Various Least Squares Fitting

Patryk Kwoczak

Introduction

Least Squares fitting has been applied for more than 200 years; since Gauss (or Legendre, I do not wish to invite controversy to the space of mathematics), it has proven useful for creating a model to describe the relationship between data sets. The coefficients of the Regression Model depend on minimizing the square power of the distances between the points of the data and the line itself by various metrics: Ordinary Least Squares (Vertical Least Squares) measures the vertical difference, Horizontal Least Squares measures horizontal, and Absolute Total Least Squares measures the Euclidean distance. The data used can be found here[1]. All graphs and code have been done using Maple.

Ordinary Least Squares

The Ordinary Least Squares, or the one which minimizes the sum of squares of the vertical distance, has its use as the conventional regression model due to the simplicity of the computations and the assumptions required: the independent variable is assumed to be exact in measurement, with normally distributed noise centered around the value of the line that reflects the relationship of the independent and the dependent variable.

The Equation

Let $y = m * x + b$ be the linear regression, we must find values for m, b which will minimize the square error, ϵ, produced by the vertical deficiency. Therefore, for every dependent data entry we have the equation $y_i = m * x_i + b + \epsilon_i$, or solving for the error, $\epsilon_i = y_i - (m * x_i + b)$. Squaring it,

$$\epsilon_i^2 = \big(y_i - (m * x_i + b)\big)^2$$

Factoring in all data entries to gain the most accurate line, the sum will come out as

$$\sum_{i=1}^{n} \epsilon_i^2 = \sum_{i=1}^{n} \big(y_i - (m * x_i + b)\big)^2 = SSE(m, b)$$

[1] http://www.math.stonybrook.edu/~scott/mat331.fall19/problems/proj1data/pkwoczak.txt

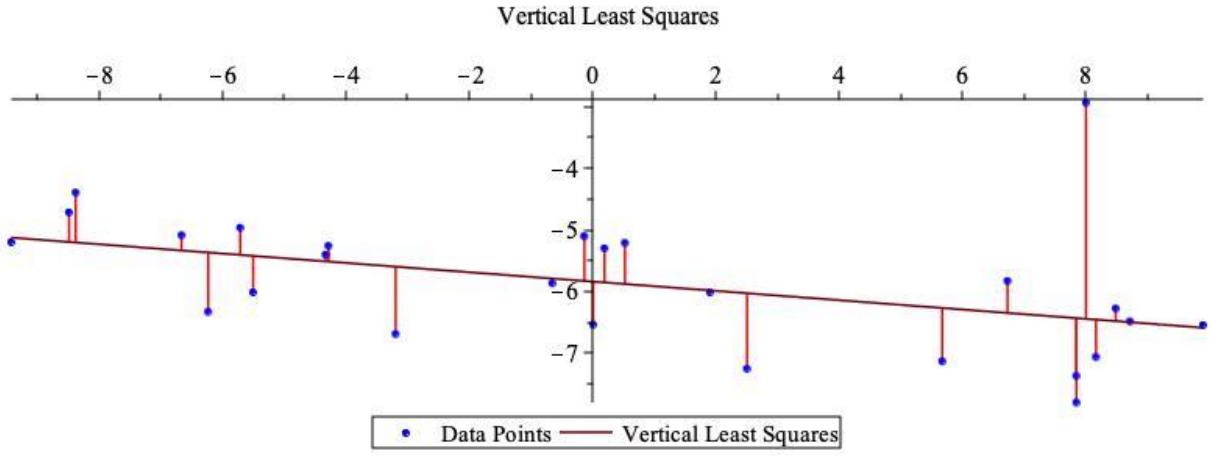
This is our function of the sum of squared errors, a function with two unknowns. Because we must minimize this value, we take the partial derivatives, set them equal to zero, and solve for the optimal values for m and b.

The Code

```
xLSQ := LeastSquares(data, x);
      xLSQ := -5.83810815099945 - 0.0757321291526669 x
```

Maple will do this for you, so finding the coefficients of the Vertical Least Square Regression is trivial with the help of a computer friend. To plot the data, one must connect the given points to the corresponding point on the line which serves as the closest point given the vertical metric; in other words, finding the projection of the entry onto the line by the standards assigned to the Vertical Least Square model. It happens to be very simple to attain the coordinates of both points in this case: taking the x-value of the data point and plugging it into the function of the line, we get the y coordinate; using the line() command, these are joined with a line segment.

```
plotDiffX := proc(trueVals, curve)
local arrayPositions, arrayDPlots, arrayDiff, arrayCVals, i, n;
n := nops(trueVals);
arrayPositions := [seq(trueVals[i][1], i = 1 .. n)];
arrayCVals := [seq(subs(x = arrayPositions[i], curve), i = 1 .. n)];
arrayDiff := [seq(trueVals[i][2], i = 1 .. n)];
arrayDPlots := [seq(line([arrayPositions[i], arrayCVals[i]], [arrayPositions[i], arrayDiff[i]], col
or = red), i = 1 .. n)];
return arrayDPlots;
end proc
```



Horizontal Least Squares

There may also be data that is inaccurate in the independent variable, yet accurately measured in the dependent. In this case, the assumptions of the Ordinary Least Squares Regression are violated and so the measurement of the error is compromised. To tackle this, we must consider an error term δ appearing in the horizontal difference between the value of the independent variable in the data, and the corresponding value of the independent variable in the graph.

The Equation

Assuming the noise is horizontal, meaning it is spread across the x-axis, the error can be defined as

$\delta_i = x_i - \frac{(y_i - b)}{m}$. Carrying out the following steps in consequence,

$$\delta_i^2 = \left(x_i - \frac{(y_i - b)}{m}\right)^2$$
$$\sum_{i=1}^{n} \delta_i^2 = \sum_{i=1}^{n} \left(x_i - \frac{(y_i - b)}{m}\right)^2 = SSE_h(m, b).$$

Where we can see $SSE_h(m, b)$ is the function of the sum of square errors in the horizontal metric. One carries out optimization of the function to find the minimum error and takes the coinciding values of m and b as coefficients for the linear regression.

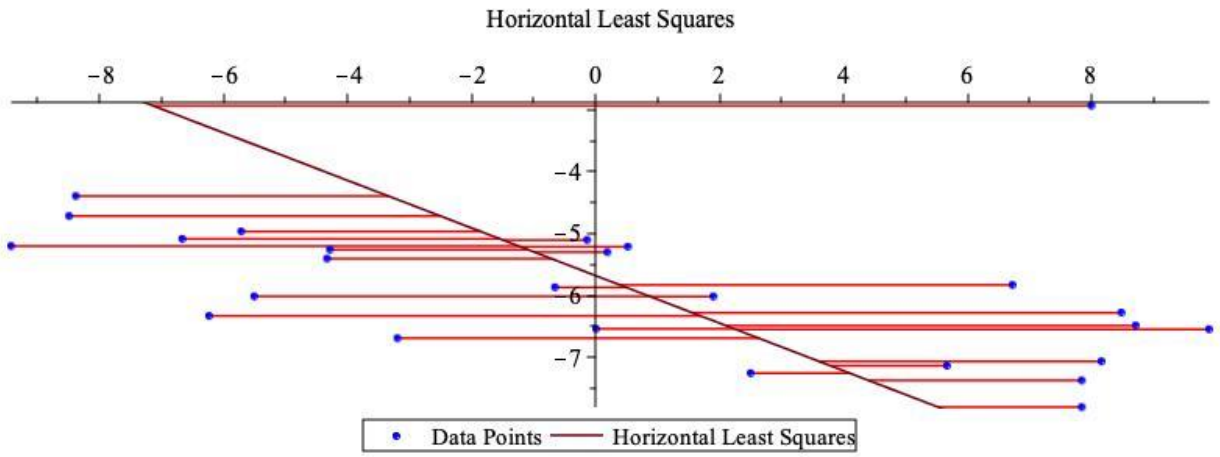
This can be done another way without violating the assumptions. First, note that the linear regression model is a function of one-to-one correspondence: it is a straight line with a constant derivative. Therefore, an injective function will exist. By transforming the coordinates to shift the y-values as the independent variable and the x-values as the new dependent, one can perform an Ordinary Least Regression in the Vertical metric, then transform the function to its injective and reinstate the original independent and dependent points back to their former selves. Essentially the computations are carried out by first reflecting the data off of the line $y = x$, carrying out Ordinary Least Squares, and reflecting everything again by the same line, including the regression model. The reason this is the same is because the δ_i error is equal to the ϵ_i error of the "flipped" points, as the new vertical error is equivalent to the original horizontal error.

The Code

```
oppsData := [seq([data[i, 2], data[i, 1]], i = 1 .. nops(data))];
yLSQInv := x -> LeastSquares(oppsData, x);
yLSQ := solve(x = yLSQInv(y), y);
        yLSQ := -5.677628245 - 0.3844819600 x
```

The code above illustrates this process. Then, plotting the original data alongside the new Horizontal Linear Regression, the points are connected by taking every dependent value of the data set, y_i, and solving for where the independent variable of the linear model will map to that y-value.

```
plotDiffY := proc(trueVals, curve)
local arrayPositions, arrayDPlots, arrayDiff, arrayCVals, i, n;
n := nops(trueVals);
arrayPositions := [seq(trueVals[i][2], i = 1 .. n)];
arrayCVals := [seq(solve(curve = arrayPositions[i]), i = 1 .. n)];
arrayDiff := [seq(trueVals[i][1], i = 1 .. n)]; arrayDPlots := [seq(line([arrayCVals[i], arrayPositi
ons[i]], [arrayDiff[i], arrayPositions[i]], color = red), i = 1 .. n)];
return arrayDPlots;
end proc:

allHorLines := plotDiffY(data, yLSQ)
display(allHorLines, plotPoints, plotyLSQ, size = [800, 300], title = "Horizontal Least Squar
es")
```



An observation worth noting is the anomaly of the point at approximately $(8, -3)$, which skews the resulting curve via the assumption of the accuracy of the dependent values recorded: the square difference it produces is about $\left(8 - (-7.137)\right)^2 = 15.13^2 = 229.122$, which relative to the rest of the points adds much more weight.

Absolute Total Least Squares

In certain circumstances with data, especially regarding the information collected by fallible humans, there will be noise in both the dependent and the independent variable. Then, it would become absurd to assume that either variable is perfectly measured; a way to combat this is to add contribution from

both horizontal error, δ_i, and the vertical, ϵ_i, between each point and the regression model by minimizing the distance between every point and the line.

The Equation

The equation for calculating this distance utilizes the fact that the point which minimizes the Euclidean distance will be normal to the regression model; distance will therefore be found between each point (x_i, y_i) and the closest point to the line $y = m * x + b$ the equation $d = \frac{|y - m*x - b|}{\sqrt{(1+m^2)}}$.

Proof

Let (x_l, y_l) be the point on the line which is closest to (x_i, y_i). Then the line segment connecting both points is perpendicular to the regression, and so the slope of the line segment is $-\frac{1}{m}$. So,

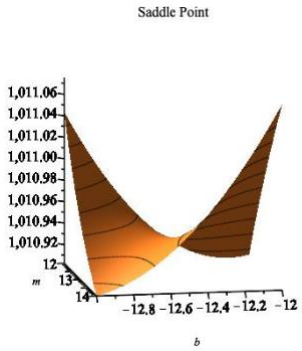
$$\frac{y_i - y_l}{x_i - x_l} = -\frac{1}{m}$$
$$-m(y_i - y_l) - (x_i - x_l) = 0$$
$$\left(-m(y_i - y_l) - (x_i - x_l)\right)^2 = 0 \Leftrightarrow$$
$$m^2(y_i - y_l) + (x_i - x_l)^2 = 2m(y_i - y_l)(x_i - x_l)$$
$$\text{Now, } \left(-m(x_i - x_l) + (y_i - y_l)\right)^2 =$$
$$m^2(x_i - x_l)^2 - 2m(y_i - y_l)(x_i - x_l) + (y_i - y_l)^2 = (m^2 + 1)\left((x_i - x_l)^2 + (y_i - y_l)\right)^2$$
$$\text{But, } \left(-m(x_i - x_l) + (y_i - y_l)\right)^2 = (-mx_i + y_i + m * x_l - y_l)^2 = (-mx_i + y_i - b)^2$$
$$\text{Due to } (x_l, y_l) \text{ being on the line } y = m * x + b, \text{and hence}$$
$$(m^2 + 1)((x_i - x_l)^2 + (y_i - y_l)^2)^2 = (-mx_i + y_i - b)^2,$$

$$\text{thus } d_i^2 = (x_i - x_l)^2 + (y_i - y_l)^2, \text{and } (m^2 + 1)((x_i - x_l)^2 + (y_i - y_l)^2) = (y_i - mx_i - b)^2,$$
$$\text{so } \mathrm{d}_i = \frac{|y_i - mx_i - b|}{\sqrt{(m^2 + 1)}}.$$

Finding the sum of square errors will be completed the ordinary way:

$$\sum_{i=1}^{n} TE_i^2 = \sum_{i=1}^{n} \frac{(y - m * x - b)^2}{1 + m^2}$$

Taking the previous Least Squares values for m, b and finding their average, it is simple to compute the minimum value and solve for the coefficients numerically. One must be careful when solving carelessly the optimization problem; a saddle point can surely also exist. In fact, one appears with this data. Obviously, this should be avoided; the graph for the saddle point will be presented below and must exist, due to the nature of the two parameters: the horizontal error can be minimized at the cost of the vertical and vice versa. The final answer should not represent that balance.



The figure's vertical axis represents the sum of square total errors as the values of m and b vary; the saddle point is at $(m, b) = (12.897, -12.581)$.

The Code

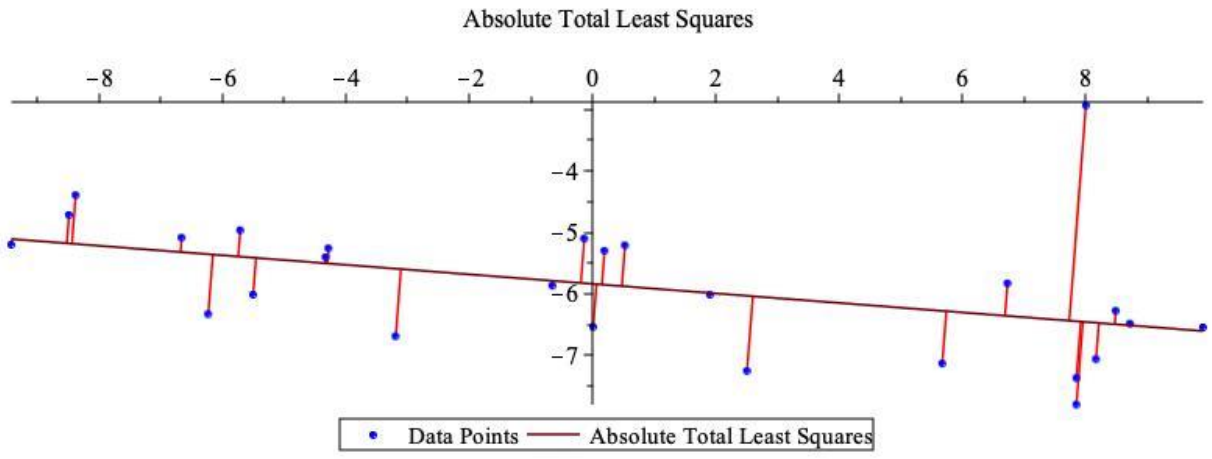
```
getEuclidianLS := proc(points, mGuess, bGuess) local i, xvals, yvals, n, myline, m, b, err, d
ist, distance, p, distanceSq, E, mb, obj, myPlot, mb2;
n := nops(points);
xvals := [seq(points[i, 1], i = 1 .. n)];
yvals := [seq(points[i, 2], i = 1 .. n)];
distanceSq := (p, m, b) -> (p[2] - m*p[1] - b)^2/(1 + m^2);
E := (m, b) -> add(distanceSq(data[i], m, b), i = 1 .. n);
myPlot := plot3d(E(m, b), m = 12 .. 14, b = -13 .. -12, style = patchcontour, color = "Chocol
ate");
mb := fsolve({diff(E(m, b), m) = 0, diff(E(m, b), b) = 0}, {m = mGuess - 1 .. mGuess + 1, b = b
Guess - 1 .. bGuess + 1});
mb2 := solve({diff(E(m, b), m) = 0, diff(E(m, b), b) = 0});
myline := x -> subs(mb, m*x + b);
return [myline(x), myPlot, mb2];
end proc
```

A note for readers interested in MAPLE, having 'x' assigned to be a local variable is a grave mistake: the returned x value is not the same as the rest in the program outside the procedure.

Then there must be an attempt made to plot the line segment representing the Euclidean distance which was calculated for. For each individual point of data I found the point which minimized difference, and connected the two through the use of the line() procedure available.

```
plotDiffE:=proc(trueVals, curve) local arrayPositions, arrayDPlots, arrayDiff,
  arrayCVals, i ,n,distanceFunc, optiDis, getClosestVal, curveX, curveVals;
  n:=nops(trueVals);
  distanceFunc := (t, p) -> sqrt((p[1] - t)^2 + (p[2] - curve(t))^2);
  optiDis := (t, p) -> diff(distanceFunc(t, p), t);
  getClosestVal := p -> solve(optiDis(t, p) = 0);
  curveX := [seq(getClosestVal(trueVals[i]), i = 1 .. n)];
```

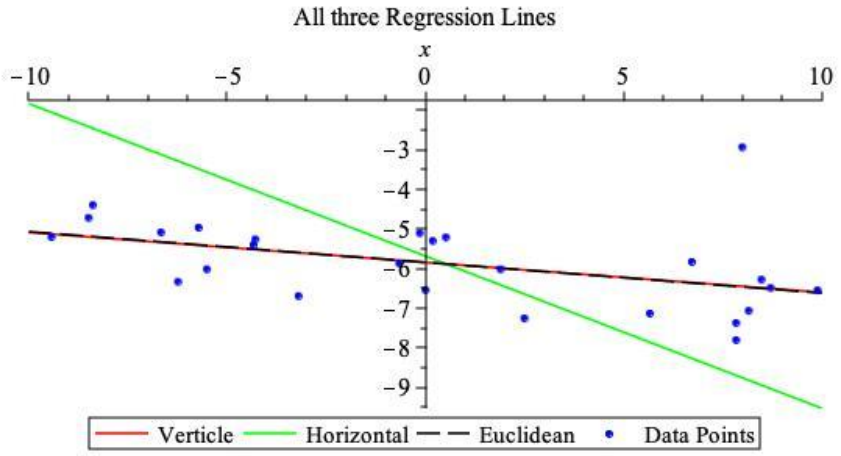
```
  curveVals := [seq([curveX[i], curve(curveX[i])], i = 1 .. n)];
  arrayDPlots := [seq(line([trueVals[i][1], trueVals[i][2]], [curveVals[i][1],curveVals[i,
2]],color=red),i = 1 .. n)];
return arrayDPlots;
end:

mGuess := (diff(xLSQ, x) + diff(yLSQ, x))/2;
                mGuess := -0.230107044576333
bGuess := (eval(xLSQ, x = 0) + eval(yLSQ, x = 0))/2;
                bGuess := -5.75786819799973

WrongData := getEuclidianLS(data, mGuess, bGuess)[3]; #Getting Sadle point info
        WrongData := {b = -12.58124095, m = 12.89748779}
PlotOfWrongData := display(getEuclidianLS(data, mGuess, bGuess)[2], title = "Sad
dle Point", orientation = [-10, 80, 0]);

EuclideanLine := getEuclidianLS(data, mGuess, bGuess)[1]; #Proper Euclid
eLine := unapply(EuclideanLine, x);
              -0.7753447908e-1*x-5.837171338
ElinePlot := plot(eLine(x), x = xRange, y = yRange, scaling = constrained, legend = "
Absolute Total Least Squares");
EuclidPlots := plotDiffE(data, eLine);
display(EuclidPlots, plotPoints, ElinePlot, title = "Absolute Total Least Squares", si
ze = [800, 300]);
```



The Algorithm for calculating the points on the regression model does not need to be so labour-intensive. All the line segments follow the same slope: it is always orthogonal to the linear regression. So the only necessary steps are to generate the perpendicular line from (x_i, y_i), which would become

$(y - y_i) = m(x - x_i)$, and solve the set of equations to arrive at (x_l, y_l). But alas, the computer is for cool derivative stuff.

Recap



It is clear that the Vertical and Euclidean are similar and follow the trend more accurately than the Horizontal. The assumptions just don't fit an accurate recording of the dependent variable. Nevertheless, it is very useful to consider the alternative options and observe the plots.