Lab 4: Echoes and Images

The goal of this lab is to learn how to implement downsampling, convolution, and FIR filters in MATLAB, and then study the response of FIR filters to various signals, including images, speech, complex exponentials, and sinusoids. You will learn about how filters can create interesting effects, such as blurring and echoes. In addition, we will use FIR filters to study the convolution operation and properties such as linearity and time-invariance.

Lab Part One

1.1 Implementing the Three Point Averager

Before you begin this section, make sure that you have watched and understand the lecture videos on FIR filters. For a brief refresher, see the **Appendix** of this document.

The following code uses `firfilt()` to implement a three-point averaging system:

```
xx =[ones(1,10),zeros(1,5)];      %<-- Input signal
nn = 1:length(xx);                %<-- Time indices
bk = [1/3 1/3 1/3];               %<-- Filter coefficients
yy = firfilt(bk, xx);             %<-- Compute the output
yy = conv(bk, xx);                %<-- Equivalent method to compute output
```

To illustrate the filtering action of the 3-point averager, run the above code and then make a stem plot of the input signal and output signals in the same figure using the following code:

```
figure(3); clf
subplot(2,1,1);
stem(nn, xx(nn))
subplot(2,1,2);
stem(nn, yy(nn), 'filled')        %<-- Make black dots
xlabel('Time Index (n)')
```

What characteristics of the input signal are **most** affected by the averager?
What characteristics of the input signal are **least** affected by the averager?
(in other words, how is the output yy the most and least different than xx?)

The following is the relation between the lengths of the input and output signals and the length of the coefficients vector:

```
length(yy) = length(xx) + length(bk) - 1
```

Did your input, output, and impulse response fulfill this relation? If so, explain why; if not, explain why not.

1.2 An Echo Filter

The data file `labdat.mat` contains two filters and three signals, stored as separate MATLAB variables:

`x1`: a stair-step signal such as one might find in one sampled scan line from a TV test pattern image.
`xtv`: an actual scan line from a digital image.
`x2`: a speech waveform ("oak is strong") sampled at $f_s = 8000$ *samples per second*.
`h1`: the coefficients for a FIR discrete-time filter.
`h2`: coefficients for a second FIR filter.

Load the `labdat.mat` data file by downloading the file to your MATLAB directory and using the MATLAB command:

```
load labdat.mat
```

If you have any errors using these vectors, the `whos()` function will verify that the vectors are in your MATLAB workspace.

Echoes and reverberation can be done by adding a delayed version of the original signal to itself. The following is an FIR filter called an echo filter, as it plays a scaled version of the original signal at some time delay:

$$y[n] = x_1[n] + r x_1[n-P]$$

(a) Suppose you have an audio signal sampled at f_s= 8000 and you want to simulate an echo. What values of r and P will give an echo with strength 85% of the original, with time delay 0.22 s? Implement this echo filter and use it on the signal in vector `x2` (Hint: see lab part 1.1).

(b) Reverberation requires multiple echoes. This can be accomplished by cascading several systems of the above form. Using the parameters determined in part (a), derive (by hand) the impulse response of a reverb system produced by cascading five "single echo" systems. Recall that two filters are said to be "in cascade" if the output of the first filter is used as the input to the second filter, and the output of the second filter is defined to be the output of the overall cascade system. This can be repeated for as many filters as are needed in the cascade system. Filter the x2 signal with this reverb filter, and submit it as a .wav file. Using what you know about this filter's impulse response, explain why it sounds the way it does.

Optional [Visual Aid]: It will be difficult to make plots to show the echo and reverberation because of their size. The MATLAB function `inout()` can plot two very long signals together on the same plot by formatting the plot so that the input signal occupies the first, third, and fifth lines, etc. while the output signal is on the second, fourth, and sixth lines etc. Type `help inout` to find out more.

1.3 Image Processing

Note: It is a good idea to refer to the Appendix section “A.5 Introduction to Images” before working on this section and any time that you feel confused.

1.3.1 Show a Test Image

You can load the images needed for this lab from the included “.mat” files.

(Note: From now on, any file given by us with the extension “.mat” is in MATLAB format and can be loaded via the load command.)

You’ll find an image file in echart.mat. After loading, use the “whos” command to determine the name of the variable that holds the image and its size. Although MATLAB has several functions for displaying images on the monitor of the computer, we have written a special function show_img() for this lab. Here is the help on show_img.m, which should be in your SPFirst toolbox:

```
function [ph] = show_img(img, figno, scaled, map)
%SHOW_IMG display an image with possible scaling
% usage: ph = show_img(img, figno, scaled, map)
% img = input image
% figno = figure number to use for the plot
% if 0, re-use the same figure
% if omitted a new figure will be opened
% optional args:
% scaled = 1 (TRUE) to do auto-scale (DEFAULT)
% not equal to 1 (FALSE) to inhibit scaling
% map = user-specified color map
% ph = figure handle returned to caller
%----
end
```

Notice that unless the input parameter figno is specified, a new figure window will be opened.

Use show_img to display the image in echart.mat (Hint: It should look like an eye chart seen in an eye doctor’s office). Submit a screenshot of the image.

1.3.2 The Lighthouse Image

Load and display the 326 × 426 “lighthouse” image from lighthouse.mat.

(Note: When you display the image it *might* be necessary to set the color mapping via a command such as colormap(gray(256)).

Use the colon operator to extract the 225th row of the “lighthouse” image, and plot it as a discrete-time one-dimensional signal using the `plot()` function;

```
xx225 = xx(225,:);
```

Observe that the range of signal values is between 0 and 1. Which values represent white? Which values represent black? Where does the 225th row cross the fence? What features of the image correlate with the periodic-like portion of the plot? Please annotate your plot to support your answer.

1.3.3 Synthesize a Test Image

To probe your understanding of the relationship between MATLAB matrices and image display, generate a synthetic image from a mathematical formula.

Display this synthetic image in which all of the columns are identical by using the following *outer product*:

```
xpix = ones(256,1)*cos(2*pi*(0:255)/16);
```

How wide are the bands in number of pixels? How is this width related to the formula for `xpix`? How would you produce an image with horizontal bands?

Create (and display/submit) a 450 × 450 image with 4 horizontal black bands separated by white bands.

1.3.4 Sampling of Images

Images on a computer (digital storage) are sampled images stored in an $M \times N$ matrix. The sampling rate in the two spatial dimensions was chosen at the time the image was digitized. Just as a sinusoid is measured in dots per unit of time, a photograph is measured in dots per inch in each direction. For example, the image might have been “sampled” by a scanner where the resolution was chosen to be 300 (dots per inch). Lowering the sampling rate (*down-sampling)*, which compresses the image for the sake of transmission or storage, throws away samples in a periodic way. If every other sample is removed, the sampling rate will be halved.

Example: A 300 *dpi* image would become a 150 *dpi* image.

The following code down-samples by a factor of :

```
wp = ww(1:p:end,1:p:end);
```

However, just as down-sampling a sinusoid can result in aliasing, down-sampling an image can also result in aliasing. Load the `lighthouse.mat` file (if it isn’t loaded already).

Down-sample the lighthouse image by a factor of 2, and show the image. Note that the image is not square. What is the size of the down-sampled image?

Notice the difference in appearance of the image, despite there not being any added points. Describe how the aliasing appears visually. Which parts of the image most dramatically show the effects of the aliasing? Why does the aliasing manifest itself in these places?

From your row plot and from zooming in on the image, estimate the frequency of the aliased features in cycles per pixel. When estimating spatial frequency, consider reoccurring features of the images as ‘peaks’. From this you can obtain a period (how many pixels until the image ‘peaks’ again), and from there a frequency.

How does your estimation of aliased features fit into the Sampling Theorem? In other words, do the features that you expect to experience aliasing indeed do so, and why are those features aliased as opposed to others?

Lab Part Two

2.1 Image Reconstruction from Downsampling

2.1.1 Introduction to Interpolation

When an image has been sampled, we can fill in the missing samples through interpolation. For images, this would be analogous to the examples shown in the textbook for sine-wave interpolation, which is part of the reconstruction process in a *digital*-to-*analog* converter. We could use a "square pulse", "triangular pulse", or other pulse shape for the reconstruction.

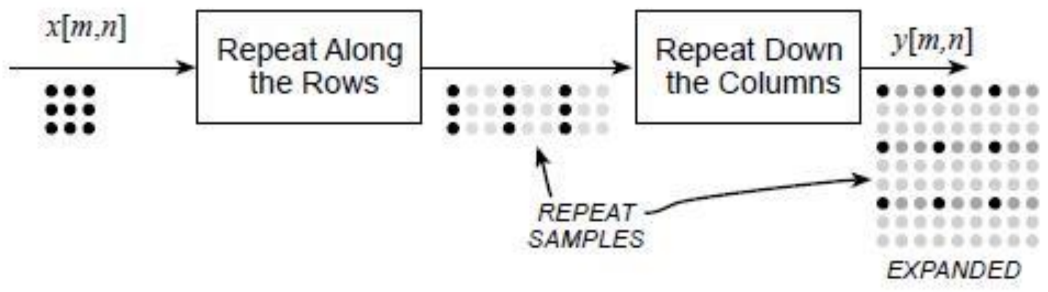


Figure 1: 2-D Interpolation broken down into row and column operations: the gray dots indicate repeated data values created by a zero-order hold; or, in the case of linear interpolation, they are the interpolated values.

A "square pulse" reconstruction is also known as a zero-order hold, and is the simplest form of interpolation for this case. Every point in the decimated array is duplicated in the row and column directions without change. Moving through the reconstituted array shows copies of each value before reaching a new value; hence this reconstruction is with a "square pulse".

The following code works for a one-dimensional signal (one row or one column or the image), assuming that we start with row vector `xr1` and the result is a row vector `xr1hold`:

```
xr1 = (-2).^(0:6);
L = length(xr1);
nn = ceil((0.999:1:4*L)/4); %<--- Round up to the
integer xr1hold = xr1(nn);
```

Plot the vector `xr1hold` to verify that it is indeed a zero-order hold derived from `xr1`. What values are in the indexing vector nn, and why are they what they are?

2.1.2 Interpolate a Lighthouse

(a) Load the "`lighthouse.mat`" image data (if you haven't already).

(b) Down-sample the lighthouse image by a factor of 3 (similar to what you did in section 1.3). Let's call the new array '`xx3`'.

(c) Perform a zero-order hold on `xx3` to fill in the missing points:

i. For an interpolation factor of 3, process all rows of `xx3`to fill in missing points in that direction. Call the result `xholdrows`. What are the dimensions of the `xholdrows` two-dimensional array?

ii. Now process all the columns of `xholdrows` likewise (interpolation factor 3, now processing in the other direction) to fill in the missing points in each column. Call this result `xhold`, and show the`xhold` and original lighthouse images on the same plot. Compare them and explain any differences that you can see. (Note that a zero-order hold will not produce a high quality reconstruction)

(d) Linear interpolation is sometimes more accurate. It assumes that if you have two values at points next to each other, you can take a weighted average of the two points to come up with a value for a point in between those two points. It is thus a "triangular pulse", interpolating on a sloped line. More full explanations are found at http://www.eng.fsu.edu/~dommelen/courses/eml3100/aids/intpol/ and https://en.wikipedia.org/wiki/Linear_interpolation .
Linear interpolation can be done in MATLAB using the interp1() function.

Read about the function options using '`help interp1`'. (You don't have to submit this help output; this is for you.) Its default mode is linear interpolation (equivalent to the '`*linear`' option), but interp1can also do other types of polynomial interpolation.

Here is an example on the same `xr1`signal that we used earlier in this section:

```
n1 = 0:6;
xr1 = (-2).^(0:6); % alternately xr1 = (-2).^n1;
tti = 0:0.1:6; % <--- locations between the n1
indices
xr1linear = interp1(n1, xr1, tti);
stem(tti, xr1linear);
```

Carry out linear interpolation operations on both the **rows** and the **columns** of the down-sampled lighthouse image `xx3`. Name and show the output `xr1linear`.

(e) Compare the output images:

i. Show the original image, the down-sampled image, the zero-order-hold reconstructed image, and the linearly-interpolated reconstructed image**.** Point out their differences and similarities.

ii. Can the linear interpolation reconstruction process remove the aliasing effects from the down-sampled lighthouse image?

iii. Can the zero-order hold reconstruction process remove the aliasing effects from the down-sampled lighthouse image?

iv. Point out regions where the linear and zero-order reconstruction result images differ and try to **justify** this difference in terms of the frequency content in that area of the image. In other words, look for regions of "low-frequency" and "high-frequency" content in the image and explain how the interpolation quality is dependent on this factor.

v. Are edges low-frequency or high-frequency features?

vi. Is the series of fence posts a low-frequency or high-frequency feature?

vii. Is the background a low-frequency or high-frequency feature?

2.2 Restoration Filter for 1-D Data

(a) The function

$$w[n] = x[n] - 0.9x[n-1]$$

acts as a FIR filter.

Use the function *firfilt*() or conv() to implement this filter on the following input signal:

```
xx = 256*(rem(0:100,50)<10);
```

(b) The following FIR filter can be used to undo the effects of the FIR filter in Eq 2.

$$y[n] = \sum_{\ell=0}^{M} r^{\ell} w[n-\ell]$$

It performs "restoration", but only approximately.

Process the filtered signal w[n] from part (a) with this restoration filter. Use r= 0.9 and M=22.

(c) Plot x[n], w[n], and y[n] on the same figure, using subplot. Make the discrete-time signal plots with MATLAB's stem function, but restrict the horizontal axis to the range 0 <= n <= 75. Note that the signals are not the same length.

(d) Make a plot of the error (difference) between y[n] and x[n] over the range 0 <= n < 50.

(e) Use the max() function to find the maximum difference between the above x[n] and y[n] over the range 0<=n<50. This is the worst-case error.

(f) What does the error plot and worst case error tell you about the quality of the restoration of x[n]?

2.3.1 Filtering Images: 2-D Convolution

One-dimensional FIR filters, such as running averagers and first-difference filters, can be applied to one-dimensional signals such as speech or music. These same filters can be applied to images if we regard each row (or column) of the image as a one-dimensional signal. For example, the 50th row of an image with N columns will be an N -point sequence, accessed as

$$xx[50, :]$$

We can filter this sequence with a 1-D filter using the `conv` or `firfilt`.

Multidimensional filtering can be accomplished with unidimensional row and column filters using loops. But MATLAB provides the optimized function `conv2()` that will do the convolution for each row/column with a single function call, instead of using a for loop to accomplish this (all the row-by-row conv() calls would take a while). It performs a more general filtering operation than row/column filtering, but since it can do these simple 1-D operations it will be very helpful in this lab.

(a) Load in the image file `echart.mat` with the `load` command. We can filter all the rows of the image at once with the `conv2()` function. To filter the image in the horizontal direction using a first-difference filter, we form a *row* vector of filter coefficients and use the following MATLAB statements:

bk = [1, −1];
rowfiltered = conv2(echart, bk);

In other words, the filter coefficients `bk` for the first-difference filter are stored in a *row* vector and will cause conv2() to filter all rows in the *horizontal* direction.

(b) Now filter the "eye-chart" image echart in the *vertical* direction with this first-difference filter to produce the image yy. This is done by calling yy=conv2(rowfiltered,bk'); note that bk', the transpose of bk, is now a column vector of filter coefficients.

(c) Using the show_img() function, display the input image echart , the intermediate image rowfiltered, and the output image yy. Compare the three images and give a qualitative description of what you see.

2.3.2 Distorting and Restoring Images

More complicated systems are often made up from simple building blocks. In the system of Fig. 3 two FIR filters are connected "in cascade." For this section, assume that the filters in Fig. 3 are described by the two equations:

$$w[n] = x[n] - qx[n-1] \qquad \text{(FIR FILTER 1)}$$

$$y[n] = \sum_{\ell=0}^{M} r^{\ell} w[n-\ell] \qquad \text{(FIR FILTER 2)}$$

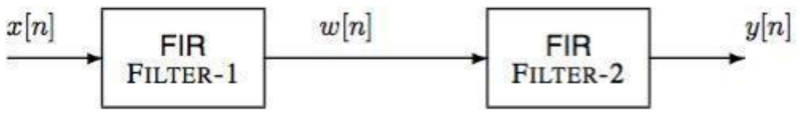


Figure 3: Cascading two FIR filters: the second filter attempts to "deconvolve" the distortion introduced by the first.

If we pick q to be a little less than 1.0, then the first system (FIR FILTER 1) will cause distortion when applied to the rows and columns of an image. The objective in this section is to show that we can use the second system (FIR FILTER 2) to undo this distortion (more or less). Since FIR FILTER 2 will try to undo the convolutional effect of the first, it acts as a *deconvolution* operator.

(a) Load the image in `echart.mat`. Pick $q = 0.9$ for the first filter (FIR FILTER 1). Using this filter, filter the echart image along the horizontal direction, and then filter the resulting image vertically. Call the result `echo90`.

(b) Convolve `echo90` with FIR FILTER 2, choosing $M = 22$and $r = 0.9$. Describe the visual appearance of the output qualitatively, showing the image, and explain its features by invoking your mathematical understanding of the cascade filtering process and why you see "ghosts" in the output image. Use some previous calculations to determine the size and location of the "ghosts" relative to the original image.

(c) Evaluate the *worst-case error* in order to say how big the ghosts are relative to "black-white" transitions which are 0 to 255. Make sure to show any code you used or plots to further your evaluation.

Hint: The resulting image after being passed through both filters is not in the same scale as the original image. To fix this, first you must set the minimum value of your image to 0, then normalize it, then scale it out of 255 (number of gray levels). Be wary of the dimensions of the images.

2.3.3 Second Restoration Experiment

(a) Now try to deconvolve `echo90` with several different FIR filters for FIR FILTER 2. You should set $r = 0.9$ and try several values for M, at the least $M = 11, 22, 33$. Pick the best result and explain why it is the best. Describe the visual appearance of the output qualitatively, and explain its features by invoking your mathematical understanding of the cascade filtering process.

Hint: determine the impulse response of the cascaded system and relate it to the visual appearance of the output image.

Hint: You can use `dconvdemo` to generate the impulse responses of the cascaded systems, as described in the Appendix.

(b) Furthermore, when you consider that a gray-scale display has 256 levels, how large is the worst-case error (from the previous part) in terms of number of gray levels? Evaluate worst-case error for each of the three filters in part (a) (and remember to normalize your outputs to 0 to 255 before you calculate this!) Can your eyes perceive a gray scale change of one level, i.e., one part in 256?
Include all images and plots for (a) and (b) to support your discussion.

Lab 4 Appendix

A.1 Introduction to FIR Filters

An FIR filter is a discrete-time system that converts a discrete-time input signal x[n] into a discrete-time output signal y[n] by means of the weighted summation:

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

The filter coefficients { b_k } are constants that define the filter's behavior.

A.2 Introduction to Averagers

For example, consider the system for which the output values are given by

$$y[n] = \frac{1}{3}x[n] + \frac{1}{3}x[n-1] + \frac{1}{3}x[n-2] = \frac{1}{3}(x[n] + x[n-1] + x[n-2])$$

For this filter, the b_k's are $b_0 = 1/3$, $b_1 = 1/3$, and $b_2 = 1/3$. This filter is a three-point averager, in that at each point it returns the average of the three most recent values of the input.

An L-point average has an output that is the average of the L most recent points of the input:

$$y[n] = \frac{1}{L}\sum_{k=0}^{L-1} x[n-k]$$

MATLAB has a built-in function called `firfilt()`. This function takes two arguments: the vector of filter coefficients and the input signal [ ]. The filter coefficients vector `bk` gets loaded as follows:

```
bk = [b0, b1, b2, …, bM];
```

Where `b0`, `b1`, `b2`, etc. are the filter coefficients. Note that `bk` must have a finite length to be implementable in MATLAB.

Note: The `conv()` function works similarly, *convolving* an input signal with filter coefficients. Additionally, the concept of convolution is fundamental to this course; therefore, once you feel confident with `firfilt()`, it might be in your best interest to try using conv() from now on.

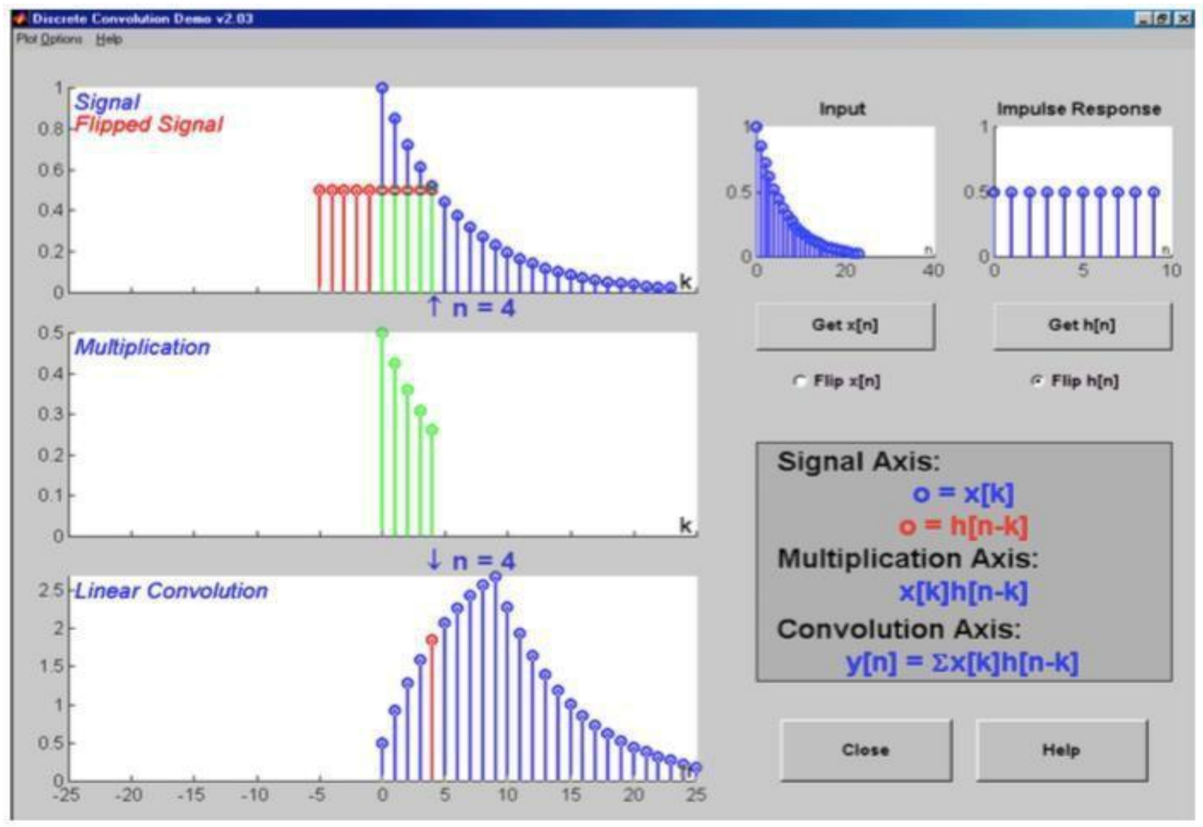
A.3 Note on "padding zeros" to a signal

In section 1.1 of this lab, the plotting range for both signals (xx and yy) is dependent on the length of the input signal *xx*, which, in this example, was "padded" with zeros. (Padding a signal simply means adding zeros to the end of the signal). This padding creates a signal burst followed by silence. While it may not be obvious at this point, by adding zeros, we are able to increase the frequency resolution; bear in mind the tactic of padding zeros as it will come into play in later labs as well.

A.4 Implementing FIR filters via a Graphical User Interface (GUI)

If you're having trouble visualizing FIR filters and convolution, the spfirst toolbox comes with a handy Graphical User Interface (GUI) that makes the process very transparent.

Run the `dconvdemo` GUI by typing `dconvdemo` into the command window. A window should pop up looking similar to this:



This GUI asks you to specify input in terms of the unit step function $u[n]$. Remember that the unit step function $u[n]$ is zero for all $n < 0$, and one for all $n \geq 0$.
In other words, $u[n]$ is

$$u[n] = \sum_{k=0}^{\infty} \delta[n-k]$$

(Below are some sample exercises from a previous version of this lab. If you are looking to understand convolution better either for this lab or a test, it is well worth the effort to complete these exercises. Please see a TA for the answers).

(a) Click on the "Get $x[n]$" button and set the input to a finite-length pulse:

$$x[n] = (u[n] - u[n-10])$$

Note: This is the same as:

$$\delta[n] + \delta[n-1] + \delta[n-2] + \delta[n-3] + \delta[n-4] + \delta[n-5] + \delta[n-6] + \delta[n-7] + \delta[n-8] + \delta[n-9]$$

(b) Set the filter to be a 3-point averager by using the Get $h[n]$ button to create the correct impulse response for the 3-point averager. Remember that the impulse response is identical to the b_k's for an FIR filter.

(c) Use the GUI to produce the output signal. Submit a screenshot of the GUI, showing input, impulse response, and output.

(d) Using the "Exponential" signal type within Get $x[n]$, set the input signal to be

$$x[n] = (0.9)^n(u[n] - u[n-10]).$$

Set the impulse response to be:

$$h[n] = \delta[n] - 0.9\delta[n-1] = (-0.9)^n(u[n] - u[n-2]),$$

using the Exponential signal type within Get $h[n]$. (Hint: length 2, multiplier −0.9). In other words, `bk = [1, -0.9]`.

Obtain the output signal $y[n]$ in the GUI (submit a screenshot).

Why is $y[n]$ zero for so many of the points? What are the index values n at which $y[n]$ is nonzero, and why?

Tip: When you move the mouse pointer over the index "*n*" below the signal plot and do a click-hold, you will get a *hand tool* that allows you to move the "*n*"-pointer. By moving the pointer horizontally you can observe the sliding window action of convolution. You can even move the index beyond the limits of the window and the plot will scroll over to align with "*n*."

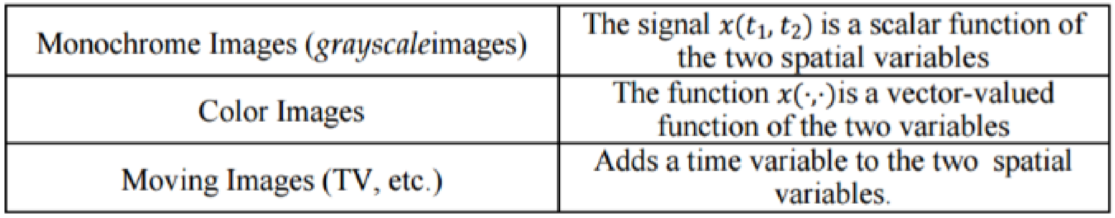
Note: These GUIs tend to be a little slow. If the current `dconvdemo` version freezes up MATLAB too much, you could download a new version of it from the SPFirst website and delete the old version of `dconvdemo` from your *spfirst* folder.

A.5 Introduction to Images

We have seen music as a sum of sinusoids. Likewise, digital images are also a signal type that can be studied to see the effect of sampling (analog to digital), aliasing, and reconstruction (digital to analog). An image is made up of tiny pixels represented as matrix coordinates

Digital Images

Think of an image as collection of "x and y" coordinates, or a matrix. This can be represented as a function $x(t_1, t_2)$ of two continuous variables representing the horizontal (t_2) and vertical (t_1) coordinates of a point in space.



Monochrome Images (*grayscale*images)	The signal $x(t_1, t_2)$ is a scalar function of the two spatial variables
Color Images	The function $x(\cdot,\cdot)$is a vector-valued function of the two variables
Moving Images (TV, etc.)	Adds a time variable to the two spatial variables.

In this lab we will consider only sampled gray-scale still images. A sampled gray-scale still image would be represented as a two-dimensional array of numbers of the form:

$$x[m,n] = x(mT_1, nT_2)$$

For m between 1 and some integer M, and n between 1 and some integer N, where T_1 and T_2 are the sample spacings in the horizontal and vertical directions. In MATLAB we can represent an image as a matrix, consisting of rows and columns. The matrix entry at (m,n), which is the sample value $x[m,n]$, is called a *pixel* (short for picture element).

Because light images are formed by measuring the intensity of reflected or emitted light (always a positive finite quantity), their values are always non-negative and finite in magnitude. When stored in a computer or displayed on a monitor, the values of $x[m,n]$ have to be scaled relative to a maximum value X_{max}. With the typical 8-bit integer representation, the maximum value (in the computer) would be

$$X_{max} = 2^8 - 1 = 255$$

Meaning there would be $2^8 = 256$ different gray levels for the display, from 0 to 255.

To summarize, in this lab each element in a 2-dimensional picture vector, $x[m,n]$, holds some integer value from 0 to 255, inclusive, where "255" means white, "0" means black, and values in between are some shade of gray.

Displaying Images

As you will discover, correct display of an image on a grayscale monitor can be tricky, especially after some processing has been performed on the image. Luckily, we have provided to us the function `show_img.m` in the *SPFirst Toolbox* to handle most of these problems, but it will be helpful if the following points are noted:

(a) All image values must be non-negative for the purposes of display. Filtering may introduce negative values, especially if differencing is used (high-pass filter).

(b) The default format for most gray-scale displays is eight bits, so the pixel values $x[m, n]$ in the image must be converted to integers between 0 and 255

(c) The actual display on the monitor is created with the `show_img()` function - which handles the color map and the "true" size of the image. The appearance of the image can be altered by running the pixel values through a "color map." In our case, we want "grayscale display" where all three primary colors (red, green and blue, or RGB) are used equally, creating what is called a "gray map." In MATLAB the gray color map is set up via `colormap(gray(256))` which gives a 256×3 matrix where all 3 columns are equal. The function `colormap(gray(256))` creates a linear mapping, so that each input pixel amplitude is rendered with a screen intensity proportional to its value (assuming the monitor is calibrated). For our lab experiments, non-linear color mappings would introduce an extra level of complication, so they will not be used.

(d) When the image values lie outside the range $[0, 255]$, or when the image is scaled so that it only occupies a small portion of the range $[0, 255]$, the display may have poor quality. In this lab, we will use `show_img.m` to *automatically rescale the image:* This requires a linear mapping of the pixel values.

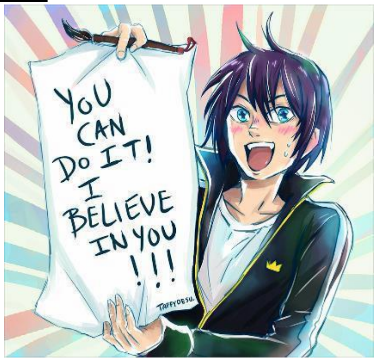
$$x_s[m, n] = \mu x[m, n] + \beta$$

The scaling constants μ and β can be derived from the min and max values of the image, so that all pixel values are recomputed via:

$$x_s[m, n] = \left\lfloor 255.999 \left(\frac{x[m, n] - x_{min}}{x_{max} - x_{min}} \right) \right\rfloor$$

Where $\lfloor x \rfloor$ is the floor function, i.e., the greatest integer less than or equal to x.

A.6 Inspirational Imagery



♥ ~Trey