Taylor Rembos
EEL 3135
Lab 4 Part 2

Lab 4: Echoes and Images

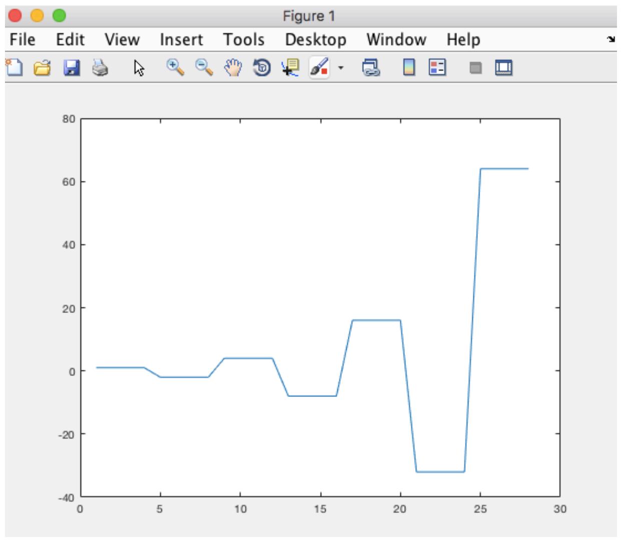
Lab Part Two

2.1 Image Reconstruction from Downsampling

2.1.1 Introduction to Interpolation

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?

```
>> xr1 = (-2).^(0:6);
>> L = length(xr1);
>> nn = ceil((0.999:1:4*L)/4);
>> xr1hold = xr1(nn);
>> nn

nn =

  Columns 1 through 8

     1     1     1     1     2     2     2     2

  Columns 9 through 16

     3     3     3     3     4     4     4     4

  Columns 17 through 24

     5     5     5     5     6     6     6     6

  Columns 25 through 28

     7     7     7     7
```



Zero-order hold because without interpolation, adding more values, additional values are still produced. The values of xr1 are a repetition of values 1 through 7 in order to maintain a regularly increasing sample.

2.1.2 Interpolate a Lighthouse

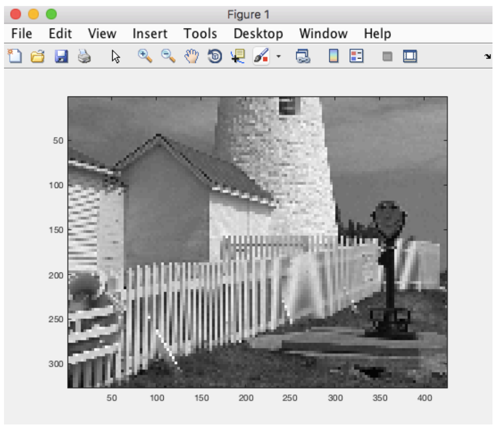
(a) Load the "lighthouse.mat" image data
(b) Down-sample the lighthouse image by a factor of 3 (similar to what you did in section
1.3). Let's call the new array 'xx3'.
(c) Perform a zero-order hold on xx3 to fill in the missing points:

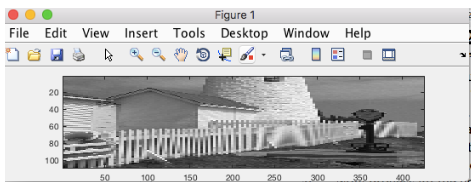
- **i. For an interpolation factor of 3, process all rows of xx3to fill in missing points in that direction. Call the result xholdrows. What are the dimensions of the xholdrows two-dimensional array?**
- **ii. Now process all the columns of xholdrows likewise (interpolation factor 3, now processing in the other direction) to fill in the missing points in each column. Call this result xhold, and show thexhold 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)**

```
>> load lighthouse.mat
>> xx3 = xx(1:3:end, 1:3:end); % new array xx3 down-sample by factor of 3
>> L = length(xx3); % interpolation formula from 2.1.1
>> nn = ceil((0.999:1:3*L)/3);
>> x = size(xx3);
>> xholdrows(1:x(1),:) = xx3(1:x(1),nn); % var x holdrows
>> x = size(xholdrows,2);
>> L = size(holdrows,(1));
Undefined function or variable 'holdrows'.

Did you mean:
>> L = size(xholdrows,(1));
>> xhold(:,1:x) = xholdrows(nn,1:x);
```

(1) xhold
(2) xholdrows
(3) original lighthouse

```
>> show_img(xhold, 1 ,2,colormap(gray(256)));
Image being scaled so that min value is 0 and max value is 255
>> show_img(xholdrows, 1 ,2,colormap(gray(256)));
Image being scaled so that min value is 0 and max value is 255
>> show_img(xx, 1 ,2,colormap(gray(256)));
Image being scaled so that min value is 0 and max value is 255
```





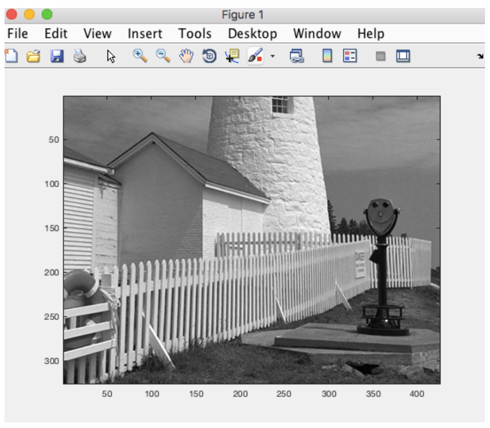


Figure 1, xhold, is the same size yet a very blurred and pixelated version of the original lighthouse. Figure 2, xholdrows, is a shrunken vertically, blurred, and pixelated version of the original lighthouse, or a shrunken vertically version of the xhold figure.

(d) Linear interpolation is sometimes more accurate. 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.

```
>> nn = 1:size(xx3,1);
>> tti = 0: 1/3 : size(xx3,1);
>> xr1linear = interp1(nn,xx3,tti);
>> stem(tti, xr1linear);
>> show_img(xr1linear); % interpolation on rows
>> xr2linear = transpose(xr1linear);
>> show_img(xr2linear);  % interpolation on columns
```

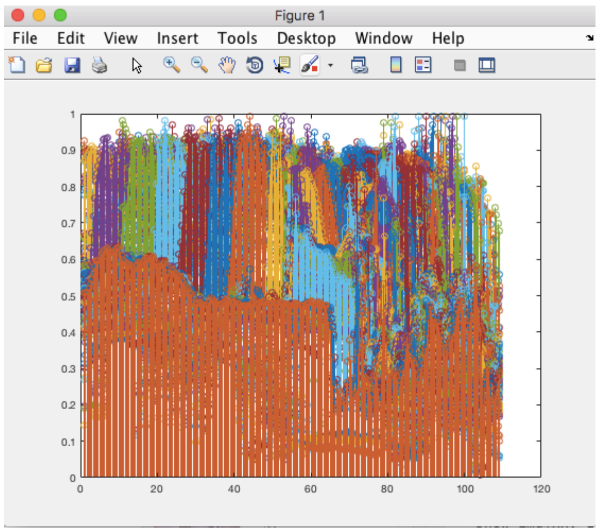


Figure 1 - Stem Plot

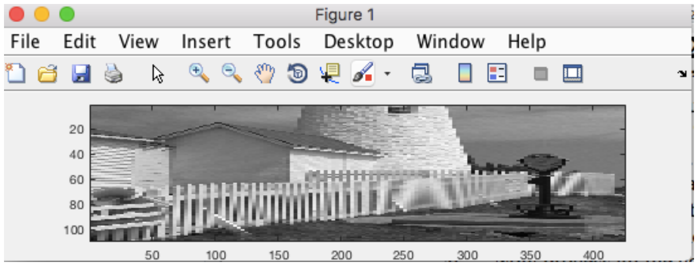


Figure 3 - Interpolation on rows

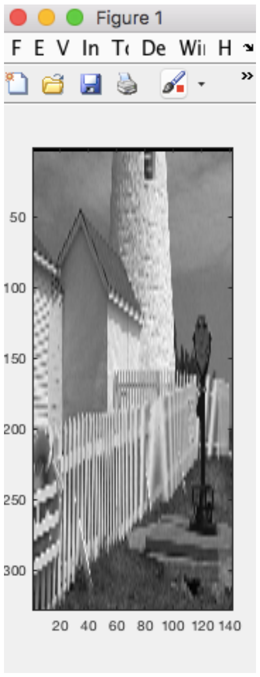


Figure 2 - Interpolation on Columns

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?

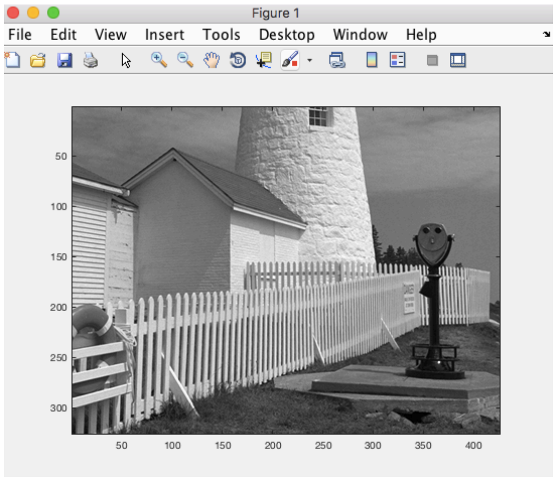


Figure 4 - Original

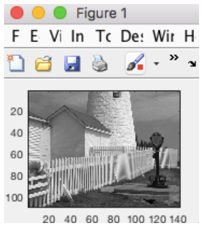


Figure 5 – Down-Sampled

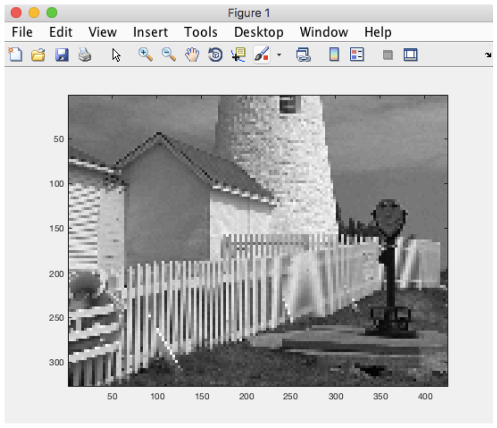


Figure 6 - Zero-Order Hold

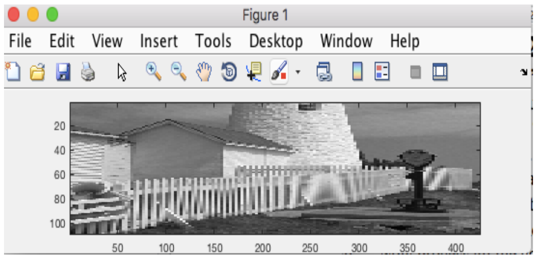


Figure 7 - Linearly Interpolated

(i) Down-Sampled is the Original picture reduced by a factor of 3. Zero-Order Hold is the same size yet a very blurred and pixelated version of the original lighthouse. Linearly Interpolated is a shrunken vertically, blurred, and pixelated version of the original lighthouse, or a shrunken vertically version of the Zero-Order Hold figure.
(ii) No, the linearly interpolated image is still a blurry, pixelated, shrunken version of the original lighthouse, not successfully reproducing it in all parts.

(iii) No, the zero-order hold image is still a blurry, pixelated version of the original lighthouse, not successfully reproducing it in all parts.

(iv) Between zero-order hold and the original, the frequency is not high enough in zero-order hold around parts of the images such as the fence, which is why they appear blurry in zero-order hold but clear in the original. In the zero-order hold image, the fence posts appear straighter because the pixelated form blurs the edges, a lower frequency.

(v) Edges are a lower frequency because of the immediate change that occurs in the features.

(vi) The fence posts are higher frequency because they need more of a rapid oscillation to clearly display the jump from black to white in the fence.

(vii) The background is low-frequency since it is a blurred gray color, fairly uniform.

2.2 Restoration Filter for 1-D Data

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

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

(b) 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.

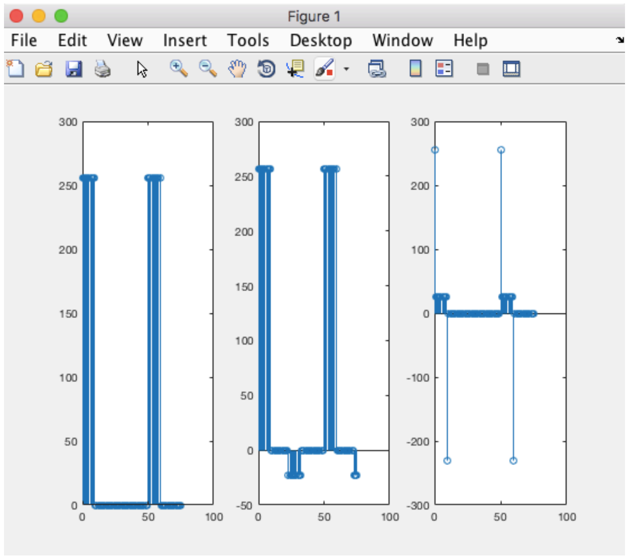
(d)Make a plot of the error (difference) between y[n] and x[n] over the range 0 <= n < 50.

(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]?

(a) & (b)

```
>> xx = 256*(rem(0:100,50)<10);
>> r = 0.9;
>> M = (0:22);
>> signal = [1,-0.9];
>> wsignal = conv(signal,xx);
>> y = r.^(M);
>> ysignal = conv(wsignal, y);
```



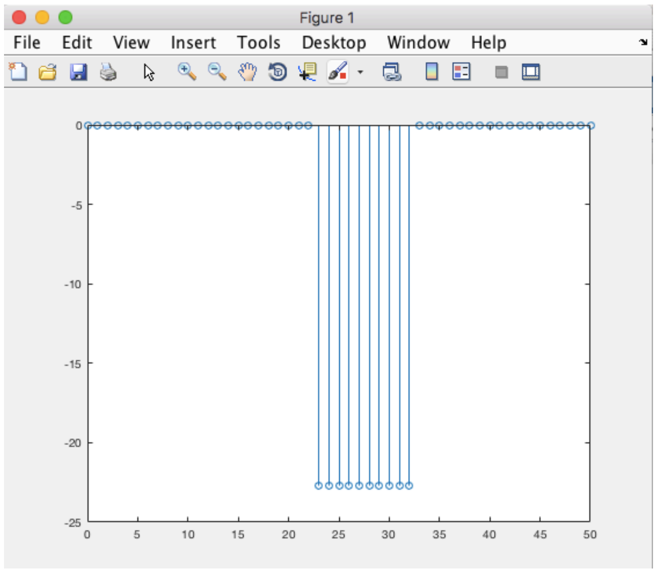
(c)

```
>> subplot(1,3,1);
>> stem(0:75, xx(1:76)) % x signal
>> subplot(1,3,2);
>> stem(0:75, ysignal(1:76)) % y signal
>> subplot(1,3,3);
>> stem(0:75, wsignal(1:76))
```

(d)

```
>> error = ysignal(1:51) - xx(1:51);
>> stem(0:50, error(1:51))
>> stem(0:50, error(1:51))
>> stem(0:50, error(1:51))
>> max(error)

ans =

     0

>> min(error)
```

```
ans =

  -22.6891
```



Max error = 0.0%
Min error = 22.7%

(f)
 Zero max error and a min error of under 25% says the quality is pretty decent of the restoration of xx.

2.3.1 Filtering Images: 2-D Convolution

(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.

(a)

```
>> bk = [1,-1];
>> load echart.mat
>> rowfiltered = conv2(echart,bk);
```

(b)

```
>> yy = conv2(rowfiltered, bk'); % vertical direction
```

(c)

```
>> show_img(echart,1);
Image being scaled so that min value is 0 and max value is 255
>> show_img(rowfiltered,2);
Image being scaled so that min value is 0 and max value is 255
>> show_img(yy, 3);
Image being scaled so that min value is 0 and max value is 255
```

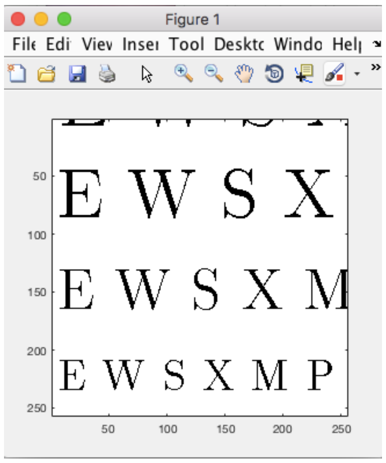


Figure 1 - Original echart

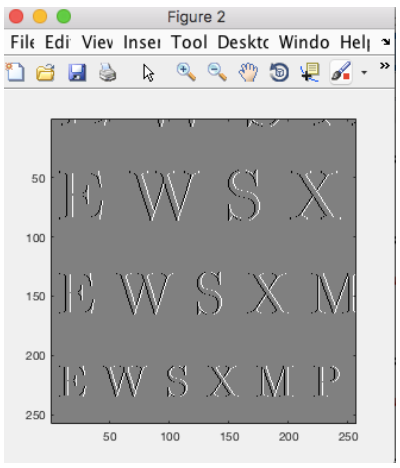


Figure 2 - Intermediate echart (rowfiltered)

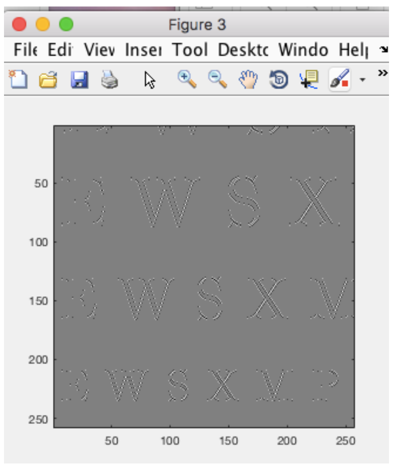


Figure 3 - Final echart (yy)

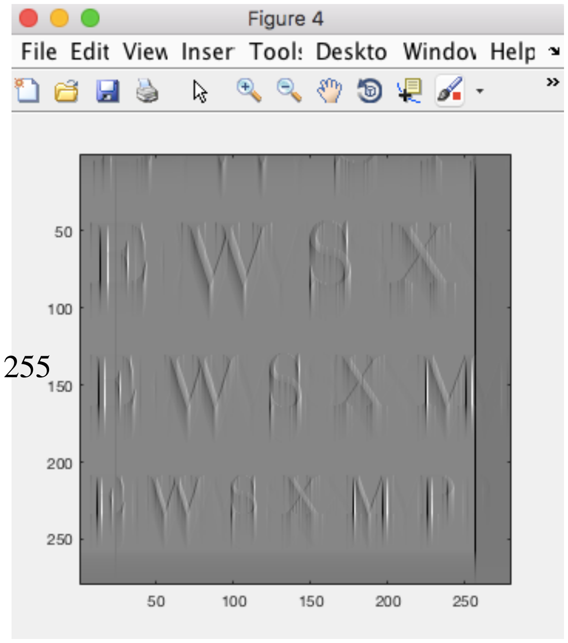
The original image is a black and white version of an eye doctor type eye chart. The second image removes the black and white and shows a grayscale version of the original photo with the look of engraved letters. The third image is lighter, as if lightly engraved, showing less detail in the edges of the letters, making them more difficult to see.

2.3.2 Distorting and Restoring Images

(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.

(a)
```
>> load echart.mat % load chart
>> bk = [1, -0.9]; q = 0.9
>> graychart = conv2(echart,bk); % horizonal transpose
>> echo90 = conv2(graychart, transpose(bk')); % vertical filter
>> show_img(echo90)
Image being scaled so that min value is 0 and max value is 255
ans =
  Axes with properties:
             XLim: [0.5000 258.5000]
             YLim: [0.5000 257.5000]
           XScale: 'linear'
           YScale: 'linear'
    GridLineStyle: '-'
         Position: [0.1301 0.1109 0.7748 0.8133]
            Units: 'normalized'
  Show all properties
```

(b)
```
>> r = 0.9;
>> M = (0:22);
>> ysignal = r.^(M);
>> ghost = conv2(echo90, ysignal);
>> ghost1 = conv2(ghost, ysignal');
>> show_img(ghost1)
Image being scaled so that min value is 0 and max value is 255
ans =
  Axes with properties:
             XLim: [0.5000 280.5000]
             YLim: [0.5000 279.5000]
```



```
          XScale: 'linear'
          YScale: 'linear'
   GridLineStyle: '-'
        Position: [0.1308 0.1108 0.7735 0.8134]
           Units: 'normalized
  Show all properties
```

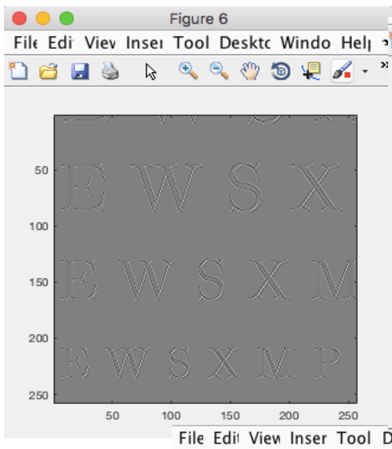
(c)

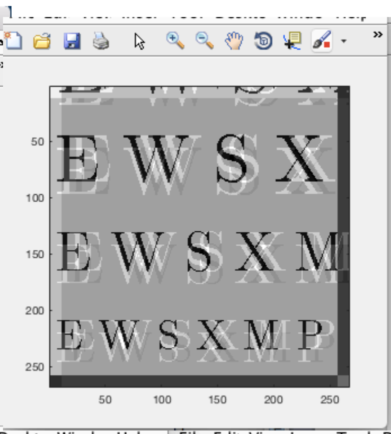
```
>> error = echart(1:256) - echo90(1:256);
>> max(error)
ans =
    0
>> min(error)
ans =
    0
```

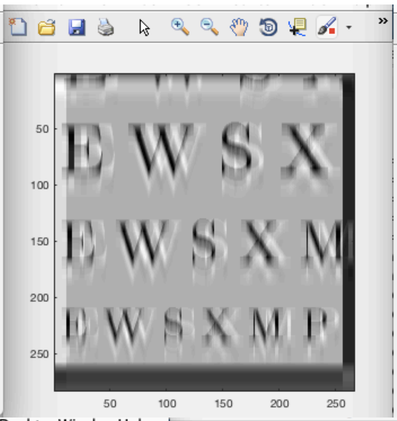
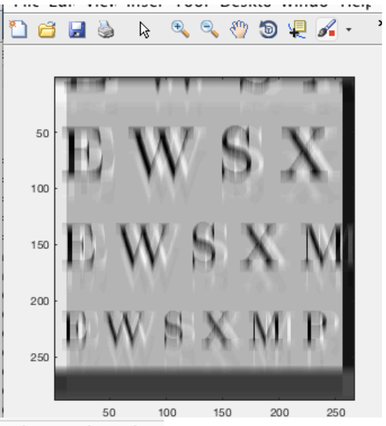
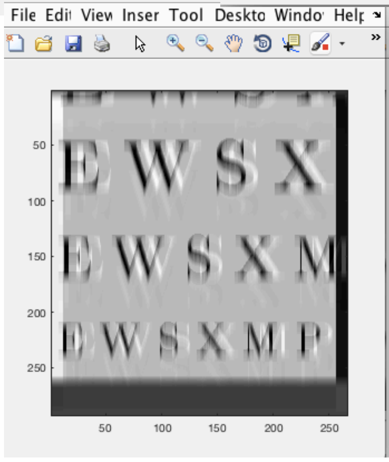
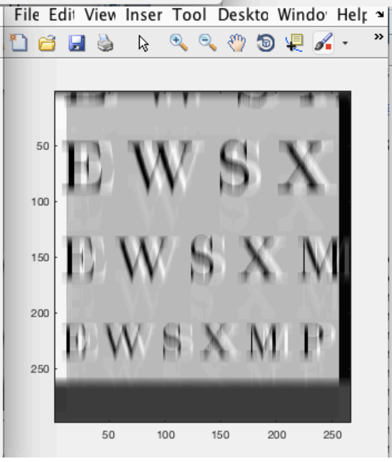
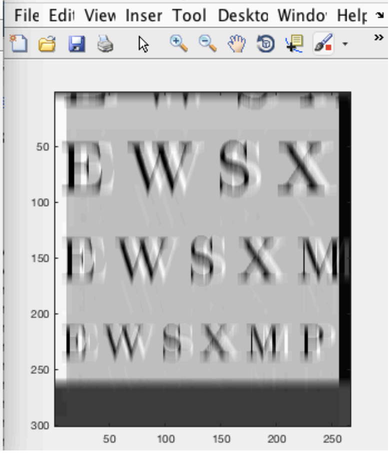
2.3.3 Second Restoration Experiment

(a)

```
>> bk = [1, -0.9];
>> echo90twisted = conv2(echart, bk);
>> echo90 = conv2(echo90twisted, bk');
>> show_img(echo90,6) % show image at 6
Image being scaled so that min value is 0 and max value is 255

ans =

  Axes with properties:

            XLim: [0.5000 257.5000]
            YLim: [0.5000 258.5000]
          XScale: 'linear'
          YScale: 'linear'
   GridLineStyle: '-'
        Position: [0.1305 0.1106 0.7741 0.8139]
           Units: 'normalized'

  Show all properties

>> M = 0.9.^(0:10); % testing from 10 to 33, increments of 5
>> Mconv = conv2(echo90, M);
>> Mconv = conv2(Mconv, M');

>> M1 = 0.9.^(0:15);
>> M1conv = conv2(echo90, M1);
>> M1conv = conv2(Mconv, M1');
>> M2 = 0.9.^(0:20);
>> M2conv = conv2(echo90, M2);
>> M2conv = conv2(Mconv, M2');
>> M3 = 0.9.^(0:25);
>> M3conv = conv2(echo90, M3);
>> M3conv = conv2(Mconv, M3');
>> M4 = 0.9.^(0:30);
>> M4conv = conv2(echo90, M4);
>> M4conv = conv2(Mconv, M4');
>> M5 = 0.9.^(0:33);
>> M5conv = conv2(echo90, M5);
>> M5conv = conv2(Mconv, M5');

>> show_img(Mconv,1);
Image being scaled so that min value is 0 and max value is 255
>> show_img(M1conv,2);
Image being scaled so that min value is 0 and max value is 255
>> show_img(M2conv,3);
Image being scaled so that min value is 0 and max value is 255
>> show_img(M3conv,4);
Image being scaled so that min value is 0 and max value is 255
>> show_img(M4conv,5);
Image being scaled so that min value is 0 and max value is 255
>> show_img(M5conv,6);
Image being scaled so that min value is 0 and max value is 255
>> show_img(echo90,6)
Image being scaled so that min value is 0 and max value is 255
```





(b)

The best figure is the second image, value of N = 10 since it is the clearest, without much blurring.

```
>> error = echart(1:256) - echo90(1:256);
>> max(error)
ans =
    0
>> min(error)
ans =
    0
```

No, my eyes cannot perceive a gray scale change of one level.