# Linear Regression and Gnuplot

## Introduction

"Least-squares" regression is a common data analysis technique that is used to determine whether a particular model explains some experimental data. The model is represented by some function $y = f(x)$, where $x$ and $y$ are the two bits of data measured in the experiment. The value $x$ is called the *independent* variable, since it is the one we can set arbitrarily. Similarly, $y$ is called the dependent variable, since it depends on $x$ and the model (function) $f$.

Hopefully you remember that a line has the parametric form

$$\begin{aligned} y &= f(x) \\ &= mx + b \end{aligned}$$

Thus, our function $f$ has two parameters, $m$, the slope of the line, and $b$, the line's y-intercept. The goal of regression is twofold. First, it is to find the "best" values of the parameters of the model $f$ (in the linear case case, $m$ and $b$). Second, we wish to determine whether this model gives a reasonable explanation for the data.

The first goal is met by finding the values of the parameters that give a curve that fits the data points best. Technically, our best line will minimize the average squared distance from the line to the points. We won't go in to the theory of why this works (you should take Math 209, or even better, Math 335!), but next we'll talk about how to easily find such a line (and visualize it) using Gnuplot.

## Gnuplot: Plotting and Regression

Gnuplot can be used interacdtively, or a series of commands can be collected into a file and run in batch mode. We'll build up our results incrementally and detail individual commands, but they may be easily be collected together. You can start Gnuplot by issuing the command `gnuplot` in a terminal window.

### Plotting Data

The data being used should be stored in a file where each line contains an $(x, y)$ pair in the simple (space separated) format

```
xVal yVal
```

To plot these as points on the axis, issue the command

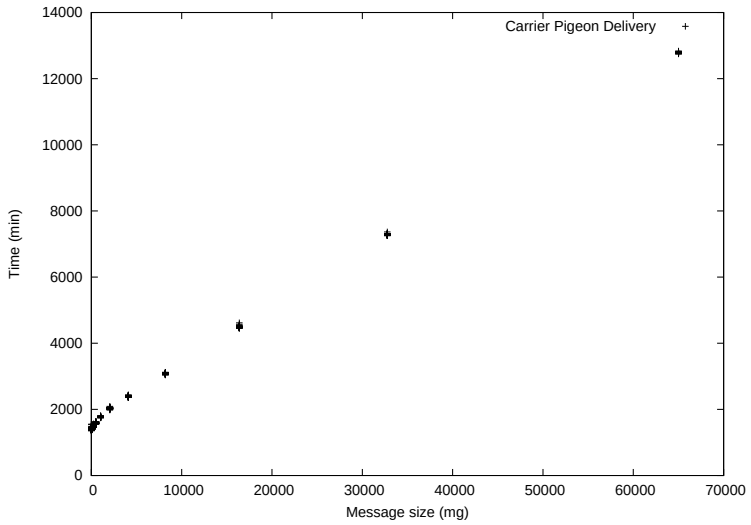
```
gnuplot> plot 'filename' using 1:2 title 'My Experiment' with points
```

where *filename* is the file containing the data you wish to plot.[1] The `using 1:2` part directs Gnuplot to use the first and second columns of the data, while `with points` part directs Gnuplot to plot these as disconnected points. There are other options, as we shall soon see.

You should of course, give the plot a meaningful title (rather than "My Experiment.") You would do well to label the axes so that others (or you, should you look back at this next week) know what is being plotted. For example, if your values represent the time to send a message via carrier pigeon with respect to the message size, you could do this with the commands

```
gnuplot> set xlabel "Message size (mg)"
gnuplot> set ylabel "Time (hours)"
```

but you must issue the label commands *before* the plot command. Don't forget the very useful units! This should give you something that looks like the following (on screen)




Now we've got some data on some axes, and we know (vaguely) how to access it with the plot command. So far so good. What about this regression bit?

**Regression**

The first thing we need to do is define the function we want to fit to the data. In our example, we're talking about a line, so we define our simple function (model) as a line for Gnuplot.

```
gnuplot> f(x) = m*x + b
```

Now, Gnuplot has the regression thing down. It knows what to do simply when you tell it:

```
gnuplot> fit f(x) 'filename' using 1:2 via m,b
```

What you should see back is the results of a few iterations of the process for fitting the line. The last bit should look something like:

[1] We show commands with the Gnuplot command prompt "`gnuplot>`". You should not type this, and if these were put into a script, you of course would not copy the "`gnuplot>`".

```
After 7 iterations the fit converged.
final sum of squares of residuals : 5.33793e+06
rel. change during last iteration : -1.30122e-12

degrees of freedom (ndf) : 448
rms of residuals      (stdfit) = sqrt(WSSR/ndf)      : 109.156
variance of residuals (reduced chisquare) = WSSR/ndf : 11915

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
m               = 0.174844         +/- 0.0002964    (0.1695%)
b               = 1506.34          +/- 5.756        (0.3821%)

correlation matrix of the fit parameters:
               m      b
m               1.000
b              -0.448  1.000
```

There's a lot of info there about the nature of the data, and how well Gnuplot was able to fit it, but the part you probably care most about is "What are the parameters?" This is probably fairly straight forward to read off from the "Final set of parameters" section. In this data, the slope is 0.1748 and the $y$-intercept is about 1500.

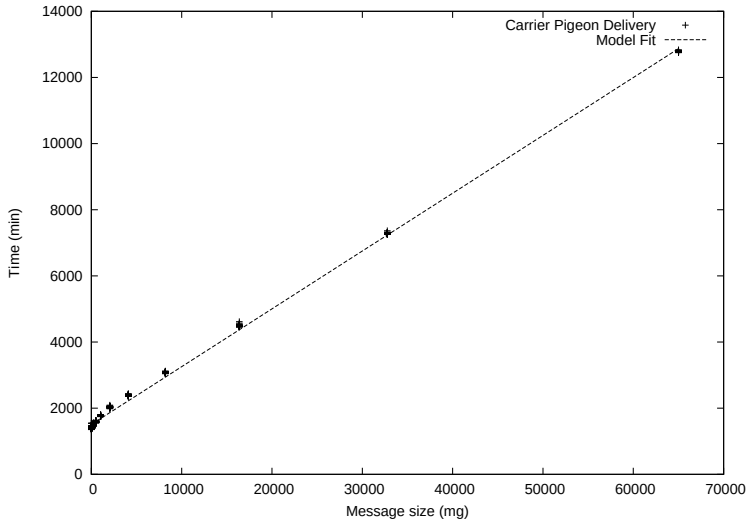
The next question is, does it look like this line fits our data? We can plot the line (Gnuplot stores the parameters $m$ and $b$ after the fitting process) with the command

```
gnuplot> plot f(x) title 'Line Fit'
```

This will in fact plot the line with the appropriate values of $m$ and $b$ over an arbitrary domain (-10,10). However, it's probably more interesting to see the line plotted *with* the original data. We can do this if we issue both directives in the same command:

```
gnuplot> plot 'filename' using 1:2 title 'Carrier Pigeon Delivery'
         with points, f(x) title 'Model Fit'
```

This should give you something like the following image on screen.



Next we need to describe how to save these images to a file.

### Gnuplot-ting to a file

Because it was designed in the days before rampant GUI usage, Gnuplot may seem a bit archaic with respect to its file-saving methods. It is more analagous to shell-based pipes and output redirection than any modern software. Saving the plots to a file involves three basic steps.

1. Direct the output to an appropriate "terminal." This acts as a filter that puts the data in the correct form for whatever format you want the output to be in. It's sort of like using a command-line pipe of the format "`gnuplot | gp2img`" where `gnuplot` generates some plot structure and `gp2img` translates the output into an image.

2. Next, we need to say where the output should finally be stored. This is just like using the shell to direct the standard output of a program into a file, such as "`gp2img > file.dat`"

3. Finally, we actually need to *issue* the plot commands. This would be the input to gnuplot that tells it what to generate (that is eventually passed to a filter and written to a file). This could be akin to directing a file to standard input, such as "`gnuplot < commands`". In actuality the plot commands may either be given in interactive mode (once the terminal and output are set above), or via a command file.

Step 3 was detailed in the previous sections. The plot commands may be put into a file, or given at the interactive prompt. Steps 1 and 2 are very easy. The syntax may vary with the output format desired, but PNG is a compact, widely supported format, so we'll use that as an example:

```
gnuplot> set terminal png
gnuplot> set output 'file.png'
```

These easily accomplish steps 1 and 2 above. Any subsequent plot directives will be stored in *`file.png`* until a new output file is specified with another "`set output`" command. You can return to plotting to the screen by issuing another "redirecting" set terminal command:

```
gnuplot> set terminal x11
```

Gnuplot is a robust and highly versatile piece of software. There are many more options than the very brief portrait given here. To learn more, you should view some of the plethora of tutorials on the web.