# CS4115 Week06 Lab Exercise

**Lab Objective:** The objective of this week's lab is to complete our experimental evaluation of the chunks of code given in the book. At the end of the last lab we asked you to estimate the big-oh running time of your pieces of code by comparing the running time plots against $n^2$, $n^3$, etc. Now we will look at a more accurate way of estimating the running time.

By the end of this lab you should have

❶ implemented as programs the chunks of code from Q. 2.7 of Chapter 2 of *Weiss*;

❷ recorded in a spreadsheet, for each program, its average running time;

❸ plotted these running times in `gnuplot`; and

❹ computes a function that accurately describes the observed running times

❺ for each case save the gnuplot "program" and the resulting plot for handing in

## In Detail

Imagine you have recorded a set of points $(x_i, y_i)$ and you now want to know what is the underlying relationship between the corresponding $x_i$ and $y_i$. You might suspect that they are acting according to a polynomial $y = f(x) = C \cdot x^m$. That is, you suspect that for every pair of values

$$y_i = f(x_i) = C \cdot (x_i)^m$$

What you now want to find is the values $C$ and $m$.

When taking readings as you have done there is always some experimental error involved so even if you had an eternity it would be very unlikely to be able to hit on the right constants $C$ and $m$. What we will do is to try and find the values of $C$ and $m$ that fit our *best* observations. This is the subject of *Least Squares* fit and here's what Wikipedia has to say about it.

## Background

If we have a function $y = C \cdot x^m$ then provided that $m > 1$, plotting this will give a curved behaviour with the steepness of the curve dictated by how large $m$ is.

Now consider what happens if we look at the logarithm of this. (Any base logarithm will do as long as we stick with it throughout.)

$$\begin{aligned} \log y &= \log C \cdot x^m \\ &= \log C + \log x^m \\ &= \log C + m \cdot \log x \end{aligned} \tag{1}$$

To make it a bit simpler we will substitute $u$ for $\log x$ and $v$ for $\log y$. So in terms of $u$ and $v$ equation 1 can be rewritten as

$$v = \log C + m \cdot u$$

This is suspiciously like the old $y = mx + b$ equation of a straight line that you learnt in school where $b$ would take the place of $\log C$ as well. The important word here is *straight*: straight lines are much easier to work with than curves.

So in summary by taking the log of both sides we have gone from $y$ being dependant on a polynomial of $x$ to $v$ being dependant on a *linear* multiple of $u$; there's a constant term as well, but that won't be too difficult to deal with. What we now want to find is the best *straight line* that fits the pairs of points $(u, v)$. Follow this link for a nice picture and example of that is going on. This page gives the formula for how the straight line can be calculated. A less precise alternative would be for you to plot the points (the log of them, actually) and then eyeball the best straight line. You could then figure out the parameters $m$ and $b$ and using the substitution $b = \log C$ you could determine out the underlying curve $y = C \cdot x^m$.

## A Better Way

The trouble with the above method is that functions aren't always as simple as

$$y = f(x) = C \cdot x^m$$

. There are usually lower order terms as well, that makes it look more like

$$y = f(x) = C_0 + C_1 x^1 + C_2 x^2 + \cdots + C_m x^m$$

Sadly, the trick above of taking logarithms won't work in this case. But happily the theory of linear least squares *will* allow us to keep track of several separate terms, so now let's spend some time developing the theory of linear least squares for generalised polynomials.

Ha ha, fooled you! We're not going to do that at all. We're just going to be happy that somebody else did it for us. And what's more we're going to use `gnuplot` to figure out the coefficients of each of the terms.

## Using the `gnuplot` fit command

❹

After we've plotted our data using `gnuplot` with something like

`plot [10:1000] 'q271.dat'` we might decide that the curve that best fits the data is a polynomial whose highest power is 3. That is, we think that

$$T(n) = a + b \cdot n + c \cdot n^2 + d \cdot n^3$$

is the best potential curve. In `gnuplot`-ese this translates to

$$f(x) = a + b \cdot x + c \cdot x^2 + d \cdot x^3$$

We now give the following commands:

```
gnuplot> f(x)=a + b*x + c*x**2 + d*x**3
gnuplot> FIT_LIMIT=1e-6
gnuplot> a=1.0;b=1.0;c=1.0;d=1.0
gnuplot> fit f(x) 'q271.dat' via a,b,c,d
```

The meanings are 1) our expected function template is a polynomial with highest power 3; 2) our required level of accuracy is $10^{-6}$; 3) to start off the iterative process, let all coefficients be 1.0; 4) now, fit the data of the file 'q271.dat' using the function template $f(x)$ and coefficients $a, \ldots, d$. In addition to this information being reported to the screen it is also written to a file 'fit.log'. You should rename this file to be, say, `q271.log`.

You can now judge the quality of the resulting estimate of your running time by plotting both the measured data and the function estimate. In `gnuplot` give the command:

```
gnuplot> plot [10:10000] 'q271.dat' with points, 0.1245 + 3.45678*x with lines
```

where the two supplied numbers are the values you had gnuplot determine for `a` and `b`. (If you had chosen to use a quadratic then a `c` term should be present, etc.)

You should now save the resulting picture as a `.png` file. This requires setting the display type to be `png` and then naming the output file. These two things can be done with the gnuplot commands

```
gnuplot> set term png
gnuplot> set output 'q271.png'
gnuplot> plot [10:10000]  'q271.dat' with points, a + b*x + c*x**2 +d*x**3 with lines
```

All of these commands for a particular code fragment can be saved in a single gnuplot "program" (say, `q271.gp`) and then run with the command from the shell prompt:

```
gnuplot q271.gp
```

Figure 1 illustrates what a typical `.gp` file might look like although...

- the range of input values you were able to run it for might require different
- if you believe that a particular code snippet does not require a quadratic or cubic term then these could be left out

This is the end of this series of labs. During this week's lab the lab instructor will check to make sure that you have been working consistently on this series of lab exercises. By the middle of next week you will be expected to hand up graphical proof that you have completed the lab. This proof will comprise files relating to each of the 6 code snippets.

For each of the 6 cases you should have a set of files with the prefix `q27x`, where `x` is one of the numbers $1, \ldots, 6$ that signifies the code snippet. For each snippet you should have in your directory the following files. As I have been doing I will use the file `q271` as an example

1. the *average* runtimes you generated in your spreadsheet, `q271.dat`

```
f(x)=a + b*x + c*x**2 + d*x**3
FIT_LIMIT=1e-6
a=1.0;b=1.0;c=1.0;d=1.0
fit f(x) 'q271.dat' via a,b,c,d
set term png
set output 'q271.png'
plot [10:10000] 'q271.dat' with points, a + b*x + c*x**2 +d*x**3 with lines
```

Figure 1: A gnuplot command file for the `q271` case.

2. the command file that you used in `gnuplot`, `q271.gp`

3. the trace of the `gnuplot` fit command, `q271.log`[1]

4. the resulting png graphic, `q271.png`

During Week07 I will want you to submit all of these files using, wait for it, `handin`. The combined lab counts for 12% of your final mark.

[1] Don't forget to rename this file from the default `fit.log` to `q271.log`.