A Multiple-Line Fitting Algorithm Without Initialization

Yan Guo

Abstract: The commonest way to fit multiple lines is to use methods incorporate the EM algorithm. However, the EM algorithm dose not guarantee to give the global optimum result since it is based on initialization of parameters. With an inappropriate initialization, the EM algorithm can generate lines that are obviously wrong according to human eyes. The goal of the method described in this paper is to find a way of multiple-line fitting when given set of points without having to guess the initial parameters. The main part of this algorithm involves some heuristic divide and conquers methodology; it also involves developing a scoring function for a line. The results show that in many situations, this algorithm will produce better results comparing to the EM algorithm.

Introduction. The least square fitting method, also known as simple linear regression in statistics, may be the most widely used way to fit a straight line from the given data points. However, EM algorithm is the widely used when come to multiple line fitting. All of the general EM algorithms require a guessing of the initial parameters. The rest of the algorithm is based on the initial parameters. If the initial parameters are poor, the later iterations of the EM algorithm might produce inaccurate results. In the multiple-line fitting case, prior to the iteration steps, one has to guess approximately how many lines are there and which points belong to which line. If some of the points are put on wrong lines when initializing, it's possible that the results will not be corrected during later iterations. Furthermore, if the number of lines is wrong, there is no way for the EM algorithm to detect or correct such error. Usually, it is not easy to guess the correct number of lines, thus it is unlikely to obtain correct results by applying the EM algorithm without optimizing the initialization. Figure 1. shows some poor results from the EM line fitting algorithm.

Figure 1.

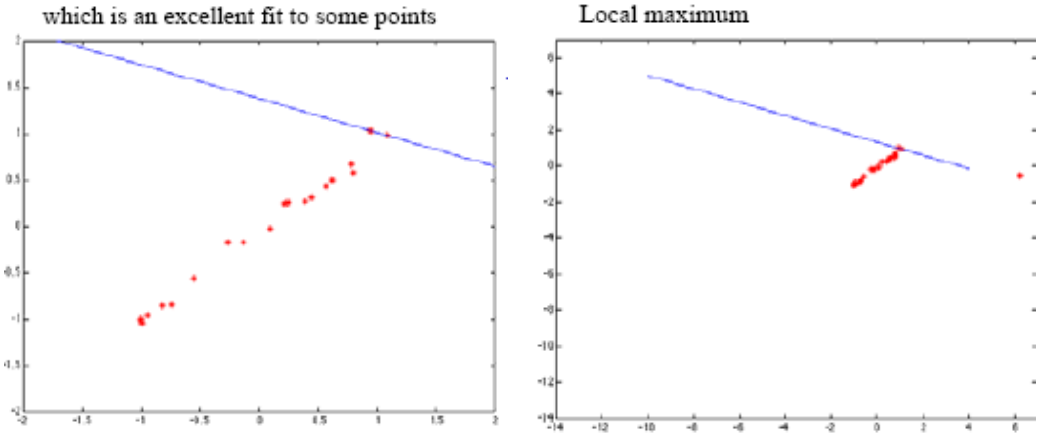


Fig1a Fig1b

Fig1a and Fig1b shows the result suffered from local minimum, even with correct guess of number of lines.

The motivation of this paper is to develop a method that does not require guessing of the initial values about the number of lines, and which points belong to which line.

Method and Algorithm. In order to avoid the initialization step, one needs to find a way to estimate how likely that some certain points are form a certain line. Therefore a function to describe the likelihood of each fitted line to be a true line is wanted. In simple linear regression, R^2 (R-Square) statistic is used to measure the percentage of the variation in the data is explained by the estimating equation. The following statistics can be calculated if given a set of data of N points represented in (X_i, Y_i) format:

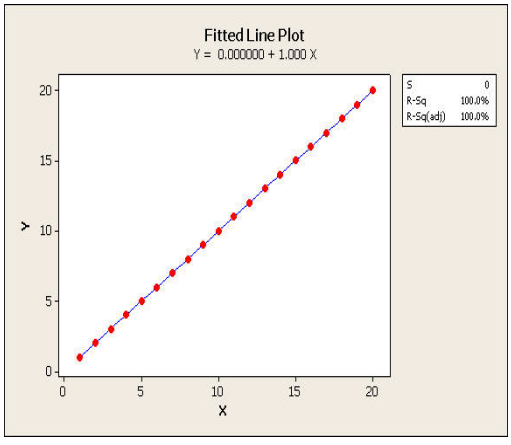
$$SS_{total} = \sum_{i=1}^{n} (Y_i - \bar{Y})^2 \qquad SS_{regression} = \sum_{i=1}^{n} (\hat{Y}_i - \bar{Y})^2$$

$$MS_{residual} = \frac{SS_{residual}}{D.F.} \qquad SS_{residual} = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$$

($\bar{Y}$ is the mean of all input Y's, $\hat{Y}_i$ is the $\hat{Y}_i^{th}$ value from regression equation $Y = \hat{\beta}_0 + \hat{\beta}_1 X$, Y_i is the Y_i^{th} data from the input, D.F. is the degree of freedom, which is always 2 in the simple linear regression, $\hat{\beta}_0$ is the estimated intercept, and $\hat{\beta}_1$ is the estimated slope, both line parameters are estimated using the least square method)

The R-square statistic (R^2) is defined as $R^2 = \frac{SS_{regression}}{SS_{total}}$. This is a proportion between 0 and 1. R^2 represents the percentage of the variation explained by the regression equation $d = \frac{ax + by + c}{\sqrt{a^2 + b^2}}$. Figure 2 shows two plots in terms of line fitting.

Figure 2 (figure 2-7 are generated using Minitab)



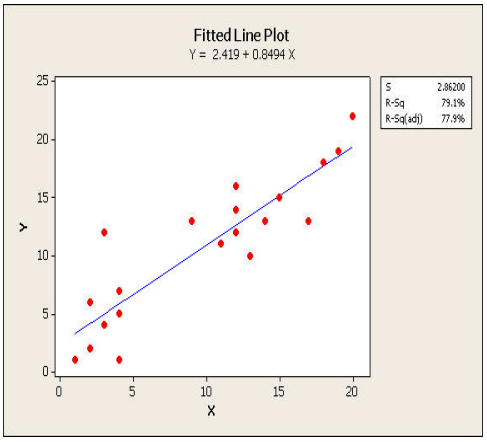


Fig 2a Fig 2b

Fig.2a shows a perfect line fitting as all points are exactly on the fitted straight line and so the R^2 statistic in this case is one. Fig.2b shows a line with R^2 =0.79. Most of the points are around the fitted line but not exactly on the line. The higher R^2 indicates the less variability and the points in general are closer to the fitted line and therefore the points are more likely to actually form a straight line.

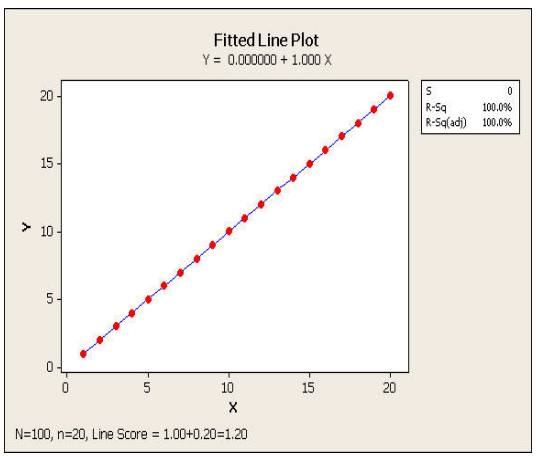
Notice that if the input data only contains 2 points, since 2 points can form a line, the R^2 statistic will always be one. For multiple-line fitting, we want a better way to indicate the likelihood of a certain set of points to actually form a line. We introduce the line score S, which is used to represent how likely that a set of points forms a line. Line Score is defined as:

$$S = R^2 + \frac{n}{N} \ ,$$

N is the total points in the input, n is the points considered in current line.

Since this line score takes the proportion of the sample size of the set of points considered in the line fitting out of overall sample size (total number of points) into considered in addition to the R-square statistic, it gives a better indication of the likelihood of this set of points actually forms a line. Line score S is the sum of 2 probabilities, thus $0 < LineScore \le 2$. In multiple-lines fitting, higher score indicates a better-fitted line. Figure 3. shows some examples of line-fitting using the line score statistic as the criteria.

Figure 3.



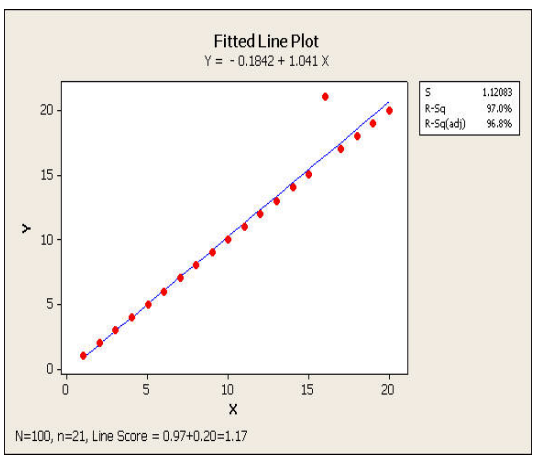
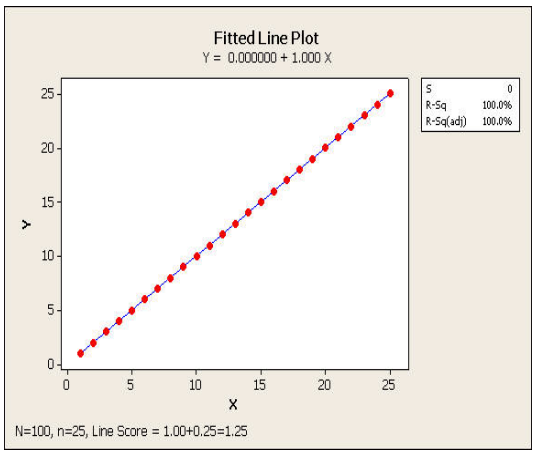


Fig3a Fig3b



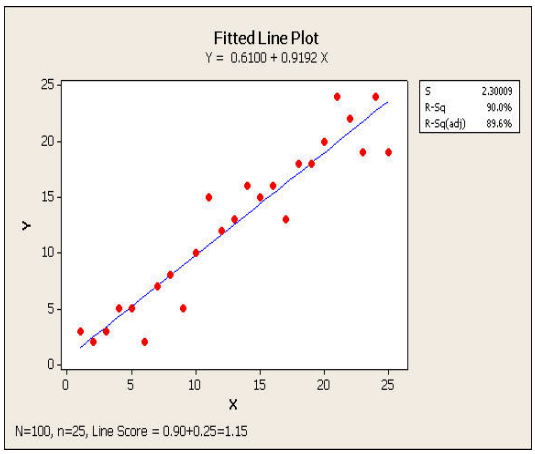


Fig3c Fig3d

Assuming there are totally 75 points in the input (N=75). The 20 points (n=20) in Fig3a form a perfect line and the line score is 1.20. Fig3b is a line also formed with 20 points (n=20) but it has one outlier, the line score is 1.17. The points in Fig3c show a perfect line (n=25) and it has a line score 1.25. Fig3d fits a line with 25 points (n=25) but some of them are noise, the line score is 1.15. These plots show that the line score can evaluate line fitting given a set of points easily and reasonably.

To avoid the initialization step, a rather heuristic approach is used here. Consider all of the input points located with X and Y coordinates on a finite two-dimensional space. Then we can cut the space into some small spaces in a way preserves the line shape properties as well as possible. To do this, the space is cut into some stripes as a stripe is thought to have the best chance of capturing a line. Then we apply the algorithm to the stripes. The algorithm is defined as following:

1. Divide the area into certain finite area of stripes (5 or 10 usually is sufficient).
2. Calculate the line score for each stripe.
3. Pick the stripe with highest line score and filter out outliers.
4. Redefine the stripe area with the fitted line by putting the picked stripe in the reference frame with intercept 0 and slope 1.
5. Recalculate the line score with the points inside the stripe.
6. If the new line score is higher than the current highest value, continue to next step, otherwise go back to step 3, and pick the next highest score stripe.

7. Go to step 4 and then step 5. Repeat until no more points are getting added into the stripe.
8. Remove the points from the final stripe from the input, and repeat from step1.

Finalize the results, detecting noise etc.

Results. Figure 4. shows a very simple example of the algorithm.

Figure 4.

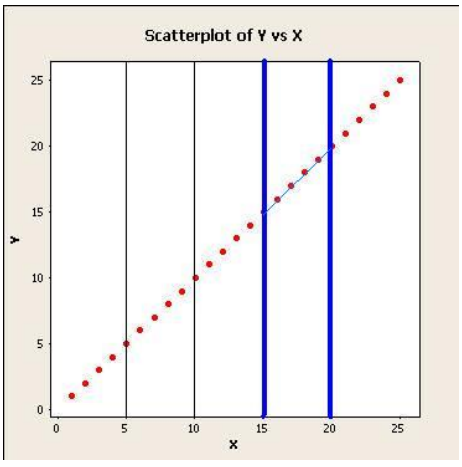


Fig4a

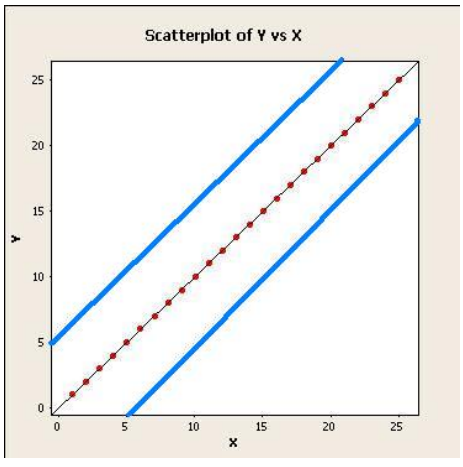


Fig4b

This is an example of applying the multiple-line fitting algorithm. The input data has 25 points, which forms a straight line. Fig4a shows the plot of the points just for demonstration. Then the space is cut into 5 stripes. Since all 5 stripes have the same line score 1.4, we can start with any of these stripes and the result is exactly the same. In this case the stripe between the highlighted blue lines are chosen (Fig4a). A line is fitted within the chosen stripe. Then we redefine the stripe as shown in Fig4b, with the line fitted in Fig4a in the middle of the stripe. To calculate the distances of points to the line within the stripe, we can apply the equation of distance between a point and a line:

$$d = \frac{ax + by + c}{\sqrt{a^2 + b^2}}$$

Then we refit the line using the new stripe to get the final line that has the line score 2 (the maximum line score). This is rather an example of ideal situation. It is only to demonstrate the basic principle of the algorithm.

Now let's look at a more complicated example. The input contains 81 points, total 3 lines, with 25 points to each line, and 6 noise points (Appendix1). The goal is to detect the 3 lines accurately using the algorithm. Figures below show in detail how the algorithm works.

Figure 5 (a).

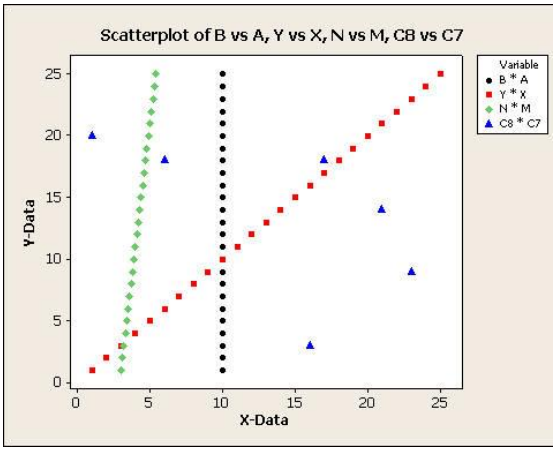


Figure a is a plot of all input points.

Green, red, and black dots indicate 3 lines; blue triangles indicate noise.

It is obvious to humans that there are 3 lines, however the computer requires sophisticated method to detect all 3 lines accurately.

Figure 5 (b).

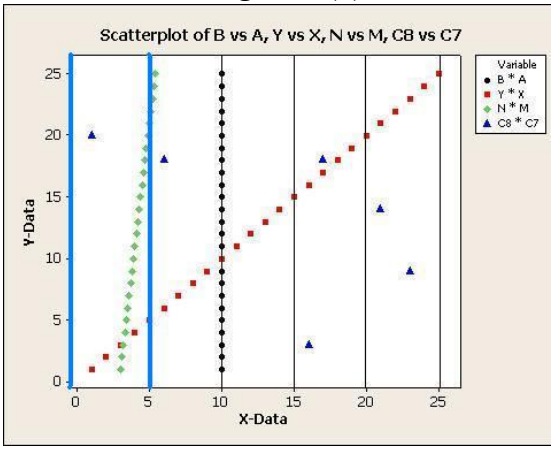


Figure b, shows the space is divided into 5 vertical stripes. (step 1 &2)

The stripe highlighted in Blue has the highest line score, 0.578

Figure 5 (c).

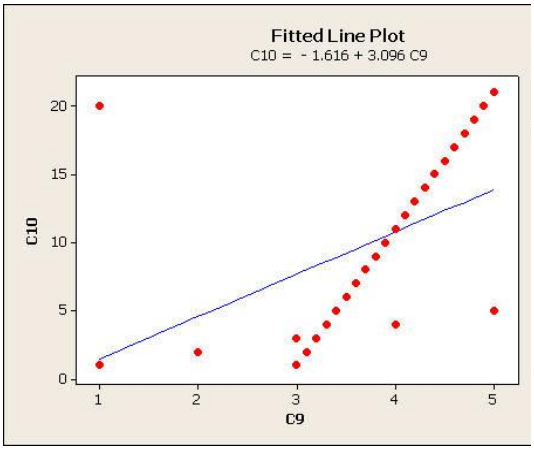


Figure c is the stripe highlighted in blue cut off from b, and stretched. (Step 3)

A line is fitted within this stripe as shown in the c.

There are some noises within this stripe and fragment of 2 lines are contained in this stripe too. To filter out the noise, 3 types of residuals are used: regular residual, student residual, and jackknife residual. The definitions of the residuals are list in appendix 2.

Figure 5 (d).

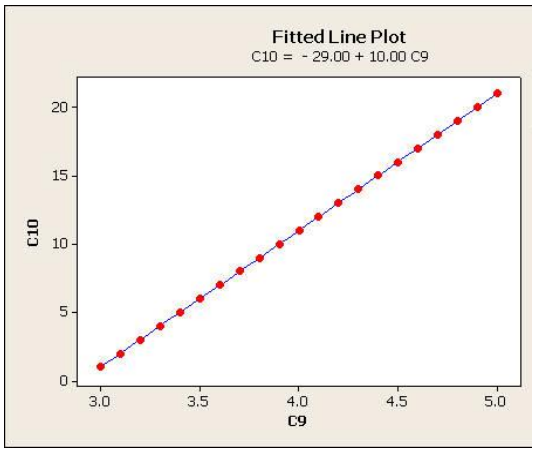


Figure d shows the stripe 1 again, this time with the outlier filtered out. (Step 3)

The outliers are filtered out based a critical value cutoff points. If any of the 3 residuals is bigger than the critical value, it is filtered out. All 3 residuals are used for conservative purpose.

Figure 5 (e).

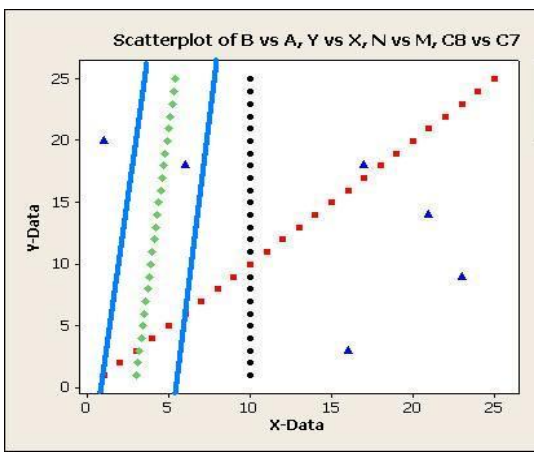


Figure e shows the recalculated stripe. (Step 3 & 4)

We fit the line within the stripe, then, remove the outliers. We should get the line represented by the green dots.

Figure 5 (f).

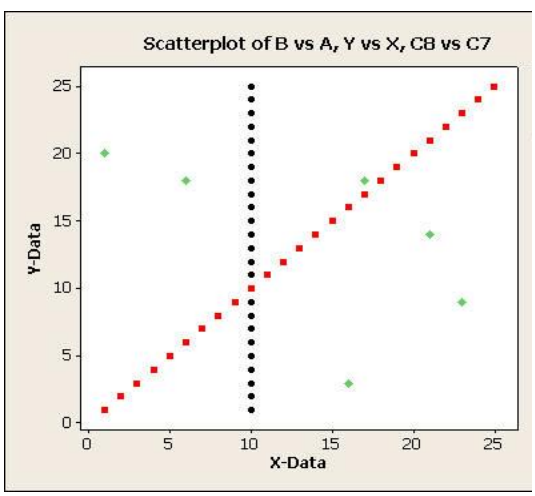


Figure f

Now we attempt to recalculate the stripe, and we find that there are no new points added to stripe, or the line score is not increasing anymore. We know that we have found 1 line. Now we remove the points on that line from the space, we get figure f.

Figure 5 (h).

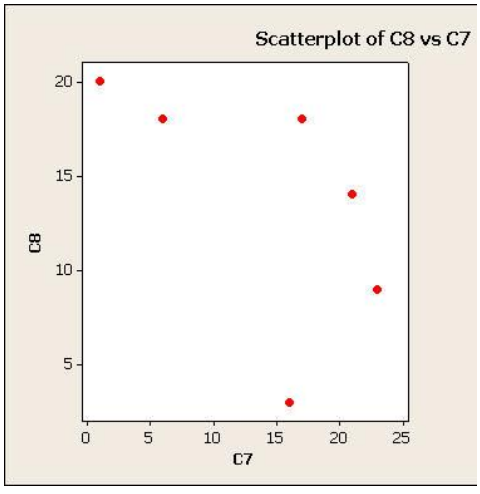


Figure h

This is the final result from applying the algorithm.

Figure 5 (g).

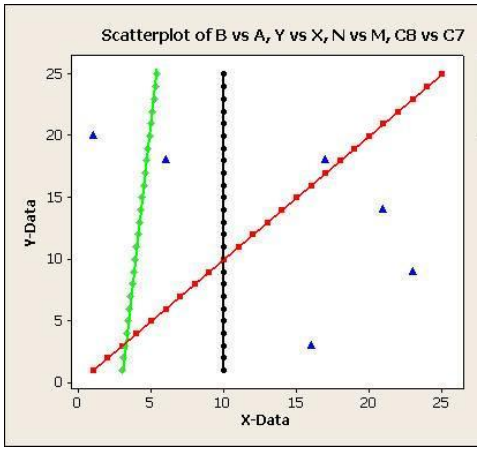
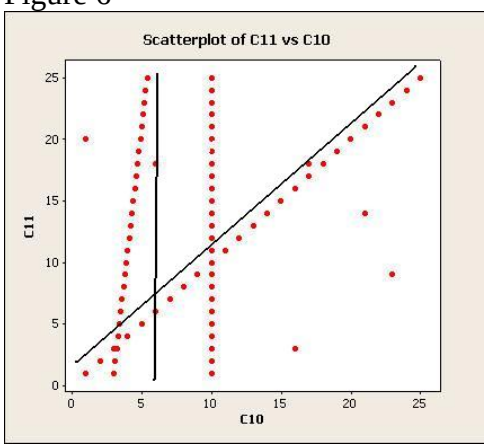


Figure g
We repeat the process, at the end we get only the noise left as shown on figure g.

At this point, we can keep fitting, however the line score of fitted noise line, will be significantly smaller than the other line scores. We then know we should stop.

With the same input, we apply the K-mean line-fitting algorithm (appendix 3) for comparison. K-mean is a special case of EM algorithm. As mentioned before, the problem with EM algorithm that we have to guess how many lines are there but there is no guarantee that such guesses are correct.

Figure 6



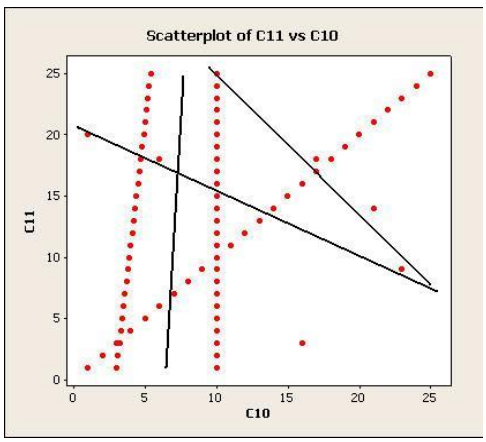


Fig 6a Fig6b

Fig 6a: K-mean algorithm applied here with 2 lines, and points are randomly assigned to each line at the beginning. Fig 6b: K mean algorithm applied with 3 lines.

K-Mean line-fitting algorithm can generate correct results if one is lucky enough to guess the correct number of lines and if the initial assignment of points is reasonable. This is the reason that the K-Mean line-fitting algorithm may generate incorrect results in this example.

Discussion. Although this multiple-line fitting algorithm is not perfect as there are some extreme cases that it cannot handle. For example, if two lines are very close to each other and almost parallel, there is a chance that the initial stripe partition does not partition them into 2 stripes. If that is the case, the algorithm will fail to recognize that those are actually two distinctive lines. Consider another case as shown in figure 7:

Figure 7

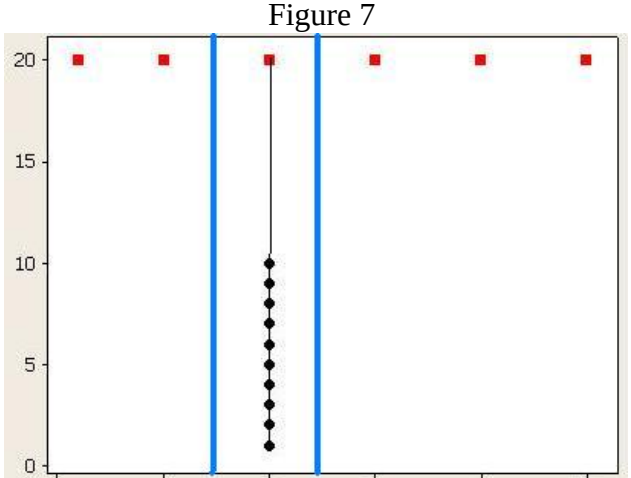


Figure 7
There are clearly 2 lines in the plot: one in red and one in black. After applying the algorithm, the black line is included in the stripe, however one red point is also included. That point is an outlier, but it is undetectable by checking the residuals since it is located on the fitted line.

To avoid this problem, first project all the points fitted by the line-fitting equation to the line. Then we assume that the distances between one point and the one next to it is normally distributed with positive values. Now we can use this normal distribution to conduct a Z test on potential outliers undetectable by checking the residual statistics. By doing this we can successfully filter out the red point from the black line.

There are several advantages of this algorithm. First and the most important, it does not require an initialization. Initial guessing of number of lines and point assignment are not needed. Without having to guess the initial line parameters, this algorithm avoids the local optimum problem. Figure 1a and 1b shows 2 examples of local optimum problem when applying the EM algorithm. However, by applying this new algorithm for this case, it will generate correct results. Secondly, with the K-Mean algorithm, all points get used even the noise. This new algorithm is able to detect the noise and filter them out.

There are also some things about this algorithm need to be improved in future. There is always a question that how to divide the space into stripes at the beginning, and to decide the direction of the stripes. In the examples showed earlier, cutting 5 vertical stripes are good enough. But in some other times, this may not be the case. This remains to be a problem to be investigated.

There are also some areas that can be improved. For example, the line score function is defined as $S = R^2 + \frac{n}{N}$. Only R^2 and $\frac{n}{N}$ are considered in this equation and a simple linear relationship between them is assumed, however, there may be more potential factors other than these two. Also we may try to weight between R^2 and $\frac{n}{N}$ to assign different proportions of these two terms.

Conclusion. The multiple-line fitting algorithm works in many of the multiple-line fitting cases. It utilized least square fitting method, which has been studied widely. Since it does not require initial guessing of the line parameters, it successfully avoids the local optimum problem in the EM algorithm. Although there are some intrinsic details need to be taken care of for this algorithm, test examples show that on average this algorithm produces more reliable results comparing to the EM algorithm. Since this algorithm takes a heuristic approach, it may require a lot of tests to decide what kind of

setting is better on a regular base rather than choosing the one works better only in some special cases.

Appendix 1. Example 2 input data

Line 1		Line 2		Line 3		Noise	
10	1	1	1	3.0	1	16	3
10	2	2	2	3.1	2	17	18
10	3	3	3	3.2	3	23	9
10	4	4	4	3.3	4	21	14
10	5	5	5	3.4	5	1	20
10	6	6	6	3.5	6	6	18
10	7	7	7	3.6	7		
10	8	8	8	3.7	8		
10	9	9	9	3.8	9		
10	10	10	10	3.9	10		
10	11	11	11	4.0	11		
10	12	12	12	4.1	12		
10	13	13	13	4.2	13		
10	14	14	14	4.3	14		
10	15	15	15	4.4	15		
10	16	16	16	4.5	16		
10	17	17	17	4.6	17		
10	18	18	18	4.7	18		
10	19	19	19	4.8	19		
10	20	20	20	4.9	20		
10	21	21	21	5.0	21		
10	22	22	22	5.1	22		
10	23	23	23	5.2	23		
10	24	24	24	5.3	24		
10	25	25	25	5.4	25		

Appendix 2 The residuals

Regular	Student	Jackknife	Leverage
$e_i = Y_i - \hat{Y}_i$			

	$r_i = \frac{e_i}{\sqrt{(MS_{residual})(1-h_{ii})}}$	$r_{(-i)} = \frac{e_i}{\sqrt{(S^2_{(-i)}(i-h_{ii})}}$	$h_{ii} = \frac{1}{n} + \frac{(X_i - \overline{X})^2}{SS_x}$

Appendix 3 K-Mean Line Fitting algorithm

Choose k lines (perhaps uniformly at random)
Or choose $\overline{L}$
Until convergence
E-step:
Recompute $\overline{L}$, from perpendicular distances
M-step:
Refit lines using weights in $\overline{L}$
End