Digital Signal Processing: Tutorial 6

Due on May 21, 2017 at 3:00pm

Friso DeBoer

S.Reynolds (262538)

Linear Interpolation

Part A & Part B

The warm up exercise provided code which took a 1D signal and provided a zoom on the signal, using a triangular filter to interpolate values. The code can be seen below:

```
% Clear any sotred variables, clear the workspace, and close any figures
clear; clc; close all;

xss = zeros(1,19);
samp = [1 3 -2 4 2 -1 -3];
xss(1:3:19) = samp;

coeff = [1/3 2/3 1 2/3 1/3];
output = firfilt(xss,coeff);

subplot(2,1,1)
stem(xss)
xlabel('time')
title('Original signal padded with zeros')
axis([0 25 -4 4])

subplot(2,1,2)
stem(output)
xlabel('time')
title('Interpolated signal')
axis([0 25 -4 4])
```

The plots of the original signal and the interpolated signal can be seen in Figure 1. We note that the lower plot has the zero values changed to values that provide a transition to non-zero values. Further, we see a shift in the filtered signal to the right by 2 time units.

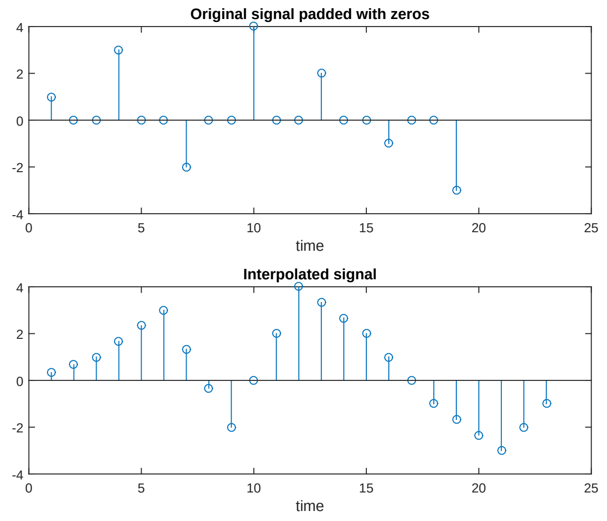

Figure 1: The top plot shows the original signal padded with zero values between actual data points. The lower plot shows the filtered signal which has values interpolated between the actual data values. Further, we note that the signal is delayed by 2 time units.

Part C

The difference equation for the interpolation filter used above, can be written simply by considering the b_k coefficients. The difference equation is as follows:

$$y[n] = {}^1\!/\!_3 \cdot x[n] + {}^1\!/\!_3 \cdot x[n-1] + x[n-2] + {}^2\!/\!_3 \cdot x[n-3] + {}^1\!/\!_3 \cdot x[n-4]$$

Suppose the input signal to the interpolation filter was as follows:

$$x[n] = \delta[n] + 0.5 \cdot \delta[n-3]$$

The output from the filter would be as follows:

$$\begin{aligned} y[n] = \quad & {}^1\!/\!_3 \cdot \delta[n] + {}^1\!/\!_6 \cdot \delta[n-3] \\ & + {}^2\!/\!_3 \cdot \delta[n-1] + {}^2\!/\!_6 \cdot \delta[n-4] \\ & + \delta[n-2] + {}^1\!/\!_2 \cdot \delta[n-5] \\ & + {}^2\!/\!_3 \cdot \delta[n-3] + {}^2\!/\!_6 \cdot \delta[n-6] \\ & + {}^1\!/\!_3 \cdot \delta[n-4] + {}^1\!/\!_6 \cdot \delta[n-7] \end{aligned}$$

Sampling of Images

Part A

Images that are stored on a computer are sampled images, which means that there are a discrete number of pixels stored in an $M \times N$ array. The sampling rate of the image was chose at the time the photograph is taken, however, the sampling rate can be changed by discarding pixel elements in a systematic way. The code shown below is the implementation of one such strategy, whereby every fourth pixel is discarded and replaced by a zero value. The code doesn't change the size of the image, rather it create a zero matrix of the same size as the original image, and then populates every fourth entry across the zero matrix by a one. Finally, to obtain the image at the lower sampling rate, the $M \times N$ matrix with zeros and ones is multiplied by the initial matrix in a piecewise fashion to achieve the result.

```
function  yy = imsample(xx, P)
%IMSAMPLE    Function for sub-sampling an image
%  usage:  yy = imsample(xx,P)
%    xx = input image to be sampled
%     P = sub-sampling period (a small integer like 2, 3, etc.)
%    yy = output image
%
[M,N] = size(xx);
S = zeros(M,N);
S(1:P:M,1:P:N) = ones(length(1:P:M), length(1:P:N));
yy = xx .* S;
```

Part B

The `imsample` function was employed using the baboon image as part of the MATLAB package - the original image and the image with the lowered sampling rate can be seen in Figures 2 and 3, respectively. The MATLAB script used to implement the sampling at the lower rate can be seen in Appendix A.

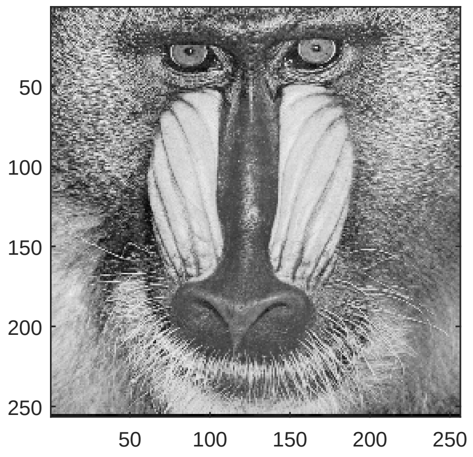

Figure 2: Original image of the Baboon

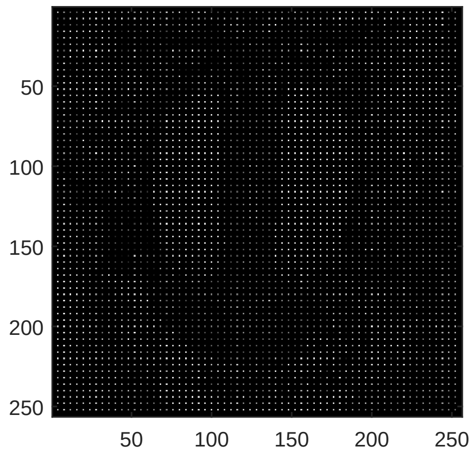

Figure 3: Baboon image at a lower sampling rate

Part C

Down sampling occurs when samples are thrown away and the image is resized. A drawback of employing this method is that aliasing may occur. The script in Appendix A demonstrates this effect.

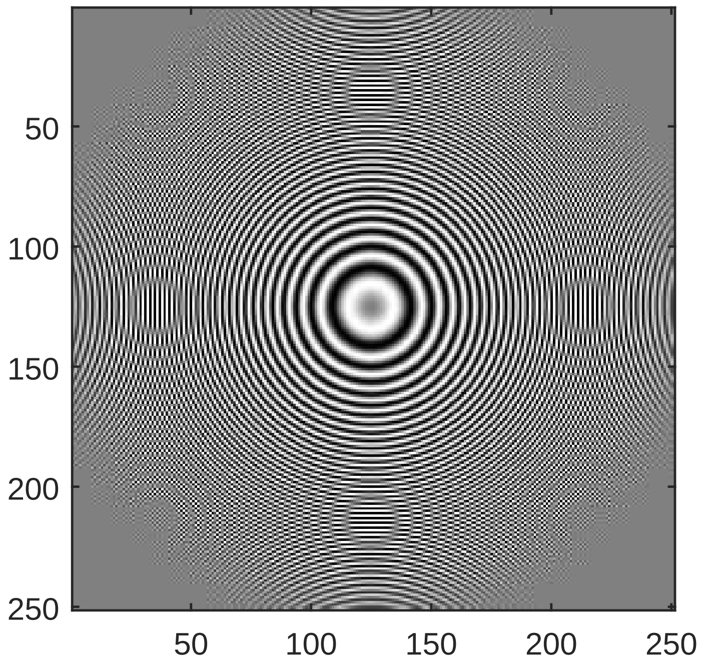

Figure 4: A downsampled image of a zone plate. The aliasing effect shown in the image is the repetition of the central image throughout the remainder of the image.

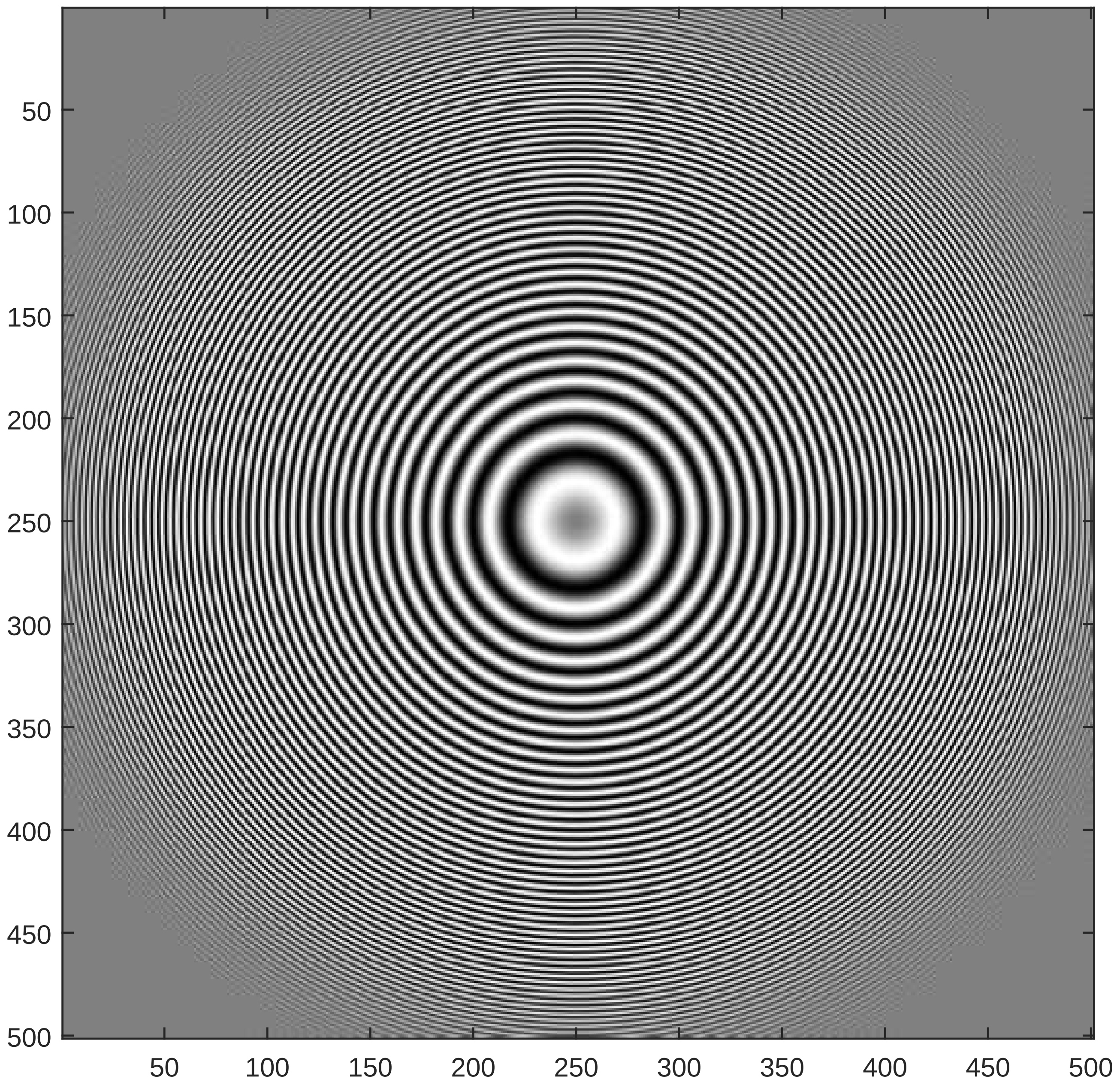

Figure 5: The original zone plate image.

The zone plate is an image in which there are higher frequencies which are integer multiples of lower frequencies present in the image. The aliasing effect can be seen in Figure 4, and the original zone plate image can be seen over the page in Figure 5. Visually, the aliasing effect in static images sees appears as repetition of a central element of the image. In this instance we see the central pattern repeated throughout the image.

The reason for the aliasing effect taking place is due to the sampling rate of the image being lowered below the recommended Nyquist sampling rate, which is 2 times the highest frequency present in the image.

The lenna image was also down sampled in the script shown in Appendix A. The original image can be seen in Figure 6 and the down sampled image can be seen in Figure 7. We note that the down sampled Lenna image has a higher level of distortion in the form of pixelation. Although there may be some aliasing in the down sampled Lenna image, it is not visually apparent. This is due to the reduced number of higher frequencies in the image which are multiples of lower frequencies.

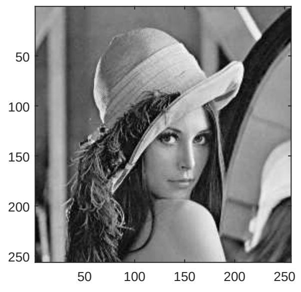

Figure 6: Original Lenna image

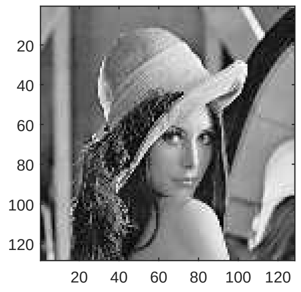

Figure 7: Down sampled Lenna image

Reconstruction of Images

Part A

A tools image, which is part of the MATLAB package, was imported and re-sampled at a lower sampling rate using the `imsample` function - the result can be seen in Figure 8. A zero order hold image reconstruction was undertaken on the rows of the re-sampled image, which resulted in the partial reconstruction of the image, which can be seen in Figure 9. The MATLAB implementation of these operations can be seen in Appendix B.

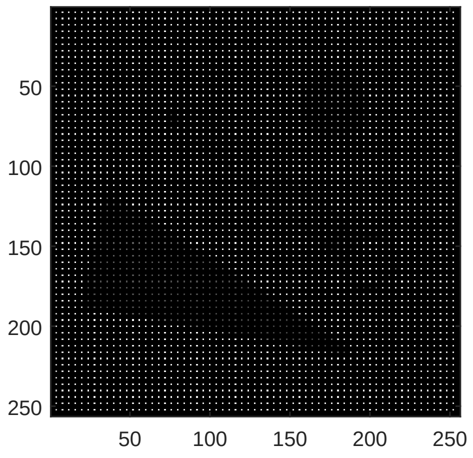

Figure 8: text

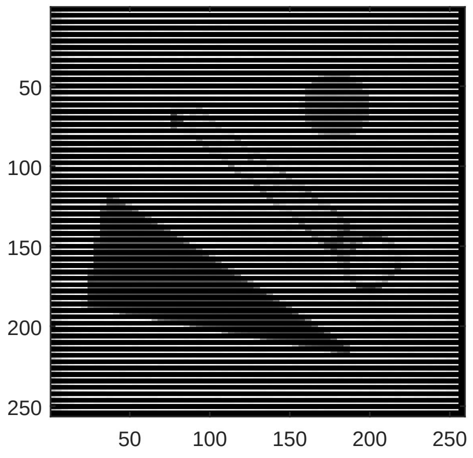

Figure 9: text

Part B

After the row interpolation, the script in Appendix B passes the filter over the columns of the lower rate tools image. The results can be seen in Figure 10. The Figure shows a successful reconstruction of the image using zero order hold interpolation filter, however, there is some loss of quality in comparison with the original image.

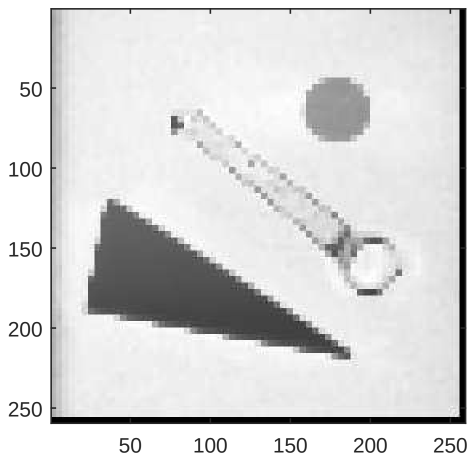

Figure 10: Image reconstructed using a zero order hold FIR interpolation filter

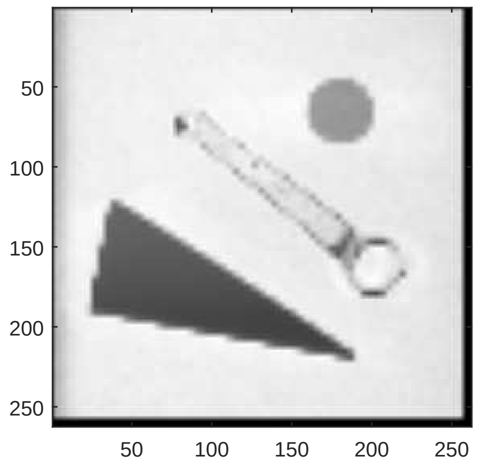

Figure 11: Image reconstructed using a triangular FIR interpolation filter

Part C

Linear interpolation uses a triangular filter on the rows and columns of the image. An FIR filter of order $M = 6$ was used to filter the image. The b_k parameters for the image are as follows:

$$\{b_k\} = \{1/4, \quad 2/4, \quad 3/4, \quad 1, \quad 3/4, \quad 2/4, \quad 1/4\}$$

Part D

Linear interpolation was carried out using the triangular FIR filter, as seen in Appendix B. Figure 11 shows the resulting filtered image. There is a noticeable improvement in the quality of the reconstructed image using the triangular filter. In particular, there is less pixelation around the edges of objects in the image.

Part E

Figure 12 shows the frequency response plot for the triangular FIR filter over the domain $-\pi \leq \hat{\omega} \leq \pi$.

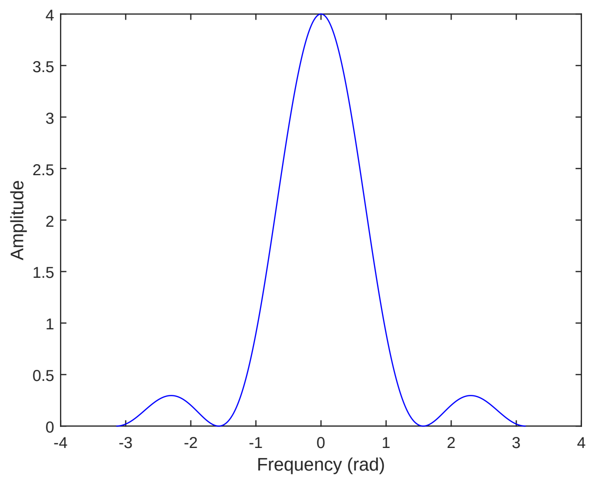

Figure 12: The plot shows the magnitude of the frequency response for the triangular FIR filter of order $M = 6$.

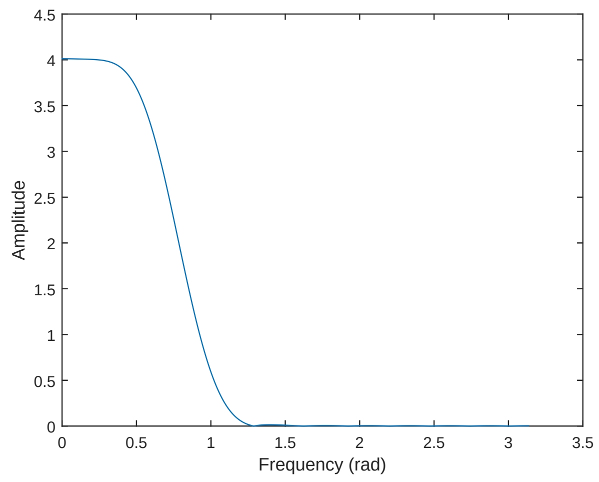

Figure 13: The plot shows the magnitude of the frequency response for the 23-point FIR filter.

Part F

A 23-point FIR lowpass filter was created using the following for the filter coefficients, b_k:

$$b_k = \frac{\sin\left(\pi(k-11)/4\right)}{\pi(k-11)/4} \cdot \omega_k \quad k = 0, 1, 2, ..., 22 \tag{1}$$

We note that ω_k is given by:

$$\omega_k = 0.54 - 0.46 \cdot \cos\left(\frac{2\pi k}{22}\right) \quad k = 0, 1, 2, ..., 22 \tag{2}$$

We note that at the instance $k = 11$, the filter coefficient b_{11} is seemingly undefined. To show that this isn't the case, we consider the following limit of equation (1):

$$\begin{aligned}\lim_{k\to 11} b_k &= \lim_{k\to 11} \frac{\sin\left(\pi(k-11)/4\right)}{\pi(k-11)/4} \cdot \omega_k \\ &= \left(\lim_{k\to 11} \frac{\sin\left(\pi(k-11)/4\right)}{\pi(k-11)/4}\right) \cdot \left(\lim_{k\to 11} \omega_k\right)\end{aligned}$$

Letting $\xi = k - 11$, we get:

$$\begin{aligned}\lim_{k\to 11} b_k &= \left(\lim_{\xi\to 0} \frac{\sin(\xi)}{\xi}\right) \cdot \left(\lim_{k\to 11} \omega_k\right) \\ &= 1 \cdot 1 \\ &= 1\end{aligned}$$

Figure 13 shows the frequency response of the 23-point low pass filter. We note that the filter lets more of the low frequencies through and has a sharper drop off - similar to an ideal LPF. The filter was used to reconstruct the image with the lower sampling rate. This was implemented in the MATLAB code shown in Appendix B, and Figure 14 shows the results of the filtering process.

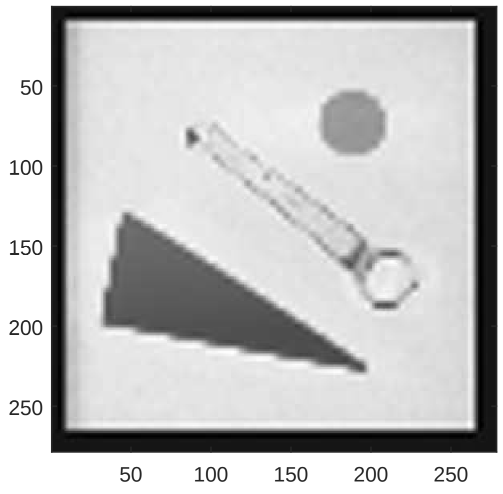

Figure 14: The reconstructed image using the 23-point LPF shown in equations (1) and (2).

Zooming for an Image

Part A

A small 50×50 section of the Lenna image was taken - her eye. By repeating the pixel elements 4 times in each direction, we can zoom in on the image. This was implemented in a script in MATLAB which is listed in Appendix C - the resulting image from this zooming technique can be seen in Figure 15. We note that the image has suffered from some pixelation in te zooming process.

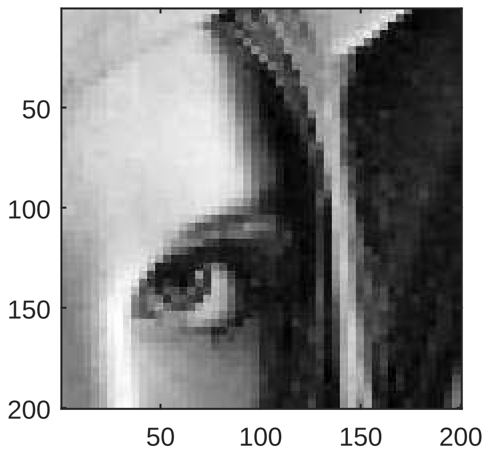

Figure 15: A zoomed in image of the

Part B

Another way to produce a larger image is by inserting zeros into the image around pixel values and then using an interpolation filter to smoother out the values. The zero padding function called `zero_insert` is as follows:

```
function yy = zero_insert(xx)
    [M,N] = size(xx);
    yy = zeros(4*M,4*N);
    yy(4:4:4*M,4:4:4*N) = xx;
end
```

Part C

The image with the zeros inserted into it was filtered using the triangular interpolation FIR, and the low pass filter seen in the previous section. The results of image filtering can be seen in Figures 16 and 17, respectively.

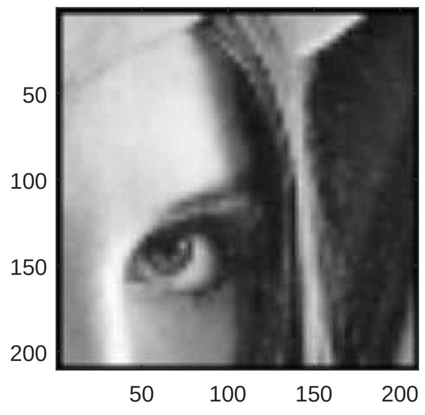

Figure 16: Image filtered using the triangular filter developed in the previous section

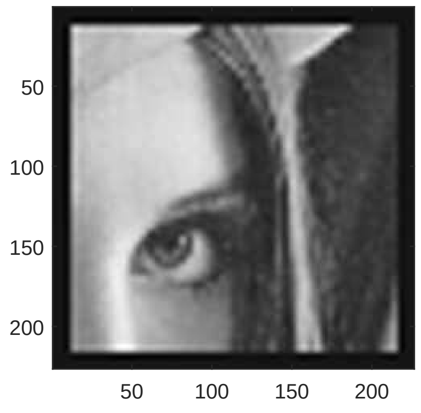

Figure 17: Image filtered using the low pass filter developed in the previous section

The image using the low pass filter or the triangular interpolation have a similar effect on increasing the quality of the zoomed image, when compared to the pixel repetition method used in part A.

Appendix A

```
% Clear any existing variables, clear the workspace, and close figures
clear; clc; close all;

% Load the baboon image and employ downsampling
load baboon.mat
show_img(xx,1)
xs = imsample(xx,4); %-- Sample every fourth point in the image
show_img(xs,2)

% Load the zone plate image and downsample to observe aliasing
x1 = imzoneplate; %-- Create the zoneplate image
show_img(x1,3)
[M,N] = size(x1);
p = 2;
xp = x1(1:p:M,1:p:N); %-- Downsample the image
show_img(xp,4)

% Load the lenna image and downsample to observe the lack of
% aliasing
load lenna.mat
show_img(xx,5)
[M,N] = size(xx);
p = 2;
xp = xx(1:p:M,1:p:N); %-- Downsample the image
show_img(xp,6)
```

Appendix B

```
% Clear any stored variables, clear the workspace, and close all figures
clear; clc; close all;

% Load the images into the workspace
load tools.mat

% Downsample the image
xs = imsample(xx,4);

% Create a zero hold interpolation filter
bs = ones(1,4);

% Filter the downsized data
yhold = conv2(xs,bs);
hold_final = conv2(yhold,bs');

% Show all three images
show_img(xs,1) %-- original image
show_img(yhold,2) %-- horizontal filter
show_img(hold_final,3) %-- horizontal and vertical filter

% Linear interpolation filter
bk = [1/4 2/4 3/4 1 3/4 2/4 1/4];

% Filter the downsize data uing the linear filter
ylin = conv2(xs,bk); %-- filter rows
ylin = conv2(ylin,bk'); %-- filter columns
show_img(ylin,4)

% Determine the frequency response for the filter
[H,W] = freqz(bk);
figure(5)
plot(W,abs(H),'b')
hold on
plot(-W,abs(H),'b')
xlabel('Frequency (rad)')
ylabel('Amplitude')

% Lowpass sinc function filter
k = 0:22;
wk = 0.54 - 0.46*cos((2*pi*k)/(22));
bkk = ( (sin((k-11)*(pi/4)))./((k-11)*(pi/4)) ).*wk;
bkk(12) = 1;

% Filter the downsized image using the low pass mdoified
% sinc filter
xs_filt = conv2(xs,bkk); %-- filter rows
xs_filt = conv2(xs_filt,bkk'); %-- filter columns

show_img(xs_filt,6)

[H,W] = freqz(bkk);
figure(7)
plot(W,abs(H))
xlabel('Frequency (rad)')
ylabel('Amplitude')
```

Appendix C

```
% Clear variables, clear the workspace, and close all figures
clear; clc; close all;

load lenna.mat

% Selection of the area on which to zoom
xx_sub = xx(100:150,150:200);

% Method 1 - Zooming by repeating values
xx_new = zeros(200,200);
a = 1;
b = 1;

for i = 1:50
    a = ( (i-1)*4 + 1 ):( (i-1)*4 + 1 + 3 );
        for j = 1:50
            b = ( (j-1)*4 + 1 ):( (j-1)*4 + 1 + 3 );
            xx_new(a,b) = xx_sub(i,j);
        end
end

show_img(xx_new,1)

% Method 2 - Zooming by filtering with linear interpolator
yy = zero_insert(xx_sub); %-- create matrix to be filtered

bk = [1/4 2/4 3/4 1 3/4 2/4 1/4]; %-- linear interpolation filter
ylin = conv2(yy,bk); %-- filter rows
ylin = conv2(ylin,bk'); %-- filter columns
show_img(ylin,2)

% Method 3 - Zooming by filtering with sinc filter

% Sinc filter creation
k = 0:22;
wk = 0.54 - 0.46*cos((2*pi*k)/(22));
bkk = ( (sin((k-11)*(pi/4)))./((k-11)*(pi/4)) ).*wk;
bkk(12) = 1;

xs_filt = conv2(yy,bkk); %-- filter rows
xs_filt = conv2(xs_filt,bkk'); %-- filter columns
show_img(xs_filt,3)
```