EXERCISE 1: ROBUST ESTIMATION AND NORMS

GCT722 MATHEMATICAL METHODS FOR VISUAL COMPUTING

20183151 Chaelin Kim

PART 1: RANSAC FOR CIRCLE FITTING

Description of implementation

There are 1 script file & 4 function files for the exercise RANSAC.

- **Script file**
  - **main_RANSAC.m**

    : This is the script for doing RANSAC for circle fitting. It calls functions for RANSAC (genCircleData, doRANSAC, doExhaustiveSearch, drawRANSACPlot). Details of these functions are explained below. We can set the initial settings in this script.

```
%% Initial settings
n = 100;                    % Number of data points
center = [0, 0];            % Center value for the synthesized circle
radius = 5;                 % Radius value for the synthesized circle
inlierThreshold = 0.1;
outlierRatio = [0.05, 0.2, 0.3, 0.7];
```

- **Function files**
  - **genCircleData.m**

    : This function generates data points on a synthesized circle with inliers and outliers. The number of inliers and outliers are decided by the given outlier ratio. The inlier data is made with random noise (between -0.1 and 0.1). In the part of outlier generation, *while* loop is executed until the number of made outlier data meets the given number of outliers.

```
function [ data ] = genCircleData(  n, center, radius, inlierThreshold, outlierRatio )
```

- Input Value
  - *n*: The number of data points
  - *center*: The center (x, y) value of the synthesized circle
  - *radius*: The radius value of the synthesized circle
  - *inlierThreshold*: The inlier distance threshold
  - *outlierRatio*: The ratio of outliers in data
- Output
  - *data*: The data in the form of circle with inliers and outliers according to the outlier ratio.

doRANSAC.m

: This function runs RANSAC for the given data. In this function, the number of RANSAC iteration is computed and used in *for* loop for RANSAC (How to calculate this number is explained in Questions category below (What about the number of RANSAC iterations with r = 5%, 20%, 30% and 70%?)). For doing RANSAC with the circle data, we can pick the random three points. The center of the circle passing through these three points is the intersection of the perpendicular bisectors of the lines connecting the two points. In calculating errors(distances), Inliers are calculated by the center and radius values of founded circle and saved in the matrix. Then the best model is updated by the number of inliers. This process is iterated 1000 times for finding best RANSAC model. (Reference how to find a circle with 3 points: http://egloos.zum.com/heilow/v/418569)

```
function [bestModel, detectedInliers] = doRANSAC(data, M, inlierThreshold, outlierRatio)
```

- Input Value
  - *data*: The data made from genCircleData function
  - *M*: The number of Iteration for re-apply RANSAC
  - *inlierThreshold*: The inlier distance threshold.
  - *outlierRatio*: The ratio of outliers in data
- Output
  - *bestModel*: The computed data [center_x_value, center_y_value, radius]
  - *detectedInliers*: The number of computed inliers of each iteration of re-apply RANSAC

- **doExhaustiveSearch.m**

  : This function runs exhaustive search for the given data. The combination value is calculated by function *nchoosek*. The method how to find inliers is similar to *doRANSAC* function. In addition, it has the code for displaying the result of exhaustive searching.

```
function [ ] = doExhaustiveSearch( data, k, inlierThreshold, outlierRatio)
```

  - Input Value
    - *data*: The data made from *genCircleData* function
    - *k*: How many points are selected for searching
    - *inlierThreshold*: The inlier distance threshold
    - *outlierRatio*: The ratio of outliers in data
  - Output
    - No output but it displays the result on the command window.

- **drawRANSACPlot.m**

  : This function draws histograms of re-applying RANSAC and plots of RANSAC results with different outlier ratios. The result is shown in category *Screenshots* below.

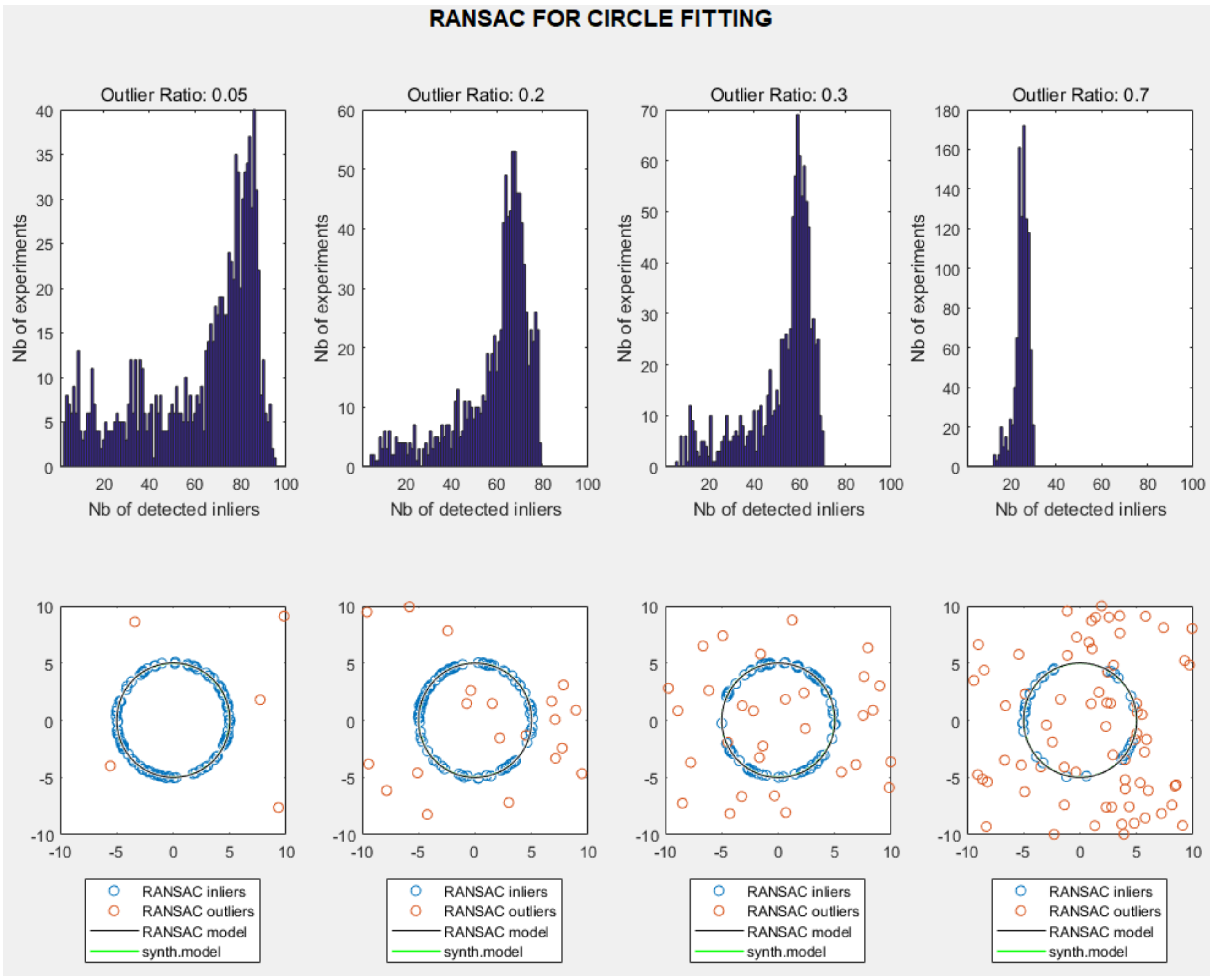
```
function [ ] = drawRANSACPlot( center, radius, data, ransacResult, histResult, outlierRatio )
```

  - Input Value
    - *center*: The center (x, y) value of the synthesized circle
    - *radius*: The radius value of the synthesized circle
    - *data*: All data made from *genCircleData* function (type: cell)
    - *ransacResult*: The results of RANSAC made from *doRANSAC* function (type: cell)
    - *histResult*: The results of the number of inliers made from *doRANSAC* function (type: cell)
    - *outlierRatio*: The ratio of outliers in data for draw outlier data
  - Output
    - No output. It shows result plots.

Instructions for running

1. Open the file “main_RANSAC.m” in Matlab.
2. Execute that file.
    a. You can see the result of exhaustive search in command window.
    b. The another window that shows plots of RANSAC results is opened.

Screenshots



Questions

- **How many combinations (exhaustive search) exist for N = 100 points?**
  - ➢ Ans: 161700

```
n = 1:1:size(data,2);
combination = nchoosek(n,k);
```

```
>> main_RANSAC
+------------- Outlier Ratio 0.05 -------------+
Number of iteration: 161700
Number of max Inlier: 95
Model(x, y, r): 0.0037808,0.009061,4.9958
----------------------------------------------
+------------- Outlier Ratio 0.2 -------------+
Number of iteration: 161700
Number of max Inlier: 80
Model(x, y, r): -0.017558,0.010588,4.9949
----------------------------------------------
+------------- Outlier Ratio 0.3 -------------+
Number of iteration: 161700
Number of max Inlier: 70
Model(x, y, r): 0.010322,-0.0037466,5.0027
----------------------------------------------
+------------- Outlier Ratio 0.7 -------------+
Number of iteration: 161700
Number of max Inlier: 30
Model(x, y, r): -0.0079792,0.00041178,5.0178
----------------------------------------------
```

- **What about the number of RANSAC iterations with r = 5%, 20%, 30% and 70%?**
  - ➢ $$N = \frac{\log(1-p)}{\log(1-(1-\varepsilon)^s}$$
    - *p*: Probability that at least one sample has no outliers
    - ε: Outlier ratio
    - *s*: Sample size (minimum data)

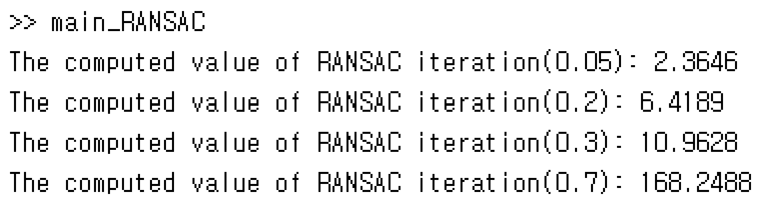
→
```
N = log(1-0.99) / log(1 - (1 - outlierRatio)^3);
```

```
>> main_RANSAC
The computed value of RANSAC iteration(0.05): 2.3646
The computed value of RANSAC iteration(0.2): 6.4189
The computed value of RANSAC iteration(0.3): 10.9628
The computed value of RANSAC iteration(0.7): 168.2488
```

- **What about when N = 10,000 points?**
  - Exhaustive search

    combinations = $\frac{n!}{(n-k)!k!} = \frac{10000!}{(10000-3)!3!} = 166{,}616{,}670{,}000$

    → Very big number…
  - RANSAC iterations → not changed!



```
>> main_RANSAC
The computed value of RANSAC iteration(0.05): 2.3646
The computed value of RANSAC iteration(0.2): 6.4189
The computed value of RANSAC iteration(0.3): 10.9628
The computed value of RANSAC iteration(0.7): 168.2488
```

→ The number of RANSAC iterations is not affected with the number of data because it is effected by precision probability, outlier ratio and sample size.

Discuss the results

In this code, I made synthesized model with outlier ratios 5% 20%, 30% and 70%, did RANSAC and exhausted search with synthesized data, and draw the plots that show the results.

In the histogram, considering with outlier ratios, RANSAC finds the number of inliers similar to real number of inliers. In the plots, RANSAC with re-applying 1000 times finds quite fitted circle model in case of all outlier ratios.

Exhaustive searching finds the best fitted model because it considers all combinations of data points. In that point, if the number of data increases, the computational cost exponentially increases. On the contrary, RANSAC is not affected with the number of data. So, if other conditions are fixed and the number of data increases, RANSAC finds a fitted model faster than exhaustive search. But if iteration for re-applying is lower and outlier ratio is higher, RANSAC would be hard to find well-fitted model.

PART 2: IRLS AND NORMS FOR LINE FITTING

Description of implementation

There are 1 script file & 4 function files for the exercise RANSAC.

- **Script file**
  - **main_LineFitting.m**

    : This is the script for doing line fitting using L_1 norm (employing IRLS and LP) and L_∞ norm (employing LP). It calls functions for line fitting (genLineData, doIRLS, doLP, drawLineFittingPlot). Details of these functions are explained below. We can set the initial settings in this script.

```
%% Initial settings
n = 100;             % Number of data points
a = 5 / 6;           % Gradient value for the synthesized line
b = 0.1;             % Bias value for the synthesized line
inlierThreshold = 0.1;
outlierRatio = [0, 0.1];
```

- **Function files**
  - **genLineData.m**

    : This function generates data points on a synthesized line with inliers and outliers. The number of inliers and outliers are decided by the given outlier ratio. The inlier data is made with random noise (between -0.1 and 0.1). In the part of outlier generation, *while* loop is executed until the number of made outlier data meets the given number of outliers.

```
function [ data ] = genLineData( n, a, b, inlierThreshold, outlierRatio )
```

- Input Value
  - *n*: The number of data points
  - *a*: The gradient value of the synthesized line
  - *b*: The bias value of the synthesized line
  - *inlierThreshold*: The inlier distance threshold
  - *outlierRatio*: The ratio of outliers in data
- Output
  - *data*: The data in the form of line with inliers and outliers according to the outlier ratio.

doIRLS.m

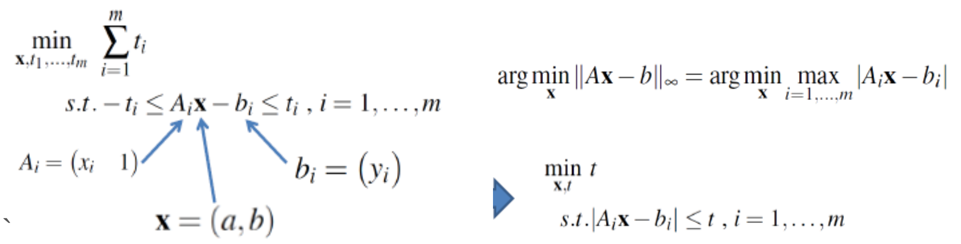
: This function runs IRLS with L_1 for the given data. For IRLS, we change L_p norm to weight * L_2 norm for easy computation. L_2 norm is computed with setting the gradient to 0. The weight is computed with the value of gradient and bias calculated by L_2 norm. With this weight, the result is recomputed. This process is iterated until the absolute value of difference between previous value and current value is smaller than 0.0001.

```
function [ result_IRLS ] = doIRLS( data )
```

- Input Value
  - *data*: The data made from *genLineData* function (type: matrix)
- Output
  - *result_IRLS*: The result data [a, b] computed by IRLS

doLP.m

: This function runs Linear Programming(LP) with L_1 and L_∞ for the given data. Linear programming solver finds the minimum of a problem. The purpose of L_1 is minimize the sum of the absolute value of all errors and the purpose of L_∞ is minimize the maximum error among the absolute value of errors. In order to solve this problem with LP, I made the matrix A and b that consist of linear equations with given data. Using *linprog()* function, the result is easily computed.



$$\min_{\mathbf{x},t_1,\dots,t_m} \sum_{i=1}^{m} t_i$$

$$s.t. -t_i \le A_i\mathbf{x} - b_i \le t_i\ ,\ i = 1,\dots,m$$

$$A_i = (x_i \quad 1) \qquad b_i = (y_i) \qquad \mathbf{x} = (a,b)$$

$$\arg\min_{\mathbf{x}} \|A\mathbf{x} - b\|_\infty = \arg\min_{\mathbf{x}} \max_{i=1,\dots,m} |A_i\mathbf{x} - b_i|$$

$$\min_{\mathbf{x},t} t$$

$$s.t. |A_i\mathbf{x} - b_i| \le t\ ,\ i = 1,\dots,m$$

→ L_1 for LP(left) and L_∞ for LP(right)

```
function [ result_LP ] = doLP( data, Lnorm )
```

- Input Value
  - *data*: The data made from *genLineData* function (type: matrix)
  - *Lnorm*: The kind of Lp norm (type: String)
- Output
  - result_LP. The result data computed by Linear programming

- **drawLineFittingPlot.m**

  : This function draws plots of line fitting results(IRLS with L_1 and LP with L_1 and L_∞) with different outlier ratios. The result is shown in category *Screenshots* below.

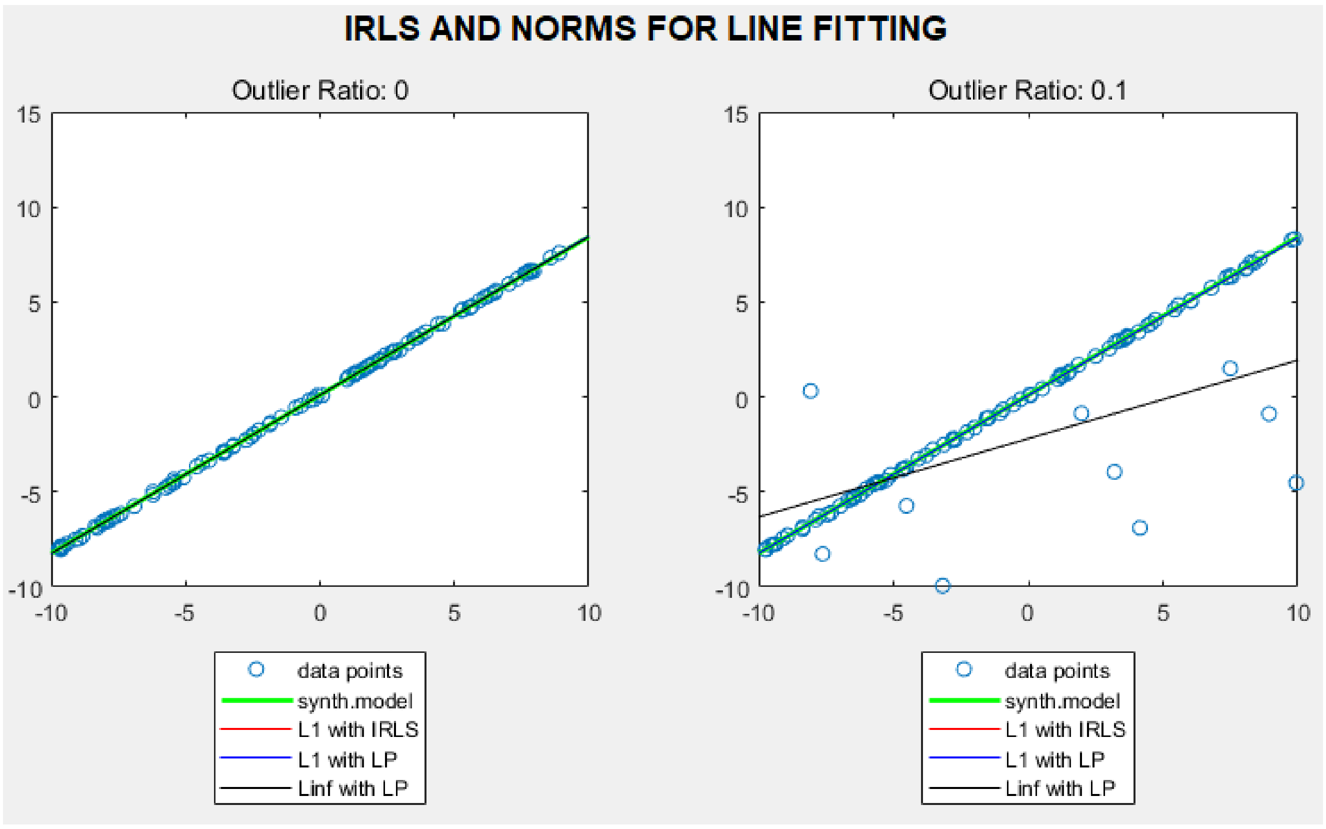
```
function [ ] = drawLineFittingPlot( a, b, data, result, outlierRatio )
```

  - Input Value
    - *a*: The gradient value of the synthesized line
    - *b*: The bias value of the synthesized line
    - *data*: All data made from *genLineData* function (type: cell)
    - *result*: The results of IRLS with L_1 made from *doIRLS* function and LP with L_1 and L_∞ made from *doLP* function (type: cell)
    - *outlierRatio*: The ratio of outliers in data for draw outlier data
  - Output
    - No output. It shows result plots.

Instructions for running

1. Open the file “main_LineFitting.m” in Matlab.
2. Execute that file
   a. The another window that shows plots of IRLS with L_1 and LP with L_1 and L_∞ norms results is opened.

Screenshots



Discuss the results

In this code, I made synthesized model with outlier ratios 0% and 10%, doing line fitting using L_1 norm (employing IRLS and LP) and L_∞ norm (employing LP) with synthesized data, and draw the plots that show the results.

In case of outlier ratio 0%, all methods show similar results each other and also almost match with the synthesized model. In case of outlier ratio 10%, however, the results of L_1 norm still similar to the synthesized model but there is quite difference between the result of L_∞ norm and synthesized model. The purpose of L_∞ norm is minimize the maximum error among all errors, so it is not robust to outliers. The results of L_1 norm with IRLS and LP are almost same but I think LP is more intuitive than IRLS