SC4061 Computer Vision
Lab 1 Report

Name: Jden Goh
Matri. No: U2222711B

1. Contrast Stretching

1.1 Input Image and Convert to Greyscale

```
% 2.1 (a) - Input image

Pc = imread("mrt-train.jpg");
whos Pc;

% Convert image from RGB to Greyscale
P = rgb2gray(Pc);
```

1.2 View Image using `imshow`

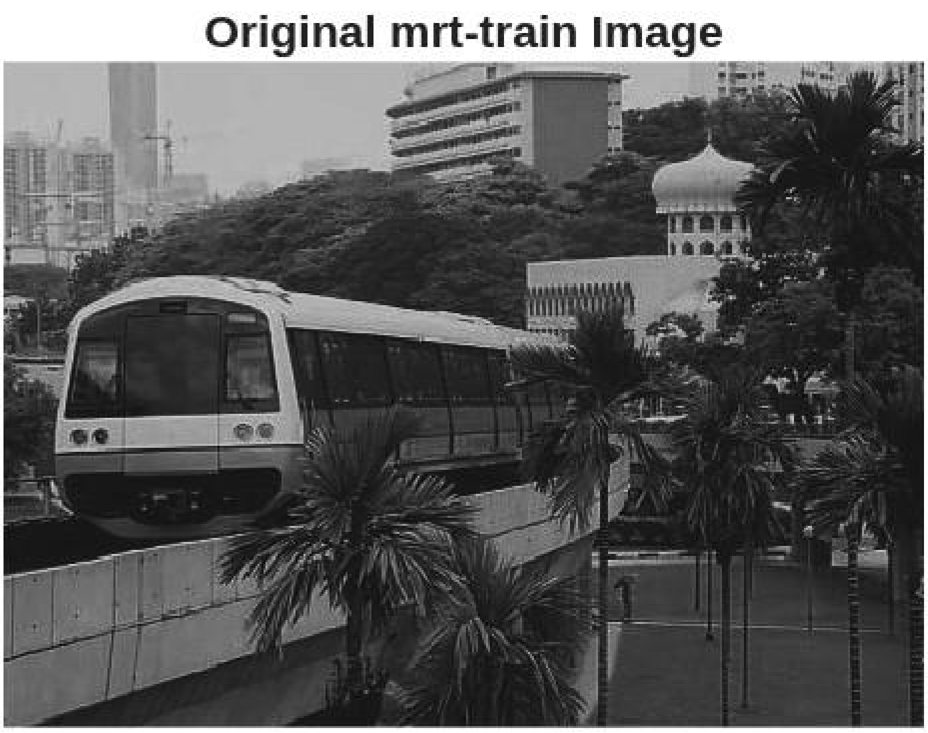
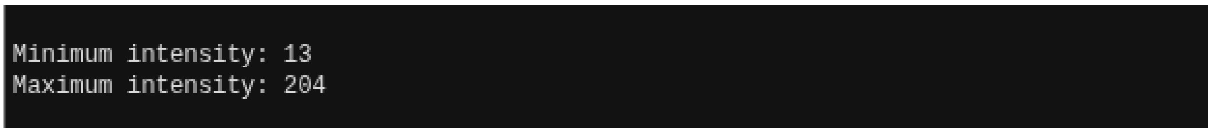


Fig 1. Original mrt-train blackscale image

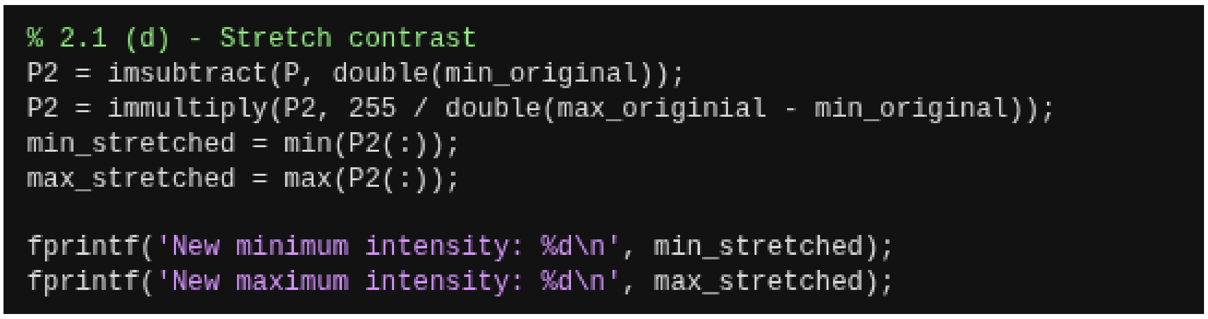
1.3 Find Max and Min Intensities

```
% 2.1 (c) - Check original max and min attribute
min_original = min(P(:))
max_originial = max(P(:))

fprintf('Minimum intensity: %d\n', min_original);
fprintf('Maximum intensity: %d\n', max_originial);
```

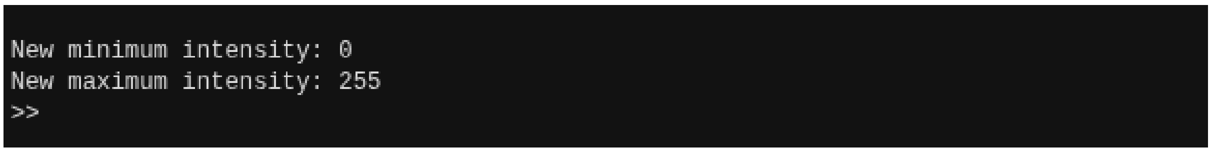


```
Minimum intensity: 13
Maximum intensity: 204
```

1.4 Contrast Stretching



```
% 2.1 (d) - Stretch contrast
P2 = imsubtract(P, double(min_original));
P2 = immultiply(P2, 255 / double(max_originial - min_original));
min_stretched = min(P2(:));
max_stretched = max(P2(:));

fprintf('New minimum intensity: %d\n', min_stretched);
fprintf('New maximum intensity: %d\n', max_stretched);
```



```
New minimum intensity: 0
New maximum intensity: 255
>>
```

1.5 Redisplay Stretched Image

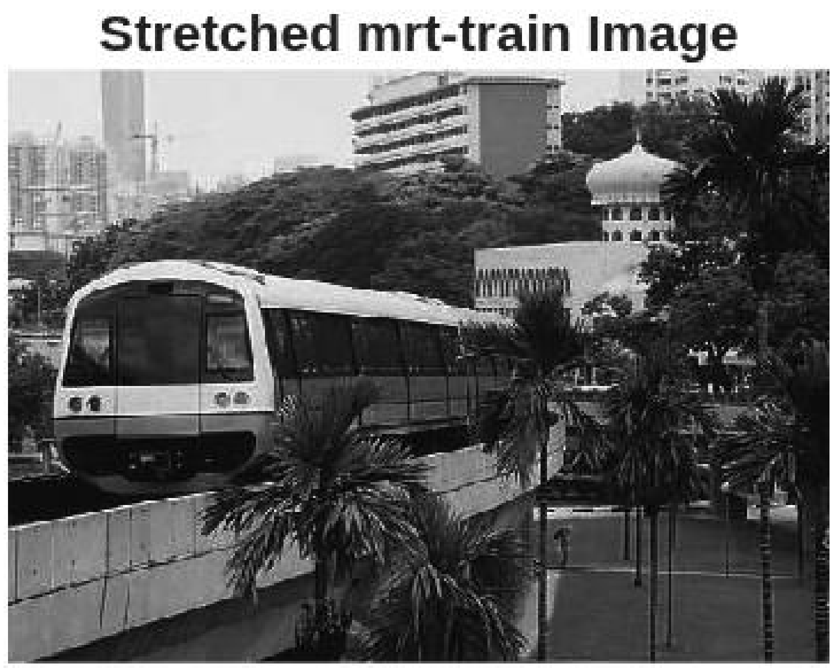


Fig 2. mrt-train image after Contrast Stretching

Analysis:
Contrast stretching allows us to utilize the full range of pixel intensity, from 0 to 255. We can set our darkest pixel at 0 and brightest pixel at 255, and then all values are scaled to fit the full range. This will help to boost the overall contrast of the image.

2. Histogram Equalization

2.1 Display Image Intensity Histogram (10 bins and 256 bins)

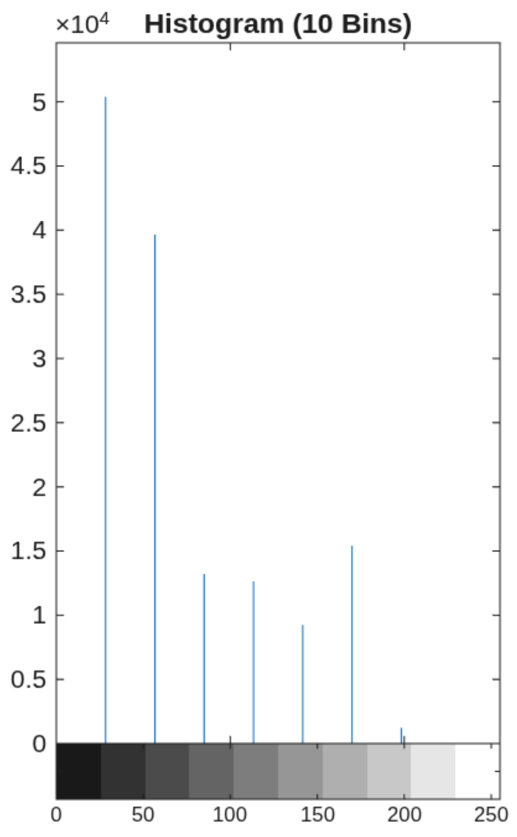


Fig 3. Image Intensity Histogram (10 bins)

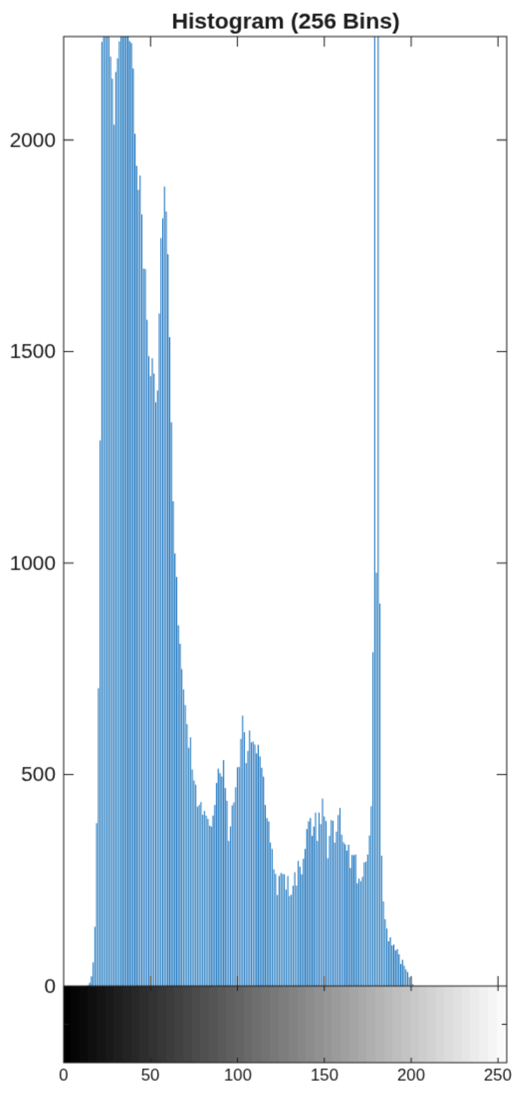


Fig 4. Image Intensity Histogram (256 bins)

Differences:

- For the Histogram with 10 bins:
    - 10 bins divide the intensity range of 0 to 255 into 10 equal groups
    - It shows us an extremely general view of the distribution, good for viewing the overall trends
    - However, we lose some key details, since this distribution generalizes the data very heavily

- On the other hand, for the histogram with 256 bins:
    - Each bin matches an intensity value
    - It provides the highest amount of details of the intensity distribution
    - It shows fine-grain patterns and peaks in our data
    - For instance, at intensity level of approximately 175, we see that there is a large number of pixels, but this information is not represented in the 10 bin histogram

2.2 Histogram Equalization

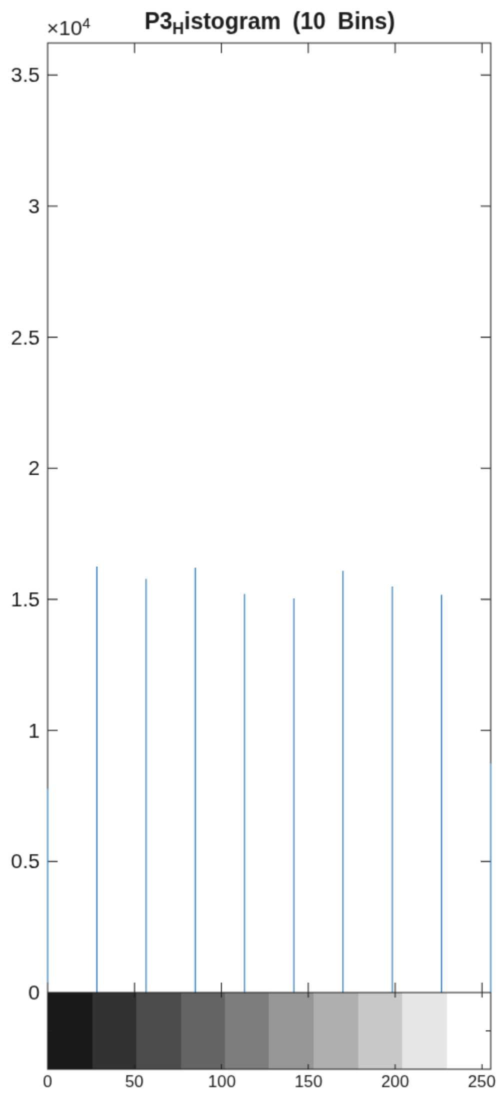
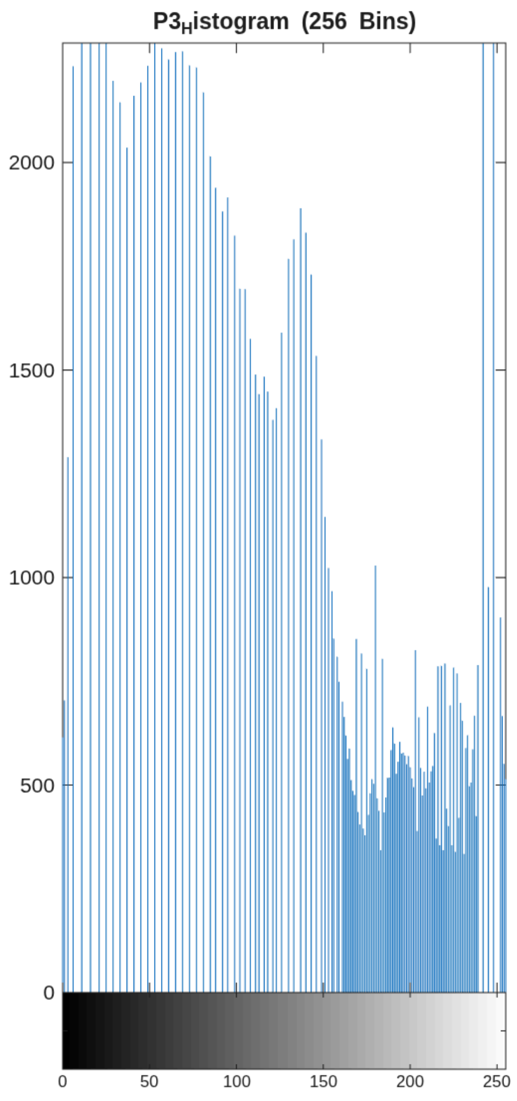
	

Fig 5. Equalized Image Intensity Histogram (10 bins)	Fig 6. Equalized Image Intensity Histogram (256 bins)

Are the histograms equalized?
Yes, both histograms are much more equalized now, compared to the original histogram. This can be observed most noticeably when we look at the 10 bins histogram.

The intensity of the pixels have been redistributed to create a more uniform distribution across the entire intensity range, hence we are able to see the more balanced bin heights in our histograms.

What are the similarities and differences between the latter two histograms?

- Similarities
    - Both histograms are more evenly distributed across the entire range of image intensity
- Differences
    - The 10 bin histogram is observed to be much more equalized, without any distinct peak of dips in overall distributions
    - On the other hand, for the 256 bin histogram, we can still see some distinct peaks and dips, for instance at the intensity level about 240

2.3 Rerun the histogram equalization on P3

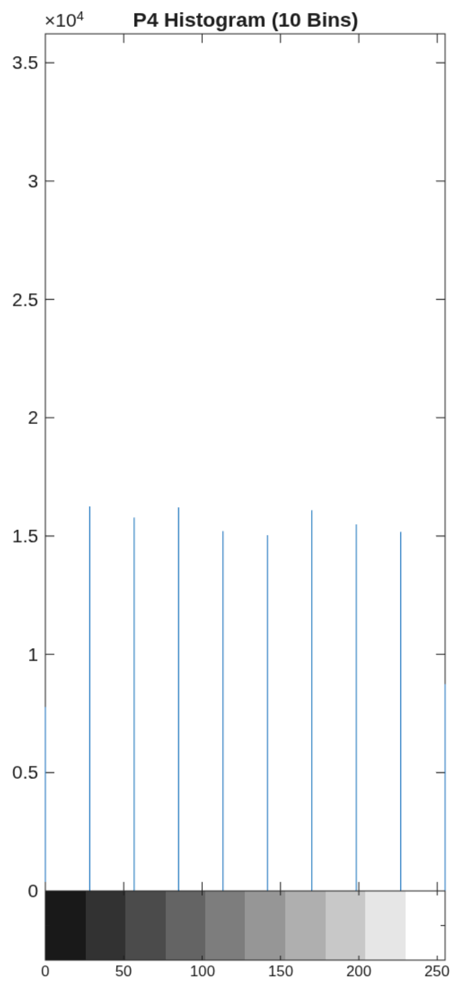


Fig 7. 2-Rounds Equalized Image Intensity Histogram (10 bins)

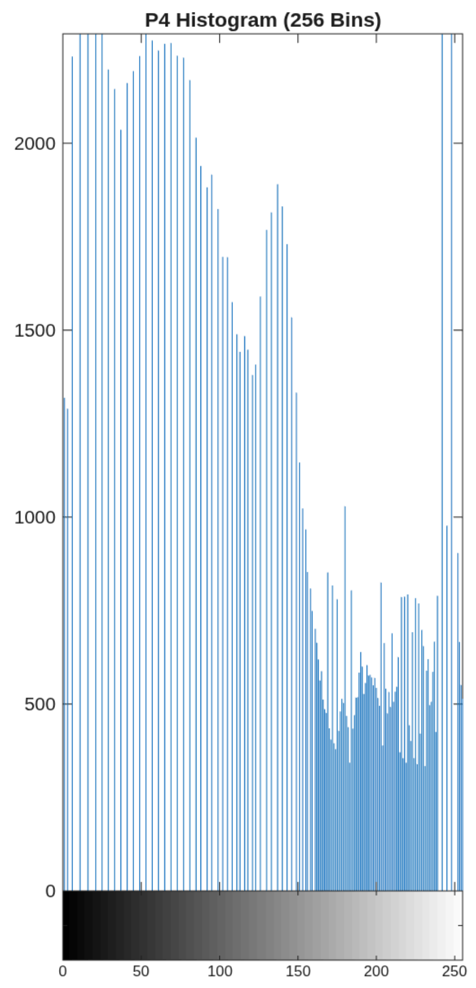


Fig 8. 2-Rounds Equalized Image Intensity Histogram (256 bins)

Does the histogram become more uniform? Give suggestions as to why this occurs.

No, the histogram remains extremely similar to that of P3, with minimal to no improvement to the uniformity.

P3 was originally equalized to achieve the most uniform distributions possible (through histogram equalization in this case). This means that the pixel intensities are already spread uniformly across the range intensity levels. Hence, performing the same histogram equalisation again will not have any significant improvement to the uniformity.

3. Linear Spatial Filtering

3.1 (i) Y and X-dimensions are 5 and σ = 1.0

```
% 2.3 (a[i]) - Y and X-dimensions are 5 and σ = 1.0
h1 = fspecial('gaussian', 5, 1);
```

3.1 (ii) Y and X-dimensions are 5 and σ = 2.0

```
% 2.3 (a[ii]) - Y and X-dimensions are 5 and σ = 2.0
h2 = fspecial('gaussian', 5, 2);
```

Normalize the filters:

```
% Normalize filters
h1 = h1 / sum(h1, 'all');
h2 = h2 / sum(h2, 'all');

% Check if h1 and h2 are normalized
fprintf('Sum of h1 elements: %f\n', sum(h1, 'all'));
fprintf('Sum of h2 elements: %f\n', sum(h2, 'all'));
```

```
Sum of h1 elements: 1.000000
Sum of h2 elements: 1.000000
```

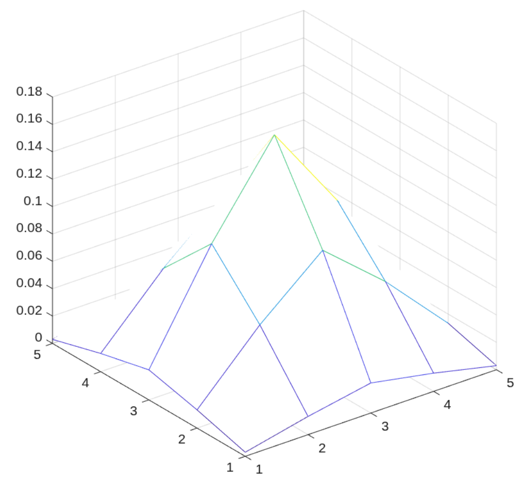
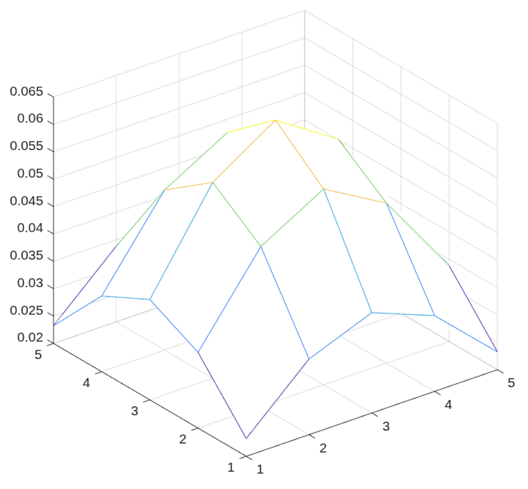
	

Fig 9. 3D Graph - h1 filter	Fig 10. 3D Graph - h2 filter

3.2 View “lib-gn.jpg”

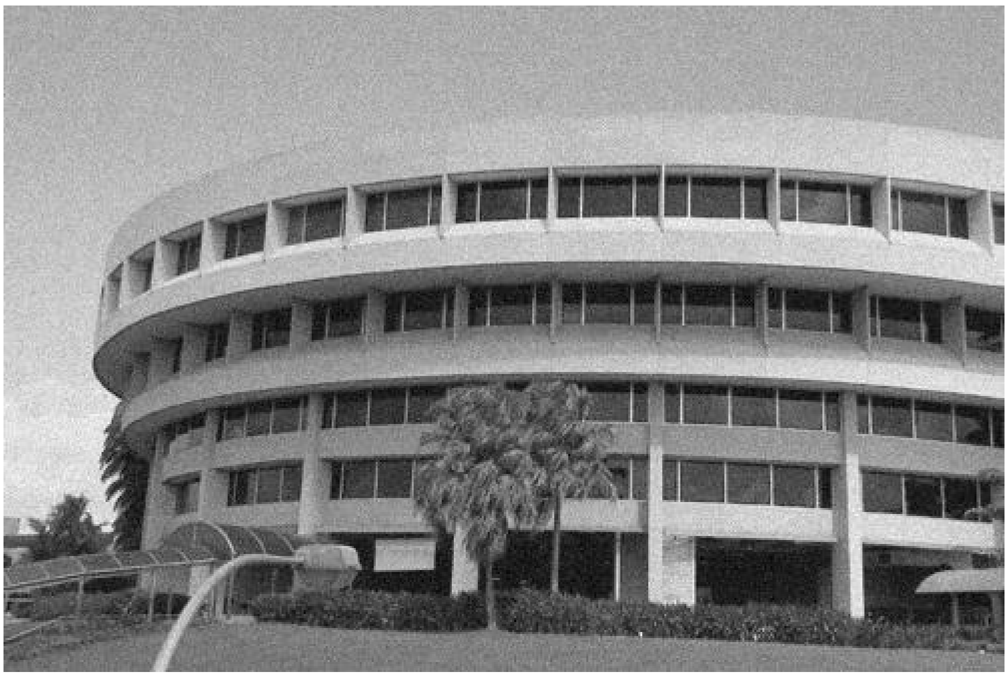


Fig 11. lib-gn.jpg

3.3 Filter the image using the linear filters using **conv2**

```
% 2.3 (c) - Linear filtering
Pgn1 = uint8(conv2(Pgn,h1));
figure
imshow(Pgn1);
saveas(gcf, 'results/pgn1.png');


Pgn2 = uint8(conv2(Pgn,h2));
figure
imshow(Pgn2);
saveas(gcf, 'results/pgn2.png');
```

Applying h1 Filter:

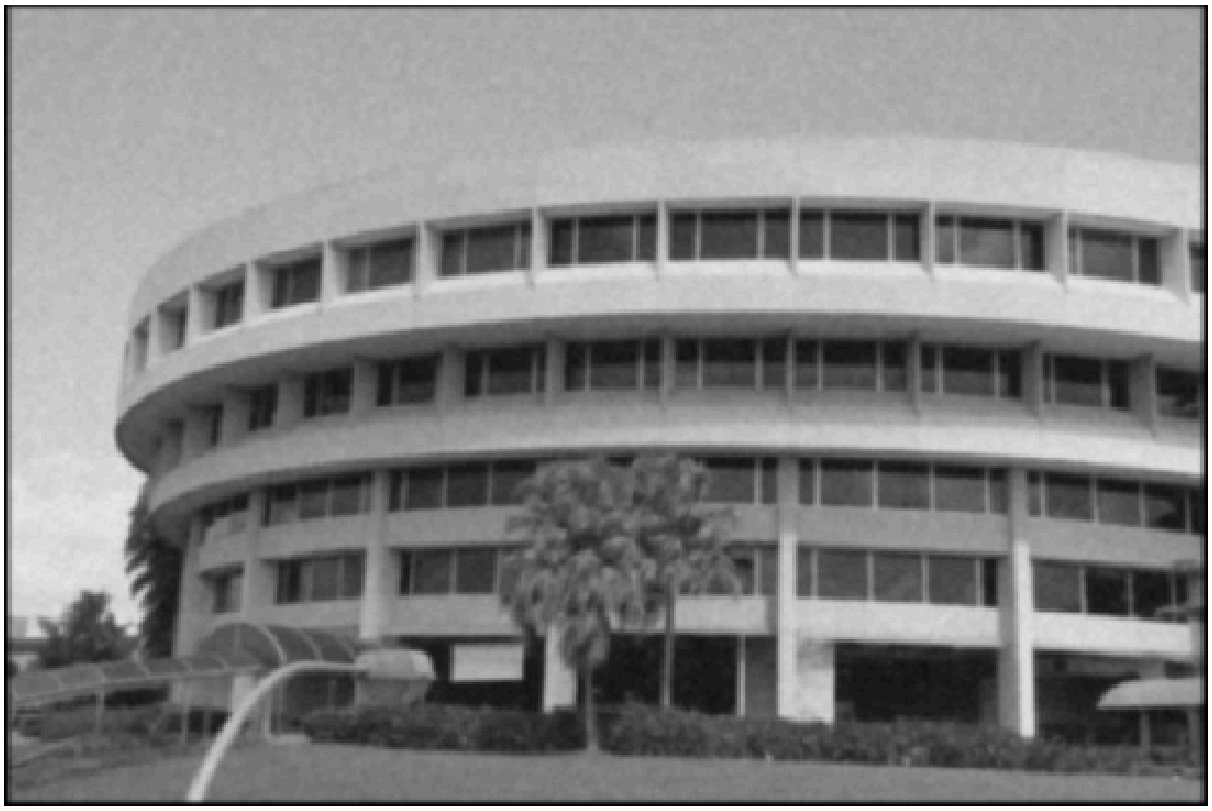


Fig 12. Lib-gn with h1 filter

Applying h2 Filter:

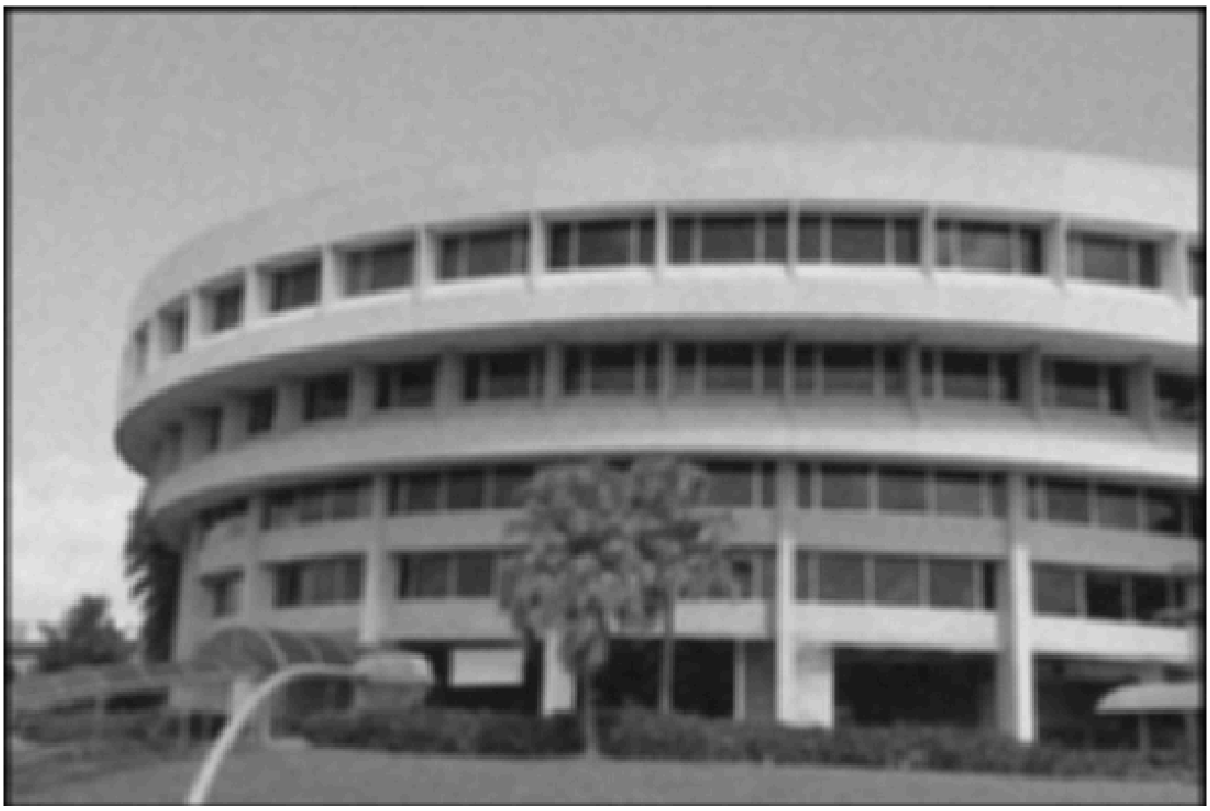


Fig 13. Lib-gn with h2 filter

How effective are the filters in removing noise?
Both filters are quite effective at reducing noise, as we can see from both images having reduced graininess. Gaussian filters work by taking the weighted average of each pixel with its neighbours, hence reducing smoothening out the random noise.

What are the trade-offs between using either of the two filters, or not filtering the image at all?

When we applied filter h2, with σ = 2.0, there is a much larger smoothing effect. A large amount of noise can be removed compared to filter h1 with σ = 1.0, but our image loses more of its details and we can observe that the image quality becomes more blur, Filter h1 maintains a balance between noise reduction and preserving the original image details.

Without any filters applied, we will preserve all of the original image details and sharpness, including the finer and sharper details, but there is quite a high amount of noise present.

3.4 View “lib-sp.jpg”

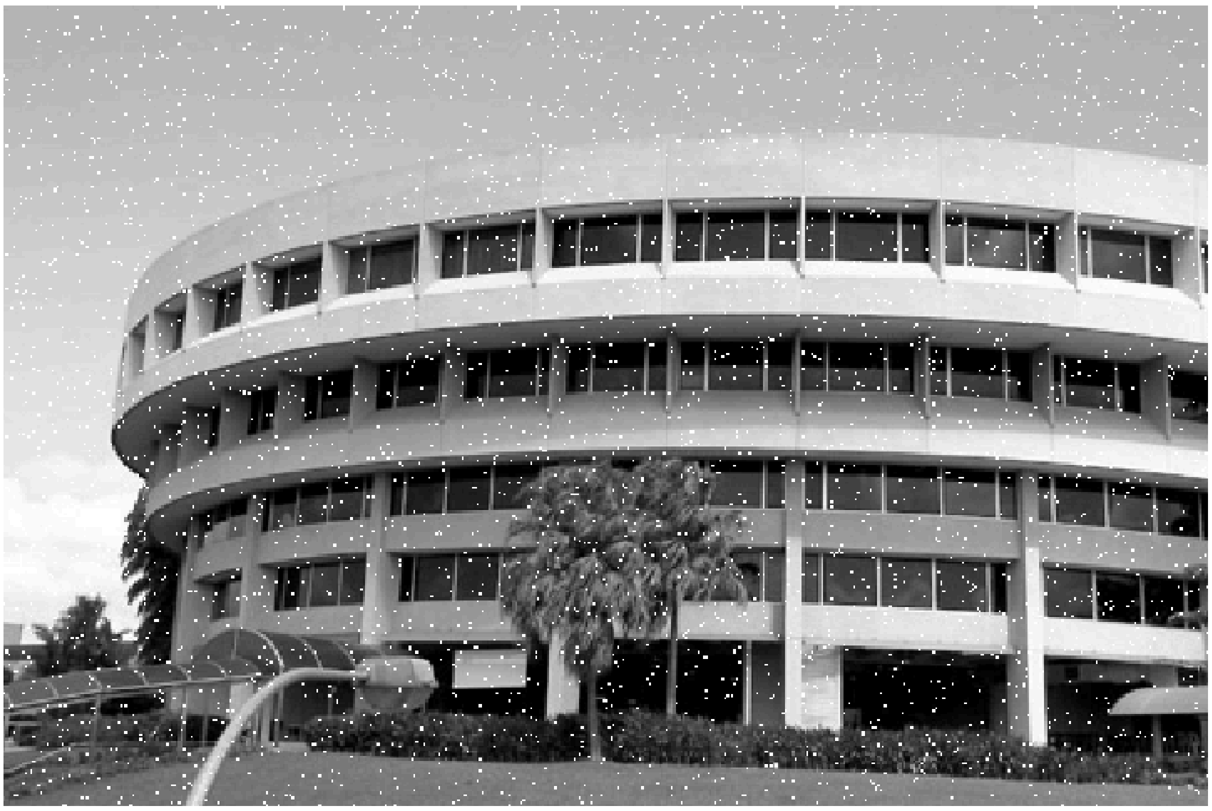


Fig 14. Lib-sp.jpg

3.5 Repeat step (c) above

```
% 2.3 (e) - Repeat step (c) above
Psp1 = uint8(conv2(Psp,h1));
figure
imshow(Psp1);
saveas(gcf, 'results/psp1.jpg');


Psp2 = uint8(conv2(Psp,h2));
figure
imshow(Psp2);
saveas(gcf, 'results/psp2.jpg');
```

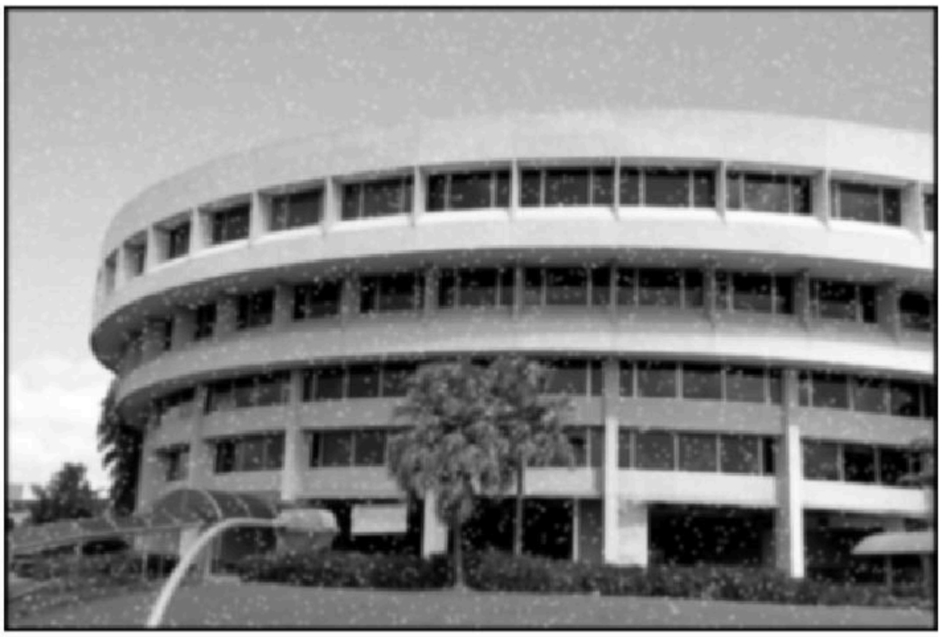


Fig 15. Lib-sp with h1 filter

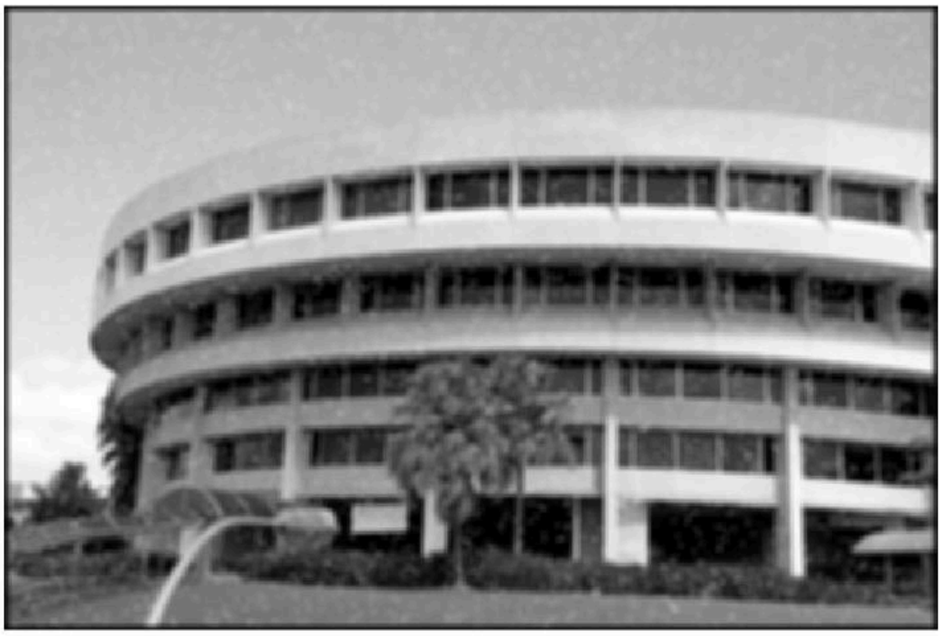
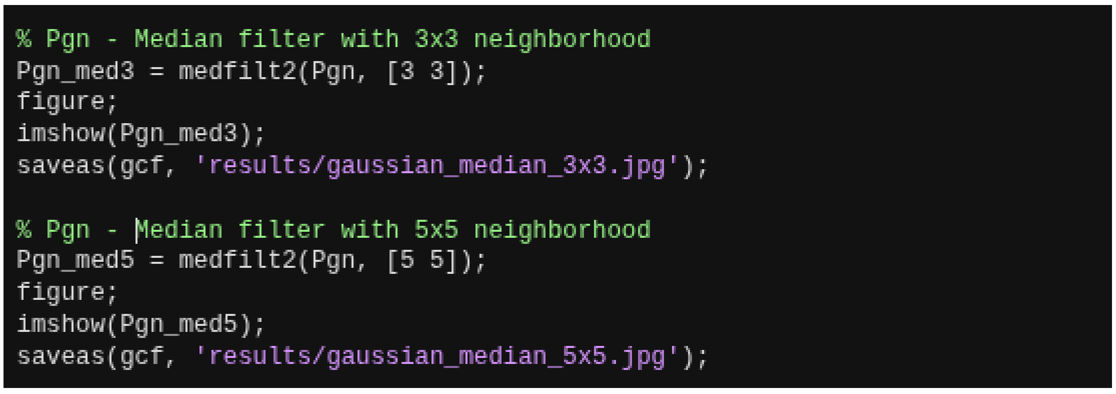


Fig 16. Lib-sp with h2 filter

Are the filters better at handling Gaussian noise or speckle noise?

We can see that filters are better at handling Gaussian noise, rather than speckle noise. The speckle noise is still quite obvious and visible after applying h1 filter and h2 filter.

4. Median Filtering



```
% Pgn - Median filter with 3x3 neighborhood
Pgn_med3 = medfilt2(Pgn, [3 3]);
figure;
imshow(Pgn_med3);
saveas(gcf, 'results/gaussian_median_3x3.jpg');

% Pgn - Median filter with 5x5 neighborhood
Pgn_med5 = medfilt2(Pgn, [5 5]);
figure;
imshow(Pgn_med5);
saveas(gcf, 'results/gaussian_median_5x5.jpg');
```

```
% Psp - Median filter with 3x3 neighborhood
Psp_med3 = medfilt2(Psp, [3 3]);
figure;
imshow(Psp_med3);
saveas(gcf, 'results/speckle_median_3x3.jpg');

% Psp - Median filter with 5x5 neighborhood
Psp_med5 = medfilt2(Psp, [5 5]);
figure;
imshow(Psp_med5);
saveas(gcf, 'results/speckle_median_5x5.jpg');
```

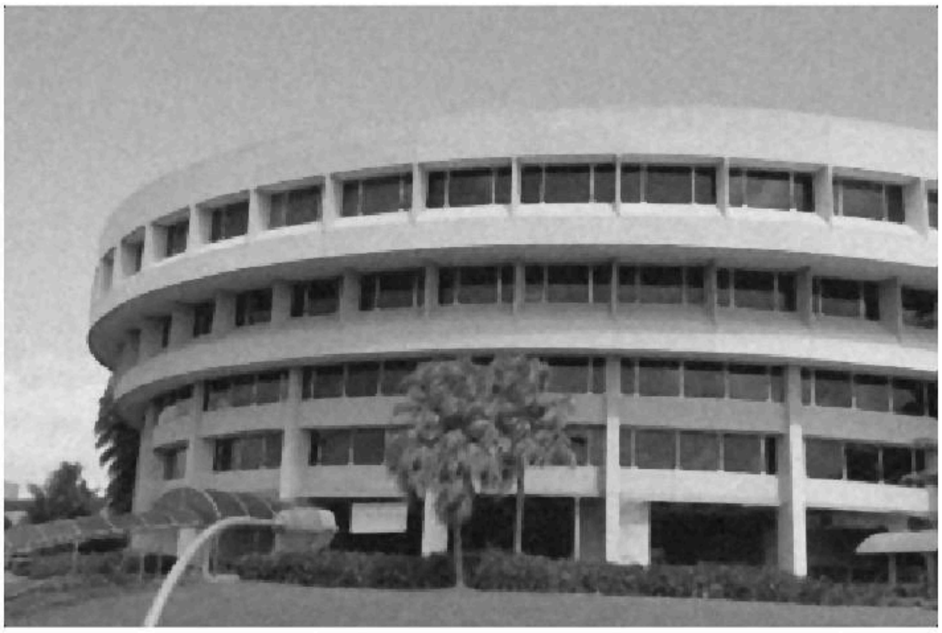


Fig 17. Lib-gn with median filter [3,3]

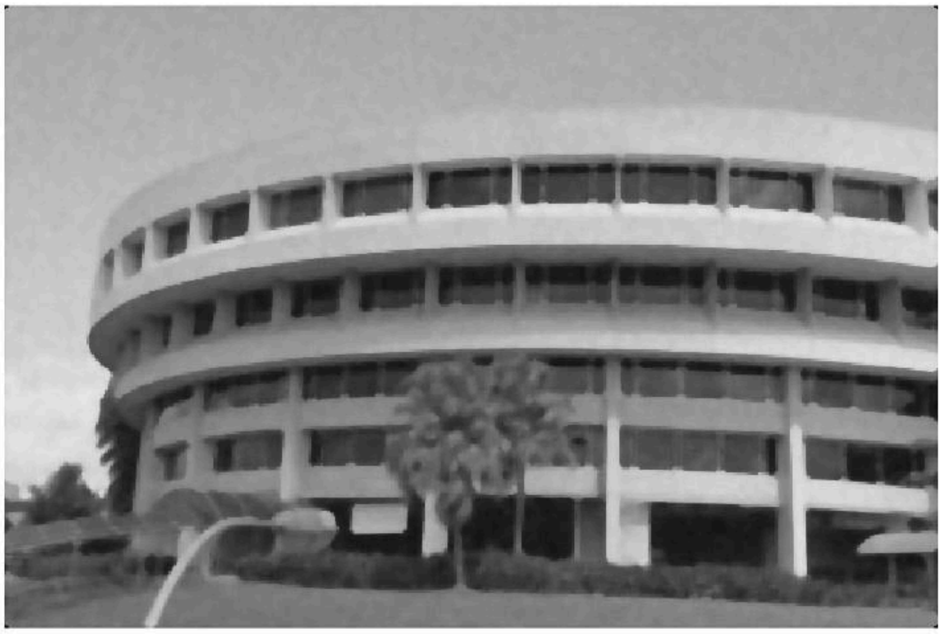


Fig 18. Lib-gn with median filter [5,5]

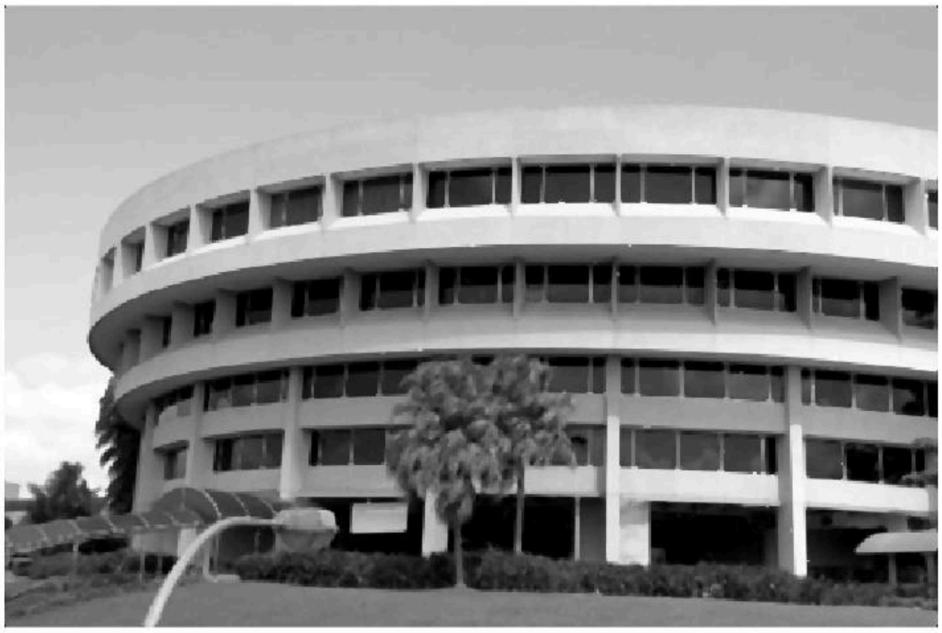


Fig 19. Lib-sp with median filter [3,3]

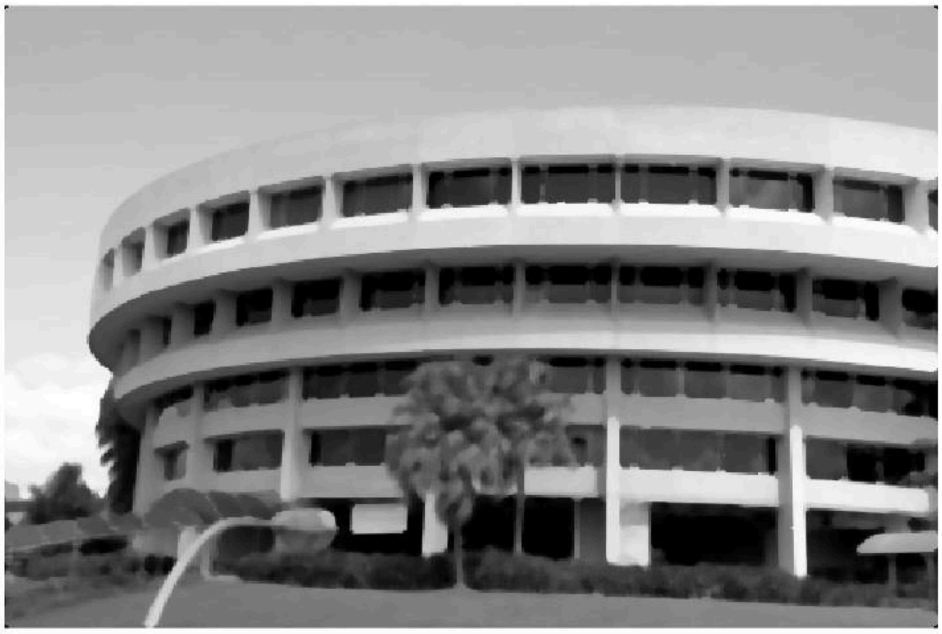


Fig 20. Lib-sp with median filter [5,5]

How does Gaussian filtering compare with median filtering in handling the different types of noise?

Gaussian filtering is more effective at handling Gaussian noise, but it performs worse at handling speckle noise compared to median filtering.

What are the tradeoffs?

For Gaussian filtering:

- Smoothens image by weighted averaging the entire image will be equally smoothened
- Effective at reducing gaussian noise
- Causes uniform blurring across entire image, blurring both noise and edges equally

For Median filtering:

- Preserves edges better while removing noise
- Performs better at isolated extreme values
- Does not blur the edges as much as gaussian filtering
- Larger neighbourhood size removes more noise but also result in greater loss of detail and increased blurring

5. Suppressing Noise Interference Patterns

5.1 View “pck-int.jpg”

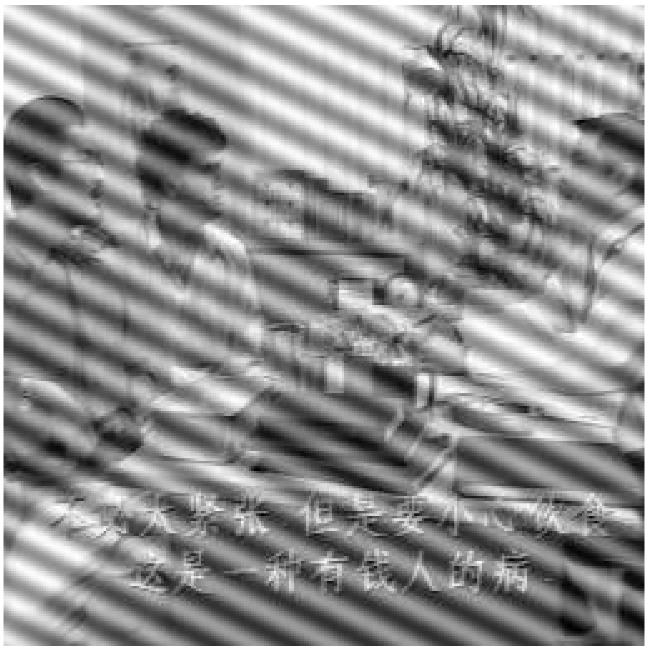
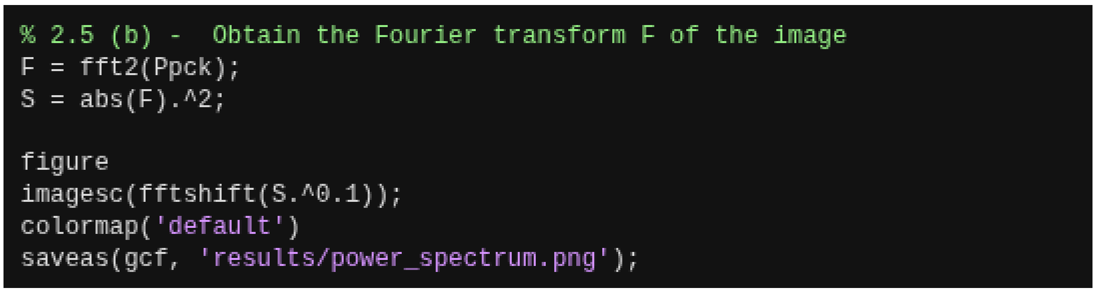


FIg 21. Pck-int.jpg

5.2 Obtain the Fourier transform F of the image using and subsequently compute the power spectrum S



```
% 2.5 (b) -  Obtain the Fourier transform F of the image
F = fft2(Ppck);
S = abs(F).^2;

figure
imagesc(fftshift(S.^0.1));
colormap('default')
saveas(gcf, 'results/power_spectrum.png');
```

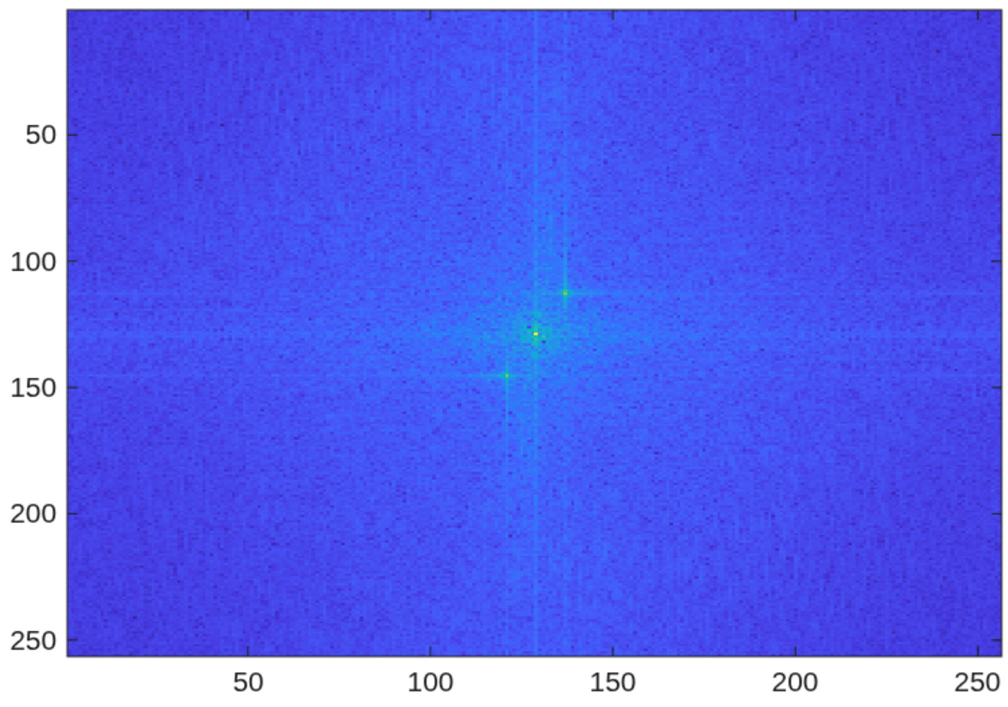
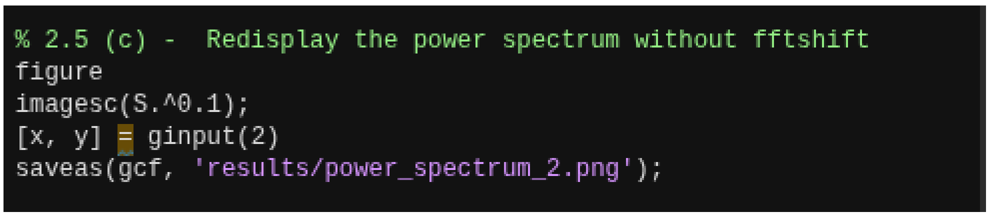


Fig 22. Power Spectrum

5.3 Redisplay the power spectrum without `fftshift`



```
% 2.5 (c) -  Redisplay the power spectrum without fftshift
figure
imagesc(S.^0.1);
[x, y] = ginput(2)
saveas(gcf, 'results/power_spectrum_2.png');
```

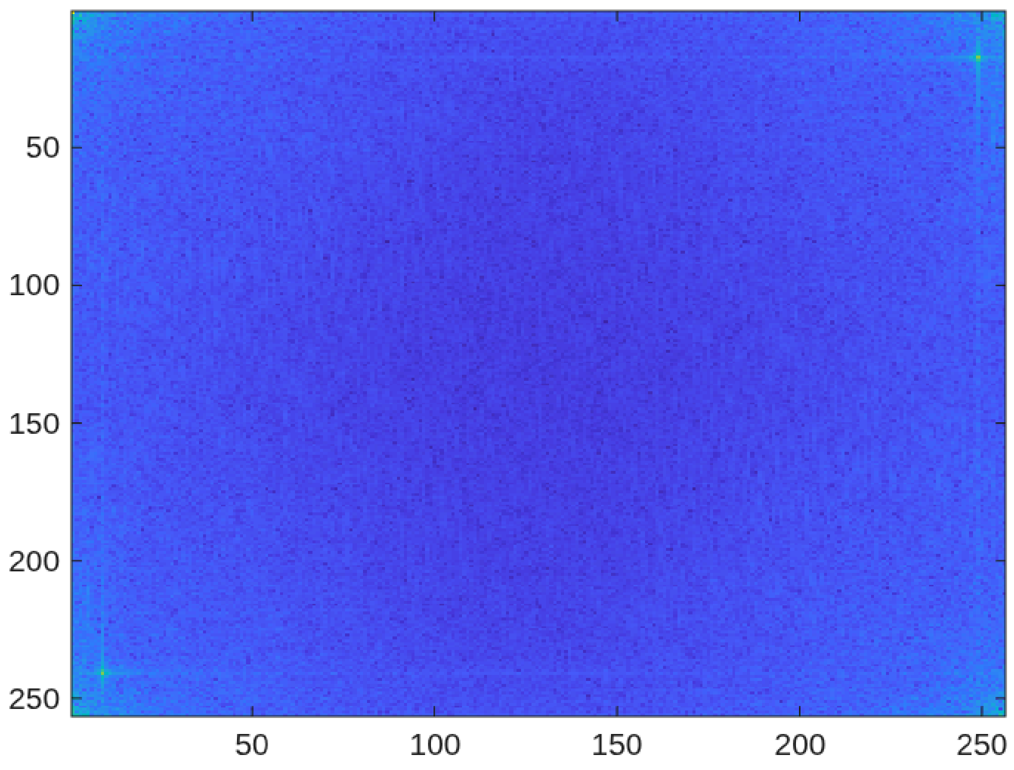


Fig 23. Power Spectrum without `fftshift`

Peaks:

- (9.2843, 240.8019)
- (240.8019, 17.8082)

5.4 Set to zero the 5x5 neighbourhood elements

```
% 2.5 (d) - Set to zero the 5x5 neighbourhood elements at locations
% corresponding to the above peaks

F(y1-2:y1+2, x1-2:x1+2) = 0;
F(y2-2:y2+2, x2-2:x2+2) = 0;

S = abs(F).^2;
figure
imagesc(fftshift(S.^0.1));
colormap('default');
saveas(gcf, 'results/power_spectrum_adjusted.png');
```

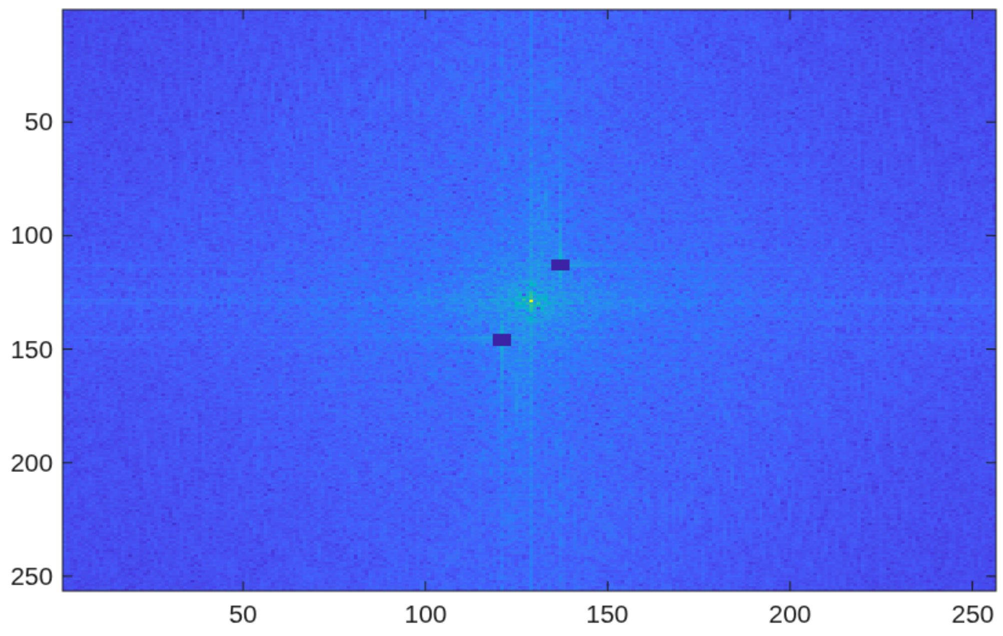


Fig 24. Power Spectrum after setting to 0 the 5x5 neighbourhood element at the above peaks

5.5 Compute the inverse Fourier transform using `ifft2`

```
F_inv = real(ifft2(F));
F_inv = uint8(F_inv);

figure;
imshow(F_inv);
saveas(gcf, 'results/f_inverse_1.jpg');
```

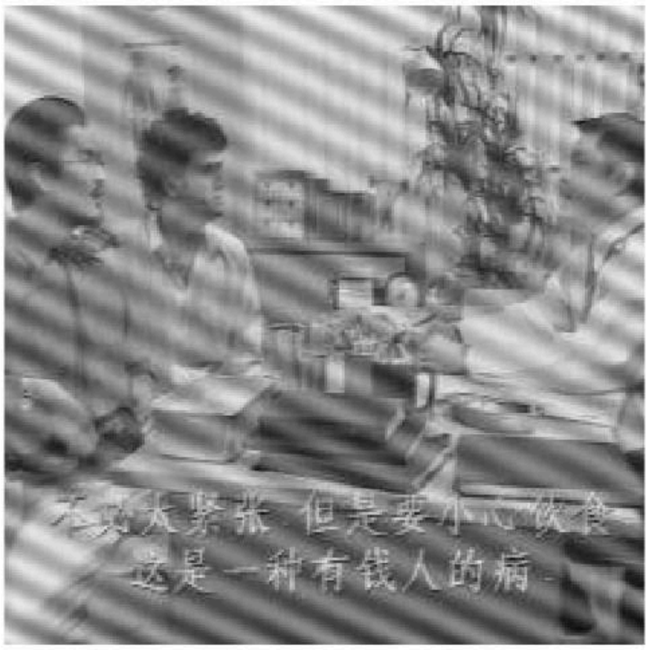


Fig 25. Pck-int after using inverse Fourier transform

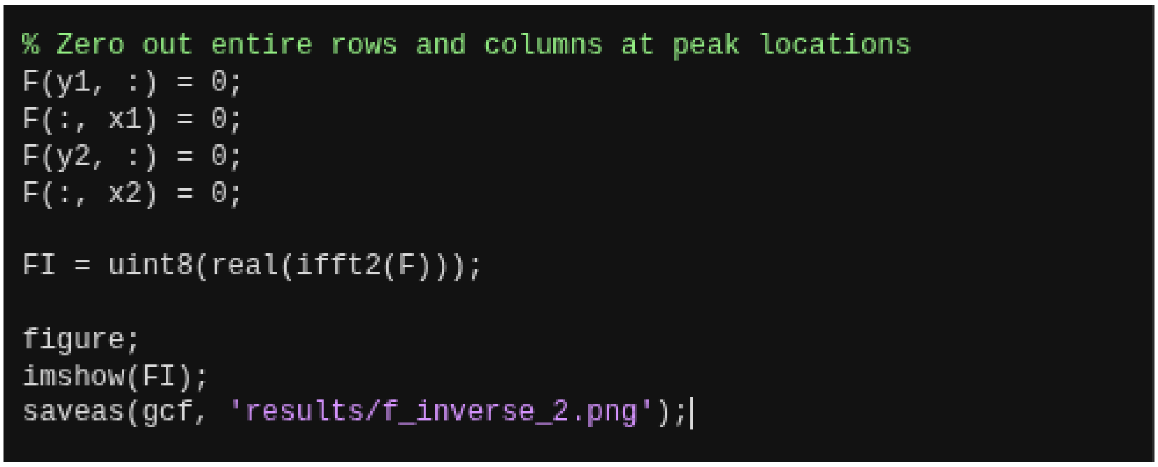
Comment on the result and how this relates to step (c).

The diagonals are partially reduced, but are not completely removed, especially at the edges of the screen. In part (c), we set the neighbouring elements that extend from the peak points in a [5,5] area. By removing these specific frequencies from the Fourier transform, we eliminate the wave patterns that create the diagonal lines in the image, hence reducing the prominence of the diagonal lines in our image.

Attempt to further improve the result.

As a possible improvement, rather than zero out the 5×5 neighbours, we can set the entire horizontal and vertical lines that extend from the peak to zero. The diagonal pattern does not exist only at the peak points, but also along the frequency components that spread along the horizontal and vertical lines in the frequency domain. Removing the entire lines will help us reduce the diagonal lines even further.

As seen below in Fig 26, the diagonal lines are further reduced in doing the proposed improvements.



```
% Zero out entire rows and columns at peak locations
F(y1, :) = 0;
F(:, x1) = 0;
F(y2, :) = 0;
F(:, x2) = 0;

FI = uint8(real(ifft2(F)));

figure;
imshow(FI);
saveas(gcf, 'results/f_inverse_2.png');
```

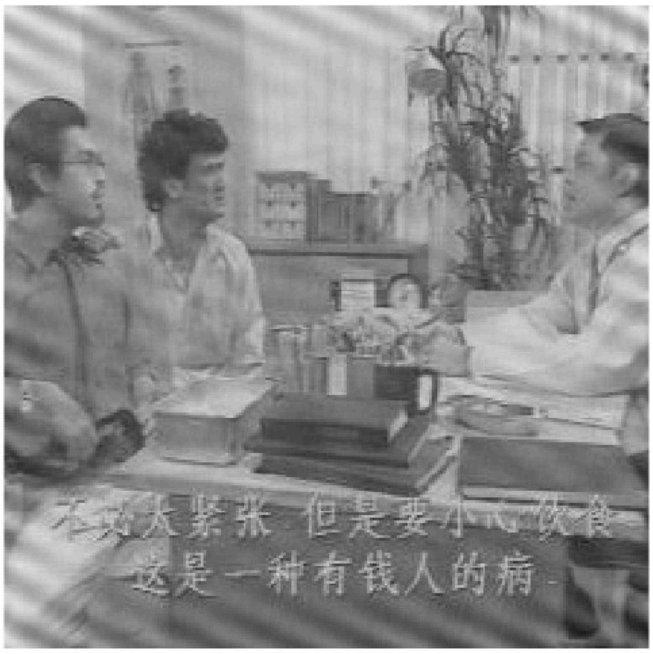
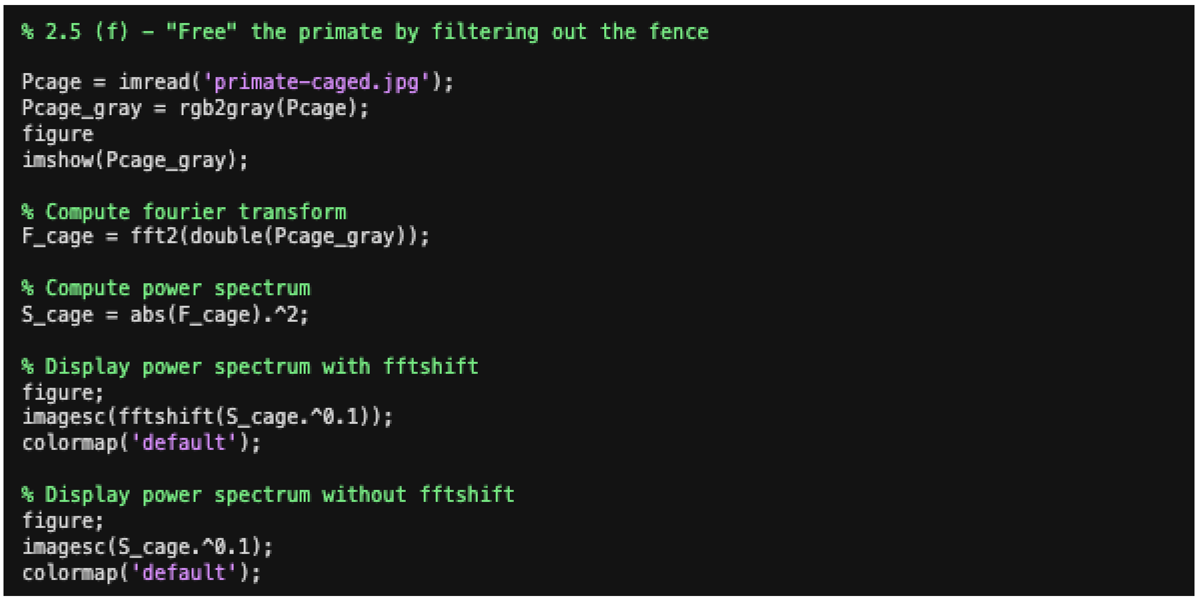


Fig 26. Pck-int after 2nd improvement

5.6 “Free” the primate by filtering out the fence



```
% 2.5 (f) - "Free" the primate by filtering out the fence

Pcage = imread('primate-caged.jpg');
Pcage_gray = rgb2gray(Pcage);
figure
imshow(Pcage_gray);

% Compute fourier transform
F_cage = fft2(double(Pcage_gray));

% Compute power spectrum
S_cage = abs(F_cage).^2;

% Display power spectrum with fftshift
figure;
imagesc(fftshift(S_cage.^0.1));
colormap('default');

% Display power spectrum without fftshift
figure;
imagesc(S_cage.^0.1);
colormap('default');
```

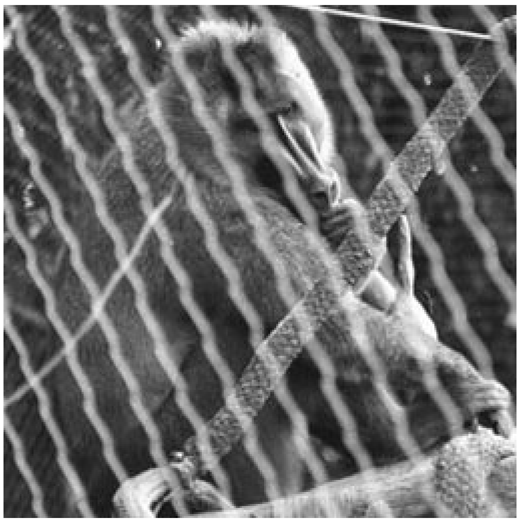


Fig 27. Primate-caged.jpg

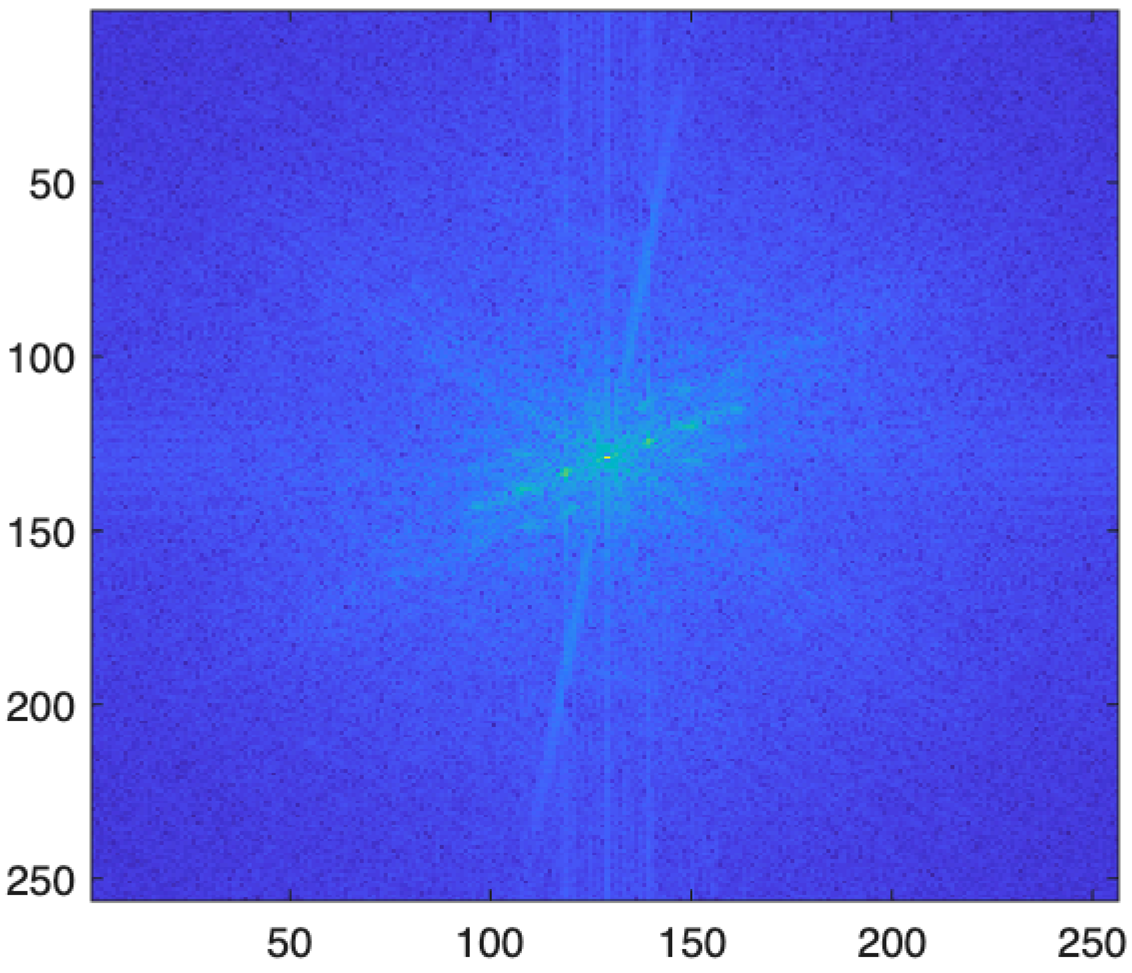


Fig 28. Power spectrum with `fftshift`

We define an inner rectangular filtering region from `(2,2)` to `(rows-1, cols-1)`. Applying a similar concept as we have done above, we are removing some frequency components by setting them to zero. In this case, rather than manually selecting the peaks, we iterate through the entire inner rectangular region and compare each frequency value with the threshold value. If it exceeds the threshold value, we set it to 0.

Intensity threshold selected in the end: **1725783662.54**

```
% Select intensity threshold
[col, row] = ginput(1);
intensity_threshold = S_cage(round(row), round(col));

fprintf('Selected intensity threshold: %.2f\n', intensity_threshold);
saveas(gcf, 'results/primate_power_spectrum_click.png');

% Define rectangular filtering region
[rows, cols] = size(S_cage);

x1 = 2;
y1 = 2;
x2 = cols - 1;
y2 = rows - 1;

fprintf('Filtering region: rows %d to %d, cols %d to %d\n', y1, y2, x1, x2);

% Filter out high-intensity frequencies based on my earlier selection
for y = y1:y2
    for x = x1:x2
        if S_cage(y, x) > intensity_threshold
            F_cage(y, x) = 0;
        end
    end
end
```

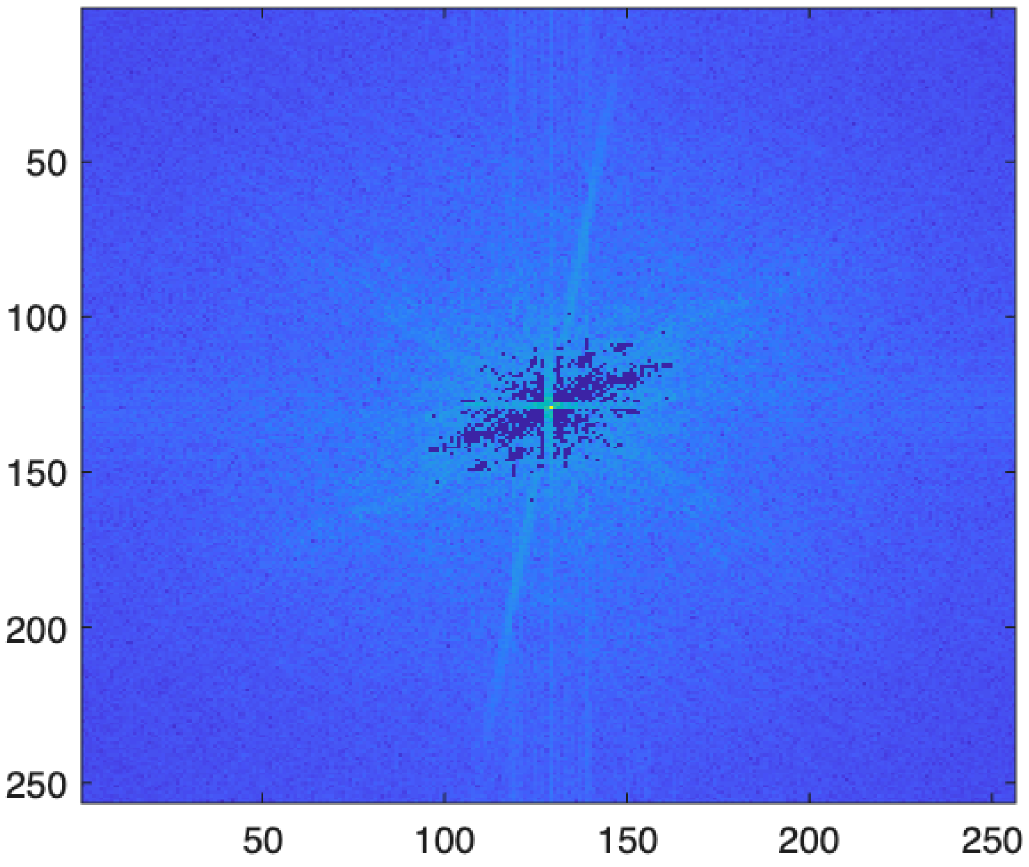


Fig 29. Power spectrum with `ffshift` post-improvement

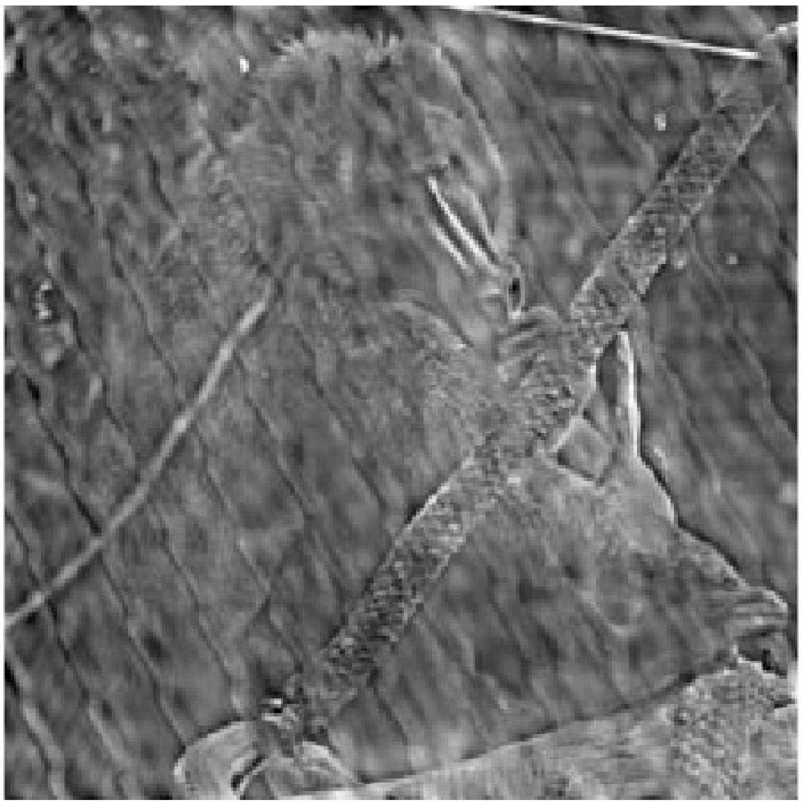


Fig 30. Primate-caged.jpg after being “freed”

6. Undoing Perspective Distortion of Planar Surface

6.1 Download `book.jpg' and display the image

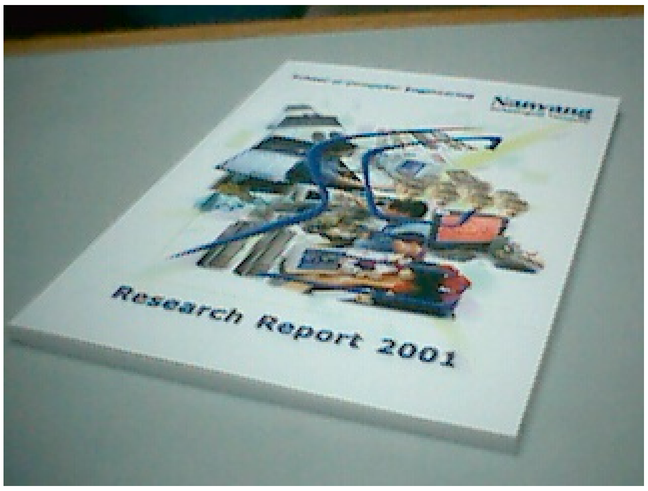
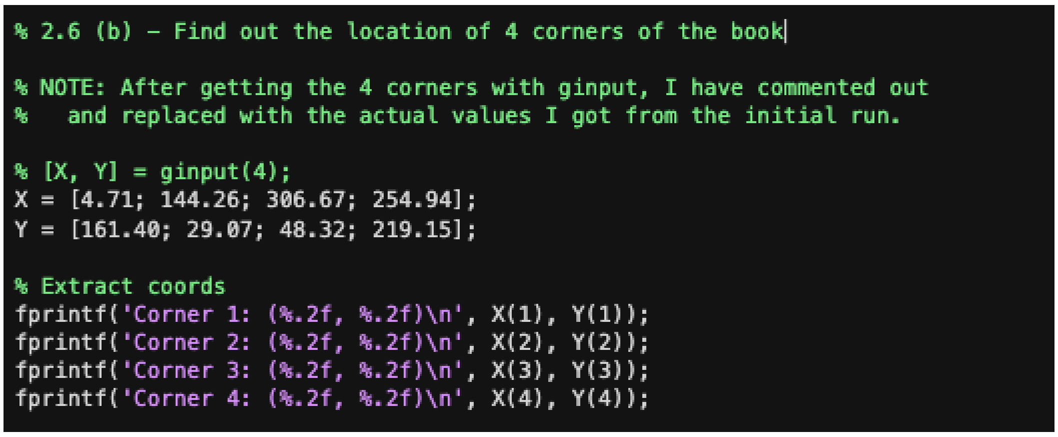


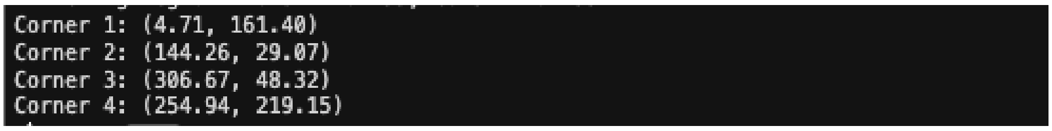
Fig 31. book.jpg

6.2 Find out the location of 4 corners of the book



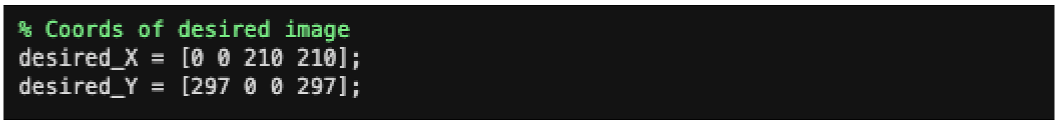
```
% 2.6 (b) - Find out the location of 4 corners of the book

% NOTE: After getting the 4 corners with ginput, I have commented out
%   and replaced with the actual values I got from the initial run.

% [X, Y] = ginput(4);
X = [4.71; 144.26; 306.67; 254.94];
Y = [161.40; 29.07; 48.32; 219.15];

% Extract coords
fprintf('Corner 1: (%.2f, %.2f)\n', X(1), Y(1));
fprintf('Corner 2: (%.2f, %.2f)\n', X(2), Y(2));
fprintf('Corner 3: (%.2f, %.2f)\n', X(3), Y(3));
fprintf('Corner 4: (%.2f, %.2f)\n', X(4), Y(4));
```

4 Corners of the book:



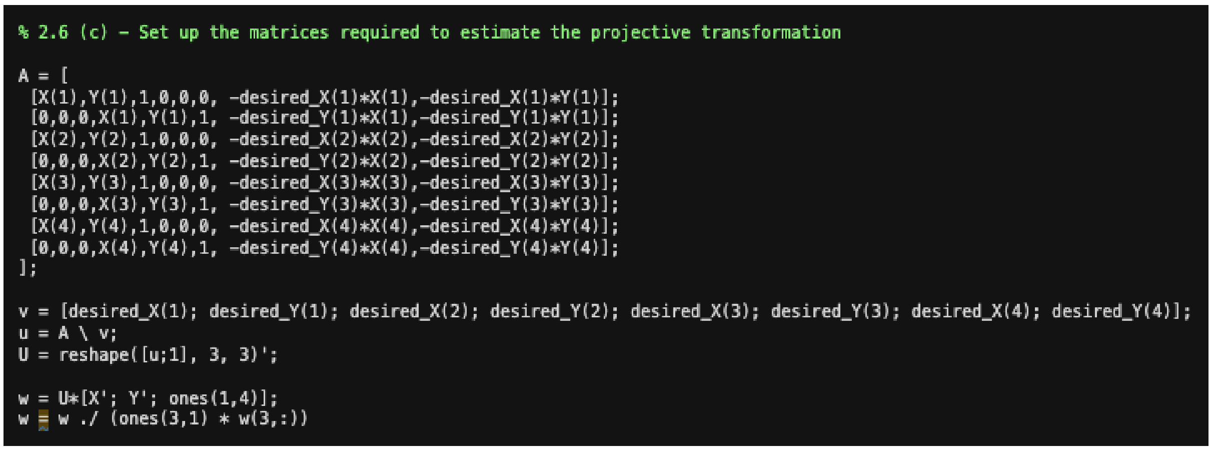
```
Corner 1: (4.71, 161.40)
Corner 2: (144.26, 29.07)
Corner 3: (306.67, 48.32)
Corner 4: (254.94, 219.15)
```

Coords of desired image:



```
% Coords of desired image
desired_X = [0 0 210 210];
desired_Y = [297 0 0 297];
```

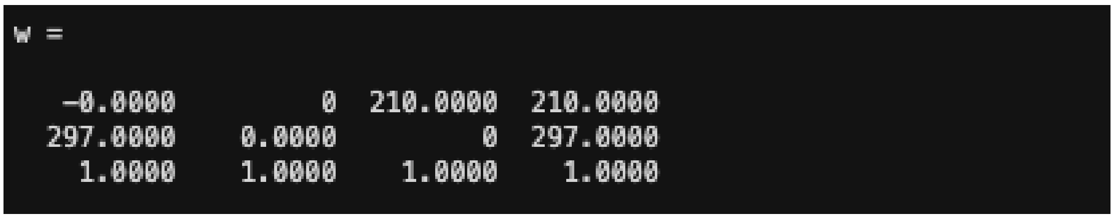
6.3 Set up the matrices required to estimate the projective transformation



```
% 2.6 (c) - Set up the matrices required to estimate the projective transformation

A = [
 [X(1),Y(1),1,0,0,0, -desired_X(1)*X(1),-desired_X(1)*Y(1)];
 [0,0,0,X(1),Y(1),1, -desired_Y(1)*X(1),-desired_Y(1)*Y(1)];
 [X(2),Y(2),1,0,0,0, -desired_X(2)*X(2),-desired_X(2)*Y(2)];
 [0,0,0,X(2),Y(2),1, -desired_Y(2)*X(2),-desired_Y(2)*Y(2)];
 [X(3),Y(3),1,0,0,0, -desired_X(3)*X(3),-desired_X(3)*Y(3)];
 [0,0,0,X(3),Y(3),1, -desired_Y(3)*X(3),-desired_Y(3)*Y(3)];
 [X(4),Y(4),1,0,0,0, -desired_X(4)*X(4),-desired_X(4)*Y(4)];
 [0,0,0,X(4),Y(4),1, -desired_Y(4)*X(4),-desired_Y(4)*Y(4)];
];

v = [desired_X(1); desired_Y(1); desired_X(2); desired_Y(2); desired_X(3); desired_Y(3); desired_X(4); desired_Y(4)];
u = A \ v;
U = reshape([u;1], 3, 3)';

w = U*[X'; Y'; ones(1,4)];
w = w ./ (ones(3,1) * w(3,:))
```

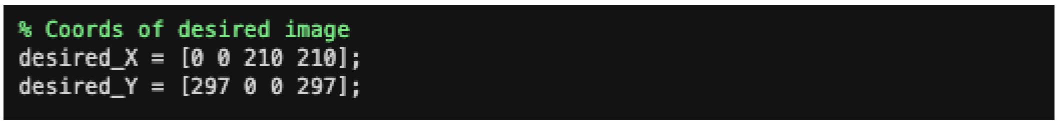
Does the transformation give you back the 4 corners of the desired image?

Below is the screenshot of the result of my w:



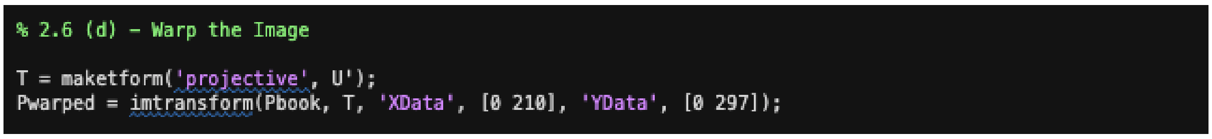
```
w =

   -0.0000         0  210.0000  210.0000
  297.0000    0.0000         0  297.0000
    1.0000    1.0000    1.0000    1.0000
```

Yes, the top 2 rows are the 4 corners of my desired image, and this matches the coordinates of my desired image, shown here (*same as above screenshot*):



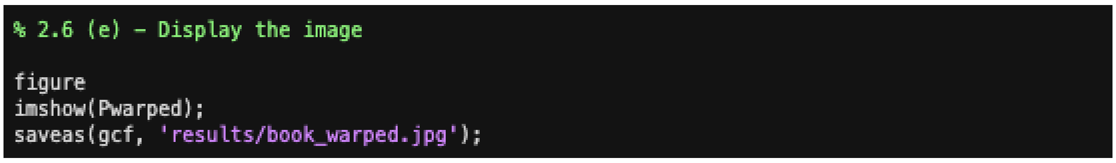
```
% Coords of desired image
desired_X = [0 0 210 210];
desired_Y = [297 0 0 297];
```

6.4 Warp the Image



```
% 2.6 (d) - Warp the Image

T = maketform('projective', U');
Pwarped = imtransform(Pbook, T, 'XData', [0 210], 'YData', [0 297]);
```

6.5 Display the warped image



```
% 2.6 (e) - Display the image

figure
imshow(Pwarped);
saveas(gcf, 'results/book_warped.jpg');
```

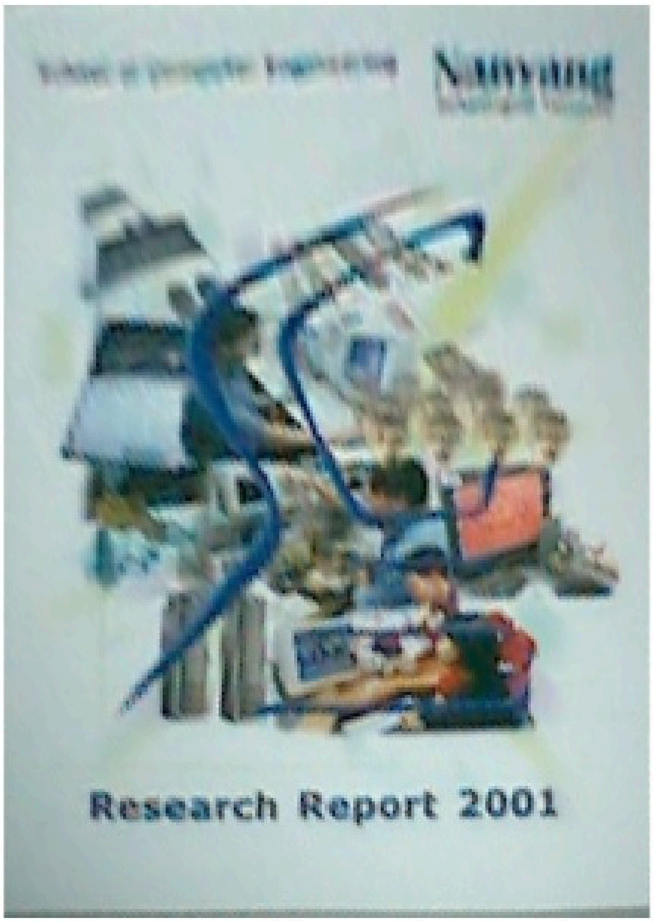


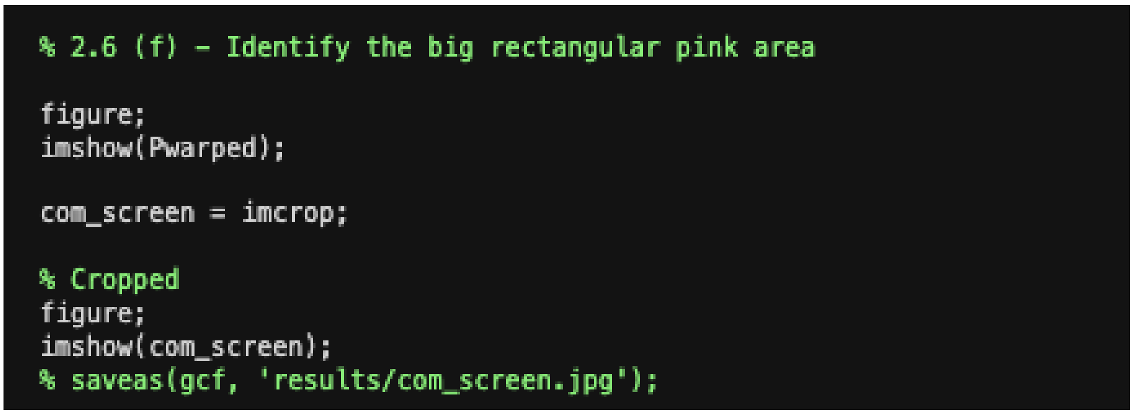
Fig 32. Warped image

Is this what you expect? Comment on the quality of the transformation and suggest reasons.

Yes, this is expected as the warped image is now showing a frontal view of the book. However, the quality has reduced, with the pictures and the features getting more distorted when we shift the perspective.

One possible explanation is because when we are mapping from slanted to frontal view, some parts of the image are stretched and compressed differently, causing the image to become more distorted. The pixels are also getting redistributed artificially, resulting in a loss of detail when we shift the view.

6.6 Identify the big rectangular pink area



```
% 2.6 (f) - Identify the big rectangular pink area

figure;
imshow(Pwarped);

com_screen = imcrop;

% Cropped
figure;
imshow(com_screen);
% saveas(gcf, 'results/com_screen.jpg');
```

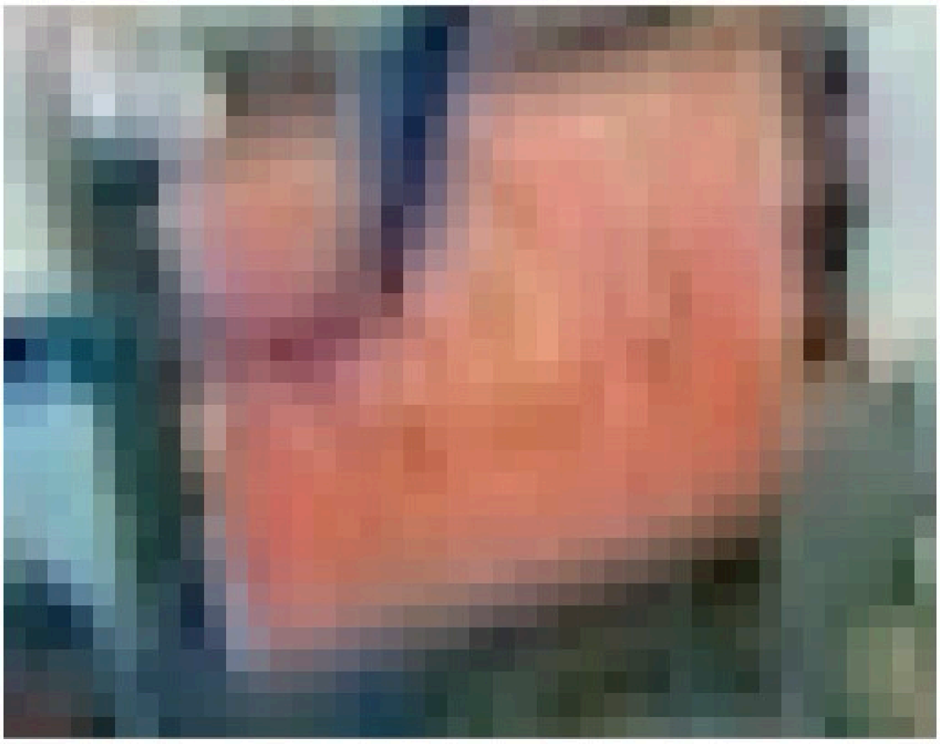
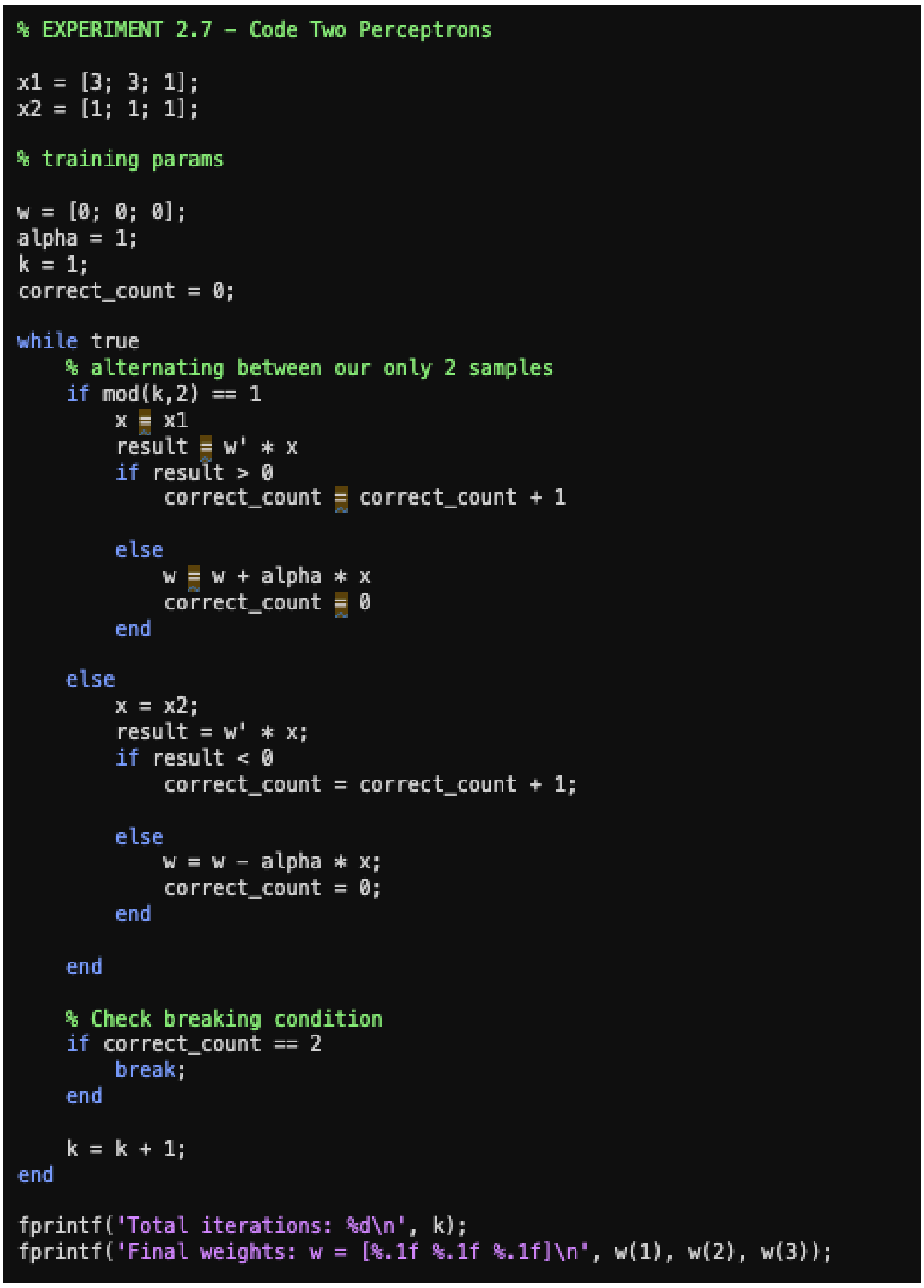


Fig 33. Rectangular pink area

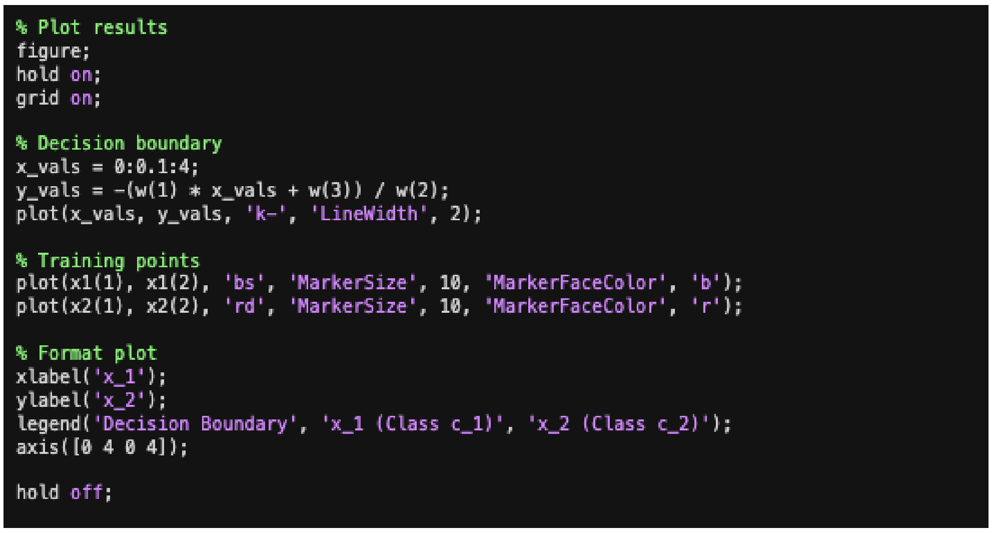
It looks like a smiley face on a screen, pink in color.

7. Code Two Perceptrons

7.1 Algorithm 1



```
% EXPERIMENT 2.7 - Code Two Perceptrons

x1 = [3; 3; 1];
x2 = [1; 1; 1];

% training params

w = [0; 0; 0];
alpha = 1;
k = 1;
correct_count = 0;

while true
    % alternating between our only 2 samples
    if mod(k,2) == 1
        x = x1
        result = w' * x
        if result > 0
            correct_count = correct_count + 1

        else
            w = w + alpha * x
            correct_count = 0
        end

    else
        x = x2;
        result = w' * x;
        if result < 0
            correct_count = correct_count + 1;

        else
            w = w - alpha * x;
            correct_count = 0;
        end

    end

    % Check breaking condition
    if correct_count == 2
        break;
    end

    k = k + 1;
end

fprintf('Total iterations: %d\n', k);
fprintf('Final weights: w = [%.1f %.1f %.1f]\n', w(1), w(2), w(3));
```



```
% Plot results
figure;
hold on;
grid on;

% Decision boundary
x_vals = 0:0.1:4;
y_vals = -(w(1) * x_vals + w(3)) / w(2);
plot(x_vals, y_vals, 'k-', 'LineWidth', 2);

% Training points
plot(x1(1), x1(2), 'bs', 'MarkerSize', 10, 'MarkerFaceColor', 'b');
plot(x2(1), x2(2), 'rd', 'MarkerSize', 10, 'MarkerFaceColor', 'r');

% Format plot
xlabel('x_1');
ylabel('x_2');
legend('Decision Boundary', 'x_1 (Class c_1)', 'x_2 (Class c_2)');
axis([0 4 0 4]);

hold off;
```

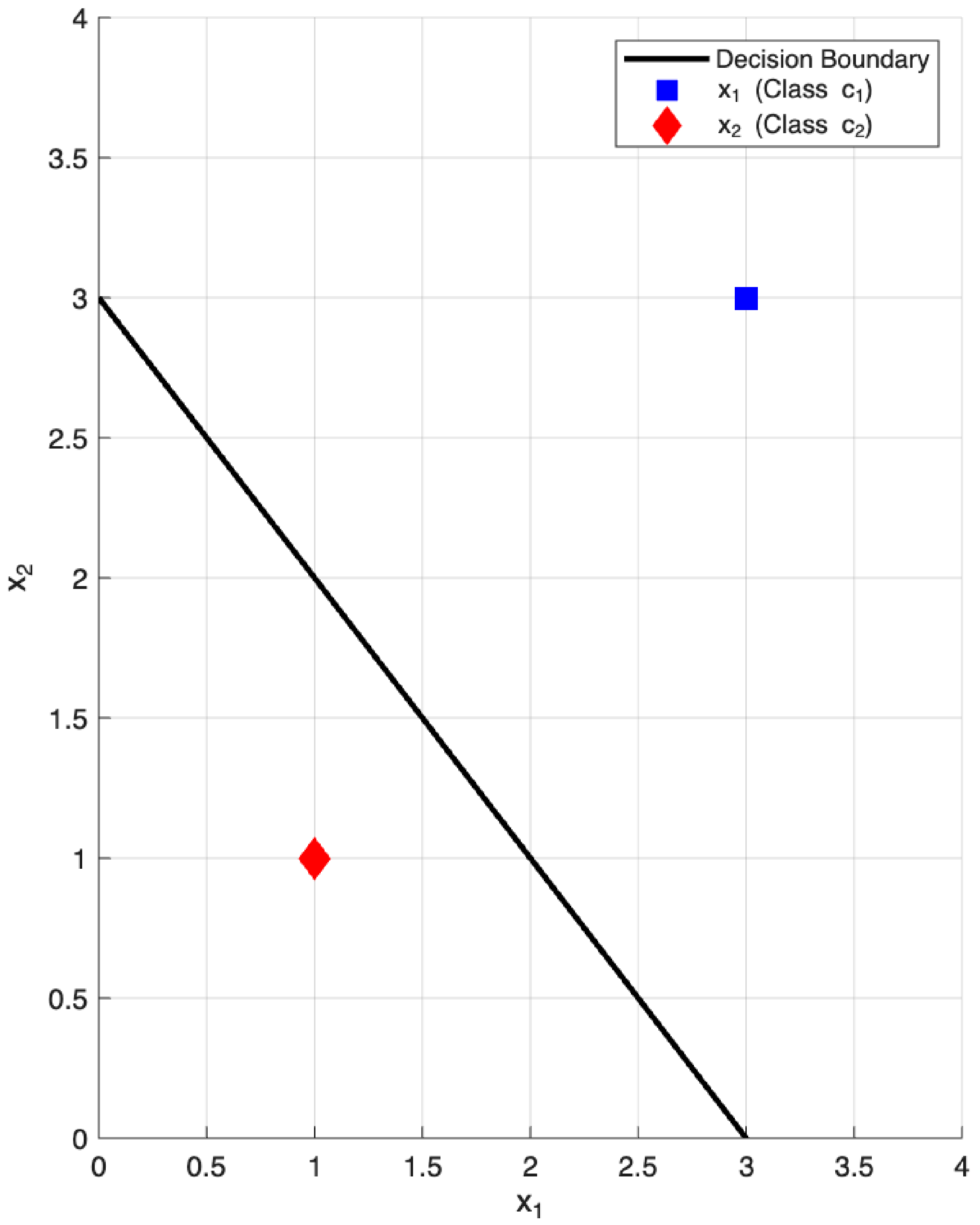
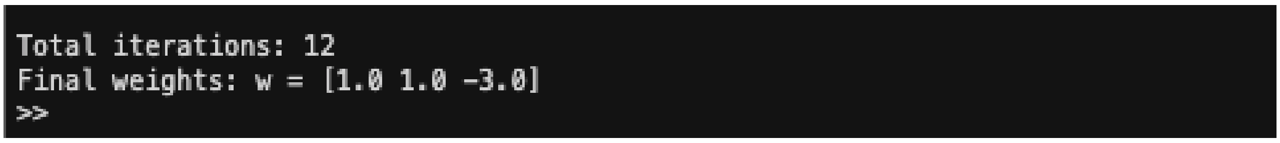


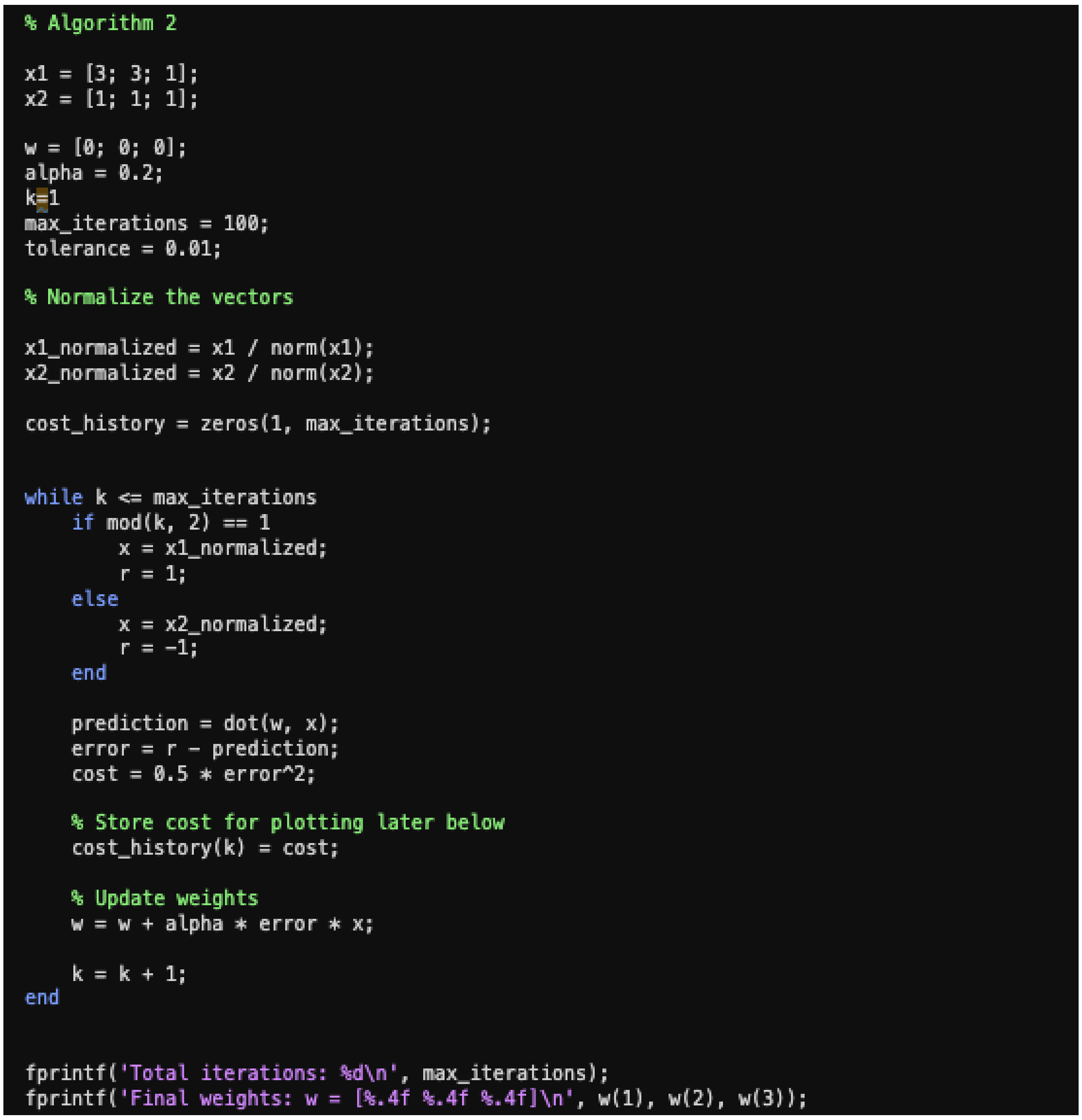
Fig 34. Algo 1’s Decision Boundary

Results for Algo 1:

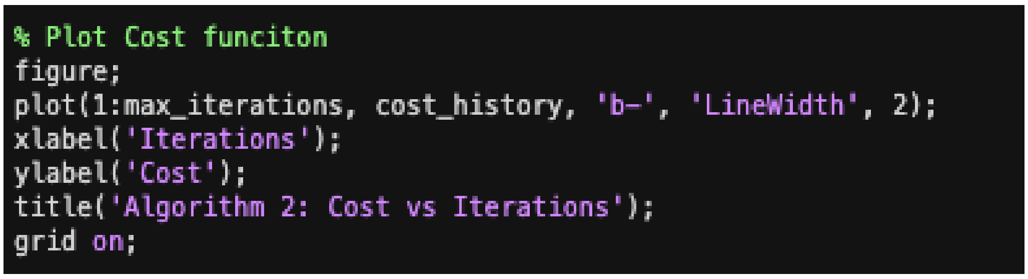


```
Total iterations: 12
Final weights: w = [1.0 1.0 -3.0]
>>
```

7.2 Algorithm 2



```
% Algorithm 2

x1 = [3; 3; 1];
x2 = [1; 1; 1];

w = [0; 0; 0];
alpha = 0.2;
k=1
max_iterations = 100;
tolerance = 0.01;

% Normalize the vectors

x1_normalized = x1 / norm(x1);
x2_normalized = x2 / norm(x2);

cost_history = zeros(1, max_iterations);


while k <= max_iterations
    if mod(k, 2) == 1
        x = x1_normalized;
        r = 1;
    else
        x = x2_normalized;
        r = -1;
    end

    prediction = dot(w, x);
    error = r - prediction;
    cost = 0.5 * error^2;

    % Store cost for plotting later below
    cost_history(k) = cost;

    % Update weights
    w = w + alpha * error * x;

    k = k + 1;
end


fprintf('Total iterations: %d\n', max_iterations);
fprintf('Final weights: w = [%.4f %.4f %.4f]\n', w(1), w(2), w(3));
```



```
% Plot Cost funciton
figure;
plot(1:max_iterations, cost_history, 'b-', 'LineWidth', 2);
xlabel('Iterations');
ylabel('Cost');
title('Algorithm 2: Cost vs Iterations');
grid on;
```

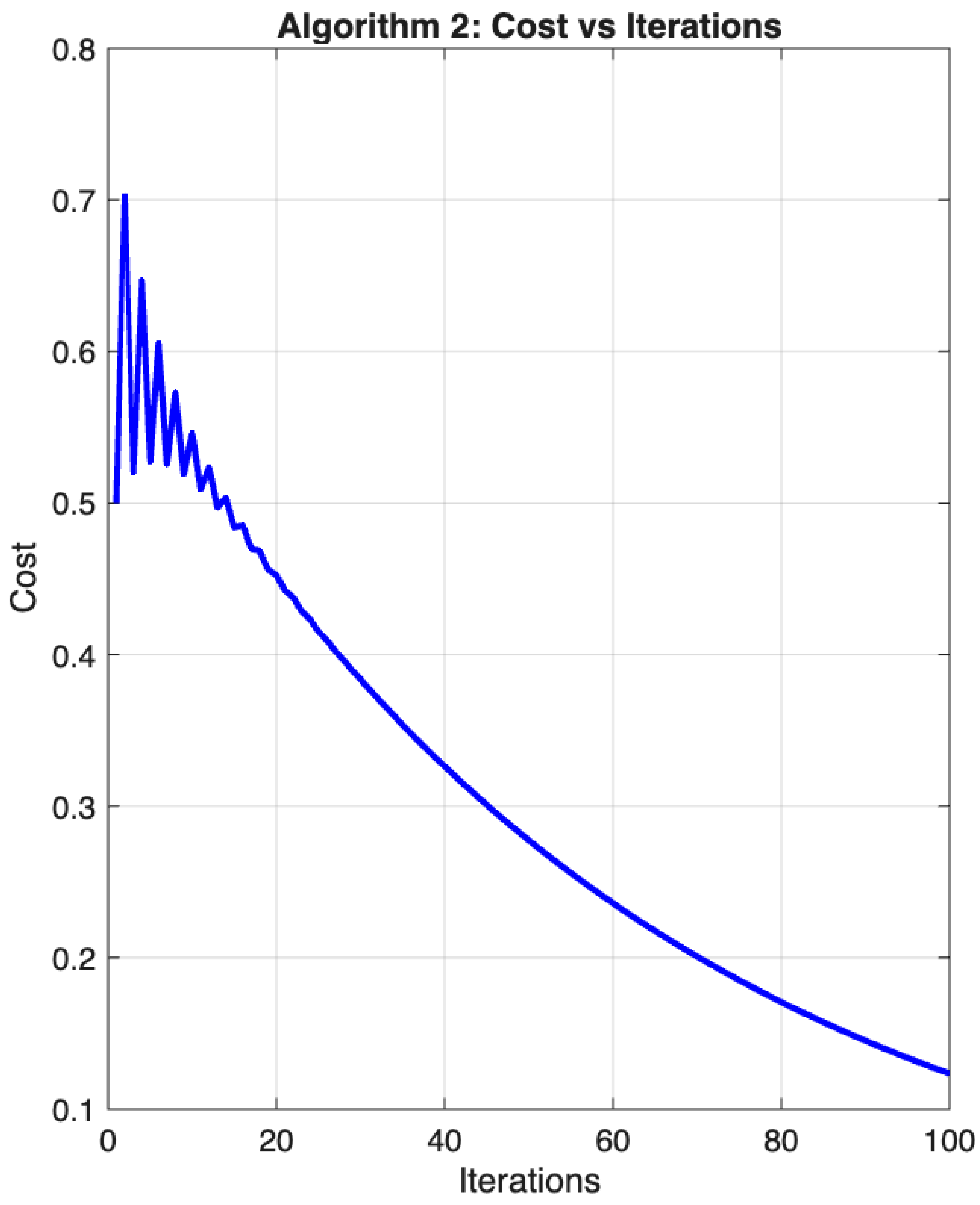
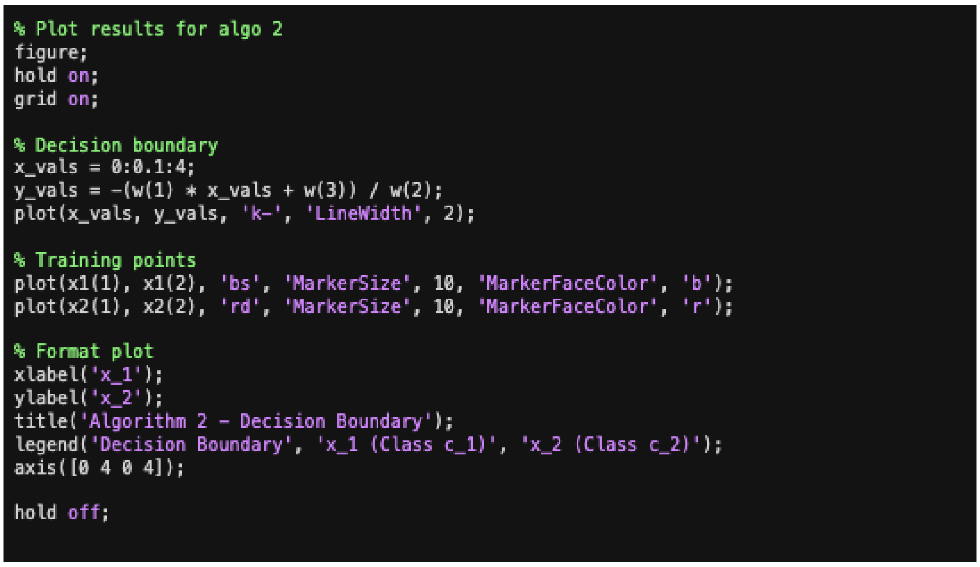


Fig 35. Cost vs Iteration for Algo 2



```
% Plot results for algo 2
figure;
hold on;
grid on;

% Decision boundary
x_vals = 0:0.1:4;
y_vals = -(w(1) * x_vals + w(3)) / w(2);
plot(x_vals, y_vals, 'k-', 'LineWidth', 2);

% Training points
plot(x1(1), x1(2), 'bs', 'MarkerSize', 10, 'MarkerFaceColor', 'b');
plot(x2(1), x2(2), 'rd', 'MarkerSize', 10, 'MarkerFaceColor', 'r');

% Format plot
xlabel('x_1');
ylabel('x_2');
title('Algorithm 2 - Decision Boundary');
legend('Decision Boundary', 'x_1 (Class c_1)', 'x_2 (Class c_2)');
axis([0 4 0 4]);

hold off;
```

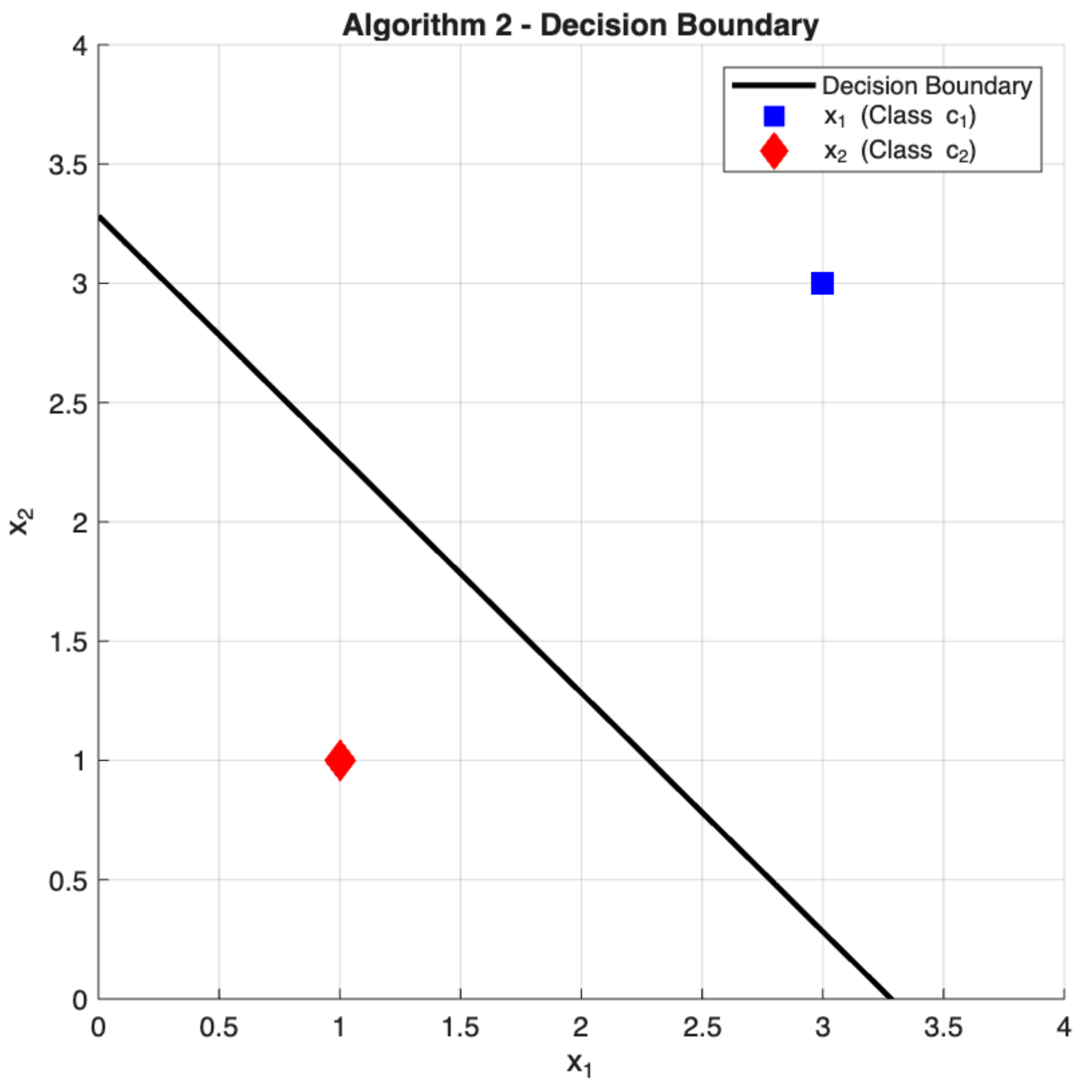


Fig 36. Algo 2’s Decision Boundary

Implementation:

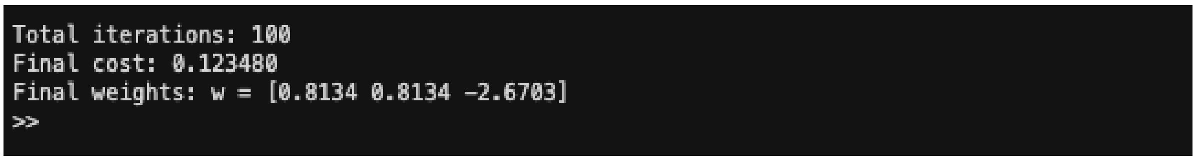
Algorithm 2 runs for 100 iterations, where at every iteration, we will calculate the loss using the chosen loss function, and then adjust the weights over the iterations. Learning rate of 0.1 is used in this case.

The loss function chosen is given in the lecture:

$$\boxed{E(\boldsymbol{w}) = \frac{1}{2}\left(r - \boldsymbol{w}^T\mathbf{x}\right)^2}$$

We are applying **Stochastic Gradient Descent (SGD)**, where we update the weights 1 sample at a time, as seen in our code alternating between x1 and x2 at each iteration.

Results for Algo 2:



```
Total iterations: 100
Final cost: 0.123480
Final weights: w = [0.8134 0.8134 -2.6703]
>>
```

Compare and comment your results.

	Algorithm 1	**Algorithm 2**
Final Weights	w = [1.0 1.0 -3.0]	w = [0.8134 0.8134 -2.6703]
Iterations	12	100

Both algorithms are able to learn and separate the 2 classes. Algorithm 1 is much simpler and faster, it only makes corrections when samples are misclassified, and terminates once it achieves sufficient consecutive correct classifications. In our case, it took only 12 iterations.

Algorithm 2 uses gradient descent with normalized inputs for a number of iterations (100 in this case), providing a much smoother convergence towards the final weights, along with a traceable cost function that we observe decreasing exponentially over the iteration.

Algorithm 2 would provide a more stable and traceable training, while algorithm 1 can potentially be faster in this case where we have 2 data points only.