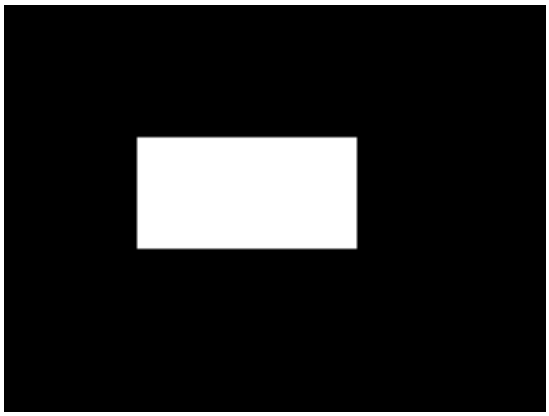
CSE 5524 - Homework #4 **9/20/2016**

Chandrasekar Swaminathan (swaminathan.42)

1) Use the **regionprops** Matlab function on boxIm1.bmp (provided on the class website) to compute its 'Area', 'Centroid', and 'BoundingBox'. Plot/mark/identify the centroid and bounding box on the image.

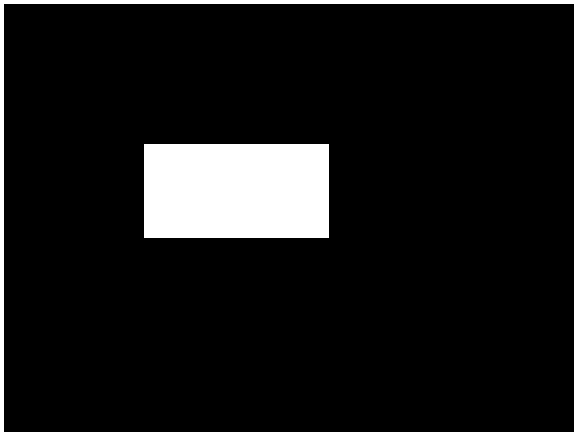
The centroid is at the center of rectangle where the pixel values are non zero. The entire area of the rectangle is **3321.**

2) Write a function to compute the seven **similitude** moment shape descriptors. Test and compare results on the rectangle box images ‘boxIm[1-4].bmp’ on the website. How do they change across the box images?

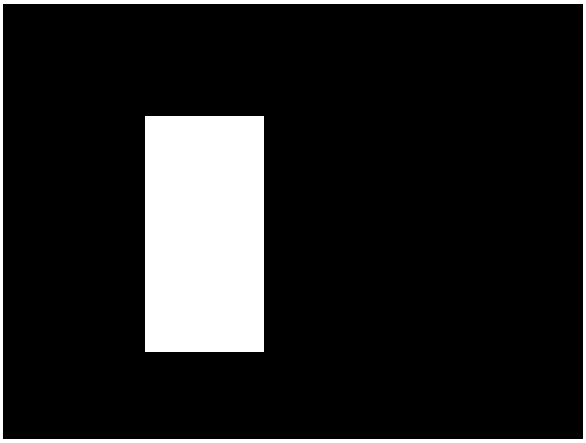


boxIm1.bmp [0.0011 0.0001 -0.0005 -0.0000 0.0006 0.0000 -0.0000]

boxIm2.bmp [0.0011 0.0001 -0.0005 -0.0000 0.0006 0.0000 -0.0000]



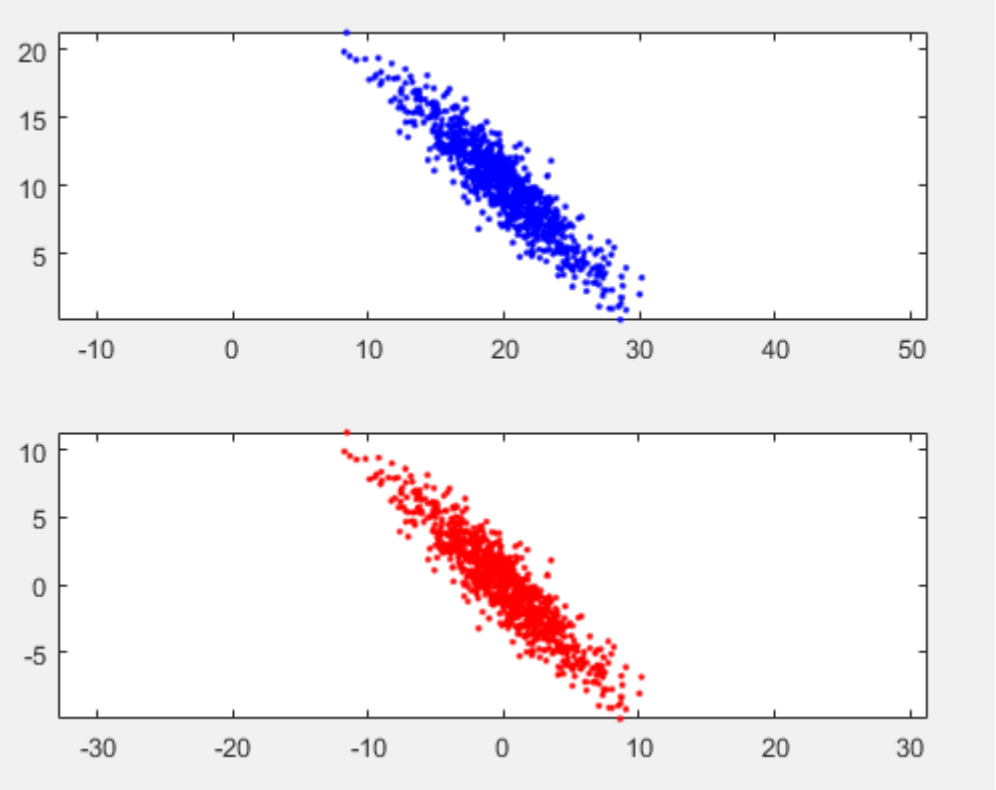
boxIm3.bmp [0.0011 0.0001 -0.0005 -0.0000 0.0006 0.0000 -0.0000]



boxIm4.bmp 1.0e-03 * [0.2834 -0.0067 -0.1181 0.0031 0.7636 -0.0083 0.0223]

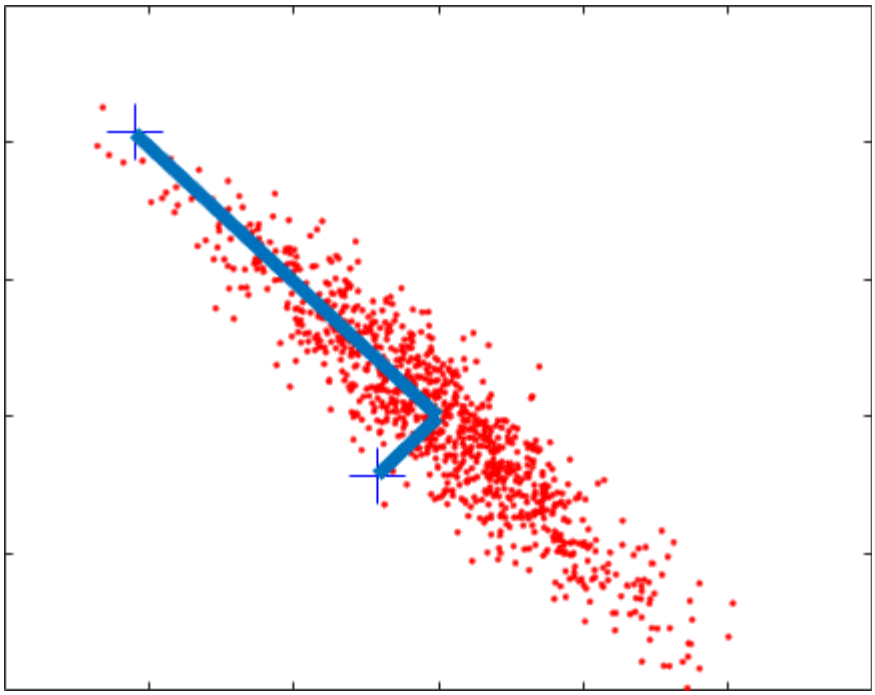
We can see from images **boxIm1.bmp, boxIm2.bmp, boxIm3.bmp** that the similitude moments are not affected by scaling or translation operations. However, the similitude moments do change if the object undergoes any rotational transformation as seen in image **boxIm4.bmp**, where the computed similitude moments are different from the similitude moments of the first three images.

3) Using the datafile (eigdata.txt) provided on the WWW site, perform the following MATLAB command.



The plot on the top represents the data present in the eigdata.txt file. The plot on the bottom represents the same data centered around origin (after subtracting mean value from the data)

4) Compute the eigenvalues (V) and eigenvectors (U) of the data (stored in Y) using the function **eig()** in Matlab (recall that you use either the covariance matrix or the Inverse-covariance matrix of the data – see class notes). Plot the mean-subtracted data Y and the 2-D Gaussian ellipse axes for given the eigenvectors in U (you can use the plot command in Matlab for this). Use the eigenvalues in V to give the appropriate 3σ (standard deviation, not variance!) length to each axis (did you compute the eigenvalues from the covariance or inverse covariance of Y? The eigenvalues will be related but different! See class notes)



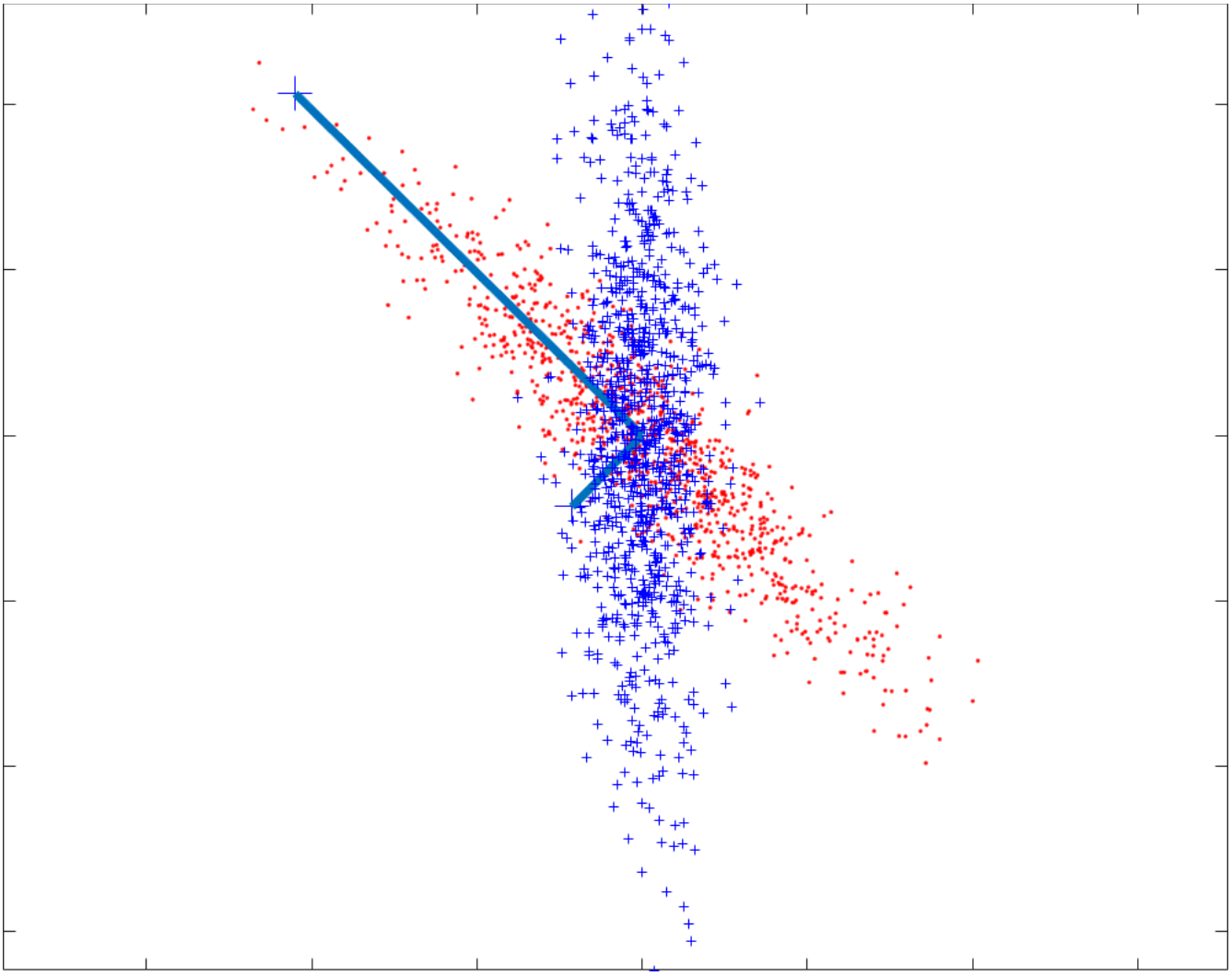
The **eig()** function in MATLAB was used to compute the eigenvectors and the eigenvalues as follows:

```
eigen_vectors =
  -0.7126   -0.7016
   0.7016   -0.7126

eigen_values =
  24.1385         0
        0    1.0142
```

We can see from the figure that the eigen vectors are orthogonal to each other. The primary eigen vector is in the direction of the largest variance, the secondary eigen vector is in the direction of the next largest variance.

5) Rotate Y using the eigenvectors to be uncorrelated (i.e., project data Y onto the eigenvectors – see class slides). Plot the results.



The eigenvector matrix was multiplied with the mean centered data matrix in order to rotate it. In the above image, the collection of points with line overlaid on top represents the original data points. The collection of points in the vertical direction represents the rotated, uncorrelated data. We can see that most of the variance is along just the vertical axis, there is very little variance in the data along the the horizontal axis.f

Code

```
% Chandrasekar Swaminathan
% swaminathan.42@osu.edu
% CSE5524 - HW4
% 9/20/2016
```

HW4.m

```
% Problem 1
mkdir('output');
image = double(imread(fullfile('input','boxIm1.bmp')));
area = computeArea(image);
[topLeft_X, topLeft_Y, length, height] = computeBoundingBox(image);
[centroidX, centroidY] = computeCentroid(image);
imagesc(image);
colormap('gray');
hold on;
plot(centroidX, centroidY, 'r.');
rectangle ('Position', [topLeft_X topLeft_Y length height], 'EdgeColor', 'r', 'LineWidth', 3);
text(10, 10, sprintf('Area: %g', area), 'Color', 'Yellow', 'BackgroundColor', 'Black', 'EdgeColor',
'White', 'FontSize', 14);
text(centroidX - 10, centroidY + 10, sprintf('Centroid (%g, %g)', centroidX, centroidY), 'EdgeColor',
'White', 'BackgroundColor', 'Black', 'Color', 'Yellow');
save_current_frame(fullfile('output', 'problem1.png'));
hold off;
pause;


% Problem 2
close all; clear;
for n=1:4
   image = double(imread(fullfile('input',sprintf('boxIm%d.bmp', n))));
   simMoments = similitudeMoments(image);
         imagesc(image);
         title(sprintf('boxIm%d.bmp', n));
         colormap('gray');
         text(10, 10, sprintf('Similitude Moments: %s', mat2str(simMoments, 2)), 'Color','Yellow',
'BackgroundColor','Black', 'EdgeColor', 'White');
   pause;
   fprintf('boxIm%d.bmp', n);
   disp(simMoments);
   save_current_frame(fullfile('output', sprintf('similiBoxIm%d.bmp', n)));
end

% Problem 3
clear; close all;
```

```
load input\eigdata.txt;
X = eigdata;
subplot(2, 1, 1);
plot(X(:,1), X(:,2), 'b.');
axis('equal');
m = mean(X);
Y = X-ones(size(X,1), 1)*m;
subplot(2,1,2);
plot(Y(:,1), Y(:,2), 'r.');
axis('equal');
pause;

% Problem 4
% the following code assumes that the covariance matrix is 2x2
close all;
covariance = cov(Y);
[eigen_vectors, eigen_values] = eig(covariance);
[eigen_vectors, eigen_values] = sort_descending(eigen_vectors, eigen_values);
C = 9; % (no-of-std-dev)^2
major_axis_length = sqrt(C*eigen_values(1,1));
scaled_major_axis = eigen_vectors(:, 1) * major_axis_length;
minor_axis_length = sqrt(C*eigen_values(2,2));
scaled_minor_axis = eigen_vectors(:, 2) * minor_axis_length;
plot(Y(:, 1), Y(:, 2), 'r.');
hold on;
plot(scaled_major_axis(1,1), scaled_major_axis(2,1), 'b+', 'MarkerSize', 20);
plot(scaled_minor_axis(1,1), scaled_minor_axis(2,1), 'b+', 'MarkerSize', 20);
line([ 0; scaled_major_axis(1,1)], [0; scaled_major_axis(2,1)], 'LineWidth', 5);
line([ 0; scaled_minor_axis(1,1)], [0; scaled_minor_axis(2,1)], 'LineWidth', 5);
save_current_frame(fullfile('output', 'axes.png'));
pause;

% Problem 5
rotated_data = eigen_vectors * Y';
rotated_data = rotated_data';
figure; plot(rotated_data(:, 1), rotated_data(:, 2), 'b+');
axis('equal');
save_current_frame(fullfile('output', 'rotated-data.png'));
```

computeArea.m

```
function [area] = computeArea (image)
   image = double(image);
   allAreas = table2array(regionprops('table', image, 'Area'));
   nonZeroAreaIndexes = find(allAreas > 0);
   if(isempty(nonZeroAreaIndexes))
      area = 0;
   else
      area = allAreas(nonZeroAreaIndexes(1, 1));
   end
end
```

computeBoundingBox.m

```
function [topLeftX, topLeftY, length, height] = computeBoundingBox (image)
   image = double(image);
   tableOut = regionprops('table', image, 'BoundingBox');
   uniqueEntries = table2array(unique(tableOut));
   firstNonZeroBoundingBox = zeros(1, 4);
   for row=1:size(uniqueEntries, 1)
      if(uniqueEntries(row, 3) > 0 && uniqueEntries(row, 4) > 0)
         firstNonZeroBoundingBox(1, :) = uniqueEntries(row, :);
         break;
      end
   end
   topLeftX = firstNonZeroBoundingBox(1, 1);
   topLeftY = firstNonZeroBoundingBox(1, 2);
   length = firstNonZeroBoundingBox(1, 3);
   height = firstNonZeroBoundingBox(1, 4);
end
```

computeCentroid.m

```
function [centroidX, centroidY] = computeCentroid(img)
   centroidsTable = regionprops('table', img, 'Centroid');
   centroidsArray = table2array(centroidsTable);
   [rows, cols] = find(~isnan(centroidsArray));
   uniqueRows = unique(rows);
   centroidX = centroidsArray(uniqueRows(1), 1);
   centroidY = centroidsArray(uniqueRows(1), 2);
end
```

similitudeMoments.m

```
function [N] = similitudeMoments(boxImg)
   [centroidX, centroidY] = computeCentroid(boxImg);
   momentsToConsider = [ 0 2; 0 3; 1 1; 1 2; 2 0; 2 1; 3 0];
   zeroOrderMoment = computeMoment(boxImg, centroidX, centroidY, 0, 0);
   N = zeros (1, size(momentsToConsider, 1));
   for row = 1:size(momentsToConsider, 1)
      p = momentsToConsider(row, 1);
      q = momentsToConsider(row, 2);
      momentValue = computeMoment(boxImg, centroidX, centroidY, p ,q);
      N(1, row) = momentValue / (zeroOrderMoment^(((p+q)/2) + 1));
   end
end
```

save_current_frame.m

```
function [] = save_current_frame (outputFileName)
% saves whatever is displayed in the window to the given file. useful when
% there is dynamic text rendered on top of the image and that should also
% be saved to the file
  frame = getframe;
  imwrite(frame.cdata, outputFileName);
end
```

sort_descending.m

```
function [sorted_eigen_vectors, sorted_eigen_values] =
sort_descending(eigen_vectors,eigen_values)
   sorted_eigen_values=diag(sort(diag(eigen_values),'descend'));
%   get the original indexes of the elements before they were sorted. the
%   columns in the eigen vectors should be rearranged to match this order
   [temp, index_matrix]=sort(diag(eigen_values),'descend');
   sorted_eigen_vectors=eigen_vectors(:,index_matrix);
end
```