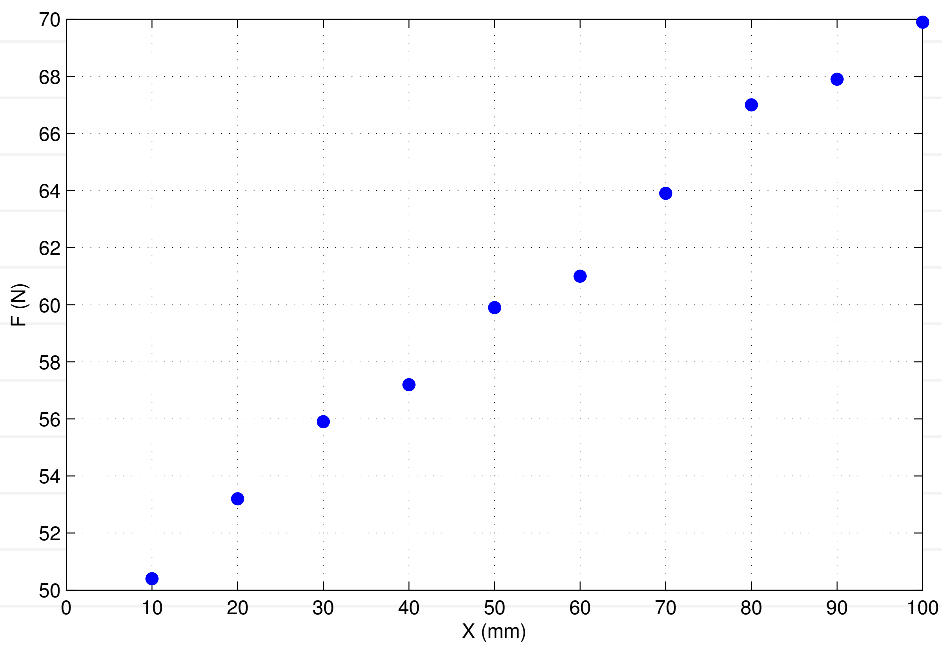
Least squares regression

Suppose we have a sample of two measurands: (1) the force F through a spring and (2) its displacement x from equilibrium. We would like to determine an analytic function that relates these variables for this spring. We assume that the displacement is measured very accurately + precisely. There is some variation in the force. Below is a plot of the data.




```
x = ... % (m) ... displacement across spring
  ( 0:0.01:.1 ).';

F = ... % (N) ... force through spring
  [ ...
    50.1
    50.4
    53.2
    55.9
    57.2
    59.9
    61.0
    63.9
    67.0
    67.9
    70.3
  ];
```

Let y denote the analytic function that we would like to fit to the data. Let y_i denote the value of $y(x_i)$, where x_i

is the i^{th} value of r.v. X from the data. Then we want to minimize the differences between F_i and y_i.

Minimizing is often achieved by writing a function that represents the value we'd like to minimize, differentiating that function, and solving for the zero-crossings (which correspond to either local maxima or minima).

This is the function of x (remember: $y_i = y(x_i)$) we would like to minimize.

Now we must choose a form for y. In this course, we choose y to be a polynomial with coefficients a_j:

If we treat D as a function of a_j, i.e. $D(a_0, a_1, \cdots, a_m)$, and minimize D for each value x_i, we must take the partial derivative of D with respect to each a_j, and set each equal to 0:

This gives us N equations and $m+1$ unknowns. Writing the system in matrix form,

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^m \\ 1 & x_2 & x_2^2 & \cdots & x_2^m \\ 1 & x_3 & x_3^2 & \cdots & x_3^m \\ \vdots & \vdots & \vdots & \ddots & \\ 1 & x_N & x_N^2 & & x_N^m \end{bmatrix} \begin{bmatrix} \\ \\ \\ \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \\ \\ \end{bmatrix} .$$

This is an overdetermined system (usually, since usually $N > m$, i.e. the sample size is greater than the polynomial order that we want to fit to the data). Therefore we can't solve the system by taking A^{-1} because A doesn't have an inverse!

Instead, we either find the Moore-Penrose pseudo-inverse A^{+}, which is inefficient, $\vec{a} = A^{+}\vec{b}$ — or we can approximate b with an algorithm such as that used by Matlab's "\" operator. In the latter case, $\vec{a} = A \backslash \vec{b}$.