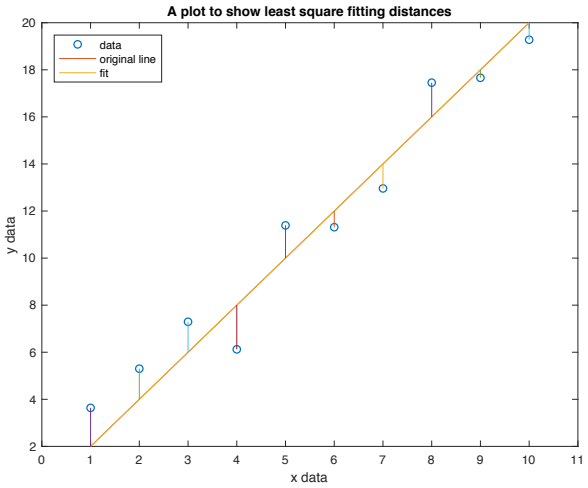
10/18 in Class – Least Square Fitting of Linear Data and creating a known test data set to test our code

Edited after class 10/18 to correct numerical error in 2 g)

In class, I showed that

$$\chi^2 = \sum_{k=1}^{N} (mx_k + b - y_k)^2$$

is the sum of the squares of the distances (in the y direction) from the fit to the data. Also sometimes called the sum of the squares of the errors. The x_k and y_k indicate the k^{th} data point, and the m and b are the slope and intercept of the line you are comparing to.



1. Step 1: Generate a set of data with a known offset. This will be used to test a chunk of code you will write to find the sum of the squares – the χ^2 in the equation above. Steps for this are further broken down as:

   (a) Start by making a vector for x. I used the integers from 1:10.

   (b) Let's all start with the same line: $y = 2x + 0$.
   Make a vector for y that has that equation.

   (c) Plot the x and y you just created as a line (no specific data points). (This should correspond to what I called "original line" in the figure above.)

   (d) Now write code to offset the y-values of the data set you just created. You want to offset them by some known amount so that you can sum the squares yourself to check your code. For the set I showed in class, I had offset every other data point by ± 0.5 units in y. (You don't want to shift them all the same direction–that does not make a great test of code. But you could start there, test if it works, then make it more sophisticated.) There are several ways to do this. You could write a for loop, or you could do it with vector commands.

   (e) Plot the newly created offset data as data points (circles, or some symbol of your choice) on top of the line. Do the data points appear offset as you planned? Show me the plot.

(f) Write out the offset data to a file called `knownOffset.txt`. So that you can use the same code in the next section, please output the data as two columns, x first, then y.

(g) Calculate by hand what your sum of the squares (χ^2) will be. (Comparing your data of known offset to the line $y = 2x$.)

2. Step 2: Write code that can find the χ^2 for any set of data for x and y, and any value of m and b. It should be able to read in your data file, `knownOffset.txt` and one I will give you. The steps to do this are broken down in more detail below:

(a) Figure out how you will do this sum–a for loop? Can you vectorize it?

(b) Write code to read in a .txt file that contains two columns of data, x and y and then calculate the sum of the squares of the distances between the data and a given line. Call your script `sumSquares.m`.

(c) Check your code by using it to read in your data file, `knownOffset.txt`, and the given line, $y = 2x$.

(d) Does it work? Do you get the χ^2 you calculated by hand? If so, then go on. If not, stop there and debug your code. Or your data. Do not go on till this part works.

(e) Record this value of χ^2, you'll want to compare to it later.

(f) Change the values of m and b in your code. Change them several times. Does the χ^2 go up each time? Should it?

(g) Test your code on a file you download from the class website. It's in today's calendar topic, and it's called, `randomY12x-4.txt`
As you can probably guess, to generate the data, I used a line with a slope 12 and y-intercept of -4. You should get 25.3045 for the χ^2. Does it work? (Don't go on until it does.)

Please get to here for class 10/21

3. Step 3: Now let's find the line of best fit to some data. One way is the method of least squares: that is, finding the line that minimizes of the sum of the squares you just calculated. I showed that finding the minimum of χ^2 (by taking the first derivative with respect to m and b and setting them each equal to zero, then solving two equations for two unknowns) yields:

$$m = \frac{c_{xy} - c_x c_y}{c_{xx} - c_x c_x} \qquad b = \frac{c_{xx} c_y - c_x c_{xy}}{c_{xx} - c_x c_x}$$

where

$$c_x = \frac{1}{N}\sum_{k=1}^{N} x_k \qquad c_{xx} = \frac{1}{N}\sum_{k=1}^{N} x_k^2 \qquad c_y = \frac{1}{N}\sum_{k=1}^{N} y_k \qquad c_{xy} = \frac{1}{N}\sum_{k=1}^{N} x_k y_k$$

(a) Go to the white board again with your group and write a structure plan for how to implement this method in code. Go into enough detail: will you use for loops? Can you vectorize it?

(b) Then write code to do this. Call it `leastSquaresLine.m`

4. Step 4: Test your scripts and play with them.

(a) First check to see if your code recovers the original, given line:

One way to do this is: run your code from Problem 3 (`leastSquaresLine.m`) on the data you generated in Problem 1. (That data should be in `knownOffset.txt`.)

(b) Does your least fit algorithm get you back your original line? ($y = 2x$, so, $m = 2$ and $b = 0$) So this is asking: does your code give least squares fit values of $m = 2$ and $b = 0$? It should be close, but not exact. Why?

(c) Now see if these new values of m and b are in fact closer to the data than our original line. A way to do this is: Take the values you found for the minimum of m and b and plug them into your script from Problem 2 (`sumSquares.m`) and see if the new fit values make the χ^2 smaller. Does it? It should–why?

(d) Show your two values of χ^2 to me.

Congratulations, you have successfully written your first data fitting algorithm!

Two other ways: 1) MATLAB has a built in function, `polyfit`. Our textbook has a nice example in section 14.9. I will adapt that code and put it on our course web page.

2) If you buy the Optimization Toolbox from MATLAB, there's a function called `lsqcurvefit` that does this. I did not require that for this class (and in fact neither I nor the lab computers have it), so let's do it another way, a way that will also be good practice writing functions.

3) The command `fminsearch` finds the root(s) that minimize the value of any function. In class Tuesday (3/20), I showed two or three examples of how to use `fminsearch`, and Chris Ray's text, section 5.4 (linked on the class web page) shows an example of one way to call it from a function.

So all we have to do is write our own function that calculates χ^2 so that we can call `fminsearch` on the function. The good thing about this method is, we can use it for fitting any function, not just a line. We still have to write the function, but we do not have to calculate the c_x things.

We will pick up here next time!