Assignment A9: Model Fitting

Clinton Fernandes

u1016390

06/05/2016

A9

CS 6320

1. Introduction

The aim of this assignment is to develop and study model fitting methods. Here, we wish to fit a structure to a set of tokens (say, a line to a set of points). This problem is usually referred to as fitting or grouping. For this we need to develop a set of functions that can retrieve lines from the image.

One question that can be asked is: How does addition of Gaussian noise affect the line parameters. Here, with the increase in noise (with standard deviation), sum of distance squared errors can be used as a measure to compare in case of least square fitting.

2. Method

The overall goal of the exercise is to fit a structure to a set of tokens. Different methods can be followed to give the structure (in our cases, lines). Total least squares and Hough Transform for fitting lines have been discussed below, along with the required functions.

2.1. <u>Fitting a Single Line: least squares</u>

We first assume that all the points that belong to a particular line are known (these are the scattered points from using which we have to estimate a best fit line).

The strategy for fitting lines which we use here is known as *least squares*. We can represent a line as the collection of points where ax + by + c = 0. Every line can be represented in this way, and we can think of a line as a triple of values (a, b, c). At each data point, we have (xi, yi); we decide to choose the line that best predicts the measured y coordinate for each measured x coordinate. This means we want to choose the line that minimize the sum of perpendicular distances between points and lines.

We need to minimize $\sum(ax_i + by_i + c)^2$ subjected to $a^2 + b^2 = 1$

Now using a Lagrange multiplier λ, we have a solution if,

$$\begin{pmatrix} \overline{x^2} & \overline{xy} & \overline{x} \\ \overline{xy} & \overline{y^2} & \overline{y} \\ \overline{x} & \overline{y} & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \lambda \begin{pmatrix} 2a \\ 2b \\ 0 \end{pmatrix}.$$

This means that $c = -a\overline{x} - b\overline{y}$,
and we can substitute this back
to get the eigenvalue problem

$$\begin{pmatrix} \overline{x^2} - \overline{x}\,\overline{x} & \overline{xy} - \overline{x}\,\overline{y} \\ \overline{xy} - \overline{x}\,\overline{y} & \overline{y^2} - \overline{y}\,\overline{y} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \mu \begin{pmatrix} a \\ b \end{pmatrix}$$

This is a 2D eigenvalue problem, two solutions up to scale can be obtained in closed form. The scale is obtained from the constraint that $a^2 + b^2 = 1$. The two solutions to this problem are lines at right angles; one maximizes the sum of squared distances and the other minimizes it. We need to find the solution that minimizes the sum of squared distances.

****CV_total_LS** function can be used for fitting a line by *least squares*:

Input: x and y coordinates of points

Output: 1. coefficients of best fit line ax + by + c

2. and error measure (sum of squares of distances points to line)

Algorithm:

- For the 2D eigenvalue problem represented by: $\begin{pmatrix} \overline{x^2} - \overline{x}\,\overline{x} & \overline{xy} - \overline{x}\,\overline{y} \\ \overline{xy} - \overline{x}\,\overline{y} & \overline{y^2} - \overline{y}\,\overline{y} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \mu \begin{pmatrix} a \\ b \end{pmatrix}$
- Find values of a and b, that is, the Eigen vectors corresponding to the minimum Eigen values.
- Now proceed to find c : $c = -a\overline{x} - b\overline{y}$
- Calculate error measure. $\sum(ax_i + by_i + c)^2$

A function **CS5320_plot_line has also been developed to verify the output of the CV_total_LS function.

Input is: line parameters(p) and endpoints of line {x1,x2,y1,y2}

On output: plot the line

Algorithm:

- if x1 ~= x2
  - for all xtmp = x1: x2
    - find ytmp = (-p(1,1)*x - p(1,3))/p(1,2);

```
        save xtmp and ytmp in x and y vectors
    end
end
if x1 == x2
    for all ytmp =  y1: y2
        find x_tmp = (-p(1,2)*y_tmp - p(1,3))/p(1,1);
        save xtmp and ytmp in x and y vectors
    end
end
plot (x,y)
```

2.2. Another method which can be used to find multiple structures (lines) is the Hough transform

To fit a structure with a Hough transform, we take each image token and determine all structures that could pass through that toke. We make a record of this set (this can be thought of as a voting) and repeat the process for each token. We decide on what is present by looking at the votes. For example, if we are grouping points that lie on lines, we take each point and vote for all lines that could go through it; we now do this for each point. The line (or lines) that are present make themselves obvious because they pass through many points and so have many votes.

Fitting Lines with the Hough Transform

A line is easily parametrized as a collection of points (x, y) such that: $x\cos Q + y\sin Q + r = 0$.

Any pair of (Q , r) represents a unique line, where r is the perpendicular distance from the line to the origin and O and $0 \leq Q \leq 2\pi$.

We call the set of pairs (Q, r) line space. The lines can be discretized with some convenient grid (Q,r). The grid elements can be thought of as buckets into which we place votes. This grid of buckets is referred to as the accumulator array. For each point token, we add a vote to the total formed for every grid element on the curve corresponding to the point token. If there are many point tokens that are collinear, we expect there to be many votes in the grid element corresponding to that line indicating that the particular line is a strong candidate.

The above discussed Hough accumulator can be obtained by the following function (CS5320_Hough**). Another output of this function is pts (rxt struct) this stores the location of the points that contributed to the line having a particular (Q,r). Input to the function is the image.

Algorithm:

- Initialize the Hough accumulator to zeros (rmax*2+1,180). {rmax represents the image diagonal size, *2 means +ve and –ve values of rho, +1 represents 0 value of rho, 180 is the angle range}
- Also initialize the pts structure to strore the points that contribute to a particular line with (r, Q)
- Find edges in the image and store in imEgde.
- For all point on the edges:
    - For all theta (0 to 179)
        - find rho = -(x*cosd(theta-1)+y*sind(theta-1))
        - rho_index = round(rmax + 1 - rho);
        - H(rho_ind,theta)= H(rho_ind,theta)+1, %vote
        - Also, save the pixel location for this combination of (rho, theta)
    - End
- End

To verify the results of the CS5320_Hough function, that is, to show the lines detected, we need to use the **CS5320_Hough_lines function. (not CS5320_Hough_lines2)

The input to this function is the image and the Hough accumulator, found from the CS5320_Hough function and a threshold (of votes received for a line).

The output of the function is lines mask (gray is line number).

Here, for all the (rho, theta) having votes > threshold, we can plot the line as: xcosQ + ysinQ + r = 0.

Another output of the CS5320_Hough function was pts (struct), which contains points that contributed to lines. To verify this, we can develop a function: **CS5320_Hough_draw_pts.

The input to this function is the image, Hough accumulator, pts (rxt struct), n (int): number of lines to show, and dr (Boolean): 1 means draw, 0 don't.

Function output: lines (mxn array): mask of n lines.

The function is interactive; it can display points of each line or not based on the Boolean value entered for 'dr'.

Algorithm:

- Find the indexes having votes ; [rhos,thetas] = find(H>=0.0);
- For all (rhos,thetas), store the number of points. Store this in HsizeMat(k,1)
- Find maximum value of points. Hmaxval = max(HsizeMat)
- linesDrawn = 0
- While loop (break this if linesDrawn == n)
    - Find Locations (r,Q) having Hmaxval ; save in currHMaxLocns
    - If currHMaxLocns is empty, then decrement the value of Hmaxval and skip following lines;
    - For all currHMaxLocns

```
        • Set currentLinePoints = pts(currHMaxLocns)
        • linesDrawn = linesDrawn+1;
        • save all points from currentLinePoints to corresponding pixel location as index
          (linesDrawn) in in lines (original image size).
        • Hmaxval = Hmaxval-1;
        • If linesDrawn == n
            • Break
        • End
    • If dr == 1
        • for all lines; lview  = 1 to n
            • prompt = '**Press any key to draw points of next line';
            • dummy = input(prompt);
            • combo(im, lines==lview);
        • end
    • End
```

A function **CS5320_line_segs has also been developed. This Function takes in the Hough points (storage of points which contributed to Hough lines), edge image and min_len (minimum points required in a line) as the input and gives the line segment information (parameters) of the line segments as the output.

<u>Algorithm:</u>

```
• [rowsH,colsH] = size(Hpts);
• For all (rowsH,colsH), find the number of points and save in H.
• For i = 1:rowsH
    • For  j = 1:colsH
        • If HptsSize(i,j) >= min_len
            • s = s+1 %structure count
            • segments(s).rho = (rowsH-1)/2+1-i;
            • segments(s).theta = j-1; % % theta goes from 0 to 179
            • create imtemp: image of zeros(size = original image)
            • for k = 1: HptsSize (i,j)
            • set the pixel values of imtemp to 1 for points contributing to a line
            • perform dilate, erode and thin operation on imtemp to get better lines.
            • Find the largest connected line after performing above operations
            • Find the endpoints of this largest line and save in segments(s).endpt1 and
              segments(s).endpt2
            • Now use CS5320_line_between function and find all the points (including) in between
              endpoints. Store these in segments(s).pts
        • end
    • end
• end
```

After having got the line segments from the image, we can now use these to form/identify shapes like rectangles and triangles in the image. One way this can be done is by grouping the line segments. We can group those line segments whose endpoints are coinciding or whose endpoints are within certain distance range from the other line.

CS5320_shapes

Algorithm:

From the output of the segments function, go to endpoints and check which endpoints are near to each other. Store the pairs of lines corresponding to the endpoints.

Now check all of the pairs, taking 2 at a time . check which of the 2 pairs have the common line

Eg:

(2 , 4)   and (5 , 2) these have common line 2, so this forms a triangle. Store the indices in shapes(s).segs

3. Verification

3.1. CV_total_LS
3.2. CS5320_plot_line

The CS5320_plot_line function is used to verify the CV_total_LS function. CS5320_plot_line will plot a line according to the parameter values given by CV_total_LS. As for the points [x,y] to be inputted to CS5320_plot_line, let us use ginput command in MatLab and get some points. These points can be plotted to verify that the plotted line represents a best fit line.

Matlab code

```
[x,y] = ginput();
[p1,s1] = CV_total_LS(x,y);
CS5320_plot_line(p1,0,7,0,0);
hold on;
for i = 1:size(x,1)
    plot(x(i),y(i),'ro');
end
grid on;
```

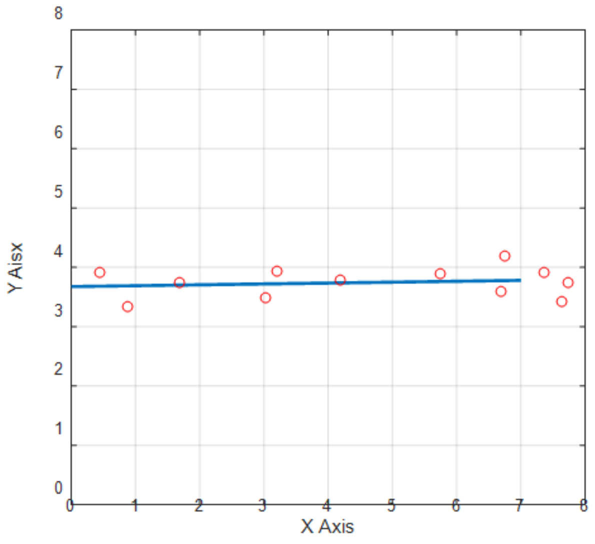


figure: line 1 (Horizontal)

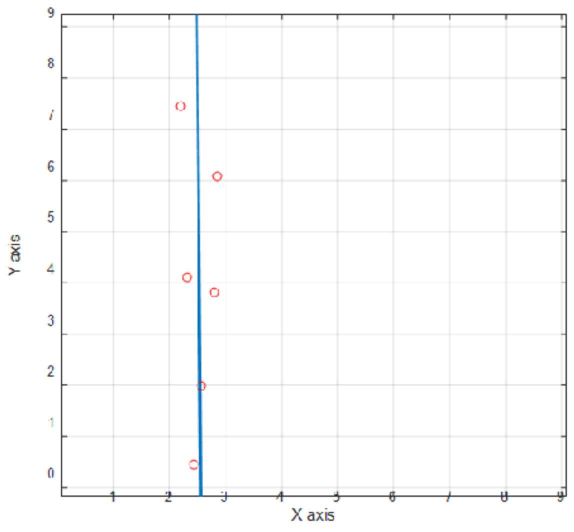


figure: Line 2 (Vertical)

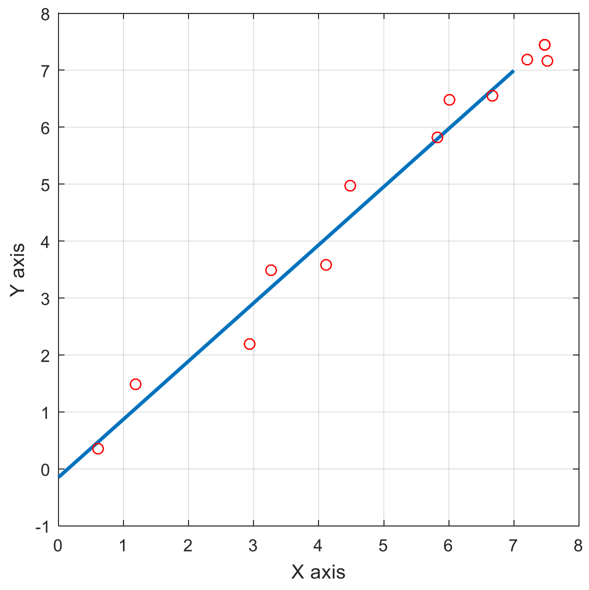


figure: line 3 (at an angle)

As can be seen from the above figures, the points for line 1, 2 and 3 were randomly selected to be almost at horizontal, vertical and at about 45 degrees respectively. The CV_total_LS function does a pretty good job in fitting a line to the given points.

3.3. CS5320_Hough

A simple test can be performed to check whether the output of the Hough function, especially the Hough Accumulator Works properly. For this an image of a line at know angle can be given as input to the Hough function. Then the Hough accumulator can be checked whether the maximum number of votes are received at the proper angle of the normal to the line.

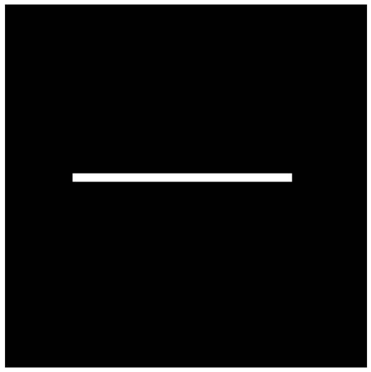


Figure: line 1 with theta = 90 degrees

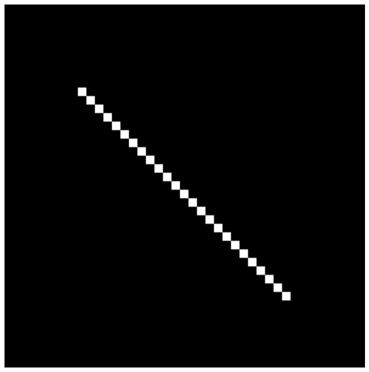


Figure: line 2 with theta = 45 degrees

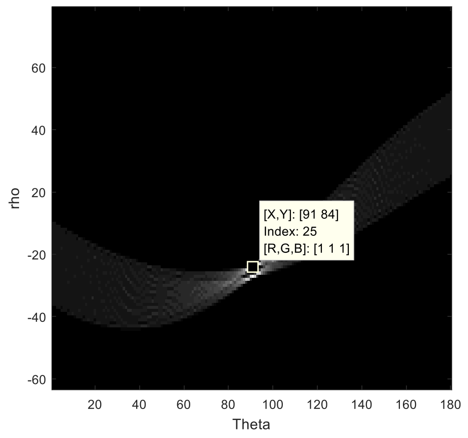


Figure: line 1 imagesc(Hough accumulator)

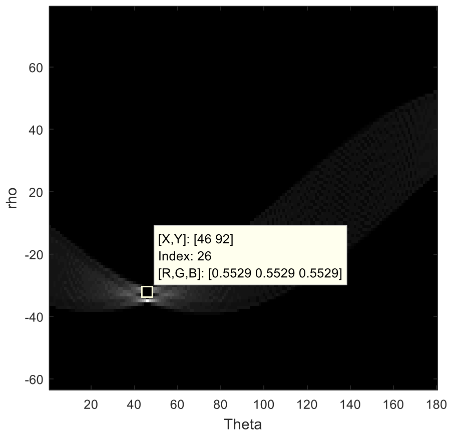


Figure: line 2 imagesc(Hough accumulator)

It is seen that we get the maximum response in the hough accumulator at the proper (rho,theta) location.

Size of original image is 43x43

For the line 1, theta = 91-1=90 %-1 because theta = 0:179

For the line 2, theta = 46 – 1=45

For the line 1, rho = 80-84-20 = -24 rho of -20 corresponds to +80 of row in the Matlab image

For the line 2, rho = 80-92-20 = -32

These are the values that are obtained by theory for the images.

3.4. CS5320_Hough_lines

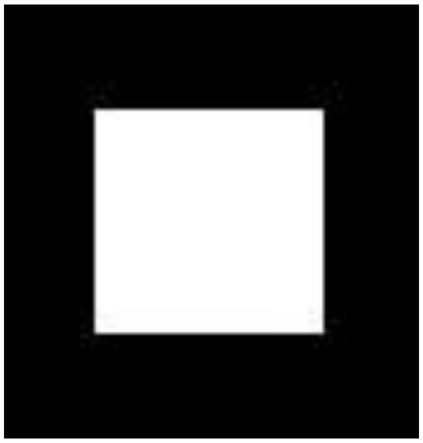


figure: Test image (square)

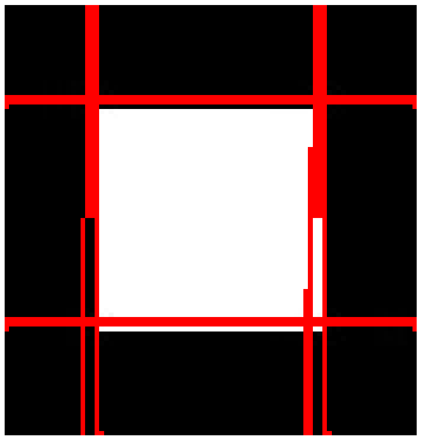


figure: combo(square,lines)

figure:hall4

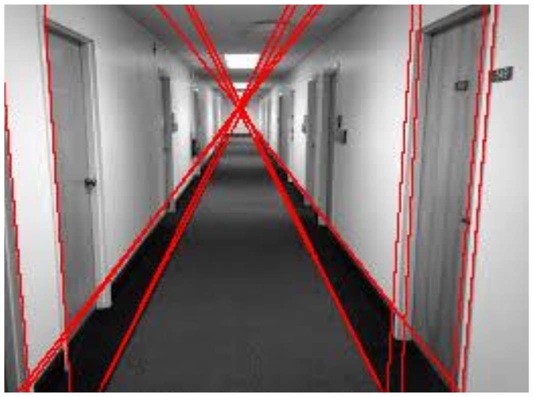


figure: combo(hall4g,lines)

The function gives the hough lines.

3.5. CS5320_Hough_draw_pts

To test this function, the Boolean value of dr (input to the function) has been set to 1 so that points belonging to different images can be observed individually.

Let us test this function on the square test image. Using the following code, with number of lines to show = 4:

```
%Verify Hough_draw_pts
testIm = zeros(50,50);
testIm(11:40,11:40) = 1;
[H2,H2pts] = CS5320_Hough(double(testIm));
linesp = CS5320_Hough_draw_pts(testIm,H2,H2pts,4,1);
figure; combo(testIm,linesp);
```

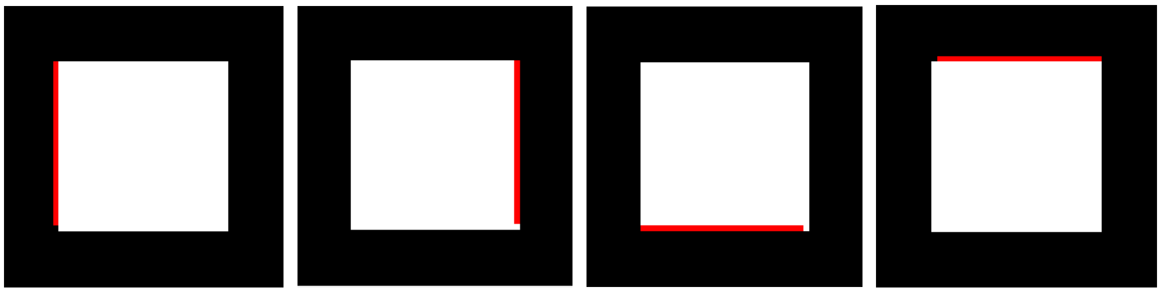


Figure: output of the above code for CS5320_Hough_draw_pts for square image

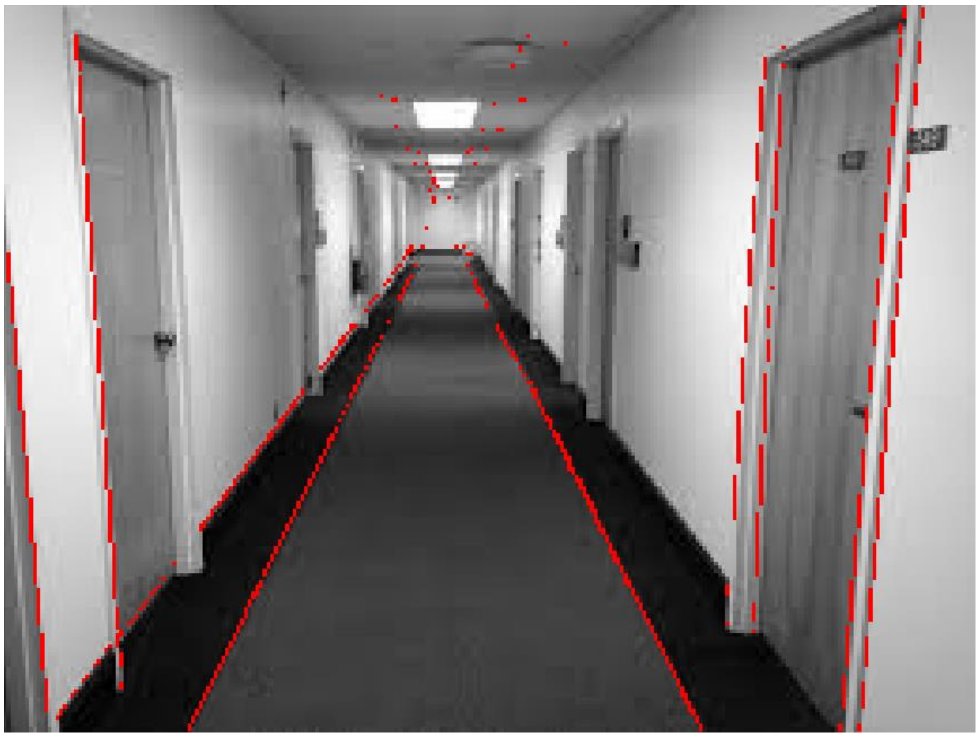
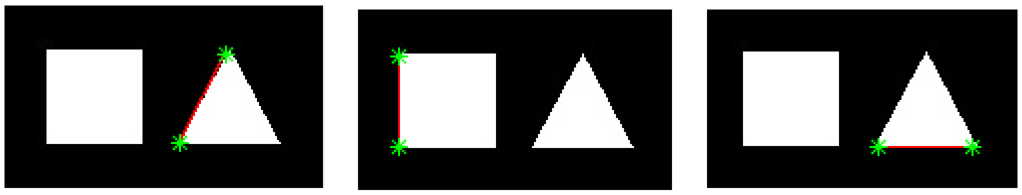


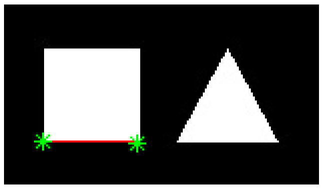
Figure: output of the above code for CS5320_Hough_draw_pts for hall4 image

3.6. CS5320_line_segs

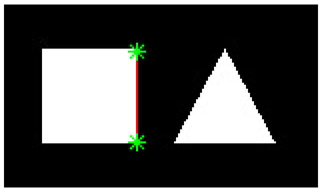
To test whether the function works properly, let us apply it onto an image having a rectangle and a square and then plot the line segments obtained from the function by plotting segments(s).pts, (s).endpt1, (s).endpt2 and showing (s).rho and (s).theta.



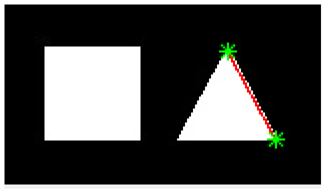
Rho = 152, theta = 66   Rho = -21, theta = 0   Rho = -23, Theta = 90



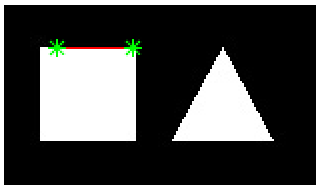
Rho = -24, Theta = 89



Rho = -69, Theta 0



Rho = -132, Theta 28



Rho = -68, Theta 92

Looking at the output obtained from the CS5320_line_segs function, it seems that it is giving the line segments at the correct locations and with correct rho and theta.

4. Data

First, we will apply Gaussian noise on an image and check the performance of the least square function.

Code

```
[x] = [1;2;4;4;5;4;6;5;8;7;9];
[y] = [1;3;3;4;5;6;5;6;7;9;9];
sMat = [];
for sigma = 0.1:0.1:1
    [x] = [x] + randn(size(x))*sigma;
    [y] = [y] + randn(size(y))*sigma;
    [p1,s1] = CV_total_LS([x],[y]);
    sMat = [sMat;s1];
end
close all;
hold on;
plot(sMat,0:0.1:1);
```

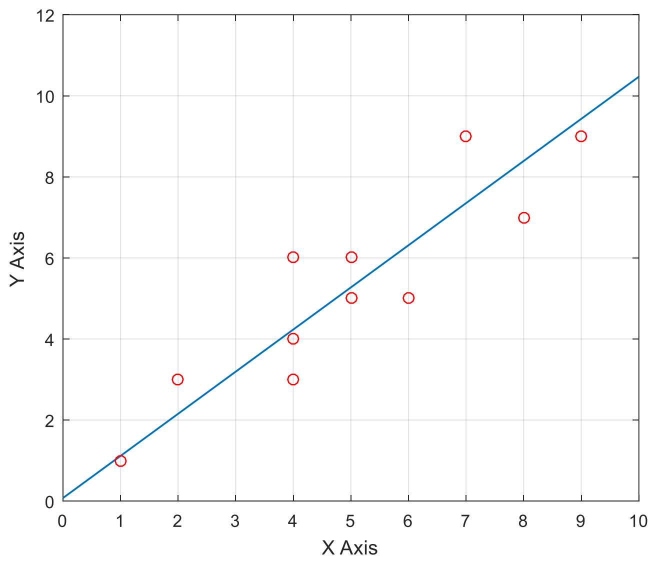


Figure: points used to check the performance of CV_total_LS with noise

sigma	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
sum of squarred distance	5.52	6.28	5.23	9.15	11.57	13.31	19.15	23.25	22.27	26.17

Table: Data for CV_total_LS

To check the performance of CS5320_Hough, let us take an image, store the theta for the maximum voted line. Apply Gaussian noise with increasing sigma. Check how the theta varies.

<u>Code:</u>

```
img = double(rgb2gray(imread('test_line.jpg')));
[H4,H4pts] = CS5320_Hough(double(img));
[rOr,cOr]  = find(H4==max(max(H4)));

sMat = [];
for sigma = 0:10:100
    img = img + randn(size(img))*sigma;
    [H4,H4pts] = CS5320_Hough(double(img));
    [r,c]  = find(H4==max(max(H4)));
    s1 = abs(cOr-c);
    sMat = [sMat;s1];
end
close all;
hold on;
plot(sMat);
```

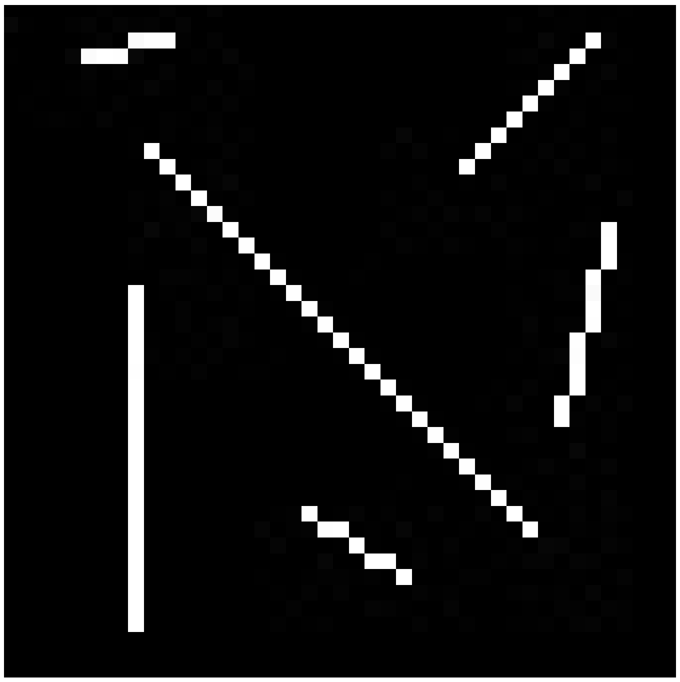


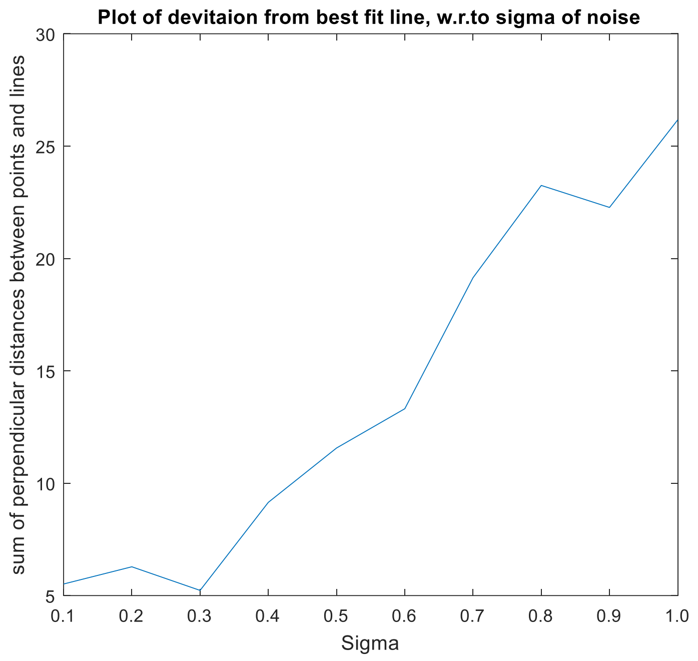
Figure: image used to check the performance of CS5320_Hough with noise

sigma	0	10	20	30	40	50	60	70	80	90	100
Theta error	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

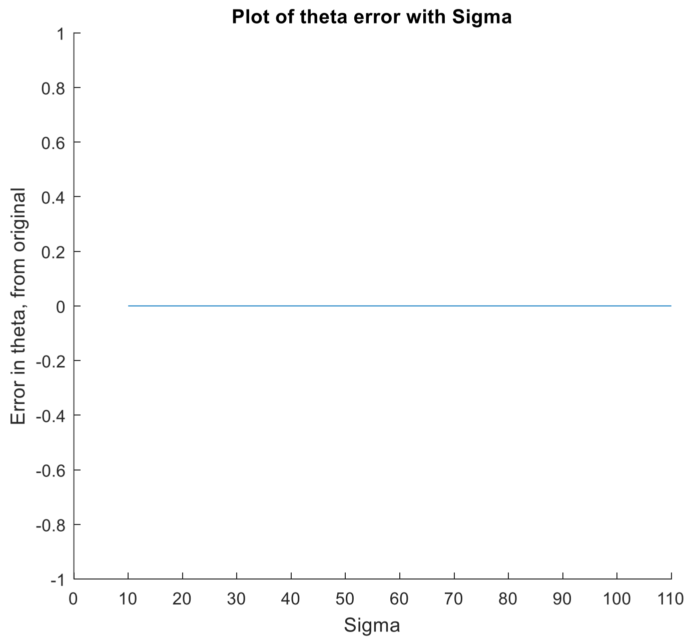
Table: Data for CS5320_Hough

5. Analysis

Analysis for CV_total_LS:



Analysis for CS5320_Hough:



6. Interpretation

For the CV_total_LS function, it is seen that with the addition of noise, the error measure. $\sum(ax_i + by_i + c)^2$ increases. Hence, the function is affected by the Gaussian noise.

For the CS5320_Hough function, it was seen that even with increase in noise, the function detects the correct theta corresponding to maximum voted line.

7. Critique

I did my best.

8. Log

20 hrs