Overview #2: Least squares and curve fitting – how to deal with polynomial fits

Pertinent reading for Problem #2:

Strang (Introduction to linear algebra, 5th ed):

(Download from Blackboard, under /Resources / Linear algebra texts)

Ch. 4.3: pages 223 - 225 (linear fit and $A^T Ax = A^T b$)

pages 226 - 227 (quadratic fit example)

Lay (Linear algebra, 4th ed) :

(Download from Blackboard, under /Resources / Linear algebra texts)

Ch. 6: pages 368 - 373 (polynomial fitting examples)

A quick reminder of the least-squares $\mathrm{Ax} = \mathrm{b}$ problem from class

Given an unsolvable system of equations of the form:

$$Ax = b$$

where A is not *full-ranked* (ie. The number of linearly independent column vectors of A is not enough to span the same vector space as "b"), we saw in class that a *least-squares* approximate solution to that equation can be found by solving:

$$A^T A x = A^T b$$

For instance, in the example we did in class, we were trying to fit a set of data points with a 1st-order polynomial model:

$$m \cdot t + c = y$$

using multiple data points (t_i, y_i) = (-1, -1), (1, 2), (2, 6), and (4,7). To construct the least-squares $Ax = b$ equation, we will write down 4 equations in which we "force" our data points to obey the same polynomial model:

$$\begin{array}{lcccccc} Model: & t \cdot m & + & c & = & y \\ \hline Point\ \#1: & -1\,m & + & c & = & -1 \\ Point\ \#2: & 1\,m & + & c & = & 2 \\ Point\ \#3: & 2\,m & + & c & = & 6 \\ Point\ \#4: & 4\,m & + & c & = & 7 \end{array}$$

$$\xrightarrow{Write\ in\ matrix\ form} \quad \underbrace{\begin{bmatrix} -1 & 1 \\ 1 & 1 \\ 2 & 1 \\ 4 & 1 \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} m \\ c \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} -1 \\ 2 \\ 4 \\ 7 \end{bmatrix}}_{b}$$

If we were to solve the least-squares equation using brute-force inverses, we would write:

$$A^T A x = A^T b \quad \xrightarrow{solve\ for\ x} \quad x = \underbrace{(A^T A)^{-1} A^T}\ b$$

This is called the *"pseudoinverse"*
of a rectangular matrix A

Now, since we are in a numerical linear algebra class, you know that it's much more efficient to use LU factorization to solve this problem ! In matlab, you would solve for x by typing:

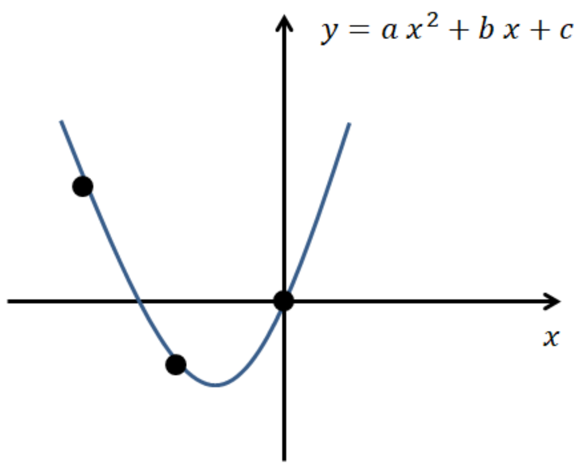
$$x = (A' * A) \backslash (A' * b)$$

Least-squares fit for ***linear*** combinations of polynomial ***functions***

In this problem set, we're going to explore another type of least-squares problem. Suppose that we have $n = 11$ data points in which we would like to fit a polynomial function through them.

Point	x	y
A	-1.5	-1.5
B	-1.2	0.20
C	-0.9	0.04
D	-0.6	-0.21
E	-0.3	0.15
F	0	0.71
G	0.3	1.00
H	0.6	1.14
I	0.9	0.70
J	1.2	-0.25
K	1.5	0

From college algebra, analytical geometry, or Calculus I in undergrad, recall that for a given polynomial of degree m, you need at least $(m + 1)$ points to truly define that polynomial function. As an example, if you wish to construct an unique parabola of polynomial order $m = 2$, you need at least $m + 1 = 3$ data points to do it !



At least 3 data points are needed to define a parabola !

Figure 1: You need at $m + 1$ data points to define an unique polynomial fit of order m

Now, since our current problem has $n = 11$ data points, logic suggest the highest-order polynomial that will fit through these points is a $(n - 1) = 10$-degree polynomial. However, we're going are going to make our data fit less-stringent in this example (such that we avoid the so-called *over-fitting* effect). Let's fit our data points using a $m =$ 5-degree polynomial instead !

$$p_5(x) = y \; = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \; c_4 x^4 \; + \; c_5 x^5$$

Just like our linear least-squares linear fit example on the previous page, we can plug all 11 data points into our polynomial model to construct a least-squares $Ax = b$ problem:

$$\begin{array}{llllllll}
Point\ A: & c_0 & +c_1(-1.5) & +c_2(-1.5)^2 & +c_3(-1.5)^3 & +c_4(-1.5)^4 & +c_5(-1.5)^5 & = -1.50 \\
Point\ B: & c_0 & +c_1(-1.2) & +c_2(-1.2)^2 & +c_3(-1.2)^3 & +c_4(-1.2)^4 & +c_5(-1.2)^5 & = 0.20 \\
Point\ C: & c_0 & +c_1(-0.9) & +c_2(-0.9)^2 & +c_3(-0.9)^3 & +c_4(-0.9)^4 & +c_5(-0.9)^5 & = 0.04 \\
 & & & & \vdots & & & \\
Point\ K: & c_0 & +c_1(1.5) & +c_2(1.5)^2 & +c_3(1.5)^3 & +c_4(1.5)^4 & +c_5(1.5)^5 & = 0
\end{array}$$

Rewriting everything in matrix form, we get:

$$\left[\begin{array}{c|c|c|c|c|c} 1 & -1.5 & (-1.5)^2 & (-1.5)^3 & (-1.5)^4 & (-1.5)^5 \\ 1 & -1.2 & (-1.2)^2 & (-1.2)^3 & (-1.2)^4 & (-1.2)^5 \\ 1 & -0.9 & (-0.9)^2 & (-0.9)^3 & (-0.9)^4 & (-0.9)^5 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1.5 & (1.5)^2 & (1.5)^3 & (1.5)^4 & (1.5)^5 \end{array}\right] \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix} = \begin{bmatrix} -1.50 \\ 0.20 \\ 0.04 \\ \vdots \\ 0 \end{bmatrix}$$

It's important to realize the column vectors of the giant matrix contains *__digitized samples of our basis functions__* __$1, x, x^2, x^3, x^4$, *and* x^5__ . The short-hand way of writing them out is:

$$\left[\begin{array}{c|c|c|c|c|c} \vec{1} & \vec{x} & \overrightarrow{x^2} & \overrightarrow{x^3} & \overrightarrow{x^4} & \overrightarrow{x^5} \end{array}\right] \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{11} \end{bmatrix}$$

$$X \qquad c \quad = \quad y$$

As usual, one can solve for the least-squares approximation of "c" by solving for:

$$X^T X\ c = X^T y$$

The resulting coefficients will be the least-squares 5th order polynomial fit for the 11 data points !

$$c = (X^T X)^{-1} X^T y = \begin{bmatrix} 0.6135 \\ 2.0153 \\ -0.1814 \\ -2.9535 \\ -0.1874 \\ 1.0117 \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix}$$

This means the least-squares polynomial fit is:

$$y = 0.61 + 2.02\,x - 0.18\,x^2 - 2.95\,x^3 - 0.19\,x^4 + 1.01\,x^5$$

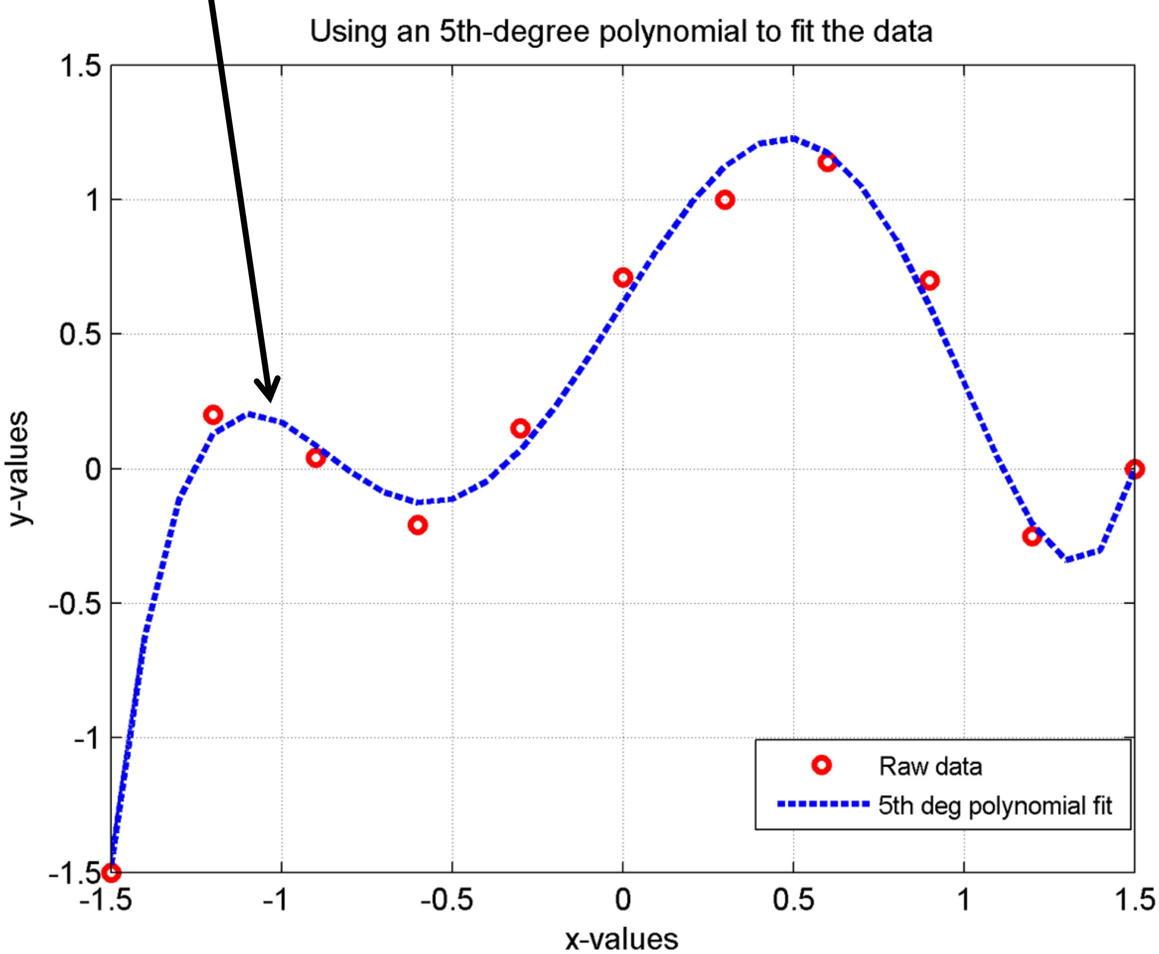


Figure 1: The 5th order least-squares polynomial fit for our 11 data points !