Military Institute of Science and Technology

Department of Computer Science and Engineering (CSE)

Subject: CSE 214 - Numerical Analysis Sessional

Name of the Exp.: Fitting Linear and Nonlinear curves from experimental data by Least Square Method

Let's do a quick recall of what we learnt in last class. Unlike previous numerical experiments we did in class, where we had many equations and had to find the roots (or data, per say) that would best solve or "fit" all the equations, we had a completely reverse scenario! We had an abundance of data (very uncommon in traditional numerical methods so far, I know!) and we had to find an equation that would best describe the data.

Now, to do that, we could draw a **line or a curve** in such a way that, the line or curve would be the **closest** to all the data points. So, notice that, we already have two ways of doing it-

a) To use a **line** to fit the data – line fitting. Here of course, we'll use a linear equation in the form of y = mx +c.
b) Or to use a **curve** to fit the data – curve fitting. Here instead, we'll have to use polynomial eqn in like

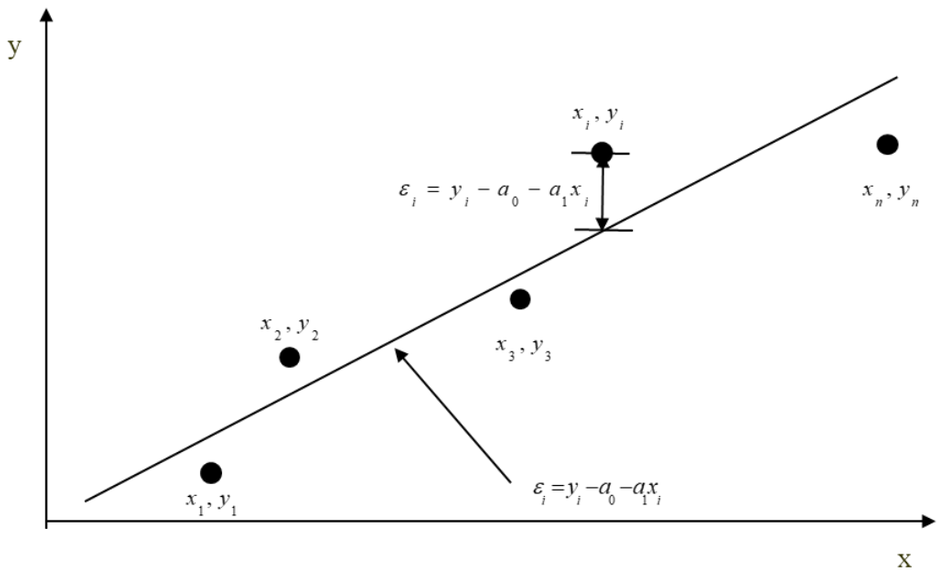
$$Y = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n$$

So, naturally the **first question** that arises is, when to use which one? a or b? I'm afraid the answer in not so simple. It would depend on the data and your judgement. If the data that we have (observed, collected, experimental, statistical, historical – data sources can be so many in real life) is very precise, that is, we are confident that collected data is as less erroneous (or "pure") as possible, we should go for curve fitting. Understand this, if I'm confident about my data, it would make sense to bend my line like a curve to make my curve lie as closely to my data points as possible, right?

On the other hand, if I'm not so sure about my data, it would make sense to cover a wide region near the data by drawing a straight line instead.

So, to summarize, the more precise my data is, the more I should lean towards curving my line toward my data points. How to do that? Well, simply, I can just increase the degree and term of the eqn above in curve fitting. But be cautious, the more we increase the degree, the more it'll become an interpolation (simply, joining the data) instead of a regression. So, it's a fine judgement!

Now, the **second question** is, how do define closest? I can't just say that this curve is the closest because my eyesight tells me so! As you know, human sight can be pretty deceiving! We need some mathematics here. That's where the concept of residual error comes in. Residual error is simply, how far or inaccurate a point in my derived line is from the actual corresponding data point, as shown in the figure below.



Theoretically, as we saw in last class, taking the square of the difference of $Y_{actual} - Y_{predicted}$ gives us a mathematical indication to which line/curve fits my dataset the best.

$$S_r = \sum_{i=1}^{n} \varepsilon_i^{\,2} = \sum_{i=1}^{n} \left(y_i - a_0 - a_1 x_i\right)^2$$

If S_r, that is sum of residual error as defined above, is minimum for the curve/line, it'd give the best fit curve for my dataset. Taking simply the difference or the mod of difference might result in multiple best fit curves generating the minimum value.

Now, this is in contradiction with what we saw in last class, right? Well, not really. Sr may be same for two curves/lines but it won't give minimum result for two different "best fit" curves/lines.

The lines we were trying in last class were not the best fit lines, they were merely two sample lines. Using the a0 and a1 eqns in slide 15, we can figure out that the actual regression eqn is 1+2x for the given dataset. Now, let's do the table and figure out if Ei^2 is the best way indeed. We can simplify Ei^2 as $(y-Y_p)^2$.

x	**y**	**x^2**	Yp for 4x-4	Ei^2 for 4x-4	Yp for y=6	Ei^2 for y=6	Yp for the **actual best fit 1+2x**	Ei^2 for 1+2x
2.0	4.0	4	4	0	6	4	5	1
3.0	6.0	9	8	4	6	0	7	1
2.0	6.0	4	4	4	6	0	5	1
3.0	8.0	9	8	0	6	4	7	1
Sr =				8		8		4

We see from the above table that, Sr is same for y= 4x-4 and y=6, which is not a problem really, since they are not the solution. But it is lower for the actual line which is 1+2x and no other line will give a lower or same value as Sr for 1+2x. (in this degree of eqn).

Then we used this assumption to generate two eqns for a0 and a1. That would give us the actual best fit line. Notice that, in doing so, we made the assumption first and then derived eqn that would fit the assumption.

So, with that accomplished, we can now easily write a MATLAB code using the matrix of the a0 and a1 eqns or just coding the eqns directly to find the best fit line that is y = a0 + a1 * x for a given data set. Mission accomplished! The solve using matrix method was shown in class.

Now, say we're really confident about our data and we want to bend our regression line according to the dataset. Then, It's time to jump into curve fitting using polynomial eqns.

<u>Fitting Nonlinear Curves:</u>

The difference between linear and non linear eqns is in their terms and degrees. The more terms and degrees an eqn has the more "curvier" it is.

$$Y = a_0 + a_1 x + a_2 x^2 + \ldots\ldots + a_n x^n$$

The above eqn is a polynomial eqn that'll generate a curve. Now, the problem is exactly the same as before. We need to find the best fit eqn. **Only difference is, here, we have to find a0, a1, a2…… a_n instead of just a0 and a1.** To do that we can use a generic eqn as derived in point c below. For example, for eqn of degree 2, we have to find a0, a1, and a2 (say for example, eqn set 2 as shown below). We can use the matrix method in MATLAB or just find the eqns mathematically.

But also, for eqns with terms less than 3, we could use a "cheat" way, we could convert the eqns from non-linear to linear by just taking logarithm in both side as shown in point a and b. This could easily be applied to power and exp functions.

a) **Power Function:** In this case a power function $y=ax^c$, will be fitted to the given data points (x_i, y_i), i=1,2,.....m. Taking logarithms in both sides , we obtain the relation ,

log y=log a +c log x

Which is of the form of a Straight line: $Y=a_0 +a_1X$, where Y=logy, a_0=log a, a_1=c and X=log x. Hence the procedure outlined in the previous section (Straight line fitting) can be followed to evaluate a_0 and a_1. Then a and c can be calculated from the formulae a_0=log a, and c=a_1.

b) Exponential function:
In this case an exponential function **$y=ae^{bx}$**, will be fitted to the given data points (x_i, y_i), i=1,2,.....m. Taking logarithms in both sides , we obtain the relation ,

ln y = ln a + b x

Which is of the form of a Straight line: $Y=a_0 +a_1X$, where Y=lny, a_0=ln a, a_1=b and X= x. Hence the procedure outlined in the previous section (Straight line fitting) can be followed to evaluate a_0 and a_1. Then a and b can be calculated from the formulae a_0=ln a, and b=a_1.

c) Polynomial of nth Degree

Let the the polynomial eqn of nth degree-

$$Y = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n$$

be fitted to the given data points (**x_i, y_i**), i=1,2,.....m.

Then , the sum of the squares of the errors at the given points x_i (I=1,2,3…..m) is

$$S = e_1^2 + e_2^2 + e_3^2 + ... + e_m^2$$

$$= [y_1 - f(x_1)]^2 + [y_2 - f(x_2)]^2 + [y_3 - f(x_3)]^2 - ....[y_m - f(x_m)]^2$$

We then have,

$$S = [y_1 - (a_0 + a_1 x_1 + a_2 x_1^2 + ... + a_n x_1^n)]^2 + [y_2 - (a_0 + a_1 x_2 + a_2 x_2^2 + ... + a_n x_2^n)]^2 + ... +$$

$$[y_m - (a_0 + a_1 x_m + a_2 x_m^2 + ... + a_n x_m^n)]^2 +$$

Equating the partial derivatives $\partial S/\partial a_0=0$, $\partial S/\partial a_1=0$,..... $\partial S/\partial a_n=0$ and simplifying we get

$$m a_0 + a_1 \sum_{i=1}^{m} x_i + a_2 \sum_{i=1}^{m} x_i^2 + a_3 \sum_{i=1}^{m} x_i^3 + ... + a_n \sum_{i=1}^{m} x_i^n = \sum y_i$$

$$a_0 \sum_{i=1}^{m} x_i + a_1 \sum_{i=1}^{m} x_i^2 + a_2 \sum_{i=1}^{m} x_i^3 + ... + a_n \sum_{i=1}^{m} x_i^{n+1} = \sum x_i y_i$$

$$a_0 \sum_{i=1}^{m} x_i^2 + a_1 \sum_{i=1}^{m} x_i^3 + a_2 \sum_{i=1}^{m} x_i^4 + ... + a_n \sum_{i=1}^{m} x_i^{n+2} = \sum x_i^2 y_i$$

.....

$$a_0 \sum_{i=1}^{m} x_i^n + a_1 \sum_{i=1}^{m} x_i^{n+1} + a_2 \sum_{i=1}^{m} x_i^{n+2} + ... + a_n \sum_{i=1}^{m} x_i^{2n} = \sum x_i^n y_i$$

----------eqn set (1)

The length and width of the above equations depend on the value of n. Say for example for 2nd degree polynomial

$$Y = a_0 + a_1 x + a_2 x^2$$

the simultaneous equations become

$$ma_0 + a_1\sum_{i=1}^{m} x_i + a_2\sum_{i=1}^{m} {x_i}^2 = \sum y_i$$

$$a_0\sum_{i=1}^{m} x_i + a_1\sum_{i=1}^{m} {x_i}^2 + a_2\sum_{i=1}^{m} {x_i}^3 = \sum x_i y_i$$

$$a_0\sum_{i=1}^{m} {x_i}^2 + a_1\sum_{i=1}^{m} {x_i}^3 + a_2\sum_{i=1}^{m} {x_i}^4 = \sum {x_i}^2 y_i$$

----------eqn set (2)

now knowing the values of $\sum_{i=1}^{m} x_i, \sum_{i=1}^{m} {x_i}^2, \sum_{i=1}^{m} {x_i}^3, \sum_{i=1}^{m} {x_i}^4, \sum x_i y_i, \sum {x_i}^2 y_i$ (from the given data points), the above simultaneous equations can be solved for a0,a1,a2 by Numerical method.(Gauss- Jordan or Gauss- Seidal) and hence we can determine the 2nd degree polynomial equation $Y=a_0+a_1x+a_2x^2$ to fit the data points.

The MATLAB Codes:

Good on you if you understood all of the above. Well done!

But if you didn't, not to worry, MATLAB has got you covered with it's magical built in functions! Understand this only, we need to find a0, a1, a2……a_n (for eqn set 1) instead of just a0, a1 depending on the terms and order of my polynomial in case of curve fitting.

To do that, we'll use built in functions polyfit and polyval. **The uses of these functions are explained in comments when used in code**. (see code 2 – using built in func).

So, to summarize, we can either use lines or curves to fit our given data. And we can find our desired best fit linear or polynomial eqns by using a0, a1….an' s eqns or by using matrix. Or we can simply use the built in functions in MATLAB.

Go through the codes, the ppt (all the slides) and understand the concepts. And you should be good to go for the online!