Exercise 1. Robust Estimation and Optimization

Dongho Kang

16-948-598

March 12, 2017

MATLAB R2016b version was used for coding and testing:

MathWorks, MATLAB R2016b (9.1.0.441655)
64-bit (maci64)

The ***code*** directory contains followings:

- ***part1.m*** Script .m file for part1.
- ***part2.m*** Script .m file for part2.
- ***functions dir*** Function .m files
  - **GenerateInlierData.m** Function .m file for generating inliers.
  - **GenerateOutlierData.m** Function .m file for generating outliers.
  - **VerifyInlier.m** Function .m file for verifying inliers.
  - **RansacForCircularModel.m** Function .m file for RANSAC implementation.
  - **ExhSearchForCircularModel.m** Function .m file for exhaustive search.
  - **CircularModelFitting.m** Function .m file for circular model fitting.
  - **PlotBestRansacResult.m** Function .m file for plotting RANSAC result.
  - **Cost.m** Function .m file for calculating vertical cost.
  - **IRLSWithL1Norm.m** Function .m file for IRLS with L_1 norm.
  - **LinProgWithLqNorm.m** Function .m file for LP with L_1 and L_∞ norm.

For running exercise parts, adjust several parameters and run the corresponding script on the MATLAB environment. More details are stated in the *Running* section.

1 EXERCISE PART 1: RANSAC FOR CIRCLE FITTING

1.1 DESCRIPTION

In exercise part 1, RANSAC algorithm was used for circular model fitting with N number of 2D data points which were corrupted by noise and certain ratio of outliers. The implementation of part 1 corresponds to the .m script file ***part1.m*** and the script consists of several parts which can be executed separately:

- Data generation and verification
- Running RANSAC algorithm for circular model fitting
- Running exhaustive search for circular model fitting
- Plotting the result

1.1.1 DATA GENERATION AND VERIFICATION

For certain outlier ratio r(%) and N number of 2D data points, the number of inliers (n_{inlier})and outliers($n_{outlier}$) are as follows:

$$n_{inlier} = N \times \frac{1-r}{100}$$
$$n_{outlier} = N \times \frac{r}{100}$$
$$(N = 100,\ r \in \{5, 20, 30, 70\})$$

In order to generate n_{inlier} inliers and $n_{outlier}$ outliers, the function **GenerateInlierData** and **GenerateOutlierData** are implemented. **GenerateInlierData** generates noise-added data points with the inlier distance threshold $\tau = 0.1$ and **GenerateOutlierData** generates data points which of distance from the synthetic model are over the threshold τ. It can be described as follows. $d(p_1, p_2)$ is euclidean distance function between two 2D points p_1 and p_2, (x_c, y_c) is the center of the synthetic circle and R_c is the radius of the synthetic circle:

$$\left| d\big((x_i, y_i), (x_c, y_c)\big) - R_c \right| \leq \tau \qquad for (x_i, y_i) \in Set_{inlier}$$
$$\left| d\big((x_j, y_j), (x_c, y_c)\big) - R_c \right| > \tau \qquad for (x_j, y_j) \in Set_{outlier}$$

After generating inliers and outliers, verified whether generated inliers follow the definition above. For this purpose, the function **VerifyInlier** is used. The specific descriptions for each steps are in the next page.

INLIER GENERATION A 2D point $(\tilde{x}_i, \tilde{y}_i)$ on the circular model $(x-x_c)^2+(y-y_c)^2=R_c^2$ which of center (x_c, y_c), and radius R_c can be described as follows:

$$(\tilde{x}_i, \tilde{y}_i) = (x_c, y_c) + R_c \times (\cos\theta_i, \sin\theta_i) \qquad \theta_i \in [0, 2\pi], i = 1, 2, \ldots n_{inlier}$$

Thus, picked n_{inlier} number of random float numbers from the range of $[0, 2\pi]$ by MATLAB function **rand** and as using these numbers as θ_i, generated (x_i, y_i) for $i = 1, 2, \ldots n_{inlier}$. Random noise $(\sigma_{x_i}, \sigma_{y_i})$ is added on the data point $(\tilde{x}_i, \tilde{y}_i)$ as follows:

$$(x_i, y_i) = (\tilde{x}_i, \tilde{y}_i) + (\sigma_{x_i}, \sigma_{y_i}) \quad s.t. \quad \sigma_{x_i}, \sigma_{y_i} \in [-0.1, 0.1], \quad \Big| d\big((x_i, y_i), (x_c, y_c)\big) - R_c \Big| \leq \tau$$

To implement this, two random float numbers by **rand** were picked for $(\sigma_{x_i}, \sigma_{y_i})$ iteratively, until it satisfies $\Big| d\big((x_i, y_i), (x_c, y_c)\big) - R_c \Big| \leq \tau$ and repeated it for $i = 1, \ldots, n_{inlier}$ to generate n_{inlier} number of inliers. Also, (x_i, y_i) is checked whether it is in the domain $[-10, 10]$.

OUTLIER GENERATION In order to generate one outlier point (x_j, y_j), two random float numbers were generated by **rand** in the domain of $[-10, 10] \times [-10, 10]$ and generated (x_j, y_j) was checked whether it satisfies $\Big| d((x_j, y_j), (x_c, y_c)) - R_c \Big| > \tau$. If not, generated (x_j, y_j) again. Repeated this for $n_{outlier}$ times to generate $n_{outlier}$ number of outliers.

DATA VERIFICATION Checked whether synthesized inliers are indeed inliers and outliers are indeed outliers. The function **VerifyInlier** was used. The function calculates $\Big| d((\tilde{x}_i, \tilde{y}_i), (x_c, y_c)) - R_c \Big|$ for each data points and check if there is a inlier point which of $\Big| d((x_i, y_i), (x_c, y_c)) - R_c \Big| > \tau$ or an outlier point which of $\Big| d((x_j, y_j), (x_c, y_c)) - R_c \Big| \leq \tau$. In this case, error is returned and the script halts.

1.1.2 RANSAC FOR CIRCULAR MODEL FITTING

For generated data point (x_i, y_i) $i \in 1, 2 \ldots, N$, RANSAC algorithm applied to get the best model parameter which of maximum number of inliers within n_{iter} RANSAC iterations. **RansacForCircularModel** was implemented for RANSAC algorithm and the algorithm is applied for 1000 times for each outlier ratio to find the distribution of the number of inliers found by RANSAC.

The number of RANSAC iterations n_{iter} for success rate p ($0 \leq p < 1$, for this exercise, $p = 0.99$), outlier ratio r(%) and sample size s (= 3 for circular model: two for x, y coordinates of center and one for radius) is defined as follows:

$$n_{iter} = \lceil \frac{log(1-p)}{log(1-(1-\frac{r}{100})^s)} \rceil$$

Thus, $n_{iter} = 3$ for $r = 5\%$, $n_{iter} = 7$ for $r = 20\%$, $n_{iter} = 11$ for $r = 30\%$ and $n_{iter} = 169$ for $r = 70\%$. During n_{iter} iterations, found the best result of RANSAC which of maximum number of inliers.

For each RANSAC iteration, sampled $s = 3$ data points without replacement. If the same data points are sampled, model fitting could be failed because it is under-determined problem.

CIRCULAR MODEL FITTING With respect to sampled data points $(x'_1, y'_1), (x'_2, y'_2), (x'_3, y'_3)$, used **CircularModelFitting** for model fitting. The function solves the following equation (variables a, b, c are parameter for fitted model which correspond to x_c, y_c and R_c for each):

$$
\begin{aligned}
(x'_1 - a)^2 + (y'_1 - b)^2 &= c^2 \\
(x'_2 - a)^2 + (y'_2 - b)^2 &= c^2 \\
(x'_3 - a)^2 + (y'_3 - b)^2 &= c^2
\end{aligned}
$$

As eliminating variable c, the equation can be expressed to linear equation as follows:

$$
\begin{bmatrix} -2(x'_1 - x'_2) & -2(y'_1 - y'_2) \\ -2(x'_2 - x'_3) & -2(y'_2 - y'_3) \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} x'^2_2 - x'^2_1 + y'^2_2 - y'^2_1 \\ x'^2_3 - x'^2_2 + y'^2_3 - y'^2_2 \end{bmatrix}
$$

For $Ax = b$, if A is square matrix and not singular then $x = A^{-1}b$. Thus $x = [a \quad b]^T$ can be obtained and after that, c also be calculated by the following equation:

$$
c = \sqrt{x_1^2 - 2ax_1 + a^2 + y_1^2 - 2by_1 + b^2}
$$

As a, b and c (center and radius of the fitted circle) have been determined, set of inliers and outliers can be determined as being done for data verification. This circular model fitting algorithm can be used for not only RANSAC but also for exhaustive search algorithm.

1.1.3 EXHAUSTIVE SEARCH

Another option for model fitting is trying every combination of data points and choosing the best combination. In order to do this, MATLAB function **nchoosek** was used. For $N = 100$ and $s = 3$, the number of all possible combination is as follows:

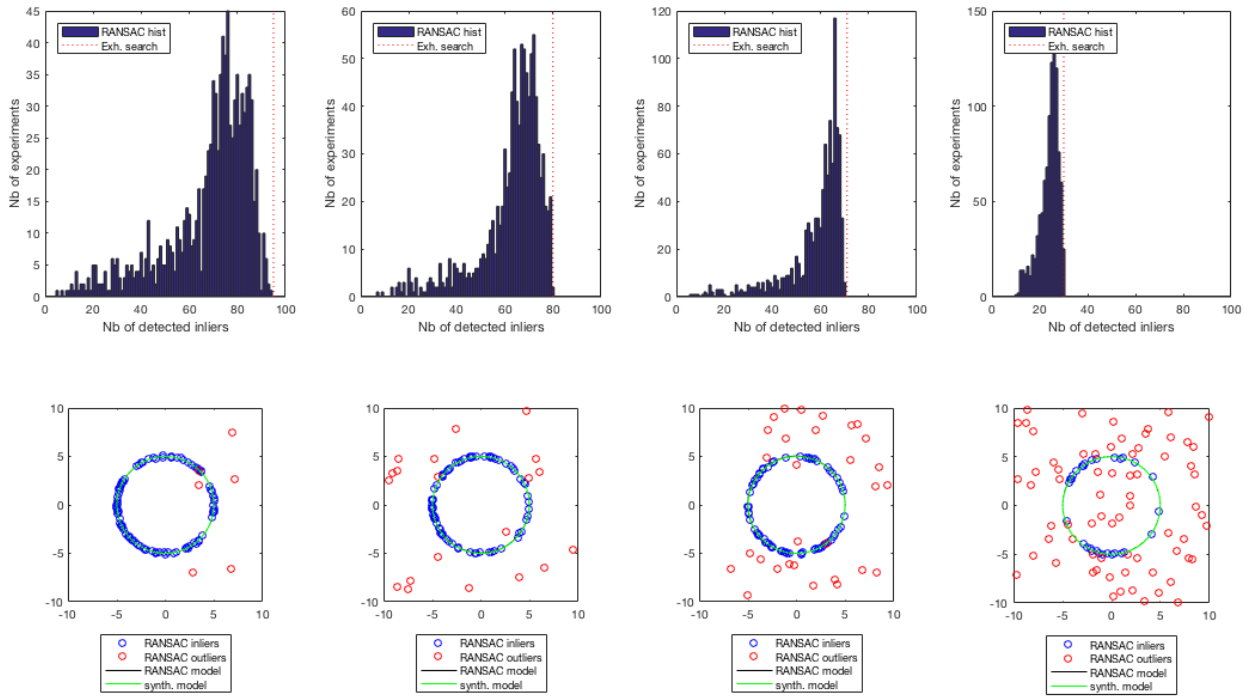
$$
n_{comb} = \binom{100}{3} = \frac{100!}{3! \times 97!} = 161700
$$

For each combination, used **CircularModelFitting** for model fitting and found the number of inliers of the fitted model. Finally, as compared result of each combinations, the best result of exhaustive search can be obtained. This was implemented as the function **ExhSearchFor-CircularModel**.

1.2 RUNNING

Several parameters can be changed for running the script but only changing model parameters (i.e. variables x_c, y_c, radius, noise_radius) is allowed to guarantee proper execution. If model parameter combination is not feasible(e.g. out of domain bound), error is returned.

1.3 RESULT



The result of RANSAC circular fitting with different outlier ratios is as the plot above. The red lines on the histogram indicate the result of exhaustive search. (here the synthetic model is $center = (0,0)$ and $radius = 5$)

The table shows the number of inliers and outliers of the fitted model by 1000 times of RANSAC test:

r	5%	20%	30%	70%
max. # of inliers	94	80	70	30
avg. # of intlier	68.13	62.71	57.96	23.88
std. # of intlier	17.74	13.13	11.87	4.10
median. # of intlier	73	66	62	25

The table shows the number inliers and outliers of the best circular model by exhaustive search:

r	5%	20%	30%	70%
# of inliers	95	80	71	30

The table shows the average running time of RANSAC and the running time of exhaustive search:

r	5%	20%	30%	70%
RANSAC (avg.)	0.0028	0.0039	0.0060	0.0762
exh. search	62.10	55.73	57.94	60.12

1.4 DISCUSSION

Comparing RANSAC and exhaustive search:

- For $N = 100$:
  - The number of combinations for $N = 100$

$$n_{comb} = \binom{100}{3} = \frac{100!}{3! \times 97!} = 161700$$

  - The number of RANSAC iterations for each outlier ratio for $N = 100$

$$n_{iter} = \lceil \frac{log(1-p)}{log(1-(1-\frac{r}{100})^s)} \rceil$$

$r =$	5%	20%	30%	70%
$n_{iter} =$	3	7	11	169

    Thus, **running time of RANSAC increases as r goes up.**

- For $N = 100,000$:
  - The number of combinations for $N = 100,000$

$$n_{comb} = \binom{100000}{3} = \frac{100000!}{3! \times 99997!} = 166661666700000$$

  - The number of RANSAC iterations for $N = 100,000$ is same as the case of $N = 100$. **The number of RANSAC iteration is regardless of the size of data set.**

- The number of inlier of the best model searched by exhaustive search is always bigger (or equal) than RANSAC result.
  - It because, exhaustive search tries all combinations of data points, and can find global optimal solution.
  - **RANSAC can only find local optimal solutions.**
  - Nevertheless, the best result of 1000 RANSAC tests are close with the result of exhaustive search.

- The running time of exhaustive search and RANSAC:
  - The running time is proportion of the number of iteration. Thus **RANSAC is much faster than exhaustive search.**
  - However, as r goes up, the number of RANSAC iteration also increases. Besides, the running time of exhaustive search is regardless of r.

- **In order to get a global optimal solution, RANSAC algorithm should be reapplied for several times, however it can be still faster than exhaustive search.** Although n_{iter} of RANSAC increases as r increases, since standard deviation of the number of inlier distribution decreases, the number of reapplying RANSAC (n_{test}) also can be decreased to save the total running time.

2 EXERCISE PART 2: IRLS AND NORMS FOR LINE FITTING

2.1 DESCRIPTION

In exercise part 2, IRLS and Linear Programming algorithm was used for line model fitting with N number of 2D data points which were corrupted by noise and certain ration of outliers. The implementation of part 2 corresponds to the .m script file ***part2.m*** and the script consists of several parts:

- Data generation and verification
- Run IRLS with L_1 norm
- Run LP with L_1 norm
- Run LP with L_∞ norm
- Plot the result

2.1.1 DATA GENERATION AND VERIFICATION

To generate data points on the synthetic model line, the new cost function(vertical cost: **Cost**) was used instead of euclidean distance function. For data point (x_i, y_i) and line model $y = ax + b$, it was defined as follows:

$$d\big((x_i, y_i), (a, b)\big) = \big|y_i - (ax_i + b)\big|$$

Thus, inliers and outliers are defined as follows:

$$\begin{aligned} d\big((x_i, y_i), (a, b)\big) &\leq \tau \qquad for (x_i, y_i) \in Set_{inlier} \\ d\big((x_j, y_j), (a, b)\big) &> \tau \qquad for (x_j, y_j) \in Set_{outlier} \end{aligned}$$

The strategy of data generation is almost same as exercise part 1 but line model and the new cost are adapted.

INLIER GENERATION A 2D point $(\tilde{x_i}, \tilde{y_i})$ on the line model $y = ax + b$ can be described as follows:

$$\tilde{y_i} = a\tilde{x_i} + b \qquad \tilde{x_i} \in [-10, 10]$$

Random noise $(\sigma_{x_i}, \sigma_{y_i})$ is added on the data point $(\tilde{x_i}, \tilde{y_i})$ as follows:

$$(x_i, y_i) = (\tilde{x_i}, \tilde{y_i}) + (\sigma_{x_i}, \sigma_{y_i}) \quad s.t. \quad \sigma_{x_i}, \sigma_{y_i} \in [-0.1, 0.1], \quad d\big((x_i, y_i), (a, b)\big) \leq \tau$$

OUTLIER GENERATION An outlier point (x_j, y_j) is generated by two random float numbers and should be in the domain of $[-10, 10] \times [-10, 10]$ Also it is checked whether it satisfies $d\big((x_j, y_j), (x_c, y_c)\big) > \tau$.

DATA VERIFICATION Inliers and outliers are verified whether satisfies the following conditions:

$$d\big((x_i, y_i), (a, b)\big) \leq \tau \qquad for (x_i, y_i) \in Set_{inlier}$$
$$d\big((x_j, y_j), (a, b)\big) > \tau \qquad for (x_j, y_j) \in Set_{outlier}$$

2.1.2 IRLS WITH L_1 NORM

For model parameter $\mathbf{x} = (a, b)^T$ and data $\mathbf{y}_i = (x_i, y_i)^T$ for $i = 1, 2 \ldots N$, the line fitting problem can be expressed with L_1 norm:

$$\arg\min_{\mathbf{x}} C_\rho(\mathbf{x})$$
$$where \quad C_\rho(\mathbf{x}) = \sum_{i=1}^{N} \rho \circ f_i(\mathbf{x}) = \sum_{i=1}^{N} d(\mathbf{y}_i, \mathbf{x})$$

The form above can be changed to weighted least squares problem as defining $f_i(\mathbf{x}) = d(\mathbf{y}_i, \mathbf{x})^2$ and $\rho(s) = \sqrt{s}$ and was solved by IRLS(Iteratively Reweighted Least Squares):

$$\mathbf{x}^{t+1} = \arg\min_{\mathbf{x}} C(\mathbf{x}, \mathbf{w}^t) = \arg\min_{\mathbf{x}} \sum_{i=1}^{N} w_i^t f_i(\mathbf{x})$$
$$where \quad w_i^t = \frac{1}{2} d(\mathbf{y}_i, \mathbf{x}^t)^{-1}$$

In the function **IRLSWithL1Norm**, the problem was described as linear equation:

$$\mathbf{x}^{t+1} = \arg\min_{\mathbf{x}} C(\mathbf{x}, \mathbf{w^t}) = \arg\min_{\mathbf{x}} \sum_{i=1}^{n} w_i^t f_i(\mathbf{x}) = \arg\min_{\mathbf{x}} (A\mathbf{x} - b)^T W^t (A\mathbf{x} - b) = (A^T W^t A)^{-1} A^T W^t b$$
$$where$$
$$W^t = diag(w_1^t, w_2^t, \ldots, w_N^t), \quad A = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_N & 1 \end{bmatrix}, \quad b = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

As iterating until $\left\| \mathbf{x}^{t+1} - \mathbf{x}^t \right\| < tol$ (tol is an error tolerance), $\mathbf{x} = (a, b)^T$ can be obtained.

2.1.3 LP WITH L_1 NORM

The line fitting problem with L_1 norm:

$$\arg\min_{\mathbf{x}} \sum_{i=1}^{N} d(\mathbf{y}_i, \mathbf{x}) = \arg\min_{\mathbf{x}} \left\| A\mathbf{x} - b \right\|_1 = \arg\min_{\mathbf{x}} \sum_{i=1}^{N} \left| A_i\mathbf{x} - b_i \right|$$

$$where$$

$$A_i = \begin{bmatrix} x_i & 1 \end{bmatrix}, \quad b_i = \begin{bmatrix} y_i \end{bmatrix}$$

It can be solved by linear programming algorithm:

$$\min_{\mathbf{x}, t_1, \ldots, t_N} \sum_{i=1}^{N} t_i$$

$$s.t. \quad \left| A_i\mathbf{x} - b_i \right| \le t_i, \quad i = 1, \ldots, N$$

The MATLAB function **linprog(A, b, f)** can be used:

$$\mathbf{A} = \left[\begin{array}{cc|c} -x_1 & -1 & \\ -x_2 & -1 & \\ \vdots & \vdots & -\mathbb{1}_{N\times N} \\ -x_N & -1 & \\ \hline x_1 & 1 & \\ x_2 & 1 & \\ \vdots & \vdots & -\mathbb{1}_{N\times N} \\ x_N & 1 & \end{array}\right], \quad \mathbf{b} = \begin{bmatrix} -y_1 \\ -y_2 \\ \vdots \\ -y_N \\ \hline y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 1 \end{bmatrix}, \quad \mathbf{x'} = \begin{bmatrix} a \\ b \\ t_1 \\ \vdots \\ t_N \end{bmatrix}$$

As **linprog(A, b, f)** solves $\min_{\mathbf{x'}} \mathbf{f}^T\mathbf{x'}$ such that $\mathbf{A}\,\mathbf{x} \le \mathbf{b}$, the line parameter $\mathbf{x} = (a, b)^T$ can be obtained from **x'**.

2.1.4 LP WITH L_∞ NORM

The line fitting problem with L_∞ norm:

$$\arg\min_{\mathbf{x}} \left\| A\mathbf{x} - b \right\|_\infty = \arg\min_{\mathbf{x}} \max_{i=1,\ldots,N} \left| A_i\mathbf{x} - b_i \right|$$

$$where$$

$$A_i = \begin{bmatrix} x_i & 1 \end{bmatrix}, \quad b_i = \begin{bmatrix} y_i \end{bmatrix}$$

It can be solved by linear programming algorithm:

$$\min_{\mathbf{x},t} t$$
$$s.t. \quad \left|A_i\mathbf{x} - b_i\right| \leq t, \quad i = 1,\ldots,N$$

Thus, **A**, **B**, **f** and **x'** for **linprog(A, b, f)** are as follows:

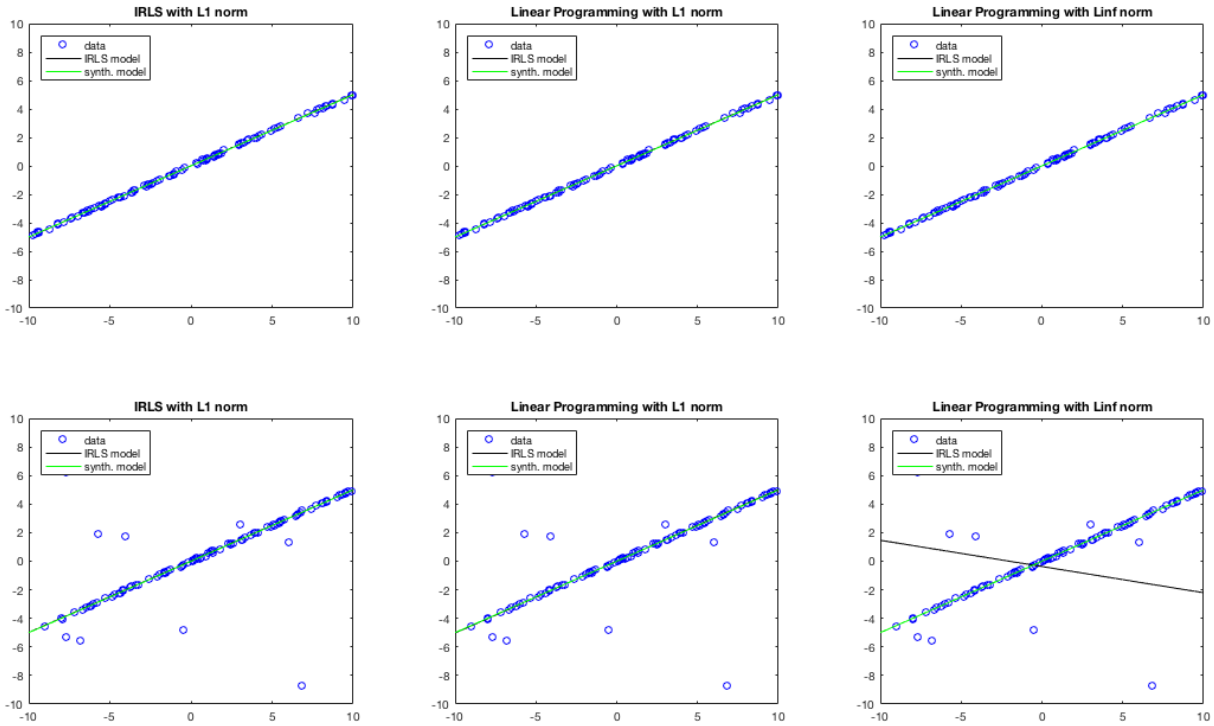
$$\mathbf{A} = \left[\begin{array}{ccc} -x_1 & -1 & -1 \\ -x_2 & -1 & -1 \\ \vdots & \vdots & \vdots \\ -x_N & -1 & -1 \\ \hline x_1 & 1 & -1 \\ x_2 & 1 & -1 \\ \vdots & \vdots & \vdots \\ x_N & 1 & -1 \end{array}\right], \quad \mathbf{b} = \left[\begin{array}{c} -y_1 \\ -y_2 \\ \vdots \\ -y_N \\ \hline y_1 \\ y_2 \\ \vdots \\ y_N \end{array}\right], \quad \mathbf{f} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{x'} = \begin{bmatrix} a \\ b \\ t \end{bmatrix}$$

The line parameter $\mathbf{x} = (a, b)^T$ can be obtained from **x'**

2.2 RUNNING

Only changing model parameters (i.e. variables a, b and noise_radius) is allowed to guarantee proper execution. If the model parameter combination is not feasible(e.g. out of domain bound), error would be returned.

2.3 Result



The result of IRLS with L_1 norm, LP with L_1 norm and LP with L_∞ for line fitting with different outlier ratios is as the plot above. The synthetic line model is $y = \frac{1}{2}x$.

The table shows the parameters and total cost for the data set of the fitted line model:

r	0%	0%	0%
algorithm	IRLS with L_1	LP with L_1	LP with L_∞
fitted model	(0.5015, 0.0069)	(0.5010, 0.0118)	(0.5006, -0.0018)
cost	2.1312	4.2592	0.09517

r	10%	10%	10%
algorithm	IRLS with L_1	LP with L_1	LP with L_∞
fitted model	(0.4996, 0.0128)	(0.5026, 0.0026)	(-0.1835, -0.3687)
cost	29.3508	58.573	7.0647

2.4 Discussion

Comparing IRSL with L_1, LP with L_1 and LP with L_∞:

- For $r = 0$, three algorithms shows equally good results.
- For $r = 10$, **algorithms using L_1 norm are robust to outliers but L_∞ is not.**

- Using L_∞, optimization algorithm(here, LP) tries to minimize only maximum value of $d(\mathbf{y}_i, \mathbf{x})$ for $i = 1, \ldots, N$. Thus, it is not robust to outliers.
- In other words, using L_∞ is the extreme case of over-penalizing outlier. Over-penalizing outliers tends to be worse as using L_q norm with larger q.

- Concerning running time, tried to check running time of each algorithms with $r = 0, 10, 70$ and the result is as follows:

r	0%	10%	70%
IRLS with L_1	0.02116	0.027398	0.056931
LP with L_1	1.3543	0.38858	0.25859
LP with L_∞	0.34789	0.22952	0.19325

For line fitting problem, IRLS converges fast even for high value of r.