2
Convolution laboratory - I

2.1 Purpose

In this lab, you will investigate the properties of convolution, and use convolution to understand what happens when systems process signals.

2.2 Background

To prepare for this exercise, review Chapter 2, Sections 3.1, 3.2, 3.3, 3.5 and 3.6. Here follows a brief summary.

2.2.1 Convolution

A linear time-invariant systems is completely characterized by its impulse response, $h[n]$. Given an input, $x[n]$, and the impulse response, the output of the linear system, $y[n]$, is given by the convolution sum,

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]. \tag{L2.1}$$

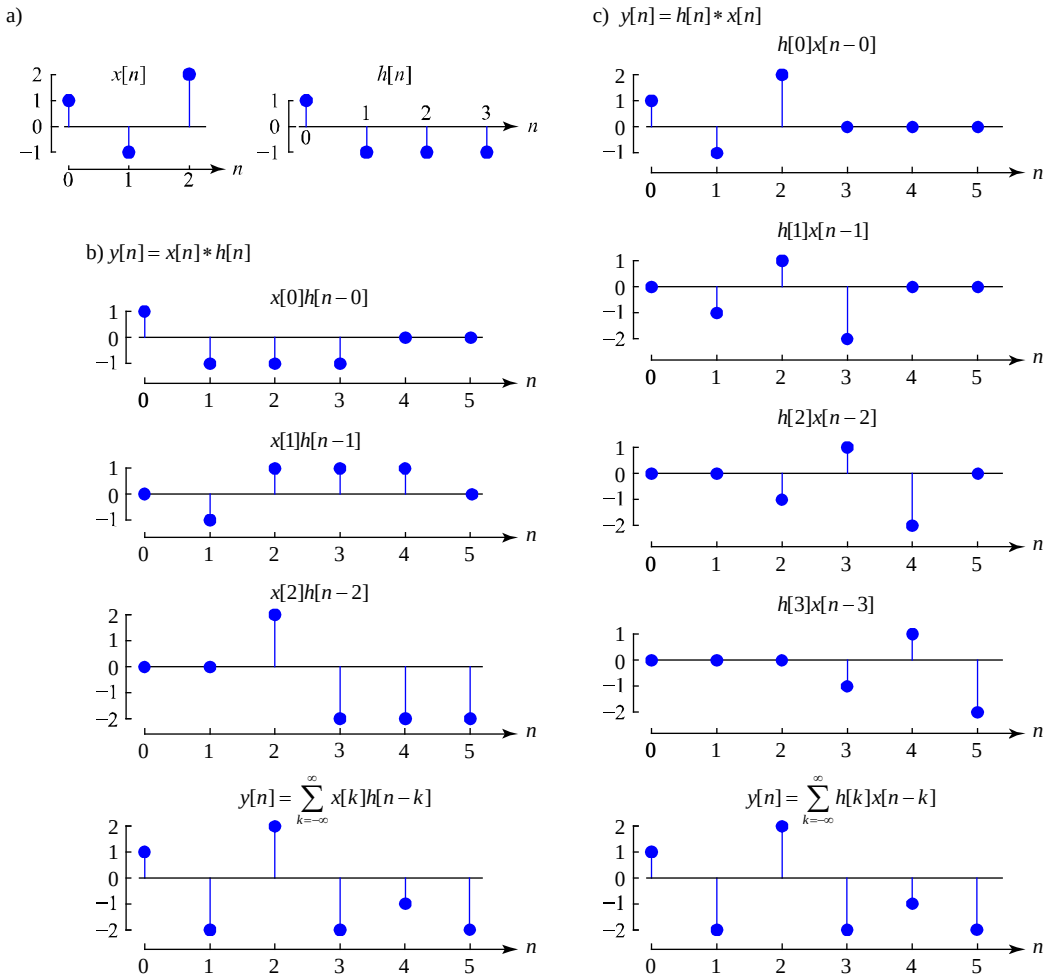

Figure 1: Convolution by the direct summation method

2.2.2 Properties

Convolution is an operator, in the same sense that multiplication and addition are operators. It satisfies commutative, associative and distributive properties.

The commutative property

Commutativity says the order of successive convolutions doesn't matter. That is

$$x[n] * h[n] = h[n] * x[n]$$

As an example, Figure 1a shows a couple of simple sequences, $x[n]$ and $h[n]$. The application of the direct summation method of convolution to these two sequences using Equation **Error! Reference source not found.** is shown in Figure 1b. The first three panels of this figure correspond to $h[n]$ shifted by 0, 1 and 2 samples and multiplied by $x[0]$, $x[1]$ and $x[2]$ respectively. The bottom panel is the sum of the scaled and shifted sequences, which form the output, $y[n]$. In this example, you have to sum three sequences of length six (including zero padding at the beginning and/or end of each sequence) to get the $y[n]$.

Q: Why do we have to pad these sequences with zeros?
Q; In general, how does the number of sequences we have to sum and the length of these sequences depend on the sizes of $x[n]$ and $h[n]$?

As an illustration of the commutative property, Figure 1c shows the convolution, $y[n] = h[n] * x[n]$, of the sequences of **Error! Reference source not found.** using the direct summation method. The first four panels of this figure correspond to $h[n]$ shifted by 0, 1, 2 and 3 samples and multiplied by $x[0]$, $x[1]$ and $x[2]$ respectively. The bottom panel is again the sum of the scaled and shifted sequences, the output, $y[n]$. This result is the same as in Figure 1b. However, to get this result, we have to sum four sequences of length six rather than summing three sequences of length six.
Q: Why the difference?

The associative property
Associativity says the grouping of successive convolutions doesn't matter. That is

$$\left(x[n] * h_1[n]\right) * h_2[n] = x[n] * \left(h_1[n] * h_2[n]\right)$$

The distributative property
Distributivity says that

$$\left(x[n] * h_1[n]\right) + \left(x[n] * h_2[n]\right) = x[n] * \left(h_1[n] + h_2[n]\right)$$

2.2.3 Implementation issues

Let's look at how you can implement the convolution in Matlab by two methods: matrix multiplication and point-by-point.

Matrix multiplication.
Consider implementing the convolution, $y[n] = x[n] * h[n]$, of the two sequences in Figure 1a by matrix multiplication. In Matlab we can create a Toeplitz matrix each of whose rows corresponds to $h[n-k]$ at one value of n, each row padded with the appropriate number of zeros at the beginning and end so that all rows are the same length as the length of $y[n]$. Here is the matrix,

$$\mathbf{H} = \begin{bmatrix} 1 & -1 & -1 & -1 & 0 & 0 \\ 0 & 1 & -1 & -1 & -1 & 0 \\ 0 & 0 & 1 & -1 & -1 & -1 \end{bmatrix}. \qquad \text{(L2.2)}$$

Now, multiply $\mathbf{H}$ by $\mathbf{x}^T$, a row vector that represents $x[n]$,

$$\mathbf{x}^T = \begin{bmatrix} 1 & -1 & 2 \end{bmatrix}$$

Then, the output row vector is

$$\mathbf{y}^T = \mathbf{x}^T\mathbf{H} = \begin{bmatrix} 1 & -1 & 2 \end{bmatrix} \begin{bmatrix} 1 & -1 & -1 & -1 & 0 & 0 \\ 0 & 1 & -1 & -1 & -1 & 0 \\ 0 & 0 & 1 & -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -2 & 2 & -2 & -1 & -2 \end{bmatrix}$$

.

The commutative property tells us that we could obtain the same result by convolving $h[n]$ with $x[n]$ as indicated in Figure 1c. Here,

$$\mathbf{X} = \begin{bmatrix} 1 & -1 & 2 & 0 & 0 & 0 \\ 0 & 1 & -1 & 2 & 0 & 0 \\ 0 & 0 & 1 & -1 & 2 & 0 \\ 0 & 0 & 0 & 1 & -1 & 2 \end{bmatrix}, \tag{L2.3}$$

and $\mathbf{h}^T = \begin{bmatrix} 1 & -1 & -1 & -1 \end{bmatrix}$. So,

$$\mathbf{y}^T = \mathbf{h}^T\mathbf{X} = \begin{bmatrix} 1 & -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 2 & 0 & 0 & 0 \\ 0 & 1 & -1 & 2 & 0 & 0 \\ 0 & 0 & 1 & -1 & 2 & 0 \\ 0 & 0 & 0 & 1 & -1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & -2 & 2 & -2 & -1 & -2 \end{bmatrix}.$$

The sizes of the matrices in Equations (L2.2) and (L2.3) are different. When we compute $x[n] * h[n]$ the size of the matrix is 3x6, whereas when we compute $h[n] * x[n]$ the size is 4x6.

Q: Why the difference?
Q: In general, how do the number of rows and columns of matrix depend on the lengths of the sequences $x[n]$ and $h[n]$?
Q: When is it more advantageous to compute $x[n] * h[n]$ and when is it better to compute $h[n] * x[n]$?

2.2.4 Deconvolution

Given a linear time-invariant system has impulse response, $h[n]$ and output, $y[n]$, both of which are assumed to be of finite length, we can find the input, $x[n]$, by the process of deconvolution. Since $y[n] = x[n] * h[n] = h[n] * x[n]$, in matrix terms, $\mathbf{y}^T = \mathbf{x}^T\mathbf{H} = \mathbf{h}^T\mathbf{X}$. Deconvolution of $\mathbf{y}^T$ to obtain $\mathbf{x}^T$ is accomplished by matrix inversion: $\mathbf{x}^T = \mathbf{y}^T\mathbf{H}^{-1}$. Consider the sequences, $h[n]$ and $y[n]$, shown in Figure 2.

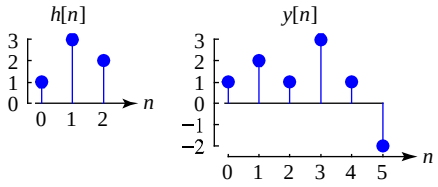

Figure 2: *h*[*n*] and *y*[*n*]

Note that $h[n]$ is of length three and $y[n]$ is of length six. Hence, $x[n]$ must be of length four, and $\mathbf{x}^T$ must therefore be a 1×4 vector. So, if $\mathbf{x}^T = \mathbf{y}^T\mathbf{H}^{-1}$, then $\mathbf{y}^T$ must be a 1×4 vector and $\mathbf{H}$ must be a 4×4 square matrix so that its inverse is 4×4. We can arbitrarily chose the first four values of $\mathbf{y}^T$, and form a 4x4 submatrix of $\mathbf{H}$: $\hat{\mathbf{y}}^T = \mathbf{x}^T\hat{\mathbf{H}}$, where $\hat{\mathbf{y}}^T = \begin{bmatrix} 1 & 2 & 1 & 3 \end{bmatrix}$ and

$$\hat{\mathbf{H}} = \begin{bmatrix} 1 & 3 & 2 & 0 \\ 0 & 1 & 3 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Then,

$$\mathbf{x}^T = \hat{\mathbf{y}}^T\hat{\mathbf{H}}^{-1} = \begin{bmatrix}1 & 2 & 1 & 3\end{bmatrix}\begin{bmatrix}1 & 3 & 2 & 0\\ 0 & 1 & 3 & 2\\ 0 & 0 & 1 & 3\\ 0 & 0 & 0 & 1\end{bmatrix}^{-1} = \begin{bmatrix}1 & 2 & 1 & 3\end{bmatrix}\begin{bmatrix}1 & -3 & 7 & -15\\ 0 & 1 & -3 & 7\\ 0 & 0 & 1 & -3\\ 0 & 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}1 & -1 & 2 & -1\end{bmatrix}$$

We could have chosen any four values of $y[n]$ and done the deconvolution by the appropriate choice of $\hat{\mathbf{y}}^T$ and $\hat{\mathbf{H}}$. For example, choosing the row corresponding to $y[0]$, $y[2]$, $y[4]$ and $y[5]$ would give

$$\mathbf{x}^T = \hat{\mathbf{y}}^T\hat{\mathbf{H}}^{-1} = \begin{bmatrix}1 & 1 & 1 & -2\end{bmatrix}\begin{bmatrix}1 & 2 & 0 & 0\\ 0 & 3 & 0 & 0\\ 0 & 1 & 2 & 0\\ 0 & 0 & 3 & 2\end{bmatrix}^{-1} = \frac{1}{12}\begin{bmatrix}1 & 1 & 1 & -2\end{bmatrix}\begin{bmatrix}12 & -8 & 0 & 0\\ 0 & 4 & 0 & 0\\ 0 & -2 & 6 & 0\\ 0 & 3 & -9 & 6\end{bmatrix} = \begin{bmatrix}1 & -1 & 2 & -1\end{bmatrix}$$

2.3 Your assignment

2.3.1 Questions.

Answer all the questions in red, above.

2.3.2 Convolution function – Matlab style

You are going to write a Matlab function to convolve two sequences objects:

```
function y = conv(x, h)
% CONV Convolve two finite-length Matlab sequence objects, x and h
%          returning sequence object, y.
```

As you can see, we are using the same name for this method as Matlab's `conv` function, which works only on numeric arrays. This is another example of method **overloading**, which is a big advantage of object-oriented programming. In order for Matlab to find your `conv` function, it should be a method placed in the methods block of your *sequence.m* file.

Each of the sequences that we used as input to the `conv` function will, of course, be of finite length. In your function, you will need to pad your sequences appropriately with zeros so that they are the same length; then, we can use Matlab's matrix operators. The sequences will be specified as in Lab#1, namely as sequence objects. For example,

```
» x = sequence([1 2 3 4 5], -1);
```

The point of this exercise is to teach you about convolution. Hence, when you convolve $x[n]$ and $h[n]$, you may *not* use MATLAB's numerical `conv` routine. However, you certainly can and should use it to check your answers.

Here are some points to consider:

- You can use your `flip` and `shift` and `mult` methods from Lab#1 to implement your `conv` function, if you wish, but I don't recommend it. You will find that your program will run faster if you write the program from scratch.
- While `for` loops in Matlab are generally slow, you can use a `for` loop to set up the matrix (L2.2) and/or (L2.3) if you need to.
- When you are convolving two sequences, say `x` and `h`, the main work of your `conv` function is in computing `y.dat=x.dat*h.dat`. Question to ponder: does the computation of `y.dat` depend on the values of `x.off` and `h.off`?
- Speed counts (sort of)!!! In this lab, part of your grade will depend on how fast your `conv` function runs. You should recognize that the way you implement convolution (i.e. whether you do $x[n]*h[n]$ or $h[n]*x[n]$) has a first-order effect on the speed of convolution. When I say, "Speed counts", I *don't* want you to go overboard and try for Matlab perfection; that's not important. The lab output will print out the time your `conv` function takes to do a convolution, as well as the time that it takes for Matlab's highly optimized, machine-coded `conv` function. If your time is within a factor of 8-10 of Matlab's, you're doing O.K.

2.3.3 'Real time' Convolution function

The convolution done in the previous part assumes that you have the entire input sequence, $x[n]$, and can process it using Matlab's vectorized operations. When you have to write a convolution routine in any 'real' programming language (e.g. C or Java or assembly), perhaps for an embedded processor application, then Matlab's vectorized operations aren't available to you. Furthermore, in any real-time application, you don't have access to the entire input sequence – in fact the input sequence may be of infinite duration. Rather, you get the input one point at a time, perhaps as the result of an A/D conversion occurring at a constant rate. We will assume that for each point of the input, we require that the application produce one point of the output sequence, $y[n]$.

To do a real-time convolution of an input with an impulse response, $h[n]$, we need to write an efficient routine that does the minimum number of multiplications and additions and requires the minimum number of storage elements. Consider a convolution of $x[n]$ with $h[n]$ where $h[n]$ has N points. For the n^{th} point of the output, we have to compute the convolution sum,

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k] .$$

So, we have to do the N multiplications, $N-1$ additions. Since we only get one new value of $x[n]$ at each time point, we have to keep the past $N-1$ values of $x[n]$ in memory (as well as the current value of $x[n]$). Hence, we need a buffer of N memory elements (plus whatever extra memory is necessary to accumulate the convolution sum).

Your job is to write a second convolution function, `conv_rt`, in Matlab that basically implements a real-time convolution strategy:

```
function y = conv_rt(x, h)
% Convolve two finite-length arrays, x and h
%          returning array, y
```

In this case, we will *not* use sequence objects, just arrays, $x[n]$ and $h[n]$, returning array, $y[n]$. You will be given the arrays: the input, $x[n]$, and the impulse response, $h[n]$. I will guarantee that the length of $h[n]$ is always smaller than that of $x[n]$. We will also not time your convolution, so any way you do it is O.K.. You can use the `length` function to figure out the length of the arrays and you can use `for` loops. Since `conv_rt` is not a function that works on sequences, it should not be a method that lives in your *sequence.m* file. Please put it in the same working directory as your sequence file.

2.3.4 Deconvolution

Write a function to deconvolve an output sequence:

```
function x = deconv(y, h)
% DECONV Convolve finite-length Matlab sequence object, y,
%        given impulse response sequence object, h
%        returning sequence object, x.
```

Your `deconv` function will work on sequence objects. Since Matlab also has a `deconv` function, your `deconv` function is considered an overloaded function and needs to live in the *sequence.m* file.

Download lab2.m, test_lab2.p, test_lab2a.p, test_lab2b.p from the website. Place them in your working directory and type `publish lab2`. Remember to publish as a .pdf, not an HTML file. Submit your file to iLearn. Make sure that your code has the names of you (and your partner if you have one). It would also be helpful if the name of the file that you upload also has the name(s).

2.3.5 Properties of convolution

This part is just for your education and amusement. You do not have to submit it.

Let
```
» x = sequence([1 2 3 4 -1], -1);
```

and
```
» h1 = sequence([1 -1], 2);
```

and
```
» h2 = sequence([-1 0 3 -1], -2);
```

Using your `conv` function, show to your satisfaction that the commutative, associative and distributive properties of convolution are satisfied. In other words, show that

Commutative:	$x[n] * h_1[n] = h_1[n] * x[n]$
Associative:	$(x[n] * h_1[n]) * h_2[n] = x[n] * (h_1[n] * h_2[n])$.
Distributive:	$x[n] * h_1[n] + x[n] * h_2[n] = x[n] * (h_1[n] + h_2[n])$

2.3.6 Sound fun

This part is also for your education and amusement. You do not have to submit it.

Download lab2.mat from the website and place it in your Matlab working directory. Now, in the Matlab command window, type
```
» load lab2
```

This places a bunch of variables in your workspace:

- `tones`. This is a 2x11025 array. Each row is one second's worth of a pure cosine sampled at 11.025 kHz. The top row is 400 Hz; the second row is 1600 Hz.
- `seashell`. This is a robust, anonymous, male voice saying the word, "Seashell", sampled at 11.025 kHz
- `fs`. This is the sample frequency, 11025 of both `tone` and `seashell`.
- `fir_lp`. This is the impulse response of a FIR lowpass filter, designed with a cut-off frequency at about 500 kHz.
- `fir_hp`. This is the impulse response of a FIR highpass filter, designed with a cut-off frequency at about 1.6 kHz.

Try the following:

- Listen to each tone separately: `sound(tones(1, :), fs)` and `sound(tones(2, :), fs)`. If you are listening with headphones, watch the volume!
- Listen to the sum of the tones: `sound(sum(tones), fs)`.
- Filter the sum of the tones with the lowpass filter: `sound(conv(fir_lp, sum(tones)), fs)`
- Filter the sum of the tones with the highpass filter: `sound(conv(fir_hp, sum(tones)), fs)`
- Listen to the speech: `sound(seashell, fs)`
- Listen to the speech filtered with the lowpass and highpass filters:

```
soundsc(conv(fir_hp, seashell), fs)
```

and

```
soundsc(conv(fir_hp, seashell), fs)
```