EE 301 Lab 8 – Image Processing

Digital filters can be used to modify and improve signals. Their attractive quality is their simplicity. Before the use of digital signals, filtering of analog signals was accomplished with RLC circuits. Now, filtering is generally accomplished digitally with simple and fast routines.

One of the biggest reasons to filter a signal is noise reduction (which can also be related to signal recovery). Noise is a signal with undesirable frequency components, e.g. it is a signal that contains frequencies that are of little interest to us. Frequency blocking filters can be use to block signal component at certain frequencies, or reduce the relative amplitude of the signal at specific frequencies. Low-pass filters block high frequency components, high-pass filters block low frequency components, and band-pass filters block all frequencies except frequencies in a particular range.

Filtering is also used widely in image processing. They are often used to improve the appearance of an image. For example, if an image has granular noise, we can smooth or blur the image to enhance the appearance. The smoothing effect is accomplished with a low-pass filter because this type of noise relative to the signal itself is predominantly higher in the high-frequency range. Alternatively, we can also sharpen an image to enhance edges by using a high-pass filter.

In this lab, we will examine a class of filters called FIR (finite impulse response) filters. FIR filtering operation simple to implement and is almost identical to the operation of running correlation. We will use FIR filters for both smoothing and sharpening effects in image processing.

1 What you will learn

In this lab we will be examining how to implement FIR filters in MATLAB. We will also be interested in improving the appearance of an image and how to remove noise or sharpen an image.

2 Background Information and Notes

2.1 Implementing FIR Filters

An FIR filter takes an input signal x[n], modifies it by the application of an equation called the difference equation, and produces an output signal y[n]. The difference equation is a weighted sum of samples of the input signal x[n]. A common form of the difference equation is given as,

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] \tag{2.1}$$

$$= b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + \ldots + b_M x[n-M] \tag{2.2}$$

The b_k's are known as the *FIR filter coefficients*, and M is the *order* of the FIR filter. The order and the coefficients can represent different kinds of filters (for example, low-pass, high-pass and band-pass filters.)

Equation (2.1) describes *causal* FIR filters. A more general expression is,

$$y[n] = \sum_{k=-M_1}^{M_2} b_k x[n-k] \tag{2.3}$$

$$\begin{aligned} &= b_{-M_1} x[n+M_1] + \ldots + b_{-1} x[n+1] \\ &\quad + b_0 x[n] + b_1 x[n-1] + \ldots + b_{M_2} x[n-M_2] \end{aligned} \tag{2.4}$$

where M_1 and M_2 are nonnegative integers. In this case, the order of the filter is $M_1 + M_2$. If $M_1 > 0$, the FIR filter is *non-causal*. The distinction and knowledge between causal and non-causal filters is necessary if we wish to implement these filters in real-time.

Let us compare equation (2.3) with the equation for running correlation between a signal b[n] and x[n]:

$$y[n] = C(b[k], x[k-n]) = \sum_{k=-\infty}^{\infty} b[k] x[k-n] \tag{2.5}$$

Running correlation involved a procedure where we slid one signal across the other, and calculated the in-place correlation at each step. The application of an FIR filter uses the same except that we assume $b_k = 0$ for k outside the range $[M_1, M_2]$, and thus can change the limits from negative infinity to positive infinity without affecting the output. Also, the signal x[n] is time-reversed with respect to the b_k coefficients[1].

2.2 Edge effects and delay

Let's consider filtering a signal, x[n], with a causal filter whose difference equation is given by,

$$y[n] = \frac{1}{5}x[n] + \frac{1}{5}x[n-1] + \frac{1}{5}x[n-2] + \frac{1}{5}x[n-3] + \frac{1}{5}x[n-4] \tag{2.6}$$

We can see that, $b_k = (1/5, 1/5, 1/5, 1/5, 1/5)$, $M_1 = 0$, and $M_2 = 4$. This filter is basically replacing each sample of x[n], with the average of that sample and the past four samples. These type of filters are called *moving-average filters*. Shown in Figure 8.1, is a result of a moving-average filter on a particular signal.

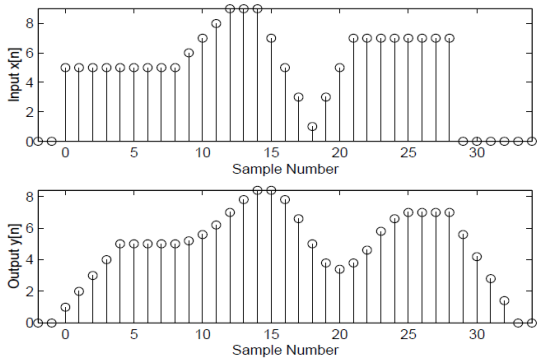

Figure 8.1: Input and output of a 5 point moving average filter.

One can see that x[n] has a support interval of [0, 28] and is zero outside of this range. Near the edge of this range, for example at y[0], we can see that y[0] will be dominated by a calculation with zeros,

$$y[0]=\frac{1}{5}x[n]+\frac{1}{5}0+\frac{1}{5}0+\frac{1}{5}0+\frac{1}{5}0 \tag{2.7}$$

Similarly, y[1], y[2], y[3], and y[4] will also contain some of these zeros. This effect can be seen in Figure 8.1 as y[n] values increase to the nominal values of x[n], and is known as a *start-up transient*. Similarly at the other edge at y[28], where the signal decreases or dies away, which is known as an *ending transient*. These transients are called *edge effects*. One can also see that from Figure 8.1, y[n] has a support length four samples longer than that of x[n]. In general, the length of the output signal will be equal to the length of the input signal plus the order of the FIR filter.

It is also of importance to notice the location of the dip in the signal in Figure 8.1. The dip occurs at sample 18 for x[n], but at sample 20 for y[n]. Not only has the support interval of y[n] increased, it has also been delayed by two samples. The delay is the result of each output sample being an average of samples to its left. If instead we define a filter that considers two samples to both the right and left, we can eliminate this delay. This difference equation is then,

$$y[n]=\frac{1}{5}x[n+2]+\frac{1}{5}x[n+2]+\frac{1}{5}x[n]+\frac{1}{5}x[n-1]+\frac{1}{5}x[n-2] \tag{2.8}$$

However the result is a non-causal filter. For causal filters, delay is a common problem. The only causal filter that does not introduce delay is a zero-order amplifier system with a difference equation y[n] = b_0x[n]. This system changes only the amplitude of a signal. In this lab, when comparing two signals with a mean-squared or RMS distortion measure, we need to account for the delay introduced by FIR filters.

2.3 Noise and distortion

A filter can be used to remove noise. While there are several definitions of noise, a common type of electrical noise has a sinusoidal characteristic with a frequency of 60 Hz. This noise originates from the AC source used to distribute electricity. This 60 Hz signal can tunnel into other systems and corrupt measurements. Another common and well-known type of noise is *random noise*. Random noise typically has a jagged-looking characteristic, and appears as static in audio signals and or as “snow” in images.

We could use a filter to remove the noise, however it requires that the noise to be at a different frequency than the signal of interest.

Random noise typically has frequency components all across the frequency spectrum. However, a large portion of these components will usually have a higher frequency than the frequencies in our signal. Thus, a *low-pass filter* could be used to block the high frequencies and reduce the amplitude of noise components.

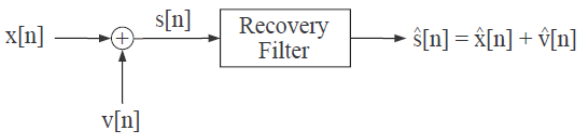

Figure 8.2: A block diagram of additive noise and a recovery filter that attempts to remove the noise.

The block diagram of Figure 8.2, shows a signal of interest, x[n], corrupted by the addition of a noise signal, v[n]. A recovery filter is applied to remove the noise component from s[n] = x[n] + v[n]. The resulting signal is $\hat{s}$ [n] = $\hat{x}$ [n] + $\hat{v}$ [n], where $\hat{x}$ [n] is the filtered signal (which we hope is similar to x[n]) and $\hat{v}$ [n] is the filtered noise signal (which we hope is small). By increasing the length of a moving average filter, we can eliminate more noise (or make the filter stronger). Unfortunately, this process

distorts the signal of interest. The goal is then to find the point where the total distortion (as measured by the mean-squared error or RMS error between x[n] and $\hat{S}$ [n]) is minimized as a function of filter strength.

Nonlinear filtering

An alternative is to use *nonlinear* filters, which have the potential to remove more noise while introducing less distortion to the desired signal. However, these filters are much more difficult to analyze.

One of the most important features of an image of a natural scene is the *edges*. An edge in an image is a sharp transition where one object ends and another begins. We know that to remove high-frequency noise from an image, we would apply a low-pass filter. However edges have considerable high-frequency content, so a low-pass filter would smooth out the edges in the image. To circumvent this problem, a nonlinear filter called a *median filter* could be used. Median filters replace each sample of a signal with the median (or the most central value) of a group of samples around the original sample. A median filter is described as,

$$y[n] = Median\left(x\left[n+M_1\right],...,x\left[n\right],...,x\left[n-M_2\right]\right) \tag{2.9}$$

where

$$Median\left(x_1,...,x_N\right) = \begin{cases} x_{((N+1)/2)} & N \text{ odd} \\ \frac{1}{2}\left(x_{(N/2)} + x_{(N/2+1)}\right) & N \text{ even} \end{cases} \tag{2.10}$$

and where $x_{(n)}$ is the nth smallest of the values x_1 through x_N. The *order* of the median filter is $M_1 + M_2$, and determines how many samples will be included in the median calculation. This filter is non-causal because its output depends on future, as well as past and present, inputs. In this lab, we will examine the effect of applying nonlinear filters to two-dimensional signals.

2.4 Filtering two-dimensional signals

There are several approaches in considering two-dimensional signals, like images. The first approach involves applying one-dimensional filters to each of the rows (or each of the columns) of an image. This approach tends to produce an uneven filtered signal, for example, it could be smoothed in one dimension but not the other. The second approach applies one-dimensional filters to *both* the rows and the columns.

In this lab we will first filter the columns, and then filter the rows of the resulting image. Most types of one-dimensional filters can be extended to two dimensions in this sense. If we apply the moving average filter with difference equation described as equation (2.6), this will have the effect of smoothing the image. The edge effects and delay issues discussed, also apply to two-dimensional filtering.

To apply a moving average filter with order 4 to the columns and then the rows of the image, we see that each sample of the image has been replaced by the average of a 5 x 5 block of pixels. Thus this filtering operation must be described in terms of two-dimensional set of filtering coefficients. The difference equation for this two-dimensional filter would then be,

$$y[m,n] = \sum_{k=0}^{4}\sum_{l=0}^{4}\frac{1}{25}x\left[m-k,n-l\right] \tag{2.11}$$

This operation involves a two-dimensional set of coefficients $b_{k,l}$, where $b_{k,l} = 1/25$ for k = 0,…,4 and l=0,...,4. The general difference equation for this approach is then,

$$y[m,n] = \sum_{k=-M_1}^{M_2} \sum_{l=-N_1}^{N_2} b_{k,l} x[m-k, n-l] \tag{2.12}$$

where $[-M_1, M_2]$ and $[-N_1, N_2]$ define the range of non-zero coefficients. Notice that a filter described by equation 2.11, is *causal* if M_1 and N_1 are non-negative. However, in image processing, causality is rarely important., and thus, two-dimensional FIR filters typically have coefficients centered around $b_{0,0}$. A schematic of a set of filter coefficients is shown in Figure 8.3.

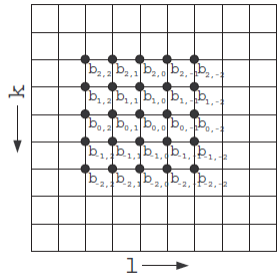

Figure 8.3: The coefficients of a two-dimensional moving average filter. In this figure, pixels exist at the intersection of the horizontal and vertical lines.

2.5 Image processing with FIR filters

In photo editing software, you may have seen operations called “smoothing” and “sharpening”. These and many other similar operations are typically implemented using two-dimensional FIR filters. In this lab, we will consider the following three operations: smoothing (or *blur*), *edge finding*, and sharpening (or *edge enhancement*).

Although we have already suggested that a moving average filter performs a smoothing operation, there are other more advanced ways of smoothing. Consider, a filter that weights samples nearby more strongly than those that are far away. This performs a weaker smoothing effect, but also introduces less distortion. Thus, these sorts of filters are more useful for random noise reduction compared to standard moving average filters.

The “edge finding” filter accentuates edges in an image by producing large positive or negative values, whilst setting constant regions of the image to zero. An edge finding filter is a one dimensional *first difference* filter, with the following difference equation,

$$y[n] = x[n] - x[n-1] \tag{2.13}$$

This filter will subsequently respond positively to increases in the signal and negatively to decreases in the signal. Adjacent input samples that are identical (or nearly identical), will tend to cancel one another, causing the output to be zero (or close to zero). Another general edge-finding filter has the following difference equation:

$$y[n] = \frac{1}{4}x[m+1,n+1] - x[m+1,n] + \frac{1}{4}x[m+1,n-1] - x[m,n+1] + 3x[m,n] - x[m,n-1] + \frac{1}{4}x[m-1,n+1] - x[m-1,n] + \frac{1}{4}x[m-1,n-1] \quad (2.14)$$

This filter "finds" edges of *almost any orientation* by outputting a large magnitude value wherever an edge occurs. Both the first difference filter and this general edge-finding filter are high-pass filters. Note the pattern of b_k values. The adjacent coefficients are negatives of one another. This is a known characteristic of high-pass filters. Also note that for both of these filters, the average of the b_k coefficients is zero. Thus these filters tend to reject constant regions of an input signal, and set them to zero.

The edge finding filter can also be used to sharpen an image. The sharpening filter produces a weighted sum of the output of an edge-finding filter and the original image. Let's suppose that x[m,n] is the original image, and y[m,n] is the result of filtering x[m,n] with the filter defined in equation (2.14). The result of sharpening, z[m,n] is,

$$z[m,n] = x[m,n] + by[m,n] \quad (2.15)$$

where b controls the amount of sharpening. A larger value of b produce a sharper image. Notice that z[m,n] also looks like an output of a single filter. For display purposes, we will have a threshold such that the output image has the same data value range as the input image. For example, if the image has values between 0 and 255, the final output of the sharpening operation, $\hat{z}$ [m,n] will be

$$\hat{z}[m,n] = \begin{cases} 0 & z[m,n] < 0 \\ z[m,n] & 0 \le z[m,n] \le 255 \\ 255 & 255 < z[m,n] \end{cases} \quad (2.16)$$

Thresholding is actually a *nonlinear* operation, but it is not crucial to the sharpening process. It can also be considered to be the output of a single nonlinear filter.

Unlike smoothing filters, though, sharpening filters tend to enhance random noise, and can make noise-like components of a signal visible where they were not visible before.

2.6 Some MATLAB commands for this lab

- **1-D Filtering in** MATLAB**:** The usual method for causal filtering in MATLAB is to use the *filter* command, like this:

  >> yy = filter(bb,1,xx);

  xx is a vector containing the discrete-time input signal to be filtered, bb is a vector of the b_k filter coefficients, and yy is the output signal. The first element of this vector, bb(1), is assumed to be b_0.

  By default, *filter* returns a portion of the filtered signal equal in length to xx. Specifically, the resulting signal includes the start-up transient but not the ending transient. This means that the output will be delayed by an amount determined by the coefficients of the filter.

A method for filtering which does not introduce delay is often desirable, i.e. a noncausal filtering method, especially when calculating RMS error between filtered and original versions of a signal. The command *filter2* is meant as a two-dimensional filtering routine, but it can be used for 1-D filtering as well. Further, it can be instructed to return a "delay-free" version of the output signal. When using *filter2*, it is important that xx and bb are either both row vectors or both column vectors. Then, we use the command

```
>> yy = filter2(bb,xx,'same');
```

where xx is the input signal vector, yy is the output signal vector, and bb is the vector of filter coefficients. If the length of the vector bb is odd, the b0 coefficient is taken to be the coefficient at the center of the vector bb. If the length of bb is even, b0 is taken to be just left of the center. The output of *filter2* has support equal to that of the input signal xx.

Though we will not use these additional options, we can also have *filter2* return the full length of the filtered signal (the length of the input signal plus the order of the filter) like this:

```
>> yy = filter2(bb,xx,'full');
```

or just the portion not affected by edge effects (the length of the input signal minus twice the order of the filter), like this:

```
>> yy = filter2(bb,xx,'valid');
```

- **2-D Filtering in** MATLAB**:** Three approaches to filtering a two-dimensional signal were mentioned in Section 2.4. The first approach, which simply applies a one-dimensional filter to each row of the image (alternatively, to each column) can be implemented with the MATLAB commands described in the previous bullet. The second approach applies a one-dimensional filter first to the columns and then to the rows of the image produced by the first stage of filtering. If the one-dimensional filter is causal with coefficients bk contained in the MATLAB vector bb and the image is contained in the 2-dimensional matrix xx, then this approach can be implemented with the command

  ```
  >> yy = filter(bb,1,filter(bb,1,xx)')';
  ```

  Note that we do not need to vectorize the image xx, because when presented with a matrix, *filter* applies one-dimensional filtering to each column. However, to perform the second stage of filtering (on rows of the image produced by the first stage), we need to transpose the image produced by the the first stage of filtering and then transpose the final result again to restore the original orientation. This approach will introduce edge effects at the top and on one side of the image; however the resulting image will be the same size as x.

  The third approach uses a two-dimensional set of coefficients $b_{k,l}$. If these coefficients are contained in the matrix bb and the image is contained in the matrix xx, then the filter can be implemented with the command

  ```
  >> yy = filter2(bb,xx,'same');
  ```

  Note that the 'same' parameter indicates that the filter is non-causal and thus the $b_{0,0}$ coefficient is located as near to the center of the matrix bb as possible. The same alternate third parameters for *filter2* that are listed in the 1-D filtering section apply here as well.

- **Generating filter coefficients:** We will be examining the effects of many types of filters in this lab. Some have filter coefficients that can be generated easily in MATLAB. Others require a function (which we will provide to you) to generate. Note that the the vectors representing the b_k's will be column vectors.
  1. An L-point moving average filter has filter coefficients given by bb = ones(L,1)/L.
  2. A first-difference filter has filter coefficients given by bb = [1; -1];
  3. The function *g_smooth* produces coefficients for a particular type of smoothing (low-pass) filters with easily tunable "strength". *g_smooth* takes a single real-valued parameter, which is the "width" of the filter, and returns a set of tapered filter coefficients of the corresponding filter, bb. For example,

     >> bb = g_smooth(1.2);

     returns the coefficients for a filter with "width" 1.2. A *g_smooth* filter with width 0 will pass the input signal without modification, and higher widths will smooth more strongly. Good nominal values for the width range from 0.5 to 2. The b_k coefficients for *g_smooth* filters with several widths are plotted in Figure 8.4.

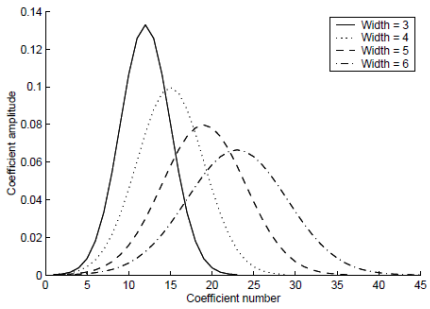

Figure 8.4: The coefficients for g_smooth filters with varying widths.

  4. The function *g_smooth2* is the two-dimensional equivalent of *g_smooth*. It again takes a single input parameter (the width of the filter) and returns the two-dimensional set of filter coefficients of the corresponding filter. For example,

     >> bb = g_smooth(0.8);

     returns the coefficients of a filter with width 0.8.
  5. We presented a general-purpose two-dimensional edge-finding filter in equation (2.14). The coefficients for this filter are given by

     >> bb = [.25, -1, .25; -1, 3, -1; .25, -1, .25];

  6. In Section 2.5, we also discussed a method for implementing a sharpening filter. Since we include a threshold operation, this operation is nonlinear and cannot be accomplished using only an FIR filter. Thus, we provide the sharpen command, which takes an image and a sharpening "strength" and returns a sharpened image:

     >> yy = sharpen(xx,0.7);

     The second parameter is the strength factor, b. A sharpening strength of 0 passes the signal without modification.

7. Median filters are a special type of nonlinear filter, and they cannot be described using linear difference equations. To use a median filter on a one-dimensional signal, we use the command *medfilt1* like this:

   >> yy = medfilt1(xx,N);

   N is the order of the median filter, which simply describes how many samples we consider when taking the median. In two dimensions3, we use *medfilt1* twice:

   >> yy = medfilt1(medfilt1(xx,N)',N)';

   Again, N is the order of the median filter. Here, we are using a one-dimensional filter on both the rows and columns of the image. Note that since *medfilt1* operates down the columns, we need to transpose the image between the filtering operations and again at the end.

3 Guided Exercises

1. (Noise reduction in 1-D) In this problem, you investigate noise-reduction on one-dimensional signals. Download the file **lab8_data.mat**, which contains various signals for this lab. In this problem, we will consider the signal **simple**, which is a noise-free one-dimensional signal, and **simple_noise**, which is the same signal with corrupting random noise.

   (a) (Effects of delay) First, we'll examine the delay introduced by the two filtering implementations, **filter** and **filter2**, that we will be using. Filter **simple** with a 7-point running average filter. Do this twice, first using **filter** and then using **filter2** with the 'same' parameter.
   - Use **subplot** and **plot** to plot the original signal and two filtered signals in three subplots of the same figure.
   - One of the filtering commands has introduced some delay. Which one? How many samples of delay have been added?
   - Compute the mean-squared error between the original signal and the two filtered signals. Which is lower? Why?

   (b) (Measuring distortion in 1-D) Now, use **filter2** to apply the same 7-point running average filter to the signal **simple_noise**. Referring to Figure 8.2, we consider **simple** to be the signal of interest x[n], **simple_noise** to be the noise corrupted signal s[n], and their difference to be the noise, v[n] = s[n] − x[n]. Note that the lower of the two mean-squared errors that you computed in Problem 1a is MS($\hat{x}$ [n] − x[n]), which is a measure of the distortion of the signal of interest introduced by the filter.
   - Compute the mean-squared error between **simple** and **simple_noise**. Referring back to Figure 8.2, this is MS(v[n]), the mean-squared value of the noise.
   - Compute the mean-square error between your filtered signal and **simple**. This value is MS($\hat{s}$ [n]− x[n]), which is a measure of how a good a job the filter has done at recovering the signal of interest.
   - Determine the distortion due to noise at the output of your reconstruction filter (i.e., MS($\hat{v}$ [n])) by subtracting MS($\hat{x}$ [n] − x[n]) from MS($\hat{s}$ [n] − x[n]).
   - Compare MS($\hat{v}$ [n]) and MS($\hat{s}$ [n]−x[n]) to MS(v[n]). What is the dominant source of distortion in this filtered signal?

(c) (Running average filters in 1-D) Use **filter2** to apply a 3-point, a 5-point, and an 9-point moving average filter to **simple_noise**.
- Use subplot, subplot, and stem to plot the original signal, the three filtered signals, and the three sets of filter coefficients, in seven panels of the the same figure. (Use plot for the signals and stem for the coefficients.)
- Compute the mean-squared error between each filtered signal and **simple**.
- Which of the four moving average filters that you have applied has the lowest mean-squared error? Compare this value to MS(v[n]).

(d) (Tapered smoothing filter in 1-D) Download the file **g_smooth.m**, and use it to generate filter coefficients with "widths" of 0.5, 0.75, and 1.0. (Note the lengths of the returned coefficient vectors. You should plot the filter coefficients to get a sense of how the "width" factor affects them.) Use **filter2** to apply these filters to **simple_noise**
- Use **plot** and **subplot** to plot the three filtered signals and the three sets of coefficients in six panels of the same figure.
- Compute the mean-squared error between each filtered signal and **simple**.
- Which of these filtered signals has the lowest mean-squared error? Compare this value to the lowest mean-squared error that you found for the moving average filters and to MS(v[n]).

2. (Noise reduction on images) In this problem, you look at the effects of applying smoothing filters to an image for noise reduction. Download the files **peppers.tif** and **peppers_noise1.tif**. The first is a "noise-free" image, while the second is an image corrupted by random noise. Load these two images into MATLAB.

(a) Examining 2-D filter coefficients) We'll be using the function **g_smooth2** to produce filter coefficients for this problem. To get a sense of what these coefficients look like, generate the coefficients for a **g_smooth2** filter with width 5. In two side-by-side subplots of the same figure:
- Display the coefficients as an image using **imagesc**.
- Generate a surface plot of the coefficients using the command **surf(bb)** (assuming your coefficients matrix is called bb).

(b) Examine the effects of noise) First, we'll consider the noisy signal **peppers_noise1**.
- Use **subplot** to display **peppers** and **peppers_noise1** side-by-side in a single figure. Remember to set the color map, set the axis shape, and include a colorbar.
- Compute the mean-squared error between these two images.

(c) (Minimizing the MSE) Our goal is to find a **g_smooth2** reconstruction filter that minimizes the mean squared error between the filtered image and the original, noise-free image. Use **filter2** when filtering signals in this problem.
- Find a filter width that minimizes the mean-squared error. What is this filter width and the corresponding mean-squared error? (Hint: you might want to plot the mean-squared error as a function of filter width.)
- Display the filtered image with the smallest mean-squared error.
- Look at some filtered images with different widths. Can you find one that looks better than the minimum mean-squared error image? What filter width produced that image?

3. (Salt and pepper noise in images) Next, we'll look at methods of removing a different type of random noise from this image. Download the file **peppers_noise2.tif** and load it into MATLAB. This signal is corrupted with *salt and pepper* noise, which may result from a communication system that loses pixels.

(a) (Examining the noise) Salt and pepper noise randomly replaces pixels with a value of either 0 or 255. In this image, one-fifth of the pixels have been lost in this manner.
- Display **peppers_noise2**.
- Compute the mean-squared error between this image and **peppers**.

(b) Using lowpass filters) Now, let's try using some **g_smooth2** filters to eliminate this noise. Start by using **filter2** to filter **peppers_noise2** with a **g_smooth2** filter of width 1.3. Note that this is very close to the optimal width value.
- Display the resulting image.
- Compute the mean-squared error.

(c) (Using median filters) Finally, let's use a median filter to try to remove this noise. Apply median filters of order 3 and 5 to **peppers_noise2**.
- Use **subplot** to display the two filtered images side-by-side in the same figure.
- Compute the mean-squared errors between the median-filtered signals and **peppers**.
- Look at the filtered images and describe the distortion that the median filters introduce into the signal.
- Compare the median filter to the **g_smooth2** filters. Discuss both measured distortion and the appearance of the filtered signals.

4. (Edge-finding and enhancing) In this last problem, we'll look at edge-finding and sharpening filters.

(a) (Applying a first difference filter) In order to see how edge-finding filters work, let's start in one dimension. Use **filter** to apply a one-dimensional first difference filter to the signal **simple** (which can be found in **lab8_data.mat**).
- Plot the resulting signal.
- There are five non-zero "features" of this signal. (These features should be clear from the plot.) Describe them and what they correspond to in simple.

(b) ("Finding" edges) Now we'd like to look at the effects of the general edge-finding filter. Use **filter2** to apply this filter to peppers.
- Display the resulting image.
- Describe the resulting image.
- Zoom in on the filtered image and examine some of the more prominent edges. What do you notice about these edges? (Hint: Are they just a "ridge" of a single color?)

(c) (Sharpening an image) Download **sharpen.m** and use the function to display several sharpened versions of the **peppers** image.
- Use **subplot** to display the sharpened image with a "strength" of 1 next to the original **peppers** image.
- Zoom in on this sharpened image. What makes it look "sharper"? (Hint: Again, look at the prominent edges of the images. What do you notice?)
- The sharpened images (especially for strengths greater than 1) generally appear more "noisy" than the original image. Speculate as to why this might be the case.

(d) (Using sharpening to remove smoothing) Finally, we want to try using the "sharpen" function to undo a blurring operation. Download the file **peppers_blur.tif** and load it into MATLAB.
- Compute the RMS error between **peppers** and **peppers_blur**.
- Use sharpen to "de-blur" the blurred image. Find the sharpening strength that minimizes the RMS error of the "de-blurred" image. Include this strength and its corresponding RMS error in your report.

- Display the "de-blurred" image with the minimum RMS error and alongside **peppers_blur** using **subplot**. Include the resulting figure in your report.

Note that sharpening is very much a perceptual operation. The minimum distortion sharpened image may not look terribly much improved. Look at what happens as you increase the sharpening factor even more. With additional "sharpening," the (measured) distortion may increase, but the result looks better perceptually.

4 Review

1. None.

5 Lab Report

1. The first page of your Lab report should be a cover sheet with your name, USC ID and Lab #. Please note that all reports should be typed.
2. Answer all the questions which were asked in the lab report. Kindly display the code lines you executed to arrive at your answer along with figures to support them. Please give written explanation or put comment lines where necessary. Please note that each figure should have proper labels for the x and y axis and should have a suitable title.
3. Answer the review questions.
4. Submit a printout of your completed M-file documenting all the lab exercises.

This lab document and the figures contained were adapted from a University of Michigan Signals and Systems course lab handout (2002).