Matlab 8 Linear Regression

(and polynomial regression too!)

What is Linear Regression?

Linear regression is a means of modeling data values as a function of other values (modeling y as a function of x)

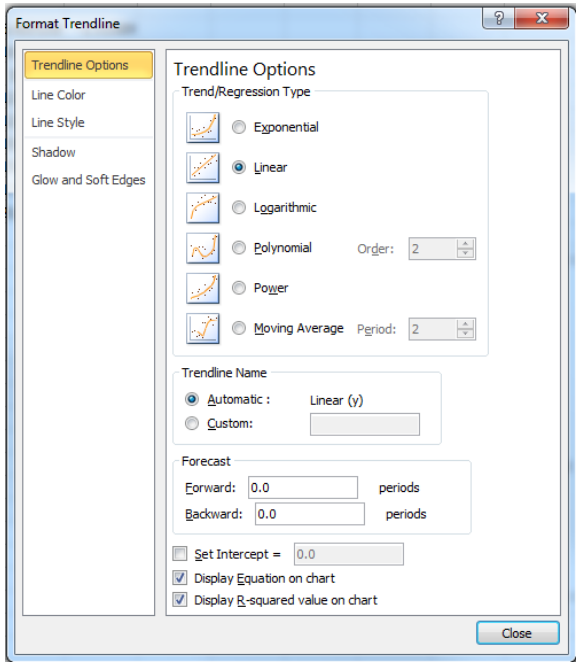
You have already seen linear regression in Excel (although you may not have realized it.)

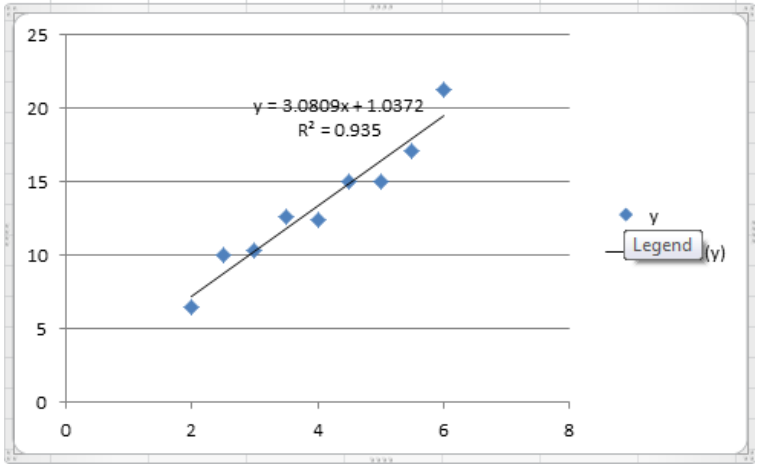
When you use the trend line tool in Excel, and you fit a straight line, you have done linear regression.

It's also very easy to do in Matlab. We'll discuss how to do it, and how to tell if what you've done makes sense.

We'll also try to tie it to some of the confidence interval stuff we've been doing.

Remember how to do this?





Hey, Linear Regression seems like a pretty powerful tool

Can anyone use it, or do I need a license?

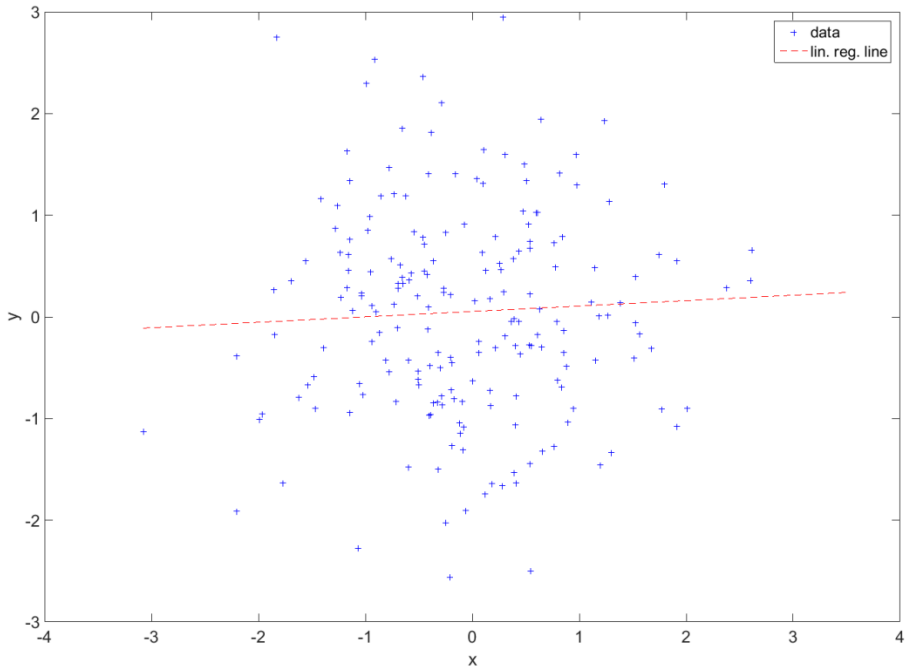
Who is allowed to do linear regression?

Actually, anyone can do linear regression

The danger is that you will always get an answer, but that answer may not tell you much about the relationship between your x and y variables.

This man is allowed to do linear regression.
He does it. Almost every year. Nationally.

This is a lousy regression. Don't do this.



Q:I can do linear regression?!
A: Yes. But you can't sing.

Mathematical model for linear regression

Linear regression assumes that the variable y is a linear function of the variable x, plus some random error ε:

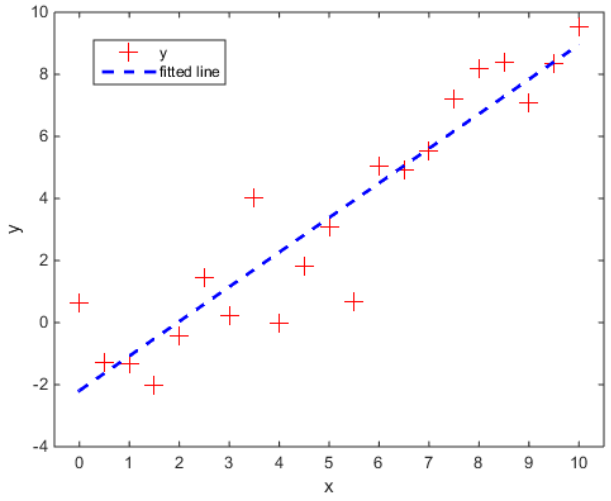
$$y = b_1 x + b_0 + \varepsilon$$

If we have n pairs of x and y data, we can write for each of the pairs

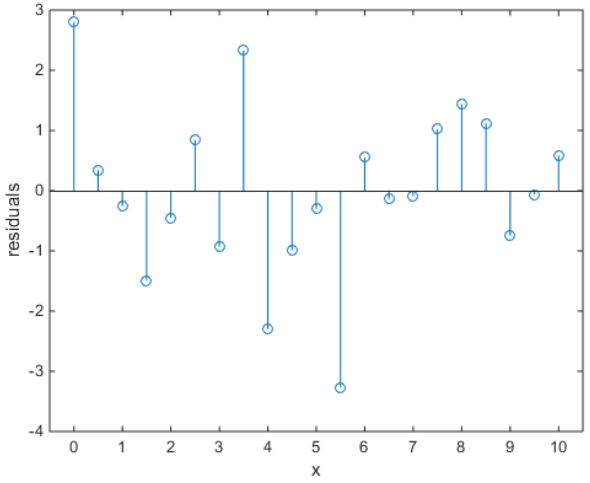
$$y_i = b_1 x_i + b_0 + \varepsilon_i$$

Given values for the slope b_1 and the intercept b_0 and the data pairs, we can calculate values for the error terms ε_i:

$$\varepsilon_i = y_i - b_1 x_i - b_0$$



Subtract the blue line from the red crosses; get the residuals shown below



Mathematical model for linear regression (2)

Our goal in linear regression is to find values of slope b_1 and the intercept b_0 which minimize the sum of the squared errors.

$$minimize \sum_{i=1}^{n} (\varepsilon_i)^2 = \sum_{i=1}^{n} (y_i - b_1 x_i - b_0)^2$$

Why squared?* Why not the sum of the errors?

$$minimize \sum_{i=1}^{n} \varepsilon_i = \sum_{i=1}^{n} (y_i - b_1 x_i - b_0)$$

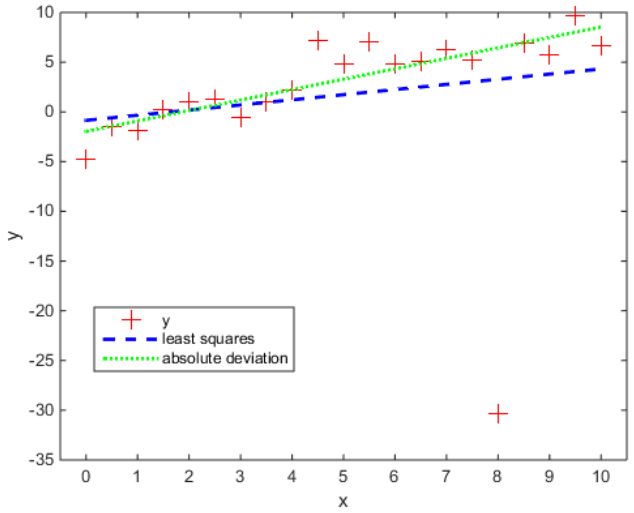
Or the sum of the absolute errors?

$$minimize \sum_{i=1}^{n} |\varepsilon_i| = \sum_{i=1}^{n} |y_i - b_1 x_i - b_0|$$

*This material is here because there's always someone who asks why we square the errors…

The sum of errors formulation is just not constrained enough, so you get nonsense. By letting b_1 and b_0 go to $\pm\infty$ (depending on the values of x) we can make the sum of errors go to $-\infty$. That's not a very useful result.

People do sometimes use the sum of absolute errors especially in robust regression. It deals better with data sets that have outliers.



The outlier at x = 8 pulls the least-squares line (blue) down more than the absolute error line (green), which stays through the main body of the points.

Minimizing the sum of squared errors is easier (results can be derived using calculus), is more efficient (fewer data points are needed for the same level of accuracy), and gives a unique answer (it is possible in special cases to have more than one set of $[b_0, b_1]$ that give the same minimum sum in the sum of absolute errors).

We'll stick using results derived using the sum of the squared errors AKA least-squares regression.

So quit your dawdling – show me the math already!

(I’m an engineering student – not one of those namby-pamby history majors)

Mathematical model for linear regression (3)

We won't actually derive the equations for b_0 and b_1 in this course – the derivation involves partial derivatives and linear algebra (you will be able to do this when you're sophomores and juniors.)

Knowing how to do the math is good, because you can extend the approach to Generalized Linear Regression, with which you can fit arbitrarily complicated models – useful when the physics of you engineering problem are complicated.

Instead, we're going to use a Matlab built-in function that will do the math calculations for us.

Ok, here's the math, but don't worry, this isn't on the test. We just need something to fill this space.

$$
\begin{aligned}
S_n &= \sum_{i=1}^{n} (y_i - b_1 x_i - b_0)^2 \\
\frac{\partial S_n}{\partial b_0} &= \sum_{i=1}^{n} -2\,(y_i - b_1 x_i - b_0) = 0 \\
\frac{\partial S_n}{\partial b_1} &= \sum_{i=1}^{n} -2x_i\,(y_i - b_1 x_i - b_0) = 0 \\
\sum_{i=1}^{n} y_i - b_1 \sum_{i=1}^{n} x_i - \sum_{i=1}^{n} b_0 &= 0 \\
\sum_{i=1}^{n} x_i y_i - b_1 \sum_{i=1}^{n} x_i^2 - b_0 \sum_{i=1}^{n} x_i &= 0 \\
n b_0 + \left(\sum_{i=1}^{n} x_i\right) b_1 &= \sum_{i=1}^{n} y_i \\
\left(\sum_{i=1}^{n} x_i\right) b_0 + \left(\sum_{i=1}^{n} x_i^2\right) b_1 &= \sum_{i=1}^{n} x_i y_i \\
\begin{bmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} &= \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \end{bmatrix} \\
Ab &= y \\
b &= inv(A) * y
\end{aligned}
$$

Actual equations for b_1 and b_0

Ok, so maybe you're driving out past Marfa, and you lose your wifi connection, and so your Matlab won't run, and you need to do your ENGR 112 regression homework in the middle of Big Bend National Park, so what do you do?

Well, for linear regression, the equations for slope and intercept aren't too bad.

$$b_1 = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - \left(\sum x_i\right)^2}$$

$$b_0 = \frac{1}{n}\sum y_i - \frac{b_1}{n}\sum x_i$$

The equations get more complicated for more complicated regression models, so people generally take the linear algebra route.

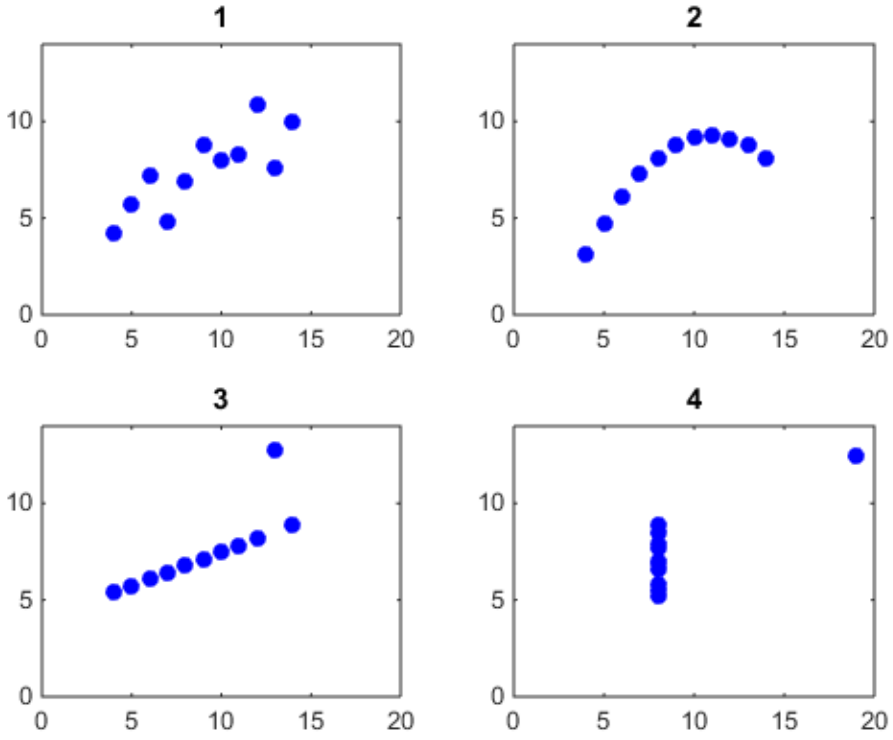
Your fancy Matlab-regression ain't no good out here

But wait!!!
Always plot your data first!!!

Simple visual inspection will tell you a lot about whether linear regression (or any regression at all) makes sense.

Consider these 4 data sets:

x1	y1	x2	y2	x3	y3	x4	y4
10	8.04	10	9.14	10	7.46	8	6.58
8	6.95	8	8.14	8	6.77	8	5.76
13	7.58	13	8.74	13	12.74	8	7.71
9	8.81	9	8.77	9	7.11	8	8.84
11	8.33	11	9.26	11	7.81	8	8.47
14	9.96	14	8.1	14	8.84	8	7.04
6	7.24	6	6.13	6	6.08	8	5.25
4	4.26	4	3.1	4	5.39	19	12.5
12	10.84	12	9.13	12	8.15	8	5.56
7	4.82	7	7.26	7	6.42	8	7.91
5	5.68	5	4.74	5	5.73	8	6.89



(From Anscombe, 1973)

PLOT YOUR DATA!

The same linear regression line (to 2 decimal places) fits all 4 of these data sets!

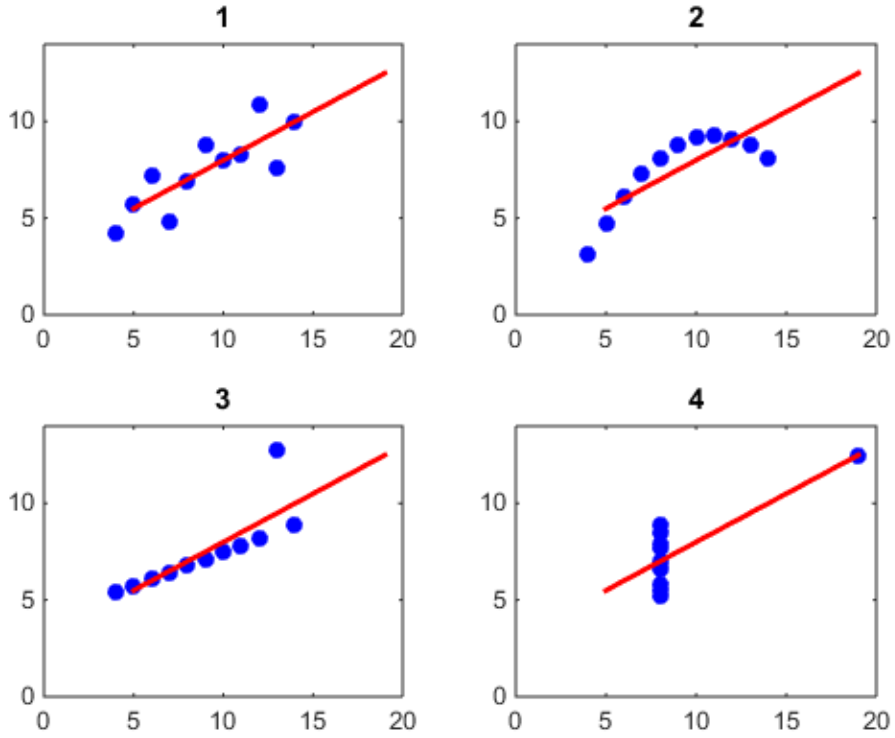
Example 1 is a good regression.

Example 2 is a quadratic – why are we fitting a straight line to it?

Example 3 should have a lower slope, but the outlier is affecting it.

Example 4 is just weird.

Use your engineering judgement before blindly accepting numbers (an important lesson for times beyond this class.)



Matlab regression

We can use the built-in Matlab function `polyfit` to do polynomial regression – linear regression is the simplest type of polynomial regression.

Given the instructions from the help shown right, it should be apparent that we can fit a line (1st order polynomial) by the command

```
P = polyfit(X,Y,1)
```

We will get back the 2-element array P.
The first element P(1) is the slope
The second element P(2) is the intercept

```
>> help polyfit
 polyfit Fit polynomial to data.
    P = polyfit(X,Y,N) finds the
coefficients of a polynomial P(X) of
degree N that fits the data Y best in a
least-squares sense. P is a row vector
of length N+1 containing the polynomial
coefficients in descending powers,
P(1)*X^N + P(2)*X^(N-1) +...+ P(N)*X +
P(N+1).
```

Let's get to it!

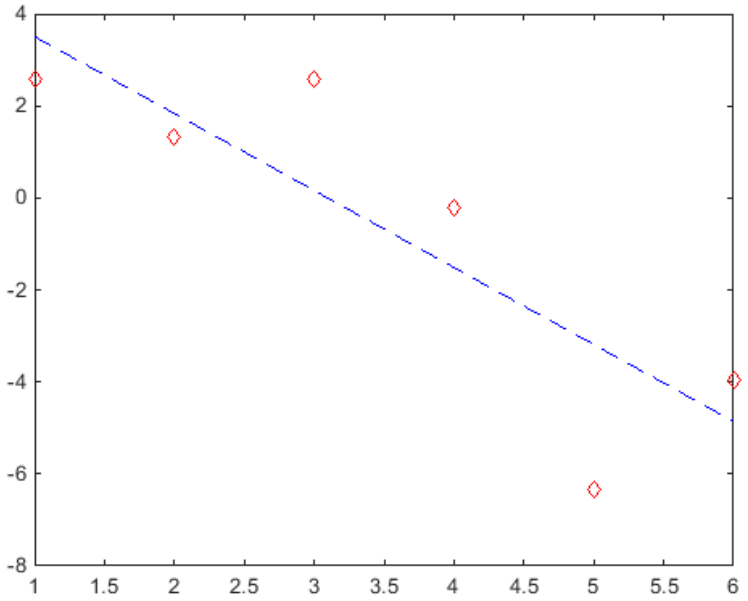
The code to the right is all you need to do simple linear regression. Matlab makes it easy because it's a very commonly done task.

Your x and y will of course be different.

```
x = 1:6;
y = [2.57 1.34 2.58 -0.23 -6.35 -3.97];
p = polyfit(x,y,1);

yfit = p(1)*x+p(2);

figure
plot(x,y,'rd',x,yfit,'b--')
```



How do we know if our regression is good?

For linear regression, one of the classic measures is r^2, AKA the correlation coefficient. You saw this when fitting trend lines in Excel; outputting r^2 was an option.

r^2 can be calculated using the formula at the right. It's straightforward to implement it in Matlab using the sum function repeatedly.

An r^2 close to 1 is good, and shows that x and y are strongly related. An r^2 close to zero shows that x and y have little relationship. But how large r^2 needs to be to make an investigator depends on the field.

Be aware: r^2 only tells you about <u>linear</u> relationships. It is useless for measuring higher-order dependence.

$$r = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - \left(\sum x_i\right)^2}\sqrt{n\sum y_i^2 - \left(\sum y_i\right)^2}}$$

and you get r^2 by squaring the result.

```
x = 1:6;
y = [2.57 1.34 2.58 -0.23 -6.35 -3.97];
p = polyfit(x,y,1);

yfit = p(1)*x+p(2);

figure
plot(x,y,'rd',x,yfit,'b--')

n = length(x);
r = (n*sum(x.*y) - sum(x)*sum(y))/(sqrt(n*sum(x.^2)-
sum(x)^2)*sqrt(n*sum(y.^2)-sum(y)^2))
r2 = r^2
```

You get r = -0.8463 and r2 = 0.7162. The negative value of r means that there is an inverse relationship between x and y; as x grows, y decreases.

Higher-order polynomial regression

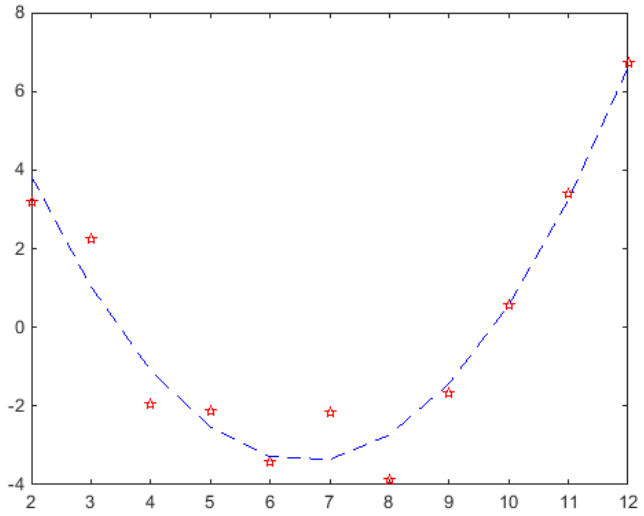
We can use polyfit to fit higher-order polynomials to data sets. Simply change the last input from 1 to a higher integer.

```
x = 2:12;
y = [3.19,2.23,-1.97,-2.13,-3.43,-2.18,-3.89,
-1.67,0.55,3.40,6.74];

p = polyfit(x,y,2); % it's 2 for a quadratic!

yfit = p(1)*x.^2 + p(2)*x + p(3);

figure
plot(x,y,'rp',x,yfit,'b--')
```



Higher-order polynomial regression (2)

Again, because polynomial regression is done so much, Matlab has a built-in function for creating the fitted line.

On the previous slide we used

```
yfit = p(1)*x.^2 + p(2)*x + p(3);
```

to make the fitted line. In place of that command, we can use the command `polyval`.

```
>> help polyval
 polyval Evaluate polynomial.
    Y = polyval(P,X) returns the
value of a polynomial P evaluated
at X. P is a vector of length N+1
whose elements are the
coefficients of the polynomial in
descending powers.

Y = P(1)*X^N + P(2)*X^(N-1) + ...
+ P(N)*X + P(N+1)
```

```
x = 2:12;
y = [3.19,2.23,-1.97,-2.13,-3.43,-2.18,-3.89,
-1.67,0.55,3.40,6.74];

p = polyfit(x,y,2);

%yfit = p(1)*x.^2 + p(2)*x + p(3);
yfit = polyval(p,x);  % using polyval instead
of writing out the polynomial

figure
plot(x,y,'rp',x,yfit,'b--')
```

(Hey if you want to see the plot, flip back a slide. It's the same thing.)

Measure of fit for higher order polynomials

We can't use r^2 with higher-order polynomial regression. Instead we introduce another measure of goodness-of-fit: the Mean-Squared Error, or MSE.

Way back on Slide 6 we introduced the squared errors

$$\sum_{i=1}^{n}(\varepsilon_i)^2=\sum_{i=1}^{n}(y_i-b_1x_i-b_0)^2$$

(remember this?)

If we take the average (mean) of the squared errors, we have

$$MSE=\frac{1}{n}\sum_{i=1}^{n}(\varepsilon_i)^2$$

A good regression model should lead to smaller MSE.

```
x = 2:12;
y = [3.19,2.23,-1.97,-2.13,-3.43,-2.18,-3.89,
-1.67,0.55,3.40,6.74];

p = polyfit(x,y,2);

yfit = polyval(p,x);

e = y - yfit;
MSE = mean(e.^2);

figure
plot(x,y,'rp',x,yfit,'b--')
fprintf('The MSE of this quadratic regression
is %f\n',MSE)
```

```
>> RegTest001
The MSE of this quadratic regression is
0.511679
```

But the absolute value of MSE is affected by the magnitude of the data, so this needs to be a relative comparison.

There's a lot more to the analysis of the errors (AKA residuals)

You can learn about this in an advanced statistics course. For now, you know enough to be dangerous.

Transformations and Polynomial Regression

You may remember from last term the transformations you can use to turn a non-linear model into a linear one. **Consult Ch. 13 of the text for review.**

With polynomial regression you can fit any data to arbitrary accuracy without transformation by using a sufficiently high-order polynomial model, given enough data. (Proof skipped)

So why transform?

(Why transform a perfectly good toy into a lousy movie?)

Sometimes the physics of the process behind the data set argue for a logical transformation. For example, decay processes have physical/mathematical reasons for following an exponential model. While a polynomial can be used to fit the decay data, it would make more sense to log-transform, and then do linear regression.

Use your engineering judgement