Ph3 Mathematica Homework: Week 4

Eric D. Black
California Institute of Technology
v2.1

1 Comparing theory and data

Exercise 1: Import the data set you were working on last week (`http://pmaweb.caltech.edu/~phy003/labs/Data1.txt`). Deal with any headers or formatting you need to, just as you did last week, and plot it using `ListPlot`.

Exercise 2: Estimate the slope and y-intercept of a line that will fit the data, and plot that line along with the data using the `Show` command.

2 Least-squares fitting

Exercise 3: Have Mathematica find the best coefficients to fit a straight line to your data using the following command, and compare the results with your estimates for the coefficients.

```
lsq = Fit[data, {1, x}, x]
```

Here, as usual, we are storing the output of the `Fit` command in the variable `lsq`. (The first character is a lower-case L, not the number one. While you can include numbers in your variable names, it's generally not a good idea to *start* your variable name with a number.) The arguments are the data, the model you want to fit, and the independent variable. The model is a shorthand notation for the form of the function we are fitting to the data, in this case a first-order polynomial in the variable x. To fit to a quadratic you would use `{1, x, x^2}`, etc.

Exercise 4: Plot the fit just as you would any other function, and compare with your rough-estimate plot from Exercise 2.

```
Show[ListPlot[data], Plot[lsq, {x, 0, 10}]]
```

3 LinearModelFit

The `Fit` command is easy to use and easy to remember, and it uses the familiar least-squares algorithm for determining your parameters. Its output is somewhat limited, however, and if you want uncertainties on those parameters, R values, etc. it is not so straightforward to get them. A newer command is `LinearModelFit`, and it works almost the same as `Fit` but gives a much richer output. The syntax for calculating the fit model is similar to `Fit`.

```
lmf = LinearModelFit[data, {1, x}, x]
```

Getting a table of parameter values with `LinearModelFit` is easy.

```
lmf["ParameterTable"]
```

Displaying the function is done in a similar manner.

```
lmf["BestFit"]
```

You can even extract the residuals with a simple command.

```
lmf["FitResiduals"]
```

The last command returns a simple list of the residuals that you can plot, *i.e.*

```
ListPlot[lmf["FitResiduals"]]
```

This simple example does not include error bars, but those are easy to add once you have this list, using the list-manipulation methods we covered previously.

The main difference between `LinearModelFit` and `Fit` is, you have to explicitly show the dependence on `x` when plotting the output of `LinearModelFit`.

```
Show[ListPlot[data], Plot[lmf[x], {x, 0, 10}]]
```

4 Fitting nonlinear functions to data

At least once in this class you will have to fit a nonlinear function to a set of data. The command for that in Mathematica is

```
fit = NonlinearModelFit[data, model, {parameters}, variable]
```

Here the variable `fit` is used to hold the model Mathematica generates as a result of the fit. You, of course, could call it anything you want. Of the arguments, `data` is a list of your data, `model` is the function you want to fit, `{parameters}` is a list (in curly brackets) of the parameters to vary, and `variable` is the independent variable (usually `x`).

The parameters list can be a simple list of the form `{a, c, w}`, but you will often find it useful to give Mathematica some initial guesses to start from.

Exercise 5: Import the data at `http://pmaweb.caltech.edu/~phy003/labs/LorentzianData.txt`, and fit a Lorentzian function to it. A Lorentzian function has the form

$$f(x) = \frac{a}{1 + \left(\frac{x-c}{w}\right)^2}$$

The parameters a, c, and w correspond to the amplitude, center, and width of the Lorentzian peak, respectively. For example, if by examining your data you think your amplitude is going to come out somewhere around 2.5, you would replace `a` in your parameter list with `{a, 2.5}`, *e.g.*

```
fit = NonlinearModelFit[data, a/(1+((x-c)/w)^2),
     {{a, 2.5}, c, w}, x]
```

Giving initial guesses for the other parameters works the same way.

```
fit = NonlinearModelFit[data, a/(1+((x-c)/w)^2),
     {{a, 2.5}, {c, 1}, {w, 1}}, x]
```

Initial guesses aren't always necessary. If you leave them out Mathematica will sometimes be able to do the fit anyway. However, sometimes it won't, and it is always good practice to look at your data and estimate what you think the fit is going to be beforehand.

Note: It is considered good programming practice to clear your parameter values before using them in a fit, or doing anything else with them for that matter. You can do this with the `Clear` command, using the following syntax.

```
Clear[a, c, w]
fit = NonlinearModelFit[data, a/(1+((x-c)/w)^2),
     {{a, 2.5}, {c, 1}, {w, 1}}, x]
```

Note that `Clear` has to be done *before* any attempt at using these parameters in a fit, and that's why I've listed the `NonlinearModelFit` command along with it. Clearing your parameters of any previous values or definitions isn't always necessary, but it never hurts. If your fit returns an error message, this is one of the first things you can try to resolve it.

Exercise 6: Plot your data and fit using

```
Show[ListPlot[data], Plot[fit[x], {x, -2, 2}]]
```

Exercise 7: Display a table of the results of the fit, along with uncertainties in the parameters, using the command

```
fit["ParameterTable"]
```

Exercise 8: Display the fitted function using

```
fit["BestFit"]
```

5 Residuals

Residuals measure the differences between your data points and your theory.

$$r_i = y_i - f(x_i)$$

Exercise 9: Calculate the residuals $\{x_i, r_i\}$ for both the linear and Lorentzian data and their associated fits, and plot both on appropriate scales with error bars. Use uncertainties of ± 0.3 on each value of r_i, just as you did for the y_i values in the linear data earlier.

If your fit is good, roughly half of your residual points should be above the x-axis, and half should be below. If the random error in your data is gaussian and your error bars are accurate, about two-thirds of them should enclose the x-axis, and the remaining third of your residuals will be more than one sigma away from zero. (Do you recall from Taylor, Chapter 5 why this is to be expected?) These are two simple but important tests that can tell you at a glance if your fit is off, your error bars are wrong, or your data is not gaussian. Any one of these should send up alarm bells, and if you see them you should try to find out what is going on.

Exercise 10: Do your residual plots satisfy these criteria? Do the error bars appear to have been chosen correctly?

6 Reference

6.1 Fitting commands

1. `Fit[data, model, variable]` - Least-squares fit of `model` to `data`. The `Fit` function can fit to a polynomial of any order by changing the second argument. For example, you could fit to a quadratic by typing

   ```
   Fit[data, {1, x, x^2}, x]
   ```

2. `LinearModelFit[data, model, variable]` - Similar to above, except with more options.

3. `NonlinearModelFit[data, model, variable]` - Fit a nonlinear model to a set of data. For more information see `http://reference.wolfram.com/language/ref/NonlinearModelFit.html`