CSC 160

Professor Chris Brown

LED Labs

04/12/2011

# Data Acquisition: The LED Labs (MATLAB)


## Abstract

Throughout the three LED labs, as both engineering students and computer scientists, we will set up our own LED circuit, interact with it by controlling it using MATLAB on computer. By using an USB 6009 with MATLAB programming language (associated with DAQ Toolbox), we can output voltages to the circuit in either digital or analog signal and get some interesting, worth analyzing results. The purpose of the LED labs is not only creating the basic image of the characteristics of LED and other circuit component, but most importantly also testing data acquisition in MATLAB.


## Introduction

### Lab 1:

This part of the lab is basically a "game" of the translation between binary digits and hex number. Each player will first receive a random number between 0 and 15, which is displayed in binary manner (e.g. 3 is 0011 in binary digits). The user will then be asked to input the corresponding hex number (input an integer from 0 to 9 or a letter {a, b, c, d, e, f} for numbers from 10 to 15). As a hint displayed to the user, the 4 LEDs in the circuit represent the four digits of the binary number. For each of the four lights, if it is on, it means the corresponding digit is 1; if the light is off, it means the corresponding digit is 0. Thus, the four LED lights in the circuit can represent a four-digit binary number. However, the user might not notice the relationship between the lights and digits and therefore, the whole "game" will show a learning process of the user. The user will have unlimited try for the input, and each time the program will notify the user whether the answer is correct or not. If the user input a negative number, the game will automatically terminate. The number of inputs to get the correct answer and time spent thinking for each trial are recorded for further analysis (e.g. plot the "learning curve").

### Lab 2:

The goal of Lab 2 is to modulate the brightness of the LEDs by adjusting the analog output voltage to the circuit. Instead of creating a digital output like in Lab 1, we will create an analog output object, and produces an analog signal representing a sine wave that varies from 0 to 5 volts. The other output voltage is dependent on a cosine wave in the same range of voltage as the first one. The process is repeated and we expect to see the two LEDs light alternatively with varying brightness (10 times oscillations in our script). Each of the two lights should be brightest and dimmest at exactly opposite time because of the characteristic of their analog input (sine and

cosine wave function).

### Lab 3a:

In Lab 3a, we will leave only one pair of LED and resistor in the circuit and has an output voltage in the range of -1 to 5V. The voltage drops across the circuit, LED and the resistor are measured by using differential input channels. The results are then stored and plotted for analysis. As a hypothesis, we would expect the voltage drop across the circuit is the sum of the voltage drop across the LED and across the resistor.

### Lab 3b:

The last part of the LED Lab is going to investigate the "pause" function in MATLAB. We will keep the same set up as in Lab 3a but adjust the pause time between two voltage outputs to a longer time than that in lab3a. The results are also plotted and we expect stepped or jagged trend lines.

## Methods

### Lab 1:

### Circuit set up:

We set up four pairs of LEDs and resistors in parallel by using a breadboard. The breadboard is supplied with four digital voltage inputs (each pair of LED and resistor get one input) by wiring with four digital output lines on USB 6009. A wire is then used to connect the main board across the red column to the blue column at the right end of the breadboard, and the blue column is connected to the ground terminal on USB 6009 so that a full circuit is established.

### Programming in MATLAB:

The main script for Lab 1 basically consists of three parts:

1. Set up the "game";
2. Record the results of the game while running;
3. Plot the data from the experiment.

First of all, we need to clear all the possible internal state of the device. A device object is created and used to associate the digital input and output subsystem with MATLAB command by using the built-in function 'digitalio'. Four digital output lines are then created and labeled with a specific number by using the function 'addline'. The specific number of the line is corresponding to the specific number of the digital terminal on USB 6009. For example, if we call line 0 on the computer, the line terminal P00 is on and supplies a digital output voltage.

For setting up the "game", we incorporate two 'while loop' along with if statements in the core of the script. For the ease of understanding, three possible conditions of the game will be introduced first. As long as the user inputs a positive number or a letter from "a" to "f", the loop will keep running. If the user's input answer is correct, the game will automatically terminate the current trial and enter the next round. However, if the answer is incorrect, the user will be asked to enter an answer again until the answer is correct and then move on to the next round. The third condition is when the user inputs a negative number, the whole game is terminated (no more

trials).

Before going to the outer loop in the script, there are two variables needed to be set up first (both of the initial values are zero), called “value” and “c_counter” in our case. As long as the “value” variable is greater than or equal to zero, we will enter the outer while loop, which means staying in the “game” and entering the next trial when one trial is over. The three possible conditions discussed above are incorporated into the loop by using if statements. In the outer ‘while’ loop, a random number between 0 and 15 is generated in the beginning of each trial. We use the built-in function ‘putvalue’ to call the assigned lines to turn on the specific lights. The user is then asked to input a hex number which corresponds to the binary number indicated by the four LED lights.

We set another variable “f_counter” at the end of the inner loop asking for user’s input, which will be used to record the total number of answers inputted by the user in one trial (when the answer inputted is incorrect). After setting up the “f_counter” variable, the “game” will start counting the time (for each trial) by calling the ‘tic’ (MATLAB built-in function).

Inside the outer ‘while’ loop, the first if statement corresponds to the condition when the user’s input of hex number is a correct translation of the binary numbers shown by the LED lights. The binary number shown by LED lights is originated from the randomly generated decimal number in the beginning. The ‘hex2dec’ built in function is used to convert the user’s input value (a hex number) to decimal number and then compared to the decimal number (0 to 15) generated in the beginning. In this case, if those two values are the same and we will notify the user that the answer is correct and add 1 to the variable “c_counter” which means the first trial is completed and also record the total number of tries in the current trial by adding one to “f_counter”. The time spent in each single trial to get the correct answer is recorded by calling the ‘toc’ function.

The second ‘ifelse’ statement is to check if the user enters a negative number and if yes, it will first be converted to decimal number and then be set equal to the variable “value”. As mentioned above, when “value” is detected as a negative number in the outer loop, we will not re-run the outer ‘while’ loop anymore and the whole game is terminated.

The third else statement is used when the answer the user entered is incorrect. An inner while loop is used to prompt the user to enter a new answer each time when the input answer does not match the correct answer (with unlimited attempts). After doing the conversion using ‘hex2dec’ function, if the two values are not equal, we will first add 1 to the variable “f_counter” in order to record the total number of tries in one trial and notify the user that the input message in incorrect. After that, the user can input a new hex number. After several tries, the user might get the correct answer, and if answer is ultimately correct, we will first add one to the “c_counter” and then record the total time and the total number of tries (the total of “f_counter” plus one) in this trial.

After running the whole game, the total number of tries, total time in each single trial and total number of trials are recorded and plotted for further analysis. (Please refer to the “Result” in next section)

## Lab 2:

### Circuit set up:

For Lab 2, we will only use two pair of resistors and LEDs. The set up is basically the same as the first lab except the breadboard is supplied with two analog output voltages by wiring with two analog output channels on 6009.

### Programming in MATLAB:

The basic set up is similar as in the first lab. A device object using analog output subsystem (instead of digital output in Lab 1) is called by using the built-in function 'analogoutput'. Two analog output channels are then created and corresponded with a specific number by using the function 'addchannel'. The number is corresponding to the number of the digital terminal on USB 6009. For example, if we call line 0 on the computer, the line terminal AO0 will be on and supply an analog output voltage.

To incorporate the sine and cosine function, we use a simple 'for' loop that loops 1000 times so that we can observe 10 oscillations (with our script). The first analog output voltage corresponding to the first channel is set with a voltage called "vout1" which is equal to $2.5 + 2.5 * \left(\sin\left(\mathrm{k} * 2 * \frac{\mathrm{pi}}{100}\right)\right)$. The second voltage "vout2" is set to equal to $2.5 + 2.5 * \left(\cos\left(\mathrm{k} * 2 * \frac{\mathrm{pi}}{100}\right)\right)$. Thus, the range of the voltage output will be from 0 to 5V, since the maximum values for both sine and cosine functions are 1 and both of their minimum values are -1. As k goes from 1 to 1000, both functions will be equal to 1 and -1 at opposite time. Since $2\pi$ is one period (one full oscillation) for both of sine and cosine functions, so we will have 10 full oscillations in total. A built-in function 'putsample' is used to send the value of these two analog output voltages through channels to the circuit.

As required, there will be a 0.02 seconds pause between each of the two voltages outputs.

## Lab 3a & Lab 3b:

### Circuit set up:

For both Lab 3a and Lab 3b, we will only use one pair of resistor and LED, which are connected in series. Similar to Lab 2, the circuit is supplied with one analog output voltage by wiring with one analog output channel (AO0) on 6009. In this lab, we will use three analog inputs channels AI0, AI1 and AI2 (with both positive and negative ends) as differential inputs each connected across LED, resistor and LED + resistor (whole circuit) respectively in order to measure the voltage drop of the three different parts.

### Programming in MATLAB:

First of all, the maximum number of input channels and the expected sample size for each channel are set to be 3 and 300. A device object using analog output subsystem is then called by using the built-in function 'analogoutput'. In Lab 3, only one analog output channel is created and associated with AO0 by using the function 'addchannel'. After that, a device object using analog input subsystem is then called by using the built-in function 'analoginput'. Three analog input channels are established and labeled with a specific number (0, 1 and 2 in this case) by

using the function ‘addchannel’ again.

For Lab 3a, we then use a ‘for’ loop iterated from 1 to 300 to set the output voltage going linearly from -1 to 5V. The built-in function ‘putsample’ is used to send the value of these two analog output voltages through channels to the circuit for each loop and a pause time of 0.01 second is added between each two output voltages. The voltages of each three parts of the circuit are recorded each time using the ‘getsample’ (MATLAB built-in function). After recording the data, another pause time of 0.01 second is added to avoid the possible race between putting and getting the samples.

The code for Lab 3b is slightly different from 3a, where the difference is we prolong the pause time from 0.01 second to 0.04 seconds inside the ‘for’ loop. In Lab 3b, we will only loop from 1 to 25 (let voltages going from -1 to 5V in 25 steps). The sampling rate can be adjusted using the built-in function ‘setverify’. We run the experiment with sampling rates both in 100 and 1000Hz and get the result of voltage drops across different parts of in the circuit.

## Results and Discussion

### Lab 1:

(Tested with the “Broadway.m” before the formal LAB 1)

We know that our circuit and codes all function correctly because the LEDs turn on normally and each of them corresponds to a specific digit of binary number correctly. The result of the binary to hex translation game is stored and plotted.

(Note: continued on next page in order to better fit the figures)

Lab 1: (raw data plotting)

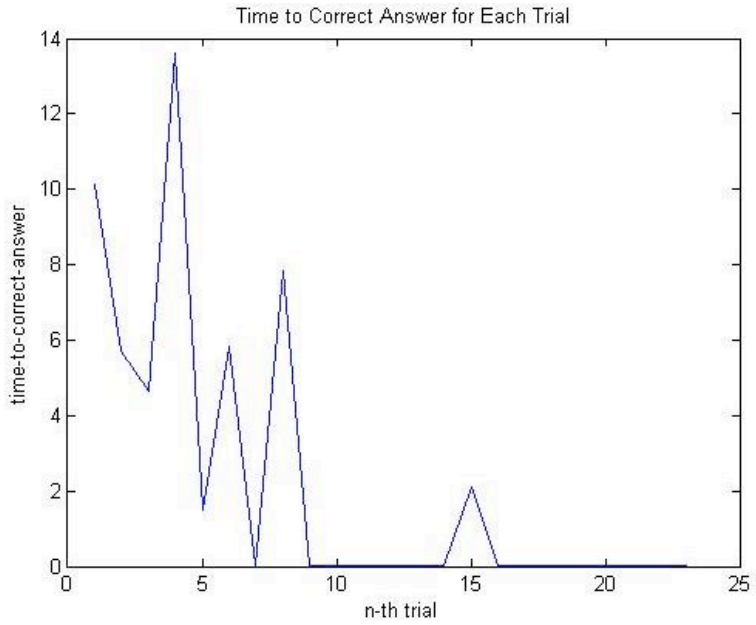



**Figure 1, Time to correct answer for each trial**

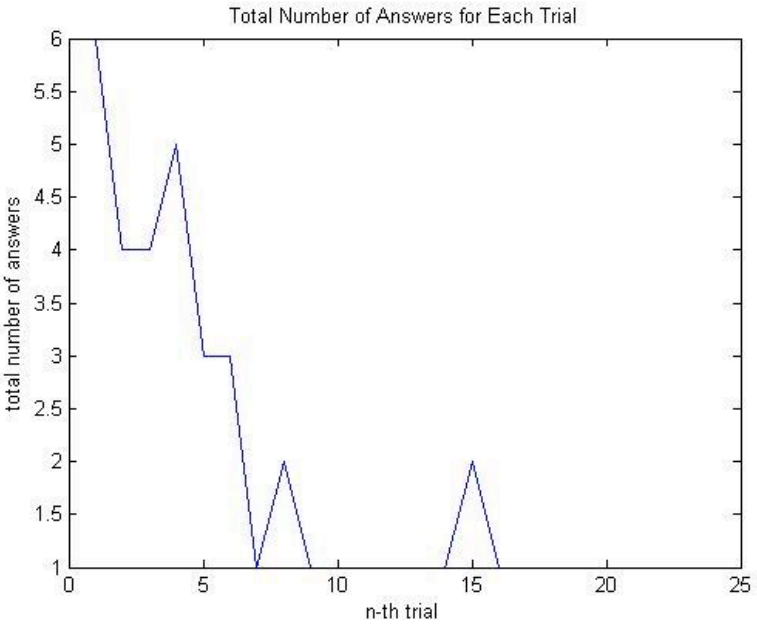



**Figure 2, Total number of answers for each trial**

According to Figure 1 and 2, both the total number of answers and time spent on each trial decreases as the n-th trial number increases. From the data, we can predict that the user begins to notice the relationship between the LED lights in the circuit and the 4 digits binary numbers, and "learn" the conversion as more trials he/she tried.

Both the error and the time reduced to around 1 after around 10 trials. The zigzagged shape in both figures might correspond to the "too easy" or "too hard" answers and it is very reasonable for the data to have some fluctuations because the thinking speed is unpredictable but the trend is close to a "learning curve". (The fitting of the data will be discussed later in "Extra" part.)

The current ratio of total incorrect answers to trials for each trial:

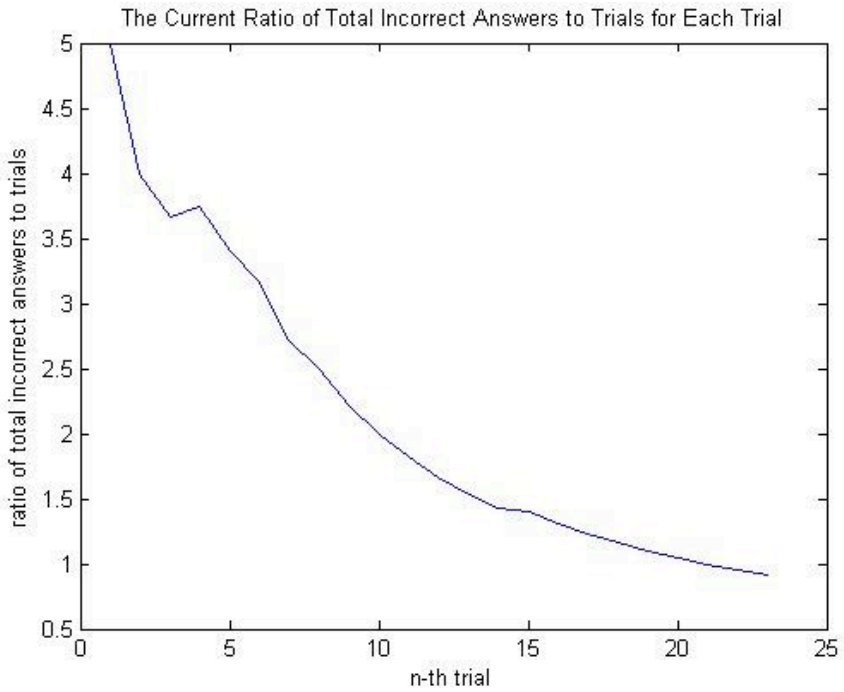



**Figure 3, the current ratio of total incorrect answers to trials for each trial**

The current average response time (over the test so far) for each trial:

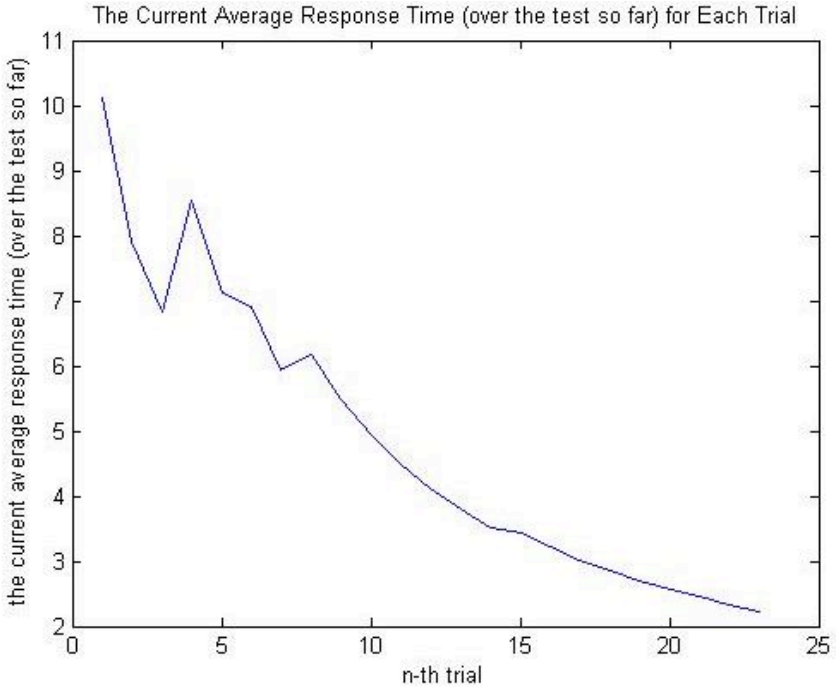



**Figure 4, the current average response time (over the test so far) for each trial**

Both the ratio of total incorrect answer in each trial to total trials and the average response time decreases as the number of trials increases. Figure 3 and 4 are better demonstration of the "learning curve" since the result is somewhat cumulative and reduce the unpredictable "noise" in the data.

(The fitting of the data will be discussed later in "Extra" part.)

## Lab 2:

The outcome phenomenon in Lab 2 is the same as what we expected. The two LED lights oscillate alternatively from increasing brightness to decreasing brightness for 10 times and turn off in the end. Two analog outputs in sine and cosine wave have the same range of oscillation magnitude, but are in the opposite phase. Thus, the brightness of each two LEDs is changing at opposite time (one light reaches the brightest when the other reaches the dimmest). They were brightest and dimmest at opposite time but the degree of brightest and dimmest are the same.

## Lab 3a:

The results of voltage drops across each of the circuit (LED & resistor), LED and resistor are plotted on the same figure.

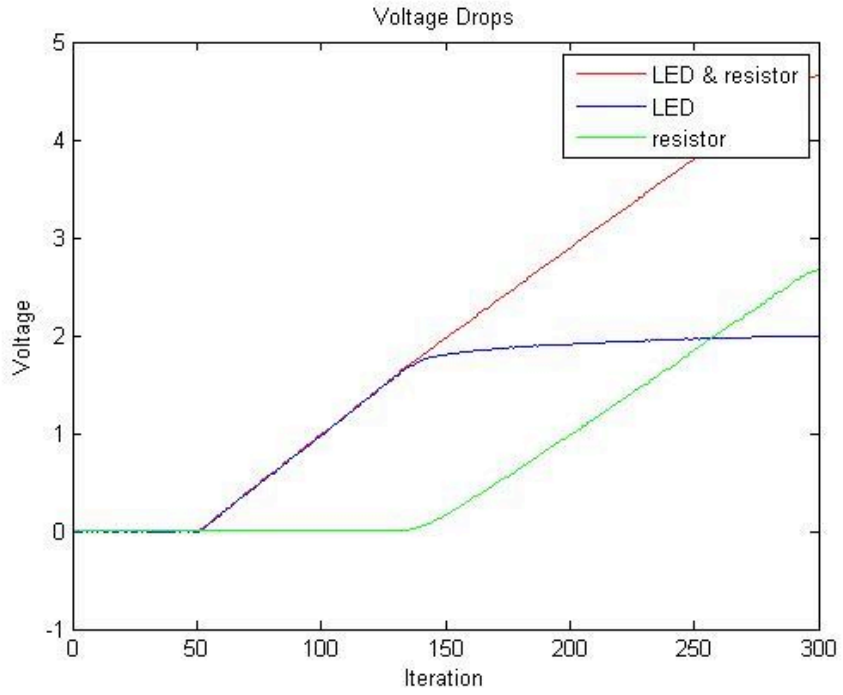



**Figure 5, Voltage Drops (only for Lab 3a)**

The figure shows the voltage drops across each of the three parts of the circuit as total voltage linearly increases from -1 to 5V. According to this figure, the voltages across LED and LED + resistor (whole circuit) start increasing linearly after 50 iterations (0V）. The voltage across LED becomes a flat curve (no significant increase) after 150 iterations (2V), meanwhile, the voltage across the resistor starts increasing linearly which seems to provide an alternative path for the voltage.

From Figure 3 and 4, we can tell that the total voltage drop across the circuit is always a sum of the voltage drops across the LED and resistor, which supported the original claim.

(The fitting of the data will be discussed later in “Extra” part.)

## Lab 3b:

The voltage drops of all three parts after adding a longer pause time (0.04seconds) between two voltage outputs in the loop (25 steps).

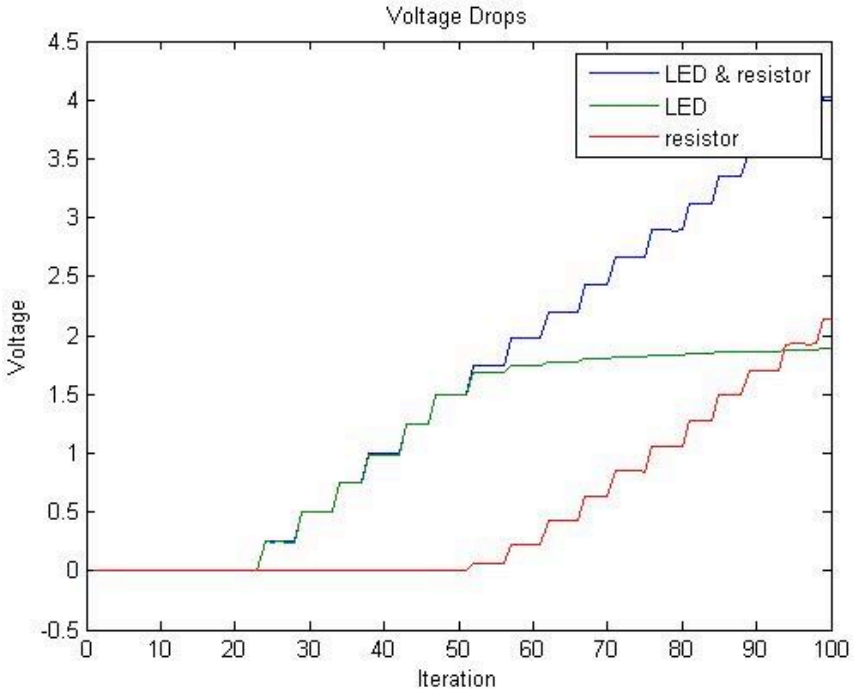



**Figure 6, voltage drops for Lab 3b (with frequency 100Hz)**

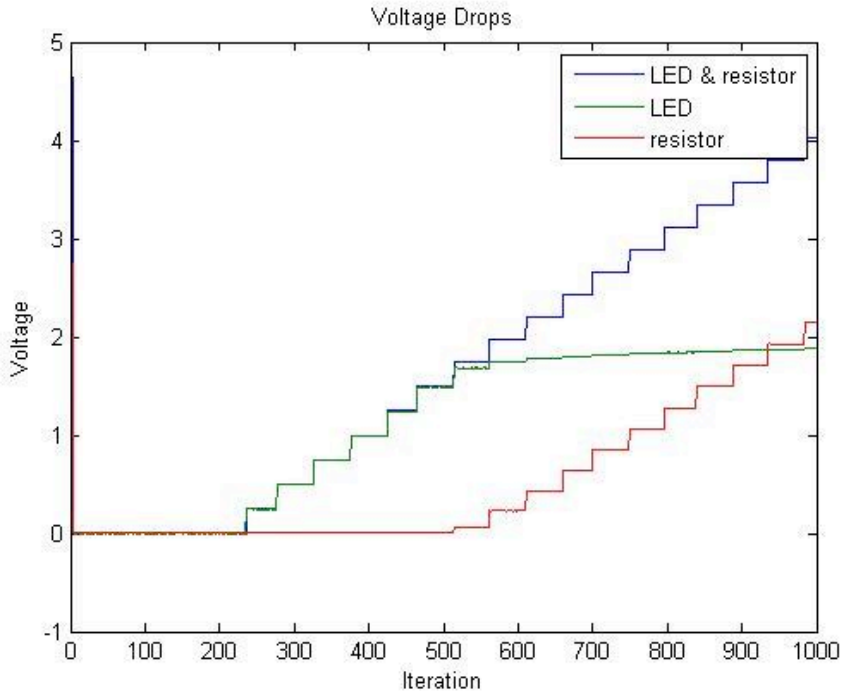



**Figure 7, voltage drops for Lab 3b (with frequency 1000Hz)**

The figure shows the voltage drops across each of the three parts of the circuit as total voltage linearly increases from -1 to 5V. Figure 6 and 7 both show a quantized representation of

the voltage drops across all three components in the circuit. The curve is "stepped", which is exactly the same as what we predicted before running the experiment. When we changes to a higher sampling rate (1000Hz) for the voltage drop, as shown in Figure 7, we will get a curve in exactly the same shape as that with frequency 100Hz.

The voltage across LED also becomes a flat curve (no significant increase) after 50 samples (2V) (or 500 samples in 1000Hz), meanwhile, the voltage across the resistor starts increasing linearly which seems to provide an alternative path for the voltage.

(Interesting Phenomenon)

LED only:

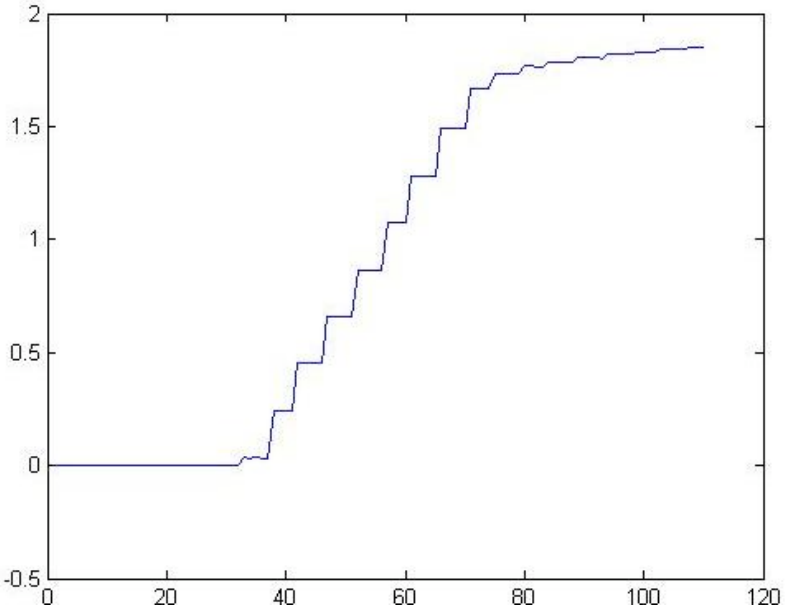



**Figure 8, voltage drop across LED with frequency 100Hz**

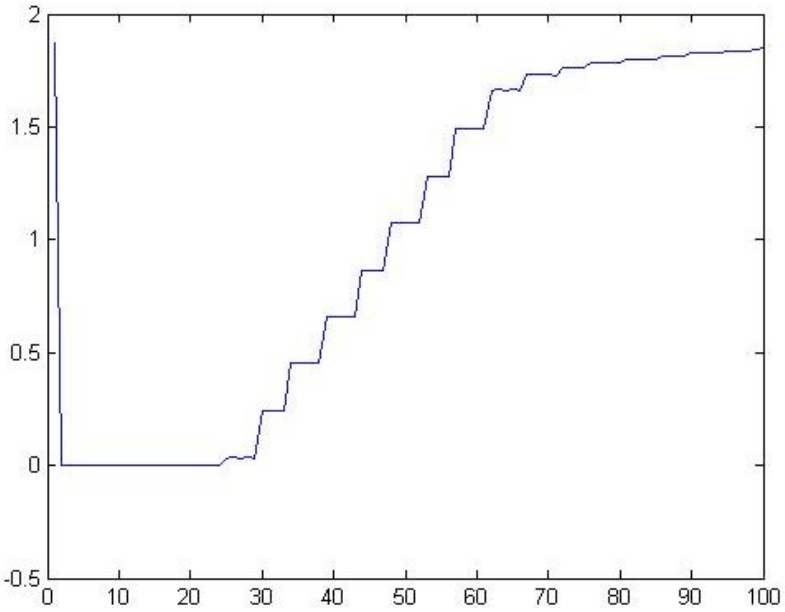



**Figure 9, voltage drop across LED with frequency 100Hz (after running the program once)**

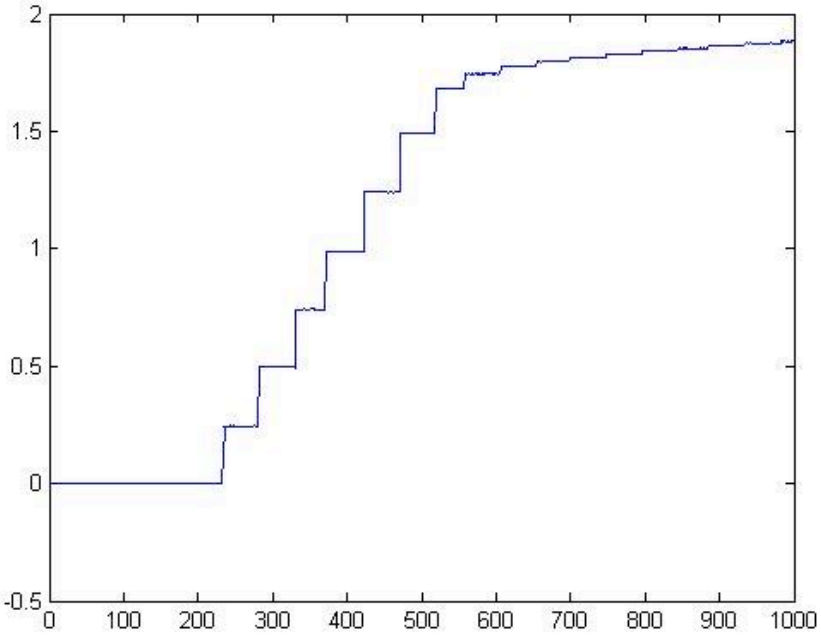



**Figure 10, voltage drop across LED (with frequency 1000Hz)**

The three figures above show the results of voltage drops across only the LED using both 100Hz and 1000Hz sampling rates.

Although Figure 8 and Figure 9 have very similar shape, it should be easy to find that, in Figure 9, there is an immediate drop in voltage in the beginning of the experiment. The reason for the sudden drop in the beginning is clear through comparison in experiments. It is because that a similar program has been run at least once before running the experiment to generate the data plotting in Figure 9. Thus, there will definitely be voltage left in the circuit when we were running the experiment for data plotting in Figure 9. It drops from 5V because we add the output voltage from -1 to 5V in the experiment before that, so the remaining voltage starts at 5V.

The high voltage suddenly drops after the program enters a new loop where the output voltage is reset to the initial value of -1V.

The voltage in the beginning for the sudden drop is not set to zero at once, which shows that MATLAB began input data before the output voltage is reset to -1V.

When the sampling rate is changed from 100Hz to 1000Hz, by comparing Figure 8 and 10, the shape of the two figures are the same (both “stepped” in the same way).

(Note: continued on next page in order to better fit the figures)

Resistor only:

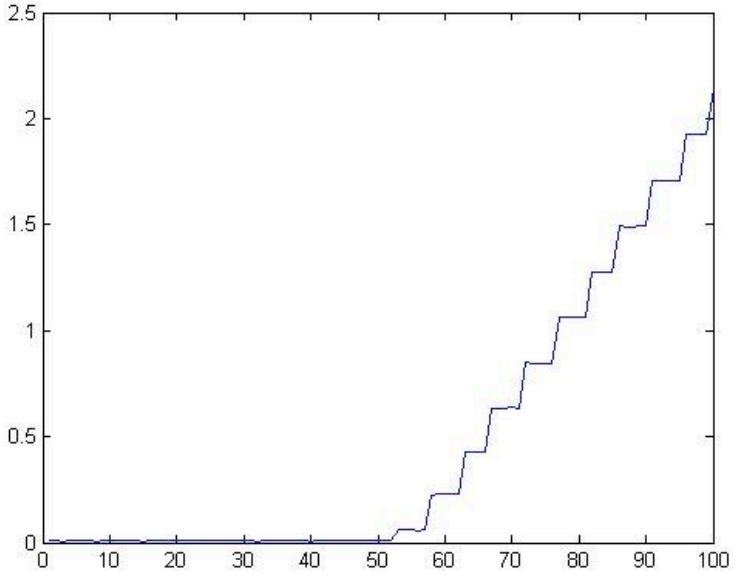



**Figure 11, voltage drop across resistor with frequency 100Hz**

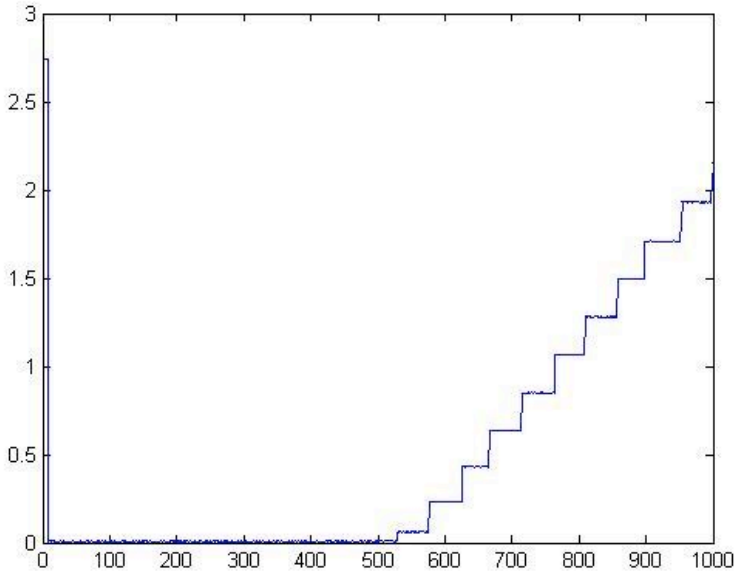



**Figure 12, voltage drop across LED with frequency 1000Hz**

According to Figure 11 and 12, the voltage drop across only the resistor shows a similar result as that of the LED. The voltages across LED and LED + resistor (whole circuit) start increasing linearly after 50 samples (0V) (or 500 samples with 1000Hz).

The two Figures with different sampling rate are in the same shape if we also adjust the iteration to 1000 when changing the sampling rate. Thus, we can conclude that only changing the sampling rate (frequency) will not matter the outcome, since the shape of the curve is basically the same (nothing interesting shows up).

In this case for the resistor, the number of iterations when the voltage increases with 1000Hz sampling rate is almost 10 multiple to that with 100Hz.

Circuit only (LED + resistor):

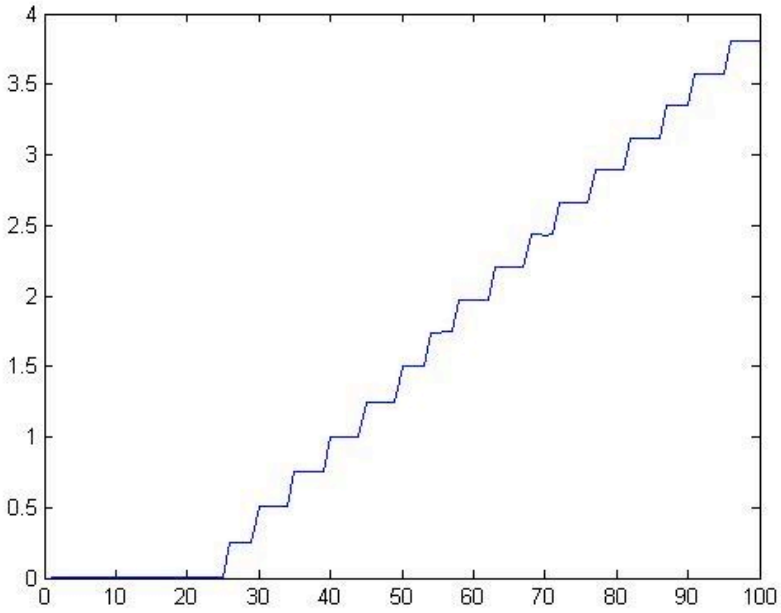



**Figure 13, voltage drop across LED and resistor with frequency 100Hz**

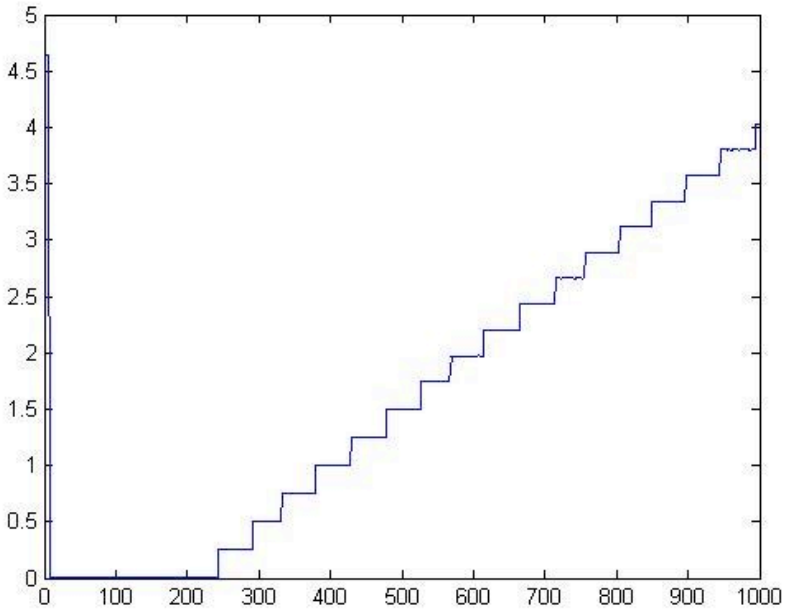



**Figure 14, voltage drop across LED with frequency 1000Hz**

According to Figure 13 and 14, for the voltage drops across the whole circuit with different sampling rates, the shape of the "stepped" curves are the same. The numbers of samples when the voltages start to increase hold the same relationship between the voltages with two different

rates (increase by a factor of 10 when sampling rate increases from 100Hz to 1000Hz). Thus, we can verify that only changing the sampling rate (frequency) will not matter the outcome, since the shape of the curve is basically the same (nothing interesting shows up).

Experiment with trigger:

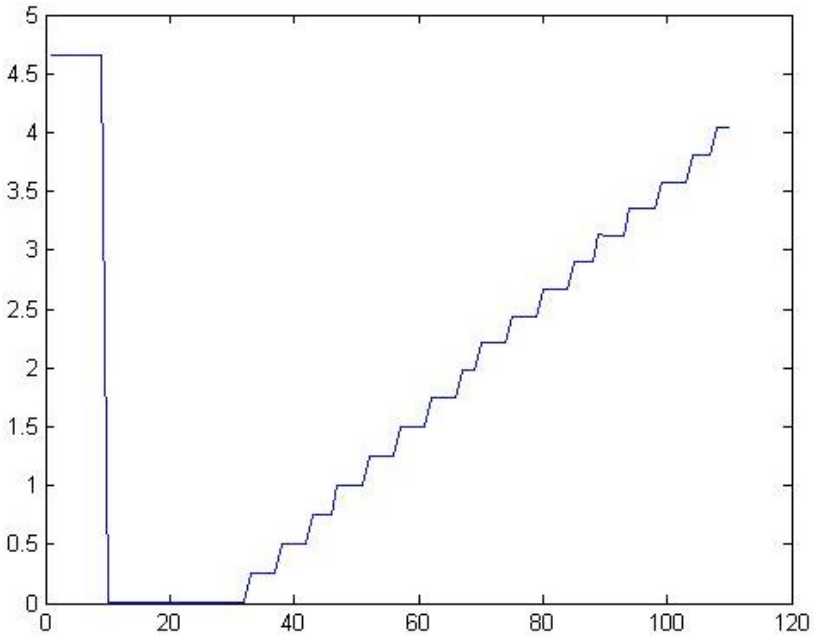



**Figure 15, voltage drop across LED with frequency 100Hz (with trigger and voltage range from -1 to 5v)**

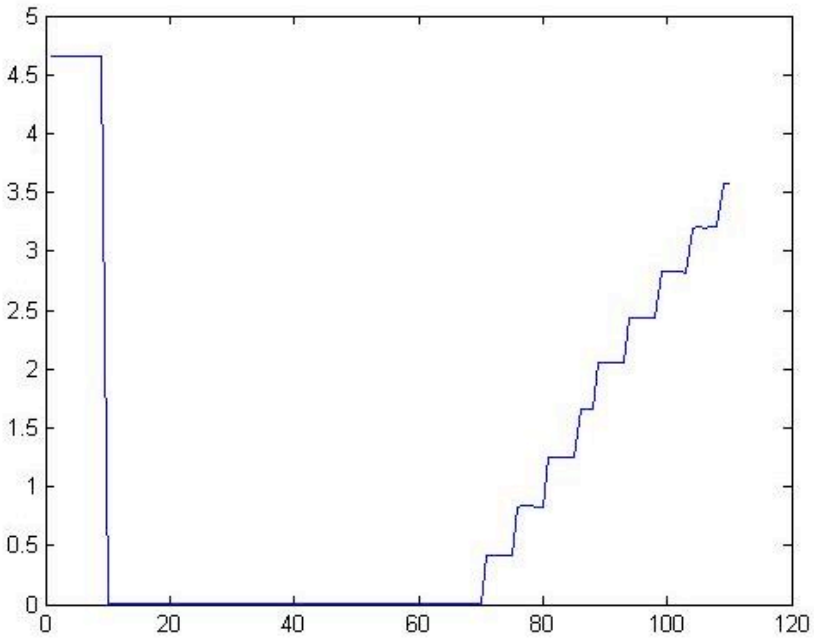



**Figure 16, voltage drop across LED with frequency 100Hz (With trigger and voltage range from -5 to 5v)**

When doing Lab 3b, we are asked better not to use the 'trigger' function, which set a trigger before starting the input acquisition. To explore the effect of trigger, we run the experiment twice, once without trigger (results have been discussed above) and once with trigger (results refer to Figure 15 and 16). When we are using trigger, it is obvious in the figure that there is a longer

waiting time before the immediate drop in the beginning, which indicate a longer time delay before we start entering the 'for' loop.

From Figure 15 and 16, we can see that there are more "steps" in the figure for voltage drop with trigger and voltage range from -1 to 5v than the one with trigger and voltage range from -5 to 5v. That is because the latter one should start increasing later than the first one, since from -1 to 5V, the voltage will start increasing from 0V (1/6 of the full range), but from -5 to 5 V, the voltage will also start increasing from 0V (1/2 of the full range). Therefore, the phenomenon indicated in the figure matched our prediction.

## Extra credit:

### Lab 3a (Data Analysis)

From the former plotting for Lab3a (shown in the above), we know that the non-linear tailing-off part of the LED's voltage curve starts at around 150 iterations, so to fit in the model, we will only analyze with the first 150 iterations.

First, use 'AnalyzeExp' and 'AnalyzePower' function to plot and check the goodness of the data fit for either exponential function or power function:

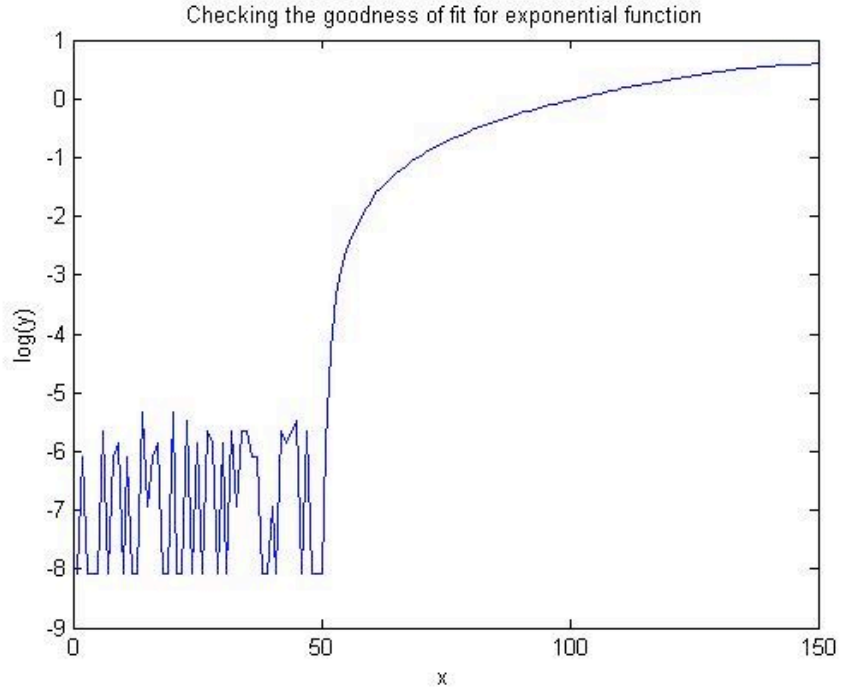



**Figure 17 Checking if voltages drop for LED fit for the exponential function**

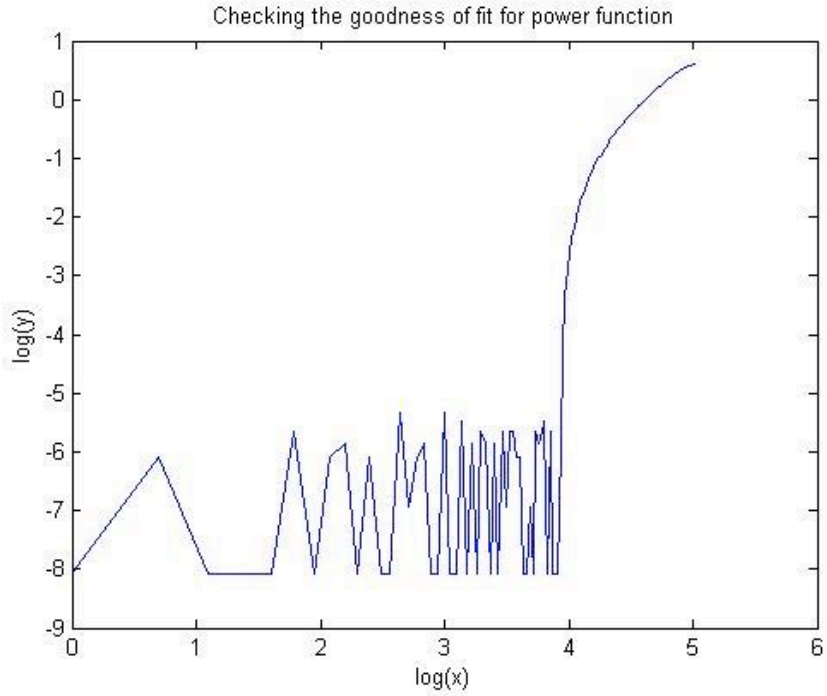
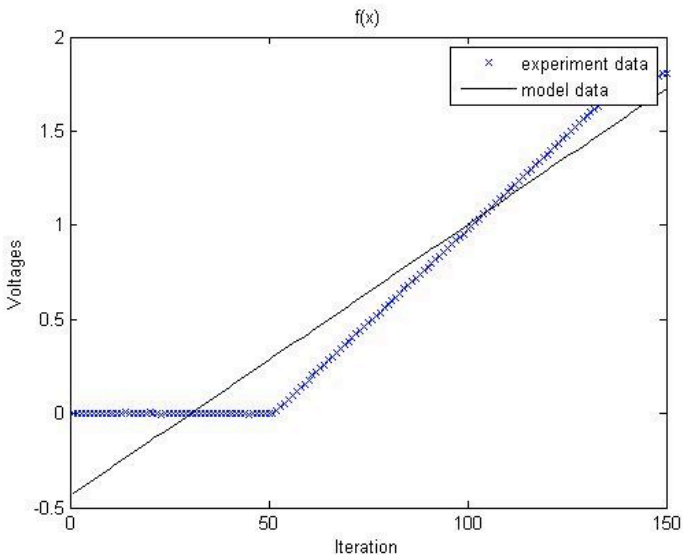



**Figure 18 Checking if voltages drop for LED fit for the power function**

## Statistical Analysis:

From Figure 17, the logarithm of y does not have a linear relationship with x, so the data for voltages drop of LED will not fit in the exponential function. Similarly, from Figure 18, the logarithm of y does not have a linear relationship with the logarithm of x, so the data for voltages drop of LED will not fit in the power function, either.

Since either exponential function or power function will not fit the data, try polynomial functions:



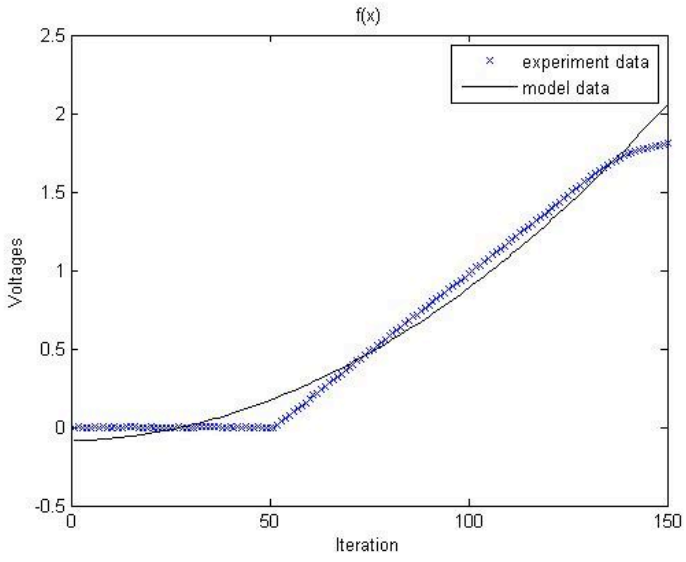
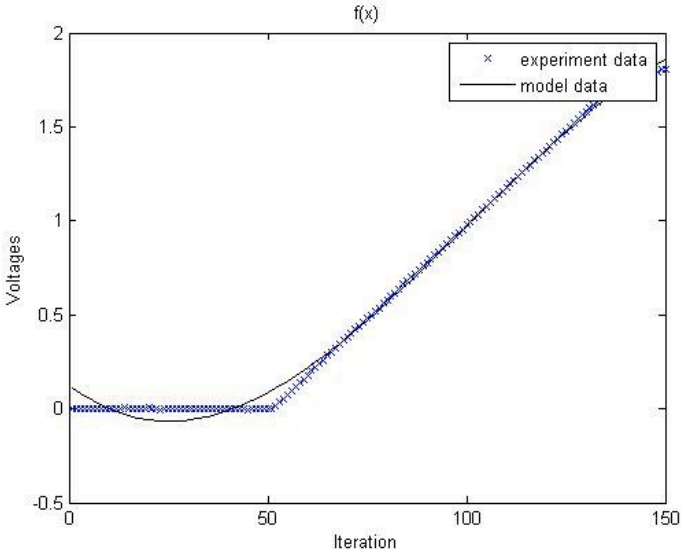




**Figure 19 1st –order polynomial function (LED Voltage – Lab 3a) Figure 20 2nd –order polynomial function (LED Voltage – Lab 3a)**



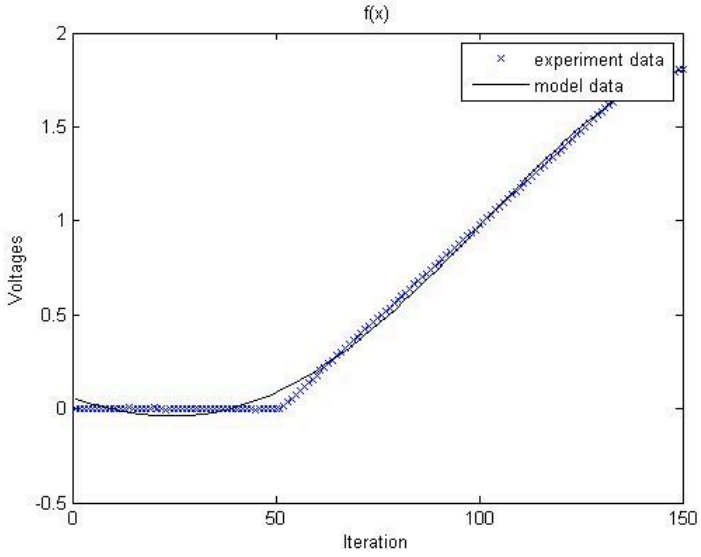




**Figure 21 3rd –order polynomial function (LED Voltage – Lab 3a) Figure 22 4th –order polynomial function (LED Voltage – Lab 3a**

Statistical Analysis:

1st order polynomial function: $y = -0.4368 + 0.0144x$

2nd order polynomial function: $y = -0.0877 + 0.0006x + 0.0001x^2$

3rd order polynomial function: $y = 0.1280 + -0.0162x + 0.0004x^2 + -0.0000x^3$

4th order polynomial function: $y = 0.0604 + -0.0075x + 0.0001x^2 + 0.0000x^3 + -0.0000x^4$

| LED Voltage (Lab 3a) Data Fits: | | | | |
|---|---|---|---|---|
| Models | 1st polynomial | 2nd polynomial | 3rd polynomial | 4th polynomial |
| Standard Error | 0.1772 | 0.0878 | 0.0381 | 0.0317 |

From the Figure 19 - 22 above, we can choose the 1st or 2nd-order polynomial to be the best fit since the model fit the data quite well (will not cause a too high standard error) and will also over-fitting.

(1st order polynomial function)

$$\text{Voltages (LED)} = -0.4368 + 0.0144(\#\text{ of iterations})$$

(2nd order polynomial function)

$$\text{Voltages (LED)} = -0.0877 + 0.0006(\#\text{ of iterations}) + 0.0001(\#\text{ of iterations})^2$$

Approaching from another way, from the statistic table above, the 4th-order polynomial has the smallest standard error, so the 3rd-order polynomial can be the best fit but since the coefficient for the third and fourth degree x is too close to zero, it may be over-fitting.

(4th order polynomial function)

$$\text{Voltages (LED)} = 0.0604 + -0.0075(\#\text{ of iterations}) + 0.0001(\#\text{ of iterations})^2 + \mathbf{0.0000}(\#\text{ of iterations})^3 + -\mathbf{0.0000}(\#\text{ of iterations})^4$$

## Lab 1 (Data Analysis)

First, use 'AnalyzeExp' and 'AnalyzePower' function to plot and check the goodness of the data fit for either exponential function or power function:

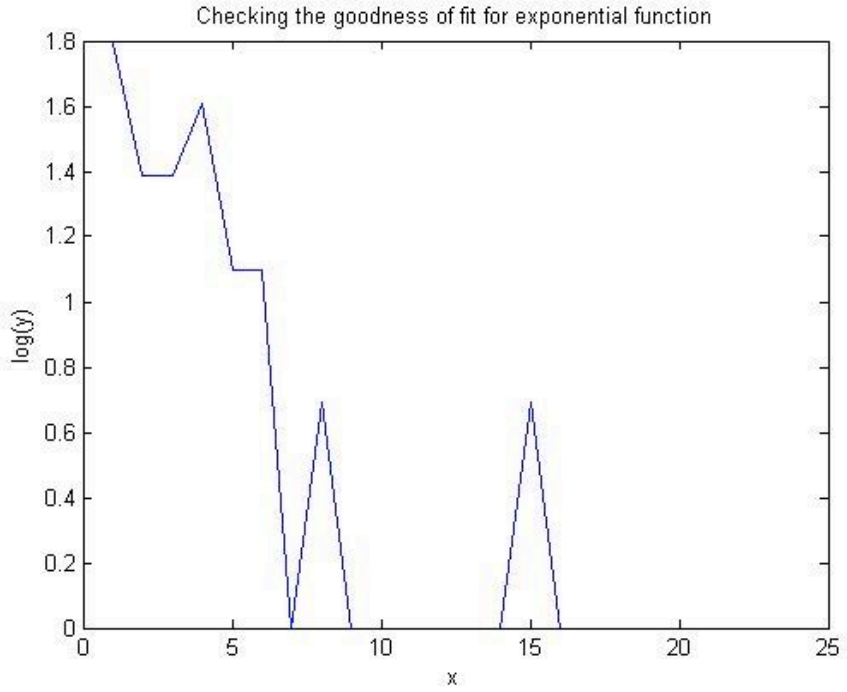



**Figure 23 Checking if "Time to correct answer for each trial" fit for the exponential function**

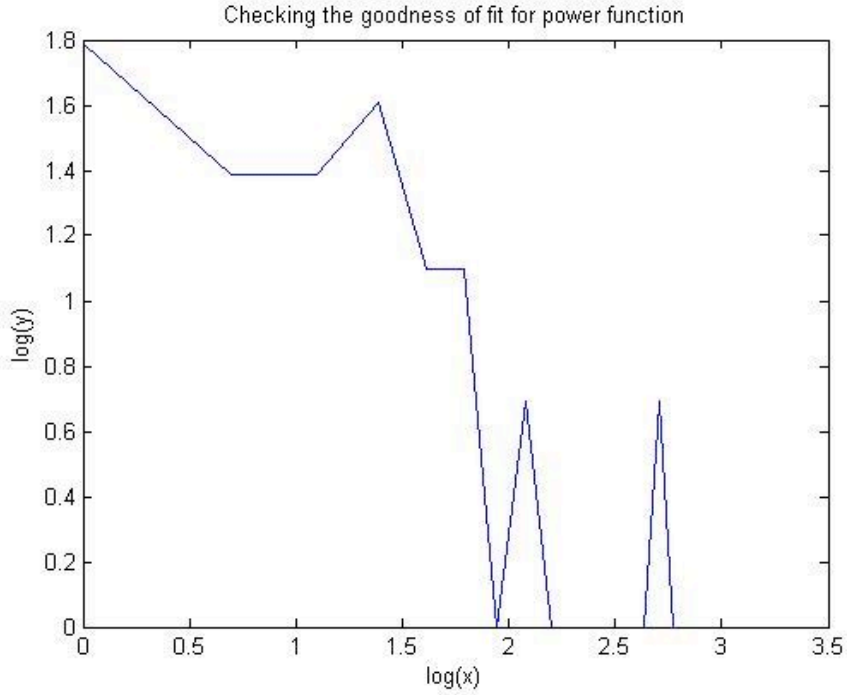



**Figure 24 Checking if "Time to correct answer for each trial" fit for the power function**

## Overall Analysis:

From Figure 23, the logarithm of y does not have a linear relationship with x, so the data for "Time to correct answer for each trial" will not fit in the exponential function. Similarly, from Figure 24, the logarithm of y does not have a linear relationship with the logarithm of x, so the data for "Time to correct answer for each trial" will not fit in the power function, either. However, the main reason for why the data is not fitting may because of the high fluctuation of the data (due to the high variation of the user's input)

Since either exponential function or power function will not fit the data, try polynomial functions:

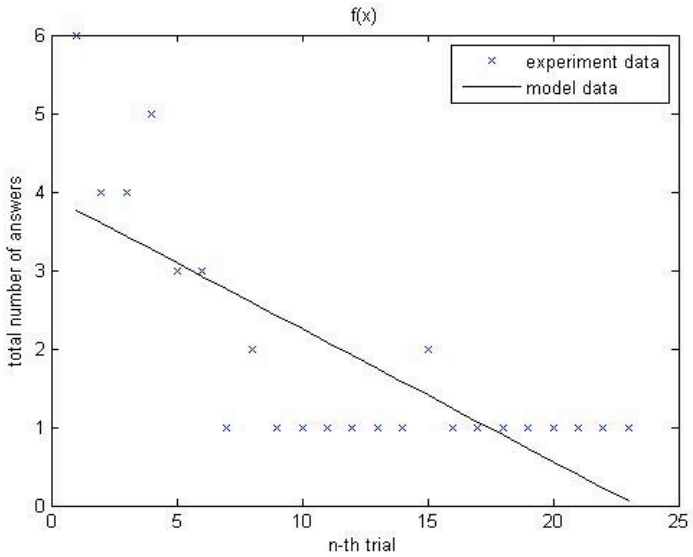



**Figure 25 1st –order polynomial function (Lab 1 – Total Time)**

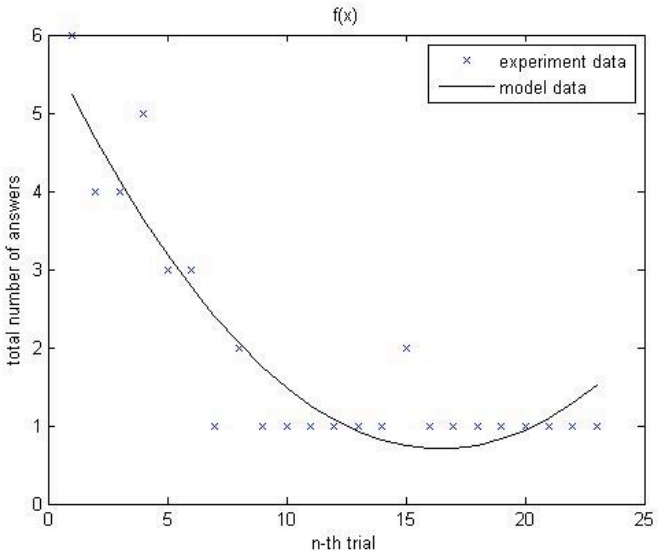



**Figure 26 2nd –order polynomial function (Lab 1 – Total Time)**

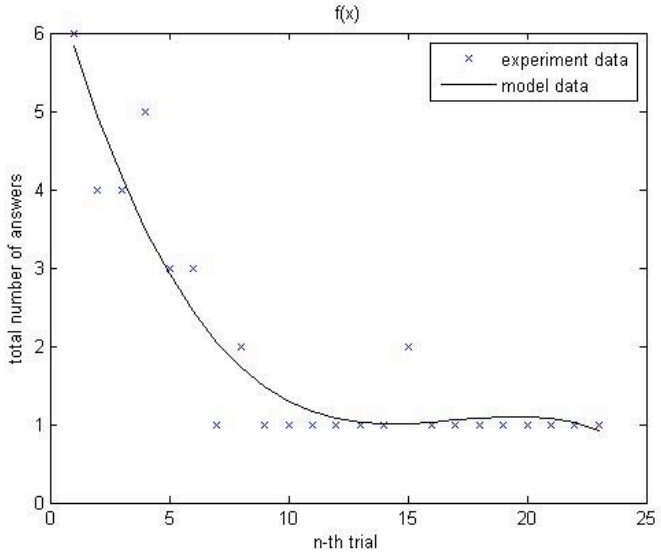



**Figure 27 3rd –order polynomial function (Lab 1 – Total Time)**

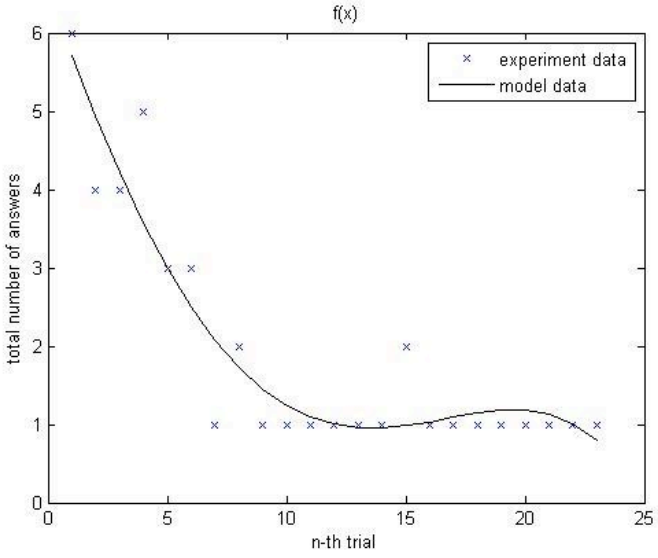



**Figure 28 4th –order polynomial function (Lab 1 – Total Time**

### Statistical Analysis:

| Total number of answers (Lab 1) Data Fits: | | | | |
|---|---|---|---|---|
| Models | 1st polynomial | 2nd polynomial | 3rd polynomial | 4th polynomial |
| Standard Error | 0.9984 | 0.6352 | 0.5655 | 0.5766 |

From the Figure 23 - 28 above, we can conclude that the raw data cannot fit in well in any of data fitting curve above (either have too high standard deviation or may count as over fitting). The main reason for why the data is not fitting may because of the high fluctuation of the data (due to the high variation of the user's input).

Similarly, after the same processing as above, we can conclude that the raw data "Total number of answers for each trial" cannot fit in well in any of data fitting curve above and the same main reason can be applied again.

### The current ratio of total incorrect answers to trials:

Since "the current ratio of total incorrect answers to trials" is the current average value derived from the raw data, the processed data will be more consistent and have lower fluctuation than the raw data.

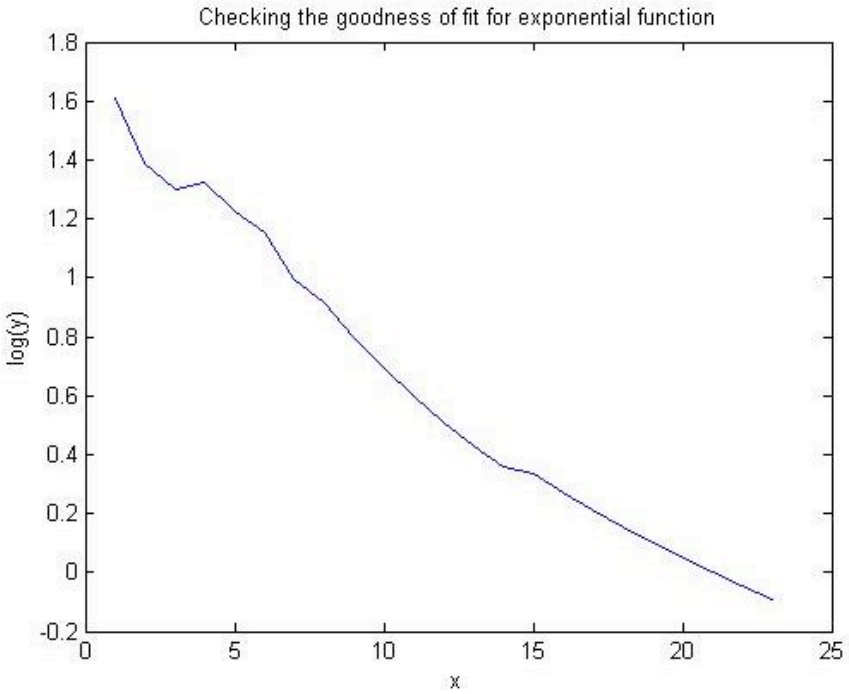



**Figure 29**

**Checking if "the current ratio of total incorrect answers to trials for each trial" fit for the exponential function**

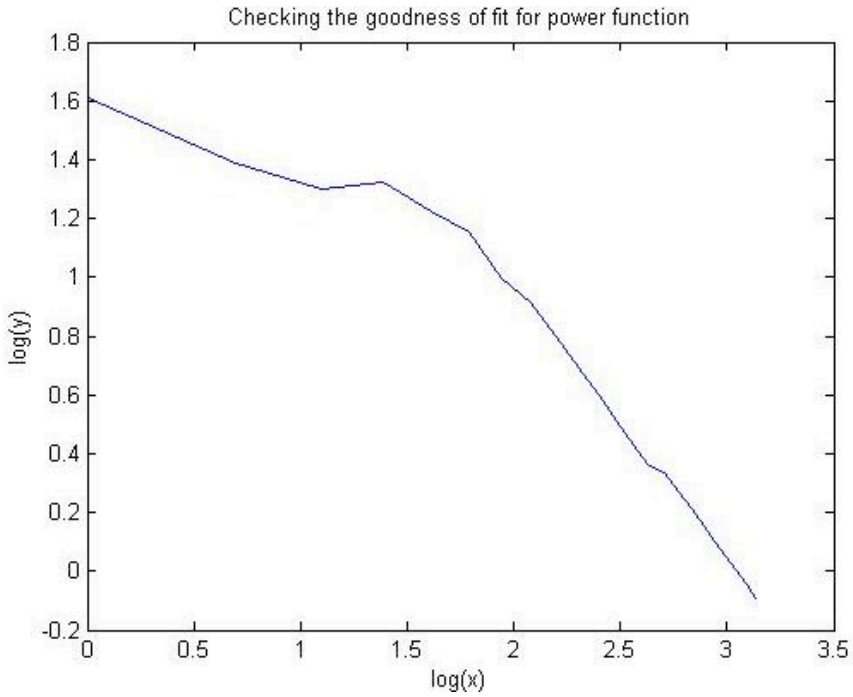



**Figure 30**

**Checking if "the current ratio of total incorrect answers to trials for each trial" fit for the power function**

From Figure 29, the logarithm of y has an almost linear relationship with x, so the data for "Time to correct answer for each trial" might fit in the exponential function. From Figure 30, the logarithm of y does not have a linear relationship with the logarithm of x, so the data for "Time to correct answer for each trial" will not fit in the power function. The high fluctuation of the raw data has been reduced by taking the average.

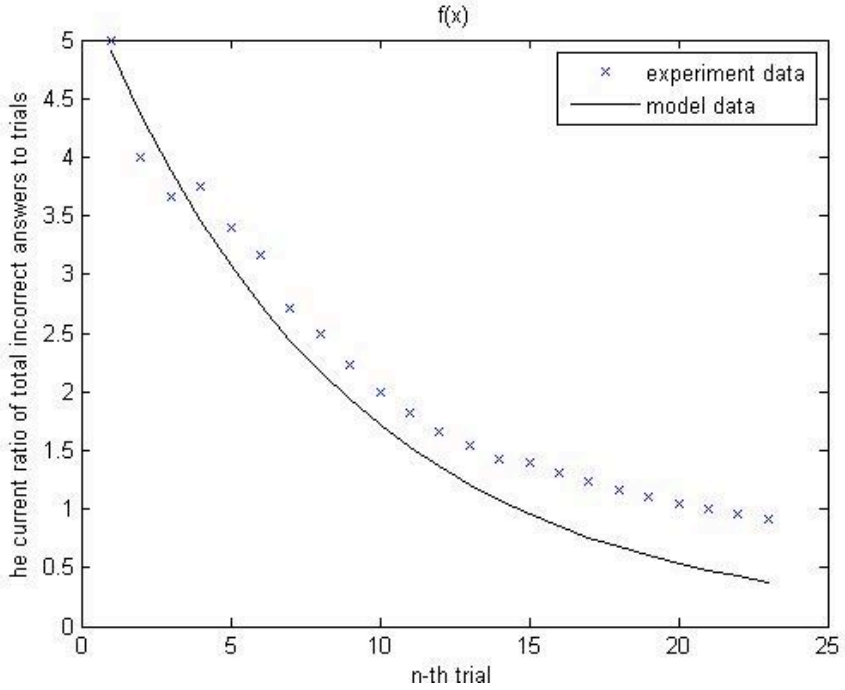



**Figure 31 - The exponential function - "the current ratio of total incorrect answers to trials for each trial"**

Statistical Analysis:

The exponential function: $y = 5.5086 * e^{-0.1165}$

Standard Error: 0.4112

From the Figure 31 above, we can conclude that the processed data cannot fit exactly well in the exponential function curve due to the first two outliers. If we eliminate the outlier, the data of "the current ratio of total incorrect answers to trials for each trial" should fit in the exponential function.

The current average response time (over the test so far) for each trial:

Since "the current average response time (over the test so far) for each trial" is the current average value derived from the raw data, the processed data will be more consistent and have lower fluctuation than the raw data.

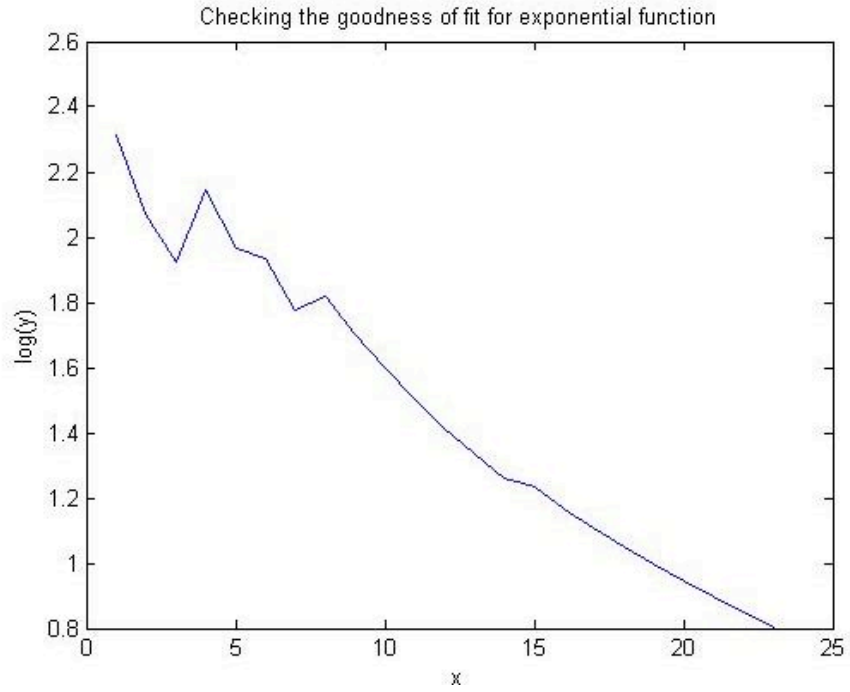



**Figure 32 - Checking if "the current average response time" fit for the exponential function**

(Note: continued on next page in order to fit figures)

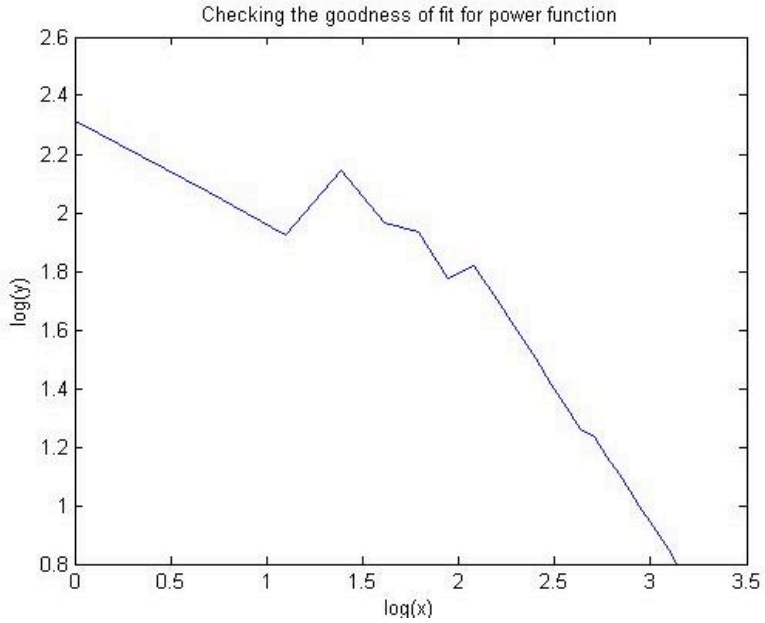



**Figure 33 - Checking if "the current average response time" fit for the power function**

From Figure 32, the logarithm of y has an almost linear relationship with x, so the data for "Time to correct answer for each trial" might fit in the exponential function. From Figure 33, the logarithm of y does not have a linear relationship with the logarithm of x, so the data for "Time to correct answer for each trial" will not fit in the power function. The high fluctuation of the raw data has been reduced by taking the average.

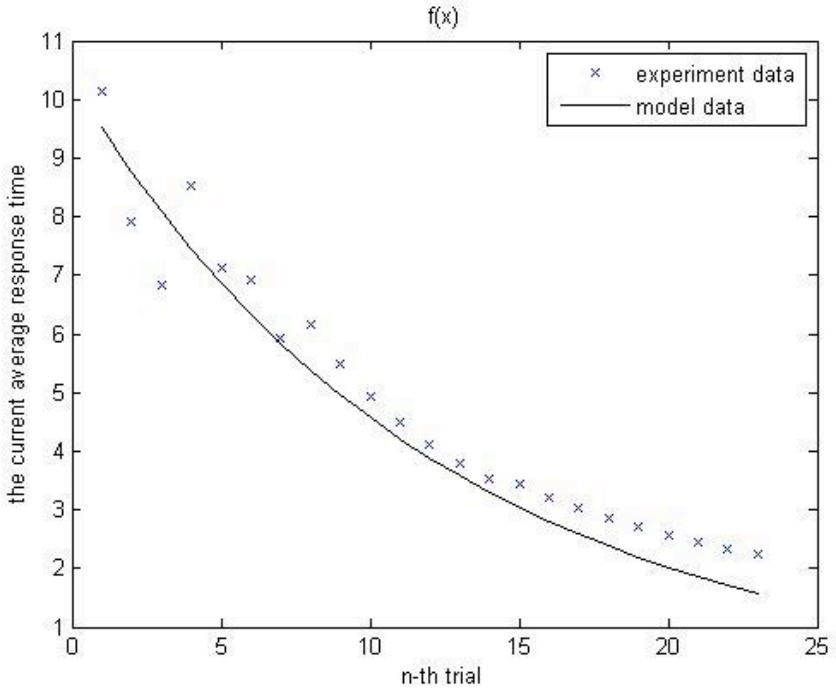



**Figure 34 - The exponential function – "the current average response time"**

Another approach from polynomial function:

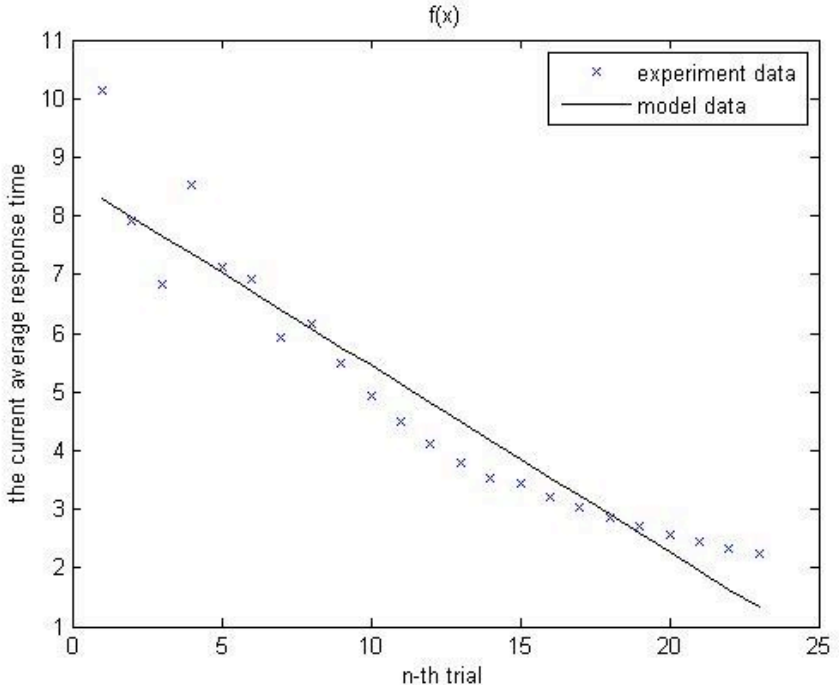



**Figure 35 - The 1st-order polynomial function – "the current average response time"**

## Statistical Analysis:

The exponential function: $y = 10.3277 * e^{-0.0815}$

1st order polynomial function: $y = 8.6175 + -0.3172x$

Standard Error (fitting in exponential function): 2.7689

Standard Error (fitting in 1st-order polynomial function): 0.6860

From the Figure 31 above, we can conclude that the processed data can fit well in the exponential function curve. In addition, the data for "the current average response time" also fit quite well in 1st-order polynomial function, since the standard deviation is even smaller than fitting into the exponential function. However, we may still consider the exponential function as the best fit, according to the flow of the original experiment data (the dots plotted on the figure).

## Further Explanation:

### Learning Curve:

(Wikipedia.org)

A learning curve is a graphical representation of the changing rate of learning (in the average person) for a given activity or tool. Typically, the increase in retention of information is sharpest after the initial attempts, and then gradually evens out, meaning that less and less new information is retained after each repetition.

## Appendix

The "Introduction" and "Procedure" part above are COMPLETELY derived from the ACTUAL EXPERIMENT, which is not the Professor's version on the website or other Internet-searched copies.

## References

Brown, C. 2011 Data Acquisition

The main ideas of the functions (including the inside computing method) mainly originated from BlackBoard CSC 160 "Data Acquisition: The LED Labs" page.

Author: Chris Brown

Complete URL:

http://my.rochester.edu/webapps/portal/frameset.jsp?tab_tab_group_id=_2_1&url=/webapps/blackboard/execute/courseMain?course_id=_40459_1

Date that accessed the site: 03/26/2011

Author: (source) Wikipedia.org

Complete URL:

http://en.wikipedia.org/wiki/Learning_curve

Date that accessed the site: 04/11/2011