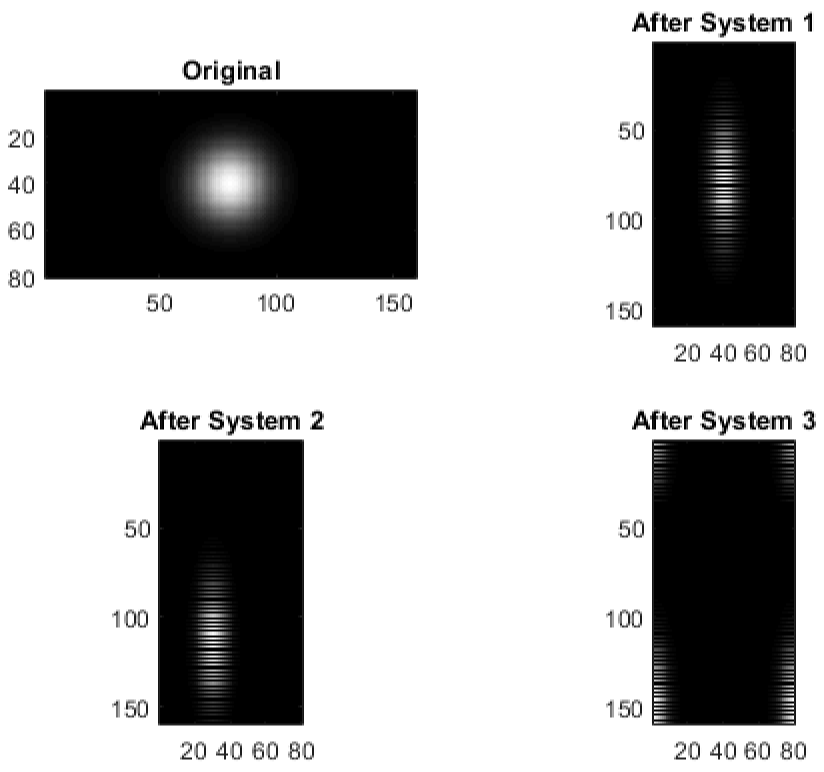
# Koby Miller Lab3

## Table of Contents

## Exercise 3.1

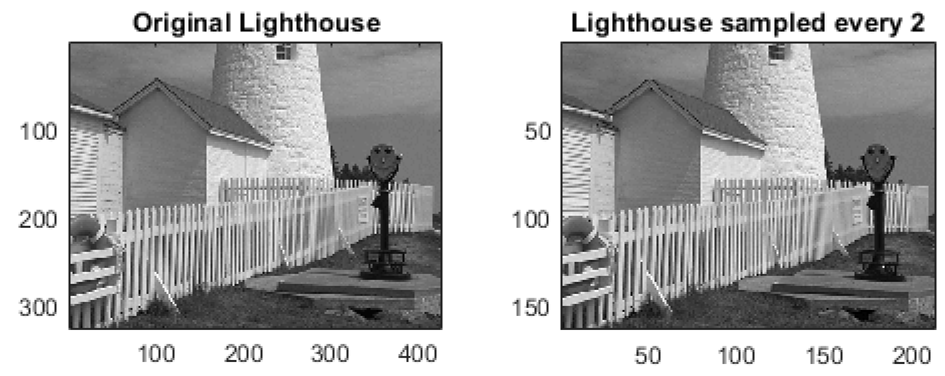
```
% USER DEFINED VARIABLES
w = 15; % Width
x = 1:160; % Horiztonal Axis
y = 1:80; % Vertical Axis
% creates a matrix z filled with color values at each (x,y) position
z = round(127 * exp(-1/w.^2 * ((y.' -40).^2 + (x -80).^2)));
% set the colormap to gray. Now the values are on a scale of black to
% white
colormap(gray);
% call image_system1 to create dimensions needed for rotated images.
% Also creates a translated
[xs, ys, zs] = image_system1(z, 2, 2);
za = image_system2(zs, -10, 35);
zb = image_system3(za, -30, 35);
% PLOT RESULT WITH SUBPLOT
figure(1);
subplot(2, 2, 1); % breaks Figure window into 2x2 matrix. Puts plot in
 position 1 (top left)
imagesc(x, y, z); % display the Z values on an x_y plane
axis image ; % each tick mark increments on the x-,y- and z-axis are
 equal in size.
             % also the plot box fits tightly around the data
title(' Original ')

subplot(2 ,2 ,2) ; % breaks Figure window into 2x2 matrix. Puts plot
 in position 2 (top right)
imagesc(xs, ys, zs); % display the zs values on an x_y plane
axis image; % each tick mark increments on the x-,y- and z-axis are
 equal in size.
            % also the plot box fits tightly around the data
title(' After System 1 ')

subplot(2, 2, 3); % breaks Figure window into 2x2 matrix. Puts plot in
 position 3 (bottom left)
imagesc(xs, ys, za); % display the za values on an x_y plane
axis image; % each tick mark increments on the x-,y- and z-axis are
 equal in size.
```

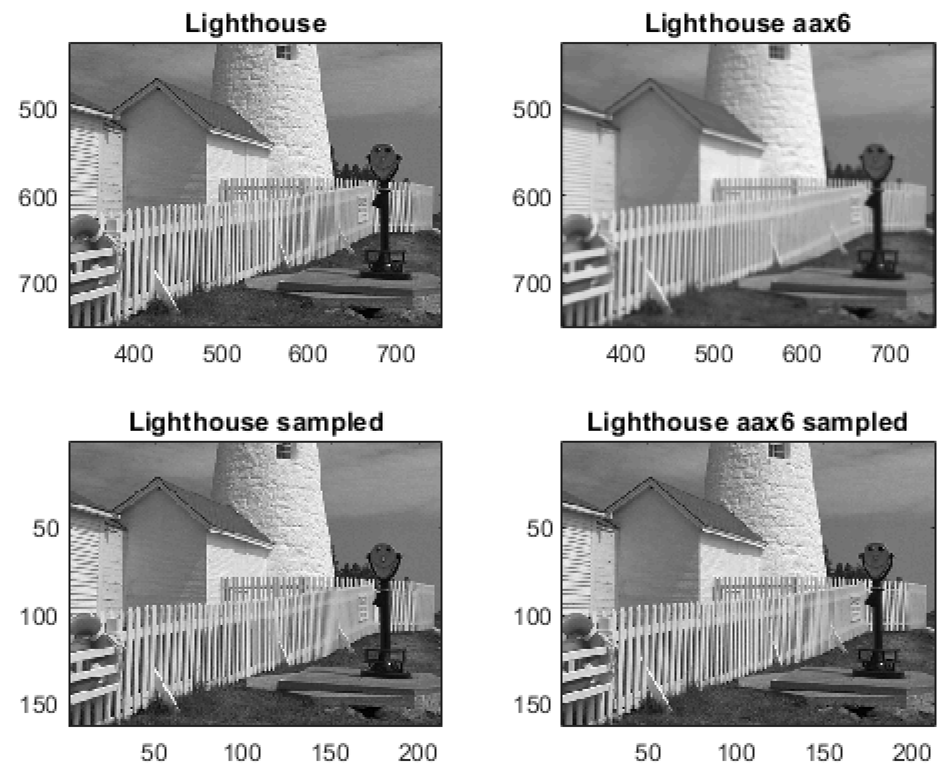
```
            % also the plot box fits tightly around the data
title(' After System 2 ')

subplot(2, 2, 4); % breaks Figure window into 2x2 matrix. Puts plot in
 position 4 (bottom right)
imagesc(xs, ys, zb); % display the zb values on an x_y plane
axis image; % each tick mark increments on the x-,y- and z-axis are
 equal in size.
            % also the plot box fits tightly around the data
title(' After System 3 ')
```

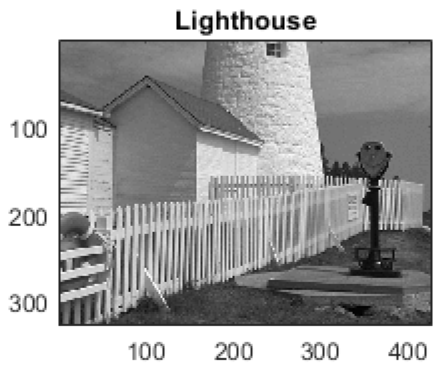


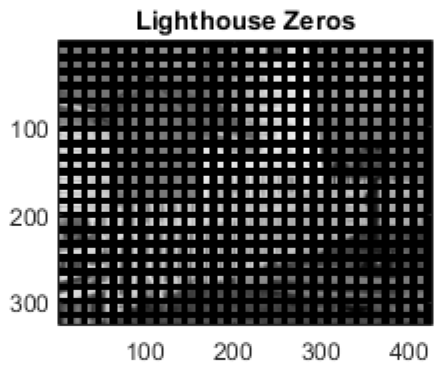
## Exercise 3.2

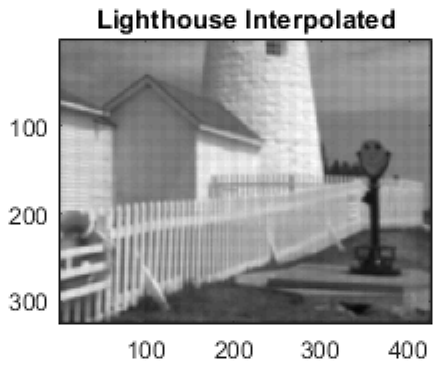
```
%(a)
% Function image_sample is under 'My Functions' section


%(b)
load ('lighthouse.mat');

figure(2)
% Sometimes throughout this lab I use image_sample with a factor of 1
% just to make sure I have the correct dimensions xs, ys.
[xs, ys, zs] = image_sample(lighthouse, 1);
subplot(1, 2, 1);
imagesc(xs, ys, zs);
axis image;
title 'Original Lighthouse';

[xss, yss, lighthouse_sampled] = image_sample(lighthouse, 2);
subplot(1, 2, 2);
imagesc(xss, yss, lighthouse_sampled);
axis image;
title 'Lighthouse sampled every 2';

% When comparing the two, you can see the number of pixels on the
% x-axis and y-axis of the original lighthouse are greater than the
% sampled. But even without seeing those numbers, you can tell that
% the sampled image looks more pixelated. It is not as sharp/clear.
% Overall, the images look very similar, but they are not the same.
% The sample that we took did not have enough values to make the
% pictures indistinguishable. Therefore, aliasing has manifested in
% our images, just like when you don't sample at a high enough
% frequency and get a signal that is different from your original.


%(c)
% Function amage_antialias is under 'My Functions' section

%(d)

lighthouse_aax6 = lighthouse;
for a = 1 : 6
    lighthouse_aax6 = image_antialias(lighthouse_aax6);
end

xs = size(lighthouse,1);
ys = size(lighthouse,2);

figure(3)

subplot(2, 2, 1);
imagesc(xs, ys, lighthouse);
axis image;
title 'Lighthouse';

subplot(2, 2, 2);
imagesc(xs, ys, lighthouse_aax6);
axis image;
title 'Lighthouse aax6';

[xs, ys, lighthouse_sampled] = image_sample(lighthouse, 2);
subplot(2, 2, 3);
imagesc(xs, ys, lighthouse_sampled);
axis image;
title 'Lighthouse sampled';

[xs, ys, lighthouse_aax6_sampled] = image_sample(lighthouse_aax6,2);
subplot(2, 2, 4);
imagesc(xs, ys, zs);
axis image;
title 'Lighthouse aax6 sampled';

% The anti-aliasing filter seems to blur the image a little bit. When
% you zoom in you can see that it sort of blends the colors together
% that are around each other. Now that the pixel groups are blended
% together, when you sample the image and take out some of the pixles,
% the image will still look very similar. This is shown with
% lighthouse_aax6_sampled. This is useful because we can now shrink
% images, yet keep most of the quality as shown with our example
% above. Smaller images are less memory to keep track of, less pixels
% to output.


%(e)
% Function image_insertzeros is under 'My Functions' section

%(f)

figure(4)
[xs, ys, zs] = image_sample(lighthouse, 1);
subplot(2, 2, 1);
imagesc(xs, ys, lighthouse);
axis image;
title 'Lighthouse';


[xz, yz, lighthouse_zeros] =
 image_insertzeros(lighthouse_aax6_sampled,2);
subplot(2, 2, 2);
imagesc(xz, yz, lighthouse_zeros);
axis image;
title 'Lighthouse Zeros';

lighthouse_interpolated = lighthouse_zeros;
for a = 1 : 6
    lighthouse_interpolated =
 image_antialias(lighthouse_interpolated);
end

subplot(2, 2, 3);
imagesc(xz, yz, lighthouse_interpolated);
axis image;
title 'Lighthouse Interpolated';

% The dimensions of lighthouse_interpolated are: 326x426 which is the
% size of the original image. This makes sense because we were
% interpolated a sampled image. So the process - relating to the
% size - shrunk, then expanded the image. The interpolation did the
% same thing as the last part, it blended the colors of each pixel.
% Becuase there were blank pixels this time, it helped fill those
% pixels with color.

colormap(gray);
```


Original Lighthouse
100
200
300
100
200
300
400
Lighthouse sampled every 2
50
100
150
50
100
150
200


Lighthouse
500
600
700
400
500
600
700
Lighthouse aax6
500
600
700
400
500
600
700
Lighthouse sampled
50
100
150
50
100
150
200
Lighthouse aax6 sampled
50
100
150
50
100
150
200







## Given Functions

```
function [xs, ys, zs] = image_system1(z, Dx, Uy)
    % IMAGE_SYSTEM1 Creates a new matrix based on z and given
    % proportions. Can stretch vertically or squish horizontally.
    % The size of the proportion need to make sure that the dimensions
    % of zs are opposite of z (meaning if z has dimensions(r,c), zs
    % needs to be (c,r) Then it fills zs with only some of the values
    % from z.

    % creates matrix zs filled with 0's.
    % dimensions(z-rows*Uy, z-col/Dx)
    zs = zeros(ceil(Uy * size(z, 1)), ceil(size(z, 2) / Dx));
    % creates 2 new arrays. One that is double the size of z's rows.
    % And one that is half of the size of z's columns.
    % Both counting from 1 to their size
    ys = 1: ceil(Uy * size(z ,1));
    xs = 1: ceil(size(z, 2) / Dx);
    % sets the zs rows that are multiples of Uy equal to the z columns
    % that are multiples of Dx. The rest of zs is 0
    zs(1: Uy : end, 1: end) = z(1: end, 1: Dx : end);
end
```

```
File 'image_stystem2.m' not found.
```

```
function [zb] = image_system3(za, Sx, Sy)
    % IMAGE_SYSTEM3 Takes a matrix and shift values. Shifts the given
    % matrix by the values with a revolving door effect for the
    % left/right direction and up/down direction
    % creates 2 new arrays with a size of the row/col length -1
    x = 0:1: size(za, 2) -1;
    y = 0:1: size(za, 1) -1;
    % takes the modulus of each index value minus the shift verses the
 size
    xs = mod(x - Sx, size(za, 2));
    ys = mod(y - Sy, size(za, 1));
    % creates zb with ys values in its rows and xs in its columns
    zb = za(ys +1, xs +1);
end
```

## My Functions

```
function [xs, ys, zs] = image_sample(z, D)
    % IMAGE_SAMPLE samples grayscale or color image z every D pixels
    % in both the horizontal and vertical direction

    % creates matrix zs filled with 0's. dimensions(z-rows/D, z-col/D)
    zs = zeros(ceil(size(z, 1) / D), ceil(size(z, 2) / D),
 ceil(size(z, 3)));
    % creates 2 new arrays. One that is half of the size of z's rows.
    % And one that is half of the size of z's columns.
    % Both counting from 1 to their size
    ys = 1: ceil(size(z ,1) / D);
    xs = 1: ceil(size(z, 2) / D);
    % sets the zs rows and columns that are multiples of D equal to
    % the z are multiples of Dx. The rest of zs is 0
    zs(1:end, 1:end, 1:end) = z(1:D:end, 1:D:end, 1:end);
end
```

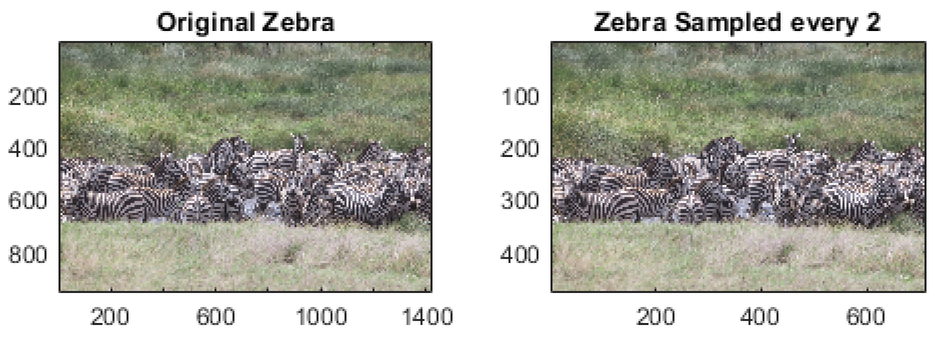
```
function [za] = image_antialias(z)
    % IMAGE_antialias filters a grayscale matrix by blending each
    % pixles with the pixels around it. Outputs the new blended matrix

    za = zeros(size(z, 1), size(z, 2), size(z, 3));
    for k = 1: size(z,3)
        for x = 1: size(z, 1)
            for y = 1: size (z, 2)
            % if your current row/colum - is within Sx/Sy distance
 from
            % each end of the matrix
                if x - 1 > 0 && y - 1 > 0 && x + 1 < size(z,1) && y +
 1 < size(z,2)
```

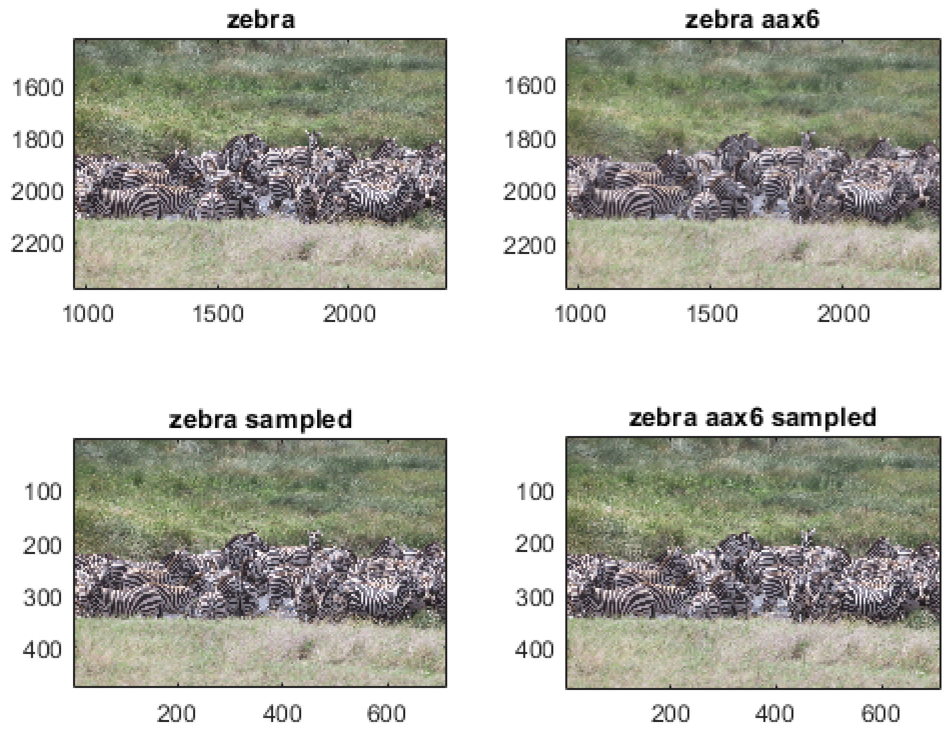
```
                % sets za value to half of the z value that is (Sx,
 Sy)
                % plus, 1/8th of each pixle on the left/right/top/
bottom
                za (x,y,k) = 1/2*z(x,y,k) + 1/8*(z(x-1,y,k) + z(x
+1,y,k) + z(x,y-1,k) + z(x,y+1,k));
                end
            end
        end
    end
end



function [xz, yz, zz] = image_insertzeros(zaas, U)
    % IMAGE_insertzeros doubles the image size and puts a value of 0
    % in between every value in the matrix

    % creates matrix zs filled with 0's.
    % dimensions(zass-rows*U, zass-col*U, zass-k)
    zz = zeros(ceil(size(zaas, 1) * U), ceil(size(zaas, 2) * U),
 ceil(size(zaas, 3)));
    % creates 2 new arrays. One that is double the size of zass' rows.
    % And one that is double the size of zass' columns
    % Both counting from 1 to their size
    yz = 1: ceil(size(zaas, 1) * U);
    xz = 1: ceil(size(zaas, 2) * U);
    % sets the zz pixels in rows and colums that are multiples of U
    % equal to the pixles in zass
    zz(1:U:end, 1:U:end, 1:end) = zaas(1:end, 1:end, 1:end);
end
```

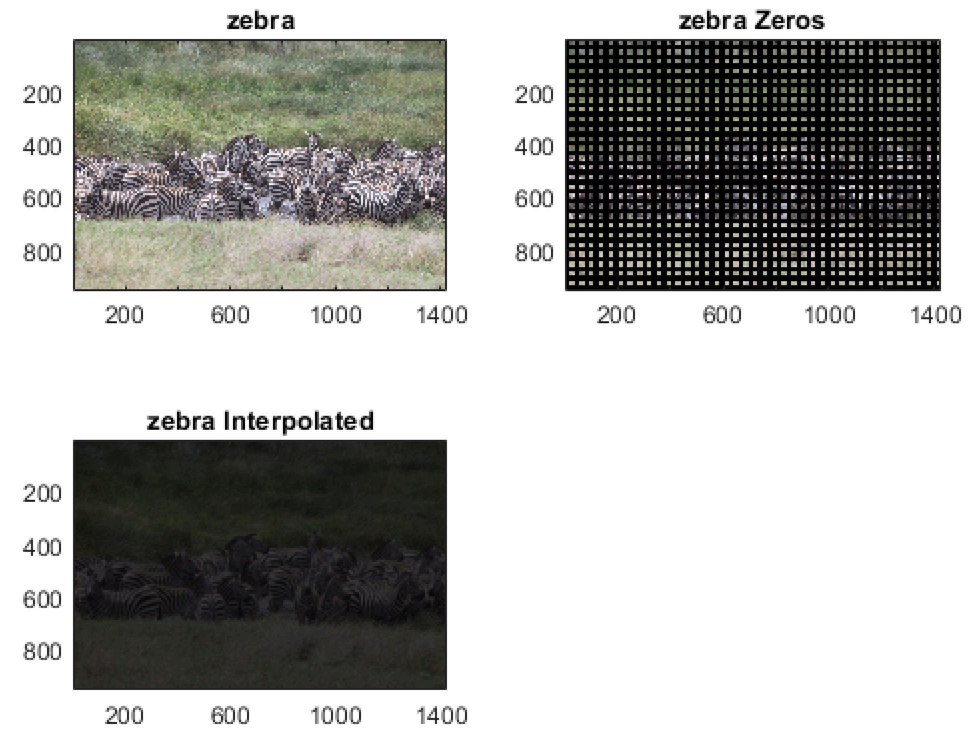
## Extra Credit

```
%(b)
load('zebra.mat');

figure(5)

[xs, ys, zs] = image_sample(zebra, 1);
subplot(1, 2, 1);
imagesc(xs, ys, zs);
axis image;
title 'Original Zebra';

[xss, yss, zebra_sampled] = image_sample(zebra, 2);
subplot(1, 2, 2);
imagesc(xss, yss, zebra_sampled);
axis image;
title 'Zebra Sampled every 2';

% This time, it is definitely harder to see that the sampled image is
% more pixilated. This is due to the original image being much bigger.


%(d)

zebra_aax6 = zebra;
for a = 1 : 6
    zebra_aax6 = image_antialias(zebra_aax6);
end

xs = size(zebra,1);
ys = size(zebra,2);

figure(6)

subplot(2, 2, 1);
imagesc(xs, ys, zebra);
axis image;
title 'zebra';

subplot(2, 2, 2);
imagesc(xs, ys, zebra_aax6);
axis image;
title 'zebra aax6';

[xs, ys, zebra_sampled] = image_sample(zebra, 2);
subplot(2, 2, 3);
imagesc(xs, ys, zebra_sampled);
axis image;
title 'zebra sampled';

[xs, ys, zebra_aax6_sampled] = image_sample(zebra_aax6,2);
subplot(2, 2, 4);
imagesc(xs, ys, zs);
axis image;
title 'zebra aax6 sampled';

%(e)

figure(7)
[xs, ys, zs] = image_sample(zebra, 1);
subplot(2, 2, 1);
imagesc(xs, ys, zebra);
axis image;
title 'zebra';


[xz, yz, zebra_zeros] = image_insertzeros(zebra_aax6_sampled,2);
subplot(2, 2, 2);
imagesc(xz, yz, zebra_zeros);
axis image;
title 'zebra Zeros';

zebra_interpolated = zebra_zeros;
```

```
for a = 1 : 20
    zebra_interpolated = image_antialias(zebra_interpolated);
end

subplot(2, 2, 3);
imagesc(xz, yz, zebra_interpolated);
axis image;
title 'zebra Interpolated';
```




zebra
zebra aax6
zebra sampled
zebra aax6 sampled
1600
1800
2000
2200
1000
1500
2000
100
200
300
400
200
400
600


zebra
zebra Zeros
zebra Interpolated
200
400
600
800
200
600
1000
1400

*Published with MATLAB® R2020a*