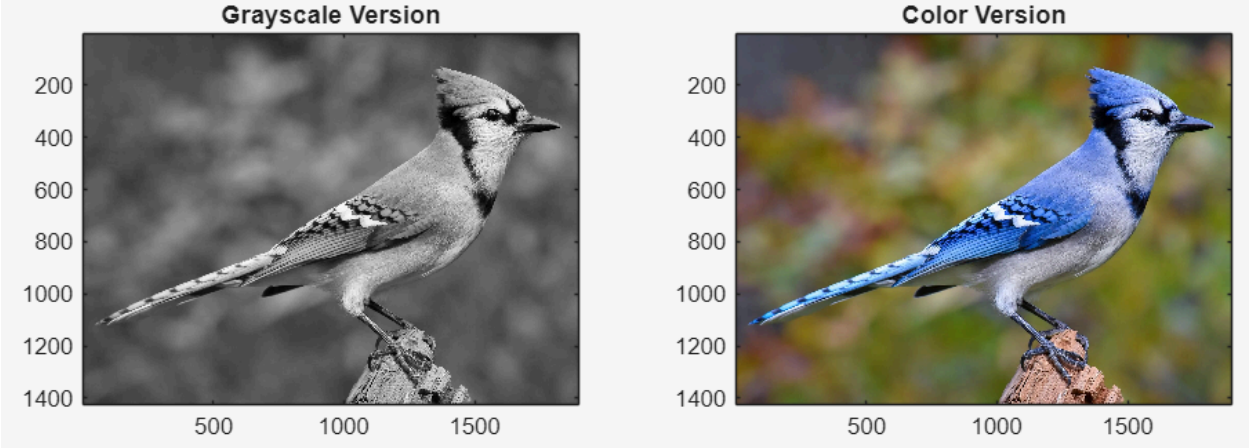
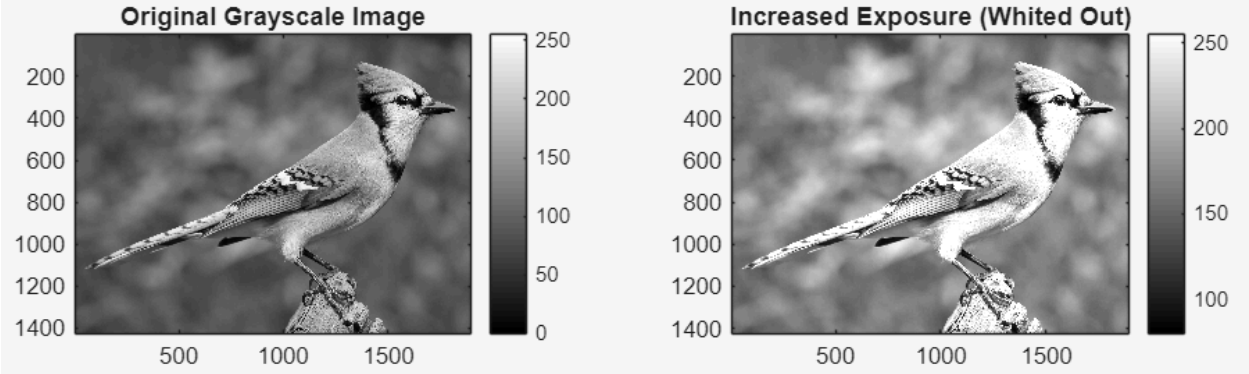
```
%Task 5.1
function [X_gray, X_red, X_green, X_blue] = readImage(filename)
   % Read the image file
   X_int = imread(filename);

   % Convert to double format for mathematical operations
   X_double = double(X_int);

   % Extract individual color channels
   X_red = X_double(:,:,1);
   X_green = X_double(:,:,2);
   X_blue = X_double(:,:,3);

   % Create grayscale image by averaging the three color channels
   X_gray = X_double(:,:,1)/3.0 + X_double(:,:,2)/3.0 + X_double(:,:,3)/3.0;
end
% Script to display grayscale and color versions of rectangle.jpg
% Make sure rectangle.jpg is in your current directory
% Read the image using the function
[X_gray, X_red, X_green, X_blue] = readImage('rectangle.jpg');
% Display the grayscale version
figure;
subplot(1,2,1);
imagesc(uint8(X_gray));
colormap(gca, 'gray');
title('Grayscale Version');
axis image;
% Reconstruct the color image from separate channels
[m, n] = size(X_gray);
X_color = zeros(m, n, 3);
X_color(:,:,1) = X_red;
X_color(:,:,2) = X_green;
X_color(:,:,3) = X_blue;
% Display the color version
subplot(1,2,2);
imagesc(uint8(X_color));
title('Color Version');
axis image;
```



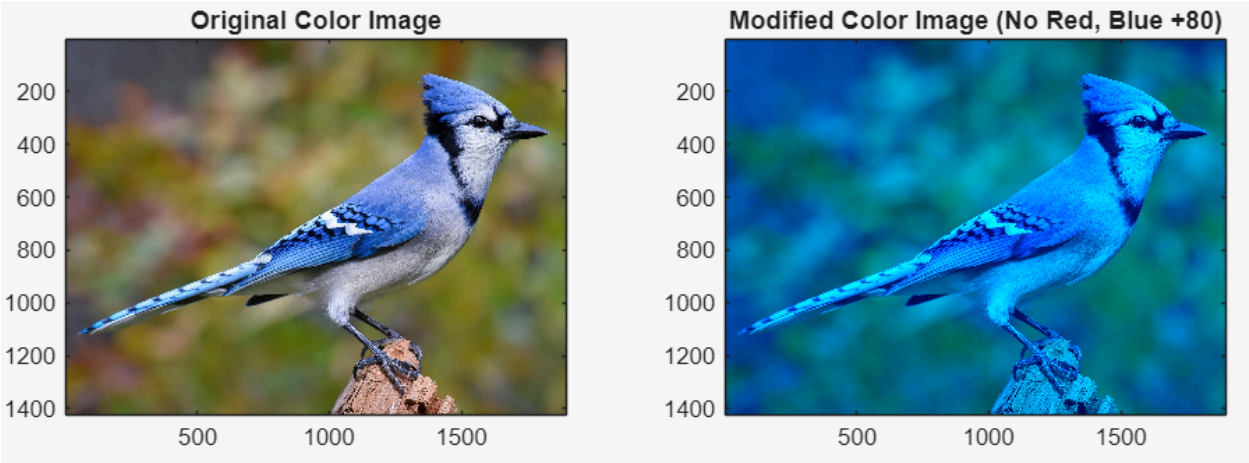

```
% Question 2: Increase exposure of grayscale image
function [X_gray, X_red, X_green, X_blue] = readImage(filename)
   % Read the image file
   X_int = imread(filename);

   % Convert to double format for mathematical operations
   X_double = double(X_int);

   % Extract individual color channels
   X_red = X_double(:,:,1);
   X_green = X_double(:,:,2);
   X_blue = X_double(:,:,3);

   % Create grayscale image by averaging the three color channels
   X_gray = X_double(:,:,1)/3.0 + X_double(:,:,2)/3.0 + X_double(:,:,3)/3.0;
end
% Read the image using the function from Question 1
[X_gray, X_red, X_green, X_blue] = readImage('rectangle.jpg');
% Increase exposure by adding a constant value to all pixel intensities
% Adding 80 units will brighten the image significantly
exposure_increase = 80;
X_gray_bright = X_gray + exposure_increase;
% Alternative method: Multiply by a factor greater than 1
% X_gray_bright = X_gray * 1.5;  % Increases by 50%
% Make sure values don't exceed 255 (the maximum intensity)
X_gray_bright(X_gray_bright > 255) = 255;
% Display original and increased exposure side by side
figure;
subplot(1,2,1);
imagesc(uint8(X_gray));
colormap(gca, 'gray');
title('Original Grayscale Image');
axis image;
colorbar;
```

```
subplot(1,2,2);
imagesc(uint8(X_gray_bright));
colormap(gca, 'gray');
title('Increased Exposure (Whited Out)');
axis image;
colorbar;
% Display the difference
fprintf('Original intensity range: [%.1f, %.1f]\n', min(X_gray(:)),
max(X_gray(:)));
fprintf('Increased exposure range: [%.1f, %.1f]\n', min(X_gray_bright(:)),
max(X_gray_bright(:)));
```




```
% Question 3: Change colors in the color image
% Remove red, keep green unchanged, increase blue by 80 units
function [X_gray, X_red, X_green, X_blue] = readImage(filename)
    % Read the image file
    X_int = imread(filename);

    % Convert to double format for mathematical operations
    X_double = double(X_int);

    % Extract individual color channels
    X_red = X_double(:,:,1);
    X_green = X_double(:,:,2);
    X_blue = X_double(:,:,3);

    % Create grayscale image by averaging the three color channels
    X_gray = X_double(:,:,1)/3.0 + X_double(:,:,2)/3.0 + X_double(:,:,3)/3.0;
end
% Read the image using the function from Question 1
[X_gray, X_red, X_green, X_blue] = readImage('rectangle.jpg');
% Manipulate the color channels:
% 1. Remove all red by multiplying by 0
X_red_modified = X_red * 0;
% 2. Leave green unchanged
X_green_modified = X_green;
% 3. Increase blue intensity by 80 units
```

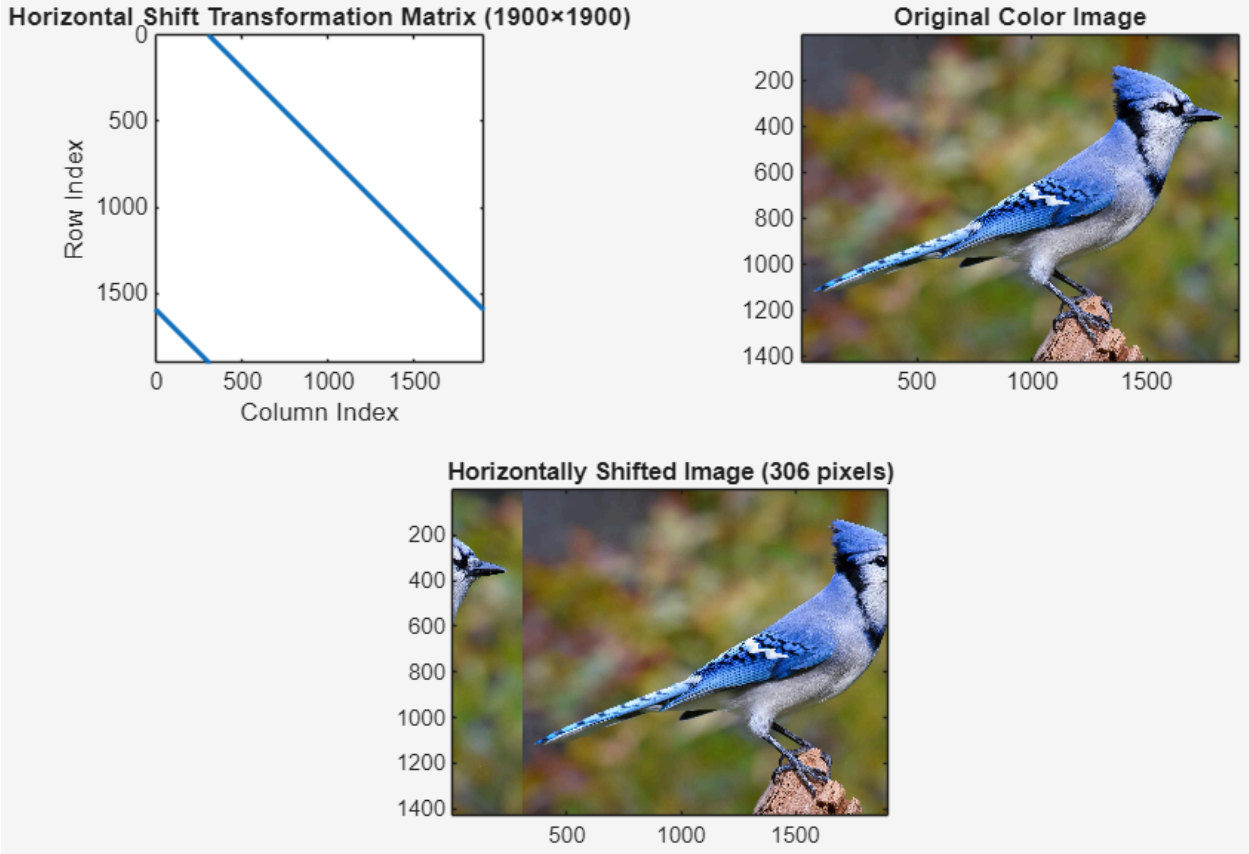
```
X_blue_modified = X_blue + 80;
% Make sure blue values don't exceed 255
X_blue_modified(X_blue_modified > 255) = 255;
% Reconstruct the original color image
[m, n] = size(X_gray);
X_color_original = zeros(m, n, 3);
X_color_original(:,:,1) = X_red;
X_color_original(:,:,2) = X_green;
X_color_original(:,:,3) = X_blue;
% Reconstruct the modified color image
X_color_modified = zeros(m, n, 3);
X_color_modified(:,:,1) = X_red_modified;
X_color_modified(:,:,2) = X_green_modified;
X_color_modified(:,:,3) = X_blue_modified;
% Display original and modified color images side by side
figure;
subplot(1,2,1);
imagesc(uint8(X_color_original));
title('Original Color Image');
axis image;
subplot(1,2,2);
imagesc(uint8(X_color_modified));
title('Modified Color Image (No Red, Blue +80)');
axis image;
% Display information about the changes
fprintf('Red channel: All values set to 0\n');
fprintf('Green channel: Unchanged\n');
fprintf('Blue channel: Increased by 80 units (capped at 255)\n');
```




```
% Question 5: Horizontal shift of 306 pixels to rectangle.jpg (color version)
% For horizontal shifts, we multiply on the RIGHT with a transformation
% matrix, refer to task 514 for my reasoning on that!!
function [X_gray, X_red, X_green, X_blue] = readImage(filename)
   % Read the image file
   X_int = imread(filename);
```

```
    % Convert to double format for mathematical operations
    X_double = double(X_int);

    % Extract individual color channels
    X_red = X_double(:,:,1);
    X_green = X_double(:,:,2);
    X_blue = X_double(:,:,3);

    % Create grayscale image by averaging the three color channels
    X_gray = X_double(:,:,1)/3.0 + X_double(:,:,2)/3.0 + X_double(:,:,3)/3.0;
end
% Read the image using the function from Question 1
[X_gray, X_red, X_green, X_blue] = readImage('rectangle.jpg');
% Get dimensions
[m, n] = size(X_gray);
% For horizontal shift, we need to transform the COLUMNS
% Start with an n×n identity matrix (since we have n columns)
I_R = eye(n);
% Horizontal shift of 306 pixels (shift columns to the right)
r = 306;
% Create transformation matrix T_horizontal
T_horizontal = zeros(n);
% Fill in the first r columns of T with the last r columns of I_R
% (wraps the rightmost columns around to the left)
T_horizontal(:, 1:r) = I_R(:, n-(r-1):n);
% Fill in the rest of T with the first part of I_R
T_horizontal(:, r+1:n) = I_R(:, 1:n-r);
% Apply horizontal shift to each color channel
% Multiply on the RIGHT for horizontal (column) transformation
X_red_shifted = X_red * T_horizontal;
X_green_shifted = X_green * T_horizontal;
X_blue_shifted = X_blue * T_horizontal;
% Reconstruct the original and shifted color images
X_color_original = zeros(m, n, 3);
X_color_original(:,:,1) = X_red;
X_color_original(:,:,2) = X_green;
X_color_original(:,:,3) = X_blue;
X_color_shifted = zeros(m, n, 3);
X_color_shifted(:,:,1) = X_red_shifted;
X_color_shifted(:,:,2) = X_green_shifted;
X_color_shifted(:,:,3) = X_blue_shifted;
% Display the transformation matrix using spy()
figure;
subplot(2,2,1);
spy(T_horizontal);
title(sprintf('Horizontal Shift Transformation Matrix (%d×%d)', n, n));
xlabel('Column Index');
ylabel('Row Index');
```

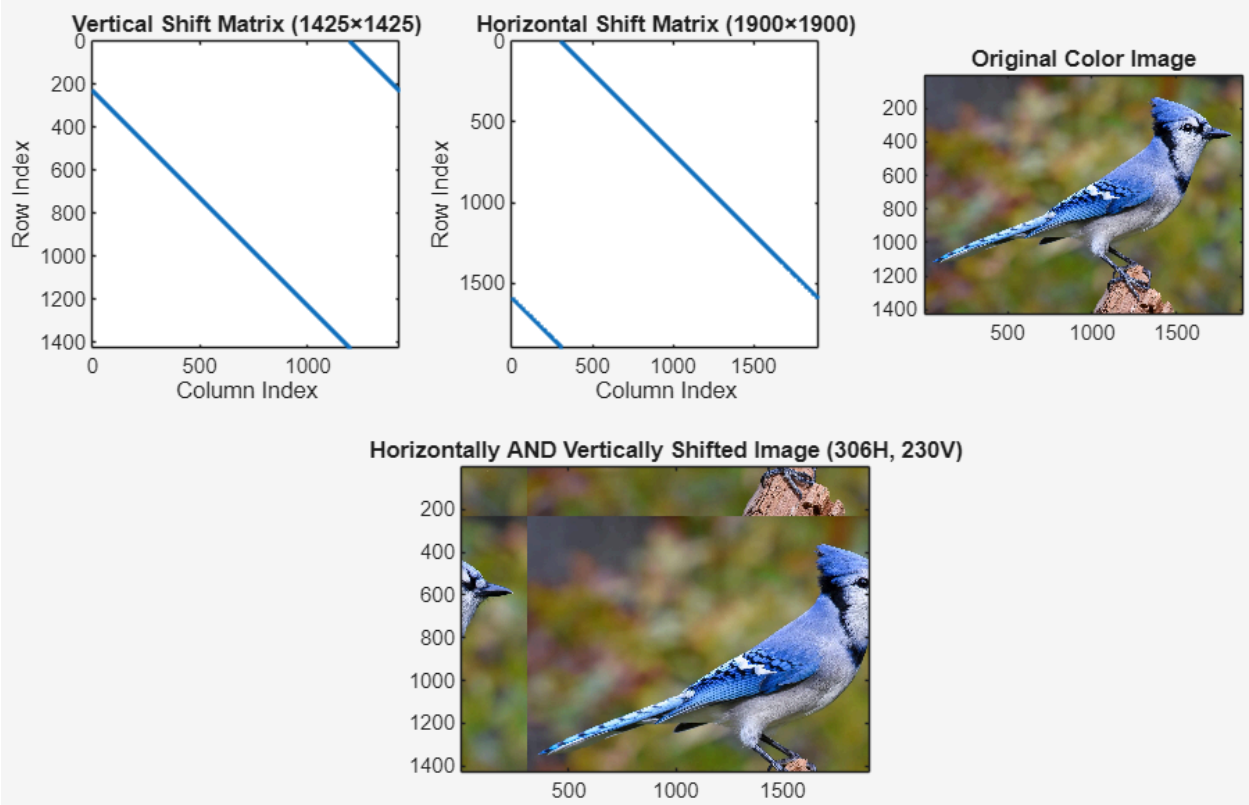
```
%To be honest I'm confused on exactly what they're asking here..? But I
%just made a matrix graph to represent, I dont know if they want the matrix
%in like handrwitten matrix form...
% Display original image
subplot(2,2,2);
imagesc(uint8(X_color_original));
title('Original Color Image');
axis image;
% Display shifted image
subplot(2,2,[3,4]);
imagesc(uint8(X_color_shifted));
title('Horizontally Shifted Image (306 pixels)');
axis image;
fprintf('Image dimensions: %d rows × %d columns\n', m, n);
fprintf('Horizontal shift: %d pixels to the right (with wrapping)\n', r);
fprintf('Transformation matrix dimensions: %d × %d\n', n, n);
```




```
% Question 6: Horizontal AND Vertical shift (306 pixels horizontal, 230 pixels
vertical)
% Horizontal shift: multiply on the RIGHT with n×n matrix
% Vertical shift: multiply on the LEFT with m×m matrix
function [X_gray, X_red, X_green, X_blue] = readImage(filename)
   % Read the image file
   X_int = imread(filename);
```

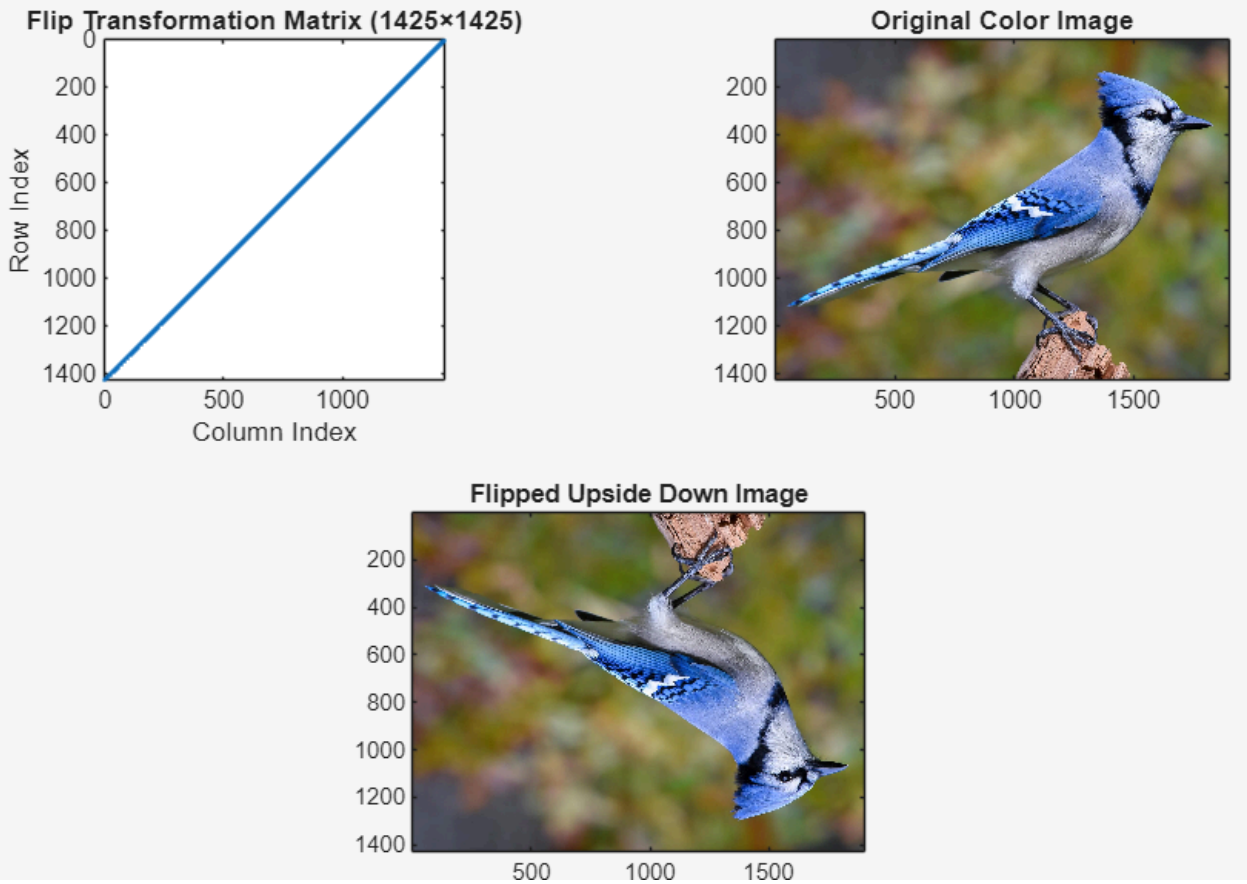
```
    % Convert to double format for mathematical operations
    X_double = double(X_int);

    % Extract individual color channels
    X_red = X_double(:,:,1);
    X_green = X_double(:,:,2);
    X_blue = X_double(:,:,3);

    % Create grayscale image by averaging the three color channels
    X_gray = X_double(:,:,1)/3.0 + X_double(:,:,2)/3.0 + X_double(:,:,3)/3.0;
end
% Read the image using the function from Question 1
[X_gray, X_red, X_green, X_blue] = readImage('rectangle.jpg');
% Get dimensions
[m, n] = size(X_gray);
% HORIZONTAL SHIFT MATRIX (n×n) - shifts columns, multiply on RIGHT
r_horizontal = 306;
I_R = eye(n);
T_horizontal = zeros(n);
% Fill in the first r_horizontal columns with the last r_horizontal columns of
I_R
T_horizontal(:, 1:r_horizontal) = I_R(:, n-(r_horizontal-1):n);
% Fill in the rest with the first part of I_R
T_horizontal(:, r_horizontal+1:n) = I_R(:, 1:n-r_horizontal);
% VERTICAL SHIFT MATRIX (m×m) - shifts rows, multiply on LEFT
r_vertical = 230;
I_L = eye(m);
T_vertical = zeros(m);
% Fill in the first r_vertical rows with the last r_vertical rows of I_L
T_vertical(1:r_vertical, :) = I_L(m-(r_vertical-1):m, :);
% Fill in the rest with the first part of I_L
T_vertical(r_vertical+1:m, :) = I_L(1:m-r_vertical, :);
% Apply BOTH transformations to each color channel
% Order: T_vertical * X * T_horizontal (vertical shift on left, horizontal on
right)
X_red_shifted = T_vertical * X_red * T_horizontal;
X_green_shifted = T_vertical * X_green * T_horizontal;
X_blue_shifted = T_vertical * X_blue * T_horizontal;
% Reconstruct the original and shifted color images
X_color_original = zeros(m, n, 3);
X_color_original(:,:,1) = X_red;
X_color_original(:,:,2) = X_green;
X_color_original(:,:,3) = X_blue;
X_color_shifted = zeros(m, n, 3);
X_color_shifted(:,:,1) = X_red_shifted;
X_color_shifted(:,:,2) = X_green_shifted;
X_color_shifted(:,:,3) = X_blue_shifted;
% Display the transformation matrices using spy()
```

```
figure;
subplot(2,3,1);
spy(T_vertical);
title(sprintf('Vertical Shift Matrix (%d×%d)', m, m));
xlabel('Column Index');
ylabel('Row Index');
subplot(2,3,2);
spy(T_horizontal);
title(sprintf('Horizontal Shift Matrix (%d×%d)', n, n));
xlabel('Column Index');
ylabel('Row Index');
% Display original image
subplot(2,3,3);
imagesc(uint8(X_color_original));
title('Original Color Image');
axis image;
% Display shifted image (spans bottom row)
subplot(2,3,[4,5,6]);
imagesc(uint8(X_color_shifted));
title('Horizontally AND Vertically Shifted Image (306H, 230V)');
axis image;
fprintf('Image dimensions: %d rows × %d columns\n', m, n);
fprintf('Horizontal shift: %d pixels (right, wrapping)\n', r_horizontal);
fprintf('Vertical shift: %d pixels (down, wrapping)\n', r_vertical);
fprintf('Vertical transformation matrix: %d × %d\n', m, m);
fprintf('Horizontal transformation matrix: %d × %d\n', n, n);
fprintf('Combined operation: T_vertical * X * T_horizontal\n');
```



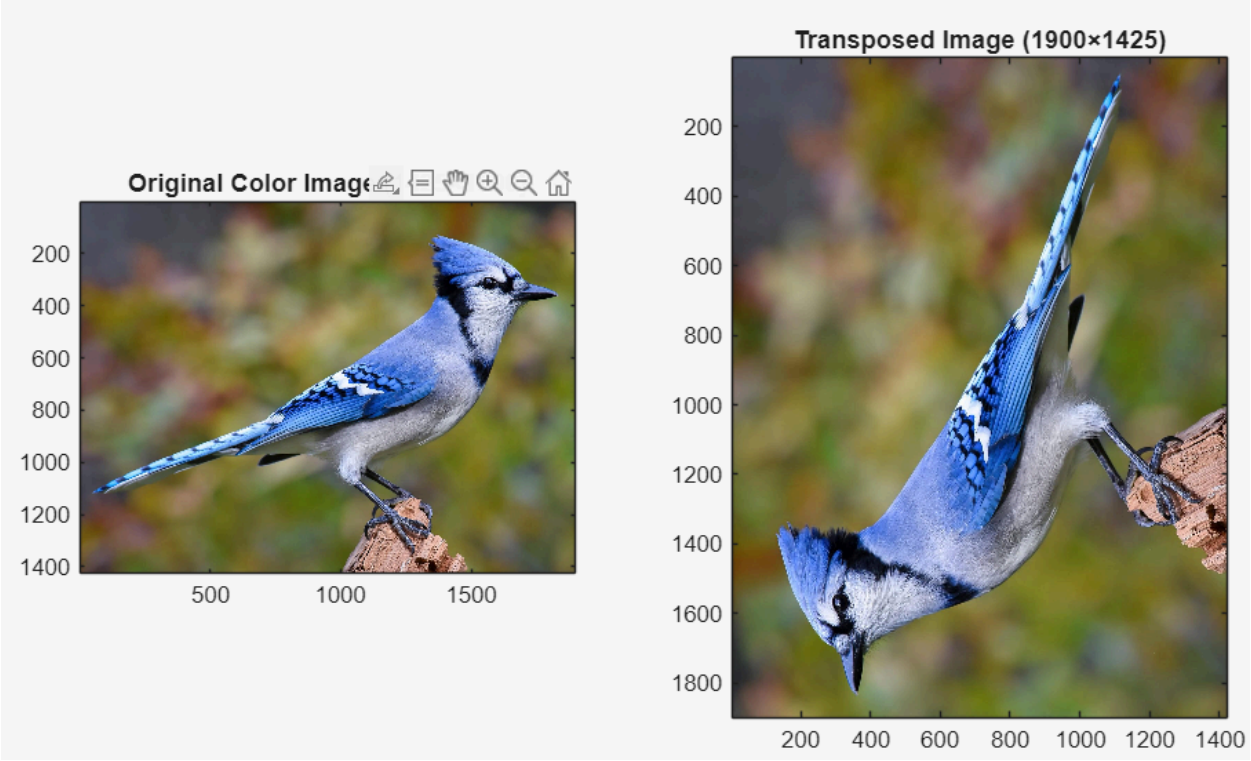

```
% Question 7: Flip image upside down using transformation matrices
function [X_gray, X_red, X_green, X_blue] = readImage(filename)
    % Read the image file
    X_int = imread(filename);

    % Convert to double format for mathematical operations
    X_double = double(X_int);

    % Extract individual color channels
    X_red = X_double(:,:,1);
    X_green = X_double(:,:,2);
    X_blue = X_double(:,:,3);

    % Create grayscale image by averaging the three color channels
    X_gray = X_double(:,:,1)/3.0 + X_double(:,:,2)/3.0 + X_double(:,:,3)/3.0;
end
% Read the image using the function from Question 1
[X_gray, X_red, X_green, X_blue] = readImage('rectangle.jpg');
% Get dimensions
[m, n] = size(X_gray);
% Create the flip transformation matrix (m×m)
% To flip upside down, we need to reverse the order of rows
% Start with identity matrix and reverse its rows
I_flip = eye(m);
% Create flip matrix by reversing the rows of the identity matrix
```

```
% Row 1 should become row m, row 2 should become row m-1, etc.
T_flip = zeros(m);
for i = 1:m
   T_flip(i, :) = I_flip(m - i + 1, :);
end
% Alternative construction (more explicit):
% T_flip = I_flip(m:-1:1, :);  % reverses rows of identity
% Apply flip transformation to each color channel
% Multiply on the LEFT for row (vertical) transformation
X_red_flipped = T_flip * X_red;
X_green_flipped = T_flip * X_green;
X_blue_flipped = T_flip * X_blue;
% Reconstruct the original and flipped color images
X_color_original = zeros(m, n, 3);
X_color_original(:,:,1) = X_red;
X_color_original(:,:,2) = X_green;
X_color_original(:,:,3) = X_blue;
X_color_flipped = zeros(m, n, 3);
X_color_flipped(:,:,1) = X_red_flipped;
X_color_flipped(:,:,2) = X_green_flipped;
X_color_flipped(:,:,3) = X_blue_flipped;
% Display the transformation matrix using spy()
figure;
subplot(2,2,1);
spy(T_flip);
title(sprintf('Flip Transformation Matrix (%d×%d)', m, m));
xlabel('Column Index');
ylabel('Row Index');
% Display original image
subplot(2,2,2);
imagesc(uint8(X_color_original));
title('Original Color Image');
axis image;
% Display flipped image
subplot(2,2,[3,4]);
imagesc(uint8(X_color_flipped));
title('Flipped Upside Down Image');
axis image;
fprintf('Image dimensions: %d rows × %d columns\n', m, n);
fprintf('Flip transformation matrix dimensions: %d × %d\n', m, m);
fprintf('Operation: T_flip * X (multiply on LEFT to affect rows)\n');
fprintf('Effect: Row 1 → Row %d, Row %d → Row 1, etc.\n', m, m);
```




```
% Task 5.18
function [X_gray, X_red, X_green, X_blue] = readImage(filename)
    % Read the image file
    X_int = imread(filename);

    % Convert to double format for mathematical operations
    X_double = double(X_int);

    % Extract individual color channels
    X_red = X_double(:,:,1);
    X_green = X_double(:,:,2);
    X_blue = X_double(:,:,3);

    % Create grayscale image by averaging the three color channels
    X_gray = X_double(:,:,1)/3.0 + X_double(:,:,2)/3.0 + X_double(:,:,3)/3.0;
end
% Read the image using the function from Question 1
[X_gray, X_red, X_green, X_blue] = readImage('rectangle.jpg');
% Get dimensions
[m, n] = size(X_gray);
% Transpose each color channel
X_red_transposed = X_red';
```

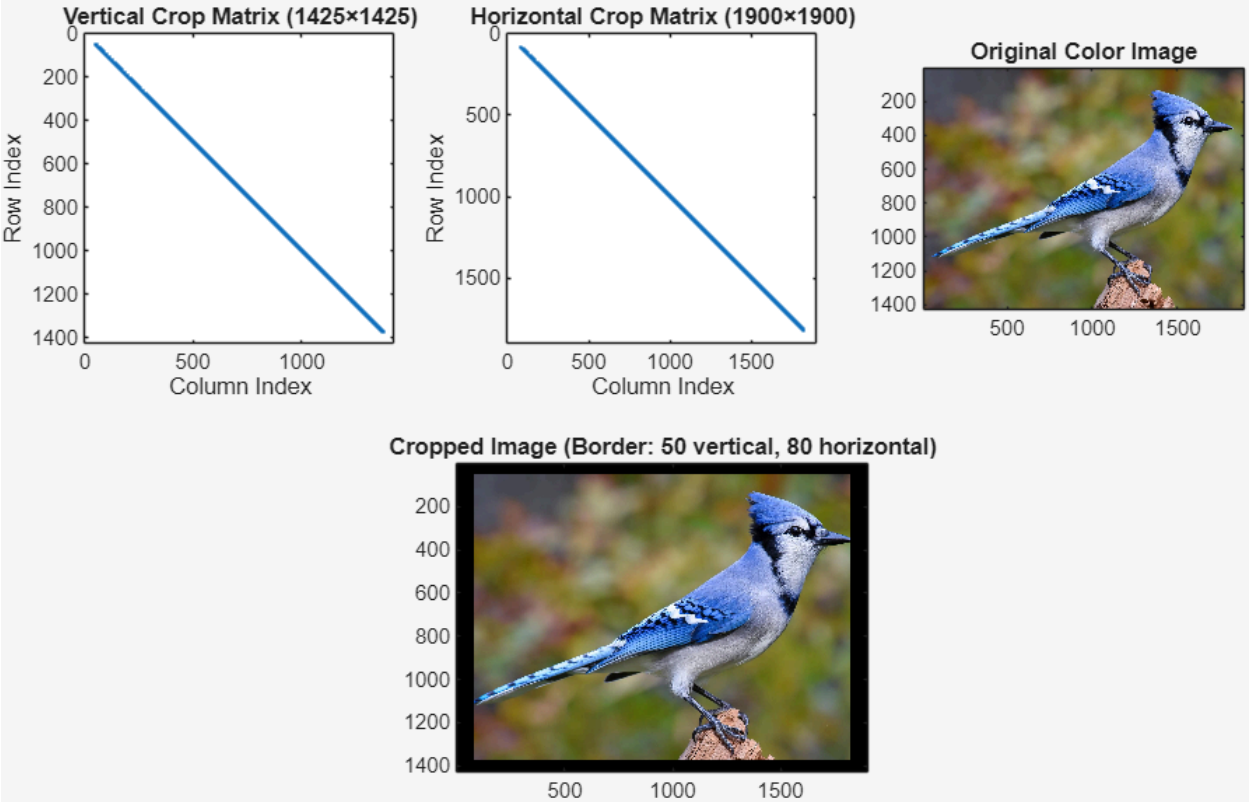
```
X_green_transposed = X_green';
X_blue_transposed = X_blue';
% Get new dimensions after transpose
[m_new, n_new] = size(X_red_transposed);
% Reconstruct the original color image
X_color_original = zeros(m, n, 3);
X_color_original(:,:,1) = X_red;
X_color_original(:,:,2) = X_green;
X_color_original(:,:,3) = X_blue;
% Reconstruct the transposed color image
X_color_transposed = zeros(m_new, n_new, 3);
X_color_transposed(:,:,1) = X_red_transposed;
X_color_transposed(:,:,2) = X_green_transposed;
X_color_transposed(:,:,3) = X_blue_transposed;
% Display original and transposed images
figure;
subplot(1,2,1);
imagesc(uint8(X_color_original));
title(sprintf('Original Color Image (%d×%d)', m, n));
axis image;
subplot(1,2,2);
imagesc(uint8(X_color_transposed));
title(sprintf('Transposed Image (%d×%d)', m_new, n_new));
axis image;
fprintf('Original image dimensions: %d rows × %d columns\n', m, n);
fprintf('Transposed image dimensions: %d rows × %d columns\n', m_new, n_new);
fprintf('\nWhat transposing does:\n');
fprintf('- Swaps rows and columns\n');
fprintf('- Element at position (i,j) moves to position (j,i)\n');
fprintf('- Reflects the image across the main diagonal (top-left to
bottom-right)\n');
fprintf('- Result: Image is rotated 90° counterclockwise AND flipped
horizontally\n');
fprintf('  (equivalent to rotating 90° clockwise and flipping vertically)\n');
```




```
% Question 9: Crop the image by adding a black border
function [X_gray, X_red, X_green, X_blue] = readImage(filename)
    % Read the image file
    X_int = imread(filename);

    % Convert to double format for mathematical operations
    X_double = double(X_int);

    % Extract individual color channels
    X_red = X_double(:,:,1);
    X_green = X_double(:,:,2);
    X_blue = X_double(:,:,3);

    % Create grayscale image by averaging the three color channels
    X_gray = X_double(:,:,1)/3.0 + X_double(:,:,2)/3.0 + X_double(:,:,3)/3.0;
end
% Read the image using the function from Question 1
[X_gray, X_red, X_green, X_blue] = readImage('rectangle.jpg');
% Get dimensions
[m, n] = size(X_gray);
% Define border sizes (make them obvious)
border_vertical = 50;    % pixels to crop from top and bottom
border_horizontal = 80;  % pixels to crop from left and right
% Create VERTICAL cropping matrix (m×m)
% This will zero out the top and bottom rows
T_vert = eye(m);
```

```
% Zero out the first border_vertical rows (top border)
T_vert(1:border_vertical, 1:border_vertical) = 0;
% Zero out the last border_vertical rows (bottom border)
T_vert(m-border_vertical+1:m, m-border_vertical+1:m) = 0;
% Create HORIZONTAL cropping matrix (n×n)
% This will zero out the left and right columns
T_horiz = eye(n);
% Zero out the first border_horizontal columns (left border)
T_horiz(1:border_horizontal, 1:border_horizontal) = 0;
% Zero out the last border_horizontal columns (right border)
T_horiz(n-border_horizontal+1:n, n-border_horizontal+1:n) = 0;
% Apply cropping to each color channel
% Use both transformations: T_vert * X * T_horiz
X_red_cropped = T_vert * X_red * T_horiz;
X_green_cropped = T_vert * X_green * T_horiz;
X_blue_cropped = T_vert * X_blue * T_horiz;
% Reconstruct the original and cropped color images
X_color_original = zeros(m, n, 3);
X_color_original(:,:,1) = X_red;
X_color_original(:,:,2) = X_green;
X_color_original(:,:,3) = X_blue;
X_color_cropped = zeros(m, n, 3);
X_color_cropped(:,:,1) = X_red_cropped;
X_color_cropped(:,:,2) = X_green_cropped;
X_color_cropped(:,:,3) = X_blue_cropped;
% Display the transformation matrices using spy()
figure;
subplot(2,3,1);
spy(T_vert);
title(sprintf('Vertical Crop Matrix (%d×%d)', m, m));
xlabel('Column Index');
ylabel('Row Index');
subplot(2,3,2);
spy(T_horiz);
title(sprintf('Horizontal Crop Matrix (%d×%d)', n, n));
xlabel('Column Index');
ylabel('Row Index');
% Display original image
subplot(2,3,3);
imagesc(uint8(X_color_original));
title('Original Color Image');
axis image;
% Display cropped image (spans bottom row)
subplot(2,3,[4,5,6]);
imagesc(uint8(X_color_cropped));
title(sprintf('Cropped Image (Border: %d vertical, %d horizontal)', ...
   border_vertical, border_horizontal));
axis image;
fprintf('Image dimensions: %d rows × %d columns\n', m, n);
```

```
fprintf('Vertical border (top & bottom): %d pixels\n', border_vertical);
fprintf('Horizontal border (left & right): %d pixels\n', border_horizontal);
fprintf('Vertical crop matrix: %d × %d\n', m, m);
fprintf('Horizontal crop matrix: %d × %d\n', n, n);
fprintf('Operation: T_vert * X * T_horiz\n');
fprintf('Effect: Creates black borders by zeroing out edge pixels\n');
```



Task Set 5.2

```
%% Task 5.2.10 (Matrix S of any size)
function S = creatematrix(n)
   %Inputs: n (number) the size of the nxn square matrix
   %Outputs: S (matrix) produces the requested matrix size

   % Initialize the matrix
   S = zeros(n);

   % Use a single loop structure for clarity and efficiency in MATLAB:
   for i = 1:n
       for j = 1:n
           S(i,j) = sqrt(2/n) * sin(pi*(i-0.5)*(j-0.5)/n);
       end
   end
end
```

```
%% Task 5.2.11 (Verify that S=S^-1)
S1 = creatematrix(5);
A = S1*S1;
disp("The result of S*S is")
disp(A)
%% Task 5.2.12 (Reverse DST)
% Because S=S^-1, SYS=x_g
% To prove, Y=S(x_g)S so SYS=S^2(x_g)S^2=I(x_g)I=x_g
%% Task 5.2.13 (JPEG-like Compression on square.jpg)
x=imread('../square.jpg');
x_double=double(x);
x_r=x_double(:,:,1);
x_gr=x_double(:,:,2);
x_b=x_double(:,:,3);
n=length(x);
S = zeros(n);
for i = 1:n
   for j = 1:n
       S(i,j) = sqrt(2/n) * sin(pi*(i-0.5)*(j-0.5)/n);
   end
end
p=0.05;
for k=1:8
   p=1/2^(k-1);
   y_r=S*x_r*S;
   for i = 1:n
       for j = 1:n
           if (i+j > p*2*n)
           y_r(i,j) = 0;
           end
       end
   end
   x_r=S*y_r*S;

   y_gr=S*x_gr*S;
   for i = 1:n
       for j = 1:n
           if (i+j > p*2*n)
           y_gr(i,j) = 0;
           end
       end
   end
   x_gr=S*y_gr*S;

   y_b=S*x_b*S;
   for i = 1:n
       for j = 1:n
           if (i+j > p*2*n)
           y_b(i,j) = 0;
```

```
            end
        end
    end

    x_b=S*y_b*S;

    x_comp=cat(3,x_r,x_gr,x_b);
    imwrite(uint8(x_comp), sprintf('square_compressed_p_%.2f.jpg', p))
    y_trans=cat(3,y_r,y_gr,y_b);
    nnz_list(k,1)=p;
    nnz_list(k,2)=nnz(y_trans);
    size_list(k,1)=p;
    size_list(k,2) = dir(sprintf('square_compressed_p_%.2f.jpg', p)).bytes;
    comp_ratio = (size_list(1,2) / size_list(k, 2));
    size_list(k, 3) = comp_ratio;
end
%% Task 5.2.14 (Best p value)
nnz_table = array2table(nnz_list, 'VariableNames', {'p_value',
'non_zero_elements'});
writetable(nnz_table, 'nnz_table.csv');
% I think the best value of p is 0.125 because it retains almost
% identical visual quality to the original image with
% about 1/30 of the data.
%% Task 5.2.15 (Compression ratios)
size_table = array2table(size_list, 'VariableNames', {'p_value',
'size_in_bytes', 'compression_ratio'});
writetable(size_table, 'size_table.csv');
% p=0.0625 gives the largest CR (1.3222) while still keeping decent
% image quality.
```