Robust Estimator Report

Matsuoka Keito 46243187

April 25, 2024

1 LS estimator

A line of least squares was applied to point group D. A diagram of the line of least squares for point group D is shown in 1.

The program at this time is shown 1. The least-squares process is defined after line 11 of the program. First, the mean and variance-covariance matrices of the point cloud are computed. Next, the smallest eigenvalue of the matrix and the corresponding vector are found and c is computed.

Listing 1 LS estimator

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # points = np.loadtxt("original_point_set.txt")
5 points = np.loadtxt("point_set_with_outliers.txt")
6 x = points[0,:]
7 y = points[1,:]
8 X_MIN = 0
9 X_MAX = 10
10
11 def least_square_estimation(points):
12     mean_point = np.average(points,axis=1)
13     variance = np.cov(points)
14     eigenvalues, eigenvectors = np.linalg.eig(variance)
15     smallest_eigenvalue = np.min(eigenvalues)
16     index_of_smallest_eigenvalue = np.argmin(eigenvalues)
17     smallest_eigenvector = eigenvectors[:, index_of_smallest_eigenvalue]
18     c = - np.dot(smallest_eigenvector,mean_point)
19     return smallest_eigenvector, c
20
21 def line_func(x,n,c):
22     n1,n2=n[0],n[1]
23     y = - (n1*x+c)/n2
24     return y
25
26 smallest_eigenvector, c = least_square_estimation(points)
27
28 plt.scatter(x,y,c='k')
29 plt.plot([X_MIN, X_MAX], [line_func(X_MIN,smallest_eigenvector,c),line_func(X_MAX,
      smallest_eigenvector,c)])
30 plt.show()
```

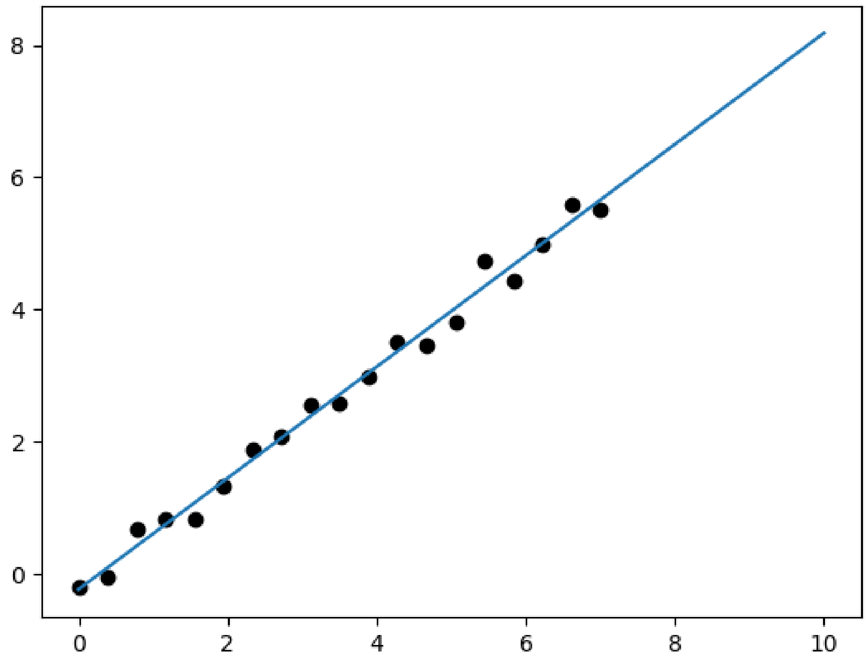

Fig. 1

2 LS estimator with outliers

The same program was run for point cloud DoI containing outliers. The result is shown in 2. The estimated line no longer fits the point cloud well due to the influence of outliers.

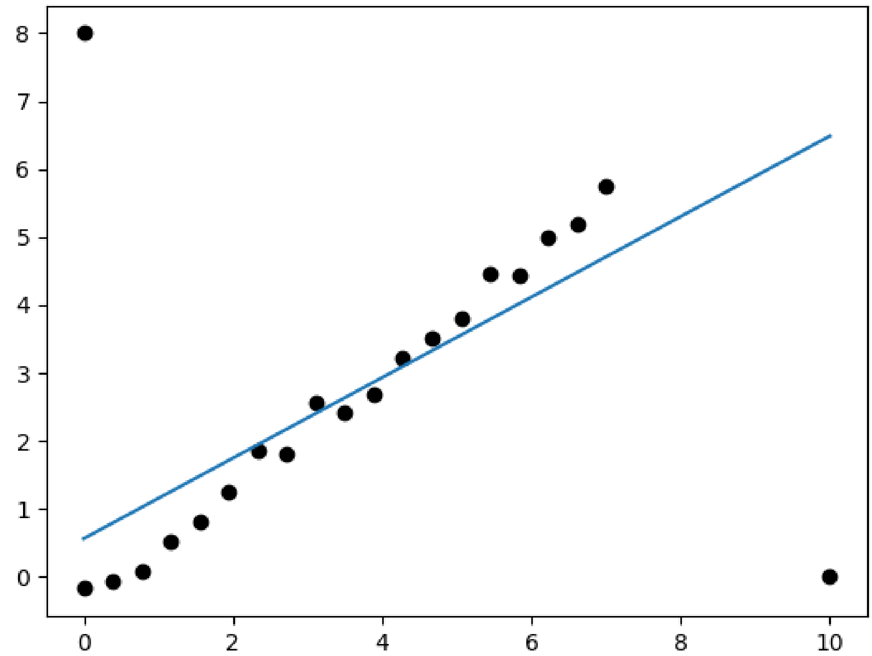

Fig. 2

3 IRLS estimator with outliers

The line L_{RE} estimated using the IRLS algorithm is shown in 3. The program used in this study is shown below. It can be seen that the estimated line by IRLS fits the point cloud without being affected by outliers. Lines 19 to 85 define the IRLS process; lines 109 to 117 perform the estimation by iteration. The process was repeated until there was almost no change in the slope of the output graph.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  def line_func(x,n,c):
5      n1,n2=n[0],n[1]
6      y = - (n1*x+c)/n2
7      return y
8
9  def gm_estimater(erros, e):
10     sigma = np.std(erros)
11     rho = e**2/(sigma**2+e**2)
12     return rho
13
```

```
def weight_func_for_gm(errors, e):
    sigma = np.std(errors)
    weight = 2/(1+e**2/sigma**2)**2
    return weight

class RobustLineEstimator_GM:
    def __init__(self, points):
        self.transposed_points = np.transpose(points)

    def _error_func(self, n, c, x1, x2):
        n1, n2 = n[0], n[1]
        error = n1 * x1 + n2 * x2 + c
        return error

    def calc_error(self, n, c):
        errors = []
        x = self.transposed_points
        for x_k in x:
            x1, x2 = x_k[0], x_k[1]
            error = self._error_func(n, c, x1, x2)
            errors.append(error)
        return errors

    def weighted_points(self, errors):
        weighted_points = np.array([weight_func_for_gm(errors, errors[k]) * self.transposed_points[k] for k in range(len(self.transposed_points))])
        return weighted_points

    def weighted_errors(self, errors):
        weighted_errors = [weight_func_for_gm(errors, errors[k]) for k in range(len(errors))]
        return weighted_errors

    def calc_weighted_covariance_matrix(self,errors):
        weighted_points = self.weighted_points(errors)
        sum_of_weightedpoints = np.sum(weighted_points, axis=0)

        weighted_errors = self.weighted_errors(errors)
        # print(f"weighted_errors:{weighted_errors}")

        sum_of_weighted_errors = np.sum(weighted_errors)

        weighted_mean = sum_of_weightedpoints / sum_of_weighted_errors

        deviation_from_mean = weighted_points - weighted_mean

        for k in range(len(errors)):
            sumof_weighted_deviation_matrix = weighted_errors[k] * (deviation_from_mean[k].reshape(2,1) @ deviation_from_mean[k].reshape(1,2))

        weighted_covariance_matrix = sumof_weighted_deviation_matrix / sum_of_weighted_errors

        return weighted_mean, weighted_covariance_matrix

    def calc_smallest_eigenvector(self, weighted_covariance_matrix):
        eigenvalues, eigenvectors = np.linalg.eig(weighted_covariance_matrix)

        smallest_eigenvalue = np.min(eigenvalues)

        index_of_smallest_eigenvalue = np.argmin(eigenvalues)

        smallest_eigenvector = eigenvectors[:, index_of_smallest_eigenvalue]

```

```python
        return smallest_eigenvector

    def calc_offset(self, smallest_eigenvector, weighted_mean):
        self.c = - np.dot(smallest_eigenvector,weighted_mean)
        return - np.dot(smallest_eigenvector,weighted_mean)

    def calc_estimation_line(self, smallest_eigenvector, c):
        errors = self.calc_error(smallest_eigenvector, c)
        weighted_mean ,weighted_covariance_matrix = self.calc_weighted_covariance_matrix(errors)
        smallest_eigenvector = self.calc_smallest_eigenvector(weighted_covariance_matrix)
        c = self.calc_offset(smallest_eigenvector, weighted_mean)
        return smallest_eigenvector, c

if __name__ == "__main__":
    points = np.loadtxt("point_set_with_outliers.txt")

    #used for plotting
    x = points[0,:]
    y = points[1,:]
    X_MIN = 0
    X_MAX = 10

    #Least square method
    mean_point = np.average(points,axis=1)

    variance = np.cov(points)

    eigenvalues, eigenvectors = np.linalg.eig(variance)
    smallest_eigenvalue = np.min(eigenvalues)
    index_of_smallest_eigenvalue = np.argmin(eigenvalues)

    ls_smallest_eigenvector = eigenvectors[:, index_of_smallest_eigenvalue]
    ls_c = - np.dot(ls_smallest_eigenvector,mean_point)

    #GM estimation
    estimater = RobustLineEstimator_GM(points)
    for i in range(10):
        if i == 0:
            smallest_eigenvector = ls_smallest_eigenvector
            c = ls_c
            lines = []
        else:
            smallest_eigenvector, c = estimater.calc_estimation_line(smallest_eigenvector, c)
            print(f"i:{i} \n smallest_eigenvector:{smallest_eigenvector} \n c:{c}")
    plt.scatter(x,y,c='k')
    plt.plot([X_MIN, X_MAX], [line_func(X_MIN, smallest_eigenvector, c),line_func(X_MAX, smallest_eigenvector, c)])
    plt.show()
    plt.close()
```

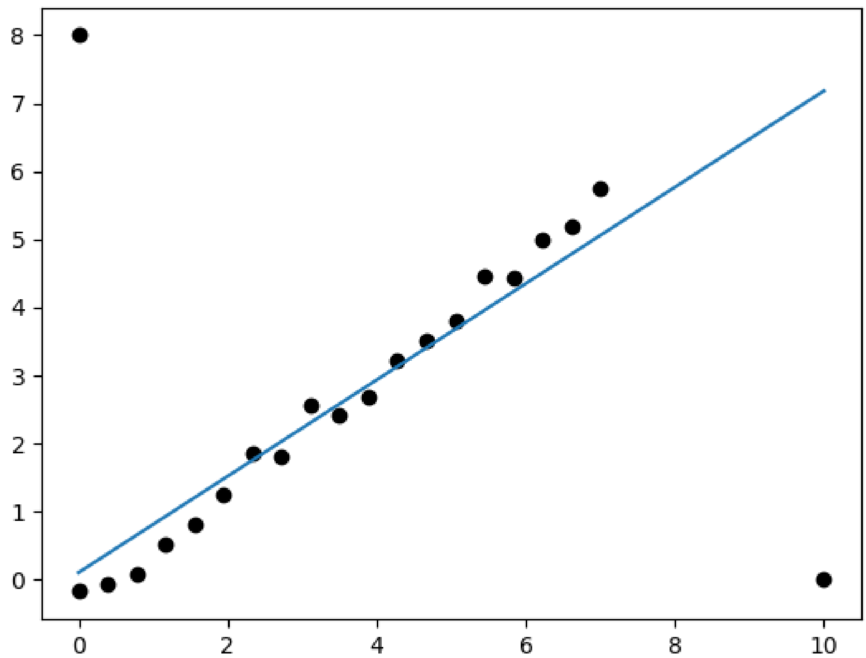

Fig. 3