Robust Maximum Likelihood Estimation of Lines From Points

3D Computer Vision – Lab Session Task

(CTU FEE subjects B4M33TDV, BE4M33TDV, XP33VID)

Martin Matoušek and Jaroslav Moravec

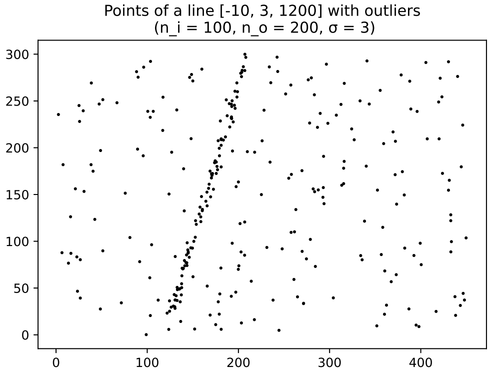
September 2024

CENTER FOR MACHINE PERCEPTION

Fitting Model in Noisy Data

A 2D line $[a, b, c] = [-10, 3, 1200]$ generates a set of 100 points (inliers), which is then corrupted by a set of 200 uniformly distributed random points not belonging to the line (outliers). There are three sets of points linefit_1.txt, linefit_2.txt, linefit_3.txt generated with the gaussian noise $\sigma = 1$, $\sigma = 2$, and $\sigma = 3$, respectively.

Task: Find the parameters of the line from the set of 300 points.



Hint: Python

```
x = np.loadtxt('linefit_1.txt').T
```

Points on a Line

The *Euclidean distance* between

$$\text{a line } \underline{\mathbf{l}} = \begin{bmatrix} l_1 & l_2 & l_3 \end{bmatrix}^\top \quad \text{and a point } \underline{\mathbf{x}} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^\top$$

is defined as follows:

$$\varepsilon_{\underline{\mathbf{l}}}(\underline{\mathbf{x}}) = \left| \frac{\underline{\mathbf{l}}^\top \underline{\mathbf{x}}}{\left(\sqrt{l_1^2 + l_2^2}\right) x_3} \right| . \tag{1}$$

For normalized parametrisation:

$$\underline{\mathbf{l}} = \begin{bmatrix} \mathbf{n}^\top & d \end{bmatrix}^\top \quad |\mathbf{n}| = 1 \quad \text{and} \quad \underline{\mathbf{x}} = \begin{bmatrix} \mathbf{u}^\top & 1 \end{bmatrix}^\top$$

the error is:

$$\varepsilon_{\underline{\mathbf{l}}}(\mathbf{u}) = \left| \mathbf{n}^\top \mathbf{u} + d \right| .$$

Least Squares Regression of Line from Points

Minimize sum of squared point-line distances w.r.t. line parameters: $\mathbf{n}^*, d^* = \arg\min C(\mathbf{n}, d)$

$$C(\mathbf{n}, d) = \sum_{i=1}^{N} \left(\varepsilon_{\underline{1}}(\mathbf{u}_i)\right)^2 = \sum_{i=1}^{N} \left(\mathbf{n}^\top \mathbf{u}_i + d\right)^2 = \sum_{i=1}^{N} (\mathbf{n}^\top \mathbf{u}_i)^2 + 2d\mathbf{n}^\top \sum_{i=1}^{N} \mathbf{u}_i + Nd^2$$

1. find d – zero first derivative gives extrema

$$\frac{\partial C}{\partial d} = 2\mathbf{n}^\top \sum_{i=1}^{N} \mathbf{u}_i + 2Nd = 0 \rightarrow \boxed{d = -\mathbf{n}^\top \boldsymbol{\mu}_u} \quad \text{where} \quad \boldsymbol{\mu}_u = \frac{1}{N} \sum_{i=1}^{N} \mathbf{u}_i \quad \text{(centroid)}$$

2. let $\mathbf{u}'_i = \mathbf{u}_i - \boldsymbol{\mu}_u$, substitute d and $\mathbf{u}_i = \mathbf{u}'_i + \boldsymbol{\mu}_u$ to $C(\mathbf{n}, d)$

$$\begin{aligned} C(\mathbf{n}, d) &= \sum \left(\mathbf{n}^\top (\mathbf{u}'_i + \boldsymbol{\mu}_u) - \mathbf{n}^\top \mu_u\right)^2 = \sum \left(\mathbf{n}^\top \mathbf{u}'_i\right)^2 = \mathbf{n}^\top \left(\sum \mathbf{u}'_i \mathbf{u}'^\top_i\right) \mathbf{n} = \\ &= \mathbf{n}^\top \mathbf{A}^\top \mathbf{A} \mathbf{n} \quad \text{where} \quad \boxed{\mathbf{A}^\top = [\dots, (\mathbf{u}_i - \boldsymbol{\mu}_u), \dots]} \quad \text{(stacked points)} \end{aligned}$$

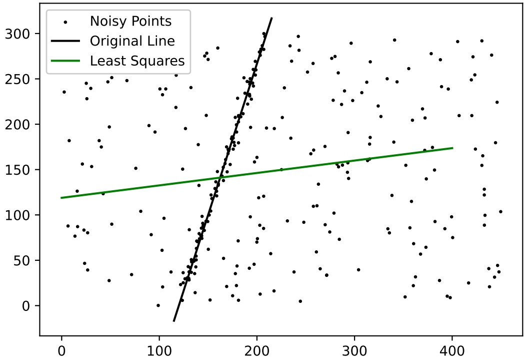
Let $\boxed{\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^\top}$ (singular value decomposition), then $C(\mathbf{n}, d) = \mathbf{n}^\top \mathbf{V}\mathbf{D}^2\mathbf{V}^\top \mathbf{n}$, and minimum is for $\mathbf{n}^\top \mathbf{V} = \begin{bmatrix} 0 & 1 \end{bmatrix}$ (must be unit vector, so select the smallest singular value)

$$\mathbf{V} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{bmatrix} \quad \rightarrow \quad \boxed{\mathbf{n} = \mathbf{v}_2}$$

Alternative: instead of computing s.v.d. of $N \times 2$ matrix $\mathbf{A}$, compute eigen values/vectors of 2×2 matrix $\mathbf{A}^\top \mathbf{A}$. The solution $\mathbf{n}$ is then eigenvector corresponding to the smallest eigenvalue.

Using Non-robust Method

For such kind of data (two thirds of points are outliers), non-robust estimation method, e.g. regression by least squares, cannot be used. First, verify this assumption by estimating the line using least squares.



Hint: Python

```
mu = x.mean(axis=1).reshape((1,2))
u, _, _ = np.linalg.svd(x - mu.T)
n = u[:, -1]
return np.hstack((n, -mu @ n))
```

Robust Estimation – RANSAC

A golden standard method for robust (i.e. in the presence of outliers) estimation is **RANSAC** algorithm, which works as follows:

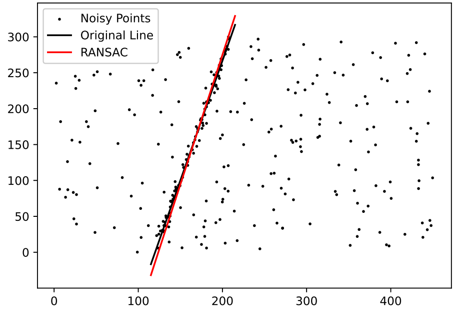
$\{x_1, \dots, x_N\}$ – the data set (points, correspondences, etc.)
N – number of data elements
n – minimum number of data elements needed to estimate the model
$\varepsilon_M(x_i)$ – error of a data element x_i with respect to the model M
parameters: threshold θ, probability P

Algorithm:

```
1. init: $k \leftarrow 0$, best-support $\leftarrow 0$
2. iteration: $k \leftarrow k + 1$
3.     sample: randomly draw $n$ data points
4.     hypothesis: compute a model $M_k$ from the sample
5.     evaluate error $\varepsilon_{M_k}(x_i)$ of every data point in the set w.r.t. $M_k$
6.     inliers: $\mathcal{I} = \left\{x_i \mid \varepsilon_{M_k}(x_i) < \theta\right\}$ (data with error smaller then threshold)
7.     support $\leftarrow N_{\mathcal{I}}$,   $N_{\mathcal{I}} = |\mathcal{I}|$ (number of inliers)
8.     update: if support $>$ best-support
           best-support $\leftarrow$ support
           $M^* \leftarrow M_k$
           $N_{\mathsf{max}} = \frac{\log(1-P)}{\log(1-w^n)}$ (stopping criterion),   where   $w = \frac{N_{\mathcal{I}}}{N}$ (inlier ratio)
9.     terminate with $M^*$ if $k > N_{\mathsf{max}}$, otherwise repeat from step 2
```

Robust Estimation – RANSAC

In this task, we randomly draw two points $\mathbf{a}$ and $\mathbf{b}$ to compute a model (line) $\underline{\mathbf{l}} = \underline{\mathbf{a}} \times \underline{\mathbf{b}}$. Use the *Euclidean distance* from (1) to estimate the ortogonal distance between the model $\underline{\mathbf{l}}$ and points in the dataset. Select a reasonable threshold θ for inliers and verify that RANSAC provides a much better estimate than the non-robust method.



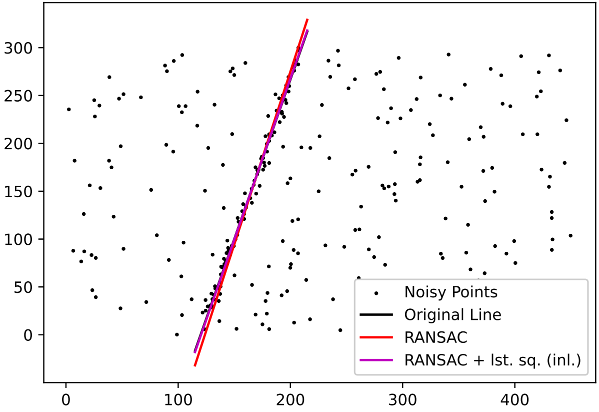
Hint: Python

```
rng = np.random.default_rng()

i = rng.choice(n, 2, replace=False)
```

Optimization of RANSAC Model

If more acurate result is needed, additional optimisation can applied to the result of RANSAC, i.e. least squares regression using the RANSAC inliers only.



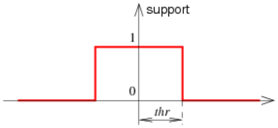
Robust Estimation – MLESAC

One of the common modifications of RANSAC method employs support function derived as a maximum likelihood estimate (MLE), instead of a standard zero-one box function

- Standard RANSAC uses 0-1 (box) support function

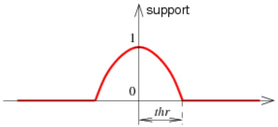
$$s_i = \begin{cases} 1 & \text{if } \varepsilon(x_i) \leq \theta \\ 0 & \text{otherwise} \end{cases}$$

$$\text{support} = \sum_i s_i$$



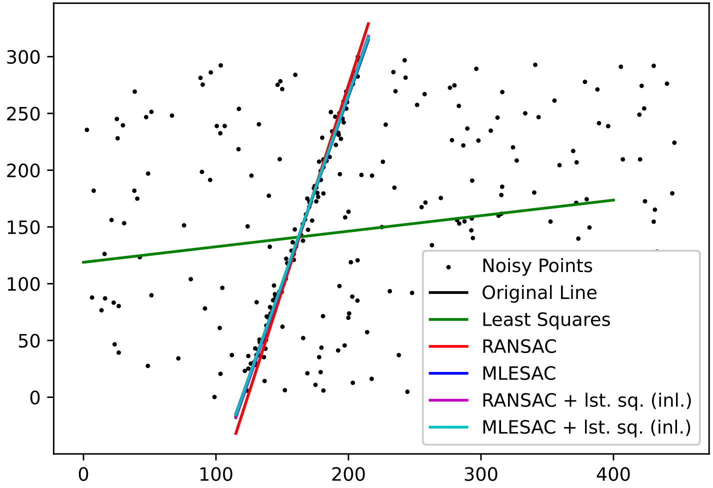
- MLESAC – modification of support for maximum likelihood error on inliers

$$s_i = \begin{cases} 1 - \frac{\varepsilon(x_i)^2}{\theta^2} & \text{if } \varepsilon(x_i) \leq \theta \\ 0 & \text{otherwise} \end{cases}$$



Show MLESAC (and MLESAC with local optimisation) results alongside the other algorithms.

Expected Result After One Run

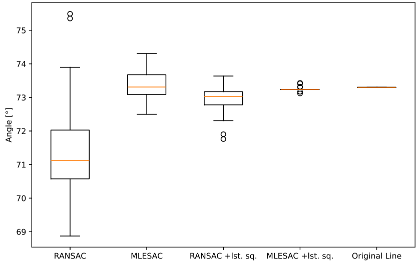


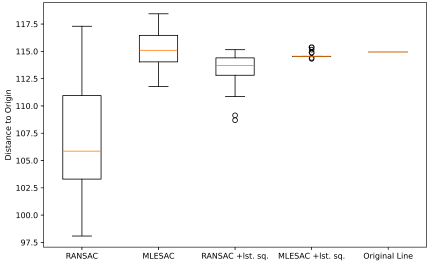
Analysis: Multiple Runs

For a robustly estimated line $\underline{\mathbf{l}} = \begin{bmatrix} l_1 & l_2 & l_3 \end{bmatrix}^\top$ estimate its angle α and distance to orgin d as follows:

$$\alpha = \frac{180}{\pi}\text{atan2}\left(-\text{sgn}(l_3) \cdot l_1, \text{sgn}(l_3) \cdot l_2\right) \quad \text{and} \quad d = \frac{|l_3|}{\sqrt{l_1^2 + l_2^2}}.$$

Execute all four robust methods (RANSAC and MLESAC, with and without optimisation) 100 times and collect angles α and distances to origin d. Visualise the results of each method in a boxplot alongside the ground truth value of the original line.





Hint: Boxplot Visualisation in Python

Implement a function

$$\text{a, d} = \texttt{estimate\_a\_d(l)},$$

which estimates the angle and distance to origin for a given line $\underline{\mathbf{l}}$. Collect the angles and distances of estimated lines over one hundred runs of each robust method into lists (or `numpy` arrays). Visulise the data as follows:

```
gt_a, gt_d = estimate_a_d([-10.0, 3.0, 1200.0])
labels = ['RANSAC', 'MLESAC', 'RANSAC +lst. sq.',
          'MLESAC +lst. sq.', 'Original Line']
plt.subplot(1, 2, 1)
plt.boxplot([ransac_a, mlesac_a, ransac_lst_sq_a, mlesac_lst_sq_a, gt_a])
plt.xticks([1, 2, 3, 4, 5], labels)
plt.ylabel('Angle [°]')
plt.subplot(1, 2, 2)
plt.boxplot([ransac_d, mlesac_d, ransac_lst_sq_d, mlesac_lst_sq_d, gt_d])
plt.xticks([1, 2, 3, 4, 5], labels)
plt.ylabel('Distance to Origin')
plt.show()
```