UW GIX - MSTI Program

TECHIN 513 HW 1 – Modifying Signals

In this HW, you will 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 two-week HW. You should plan on completing the first 2 assignments in the first week.

HW 1 Turn-in Checklist

- HW 1 Jupyter notebook or colab notebook file as pdf with code, outputs, and comments for the first 4 exercises assignment in separate cells. Use the template provided to you on canvas.

Note: The pre-HW should be done before you start on these assignments

Assignments

This HW has 4 exercises to be completed individually. 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*, and *matplotlib.pyplot*.

Assignment 1: Working with Sound Files

Start a new cell following the guidelines in the **HW 1 template**, dividing it into Parts A-C.

A. Download the **train.wav** sound file provided. Using the *scipy.io wavfile* package (see Background document), read in the file using the saving the audio vector and sampling frequency in variables **x1** and **fs1,** respectively. Print the sampling rate (which should be 32kHz) and the shape of **x1,** which will tell you the length and number of channels.

B. Write out two new versions of the file in wav format using different sampling rates: **fs2=fs1/2** (16 kHz) and **fs3=1.5*fs1**. Note that the wav.write function requires the sampling frequency to be an integer. Also, the simpleaudio package is limited in the sa.mpling frequencies that it will support, so you cannot use arbitrary frequencies.

C. Using the *simpleaudio* package (see Background document), read in the three different versions of the train sound file and play each one.

Report discussion: Comment on how the audio changes when the incorrect sampling frequency is used.

Assignment 2: Amplitude Operations on Signals

Again, following the guidelines in the **HW 1 template**, start a new cell and write a script to meet the following specifications. This assignment will have four parts, A-C, each of which should be indicated with comments.

A. Create a discrete time signal **s1** that is the same length as **x1** and has value 1 for t=[0,0.5] and value 0.2 for t>0.5. You can use the command below where len1 is the length of x1 and n0 is the index corresponding to t=0.5

   s1 = np.concatenate((np.ones(n0),0.2*np.ones(len(x1)-n0))

   Multiply **x1** with **s1** to create **v1.** Save this signal to a wav file.

B. Create a discrete-time decaying ramp signal **r1**, that is the same length as **x1**. The signal should have value 1 at time 0 and linearly decay to value 0. (Hint: use numpy.arange.) Multiply **x1** with **r1** to create **v2.** Save this signal to a wav file.

C. Read in **v1** and v**2** using simpleaudio and play the two different modifications together with the original, to verify that the volume of the second whistle is reduced.

Report discussion: Discuss the differences that the two modifications have on the signal. What would happen if you defined s1 to take value 2 for the [0,0.5] range? If you wanted a smooth but faster decay in amplitude, what signal might you use?

Assignment 3: Time Scaling Audio Signals

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

A. Replicate the **timescale** function in the HW 1 background document, and save it in its own cell, as indicated in the **HW 1 template**.

B. Use the timescale function to obtain **w(t)=x1(2t)** and **v(t)=x1(0.5t)**

   a. Create **w(t)** using **a=2** and store the outputs of the **timescale** function as **w** and **t_w.**

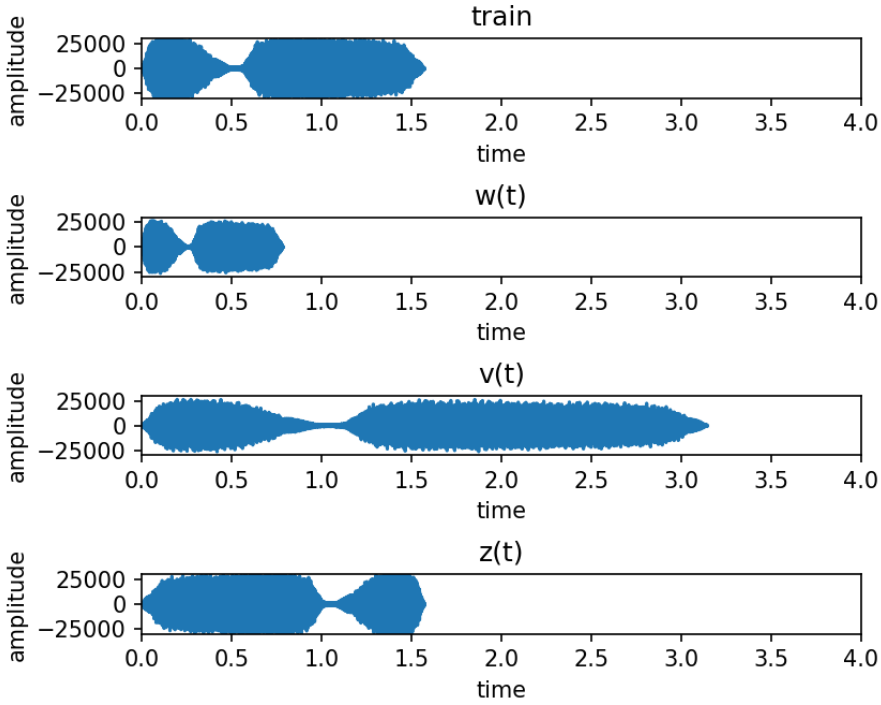
   b. Create **v(t)** using **a=0.5** and store the outputs of the **timescale** function as **v** and **t_v.**

   c. Create a time-reversed version of the signal **z(t)=x1(-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_x1.**

      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 HWel axes appropriately. For the time axis, you should plot time in msec, for which you will need to multiple the time array by 1000 as in the example in the Background document. 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.



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

Report discussion: 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?

Assignment 4: Time Shift Operation

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 **HW 1 template**.

A. Write a function called **timeshift** that takes as input: a signal **x**, the sampling frequency **fs** (in Hz), and a time shift **t0** (in seconds). The function should implement y(t) = x(t+t0) 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 **n0** given **t0** and **fs**.

2. Use conditional control that tests whether the time shift is positive or negative
    1. For a time delay, create **y** by concatenating a zero vector with the original signal. (The output should be longer than the original signal.)
    2. For a time advance, create **y** by copying the portion of the starting from **n0** and then appending **n0** zeroes at the end of the signal. (The output should be the same length as the original signal.)
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 **HW 1 template**.

B. Use the function 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. HWel axes and title the plot.

C. Play all three signals. For the signal that has been advanced, you should be able to hear that part of the sound is missing, since we have not preserved those samples.

Report discussion: 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.

Bonus Problem: Create Your Own Sound Mash-Up

Open a new cell in the **HW 1 template**. Download 1-2 new sounds, which could be from the extras provided, from some open repository or from your own collection. Keep them short. If you choose a sound that has 2 channels, extract audio data from only one channel.

Using the code developed in the exercises above, create a new signal that makes use of your sounds and four signal modifications: **addition, multiplication, time scaling and time shifting**. You can also concatenate sounds, and if you want to insert a brief silence between sounds then just use a vector of zeroes. Remember that when you add or multiply two signals, they must be the same length. Also, be careful not to increase the amplitude too much, since it can result in clipping. For example, if you add two signals, you may want to scale the result by 0.5. Save the result as a wav file (no more than 20 sec in length) and upload to canvas with your notebook alongside with the original sounds you picked.