DISCRETE-TIME SYSTEMS AND CONVOLUTION 4

Electrical Engineering 20N
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

HSIN-I LIU, JONATHAN KOTKER, HOWARD LEI, AND BABAK AYAZIFAR

1 Introduction

In this lab, we will explore discrete-time convolution and its various properties, in order to lay a better foundation for material to be presented later in the course. Convolution is an ubiquitous operation in signal processing, not least because it provides an elegant way to represent linear, time-invariant systems. The **convolution** of two signals x and y, in *discrete-time*, is defined as

CONVOLUTION

$$(x * y)(n) = \sum_{k=-\infty}^{\infty} x(k)y(n-k) = \sum_{k=-\infty}^{\infty} y(k)x(n-k).$$

In the case of LTI systems, the output signal of a system, $y(n)$, can be determined merely by convolving the input signal $x(n)$ with the impulse response $h(n)$ of that system:

$$y(n) = (x * h)(n).$$

Furthermore, as you shall see later in Fourier analysis, convolution in the time domain translates directly into multiplication in the frequency domain. In this lab session, we will try several different approaches of implementing convolution in LabVIEW.

Finally, we will see our first special-purpose discrete-time system, known as a discrete-time low-pass filter. We will analyze its behavior in the frequency domain and discover where it gets its name.

1.1 Lab Goals

- Explore the properties of discrete-time convolution.
- Implement discrete-time convolution in LabVIEW through different methods.
- Implement basic discrete-time filters in LabVIEW in both time and frequency domains.

1.2 Checkoff Points

2 Pre-Lab Section

2.1 LTI Systems and Impulse Responses

We consider any system F, which takes in an input signal $x(n)$ and produces the corresponding output signal $y(n)$; in other words, $y(n) = (F(x))(n)$[1]. We say that the system is **linear** if it satisfies the following properties: LINEAR

1. **Scaling Property**. If we were to scale any input signal $x(n)$ by a scalar α, the corresponding output signal should also be scaled by α. Formally, if $\hat{x}(n) = \alpha x(n)$, and $\hat{y}(n) = (F(\hat{x}))(n)$, then $\hat{y}(n) = \alpha y(n)$ as well.

[1]Note the notational subtlety: F is the *name* of the system, while F represents the system as a function that operates on an input signal. When writing, it is not always possible to maintain this distinction, so we do not usually worry about it, as long as we understand that there is, in fact, a distinction.

2. **Additivity**. Consider any two input signals $x_1(n)$ and $x_2(n)$, and let

$$y_1(n) = (F(x_1))(n), \quad y_2(n) = (F(x_2))(n).$$

   If we define a new signal

$$x(n) = x_1(n) + x_2(n),$$

   then the corresponding output signal $y(n) = (F(x))(n)$ is also

$$y(n) = y_1(n) + y_2(n).$$

   In simpler words, if we were to add together two input signals and feed the resultant signal through a system, the output signal should be the same as if we fed the two input signals separately, and added the corresponding output signals together.

We say that the system is **time-invariant** if, for any input signal $x(n)$, shifting the signal in time also shifts the corresponding output signal in time. Formally, for any input signal $y(n) = (F(x))(n)$, if $\hat{x}(n) = x(n-n_0)$, where n_0 is an integer, and $\hat{y}(n) = (F(\hat{x}))(n)$, then $\hat{y}(n) = y(n-n_0)$ if the system is time-invariant.

TIME-INVARIANT

A system that satisfies the linear and time-invariant properties described above is referred to as a **linear, time-invariant (LTI) system**[2]. LTI systems have several interesting features and properties, which will be the basis of much of our future study in this class. One of these interesting properties is the existence of an **impulse response**.

LTI SYSTEM

2.2 Echo Method

Based on the definition of convolution, we can describe two methods to graphically convolve two signals. The first method is known as the **echo method**, and is derived from the properties of an LTI system. We will describe the idea behind it here.

Consider any LTI system H. If we provide the Kronecker delta signal (or the discrete-time impulse) as an input signal, then the corresponding output signal is known as the **impulse response** of the system. It is denoted by $h(n)$. Since the impulse response is defined for an LTI system, we can deduce that

IMPULSE RESPONSE

$$\begin{array}{r} \delta(n) \to \boxed{\quad \text{H} \quad} \to h(n) \\ \alpha\delta(n) \to \boxed{\quad \text{H} \quad} \to \alpha h(n) \\ \delta(n-k) \to \boxed{\quad \text{H} \quad} \to h(n-k) \end{array}$$

We also know that $x(n)$ can be expressed as

$$x(n) = \sum_{k=-\infty}^{\infty} x(k)\delta(n-k).$$

In simpler terms, every input signal can be expressed as the summation of several shifted Kronecker delta functions ($\delta(n-k)$), each one scaled by $x(k)$. As a result, the output signal $y(n)$ must be described by

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) = (x * h)(n).$$

This is a very important (and cute) observation: when we feed a system with an input signal, we can determine the output signal simply by *convolving the input signal with the impulse response of the system*!

[2]No duh.

Be particularly wary here, though: this only applies to LTI systems in general. Why? What assumptions have we made to reach this conclusion? Also, note that whenever we speak of an impulse response, there is an implict assumption that the system *must* be LTI. Non-LTI systems may have responses to the Kronecker delta, but we do not label these responses as 'impulse responses'; the term is reserved for LTI systems alone.

Conversely, if we wanted to convolve the impulse response with an input signal, we would break the input signal up into its constituent impulses, find the (scaled and shifted) response to each of these impulses, and then add the responses together. This is referred to as the **echo method** of interpreting convolution. We will explore this in further detail during the lab.

2.3 Flip-and-Shift Method (Using the Toolkit)

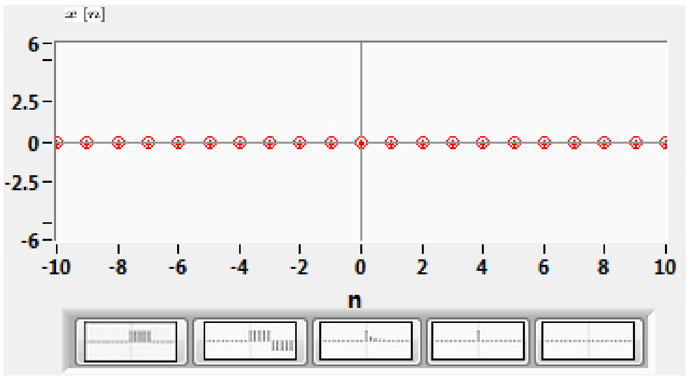
The second method to graphically convolve two signals is known as the **flip-and-shift method**, which we have explored in lecture and in the problem sets. The following exercises are meant to help you understand this method of convolution better, by means of a visual LabVIEW tool.

Download `Convolution Tool.exe` from bSpace and run it[3].

To run this tool, you will need the LabVIEW 8.6 runtime engine.[4]
Alternately, you may use the online Joy of Convolution web app, by Steve Crutchfield at John Hopkins University. However, we will not provide instructions or support for this app.[5]

The following are a few salient features of the LabVIEW `Convolution Tool.exe` tool:

1. $x[n]$:



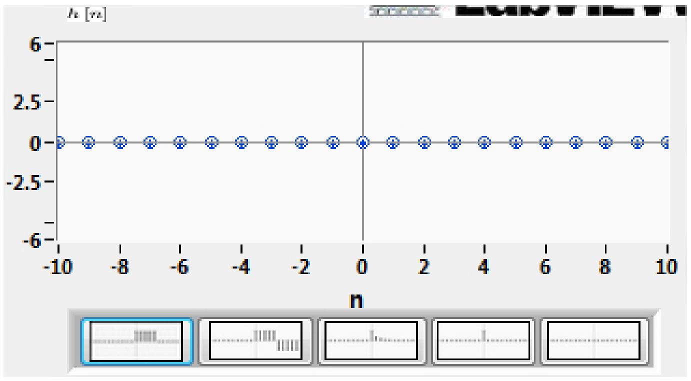
This input represents the **input signal** to an LTI system. It can be manipulated by clicking on the signal and changing the heights of different impulses. There are also preset signals; from left to right, these signals are a finite five-impulse train, a finite five-impulse train followed by a negative five-impulse train, a decaying exponential signal, a Kronecker delta, and the zero signal.

Note that at the time of this writing (Oct 1, 2010), the preset Kronecker delta in the `Convolution Tool.exe` has magnitude 3. I'm working on fixing this. You can manually adjust the magnitude

[3]We have not tested the tool with the Mac and Linux operating systems. If you are working on machines with these systems, consider working on a Windows machine, such as through a Remote Desktop Connection, just for this pre-lab. Alternately, consider using the online Joy of Convolution web app. See footnote below.
[4]PC LabVIEW 8.6 RTE: http://joule.ni.com/nidu/cds/view/p/id/1244/lang/en
Mac LabVIEW 8.6 RTE: http://joule.ni.com/nidu/cds/view/p/id/1245/lang/en
[5]Joy of Convolution web app: http://www.jhu.edu/signals/discreteconv/index.html

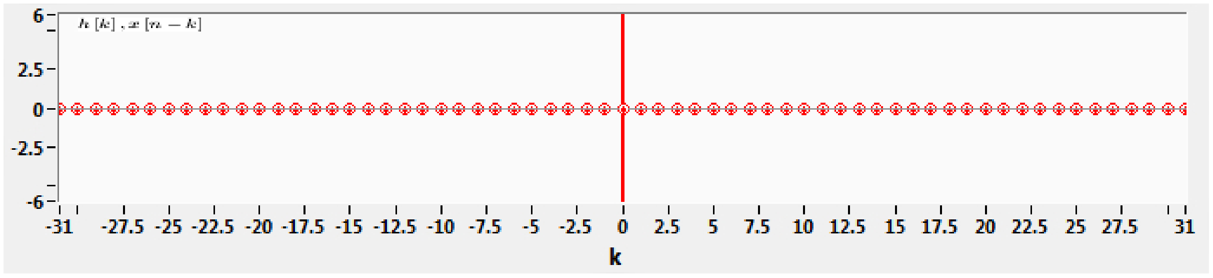
back to 1 for the exercises in this pre-lab.

2. $h[n]$:



This input represents the impulse response of that LTI system. It can be manipulated in the same ways that $x[n]$ can.

3. $h[k]$, $x[n-k]$:



This window allows you to drag the time-reversed version of x over h. The $h[k]x[n-k]$ and $y[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k]$ windows update as $x[n-k]$ is dragged above $h[k]$. **Note that the axis is the k-axis.**

4. Use the `STOP` button, followed by the `Run` button, to reset the signals.

2.4 Exercises

With the help of the convolution tool, answer the following questions. Supplement your answers with the necessary screenshots.

If you're not familiar with how to take screenshots on a Windows machine, see this helpful guide. Note that ALT-PrintScreen copies only the active window! For Mac machines, see this Mac screenshot guide.

1. In mathematics, the identity element (e) of an operation ($*$) satisfies the property that $a * e = e * a = a$ for any element a for which the operation $*$ makes sense.

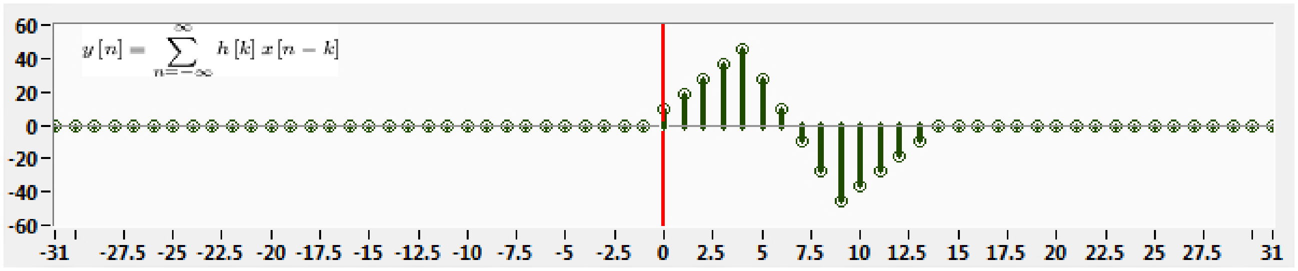
   In the discrete-time convolution tool, set the impulse response ($h[n]$) of the system to the Kronecker delta. Using various signals as input ($x[n]$), explain why the Kronecker delta is known as the *identity element of convolution*.

2. Select any one preset signal as $h[n]$ and another preset signal as $x[n]$, and observe the result $(h * x)[n]$. Reverse the roles of $h[n]$ and $x[n]$, and determine if the observations still remain the same. Do your observations support the commutativity of convolution?

3. Set $h[n]$ to an impulse shifted to the right by certain units; in other words, set $h[n] = \delta[n - n_0]$ for any integer value of n_0. Before conducting the exercise, predict the output signal if the input signal is the finite five-impulse train. Again, predict the output signal if the input signal is the decaying exponential signal. Test your predictions using the tool. From your observations, extrapolate what the result of $(x * h)[n]$ might be, where $h[n] = \delta[n - n_0]$, for *any* $x[n]$. Confirm your conjecture using the other kinds of input signals, and prove it using the definition of discrete-time convolution.

4. Determine, first on paper and then using the LabVIEW tool, the convolution of which two preset signals will yield the following signal[6]:

Figure 1 The output signal $y[n]$ of the mystery convolution.



Describe the process by which you determined your answer. (Hint: "I tested all 25 possible pairs of preset input signals" is *not* the greatest of answers.)

2.5 Submission Rules

1. Late submissions will *not* be accepted, except under unusual circumstances.

2. If the pre-lab exercises are not performed, you will get an immediate zero for the entire lab.

3. These exercises should be done *individually*.

2.6 Submission Instructions

1. Log on to bSpace and click on the `Assignments` tab.

2. Locate the assignment for `Lab 4 Pre-Lab` corresponding to your section.

3. Attach your answers to the questions presented in section 2.4. Templates for this assignment are available, in DOC and TEX formats, as part of the lab 4 resources on bSpace, but you need not use them.

4. This assignment is due **10 minutes** after the beginning of your lab session during the week of **October 4, 2010**. Please do not wait until (literally) the last minute to submit your work though; bSpace stops allowing submissions precisely on the minute, and since it takes a while to upload and submit your work, you may not be able to complete your submission.

[6]Note that if you used the online Joy of Convolution app, you will need to create the signals manually, as there are no two presets which give you the signal shown here

3 In-Lab Section

3.1 Objective

Our main objective in this lab is to convolve two signals, using an important property of LTI systems: Consider any LTI system H with impulse response $h(n)$. We know, from the pre-lab, that if we feed the system with an input signal $x(n)$, then the system produces the output signal $(x * h)(n)$.

We use this to our advantage: if we wanted to convolve *any* two signals, we name one of the signals as $x(n)$ and the other signal as $h(n)$. Then, we *construct* an LTI system H whose impulse response is $h(n)$, and feed the other signal x through this system H. From the above discussion, the output signal of this system *must* be $(x * h)(n)$, which is what we want.

Our sole concern then becomes actually *creating* the system, which is where LabVIEW proves useful. In this lab, we will attempt three different ways to construct an LTI system H with a particular impulse response.

We will use the mystery signal in pre-lab section 2.4, step 4 as our example. Recall that you had to determine which two preset signals generated that mystery signal. In order to make the implementation simpler, we denote the **signal with the longer nonzero sequence** as $x(n)$, and the other signal as $h(n)$. At this point, you may want to look back at your pre-lab solutions and recall what $x(n)$, $h(n)$, and $y(n)$ were. Note that $y(n)$ will be longer than $x(n)$ and $h(n)$. Throughout this lab we will use l to represent the length of the longest signal $y(n)$.

3.2 Dataflow Implementation

Our first method of implementing the system H employs dataflow programming. Here, you will implement the convolution sum using the echo method, where the output $y(n)$ is represented as a weighted sum of delayed versions of the input $x(n)$:

$$y(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k) \quad (1)$$

$$= \cdots + h(-1)x(n+1) + h(0)x(n) + h(1)x(n-1) + h(2)x(n-2) + \dots \quad (2)$$

For our particular choice of impulse response h, we know that $h(n) = 0$ for $n < 0$, so the terms comprising $y(n)$ will only consist of the current and delayed versions of $x(n)$. First, we need to prepare the front panel to display our input and output signals.

You may find the `Context Help` (`Ctrl-H`) useful for this lab and the following labs. When you hover over a block, the `Context Help` displays a brief summary of what the block does; when you hover over a wire, the `Context Help` displays the type of the data flowing through the wire.

1. Create a new VI called `Convolution Data Flow.vi`.
2. Use the `Array` subpalette (under `Programming`) in the block diagram to generate an array representing the signal $x(n)$ for $n > 0$. Pad the array with enough zeros so that the *length* of the signal is l. Also, display its stem plot on the front panel.
   - Use any of the available functions to construct your version of $x(n)$. We recommend the `Initalize Array` and `Build Array` blocks. Do not make a control on the front panel and manually type in $x[n]$.

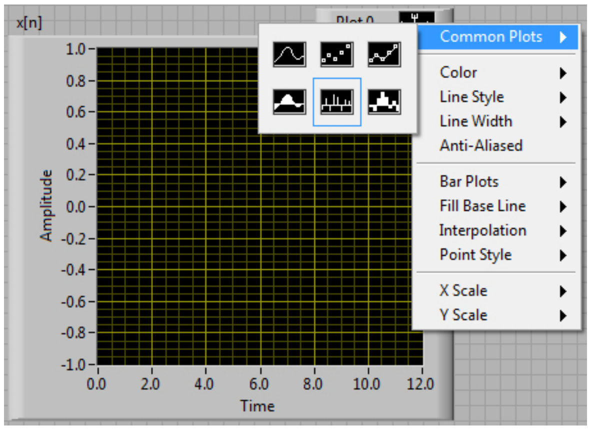
     If you will be using the `Build Array` block, you will probably be using it to concatenate several arrays into one. In order to achieve this, you will have to bring the `Build Array` into the `Concatenate Inputs` mode: right-click on the block and select `Concatenate Inputs`.

- In order to plot your signal, use the `Express XY Graph` present under the `Graph` subpalette of the `Modern` palette, available on the front panel. $x[n]$ will go on the y axis, and 0 through $l-1$ will go on the x-axis.

  Note that you'll need to generate an array of integers between 0 and $l-1$ to represent your variable n. You can do this, for example, by wiring the index out of an empty for loop.

- The indicator will not plot a stem plot by default. As shown in Figure 2, click on the plot legend in the upper-right corner of the `XY Graph`, go to `Common Plots` and change the plot to a stem plot. In addition, go to `Point Style` to make the points more obvious.

Figure 2 Plot Legend Menu



3. Now, we need to generate the Kronecker delta signal, otherwise referred to as the 'impulse', in order to feed it through the system H and to confirm its impulse response. In other words, create a one-dimensional array with the sole nonzero element at the beginning of the array. Pad the array with enough zeros so that the *length* of the Kronecker delta signal is l.

   From lab 02, recall that if the `For Loop` only has one input array, the upper bound of iterations `N` will be the length of the array.

4. It is time to create our system H with impulse response $h[n]$! In other words, we want to make a `LabVIEW` function such that if we feed it our Kroenecker delta, it will give us $h[n]$.

   Remember that our impulse respsne (and indeed) any signal is merely a linear combination of shifted and scaled Kronecker delta signals, as shown in Figure 3. As a result, in order to create H, we need to be able to take our impulse signal from step 3 and delay it by several time steps. We do so using a `For Loop` and shift registers. See Figures Figure 3 and Figure 4 below for more exposition.

Figure 3 An impulse response (and indeed, any discrete-time signal) is merely a summation of shifted and scaled Kronecker deltas.

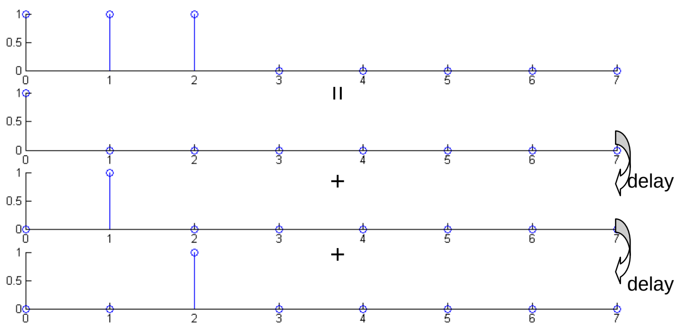
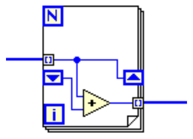


Figure 4 The LabVIEW program below implements the system H where impulse response $h[n] = \delta[n] + \delta[n-1]$.



3.3 Shift Registers and Multiple Delays

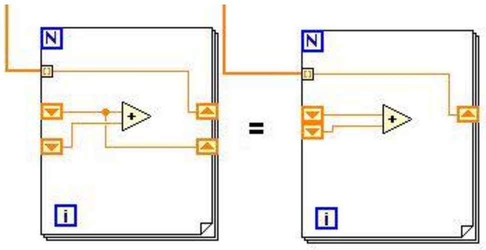
Recall that shift registers can be used to store and access array values from previous iterations of loops. This is precisely what delay elements do! By accessing array values from previous iterations of the loop during the current iteration, you are accessing values of the array that have been delayed.

Using multiple pairs of shift registers, you can access not only array values from the previous loop iteration, but values from the past several iterations. In the simplest case, the value from the left shift register is the value from the last iteration of the `For Loop`. If you were to delay this value as well, by sending the value received from a left shift register unchanged into a *different* right shift register, then the value in the next loop iteration appearing from the left shift register, corresponding to this different right shift register, would represent the array value delayed by two samples instead of one. Examine the left block diagram shown in Figure 5.

This operation is so common that it has a shortcut in LabVIEW: now, instead of using multiple pairs of shift registers to accomplish delays of multiple samples, we can use just one pair of shift registers. When we create one pair of shift registers, we right-click on either shift register, and select `Add Elements` to add multiple delays. This mimics having a stream of shift registers, each having their right end connected to the left end of the previous one, without all of the messy wires in between. The result is shown in the right block diagram of Figure 5. Note that you will create a stack of shift registers on the left side of the loop structure. The top shift register in the stack stores the array value from the previous iteration, the next shift register down stores the array value from two iterations ago, and so on.

You can also think of the multiple shift-registers as a data pipeline. Note that all of the shift-registers point downward. With each iteration, the values in the shift registers move downwards. What goes in to the top one? Well, whatever is wired to the shift register on the right hand side of the for loop.

Figure 5 Multiple shift registers can be combined into one.



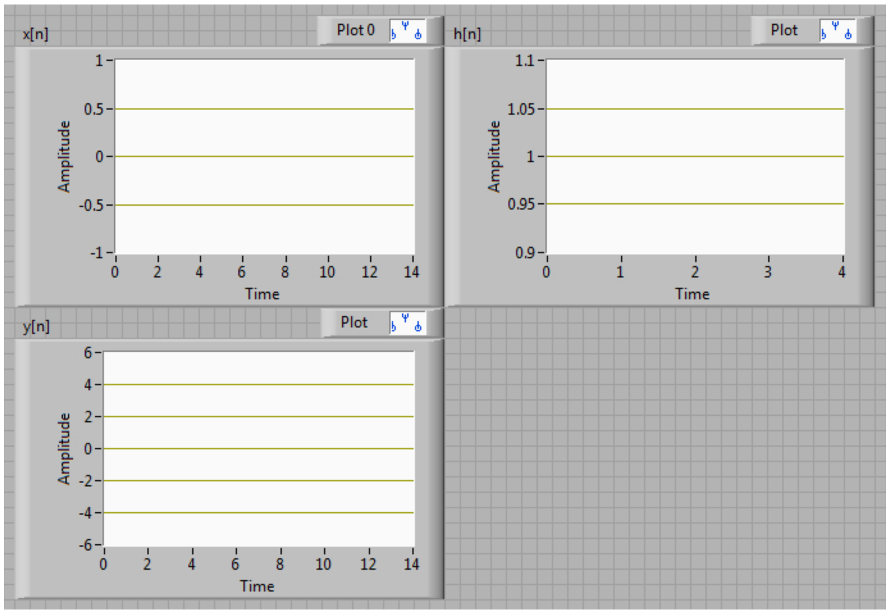
5. With the help of the discussion in section 3.3, create a `For Loop` that would take the impulse from step 3 and generate $h(n)$. Confirm your answer by actually feeding the impulse signal into the `For Loop` and producing a stem plot for the output $h(n)$.

   - How many delay elements will you need?
   - Use the `Compound Arithmetic` block under the `Programming` $\rightarrow$ `Numeric` subpalette to add all of the delayed signals together, and draw the combined signal out of the `For Loop`.
   - Do not forget to create a stem plot for $h(n)$!

6. Congratulations! You have just created a system H with impulse response $h(n)$; the `For Loop` with shift registers *is* the system. (*Woah!*) We can now perform the convolution $(x * h)(n)$; all we need to do is replace the input signal to the system. Currently, the input signal is an impulse signal. Make a copy of the system H and **replace its input signal with** $x(n)$.

7. Display the result $y(n)$ on the front panel. This should be the same as the mystery signal in the pre-lab section 2.4, step 4. The front panel of the finished VI should look similar to that shown in Figure 6.

Figure 6 `Convolution Data Flow.vi` Front Panel



3.4 `MathScript Node`s Implementation

Since we know how convolution is defined mathematically, we should be able to implement convolution in a `Mathscript Node`.

1. Create a new VI called `Convolution MathScript.vi`.
2. Create a `MathScript Node` with three outputs: `x`, `h`, and `y`, each output representing the corresponding signal.
3. Generate the signals $x(n)$, $y(n)$, and $h(n)$ inside the `MathScript Node` and display them using stem plots. Your program must be able to handle arbitrary $h(n)$ of length less than $x(n)$. In other words, your code should actually use $h(n)$.
   - Remember that, in `MathScript`, the first element in an array has an index of `1`, not `0`. Also, the signals themselves do not exist before index `1`. As a result, the convolution formula becomes

     $$y(n) = \sum_{k=1}^{\infty} x(k)h(n-k+1).$$

   - In order to achieve the convolution $(x * h)(n)$, ensure that *all the signals have the same length*; pad the signals at the end with enough zeros to give them the same length.
   - You may find the `zeros` and `ones` commands useful. Note that you probably want to generate row vectors instead of column vectors (e.g. zeros(1,5) instead of zeros(5))

     From lab 03, use the `help` command to read the LabVIEW Help file on any command. Yes, including `help` itself.

   - You can concatenate several arrays, such as `A`, `B`, and `C`, into one array using the following command: `D = [A B C]`. Notice the whitespace in between the names of the arrays.
   - Do not forget to initialize the array for the signal $y(n)$ with zeros before doing the convolution.

     As a debugging tip, always ensure that your `MathScript` code works in the `MathScript Window` before using it in a `MathScript Node`.

4. The front panel of the resulting VI should again look similar to that shown in Figure 6.
5. Upon completing your implementation, make sure that your convolution output matches that of your data-flow implementation.

3.5 Using Built-In `MathScript` Libraries

As you may be noticing by now, implementing operations in `MathScript` with nested `for` loops can be time-consuming, both in coding and executing. However, this does not mean that you should not use `MathScript` to implement convolution. In this section, we will introduce yet another, more efficient, way to perform convolution in `MathScript`.

1. Create a new VI called `Convolution MathScript Conv.vi`.
2. Use the `help` command in the `MathScript Window` to read the LabVIEW Help file on the `conv` command.
3. Create a `MathScript Node` with three outputs: `x`, `h`, and `y`, each output representing the corresponding signal.
4. Generate the signals $x(n)$, $y(n)$, and $h(n)$ inside the `MathScript Node`, this time using the `conv` command, and display them using stem plots. *Do not use* `For` *loops for this exercise.*

`MathScript` provides a rich library for general signal processing. In the future, try to utilize this library as much as possible by searching through the LabVIEW help files. Remember this general rule of thumb: *Do not use* `for` *loops unless you really have to!* `MathScript` is optimized for whole matrix operations, and using `for` loops that iterate through a matrix tends to produce a slower VI. As far as possible, utilize methods that operate on an array or on a matrix as a whole, as opposed to treating each element one-by-one.

4 Post-Lab Section

Now that we have gained a little bit of intuition regarding convolution, we will now focus on, and experiment with, specific LTI systems. In this section, we will use LabVIEW to help us explore the functioning of certain special discrete-time filters.

4.1 Discrete-Time Low-Pass and High-Pass Filters

Filtering can be found in various disciplines, from water filtration to audio amplification. It simply means to remove or to amplify different aspects of a signal or a medium, such as water. While water filtration removes dirt and other particles, a filter in electrical engineering may remove noise or boost a certain frequency, as is done by a bass boost on a CD player.

In lecture, we have learned about the LTI system $\mathsf{H_L}$ that relates an arbitrary input signal x and its corresponding output signal y with the equation

$$\forall n \in \mathbb{Z}, \quad y(n) = \frac{1}{2}\left(x(n) + x(n-1)\right). \tag{3}$$

Thus, the system $\mathsf{H_L}$ performs a two-point moving average. This system is called a **discrete-time low-pass filter**, and we will soon discover why.

DISCRETE-TIME LOW-PASS FILTER

Similarly, we consider an LTI system $\mathsf{H_H}$ that relates an arbitrary input signal x and its corresponding output signal y with the equation

$$\forall n \in \mathbb{Z}, \quad y(n) = \frac{1}{2}\left(x(n) - x(n-1)\right). \tag{4}$$

Thus, the system $\mathsf{H_H}$ performs a two-point moving difference. This system is called a **discrete-time high-pass filter**. Note that the change in sign produces a completely different system with different properties.

DISCRETE-TIME HIGH-PASS FILTER

4.2 Discrete-Time Frequencies

The frequencies of discrete-time signals lie in the interval $(-\pi, \pi]$. It is, of course, possible for a discrete-time signal to have frequencies beyond the bounds of the interval $(-\pi, \pi]$, but such frequencies can be 'folded in' to the interval $(-\pi, \pi]$. For example, consider the signal $x(n) = e^{i\Omega n}$, where $\Omega > \pi$. The frequency of this signal is Ω, but we know that $\Omega = \omega + 2\pi k$, for some $k \in \mathbb{Z}$ and $-\pi < \omega \leq \pi$. Then, we have

$$x(n) = e^{i\Omega n} = e^{i(\omega + 2\pi k)n} = e^{i\omega n} \cdot e^{i2\pi k n} = e^{i\omega n},$$

because $e^{i2\pi kn} = 1$ if $k, n \in \mathbb{Z}$. This is an extension of the idea that any real-valued angle can be expressed in terms of an angle between $-\pi$ and π. Since our frequencies are now restricted to the interval $(-\pi, \pi]$, we consider frequencies near zero to be *low-frequency signals*, and frequencies near $\pm\pi$ to be *high-frequency signals*.

Note that this discussion is not necessarily true for continuous-time signals. Why not? What assumptions have we made that only apply for discrete-time signals?

4.3 Creating the Low-Pass Filter

We now observe the effects of the two-point moving average filter on discrete-time signals of varying frequencies by modifying the `Convolution Data Flow.vi` you created in lab, in section 3.2.

1. Begin by examining the input-output relationship of the discrete-time low-pass filter H_L described by Equation 3, and relate it to the convolution sum. Determine the impulse response $h_L(n)$ corresponding to the discrete-time low-pass filter.

2. Create a copy of the `Convolution Data Flow.vi` under the different name `Convolution Data Flow LPF.vi`. Modify the impulse response of the system to be the impulse response $h_L(n)$ of the low-pass filter. Note that in doing so, you have created the discrete-time LTI system H_L.

3. Replace the input $x(n)$ with the inputs $\cos(0n)$, $\cos(\frac{\pi}{2}n)$, and $\cos(\pi n)$ of 30 samples each. Create a separate identical system for each input by copying and pasting the system you created initially.

   Hint: You must generate and store into an array the values 0 through 29, by using either a `For Loop` to generate and store the values (with the help of array output indexing), or a `Mathscript Node` to generate the array. Then, you will feed the array into a cos function, located in `Mathematics` $\rightarrow$ `Elementary and Special Functions` $\rightarrow$ `Trigonometric Functions`.

4. Observe the outputs corresponding to each of the three inputs and determine whether the system attenuates, magnifies or preserves its input.

5. Note that the input $\cos(0n)$ is of zero frequency, the input $\cos(\pi n)$ is of the highest possible discrete-time frequency, and the input $\cos(\frac{\pi}{2}n)$ is of medium frequency, halfway between the lowest and highest possible discrete-time frequencies. Based off of this, which frequencies does the system seem to allow (not attenuate)?

6. Recall that the system H_L performs a two-point moving average. If we feed the system a constant signal, such as $\cos(0n)$—a signal that does not vary at all—how does the system modify the signal? However, if we feed the system a signal, such as $\cos(\pi n)$—a signal that varies really quickly—how does the system modify these variations in the signal, and thus the signal itself?

7. Based on your answers to previous questions, why do you think the discrete-time two-point moving average system that you created is also called a discrete-time low-pass filter?

4.4 Creating the High-Pass Filter

We now observe the effects of the two-point moving difference filter on discrete-time signals of varying frequencies by modifying the `Convolution Data Flow.vi` you created in lab, in section 3.2.

1. Begin by examining the input-output relationship of the discrete-time high-pass filter H_H described by Equation 4, and determine the impulse response $h_H(n)$ corresponding to the discrete-time high-pass filter.

2. Create a copy of the `Convolution Data Flow.vi` under the different name `Convolution Data Flow HPF.vi`. Modify the impulse response of the system to be the impulse response $h_H(n)$ of the high-pass filter. Note that in doing so, you have created the discrete-time LTI system H_H.

3. Replace the input $x(n)$ with the inputs $\cos(0n)$, $\cos(\frac{\pi}{2}n)$, and $\cos(\pi n)$ of 30 samples each. Create a separate identical system for each input by copying and pasting the system you created initially.

4. Observe the outputs corresponding to each of the three inputs and determine whether the system attenuates, magnifies or preserves its input.

5. Note that the input $\cos(0n)$ is of zero frequency, the input $\cos(\pi n)$ is of the highest possible discrete-time frequency, and the input $\cos(\frac{\pi}{2}n)$ is of medium frequency, halfway between the lowest and highest possible discrete-time frequencies. Based off of this, which frequencies does the system seem to allow (not attenuate)?

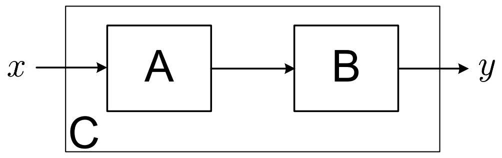
6. Recall that the system $\mathsf{H_H}$ performs a two-point moving difference. If we feed the system a constant signal, such as $\cos(0n)$—a signal that does not vary at all—how does the system modify the lack of variations in the signal, and thus the signal itself? However, if we feed the system a signal, such as $\cos(\pi n)$—a signal that varies really quickly—how does the system modify these variations in the signal, and thus the signal itself?

7. Based on your answers to previous questions, why do you think the discrete-time two-point moving difference system that you created is also called a discrete-time high-pass filter?

4.5 Cascading Filters

We can create more complicated filters by cascading pre-existing filters. In the configuration shown in Figure 7, LTI systems A (with impulse response $a(n)$) and B (with impulse response $b(n)$) have been cascaded in series to produce another LTI system C (with impulse response $c(n)$).

Figure 7 Two LTI systems A and B cascaded in series to produce a third filter C.



We can derive a relationship between a, b, and c as follows: Consider the input signal $x(n) = \delta(n)$, the Kronecker delta. If we feed it into the system C, we expect the impulse response $c(n)$, by definition. Now, we decompose the system C into its constituent systems A and B. The output of the system A in response to $x(n)$ is $a(n)$ (why?). This output is the input to the system B. We know that, for any arbitrary input signal, the output signal of an LTI system is the convolution of its impulse response with the input signal. This implies that the output of the system B corresponding to the input $a(n)$ must be $(a * b)(n)$. But, the output of the system B is the output of the system C.

1. From the above discussion, generate a relationship between a, b, and c.

2. Now, we will cascade the discrete-time low-pass filter $\mathsf{H_L}$ with the discrete-time high-pass filter $\mathsf{H_H}$. What is the impulse response $h_M(n)$ of the resulting overall filter $\mathsf{H_M}$?

3. Does the order in which you convolve the two impulse responses matter? In other words, does the order of the individual systems in a cascade make a difference in the impulse response of the cascaded system?

4. Create a copy of the `Convolution Data Flow.vi` under the different name `Convolution Data Flow MPF.vi`. Modify the impulse response of the system to be the impulse response $h_M(n)$ of the filter $\mathsf{H_M}$. Note that in doing so, you have created the discrete-time LTI system $\mathsf{H_M}$.

5. Replace the input $x(n)$ with the inputs $\cos(0n)$, $\cos(\frac{\pi}{2}n)$, and $\cos(\pi n)$ of 30 samples each. Create a separate identical system for each input by copying and pasting the system you created initially.

6. Observe the outputs corresponding to each of the three inputs and determine whether the system attenuates, magnifies or preserves its input.

7. Based on your answers to previous questions, why do you think the filter that you just created is also called a discrete-time mid-pass filter or a discrete-time band-pass filter?

4.6 Submission Rules

1. Late submissions will *not* be accepted, except under unusual circumstances.
2. These exercises are recommended to be done *in groups of two*. Only one person need submit the required files, however.

4.7 Submission Instructions

1. Log on to bSpace and click on the `Assignments` tab.
2. Locate the assignment for `Lab 4 Post-Lab` corresponding to your section.
3. Attach the following files to the assignment:
   (a) The VIs `Convolution Data Flow LPF`, `Convolution Data Flow MPF`, and `Convolution Data Flow HPF`.
   (b) Your answers to the questions in section 4.2 through section 4.5.
   (c) A text file called `PARTNERS.txt` containing the names of the students in the pair.
4. Templates for this assignment are available, in DOC and TEX formats, as part of the lab 4 resources on bSpace, but you need not use them.
5. This assignment is due **10 minutes** after the beginning of your lab session during the week of **October 11, 2010**. Please do not wait until (literally) the last minute to submit your work though; bSpace stops allowing submissions precisely on the minute, and since it takes a while to upload and submit your work, you may not be able to complete your submission.

5 Acknowledgments

Special thanks go out to the teaching assistants (TAs) of the Spring 2009 semester (Vinay Raj Hampapur, Miklos Christine, Sarah Wodin-Schwartz), of the Fall 2009 semester (David Carlton, Judy Hoffman, Mark Landry, Feng Pan, Changho Suh), and of the Spring 2010 semester (Xuan Fan, Brian Lambson, Kelvin So) for providing suggestions, ideas, and fixes to this lab guide.