San Francisco State University
Engineering 451

Laboratory #2 - Convolution

Outline

1. Purpose

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

Background reading includes:

- Oppenheim and Schaffer, Chapter2
- Holton notes, Units 2 and 3

2. Background

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,

$$y[n]=\sum_{k=-\infty}^{\infty} x[k]h[n-k]. \tag{L4.1}$$

As discussed in lecture, there are two ways of understanding the convolution sum, the *direct summation method* and the *flip-and-shift method*.

Direct summation method

In the direct summation method, we look at Equation (L4.1) and take $h[n-k]$ to be a sequence on n and consider k to be an index. At each value of k, we shift impulse response $h[n]$ by an amount k to form $h[n-k]$ and scale this shifted sequence by multiplying it by a value, $x[k]$. We then sum all the shifted, scaled impulse response together to form $y[n]$. Here's the same equation with a few notations

$$\underbrace{y[n]}_{\substack{\text{sequence}\\ \text{on } n}} = \sum_{k=-\infty}^{\infty} \underbrace{x[k]}_{\text{value}}\underbrace{h[n-k]}_{\substack{\text{sequence}\\ \text{shifted by } k}}$$

To give an example, assume $x[n]$ and $h[n]$ are a couple of simple sequences, as shown in Figure 1.

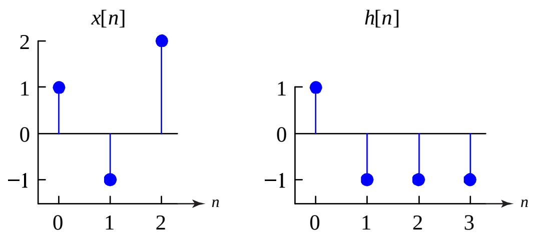


Figure 1: Sequences $x[n]$ and $h[n]$

The application of the direct summation method to these two sequences is shown in Figure 1. The first three panels of this figure correspond to $h[n]$ shifted by 0, 1 and 2 and multiplied by $x[0]$, $x[1]$ and $x[2]$ respectively. The bottom panel is the sum of the scaled and shifted sequences.

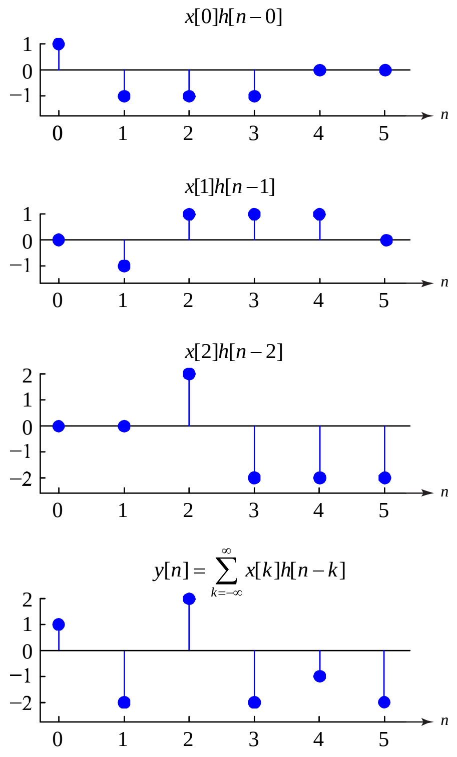


Figure 2: Convolution by the direct summation method

Notice that 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 output, $y[n]$.

Q: Why do we have to pad these sequences?

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 $h[n]$?

Flip and shift method

The flip and shift method is another way of looking at convolution that is of particular practical interest, since this is how low-level convolution routines are usually be coded in a language like C. In the flip and shift method, we compute the output sequence, $y[n]$, point by point; that is, we fix n at a particular value, and then evaluate the convolution sum for this value of n. Then we choose the next value of n and compute the sum and so on. To understand the computation of the convolution sum using the flip and shift method, look again at Equation (L4.1), but this time we interpret $x[k]$ as a sequence on k, and $h[n-k]$ as a sequence on k that has been flipped to form $h[-k]$ and then shifted by value n to form $h[n-k]$. These two sequences on k are then multiplied together and the resulting sequence, $x[k]h[n-k]$, is then summed on k to give a single value corresponding to $y[n]$ at the specified value of n.

$$\underbrace{y[n]}_{\substack{\text{value of } y\\ \text{at a given}\\ \text{value of } n}} = \sum_{k=-\infty}^{\infty} \underbrace{x[k]}_{\substack{\text{sequence}\\ \text{on } k}} \underbrace{h[n-k]}_{\substack{\text{sequence on } k,\\ \text{flipped and}\\ \text{shifted by } n}}$$

Figure 3 shows the computation of convolution sum at $n = 0$.

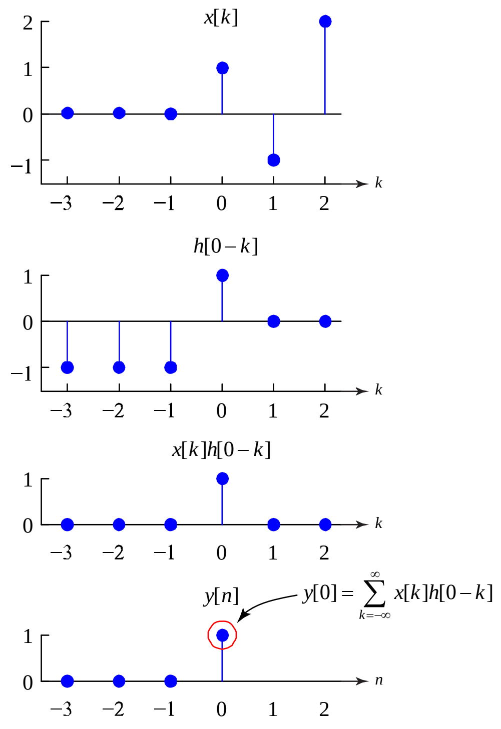


Figure 3: Convolution by the flip and shift method for $y = 0$

The top panel shows $x[k]$. The second panel shows $h[0-k]$, the flipped version of $h[k]$. Since $n = 0$, $h[0-k]$ is not shifted. The third panel shows $x[k]h[0-k]$. $x[k]$ and $h[0-k]$ overlap by only one nonzero value (at $k = 0$). The sum of $x[k]h[0-k]$ is one; therefore $y[0] = 1$.

Figure 4 shows convolution evaluation of $y[1]$ by the flip and shift method. Here again, the top panel shows $x[k]$. The second panel shows $h[1-k]$, the flipped version of $h[k]$ that is shifted by one to the right. The third panel shows the product $x[k]h[1-k]$. Since $x[k]$ and $h[1-k]$ overlap by two nonzero values (at $k = 0$ and 1), the product, $x[k]h[1-k]$, has two non-zero elements whose sum is -2; therefore $y[1] = -2$. If we evaluate the convolution sum for all necessary values of n, we get the same $y[n]$ shown in Figure 3.

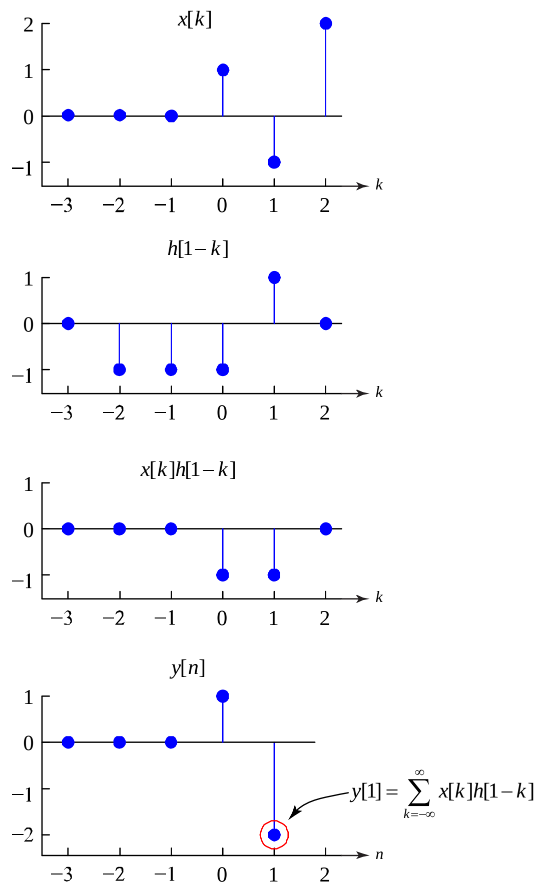


Figure 4: Convolution by the flip and shift method for $y=1$

Properties

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

<u>The commutative property</u>

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

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

This is illustrated in Figure 5 in terms of two experiments. In the first experiment, we first pass input $x[n]$ through a system characterized by impulse response $h[n]$ yielding $y[n]=x[n]*h[n]$. In the second experiment, we convolve $h[n]$ with $x[n]$, so $y[n]=h[n]*x[n]$. The commutative property says that results of the two experiments is the same.

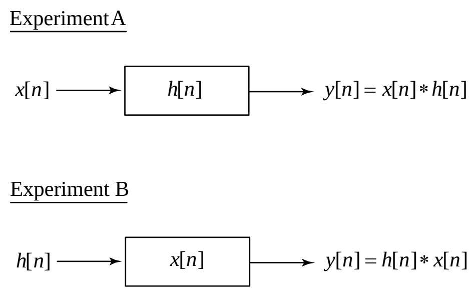


Figure 5: Commutativity of convolution

Figure 6 shows an illustration of the commutative property using the direct method with the sequences of Figure 1. You can see that the final result is the same, and to get that result we have to sum four sequences of length six rather than summing three sequences of length six, as shown in Figure 1.

Q: Why the difference?

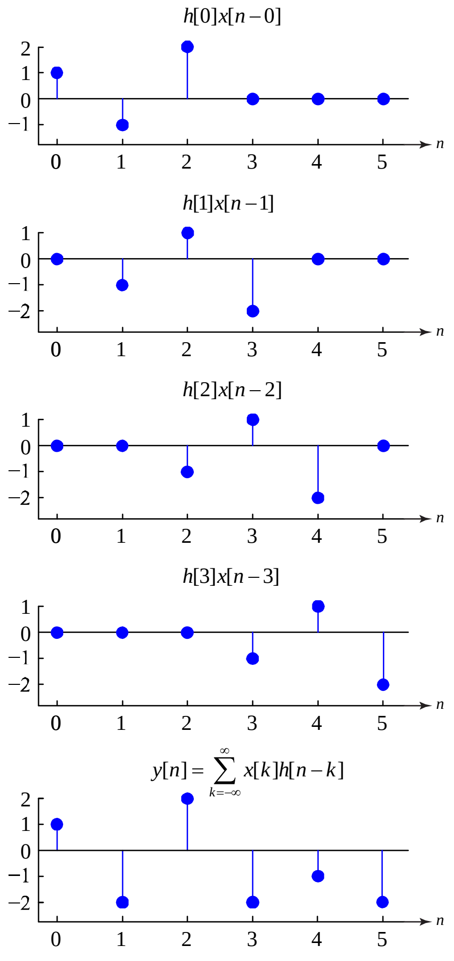


Figure 6: Convolution of $h[n]$ and $x[n]$ illustrating the commutative property

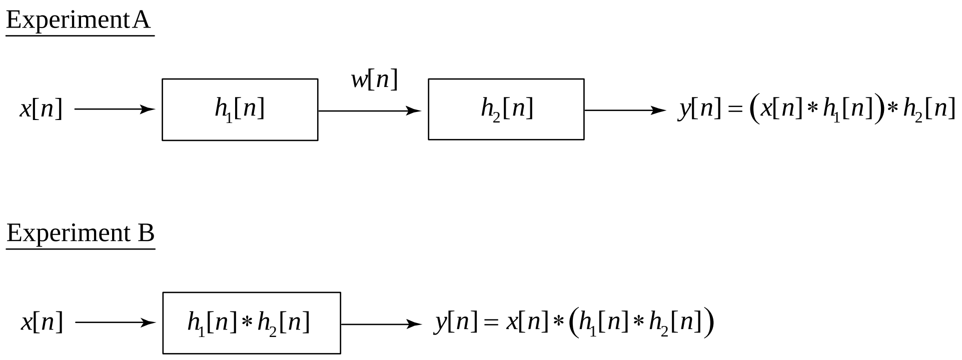
You should also convince yourself that the commutative property works with the flip and shift method.

The associative property

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

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

This is illustrated in Figure 7 in terms of two experiments. In the first experiment, we convolve $x[n]$ with $h_1[n]$ and then convolve the result with $h_2[n]$. Hence, $y[n]=\left(x[n]*h_1[n]\right)*h_2[n]$. In the second experiment, we first convolve $h_1[n]$ with $h_2[n]$ and then convolve the input $x[n]$ with the result, yielding $y[n]=x[n]*\left(h_1[n]*h_2[n]\right)$. The associative property says that results of the two experiments is the same.



Experiment A
$x[n]$
$h_1[n]$
$w[n]$
$h_2[n]$
$y[n]=\left(x[n]*h_1[n]\right)*h_2[n]$
Experiment B
$x[n]$
$h_1[n]*h_2[n]$
$y[n]=x[n]*\left(h_1[n]*h_2[n]\right)$

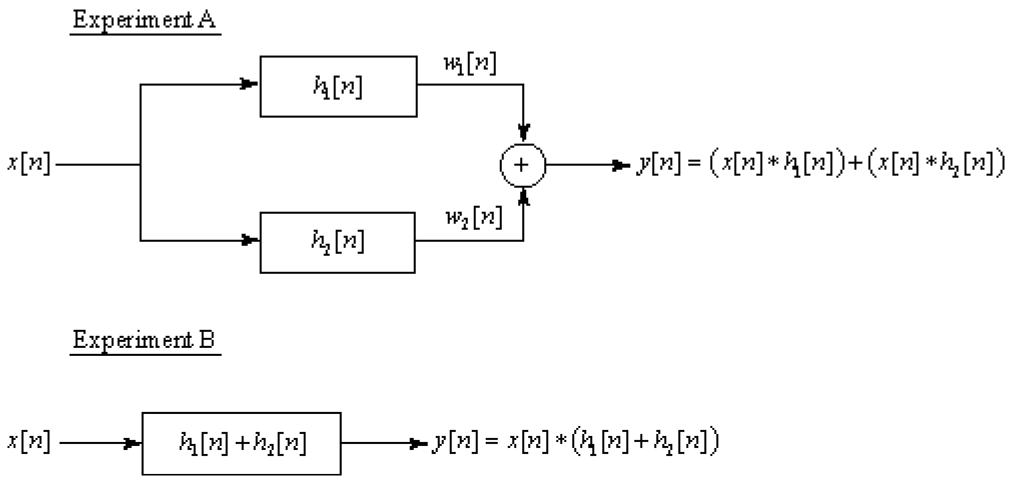
Figure 7: Associativity of convolution

<u>The distributative property</u>

Distributivity says that

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

This is illustrated in Figure 8 in terms of two experiments. In the first experiment, we convolve $x[n]$ with $h_1[n]$ and convolve $x[n]$ with $h_2[n]$ and then add the results of these two convoltions; hence, $y[n]=x[n]*h_1[n]+x[n]*h_2[n]$. In the second experiment, we first add $h_1[n]$ and $h_2[n]$ and then convolve the input $x[n]$ with the result, yielding $y[n]=x[n]*\left(h_1[n]+h_2[n]\right)$. The distributative property says that results of the two experiments is the same.



Experiment A
$h_1[n]$
$w_1[n]$
$x[n]$
$+$
$y[n]=\left(x[n]*h_1[n]\right)+\left(x[n]*h_2[n]\right)$
$h_2[n]$
$w_2[n]$
Experiment B
$x[n]$
$h_1[n]+h_2[n]$
$y[n]=x[n]*\left(h_1[n]+h_2[n]\right)$

Figure 8: Distributivity of convolution

Implementation issues

Let’s look at how you can implement the two methods of convolution in Matlab.

Direct summation method.

Consider the convolution $y[n] = x[n] * h[n]$ by the direct method, as shown in Figure 2. One way to do this with Matlab is to form a 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:

$$\mathbf{H} = \begin{pmatrix} 1 & -1 & -1 & 1 & 0 & 0 \\ 0 & 1 & -1 & -1 & 1 & 0 \\ 0 & 0 & 1 & -1 & -1 & 1 \end{pmatrix} \tag{L4.2}$$

Now we need to multiply each row by the appropriate value of $x[k]$ and then sum the rows. One way to do this is (not a very good way) to create an array,

$$\hat{\mathbf{X}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix},$$

which we multiply by **H** using the Matlab '`.*`' operator

$$\hat{\mathbf{X}} .* \mathbf{H} = \begin{pmatrix} 1 & -1 & -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 1 & -1 & 0 \\ 0 & 0 & 2 & -2 & -2 & 2 \end{pmatrix}. \tag{L4.3}$$

The three rows of this matrix correspond exactly to the values of the sequences plotted in the first three panels of Figure 2, namely $x[0]h[n-0]$, $x[1]h[n-1]$ and $x[2]h[n-2]$. We then sum the rows of the matrix to give a row vector that corresponds to the sequence in the bottom panel of the figure,

$$\mathbf{y} = \text{sum}(\hat{\mathbf{X}} \cdot * \mathbf{H}) = \begin{pmatrix} 1 & -2 & 2 & 0 & -3 & 2 \end{pmatrix}. \tag{L4.4}$$

Of course, the commutative property tells us that we could obtain the same result by convolving $h[n]$ with $x[n]$ as indicated in Figure 6. Here,

$$\mathbf{X} = \begin{pmatrix} 1 & -1 & 2 & 0 & 0 & 0 \\ 0 & 1 & -1 & 2 & 0 & 0 \\ 0 & 0 & 1 & -1 & 2 & 0 \\ 0 & 0 & 0 & 1 & -1 & 2 \end{pmatrix}, \tag{L4.5}$$

and

$$\hat{\mathbf{H}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

So,

$$\mathbf{X} .* \hat{\mathbf{H}} = \begin{pmatrix} 1 & -1 & 2 & 0 & 0 & 0 \\ 0 & -1 & 1 & -2 & 0 & 0 \\ 0 & 0 & -1 & 1 & -2 & 0 \\ 0 & 0 & 0 & 1 & -1 & 2 \end{pmatrix}. \tag{L4.6}$$

The four rows of this matrix correspond to the sequences plotted in the first four panels of Figure 6, namely $h[0]x[n-0]$, $h[1]x[n-1]$, $h[2]x[n-2]$ and $h[3]x[n-3]$. Summing the rows of the matrix gives a row vector that again corresponds to the sequence in the bottom panel of the figure,

$$\mathbf{y} = \text{sum}(\mathbf{X} \cdot * \hat{\mathbf{H}}) = \begin{pmatrix} 1 & -2 & 2 & 0 & -3 & 2 \end{pmatrix}.$$

Notice that the sizes of the matrices in Equations (L4.3) and (L4.6) 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 $h[n]*x[n]$?

Flip and shift method

This method implements the process shown in Figure 3 and Figure 4. Here, we can again use simple matrix multiplication to make this work.
You can show, for example that the convolution $x[n]*h[n]$ by the flip and shift method corresponds to the simple matrix multiplication

$$\mathbf{y} = \mathbf{x} * \mathbf{H} = \begin{pmatrix} 1 & -1 & 2 \end{pmatrix} \begin{pmatrix} 1 & -1 & -1 & 1 & 0 & 0 \\ 0 & 1 & -1 & -1 & 1 & 0 \\ 0 & 0 & 1 & -1 & -1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & -2 & 2 & 0 & -3 & 2 \end{pmatrix} \qquad \text{(L4.7)}$$

and the convolution $h[n]*x[n]$ by the flip and shift method corresponds to

$$\mathbf{y} = \mathbf{h} * \mathbf{X} = \begin{pmatrix} 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 & 2 & 0 & 0 & 0 \\ 0 & 1 & -1 & 2 & 0 & 0 \\ 0 & 0 & 1 & -1 & 2 & 0 \\ 0 & 0 & 0 & 1 & -1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & -2 & 2 & 0 & -3 & 2 \end{pmatrix} \qquad \text{(L4.8)}$$

The end result is *exactly the same* answers shown in the previous section. To understand this, look at the multiplication of the row vector **x** by a single column --say the first column -- of **H** in Equation (L4.7). This product is $x[0]h[0]+x[1]h[-1]+x[2]h[-2]=(1\cdot 1)+(-1\cdot 0)+(2\cdot 0)=1$. If you look carefully, you'll see that this is exactly the calculation that is done in Figure 3 to compute $y[0]$. The multiplication of the row vector **x** by a second column of **H** is $x[0]h[1]+x[1]h[0]+x[2]h[-1]=(1\cdot -1)+(-1\cdot 1)+(2\cdot 0)=-2$, which is the calculation done in Figure 4 to compute $y[1]$. Thus, each multiplication of **x** by each column of **H** is equivalent to evaluating the convolution sum at a different value of *n*.

Q: Show the results described in Equations (L4.7) and (L4.8).
Q: Explain how the matrix multiplications of Equations (L4.7) and (L4.8) can be interpreted as resulting from the flip and shift method.

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, $\mathbf{y}=\mathbf{x}*\mathbf{H}=\mathbf{X}*\mathbf{h}$. Deconvolution of **y** to obtain **x** is accomplished by matrix inversion: $\mathbf{x}=\mathbf{y}\mathbf{H}^{-1}$. Consider the sequences, $h[n]$ and $y[n]$, shown in Figure 9.

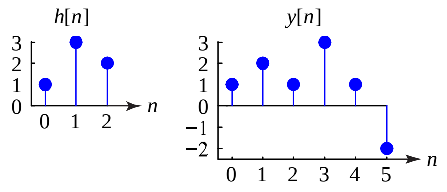


Figure 9: *h*[*n*] and *y*[*n*]

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

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

Then,

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

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

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

3. Assignment

I. Questions.

Answer all the questions in red, above.

II. Convolution function – Matlab style

Now, it's your turn! You are going to write a Matlab function to convolve two sequences using either the direct summation method or the flip and shift method:

```
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 your *sequence.m* file.

Of course, each of the sequences that we sum must 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 (L4.2) and/or (L4.5) 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`?
- You may *not* use Matlab's `toeplitz` function to build your matrices, though you might want to look at the documentation for this function and see why it could have been useful.
- 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.

III. 'Real time' Convolution function

The convolution we've just done above 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), for example 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, for example as the result of an A/D conversion occurring at a constant rate. For each point of the input, the application needs to 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 which 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 M points. For the n^{th} point of the output, we have to compute the convolution sum,

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

So, we have to do the M multiplications, $M-1$ additions. Since we only get one new value of $x[n]$ at each time point, we have to keep the past $M-1$ values of $x[n]$ in memory (as well as the current value of $x[n]$). Hence, we need a buffer of M 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, $y[n]$. Furthermore, I will guarantee that $h[n]$ is always the impulse response. We will also not time it, so any way you do it is O.K.. You can use the `length` function to figure out the length of the input arrays and you can use `for` loops. You may attempt to use a circular buffer if you wish, as described in the notes, but it's basically a big kludge without pointers. If you do, you may use Matlab's `circshift` function. 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.

IV. 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.
```

Since Matlab also has a `deconv` function, your `deconv` function that works on sequences is considered an overloaded function, and so it should 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`. (Incidentally, read the helpfile for the `publish` command. You can actually publish in other formats than html if you wish.) 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).

V. 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:	$x[n]*h_1[n]+x[n]*h_2[n]=x[n]*(h_1[n]+h_2[n])$

VI. 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:

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

and

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