How can I fit a curve to my data?

How can I fit a curve to my data?

There are 2 ways to fit a curve through your data,

- Fit cubic splines which pass through each data point.
- Find a function that 'best' fits your data.

Cubic spline

The cubic spline method fits separate curves between each pair of data points (subject to the requirement that the curves must match both in value and in 1st derivative at each data point). When you have very few data points and you want to draw a nice smooth curve through them, cubic splines work well. However when you have lots of data points or the data is scattered, cubic splines are not the best choice.

Here is an example of how to use cubic splines,

```
>> x=1:5;

 >> y=[1 3 5 3 1];    % sample data

 >> plot(x,y,'*')     % Shows raw data points

 >> xi=1:.1:5         % new, denser independent variable

 >> yi=spline(x,y,xi);% calculate the splines at each point in xi

 >> hold on           % hold previous plot

 >> plot(xi,yi)       % show fitted curve
```

Best fit

If your data is scattered, or you need to determine the slope, intercept or functional form of your data, the best approach to use is to fit a function to your data. The easiest way to do this in matlab is to use the curvefitp and curvefitnl functions.

CURVEFITP (a local function, not provided by the MathWorks) fits a nth order polynomial to your data and plots the resulting curve along with your data points. n can be any integer in principal, but after about 9 you start to get errors. n=1 is the special case of a linear fit.

Here is an example of how to use CURVEFITP,

```
>> x=1:10;

 >> y=[1 5 8 15 26 37 48 65 82 99];

 >> c=curvefitp(x,y,1)               % linear fit
```

This plots the data points and a fitted curve. Slope and intercept are in c(1) and c(2), respectively.

```
>> c2=curvefitp(x,y,2)             % 2nd order fit
```

This plots a second order polynomial to the data. The coefficients of the best fit polynomial is now in c2. The fitted line is,

```
 y = c2(1)x^2 + c2(2)x +c2(3)
```

CURVEFITNL fits an arbitrary function to your data. It starts from an initial guess and then varies the coefficients in an attempt to find the best fit.

For example, to fit an function,

```
y=p(1)exp(p(2)x)
```

to your data, you would,

```
>> x=1:10;
 >> y=[11 12 13 15 16 18 20 22 25 27];
 >> ce=curvefitnl(x,y,'p(1)*exp(p(2)*x)',[1 1])
```

Here, the [1 1] is an initial guess as to what the final coefficients will be. If the initial guess is too far away from the 'true' coefficients, unpredictable things can happen. (the function can exit with an error message, or the coefficients calculated can be wrong) If this happens, choose a new initial guess. Note that you MUST use 'p' and 'x' in the function string.

For more help on these functions, look at the help message
for each of them, for example,

```
>> help curvefitp
```