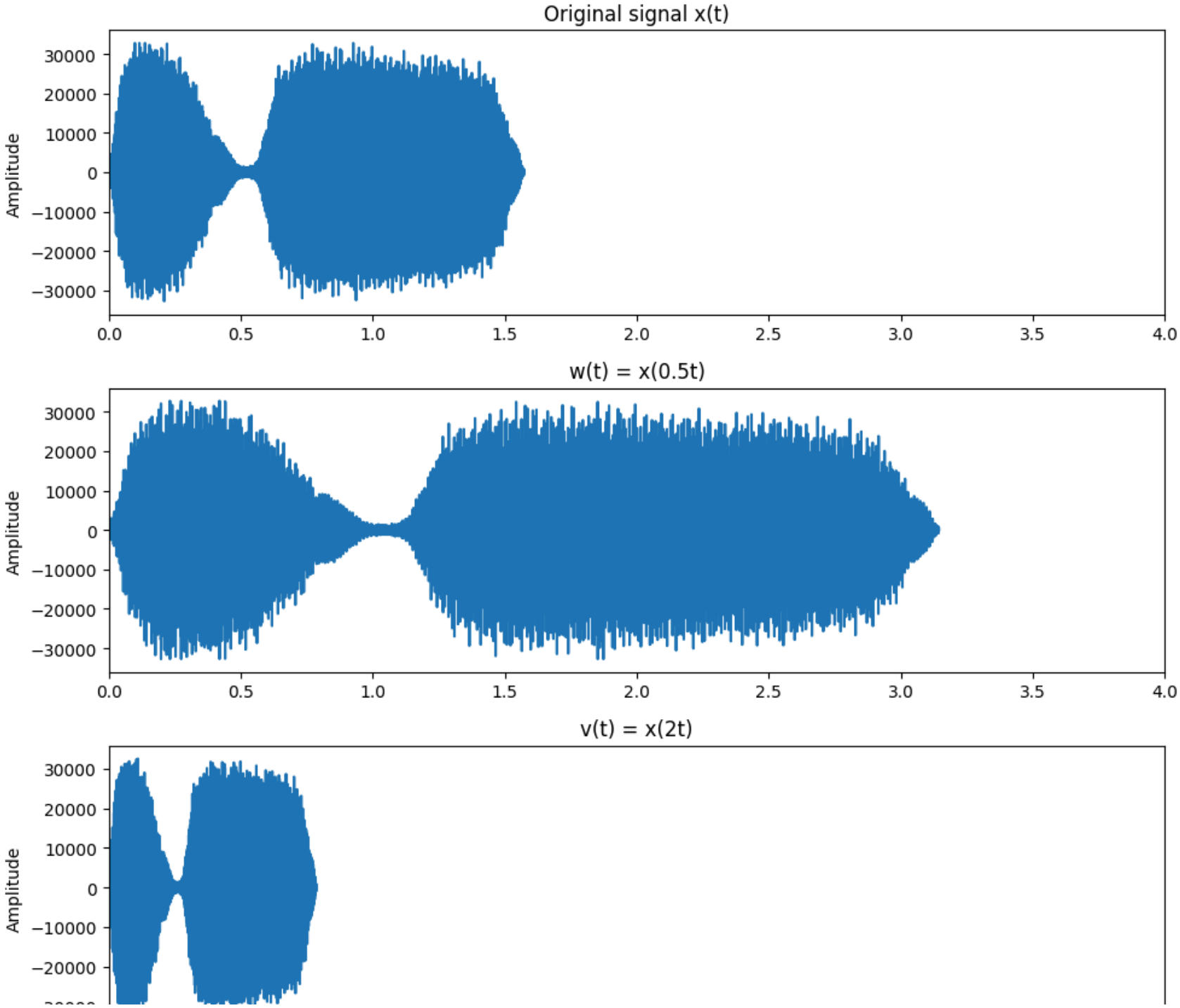
D. Read in the signals you created and play them to verify that they sound different based on what you would expect from these transformations.

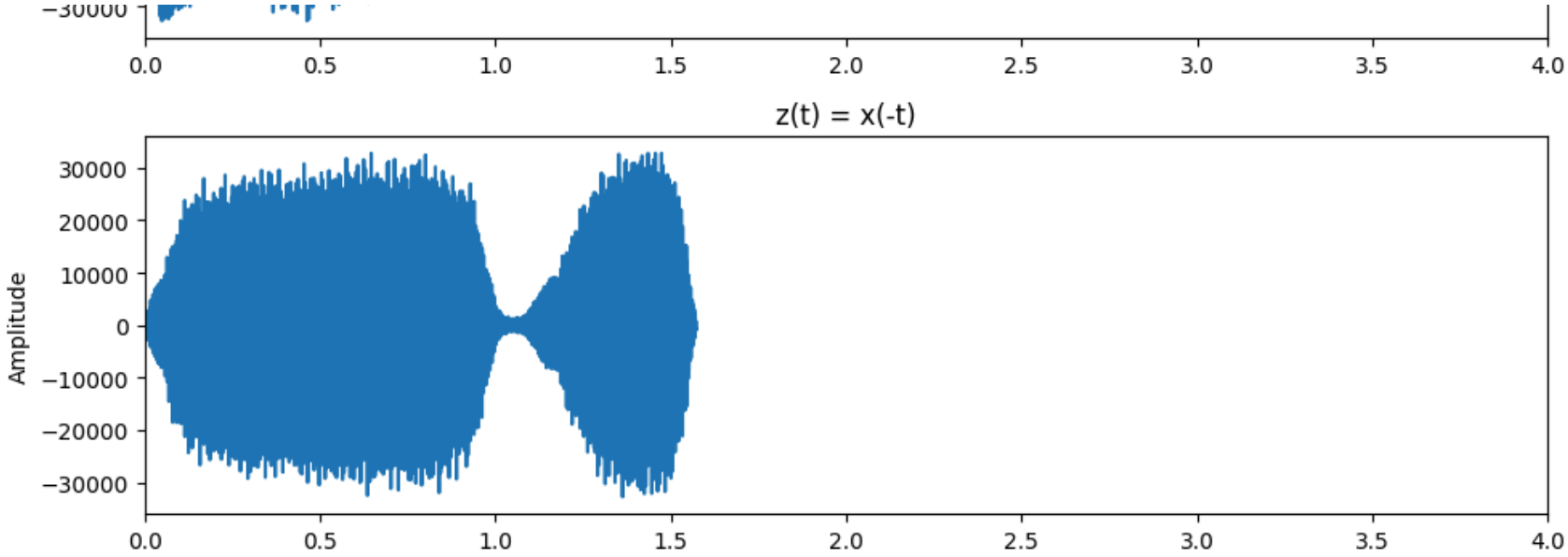
In [99]:
```
# Start with a comment section that explains what the input and output variables are.

# x: input signal vector. This is a 1D NumPy array of samples
# fs: sampling rate (in Hz). This is an integer scalar value.
# a: scaling parameter. This has to be a decimal value for as_integer_ratio to work.
# So, explicitly casting it into a float or a double or any fractional data type will help.
# returns t: the time samples vector corresponding to y: the scaled signal
def timescale(x, fs, a):
    [n, d] = (np.double(a)).as_integer_ratio()
    y = sig.resample_poly(x.astype(float),d,n)
    t = np.arange(0,len(y),1)*(1.0/fs)
    return y.astype("int16"),t

## NOTICE: There is a current bug with scipy.signal that requires casting the input signal to a non-int16/int32 type
## NOTICE: Must cast the array as another datatype, then re-cast back into int16 to play using IPython
```

In [100…
```
##################################################################################################
# PART A
##################################################################################################

# Read the train.wav file
fs_x, x = wav.read("train32.wav")

##################################################################################################
# PART B
##################################################################################################

# Apply the timescale function with a=0.5 and a=2 on x(t) to create w(t) and v(t)
w, t_w = timescale(x, fs_x, 0.5)
v, t_v = timescale(x, fs_x, 2.0)

# Reverse the original signal x1(t) to get z(t)
z = np.flip(x, axis=0)
fs_z = fs_x

# Save all of them to slow.wav, fast.wav and reverse.wav respectively
wav.write('slow.wav', fs_x, w.astype(np.int16))
wav.write('fast.wav', fs_x, v.astype(np.int16))
```

```
wav.write('reverse.wav', fs_x, z.astype(np.int16))

##########################################################################################
# PART C
##########################################################################################

# Plot the figures in a proper manner
fig, axs = plt.subplots(4, 1, figsize=(10, 12))

# Plot the original signal.
axs[0].plot(np.arange(len(x))/fs_x, x)
axs[0].set_title('Original signal x(t)')
axs[0].set_ylabel('Amplitude')
axs[0].set_xlim(0, 4)

# Plot the w(t) signal.
axs[1].plot(t_w, w)
axs[1].set_title('w(t) = x(0.5t)')
axs[1].set_ylabel('Amplitude')
axs[1].set_xlim(0, 4)

# Plot the v(t) signal.
axs[2].plot(t_v, v)
axs[2].set_title('v(t) = x(2t)')
axs[2].set_ylabel('Amplitude')
axs[2].set_xlim(0, 4)

# Plot the z(t) signal.
axs[3].plot(np.arange(len(z))/fs_z, z)
axs[3].set_title('z(t) = x(-t)')
axs[3].set_ylabel('Amplitude')
axs[3].set_xlim(0, 4)

# Organize layout of the plots.
plt.tight_layout()
plt.show()

##########################################################################################
# PART D
##########################################################################################

# Read and play the four files using IPython package as before.
```

```
display(Audio(x, rate = fs_x))
display(Audio(w, rate = fs_x))
display(Audio(v, rate = fs_x))
display(Audio(z, rate = fs_x))
```


Original signal x(t)
w(t) = x(0.5t)
v(t) = x(2t)
Amplitude
30000
20000
10000
0
−10000
−20000
−30000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0



0:00 / 0:01

0:00 / 0:03

0:00 / 0:00

0:00 / 0:01

Discussion Question

Suppose a student runs the figure command before every call to subplot. When you run your script, what changes do you expect to see? How will the plots change?

Answer

The figure command creates a new MatPlotLib figure *each time* that it is called. Creating a new figure for each subplot would create many individual graphs instead of one unified set of graphs. If we instead redefined our figure variable each time instead of adding new ones, this would overwrite our previous graphs. **The plots of our signals themselves won't change.**

Assignment 4 -- Time Shift Operations

We will now implement and test a timeshift function. This assignment will have four parts, A-D, each of which should be indicated with comments, following the guidelines in the Lab 1 template.

A. Write a function called timeshift that takes as input: a signal x, the sampling frequency f_s (in Hz), and a time shift t_0 (in seconds). The function should implement $y(t) = x(t + t_0)$ and produce as output the portion of the shifted signal starting at time 0. Assume that the original signal has value zero outside the time window. Your function should:

1. Find the integer shift n_0 given t_0 and f_s.
2. Use conditional control that tests whether the time shift is positive or negative
2.1. For a time delay, create y by moving the values of the samples to the right. (if t_0 is negative, then move the signal to the right)
2.2. For a time advance, create y moving the values of the samples to the left. (if t_0 is positive, then move the signals to the left)
3. Based on the length of the final signal and the sampling frequency, create a time vector that corresponds to the output signal length, starting at 0.
4. Return the new signal and the time vector. Save the function it in its own cell, as indicated in the Lab 1 template.

B. Use the function timeshift to create x1(t+0.5) and x1(t-2). Plot the shifted signals with the original in a 3x1 plot: x1(t), x1(t+0.5), and x1(t-2). The x-axis should be between 0 and 4 for all three plots. Label axes and title the plot.

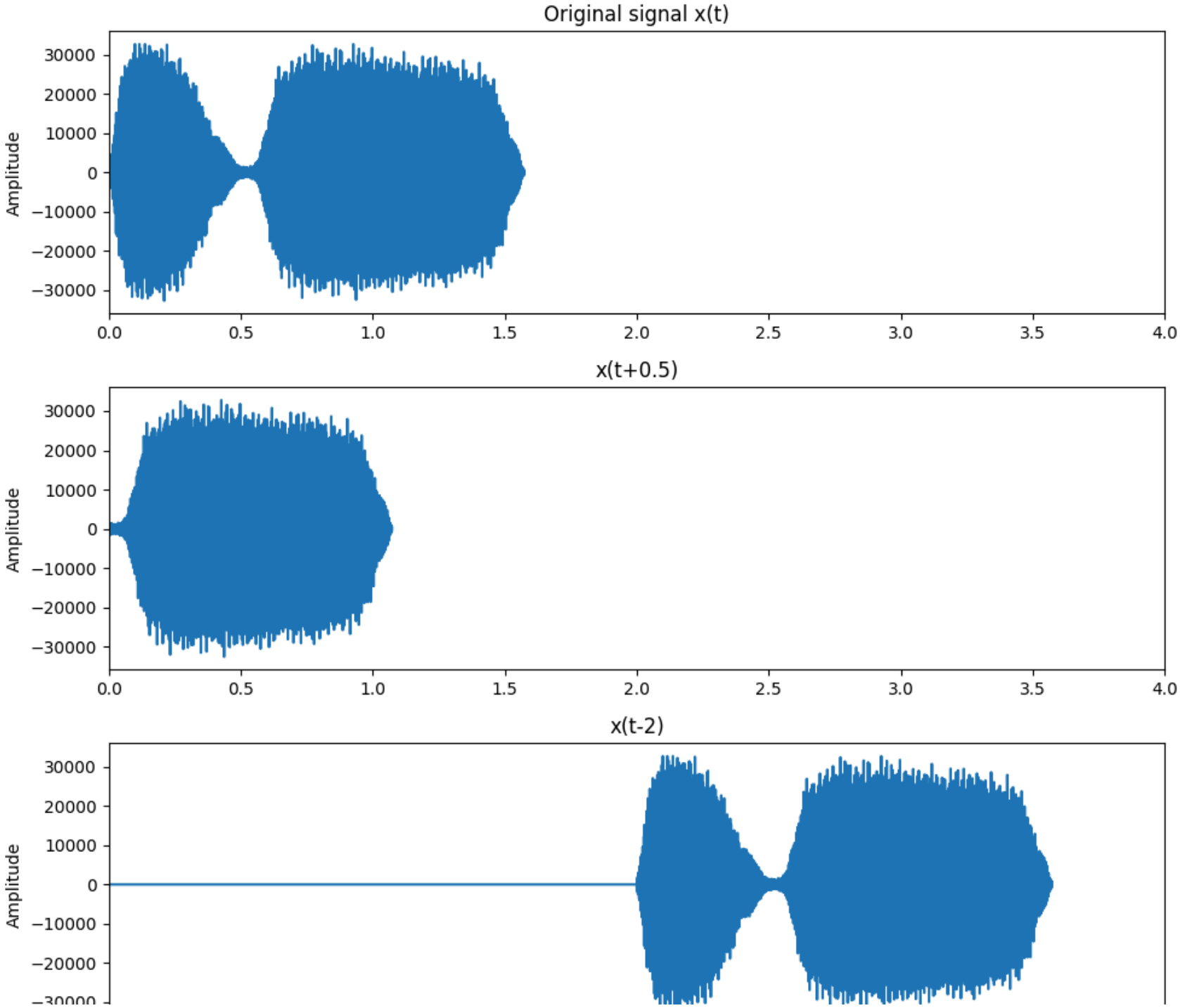
C. Play all three signals.

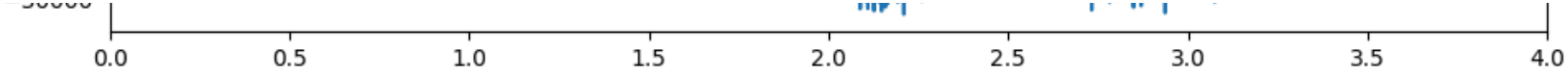
In [101…

```
##############################################################################################
# PART A
##############################################################################################

def timeshift(x, fs, t_0):
    """
    Shifts the input signal x by t_0 seconds.
```

```
    Parameters:
        x: input signal vector (1D NumPy array)
        fs: sampling rate (in Hz) (integer scalar)
        t_0: time shift in seconds (float)

    """
    # Calculates the number of samples to shift.
    shift_samples = int(np.abs(t_0 * fs))

    # Create an empty array for the output signal
    blank = np.zeros(shift_samples, dtype = np.int16)

    # Shifts the signal by either concatenating zeros or taking form
    # the end of the array (if shifting left)
    if t_0 <= 0:
        x = np.concatenate((blank, x), 0)
    else:
        if x.shape[0] < shift_samples:
            x = np.array([]) # Returns a blank array if the shift is larger than the signal.
        else:
            x = x[shift_samples:] # Otherise, it shifts the signal to the left.

    # Determine the timestep and return as tuple
    t = np.arange(len(x))/fs
    return x.astype(np.int16), t

##########################################################################################
# PART B
##########################################################################################

# Use the timeshift function implemented above with values of t_0 = 0.5 and -2. Save the signals.
y1, t1 = timeshift(x, fs_x, 0.5)
y2, t2 = timeshift(x, fs_x, -2)

# Create a figure with 3 subplots
fig, axs = plt.subplots(3, 1, figsize=(10, 9))

# Plot the original signal
axs[0].plot(np.arange(len(x))/fs_x, x)
axs[0].set_title('Original signal x(t)')
axs[0].set_ylabel('Amplitude')
axs[0].set_xlim(0, 4)
```

```
# Plot the advanced signal (t+0.5)
axs[1].plot(t1, y1)
axs[1].set_title('x(t+0.5)')
axs[1].set_ylabel('Amplitude')
axs[1].set_xlim(0, 4)

# Plot the delayed signal (t-2)
axs[2].plot(t2, y2)
axs[2].set_title('x(t-2)')
axs[2].set_ylabel('Amplitude')
axs[2].set_xlim(0, 4)

# Adjust the layout
plt.tight_layout()
plt.show()

##############################################################################################
# PART C
##############################################################################################

# Play the three signals
display(Audio(x, rate=fs_x))
display(Audio(y1, rate=fs_x))
display(Audio(y2, rate=fs_x))
```


Original signal x(t)
x(t+0.5)
x(t-2)
Amplitude
30000
20000
10000
0
−10000
−20000
−30000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0



0:00 / 0:01

0:00 / 0:01

0:00 / 0:03

Discussion Question

There is a trivial case that you should ideally test for. If the shift is zero, then the output is the original signal. If the shift is an advance bigger than the original signal, then the output will be zero. Comment on whether your current implementation correctly handles these cases and whether there is a better implementation.

Answer

Our implementation **correctly handles** both the case where the *shift is zero* and the case where the *shift is an advance bigger* then the current signal duration. In order to handle the situation when the shift is zero, we simpily return the original signal. To handle the situation when the shift is an advance bigger then the current duration of the signal we simpily cap the maximum slicing index to the size of the original signal.