Least-Squares Fitting
Ethan Chen

I have fitted multiple lines to my data using three different methods related to least squares. The concept of least-squares fitting is fundamental to statistics and data analysis and is seen from simple linear regression to predictive modeling. First, we apply the traditional method of minimizing vertical distances; then, we minimize the horizontal distances; and lastly, we minimize the absolute Euclidean distances from the points to the line. These three methods provide different assumptions based on x and y and have unique applications and limitations.

Method 1: (Minimizing Vertical Distances)

The first method is the "usual" least squares method, which fits a line to the data by minimizing the sums of the squares of the vertical distances between the points and the line. The line is formed by the formula $y = mx + b$, where m is the slope and b is the y-intercept. We minimize the sum of squares by using the mean of all the x and y points, respectively, and using this formula:

$$\frac{\sum((x-\bar{x})(y-\bar{y}))}{\sum(x-\bar{x})^2}$$

Below is the data and packages used. Two common Maple packages were utilized: Statistics and Optimization. The Statistics package includes functions and procedures commonly used for linear regression. For example, we used the LinearFit function which minimizes the vertical distances and is further explained below. The Optimization package was heavily used for the second and third methods, where we minimized different types of error measurements. This allowed us to solve nonlinear optimization problems and find best-fit lines.

```
> data := [[8, 2.894746170], [-1.556499717, 1.613907321],
  [-5.708967368, 0.7593346346], [-7.980407785, 1.181413808],
  [5.724224072, -1.549787358], [8.309756060, -1.509077657],
  [-3.653465654, 0.6749916993], [-5.362317535, 0.8348877924],
  [0.4817122490, 1.074991931], [-7.284849105, 0.8455075900],
  [-2.595655524, 0.0986332061], [9.584509036, -1.275190377],
  [-4.131729772, -0.0434578545], [6.146808246, -1.330197144],
  [-9.176855367, 2.684848513], [-9.134106432, 2.798912260],
  [-9.149810024, 2.795265592], [-2.902296960, 1.821119870],
  [-6.353773368, 1.020665625], [-6.184460387, 1.359524694],
  [-7.148214392, 1.791988079], [-5.553685070, 1.770822006],
  [8.374775978, -1.056271267], [7.119268716, -0.2723424370],
  [5.361547724, 0.1457619670], [-1.191146386, 1.238537489]];
  with(Statistics):
  with(Optimization):
```

(1)

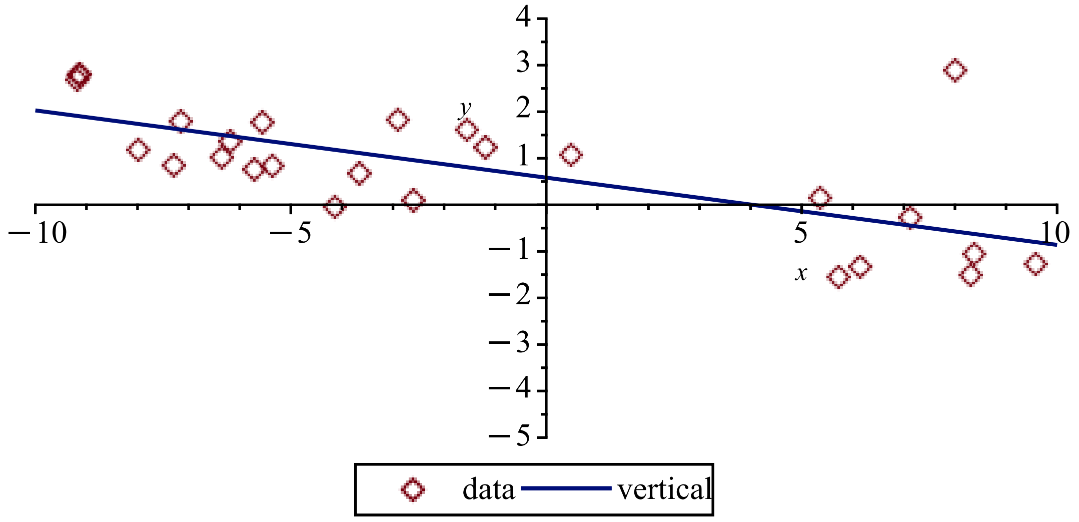
$$data := [[8, 2.894746170], [-1.556499717, 1.613907321], [-5.708967368, 0.7593346346], [-7.980407785, 1.181413808], [5.724224072, -1.549787358], [8.309756060, -1.509077657], [-3.653465654, 0.6749916993], [-5.362317535, 0.8348877924], [0.4817122490, 1.074991931], [-7.284849105, 0.8455075900], [-2.595655524, 0.0986332061], [9.584509036, -1.275190377], [-4.131729772, -0.0434578545], [6.146808246, -1.330197144], [-9.176855367, 2.684848513], [-9.134106432, 2.798912260], [-9.149810024, 2.795265592], [-2.902296960, 1.821119870], [-6.353773368, 1.020665625], [-6.184460387, 1.359524694], [-7.148214392, 1.791988079], [-5.553685070, 1.770822006], [8.374775978, -1.056271267], [7.119268716, -0.2723424370], [5.361547724, 0.1457619670], [-1.191146386, 1.238537489]] \quad (1)$$

The LinearFit function from the Statistics package is used to fit a linear model function to data. In this case, it performs simple linear regression by minimizing the least-squares error. This relationship is then represented through a linear model of $y = mx + b$, and provides us with an equation of the line that best fits our data in terms of vertical distances. Our call includes "[1, x]" being the model used, "data" represents a list of points, and x denotes the variable name used in the function.

```
> fit1 := LinearFit([1, x], data, x)
```

$$fit1 := 0.583636986475381 - 0.144442714850253\,x \quad (2)$$

```
> plot([data, fit1], x = -10 .. 10, y = -5..4, style = [point,
  line], symbolsize = 20, size = [0.8, 0.5], legend = ["data",
  "vertical"]);
```



As shown above, minimizing the sums of the squares of the vertical distances gives us a fitted line of roughly$y = 4.7784\,x - .3009$.

Method 2: (Minimizing Horizontal Distances)

Instead of minimizing vertical distances, we will now use horizontal distances to minimize sum of the squares. It also follows the same formula but assumes x-values may contain errors while y-values are exact.

First, we define the sum of squares function. S(m, b) represents the sum of squares of errors between actual y-values and predicted y-values. In this function, $y - b - mx$ is the difference between the actual y-value of a data point and the predicted y-value for the corresponding x-value. Dividing by m adjusts the distance for the horizontal component, and we square it to ensure distances are positive and emphasize larger errors. Lastly, we take the sum to calculate the total error between data points and the fitted line.

```
> S := (m2, b2) -> local i; add((data[i][2] - b2 - m2*data[i][1])
  ^2/m2^2, i = 1 .. nops(data))
```

$$S := (m2, b2) \mapsto add\left(\frac{\left(\left(data_i\right)_2 - b2 - m2\cdot\left(data_i\right)_1\right)^2}{m2^2}, i = 1..nops(data)\right) \tag{3}$$

We need to find the values of the slope (m) and y-intercept (b) that minimize the total error between predicted and actual values. The total error is quantified by the sum of squares function S(m, b). To find the values of m and b that minimize the total error, we set these partial derivatives equal to zero (the function does not increase or decrease) and solve the resulting system of equations.

```
> dm := diff(S(m2, b2), m2):
  db := diff(S(m2, b2), b2):
```

Then we solve for b and m and plot the function.

```
>  solutions := [solve({db = 0, dm = 0}, {b2, m2})]
```

$$solutions := \left[\{b2 = 0.3553678703, m2 = -0.3094612499\}, \{b2 = -5.066370326 \times 10^9, m2 = -3.662541054 \times 10^9\}\right] \tag{4}$$

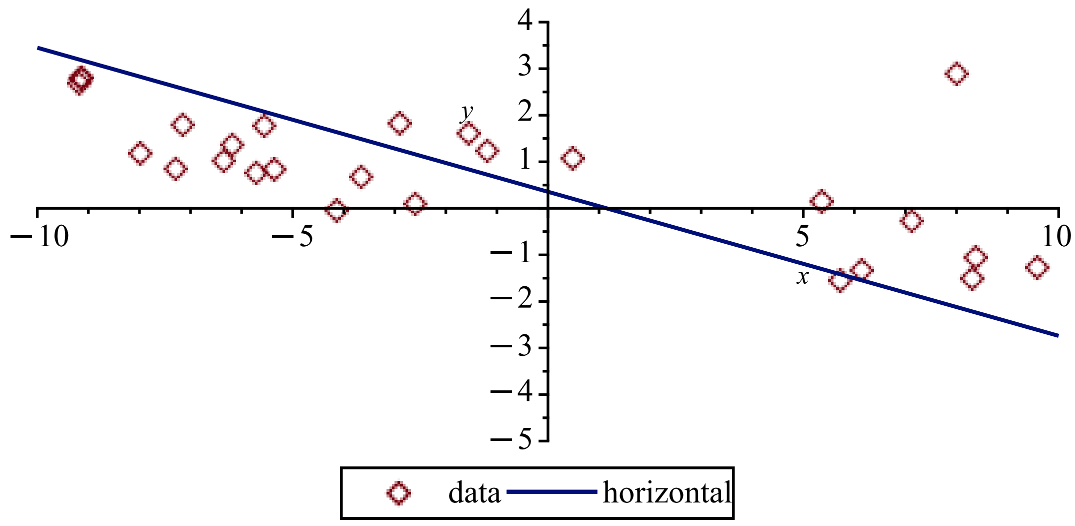
```
> slope2 := eval(m2, solutions[1]);
  intercept2 := eval(b2, solutions[1]);
```

$$slope2 := -0.3094612499$$
$$intercept2 := 0.3553678703 \tag{5}$$

```
> fit2 := intercept2 + slope2*x;
```

$$fit2 := -0.3094612499\,x + 0.3553678703 \tag{6}$$

```
> plot([data, fit2], x = -10 .. 10, y = -5..4, style = [point,
  line], symbolsize = 20, size = [0.8, 0.5], legend = ["data",
  "horizontal"])
```



Minimizing the horizontal distances gives us a fitted line of roughly $y = -.3095\,x + .3554$ •

Method 3: (Minimizing Absolute Euclidean Distance)

For this method, we will assume x and y values are approximate and find the line that minimizes the sum of the squares of the absolute distances from the points to the line.

To find a line that best represents the relationship between the variables under these new assumptions, we first define a function called "f" that calculates the sum of squared perpendicular distances. This uses the formula for distances from a point to a line. The formula $\left(\frac{(y - mx - b)}{\sqrt{1 + m^2}}\right)$ comes from the a line being expressed as $Ax + By + C = 0$. If we rewrite $y = mx + b$, we get $mx - y + b = 0$, which gives us $A = m$, $B = -1$, **and** $C = b$. Once we substitute those back in, we get $d = \left(\frac{(-mx + y - b)}{\sqrt{m^2 + 1}}\right)$, which gets rewritten as $\left(\frac{(y - mx - b)}{\sqrt{1 + m^2}}\right)$.

```
>
>
> f := (m3, b3) -> sum((data[i][2] - m3*data[i][1] - b3)^2/(1 +
  m3^2), i = 1 .. nops(data));
```

$$f := (m3, b3) \mapsto \sum_{i=1}^{nops(data)} \frac{\left(\left(data_i\right)_2 - m3\cdot\left(data_i\right)_1 - b3\right)^2}{1 + m3^2} \tag{7}$$

We then calculate the partial derivatives with respect to the slope and intercept to represent the total squared distance changes.

```
> dfdm:=diff(f(m3,b3),m3):
  dfdb:=diff(f(m3,b3),b3):
```

Then solve for the system of equations after setting partial derivatives equal to 0. This will give us the critical points.

```
> solve({dfdb = 0, dfdm = 0}, {b3, m3});
```

$$\{b3 = 10.13671296, m3 = 6.761592708\}, \{b3 = 0.5788626393, m3 = -0.1478941488\}, \{b3 = -5.066370336 \times 10^{9}, m3 = -3.662541061 \times 10^{9}\} \quad (8)$$

The first two critical points seem reasonable, while the third one is significantly greater, which could be due to rounding errors and is most likely not meaningful.

To figure out which critical point is the minimum, we can plug it into f and compare to find the smallest value. We can also confirm it by using the minimize function to consider the error of all dimensions and ensure the line is as close to all points simultaneously.

```
> dist1 := f(3.183109885, 0.5753026080, data);
  dist2 := f(-0.3141581774, 4.794406801, data);
  dist3 := f(-6.731714688*10^10, 8.121140603*10^10, data);
  result := Minimize(f(m3, b3, data), m3 = -10 .. 10, b3 = -10 ..
  10);
```

$$dist1 := 1081.301620$$

$$dist2 := 517.2607223$$

$$dist3 := 1209.330213$$

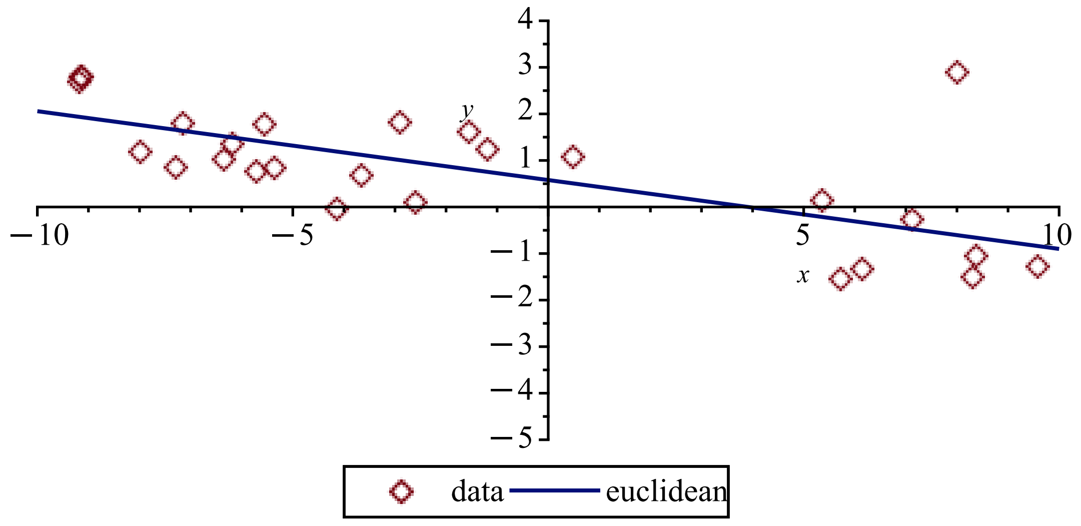
$$result := [24.1530787312199031, [b3 = 0.578862636907827, m3 = -0.147894148479204]] \quad (9)$$

> As seen, dist2 or the second critical point is the minimum, and it's confirmed by the minimize function.

```
> fit3 := -0.147894148479204*x + 0.578862636907828;
```

$$fit3 := -0.147894148479204\,x + 0.578862636907828 \quad (10)$$

```
> plot([data, fit3], x = -10 .. 10, y = -5..4, style = [point,
  line], symbolsize = 20, size = [0.8, 0.5], legend = ["data",
  "euclidean"])
```



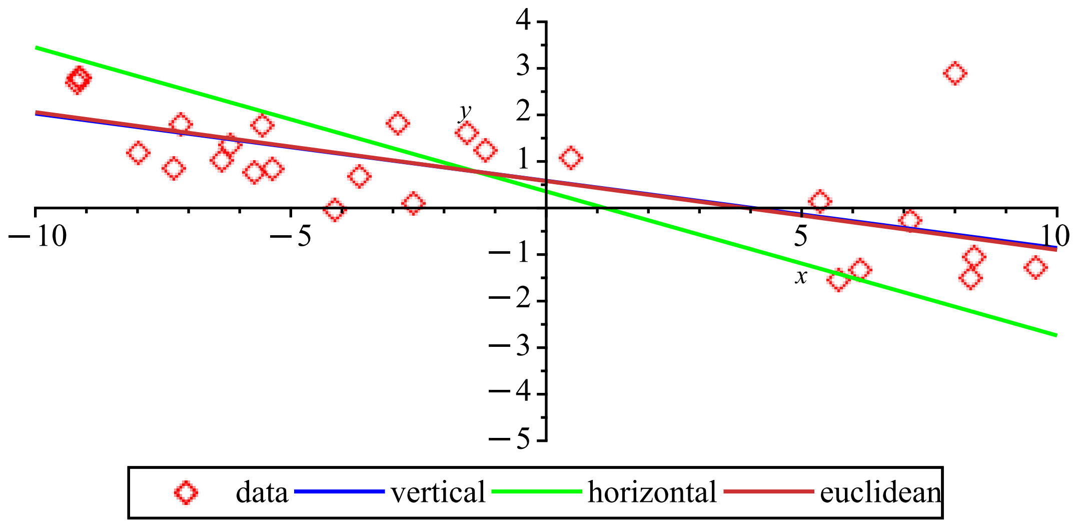
We now have our three fitted functions.

```
> vertical:=fit1;
  horizontal:=fit2;
  euclidean:=fit3;
  plot([data, fit1, fit2, fit3], x = -10 .. 10, y = -5 .. 4, style
  = [point, line, line, line], symbolsize = 20, size = [0.8, 0.5],
  legend = ["data", "vertical", "horizontal", "euclidean"],
  thickness = [0, 1, 1, 1], color = [red, blue, green, orange])
```

$$vertical := 0.583636986475381 - 0.144442714850253\,x$$

$$horizontal := -0.3094612499\,x + 0.3553678703$$

$$euclidean := -0.147894148479204\,x + 0.578862636907828$$



This highlights the application and impact of three different least-squares fitting methods: "usual" vertical, horizontal, and absolute Euclidean distances. The Euclidean method accounts for both x and y values, which presents a more comprehensive analysis due to its balanced approach to error distribution. This contrasts with minimizing horizontal distances, which is more useful when dealing with independent variables (x) subject to significant measurement errors. The traditional approach of minimizing vertical distances yields nearly identical results to the Euclidean method, which could explain why it's the "usual" method. It offers good approximations for most data where errors are presumed to be in the dependent variable. The outcomes provide clear functionalities for the three different methods, and signify the importance of selecting the proper least-squares fitting method based on the characteristics and error distributions of the dataset.

```
>
```