Wanna see something cool?

Remember how you can make a straight line from 2 points, a parabola (quadratic polynomial) from 3 points, a cubic polynomial from 4 points, etc?

Let's take the parabola as an example. Any 3 points (x-y pairs) uniquely define a parabola, which has the general form:

$$y = ax^2 + bx + c \tag{1.1}$$

where *a*, *b*, and *c* are constants that we need to determine. Let's say our 3 x-y pairs are (x_1, y_1), (x_2, y_2), and (x_3, y_3). Since there are 3 unknowns (a,b,c) we need to come up with 3 independent equations. All we do is plug each of the 3 x-y pairs into (1.1):

$$y_1 = ax_1^2 + bx_1 + c$$
$$y_2 = ax_2^2 + bx_2 + c$$
$$y_2 = ax_2^2 + bx_2 + c$$

You're pretty much done at this point, cuz you've written 3 equations that can be used to solve for 3 unknowns. In standard ($\mathrm{A}\mathbf{x} = \mathbf{b}$) matrix form, the above 3 equations can be written:

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \tag{1.2}$$

You can solve this using any method you want, such as Gaussian elimination, Kramer's rule, inverting matrix *A*, etc. (Solving a linear system of equations like this is *the* most important thing to learn in linear algebra.) After solving this system, you've got the 3 constant coefficients that describe the parabola that goes through your 3 points.

Now we get to the cool thing. What if you have 145 x-y points that you'd like to fit to a parabola to. By plugging in each point to (1.1), you could come up with 145 independent equations. But the problem is we only need 3 equations. In other words, we've over-constrained the problem. To generalize a bit, let's say we have *N* points. Then we can write *N* equations:

$$y_1 = ax_1^2 + bx_1 + c$$
$$y_2 = ax_2^2 + bx_2 + c$$
$$\vdots$$
$$y_N = ax_N^2 + bx_N + c$$

We can also put these equations into the ($\mathrm{A}\mathbf{x} = \mathbf{b}$) matrix form:

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ \vdots & \vdots & \vdots \\ x_N^2 & x_N & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{1.3}$$

But can you solve this system of equations? Let's first look at the dimensions of the matrices above. The *A* matrix has *N* rows and *3* columns, so its dimension is *Nx3*. The **x** matrix is a *3x1* vector. And the b matrix is an *Nx1* vector. Recall that an *Nx3* matrix multiplied by a *3x1* vector

results in an *Nx1* vector. So at least we're allowed to write (1.3) without breaking any rules. But we can't solve for the **x** vector, because the *A* matrix is not a square matrix. Here's where the craziness comes in...

Before we get too crazy let me write (1.3) in a more general form:

$$A_{Nx3}\,\mathbf{x}_{\mathbf{3x1}} = \mathbf{b}_{\mathbf{Nx1}}, \qquad (1.4)$$

where the subscripts indicate the dimensions of the matrices.
Now, say I want to force (1.4) into some form that I do know how to solve. Wait. I mean put it into the *only* form I know how to solve: standard ($A\,\mathbf{x} = \mathbf{b}$) matrix form, with *A* being a square matrix. Multiply (1.4) by the transpose of matrix A (look up the transpose if you need to):

$$A^{T}_{3xN}\,A_{Nx3}\,\mathbf{x}_{\mathbf{3x1}} = A^{T}_{3xN}\,\mathbf{b}_{\mathbf{Nx1}} \qquad (1.5)$$

Look what we've done. When you multiply any matrix (square or not) by its transpose, the resulting matrix has to be square. Let's define some stuff here:

$$\tilde{A}_{3x3} = A^{T}_{3xN}\,A_{Nx3}$$

$$\tilde{\mathbf{b}}_{\mathbf{3x1}} = A^{T}_{3xN}\,\mathbf{b}_{\mathbf{Nx1}}.$$

So we can rewrite (1.5) more cleanly:

$$\tilde{A}_{3x3}\,\mathbf{x}_{\mathbf{3x1}} = \tilde{\mathbf{b}}_{\mathbf{3x1}}. \qquad (1.6)$$

And amazingly, you can solve that bad boy for $\mathbf{x}_{\mathbf{3x1}}$. But don't get too excited, because what does $\mathbf{x}_{\mathbf{3x1}}$ represent? They're the 3 coefficients of *some* quadratic polynomial. But what polynomial? We over-constrained the problem by having more equations than unknowns. This is taking longer than I thought it would, so I'm going to skip the freaking beautiful proof, but here's the kicker. It just so happens that the coefficients you solve for in (1.6) describe THE least-squares best-fit parabola through your *N* data points. I'm sure you guys have done least-squares curve fits to data before, but not this way!

Let's abstract our results to the most general case. Say you have a boat-load of data points that you'd like to fit a particular type of curve to (it doesn't have to be just a polynomial). Write out the general form of the curve like we did in (1.1). Then plug each of your data points into this equation to yield a boat-load of independent equations. Then write these equations in matrix form:

$$A\,\mathbf{x} = \mathbf{b}.$$

You can't solve for **x** directly cuz *A* isn't square, so multiply both sides by the transpose of *A*:

$$A^{T}\,A\,\mathbf{x} = A^{T}\,\mathbf{b}.$$

You can rewrite it cleanly as:

$$\tilde{A}\,\mathbf{x} = \tilde{\mathbf{b}},$$

where $\tilde{A} = A^{T}\,A$ and $\tilde{\mathbf{b}} = A^{T}\,\mathbf{b}$. Just solve this linear system for **x**, and you've got the least-squares fit to your data. If you're down with inverting a square matrix (one of many ways to solve a linear system), you can write:

$$\mathbf{x} = \tilde{A}^{-1}\,\tilde{\mathbf{b}},$$

or in terms of the original matrices:

$$\mathbf{x} = \left(A^{T}\,A\right)^{-1}A^{T}\,\mathbf{b}. \qquad (1.7)$$

To me, equation (1.7) is a freaking thing of beauty. In one shot, you've got a totally genera-purpose formula for least squares curve fitting. We didn't use any calculus (although the proof needs it). All we did was write a bunch of equations, put 'em in matrix form, multiply both sides by something (A-transpose), and do a matrix inversion (or linear system solution). The standard way of doing least squares is also cool, but the derivation is different for every type of curve.

BTW: If you're gonna be doing any real matrix calculations (dimensions higher than 3) you need to be using a good mathematical tool, such as MATLAB. You can use it free from the USC UNIX sever. I'll show you how.