3
Convolution laboratory - II

3.1 Purpose

In Laboratory #2, you wrote a program to perform convolution of sequences in which both $x[n]$ and $h[n]$ were of finite length. In this exercise, you will deal with the situation in which $x[n]$ is of unknown, possibly infinite length.

3.2 Background

To prepare for this exercise, review Chapter 3, Section 3.7, which covers the overlap-add and overlap-save methods.

3.3 Assignment

You will write a Matlab function to convolve two sequences using both the overlap-add and overlap-save methods. For this assignment, we will not bother with the sequence structures we used in the previous laboratories. Instead, we will assume that $x[n]$ and $h[n]$ are standard Matlab sequences. Here are the function headers:

```
function y = overlap_add(x, h, lc)
% OVERLAP_ADD  Convolve x and h using overlap-add method
%              y = overlap_add(x, h, lc)
%              x and h are arrays,
%              lc is the chunk size (default 50)
```

and

```
function y = overlap_save(x, h, lc)
% OVERLAP_SAVE  Convolve x and h using overlap-save method
%              y = overlap_save(x, h, lc)
%              x and h are arrays,
%              lc is the chunk size (default 50)
```

3.3.1 Notes on the assignment:

$h[n]$ will be of reasonable length, containing no more than 50 samples. However, $x[n]$ can be very, very long. The whole point of this exercise is to figure out how to cut up $x[n]$ in order to convolve it with $h[n]$. A good way to get a very long $x[n]$ is to read in and convert a .wav file. One such file is seashell.wav, which is included in lab2.mat, which you can download from the website. Once you have downloaded this file, you can get Matlab to extract the data from the file using the `wavread` or `audioread` function (depending on your version of Matlab) and play it using the `soundsc` function:

```
» [x, fs] = wavread('seashell');
» soundsc(x, fs)
```

To see what filtering convolution (a.k.a. FIR filtering) with simple $h[n]$ sounds like, try the following:

```
» soundsc(conv(x, [1 -1]), fs); % this is a simple high-pass filter
» soundsc(conv(x, [1 1 1 1 1 1 1]), fs); % this is a simple low-pass filter
```

Later in the term, we'll learn how to design filters for specific purposes, but for now, notice how convolution is used to implement filtering with these simple $h[n]$.

Internally, your program should cut the long $x[n]$ into 'input chunks' of size, N, and convolve each chunk with $h[n]$ to form output chunks, then assembling these output chunks using the overlap-add and overlap-save methods. Once you have created each output chunk, you can assemble the output sequence by a couple of methods:

- Create your entire output array, `output_array`, in advance, for example using the `zeros` function, and then copy each chunk into the appropriate place in the output array. For example, the input array, `input_array`, which is of size `N`, can be copied efficiently to the `i`th chunk of an `output_array` as follows:

```
indx = 1 + (i - 1)* N;
output_array(indx:indx+N-1) = input_array;
```

- Instead of creating the entire array in advance, you can just append a given chunk to the end of the current output array as follows:

  ```
  output_array = [output_array chunk];
  ```

  However, this method is less efficient and slower because every time you do this, Matlab must internally reallocate a new `output_array` and then delete the previous `output_array`.

Your chunk size, `N`, should be a maximum of 500 points. Should it be smaller? Why or why not? For starters, try the something simple for $h[n]$, for example, [1 -1] or [1 1 1 1 1 1 1]. You should not assume that the length of $x[n]$ is a nice multiple of 500. You may have to pad the last frame of data with zeros (how do you do this?) You *may* use the `length` or `size` command to figure out the length of the input. Of course, in a real application, you would not know this information.

Since the main point of this exercise is to teach you about the overlap-save and overlap-add methods of convolving infinite-length sequences, you don't have to write your own convolution function. Instead, you may use MATLAB's `conv` function to do the convolution of the chunks. You'll notice that the first and last few points of the entire convolution (how many?) will be "bogus", in the sense that they ramp up from zero at the beginning and ramp down to zero at the end. You should include these ramp sections so that your functions match the output of Matlab's `conv` function exactly. There are at least a couple of ways to do this:

- Put an `if` statement in the `for` loop that handles the input chunks so that it handles the first and last chunks differently; namely, that it retains the first points of the first frame and the last points of the last frame instead of tossing them.
- Pad the beginning and end of the *entire* input sequence with an appropriate number of zeros (how many?).

The test of whether you do your convolutions correctly is to check against Matlab's `conv` function. Download lab3.m and test_lab3.p from the website. Also download lab2.mat from the website (same as the previous lab), which contains the seashell wavefile as well as the two filters, and place all of them in your Matlab working directory. Then, in the Matlab command window, type

```
» load lab2
```

Then type

```
» publish lab3
```