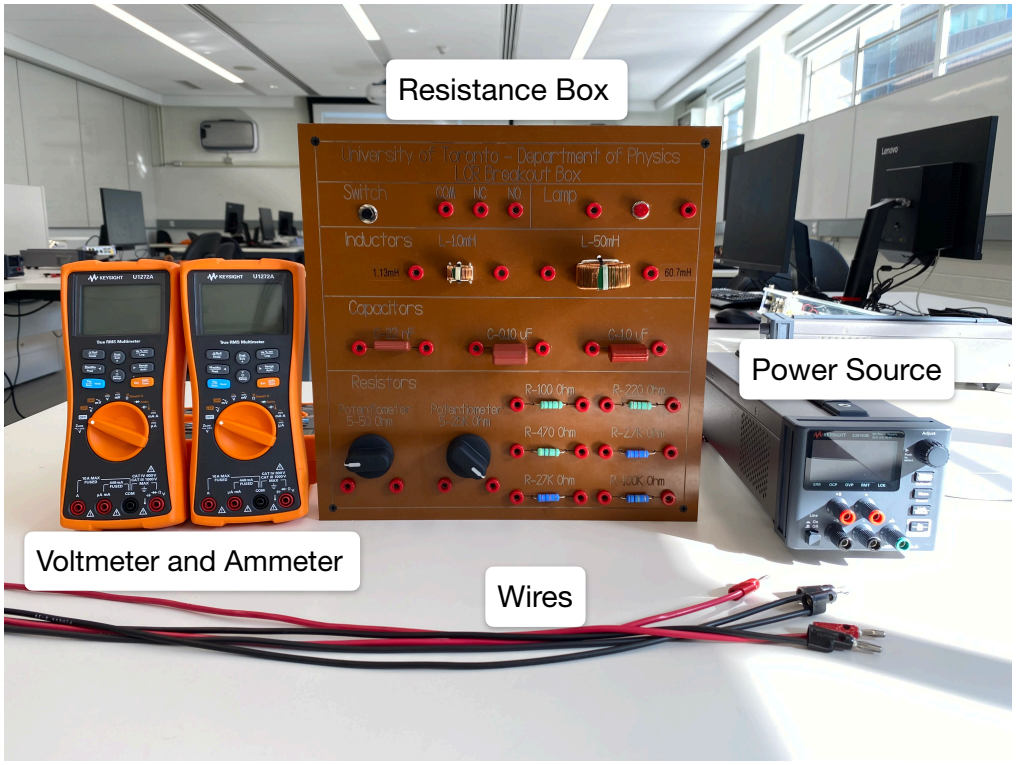
Physics
UNIVERSITY OF TORONTO

Ohm and Power laws



Revisions

2022 T. Vahabi, E. Horsley, A. Harlick
2017 C. Lee
2016 J. Ladan and R. M. Serbanescu

current revision: e2ee8b3
date: September 28, 2023

Introduction to fitting methods

Background knowledge for Part I

- **Python**: `arrays`, `numpy`, `pyplot`, `scipy`, `curve_fit()`
- **Error Analysis** Chi squared (χ^2), goodness of fit
- **Physics**: Ohm's law. Review appropriate sections of your textbook.

Introduction

The function `curve_fit()` is introduced for fitting linear data. You will perform voltage and current measurements on a resistor to find the resistance via Ohm's law.
You are expected to maintain a notebook (physical or electronic) where you record your measurements, create plots, and write Python code appropriate for the lab.

This lab contains explicit questions for you to answer labelled with "Q". They are intended as prompts for analysis and discussion. The answers to the questions should be organically interwoven into your report.

In this exercise, we will introduce how to fit experimental data to a certain function, and get *quantitative* observations from the experiment.

The package "Notes on Error Analysis" can be found at https://www.physics.utoronto.ca/~phy224_324/web-pages/Notes_Error.pdf. It introduces linear and non-linear regression methods, aimed at finding the values of parameters which give the closest agreement between the data and the theoretical model. You should read the section on least-squares fitting.

Suppose our data consisted of N measurements (y_i, x_i) with $i = 1, 2, ..., N$. A **first step is to plot the data with error bars representing the reading error** (or the standard deviation σ_i) on each point. This gives a visual check for what relation our data has.

Next, assume that we want to make a function f that will predict a value $\bar{y}_i$ if we call the function with the measurement x_i and some set of parameters $p_0, p_1. \ . \ .$ that represents the experiment (e.g. resistance, gravity, spring constant could all be parameters). Our goal is to find the value of the parameters that will allow the function f to make the best predictions given the available data.

We could use this function f to make predictions for each measurement pair (x_i, y_i), and make guesses about the parameters until we find 'good' agreement, but a more robust method is to use the data fitting capabilities of Python to find the parameter values for us, using a form of *linear regression*.

Curve fitting

To make use of Python in our regression, we need to write our mathematical relationship between x and y as a Python function. In this experiment we'll be looking at an experiment that we think is modeled by a linear equation $y = ax + b$, where a and b are the parameters we are trying to determine. In Python code this becomes

```
def model_function(x, a, b):
    return a*x + b
```

Note that **the independent variable x must be the first argument**. In order for your `model_function` to work it must be able to take a numpy array and return an array.

This function only makes a prediction of the measurement of y given x. The job of finding the parameters is done by the scipy package in Python, using the `curve_fit()` function from the scipy.optimize module.

Every time you call curve_fit() you will give it 3 arguments, and usually more. The first argument is your `model_function` that `curve_fit()` will use to make predictions, followed by the x data, and the y data. `curve_fit()` can also take three additional arguments — you can provide an initial guess for your parameters using the $p0$ keyword argument, the uncertainties in the *dependent* data in the sigma keyword argument, and you should (almost always) and force `curve_fit()` to consider uncertainties as absolute values (instead of relative values) using the `absolute_sigma=True` keyword. Your code will then look like this:

```
def model_function(x, a, b):
    return a*x + b

p_opt, p_cov = curve_fit(model_function, xdata ,
                         ydata , p0=initial_guess ,
                         sigma=sigmadata , absolute_sigma=True)
```

`curve_fit()` returns two values. The first value `p_opt` contains the estimates of the parameters you used in the model (in this case `p_opt` has two elements). The second value `p_cov` contains the covariance matrix giving the uncertainties in estimates of the parameters you used in the model (in this case `p_cov` is a 2×2 matrix). In most experiments the important values in `p_cov` will be the diagonal elements, which represent the variance (the uncertainty–squared) of the parameters. An easy way to extract these values and convert them to uncertainties is :

```
p_std = sqrt(diag(p_cov))
```

where `sqrt` and `diag` are functions from numpy that calculate the square root, and extract the diagonal elements, respectively. `p_std` will now contain your parameter uncertainties.

Curve fitting internals

Internally, `curve_fit()` uses an algorithm to find the best parameters. It does this by defining a *metric* called χ^2 that it tries to minimize. Starting with your initial guess for the parameters, `curve_fit()` will calculate the value of χ^2 based on the experimental data and your model, and adjust the parameters systematically until it finds the minimum value of χ^2 within a (very small) uncertainty.

For simple equations, like the linear model above, `curve_fit()` will take approximately seven tries to find the best set of parameters. With more complex non–linear functions it can take about 50 tries *if you make a good initial guess*, and more than $1,000$ tries *if you make a bad initial guess*. In the default setup `curve_fit()` is configured to stop trying after 800 tries, where it will report that it failed and stop the program.

It is, however, possible for `curve_fit()` to stop 'successfully' and have an ill fitting model if your model function is inappropriate, your data is inconsistent, or your initial parameter guesses were bad. In this case you might see that the parameters haven't been changed from your initial guess, or the variances are infinite (`inf`) or "not a number"(`NaN`).

We will use the metric χ^2 later to measure how good the model function fits the data, but you don't need to define it for `curve_fit()` to work. In words, χ^2 can be written as

$$\chi^2 = \sum_{measurements} \left(\frac{\text{dependent data value - predicted value}}{\text{dependent data measurement error}} \right)^2 \tag{1}$$

For a 'good' fit we want our model prediction to be about as close to each measurement (the numerator) as the uncertainty in each measurement (the denominator). We square the function because a model that is *too high* is just as bad as a model that it *too low*.

Mathematically, χ^2 is written as

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - y(x_i)}{u(x_i)} \right)^2 \tag{2}$$

where x_i and y_i are the data from the i^{th} measurement, $f(x_i)$ is the prediction from the model of the value y_i, and $u(x_i)$ is the uncertainty of the i^{th} measurement.

There are many keyword arguments to curve_fit(), which can be found in the documentation. Commonly, you may use `maxfev` to control the maximum number of function calls. The default value is 800.

The experiment

We will be testing Ohm's Law – the linear relation between voltage and current in electrical circuits. The voltage is our independent variable, current is the dependent variable, and resistance is the parameter we would like to measure. There are two subparts to the experiment: testing a known resistor and testing a potentiometer.

Uncertainty from multimeters

Our later analysis requires an idea of the uncertainty in the measurements. Digital multimeters have two types of uncertainty:

- **Error of accuracy** which depends on function (DCV, ACV, DCA, ACA and Ohm (Ω)). This error is provided by the instrument manufacturer. This error is expressed as a percentage of the reading. Please check the link below for complete instrument specifications. The reading can be done on different scales of the same function, which changes the number of digits on display and also the error of accuracy.
- **Error of precision** which is due to fluctuations at the level of the last digit.

These errors are both present at the same time and vary depending on your reading, the range of values it falls int, and the type of multimeter used. Use the instructions provided in the multimeter manual to determine the values of uncertainties for your voltage, current, and resistance (where appropriate).

Be sure to track the uncertainty in measurements from the multimeter. Also, keep in mind that there are other sources of uncertainty that are harder to measure, like the quality of electrical connections, change in the temperature of the resistor, etc.

Experimental Procedure

You will be provided with two multimeters, a board with electrical components (similar to that shown in Figure 1, and enough wires to connect everything. Choose one resistor, and one potentiometer for the experiment.

Figure 1: The board of electrical elements used in this experiment. The set of resistors is on the bottom-right, the potentiometers are on bottom-left, lightbulb (needed for Part II of this experiment) is in the top-right of the board.

For the known resistor, perform the following.

1. Connect the ammeter, voltmeter, and power supply to the chosen resistor as shown schematically in Figure 2). *Ask the TA if you're in doubt.* Note that Figures 3a and 3b show the visual representation for the set up and the closeup of the power supply indicating appropriate terminals for the connection.

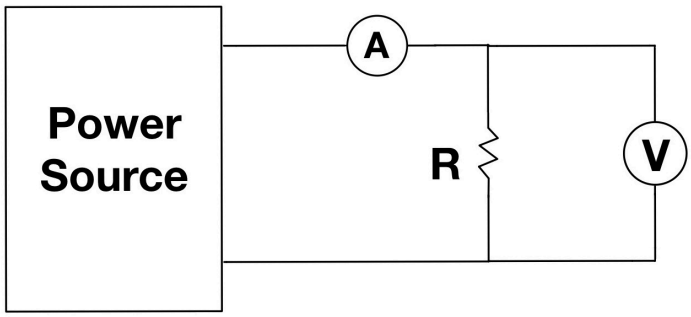
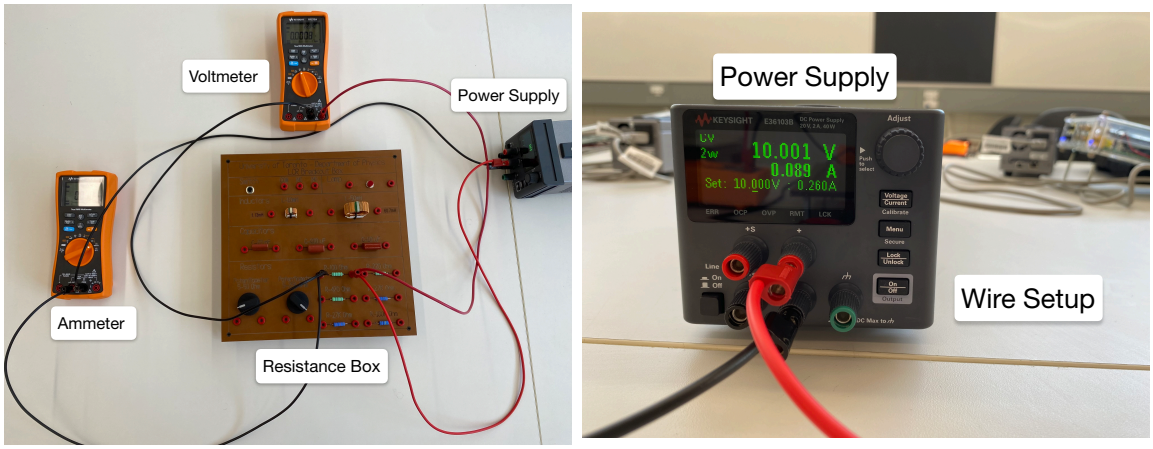


Figure 2: Circuit diagram of the experimental setup.



(a) (b)

Figure 3: Visual representation of the connections for Part I. Figure (a) shows the set up for the resistor, the voltmeter and the ammeter. Figure (b) is a close up of the power supply wire set up

2. Vary the voltage on the power supply.

3. Record the voltage and current from the multimeters, along with uncertainty. Make sure to choose the most appropriate range on the multimeters that allows you to measure your values with the highest precision.

4. Change the voltage, and record the new values – repeat sufficient number of times to have enough data to confidently fit.

5. Save the data (on a memory stick or in a file you store online) as a text file (extension.txt) with the data in two columns: then first column being the independent variable (voltage).

6. After performing all the above measurements, disconnect the power, and switch the voltmeter to become a resistance meter. This will give you a reference value to compare against.

7. Repeat these measurements for a potentiometer, making sure not to move the dial between measurements. If your resistor board has no potentiometer, use another resistor.

Resistors are marked with coloured bars to indicate their nominal resistance through a code (there are many calculators for this online). As part of this exercise, you may also compare your results against the nominal resistance $\pm$ the tolerance.

Analyzing the data

We will analyze the dependency of the current on the voltage using a linear fitting program. Ohm's law for resistors states

$$I = \frac{V}{R}. \tag{3}$$

Thus, the resistance of a linear component can be determined by plotting I vs. V , and measuring the slope.

There are other electrical components that have nonlinear voltage-current relations, like diodes, transistors, and amplifiers. Using a linear model for these would result in a bad fit. In Part II of this exercise, we will perform a nonlinear regression with a lightbulb.

Build a linear fitting program

Linear fitting is done via the Levenberg-Marquadt algorithm (an advanced least-squares method), as implemented in the `scipy.optimize module`. Today, you will use the `curve_fit()` function (i.e. `scipy.optimize.curve_fit)`. Calculation of the statistics from the output of `curve_fit()` will be done in the next exercise.

Here is the outline of the Python program for **Part I**:

- import the necessary modules and functions (e.g. numpy and curve_fit())
- Read the data file into an array with the loadtxt() function.
- Extract the variables from the array columns.
- Define the model function $f(x) = ax + b$.
- Call curve_fit() with the function, data, and initial guess for the parameters.
- Output the calculated parameters.
- Create relevant plots (this includes plotting the residuals).

Write a program to perform a linear fit for the data you collected in this experiment. It should create a plot of current vs. voltage with error bars and the line of best fit, and output the calculated value of resistance (the slope of the line of best fit). Run this program using the data for the resistor, and again using the data for the potentiometer.

Q1 In Ohm's law, $V = IR$, the line should pass through $I = V = 0$. Did your linear fit pass through zero as well? If it did not, why do you think that happened?

Q2 What result do you find for resistance if you force the line to pass through zero? (i.e. try the model function $f(x, a) = ax$)

Q3 How does your resistance from using `curve_fit()` compare to the value measured with the multimeter?

Goodness of fit - reduced chi-squared

Recall that the χ^2 distribution gives a measure of how unlikely a measurement is. The more a model deviates from the measurements, the higher the value of χ^2. But if χ^2 is too small, it is also an indication of a problem: usually that there were not enough samples. This best (most likely) value of χ^2 depends on the number of degrees of freedom of the data, $v = N - n$, where N is the number of observations and n is the number of parameters. This dependence is removed with the reduced chi-squared distribution,

$$\chi^2_{red} = \frac{1}{v} \sum_{i=1}^{N} \left(\frac{y_i - y(x_i)}{u(y_i)} \right)^2 \tag{4}$$

where y_i is the dependent data you measured, x_i is the independent data, $u(y_i)$ is the measurement error in the dependent data.

For χ^2_{red}, the best value is 1:

$\chi^2_{red} \gg 1$ indicates that the model is a poor fit;

$\chi^2_{red} > 1$ indicates an incomplete fit or underestimated error variance;

$\chi^2_{red} < 1$ indicates an over-fit model (not enough data, or the model somehow is fitting the noise).

Q4 Add a function to your program to calculate χ^2_{red}. What values were computed? How would you interpret your results?

Resistors for electrical circuits are designed to be very linear and reliable, so the three measurements for resistance will likely be very close. The lines may even be indistinguishable on the plot.

Nonlinear fitting methods

Nonlinear circuits

We continue with the curve fitting from the previous exercises, but extend it to power law models. A lightbulb demonstrates various power laws; resistance and radioactive power are both dependent on temperature for blackbodies. In this lab, we will examine these relations with the voltage-current curve for a lightbulb.

Background knowledge for Part II

- **Python**: lists, arrays, numpy, scipy, pyplot, `curve_fit()`
- **Error Analysis** Chi squared (χ^2), goodness of fit
- **Physics**: Power Law. You may need to review appropriate section of your textbooks to analyze and interpret your data.

Introduction

An incandescent lightbulb works by heating a tungsten filament until it glows. All matter emits electromagnetic radiation, called thermal radiation, when it has a temperature above absolute zero. The simple theoretical material, a blackbody, is one that absorbs all radiation and follows a strict power law:

$$P = A\sigma T^4, \tag{5}$$

where σ is the Stefan-Boltzmann constant (5.670373×10^{-8} W m^{-2}K^{-4}).

Real materials are not ideal blackbodies. The relation for these "grey bodies" adds an emissivity, $\epsilon < 1$,

$$P = A\sigma\epsilon T^4, \tag{6}$$

The emissivity itself is typically not constant, and depends on temperature through a power law. For tungsten this relation is

$$\epsilon(T) = 1.731 \times 10^{-3} T^{0.663} \tag{7}$$

Including the emissivity leads to a power law between *power* and *temperature*. There is also a power law for the resistance of the bulb. In the simplest case, it is treated as linear ($R \propto T$), but for tungsten it is more accurately,

$$R(T) \propto T^{1.209} \tag{8}$$

Combining the dependencies on temperature with $P = VI$ and $V = IR$, a power law can be found between current and voltage. For an ideal black body with a linear relation between resistance and temperature,

$$I \propto V^{\frac{3}{5}} \tag{9}$$

For most of this experiment, we are not concerned with the constant of proportionality. However, when comparing the theoretical curve to your data you might need to define a suitable value.

Analysis of power laws with logarithms

The tool for analyzing power laws is, again, logarithms. Start with the general power law,

$$y = ax^b, \tag{10}$$

and take the logarithm of both sides,

$$log(y) = blog(x) + log(a). \tag{11}$$

Thus, log(y) depends linearly on log(x), with slope b. We can view this linear relation on a log-log plot (logarithmic scales for both x and y).

Linear regression works again, with the same linear model function, except using $\log(x_i)$ and $\log(y_i)$ as the input data.

Note: The same downfalls of using transformations apply. Again, using `curve_fit()`, we can use a nonlinear function directly as our model function.

The experiment

We will observe the power law for blackbody radiation through a lightbulb's voltage-current graph. Set up the Ohm's Law experiment as you did in Part I, but with a lightbulb instead of a resistor, as shown in Figure 4.

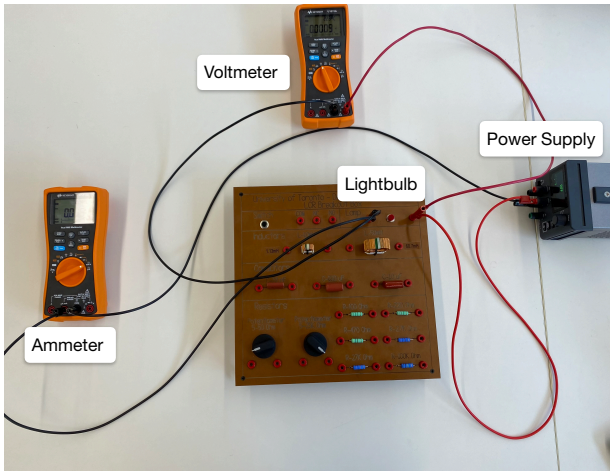


Figure 4: Visual representation of the experimental set up used in this part of the experiment.

After setting up the apparatus do the following:

1. Adjust the power supply voltage to its lowest value at which your lightbulb turns on..
2. Wait for the voltage and current to stabilize (it should not take long).
3. Record the voltage and current values.
4. Record the uncertainties in your measurements.
5. Repeat this process for at least 15 different values of voltage. Pay attention to the frequency with which you collect the data - if the dependent values change quickly, decrease the "step".
6. Save the values in a text file (.txt) and save it on an external memory device or online.

The Python program

You will use your programs to compare the transformation method and the non-linear least-squares method. The best parameters will be found with both methods, and used to plot best fit curves over the data.

The program should be organized as follows:

- Import the required functions and modules (`numpy`, `scipy.optimize.curve_fit()`, `matplotlib.pyplot`).
- Define the model functions(linearized: $f(x, a, b) = ax + b$,non–linear: $g(x, c, d) = cx^d$)
- Load the data and uncertainty measurements using `loadtxt()`.
- Perform the linear regression on (`log(xi)`,`log(yi)`) using $f(x, a, b)$ as the model function.
- Perform the nonlinear regression on (`xi`,`yi`) using $g(x, c, d)$ as the model function.
- Output both of the power law relations you calculated.
- Plot the error bars, both curves of best fit, and the theoretical curve. Include a legend for the different curves.
- Plot all the same things on a log–log plot. One way of doing this is with the `pylab.loglog()` function, another way is to call `pylab.yscale('log')` and `pylab.xscale('log')` after you make the plot.

The a and b parameters in the two model functions represent different parameters in the original, un-transformed, model function. Make sure you can convert between the functions correctly to compare the parameters accurately.

Write the program, and run it using the data gathered in the experiment. Save all plots and the parameters you determined.

> **Q5** Which regression method gave an exponent closer to the expected value? Can you see the difference on the plots?

Analyzing the quality of the fit

As covered in previous exercises, there are two ways that we can assess the quality of the fit of our model: variance of the calculated parameters, and the reduced chi-squared statistic.

The variance of the parameters is returned by `curve_fit()` as the diagonal entries in the covariance matrix, `p_cov`. Recall that the uncertainty in measurements is understood as the standard deviation,

$$a = \bar{a} \pm \sigma_a, \tag{12}$$

where $\sigma_a = \sqrt{Var(a)}$ In the python program, the variance of the first parameter in your model function is given in `p_cov[0, 0]`, the variance of the second parameter in `p_cov[1, 1]`, and so on.

Modify your program from the Python program section to calculate the standard deviation of the parameters.

> **Q6** What values did you find? Does the value of your fitted exponent fall within the range of the blackbody values, $\frac{3}{5}$, with your calculated standard deviation? What about in comparison to the expected value for tungsten?

Reduced chi-squared

Recall that the χ^2 distribution gives a measure of how unlikely a measurement is. The more a model deviates from the measurements, the higher the value of χ^2. But if χ^2 is too small, it's also an indication of a problem: usually that there were not enough samples. Add a function to your program to calculate χ^2_{red}.

> **Q7** What values were computed? How would you interpret your results?

Your submission for this exercise should include:

- All sets of data, clearly labeled, presented in an appropriate manner (tables/figures).
- All uncertainties, uncertainty budgets, sample uncertainty calculations
- Important information about the set up (models of the multimeters, band colours of the resistor used, any other details that might be relevant).
- Answers to all the questions, weaved throughout the report.
- All relevant current vs voltage plots with sufficient captions
- All obtained values of the resistance with uncertainties
- All obtained values for power fit parameters with uncertainties
- Plots of residuals
- The final version of your Python code (included separately as a .py file)
- Analysis and Discussion of your results.