Laboratory 3: Discrete-Time Systems in the Time Domain

1. Objectives

- Learn how to determine the output y[n] by using the Matlab command
  - `conv` to convolve the input x[n] with the system's impulse response h[n]
  - `filter` to apply a given difference equation to a given input x[n]
- Experimentally verify the communicative, distributive, and associative properties of the convolution operator
- Gain real-world experience by using a simple model of echo reduction to reduce echo contamination in a voice recording

2. Introduction

In the previous lab you learned the basic (and some advanced!) concepts involved in using Matlab to represent signals. You developed the necessary tools for programming m-files, keeping track of the indices of vector representations of discrete signals, and became proficient at creating graphs of signals. This laboratory will focus on the properties of the systems that operate on signals you studied in Lab 2, specifically the time-domain properties of discrete systems.

EE431 works with systems that are linear and time-invariant (LTI). Linear systems are systems that conform to two rules. First, if the input signal x[n] is scaled by some amount k (an unfiltered sound recording's volume is increased) then the output is scaled by the same amount k (the output after a noise filter is proportionally louder). Second, if input $x_1[n]$ causes output $y_1[n]$, and a different input $x_2[n]$ causes $y_2[n]$, then a single combined input ($x_1[n]+x_2[n]$) causes the output ($y_1[n]+y_2[n]$). Imagine that you design a noise filter to remove a 60Hz hum that contaminated an audio recording because of poor equipment grounding techniques, and you design the filter using DSP techniques we'll cover in future classes to have minimal effect on the music (x_1 is the input music, and y_1 is the post filtered music). If you know how much 60Hz remains after filtering a quiet section of the recording (x1 is the input noise alone and y1 is the post-filtered noise alone), then linearity means you know the output will sound like the sum of the filtered noise alone and the filtered music alone ($y_1 + y_2$). Linearity means that you can determine a system's impulse response h[n] and convolve it with the input x[n] to determine the output.

Unfortunately, linearity does not guarantee that the system remains the same with time. Time-invariance does. A non-time-invariant system may look like a linear lowpass filter now and a linear highpass filter a second later, meaning that its impulse function h[n] changes with time. Linear, time-invariant systems have a single impulse response h[n], and their output y[n] can be computed using the convolution sum

$$y[n] = \sum_{m=-\infty}^{\infty} h[n-m]x[m] \qquad \text{(eqn 1)}$$

Notice how similar this is to the concept of continuous-time LTI systems you studied in EE230 in which the output y(t) was related to the input x(t) by the convolution integral

$$y(t) = \int_{-\infty}^{\infty} h(t-\tau)x(\tau)d\tau \qquad \text{(eqn 2)}$$

3. Laboratory

1. Convolution

The Matlab function `conv` computes the convolution sum

$$y[n] = \sum_{m=-\infty}^{\infty} h[n-m]x[m] = \sum_{m=-\infty}^{\infty} x[n-m]h[m] \qquad \text{(eqn 3)}$$

assuming that x[n] and h[n] are finite-length sequences. If x[n] is non-zero only on the interval $n_x \le n \le n_x + N_x - 1$ (i.e. x[n] has N_x non-zero samples that begin at n_x) and similarly h[n] is non-zero only on the interval $n_h \le n \le n_h + N_h - 1$, then y[n] can be nonzero only on the interval

$$(n_x + n_h) \le n \le (n_x + n_h) + N_x + N_h - 2 \qquad \text{(eqn 4)}$$

meaning that conv need only compute [n] for the $N_x + N_h - 1$ samples on this interval. If the Matlab vector variable **x** contains N_x numbers representing the input signal, and **h** is another Matlab vector containing N_h numbers representing the LTI system's impulse response then

```
y = conv(h,x)
```

returns in **y** the $N_x + N_h - 1$ samples of y[n] on the interval of eqn. 4. However, `conv` does not return the indices of the samples of y[n] stored in **y** since `conv` does not take as inputs the index vectors for **x** and **h**. Instead, you are responsible for keeping track of these indices and will be shown how to do this below.

Problem 1
Consider the finite-length signal

$$x[n] = \begin{cases} 1, & 0 \le n \le 5 \\ 0, & \text{otherwise} \end{cases}$$

a) Analytically (use the convolution summation formula for n=0, 1, etc.) find y[n] = x[n] * x[n].

b) Compute the nonzero samples of y[n] = x[n] * x[n] using `conv`, and store those samples in the vector y. Your first step should be to define the vector x to contain the samples of x[n] on the given non-zero interval. Also construct an index vector ny, where ny(i) contains the index of the sample of y[n] stored in the i^{th} element of y, so that stem(ny,y) plots y correctly. For example, ny(1) should contain $N_1+N_1 = 2\ N_1$ where N_1 is the first nonzero index of x[n]. Plot your results using stem and make sure that your plot agrees with the signal you determined in part a. As a check, your signal should look triangular, with increasing amplitude to a peak, and then decreasing amplitude.

c) Consider the finite length impulse response

$$h[n] = \begin{cases} n, & 0 \le n \le 5 \\ 0, & \text{otherwise} \end{cases}$$

Graphically compute y[n] = x[n] * h[n]. Next, compute y using `conv` , where your first step should be to define the vector h to contain h[n] on its non-zero interval. Again construct vector ny which contains the interval of n for which y contains y[n]. Plot your results using stem(ny,y). As a check, your results should agree with your analytical derivation.

d) **Challenge**: Rather than continuing to determine y using `conv` , then determining ny, and then plotting it with stem(ny,y), write a Matlab function to do it all for you. The first and last lines of the Matlab function are:

```
function conv1(nx, x, nh, h)
<add your code here>
stem(ny, y)
```

It need not return any variables since it does the plotting for you. Check it using the values of x, nx, h, and nh you used in part c above – i.e. test using

```
conv1(0:5, ones(1,6), 1:5, 1:5)
```

For the problem just considered—implementing y[n] = x[n]*h[n] and using `conv` –the signal h[n] can be viewed as the impulse response of an LTI system for which x[n] is the system input and y[n] is the system output. Because the given h[n] is zero for n<0, this system is causal. It doesn't start creating an output until it is given a non-zero input. Systems are usually causal if n represents time samples and you are working with a real-time system, such as when developing robotic motor control system (a Patriot's missile system h[n] can't target an enemy incoming missile until after its sensors x[n] report the missile's presence). Real-world DSP systems do not have to be causal, however. ELINT (electronic intelligence) describes the process of determining (among other things) the location of enemy air-defense artillery (ADA) locations by analyzing received radar signals, often collected by specially-instrumented aircraft. Within these aircraft, radio waves of radar frequencies are recorded as long sequences of numbers, and analyzed off-line. In a simplified sense, the input vector x[n] of the radio signal is analyzed for the presence of a radar ping. The DSP algorithm filters x[n] to create an output y[n] such that peaks in y[n] correspond to radar pings present in x[n]. Because the data is being processed offline and not in real-time, the output at a particular sample (e.g. y[500]) may be a function not only of previous value of the input (e.g. x[1], x[2], … x[500]), but also of a future values of x (e.g. x[501], x[502]). This is an example of a non-causal LTI system.

Problem 2
Consider the system with the non-causal impulse response given by h[n+5], where h[n] is defined in Problem 1. Plot the output y[n] caused by the input x[n] given in Problem 1 to this new system function (you may use your conv1 function if you created it, or you may do it graphically or using the mathematical definition). How does it compare to the output y[n] computed by the causal system?

2. Matlab's `filter` command

The `filter` command computes the output of a causal, LTI system for a given input when the system is specified by a linear constant-coefficient difference equation. Specifically, consider an LTI system satisfying the difference equation

Math notation
$$\sum_{k=0}^{N-1} a_k y[n-k] = \sum_{m=0}^{M-1} b_m x[n-m] \qquad \text{(eqn 5)}$$

where x[n] is the input and y[n] is the output. I find abstract equations with loads of variables such as this hard to understand because it's difficult to see what variables change and which ones are constant for a given system. Here is a specific example with N=2, a_0 = 1, a_1 = 2, M=3, b_0=2, and b_1=-3, and b_2=5:

$$y[n] + 2\,y[n-1] = 2\,x[n] - 3\,x[n-1] + 5\,x[n-2] \qquad \text{(eqn 6)}$$

If **x** is a Matlab vector containing the input x[n] on the interval $n_x \le n \le n_x + N_x - 1$ and vectors **a** and **b** contain the coefficients a_k and b_k, then `y=filter(b,a,x)` returns the output of the causal LTI system satisfying:

Matlab notation
$$\sum_{k=0}^{N-1} a(k+1)\, y(n-k) = \sum_{m=0}^{M-1} b(m+1)\, x(n-m) \qquad \text{(eqn 7)}$$

Note that a(k+1) = a_k and b(m+1) = b_m, since Matlab requires that all vector indices begin at one. For example, the difference equation described by eqn. 6 is represented by the Matlab vectors `a = [1 2]` and `b = [2 -3 5]`, and the output y[n] to a given input x[n] to the system is computed with the following command: `y = filter(b, a, x);`

The output vector y returned by `filter` contains samples of y[n] on the same interval as the samples in x, i.e., $n_x \le n \le n_x + N_x - 1$, so that both vectors **x** and **y** contain N_x samples. Note from eqn 5, however, that `filter` (or any method of solving the difference equation) needs M-1 prior values of x[n] and K-1 prior values of y in order to compute the first output value $y[n_x]$. The function `filter` assumes that these samples are zero.

Problem 3: Using `filter` with difference equations

Define vectors **a1** and **b1** to describe the LTI system $y[n] = 0.5x[n] + x[n-1] + 2x[n-2]$
Define vectors **a2** and **b2** to describe the LTI system $y[n] = 0.8y[n-1] + 2x[n]$
Define vectors **a3** and **b3** to describe the LTI system $y[n] - 0.8y[n-1] = 2x[n-1]$

For each of these three systems, use `filter` to compute the response y[n] on the interval $1 \le n \le 4$ to the input signal x[n] = n u[n]. You should begin by defining the vector x = [1 2 3 4], which contains x[n] on the interval $1 \le n \le 4$. Include in your report the values of the three output vectors y1[n], y2[n], and y3[n] produced by their respective difference equation. Stem plot the results (remember from the last lab how to use subplot?) using their correct indices.

Hint 1: the output index ny for vector y[n] begins at the same value as the input index vector nx for vector x[n] does.

Hint 2: y1(1) = 0.5. Note that y1(1) shows that filter has forced x[0] and x[-1] equal to zero, since both of these samples are needed to determine y1[1].

The function `filter` can also be used to perform discrete-time convolution. Consider what happens to the general equation for an LTI system described by eqn 5 when $a_k = \delta[k]$. Then every a_k coefficient becomes zero except when k is 0, and eqn 5 simplifies to:

$$y[n] = \sum_{k=0}^{M} b_m x[n-k] \qquad \text{(eqn 8)}$$

noting that b_k is really a vector and could equally as well be written b[k], eqn 8 becomes

$$y[n] = \sum_{m=-\infty}^{\infty} b[k]x[n-k] \qquad \text{(eqn 9)}$$

(increasing the summation from M to ∞ doesn't change the answer since b[m] is zero for k > M). Note the similarity between eqn 9 and eqn 3; we can use `filter` to convolve! Because the impulse response **h** of this type of filter when **a** = 1 is finite (specifically, M+1 samples long), these kind of filters are called finite-length impulse response (FIR) filters. If you stimulate them with a short x[n], their output will return to zero M+1 samples after x[n] stops. This is in contrast to what can happen if K>0 in eqn 5; then the response to a short x[n] can last forever (e.g. y[n] = y[n-1] + x[n] has a step function for its impulse response). Filters of this sort have impulse responses that last forever and are called infinite-length impulse response (IIR) filters.

Let's review the differences between `filter` and `conv`, and what we just found:

- `filter` lets one take the difference equation coefficient vectors **a** and **b** and the input vector **x**, and returns the output vector **y**, using `y = filter (a,b,x)`.
- `conv` takes the impulse response vector **h** and the input vector **x**, and returns the output vector **y** using `y = conv (h,x)`.
- We just derived that you can also use `filter` with impulse response vector **h** and input vector **x** by setting **a**=1 as follows: `y = filter(h, 1, x)`.

Problem 4: Using `filter` to convolve

Consider the convolution of the signal x[n] and h[n] defined in Problem 1. Redo this using the `filter` command as just described. Put the commands you used and the resulting stem plot in your report. Hint: for `filter(h,1,x)` to return the same answer as `conv(h,x)` then the input vector **x** to `filter` must contain $N_x + N_h - 1$ samples, since that is the number of samples `conv` returns, and `filter` always returns as many samples in **y** as it is given in **x**. To do this, add zeros to x until it is long enough; this is called "zero padding".

3. Properties of LTI systems

In this section you will experimentally verify the commutative, associative, and distributive properties of convolution for a specific set of signals. In addition you will examine the implications of these properties for series and parallel connections of LTI systems. The problems in this exercise will make heavy use of the `conv` function. Although the properties in this section are only tested for discrete-time systems, they are also valid for continuous-time systems.

a) The commutative property

The commutative property for an operation means that the ordering of the operands can always be exchanged without changing the result. Multiplication of scalars, for instance, is commutative since ab = ba. Multiplication of matrices, is not; AB ≠ BA (for instance, a 1xN row vector multiplied by a Nx1 column vector yields a scalar, but a Nx1 column vector multiplied by a 1xN row vector yields a NxN square matrix).

Convolution is a commutative operation, and the physical significance of being able to interchange operands $h_1[n] * h_2[n] = h_2[n] * h_1[n]$ means that systems can be interchanged without affecting the output signal as shown in Figure 1.

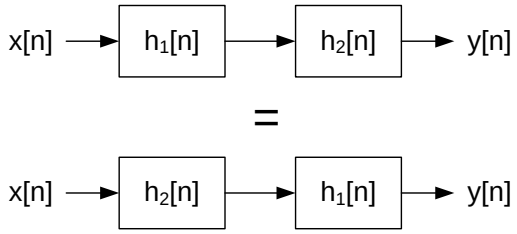

Figure 1: The commutative property of convolution allows the location of two filters to be switched without affecting the output.

b) The distributive property

The distributive property signifies that

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

This implies that the output of two LTI systems connected in parallel is the same as one system whose impulse response is the sum of the impulse responses of the parallel systems. Figure 2 graphically illustrates this.

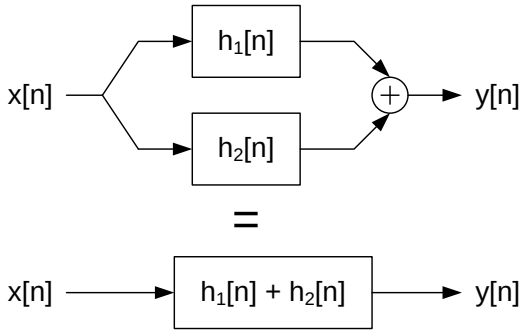

Figure 2: The distribute property of convolution as illustrated by a block diagram showing the equivalence of two differently-connected systems.

c) The associative property

Convolution obeys the associative property, which mathematically states that

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

As shown in Figure 3, this implies that the result of processing a signal with a series of LTI systems is equivalent to processing the signal with a single LTI system whose impulse response is the convolution of all the individual impulse responses of the connected systems.

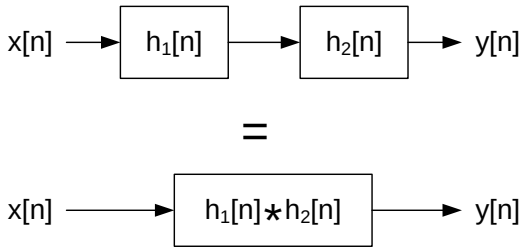

Figure 3. The associative property of convolution allows multiple system blocks connected in series to be replaced by a single system block.

Problem 5: Properties of LTI systems

a) To verify the properties of convolution, the following 3 signals are used:

$$x[n] = \begin{cases} 1, & 0 \le n \le 4 \\ 0, & \text{otherwise} \end{cases}, \qquad h_1[n] = \begin{cases} 1, & n = 0 \\ -1, & n = 1 \\ 3, & n = 2 \\ 1, & n = 4 \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \quad h_2[n] = \begin{cases} 2, & n = 1 \\ 5, & n = 2 \\ 4, & n = 3 \\ -1, & n = 4 \\ 0, & \text{otherwise} \end{cases}$$

Define the Matlab vector **x** to represent x[n] over the interval $0 \le n \le 9$, and define its index vector nx. Define Matlab vector h1 and h2 over the interval $0 \le n \le 4$, and define their index vectors nh1 and nh2. Make appropriately-labeled stem plots of these 3 using a single row of 3 columns of subplots.

For parts b, c, and d you will calculate two output vectors $y_1[n]$ and $y_2[n]$. For each part in your report include the commands you used to generate each. Since both output vectors should be identical (that's your check) include only a single correctly-indexed stem plot for each part; use subplot to draw them side by side in one row of two.

b) Verify the commutative property of convolution in Matlab by calculating
$y_1[n] = x[n] * h_1[n] * h_2[n]$, and
$y_2[n] = x[n] * h_2[n] * h_1[n]$.

c) Verify the distributive property of convolution in Matlab by calculating
$y_1[n] = x[n] * (h_1[n] + h_2[n])$ and
$y_2[n] = x[n] * h_1[n] + x[n] * h_2[n]$.

d) Verify the associative property of convolution in Matlab by calculating
$y_1[n] = (x[n] * h_1[n]) * h_2[n]$ and
$y_2[n] = x[n] * (h_1[n] * h_2[n])$.

4. Real-World Comprehension Problem
Echo Cancellation via Inverse Filtering

In this section, consider the problem of removing an echo from a recording of a speech signal. Perhaps you wish to build a portable headphone system for sports fans or classical music enthusiasts that removes the echoing effects of the stadium or concert hall making the referees statements or music clearer.

A model for echo generation is given in eqn 10 and shown graphically in Figure 4.

$$y[n] = x[n] + \alpha\, x[n-N] \qquad \text{(eqn 10)}$$

where x[n] is the original sound signal, which has been delayed by N samples and then added back with its amplitude decreased by $\alpha < 1$ (the echoed signal is not as strong as the original signal). This is a reasonable model for an echo resulting from the signal reflecting off an absorbing surface like a wall. Inside a room the recording will contain the sound which travels directly to the microphone, as well as an echo which traveled across the room, reflected off the far wall, and then into the microphone. Since the echo must travel further, it will be delayed in time. Also, since the speech is partially absorbed by the wall, it will be decreased in amplitude. For simplicity, ignore any further reflections or other sources of echo.

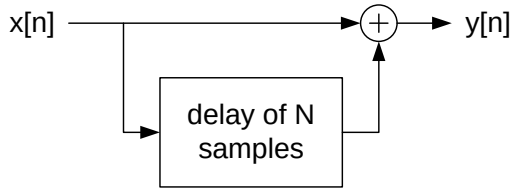

Figure 4: The graphical block diagram of eqn 10. Get used to being able to "see" either the block diagram or equation given the other; often problems lend themselves to a certain way of viewing them.

Matlab has the command `sound(y, Fs)` which plays a sound recorded in vector y that was recorded at sampling frequency Fs. If Fs is not entered it defaults to a value of 8.192kHz.

Problem 6

This project will use the audio capabilities of Matlab to play recordings of both the original sound with echo and the result of your processing. First load the speech file lineup.mat which is contained on the EE431 coursepage. This will load the vector y, which is a low-quality (Fs = 8.192kHz) recording of the words "line up" that sounds like a stadium recording with a strong echo. This signal was created using the model of eqn 10 from an uncorrupted recording with N = 1000, and an echo amplitude $\alpha = 0.5$. The original, uncorrupted sound vector x is not available, mimicking the real-world problem of echo cancellation. Listen to the sound by typing `sound(y)`.

Problem 6 continued

a) In this part you will remove the echo by linear filtering. Since the echo can be represented by a linear system of the form eqn 10, determine and plot its impulse response h[n] according to eqn 10. Store this impulse response in the vector hEcho for $0 \le n \le 1000$.

b) Consider an echo removal system described by the difference equation

$$z[n] + \alpha\, z[n-N] = y[n] \qquad \text{(eqn 11)}$$

where y[n] is the signal with the echo which is the input to the system, and z[n] is the output with the echo removed. Show that eqn 11 is indeed the inverse of eqn 10 by deriving the overall difference equation relating z[n] to x[n]. Hint: to do this, just prove that z[n] = x[n] is a valid solution to the overall difference equation.

c) Implement the echo removal system using `z=filter(1,a,y)`, where a is the appropriate coefficient vector derived from eqn 11. Plot the output using plot (it has too many samples to use with stem; the stem circles would overlap each other), and listen to the output using sound. The echo should be noticeably reduced (although the recording quality remains poor).

d) The echo removal system of eqn 11 will have an infinite-length impulse response, since a $y[n] = \delta[n]$ input will causes an initial $z[0] = 1$, which will be reflected N samples later as a $z[N] = \alpha$, which will in turn cause a later $z[2N] = \alpha^2$, and so on. This is an example of an infinite impulse response (IIR) filter. Assuming that N = 1000 and $\alpha = 0.5$, compute the impulse response using `filter` with an input that is an impulse given by `d=[1 zeros(1,4000)];` Store this 4001 sample approximation to the input response as hInverse and plot it.

e) **Challenge**: Calculate the overall impulse response of the cascaded echo system of eqn 10 and the echo removal system of eqn 11 by convolving hEcho with hInverse and store the results in hOverall. Plot the overall impulse response. You should notice that the result is not a perfect unit impulse. Given that you have computed hInverse to be the inverse of hEcho, why is this the case?

4. Matlab Commands Used in Laboratory 3

Matlab vector commands

min(v)	returns the smallest value in vector v
max(v)	returns the largest value in vector v

Linear filtering commands

conv(x,h)	returns the output vector y resulting from filtering x with impulse response h
filter(a,b,x)	returns the output y resulting from filtering x with the difference equation represented by vectors a and b

Other

sound(x,Fs)	plays the sound in vector x recorded at sampling frequency Fs
sound(x)	plays the sound in vector x, using an assumed sampling frequency of 8.192kHz.