3

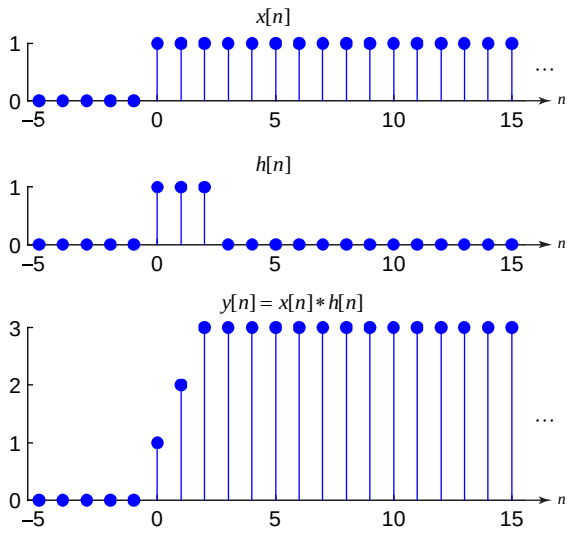
Convolution laboratory - II

3.1 Purpose

In the previous laboratories 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 a real-world situation in which $x[n]$ is of unknown, possibly infinite length.

3.2 Background

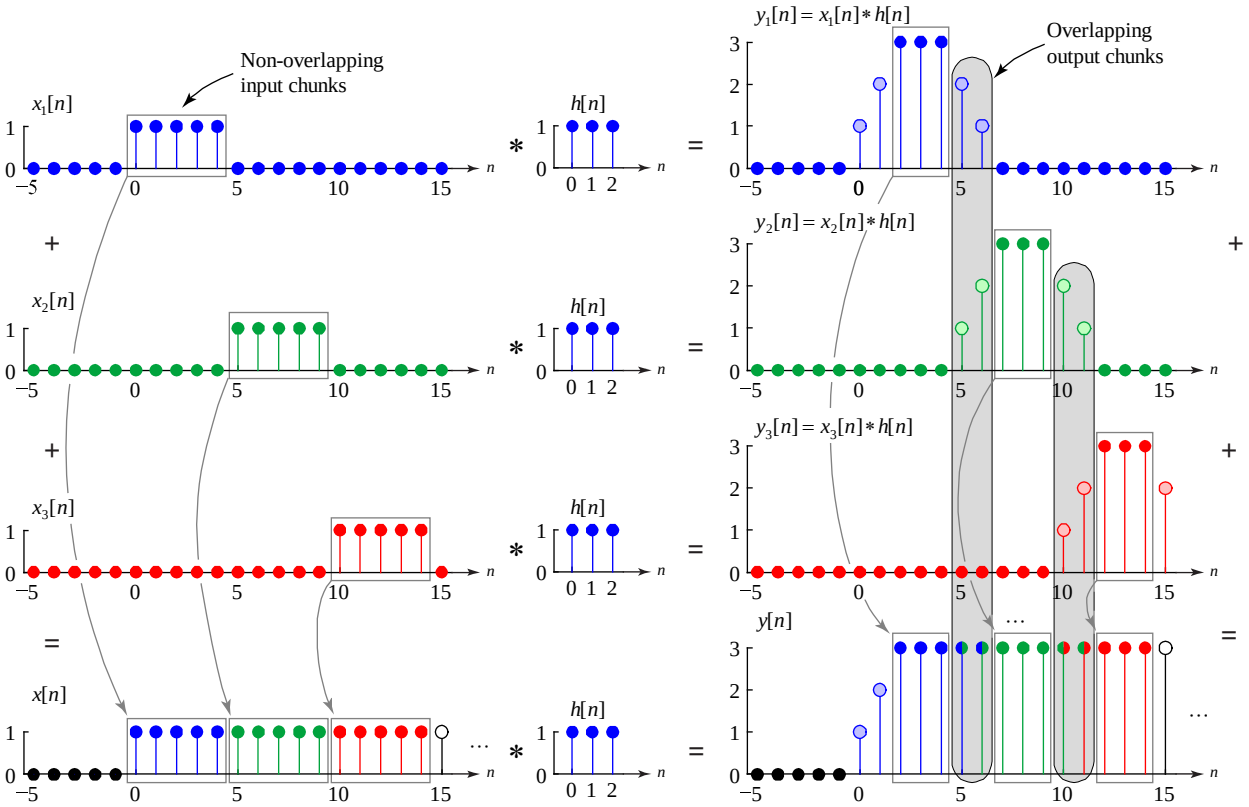
Assume we want to convolve the sequences $x[n]$ with impulse response $h[n]$. $h[n]$ is assumed to be finite length, but $x[n]$ could be very long, as shown in the following figure



There are two methods two deal with infinite-length sequences, termed the *overlap-add* and the *overlap-save* methods.

Overlap-add method

The following picture shows the basic scheme of the overlap-add method:



In the overlap-add method, the signal $x[n]$ is cut into finite, *non-overlapping* input chunks (subsequences): $x_1[n]$, $x_2[n]$, $x_3[n]$,… so that $x[n] = x_1[n] + x_2[n] + x_2[n] + \cdots$. Each chunk is then convolved with the impulse response, $h[n]$, to form output chunks $y_1[n] = x_1[n] * h[n]$, $y_2[n] = x_2[n] * h[n]$,…. By the distributive property, when these output chunks are added together, they produce $y[n]$:

$$\begin{aligned} y[n] = x[n] * h[n] &= \left(x_1[n] + x_2[n] + x_3[n] + \cdots\right) * h[n] \\ &= x_1[n] * h[n] + x_2[n] * h[n] + x_3[n] * h[n] + \\ &= y_1[n] + y_2[n] + y_3[n] + \cdots \end{aligned}$$

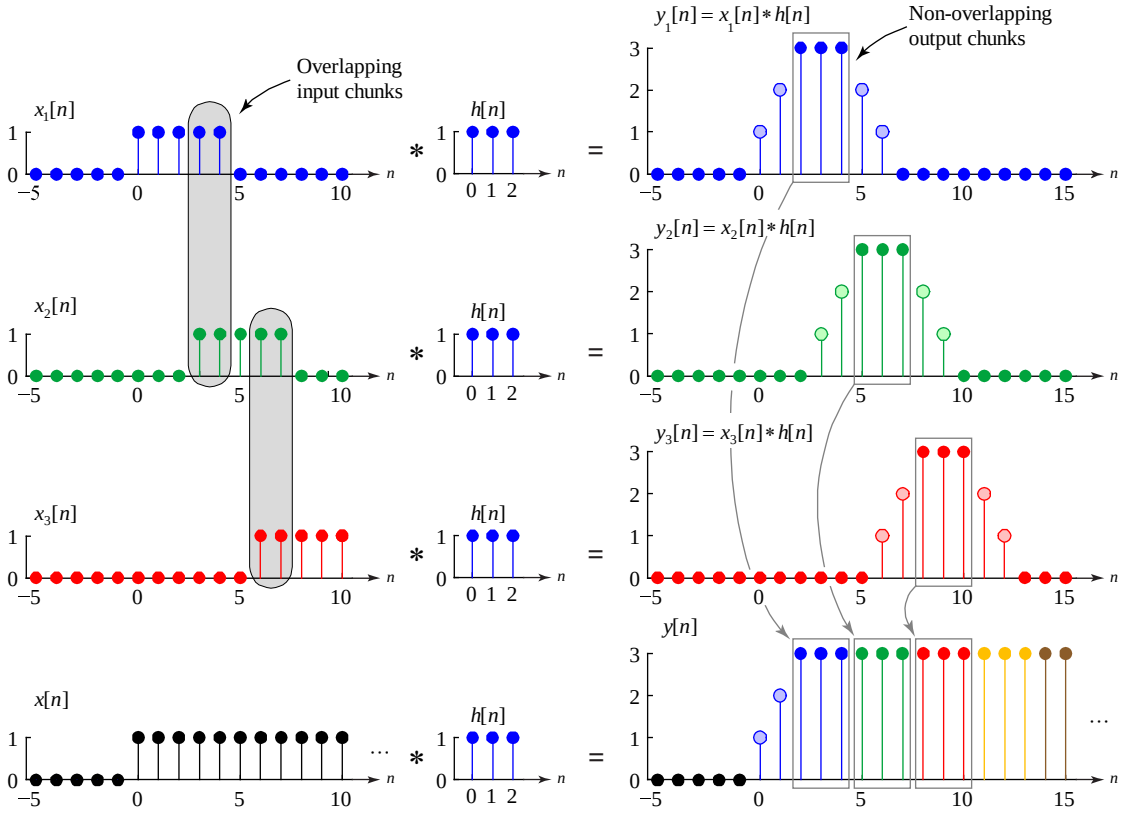
If each input chunk is of length N and the impulse response is of length M, then each output chunk is of length $P = N + M - 1$. In the example figure, $N = 5$, $M = 3$ and so $P = 7$. The input sequences are *non-overlapping* in time, while the output subsequences are *overlapping* in time, as marked by the grey ovals. Whereas some of the points in $y[n]$ (denoted by the single solid colors) derive from a single output chunk, other points in $y[n]$ (marked with two colors) require the summation of values from two successive output chunks. Specifically, the center $N - M + 1$ points of each output chunk, indicated by the solid dots in $y_1[n]$ $y_2[n]$ and $y_3[n]$, contribute directly to the corresponding points in the output sequence, $y[n]$. For example, $y[4] = y_1[4]$. However, the last $M - 1$ points of each chunk must be added to the beginning $M - 1$ points of the subsequent chunk to form points in the output sequence. For example,. $y[5] = y_1[5] + y_2[5]$.. If the input subsequences are adequately long with re-

spect to the length of $h[n]$, no points in $y[n]$ requires summation of data from no more than two output chunks.

From the point of view of implementation, the problem with the overlap-add method is that we must add points from pairs of output subsequences to get some points in the final output array, hence we must store parts of two output subsequences in memory.

Overlap-save method

The following picture shows the basic scheme of the overlap-save method:



First, the signal $x[n]$ is segmented into *overlapping* input chunks $x_1[n], x_2[n], x_3[n], \ldots$ as shown in the left column of the figure. The points of each chunk that are overlapped with the following subsequence are marked with grey ovals. Each input chunk is then convolved with $h[n]$ to form output subsequences $y_1[n], y_2[n], y_3[n], \ldots$ where $y_1[n] = x_1[n] * h[n]$, $y_2[n] = x_2[n] * h[n]$ and so on. In the overlap-save method, we only choose the points from the output subsequences for which are 'good', that is, values of n for which the convolution of $h[n]$ and the corresponding input subsequence (e.g. $x_1[n]$) is exactly the same as the convolution of $h[n]$ and the original sequence $x[n]$. For example, consider the convolution $y_1[n] = x_1[n] * h[n]$. When $n = 2$, you can see that $y_1[2] = x_1[2]h[0] + x_1[1]h[1] + x_1[0]h[2]$. Since $x_1[2] = x[2]$, $x_1[1] = x[1]$ and $x_1[0] = x[0]$, thus $y_1[2]$ equals $y[2]$. The same conclusion applies to the next two output points from this

chunk: $y_1[3] = y[3]$ and $y_1[4] = y[4]$. The final two points of this chunk are not usable since $y_1[5] \neq y[5]$ and $y_1[6] \neq y[6]$. However, looking at the next chunk, you can see that $y_2[5] = y[5]$, $y_2[6] = y[6]$ and $y_2[7] = y[7]$. Therefore, we can get can construct a plete $y[n]$ by taking only the appropriate 'good' points: the center $N - M + 1$ points of each output chunk, indicated by the solid dots in $y_1[n]$ $y_2[n]$ and $y_3[n]$.

Here's a question for you: determine by how many points the subsequences $x_1[n], x_2[n], \ldots$ must overlap so that $y[n]$ can be constructed from the 'good points' of, $y_2[n]$, …? Hint: the number depends on the length of $h[n]$, right?

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* and *h* 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)
```

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 lab3.zip. Once you have downloaded this file, you can get Matlab to extract the data from the file using the `audioread` function and play it using the `soundsc` function:

```
» [x, fs] = audioread('seashell.wav');
» soundsc(x, fs)
```

Also included in the zip file are the impulse responses of a couple of FIR filters. fir_lp is a relatively sharp lowpass filter with an equivalent analog bandwidth of about 700 Hz. fir_hp is a highpass filter with a cutoff at about 1400 Hz. To hear what filtering speech $h[n]$ sounds like, try the following:

```
» soundsc(conv(x, fir_hp]), fs); % this is a simple high-pass fil-
ter
» soundsc(conv(x, fir_lp), fs); % this is a simple low-pass filter
```

Cool, huh? Also included in the zip file is the file tones.wav, which is the sum of pure two tones, one at 400 Hz, the other at 1600 Hz. Now try filtering this file with the two filters. You should hear that the filters separate the two tones completely. 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 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 to the `i`th chunk of an `output_array` as follows:

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

  This is the most efficient method.

- 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];
  ```

  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 same $h[n]$ we used above, namely [1 -1] and/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 routine (yeah!). Instead, you may use MATLAB's `conv` routine 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". You should include them, in order to match Matlab's `conv` function exactly. There are at least a couple of ways to do this:

- You can 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.
- If you have written your routine correctly, it will basically 1) chunk the input data, 2) do the convolution and 3) toss the first and last few points of each chunk. If you have done this, then you can pad the beginning and end of the *entire* input sequence with an appropriate number of zeros (how many?). Then, when the first points of the first chunk and the last points of the last chunk are tossed, you will be assured that the beginning and end of the output array will match Matlab's `conv` function.

The test of whether you do it correctly is to check against Matlab's `conv` function. Download lab3.zip from the website. It contains `lab3.m`, `test_lab3.p` and `lab3.mat`, which includes the seashell wav file as well as the impulse responses of the two filters, and place all of them in your Matlab working directory. Then, in the Matlab command window, type

```
» load lab3
```

Then type

```
» publish('lab3', 'pdf')
```