# Curve-fitting and Scripted Plotting Tutorial

In this assignment, you will need to utilize matlab's statistics toolbox. We will be using the lsqcurvefit function to find the fit coefficients of a sample function applied to a random set of data. Start by copying and running the following script in the command prompt:

```
clear
close all
clc
format long

x = 1:.01:5;
y = 8.45*x + 2.654;
y = y/max(y);

count = 30;
iset = randi([1,length(x)],1,count);
xset = x(iset) + randi([-1,1],1,count).*rand(1,count).*x(iset)*.1;
yset = y(iset) + randi([-1 1],1,count).*rand(1,count).*y(iset)*.1;

save('file.mat','xset','yset')
plot(x,y,xset,yset,'*')
```

This code will generate a random set of data distributed around the function $y = 8.45x + 2.654$ (scaled by the maximum value in the range), saves it to a .mat file, and plots it so that you will be able to see it.

We will try to fit two different functions to these random data points:

$$y = C_1 x + C_2$$

$$y = C_1 x e^{C_2 x} + C_3$$

The lsqcurvefit function allows you to minimize the difference between scattered data and the function you're fitting in a least-squares sense. Let's set up the syntax. First we need to define the function we are trying to curve fit. We'll try the linear fit first. Start a new .m file.

```
clear
close all
clc
format long

load file

c01 = [1 1];                                    % Initial guess for coefficients
F1 = @(c,xdata)c(1)*xdata + c(2);               % Function definition
```

In the above lines, the function F1 is a function of c and xdata, where c is a vector of coefficients (i.e., [$C_1$ $C_2$]). c01 is going to be our initial guess for these values; their value is not particularly meaningful. Wherever you place the term xdata then represents an 'x' in the equation you're fitting. To solve for the coefficients, put try the following line:

```
c11 = lsqcurvefit(F1,c01,xset,yset);
```

This invokes the lsqcurvefit function, applies it to F1 with initial guess for the coefficients c01, then fit to the scattered data in xset and yset. Running this will automatically calculate the coefficients $C_1$ and $C_2$. Let's display this so you can see the fit:

```
x = 0:0.01:5;
plot(xset,yset,'*',x,F1(c11,x));
```

That's it! We'll follow a similar procedure for the other function we're fitting:

```
c02 = [1 1 1];
F2 = @(c,xdata)c(1)*xdata.*exp(c(2)*xdata) + c(3);
c12 = lsqcurvefit(F2,c02,xset,yset);
```

```
close all
plot(xset,yset,'*',x,F1(c11,x),x,F2(c12,x))
```

You can see from the plot that both functions are reasonable approximations of the scattered data. As a first-order quantification of the fit, we can calculate the $R^2$ value for both. This is a function that compares the fit of the current function to a simple mean of all the data points. It is calculated as follows:

$$R^2 = 1 - \frac{SSe}{SSt}$$
$$SSt = \sum (y_i - \bar{y})^2$$
$$SSe = \sum \left(y_i - F(x_i)\right)^2$$

Here, it can be see that the SSt term is how far from each individual data point is from the mean, while the SSe term is the distance of each data point to our fitted function's value at the same x location. Let's compute this for both of our test functions:

```
SSt = sum((yset - mean(yset)).^2);
SSe1 = sum((yset - F1(c11,xset)).^2);
R21 = 1 - SSe1/SSt

SSe2 = sum((yset - F2(c12,xset)).^2);
R22 = 1 - SSe2/SSt
```

Here, R21 corresponds to our linear function and R22 to the compound exponential. Finally, let's generate a plot that has the correct formatting without having to do a bunch of tweaks in the figure editor. Scripting a plot in this manner is particularly useful because it allows you to save a plotting style, while only needing to change the data. To do this, we'll have to define each object in the plot window and set its style. We'll start with the figure window itself:

```
figure1 = figure('Color','w');
```

Next, we'll define the axes and set their limits, and aspect ratio:

```
axes1 = axes('parent',figure1,'PlotBoxAspectRatio',[1 1 1],'FontSize',15,'FontName','Goudy Old Style');
xlim(axes1,[0 5]);
ylim(axes1,[0 1]);
hold(axes1,'all');
```

The first line defines the axes with the parent handle, which allows you to apply all subsequent articles to them. At the same time, the aspect ratio of the axes is set to 1:1 (forcing them to be a square, regardless of the data aspect ratio) and the font size and style. Here we'll use 15 pt Goudy Old Style. The next few lines simply set the axis limits and turns on 'hold' for our axis object.

We'll start by plotting the scattered data (this should all go on one line):

```
plot(xset,yset,'parent',axes1,'MarkerSize',8,'Marker','x','LineWidth',1,'LineStyle','none','DisplayName','Data','Color','k');
```

Next we'll plot our two fitted functions:

```
plot1 = plot(x,[F1(c11,x);F2(c12,x)],'parent',axes1,'LineWidth',1);
set(plot1(1),'LineStyle',':','Color','k','DisplayName','Curve Fit 1');
set(plot1(2),'LineStyle','--','Color','k','DisplayName','Curve Fit 2');
```

The first line plots our two fitted functions against x; note that the fits F1 and F2 must be in a concatenated form. Note that this assigns the plots to the 'parent' handle and sets the line width to 1. The next two lines take the two separate functions (plot1(1) and plot1(2)) and changes their style. Note that two different types of dashed lines have been applied to these functions. Dashed line indicators include ':', '--', and '-.'. Try the different types out to see the patterns they use. Finally, we need to put labels and a legend on the plot:

```
xlabel('$\displaystyle\sqrt{\frac{d\omega}{\Omega}}$','Interpreter','latex','FontSize',16);
ylabel('$\displaystyle\frac{y}{\max(y)}\;\;\;\;\;\;\;\;\;\;$','Interpreter','latex','FontSize',16,'rotation',0);
title('Curve Fitting','Interpreter','latex','FontSize',16);
legend('location','SouthEast','Orientation','horizontal');legend boxoff
```

That should do it. The first three lines set the x-axis, y-axis, and title labels. Note here that we're no longer using the default matlab text interpreter ('Tex'), opting instead to use the 'Latex' interpreter. This is the standard font and style used in most journals. Latex is a scripted text language, allowing you to put clean looking math symbols and text on the plot axes. Some basic functions you may want to use:

```
\sqrt{ }            % Puts a square root symbol over the quantity in the brackets

\frac{}{}           % Sets up a fraction with the quantities in the {} as the numerator (first) and denominator (second)

\;                  % Large space
```

Greek symbols work the same way that they do with the standard 'Tex' interpreter (\lowercasegreek for a lowercase symbol and \Uppercasegreek for an uppercase version).

Note also in the y-axis label that the text has been rotated back to its initial orientation of 0 degrees. This is why all the spaces are necessary to offset the label from the axis. Try it without the spaces to see the problem.

The legend is pretty standard, with the only hiccup being the orientation (horizontal). That's it. Good luck.