PyLab - Ohm and Power laws

PHY224 Lab 1

Fredrik Dahl Bråten, Pankaj Patil

October 3, 2021

Contents

List of Figures

List of Tables

1 Exercise 1: Introduction to fitting methods

1.1 Abstract

Ohm's law explains the relationship between current flowing through a resistor and voltage applied across it. As per Ohm's law, the current flowing through a resistor is directly proportional to the voltage applied to its ends. So we have,

$$V = IR$$

Where,

V = Voltage

I = Current

R = Constant of Proportionality (Resistance)

In this exercise, using Python modules for data analysis, namely numpy, scipy, we verify the Ohm's law. Voltage and Current values are recorded for a known resistor and for a Potentiometer using digital multimeters. These recorded values are then used to arrive at the linear relation and obtain the values of the proportionality constants between the voltages and currents.

The quality of linear fit is obtained by calculating the χ^2_{red} for the recorded measurements for the resistor and Potentiometer.

The visual analysis of the data is done by plotting the measured data with uncertainty values, along with the linear model fit graphically, using matplotlib.pyplot Python module.

1.2 Introduction

In this exercise we analyze the relationship between current through a circuit element (Resistor or Potentiometer), and the voltage applied across that element. With this analysis we establish the Ohm's law which predicts the value of Current, given the value of Resistance and the Voltage across the resistor.

The relation we want to verify is

$$I = \frac{V}{R}$$

Here we take V, the voltage, as independent variable, and I, the current as dependent variable. Using the measured data we use the linear model function

$$f(x) = ax + b$$

to fit the data.

In above expression x is the independent variable which we measure as Voltage, and $f(x)$ is the dependent variable, which is the Current.

1.3 Methods, Materials and Experimental Procedure

We followed the method described in the lab manual [1] to setup the circuit. Setup as shown in circuit diagram in Figure 1, was used to do the measurements.

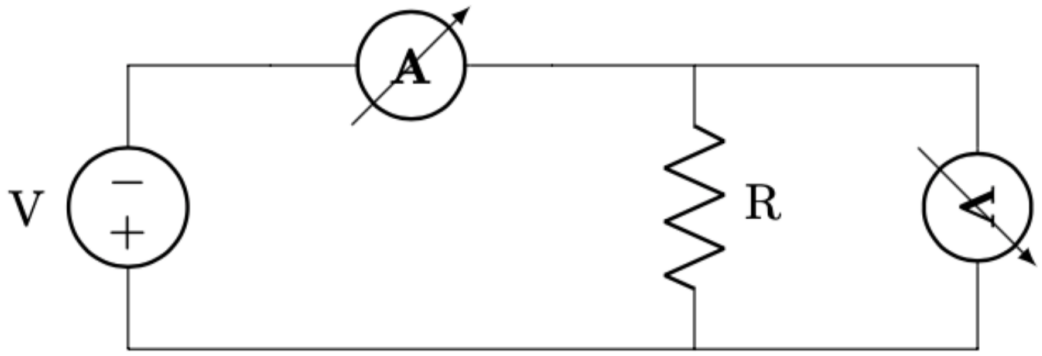


Figure 1: Electrical Circuit Setup

Measurements were taken for $100k\Omega$ resistor and for a Potentiometer. Reference values for 100 $k\Omega$ resistor and Potentiometer were recorded as shown in Table 1. The reference value for $100k\Omega$ resistor was computed using the color codes, and that for the Potentiometer was recorded using the multimeter.

Resistor	Reference Resistance
100 $k\Omega$	98.8 ± 0.5 $k\Omega$
Potentiometer	15.64 ± 0.08 $k\Omega$

Table 1: Reference Resistance Values

For the Potentiometer, we accidentally changed the resistance before noting down the reference value. We chose to discard the readings taken, and took different set of measurements along with reference resistance without disturbing the setup the second time.

The uncertainty computation is done using the multimeter accuracy error (0.75%) and the error in precision, which is the fluctuation in last digit of the multimeter (0.01 mA). To account for the other external factors as described in the lab manual section 4.1 [1], we assume additional 2% error in our measurements.

1.4 Results

Following the experimental procedures described in previous section, data for Voltage and Current values was recorded for 100 $k\Omega$ resistor and Potentiometer in Table 2, and Table 3 respectively.

The plots of measured current versus voltage for 100 $k\Omega$ and Potentiometer are shown in Figure 2 and Figure 3 respectively.

In both the plots, the linear fit for measured data is drawn using the predicted value of currents by the linear model parameters obtained by scipy.optimize.curve_fit function.

For 100 $k\Omega$ resistor, we get the optimal parameters for model $f(x) = ax + b$ as

$$[a, b] = [0.0102\ 0.0001] \pm [0.0004\ 0.0064]$$

We compute the resistance,

$$R = \frac{1}{a} \pm \Delta a = 98.1500 \pm 0.0004\ k\Omega$$

For Potentiometer, we get the optimal parameters for model $f(x) = ax + b$ as

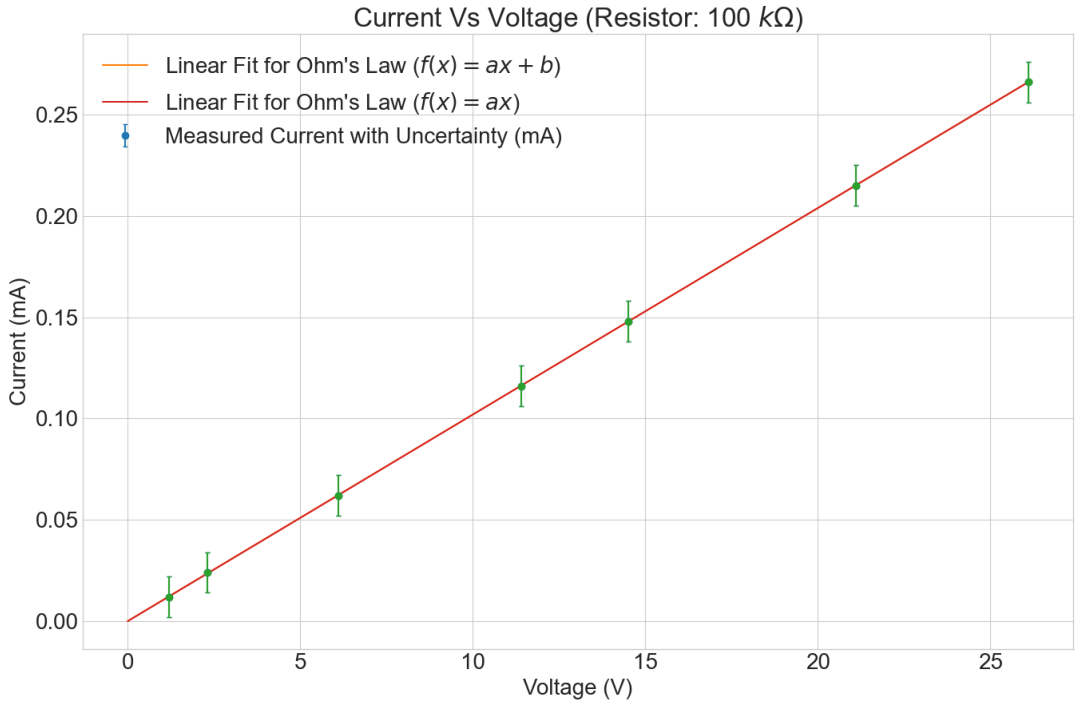
$$[a, b] = [0.0638\ 0.0020] \pm [0.0011\ 0.0077]$$

We compute the resistance as,

$$R = \frac{1}{a} \pm \Delta a = 15.6634 \pm 0.0011\ k\Omega$$

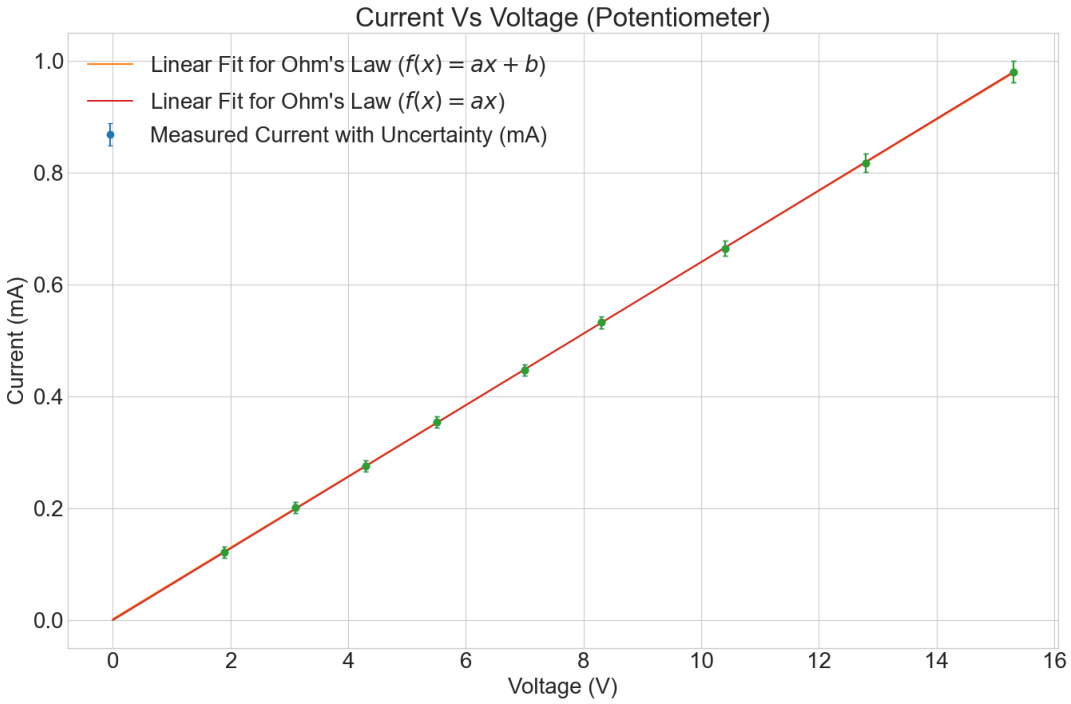
The uncertainty in resistance R is given by quadrature formula

$$R = \frac{1}{a} \implies \Delta R = \Delta a$$



$f(x) = (0.0102 \pm 0.0004)x + (0.0001 \pm 0.0064), \quad \chi^2_{red} = 0.0009, \quad R = \frac{1}{a} \pm \Delta a = 98.1499 \pm 0.0004\ k\Omega$
$f(x) = (0.0102 \pm 0.0003)x, \quad \chi^2_{red} = 0.0008, \quad R = \frac{1}{a} \pm \Delta a = 98.1189 \pm 0.0003\ k\Omega$

Figure 2: 100 $k\Omega$ Data Curve Fitting



$$f(x) = (0.0638 \pm 0.0011)x + (0.0020 \pm 0.0011), \quad \chi^2_{red} = 0.0201, \quad R = \frac{1}{a} \pm \Delta a = 15.660 \pm 0.0011 \ k\Omega$$
$$f(x) = (0.0641 \pm 0.0005)x, \quad \chi^2_{red} = 0.0257, \quad R = \frac{1}{a} \pm \Delta a = 15.660 \pm 0.0005 \ k\Omega$$

Figure 3: Potentiometer Data Curve Fitting

1.5 Discussion

In this exercise, the data sets obtained using the measurements were used as is without discarding any data points. As was suggested in the manual the data analysis was done for linear models, $f(x) = ax + b$ and $f(x) = ax$.

The uncertainty calculations are based on error in multimeter accuracy (0.75%) provided by the manufacturer and error in precision of the measurement obtained. The error in precision was taken to be fluctuations in last digit of each measurement, which we found to be consistent at 0.01 mA. To account for various other factors contributing to error in the measurement we added 2% correction factor to the error estimate. The final uncertainty was taken as higher of the three uncertainty values.

The readings were then fit to linear model using the scipy.optimize.curve_fit function. In both the cases, namely 100 $k\Omega$ resistor and Potentiometer, the data is plotted along with the model predictions, as shown in Figure 2 and Figure 3. We note that for both models and both circuit elements, the predicted graph passes through the data points error bars nicely. The linear models are more than good fit is evident from the fact that in all cases we get low values of χ^2_{red}.

When data is fit to $f(x) = ax + b$, we note that the y-intercept of the line is not zero, although

it is a really low value for all the cases. Hence for this model the predicted graph does not pass through point (0,0). The reason for non-zero y-intercept can be attributed to the uncertainties in the measurements. When we force the linear model $f(x) = ax$, the χ^2_{red} values are little higher, for example, in Potentiometer case, it is 0.0257 for $f(x) = ax$ against 0.0201 for $f(x) = ax + b$. But in both the cases it is less than 1, giving more than a good fit.

The graph lines for both the models are indistinguishable on the plots, in both 100 $k\Omega$ and Potentiometer cases. The uncertainty in the measurement of resistance in $f(x) = ax$ is slightly less than the same obtained with $f(x) = ax + b$.

Using both the models the resistance values match with the measured values recorded in Table 1, within uncertainties of measurements, really well.

The low values of χ^2_{red} indicate that we have an over-fit model and we needed more readings to do the data analysis. But the resistors for electrical circuits are designed to be highly linear and reliable [1], hence we expect to get low values for χ^2_{red}.

1.6 Conclusions

In this exercise, we successfully demonstrated the linear relationship between the Voltage across a resistor and the current flowing through it. Using the Python data analysis modules like numpy and scipy we were able to obtain optimal parameters for linear model function. The optimal parameters matched within the uncertainty to the reference values of the resistance we recorded for 100 $k\Omega$ resistor and Potentiometer. The low values of χ^2_{red} in both the cases indicate that we have over-fit model. But as noted in the lab manual [1], resistors for electrical circuits are designed to be very linear and reliable, hence we expect low values of χ^2_{red} for measured data.

2 Exercise 3: Nonlinear fitting methods II

2.1 Abstract

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. In this report we present how we performed an experiment to compare the blackbody power law: $I \propto V^{\frac{3}{5}}$ and the power law for a tungsten wire: $I \propto V^{0.5882}$, to the model power laws we estimated from our data: measuring voltage and current in a circuit with a lightbulb. These theoretical power laws are graphically compared with our model power laws. Furthermore, to determine the quality of our models, we calculate the χ^2_{red} values of each model. We also confirmed that the theoretical voltage exponent values fell within the uncertainty range of our estimated model exponent values. This analysis was done in Python with use of the numpy, scipy and matplotlib modules.

2.2 Introduction

In this exercise we will model the relationship between voltage and current flowing through a circuit with a lightbulb. The corresponding theoretical power law for a radiating blackbody is: $I \propto V^{\frac{3}{5}}$, while for a tungsten wire, it is approximately: $I \propto V^{0.5882}$. Here I is the measured current, and V is voltage. In our experiment, V is the independent variable, and I is the dependent variable.

We will use two models to approximate the power law from our data: $I = a * V^b$ will be our non-linear model, and: $\ln(I) = a * \ln(V) + b$ the linear model for which we will fit to our data. a and b are parameters to be adjusted for a best fit, while V and I are our measured values of voltage and current. ln is the natural logarithm.

2.3 Methods, Materials and Experimental Procedure

We successfully followed the procedures as described in the exercise3.pdf document [2].

2.4 Results

Below in Figure 4 and 5, we see our data from the experiment plotted as points with corresponding error bars. Furthermore, we see the theoretical curves as described in the introduction, along with the curves corresponding to our two models best fitted to our data. The two models are so similar that it is hard to distinguish them in the plots below.

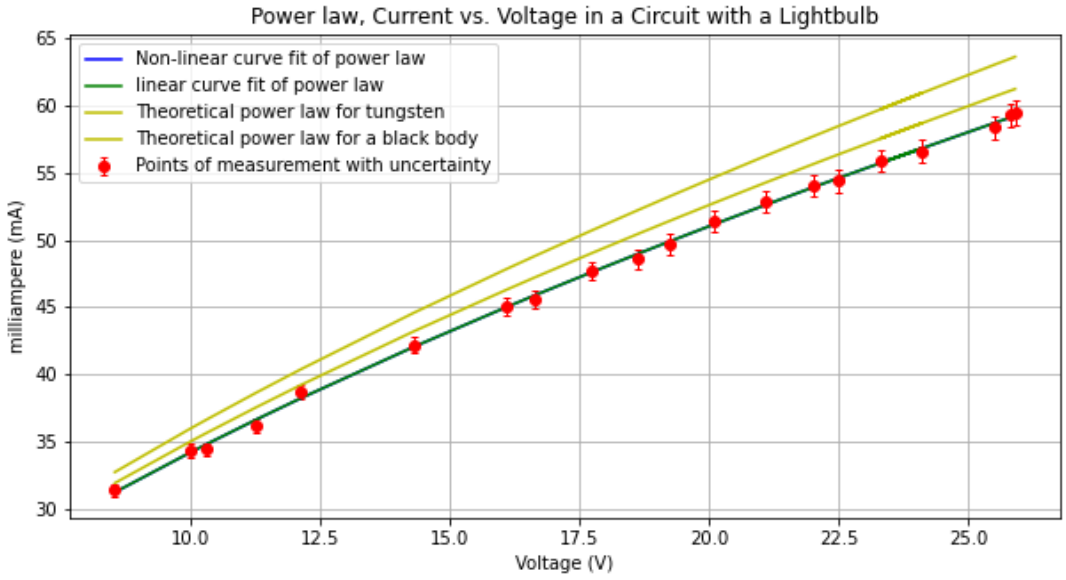


Figure 4: Points of measurement with corresponding error bars, theoretical power laws for tungsten and a blackbody, along with our two power law models best fitted to our data points.

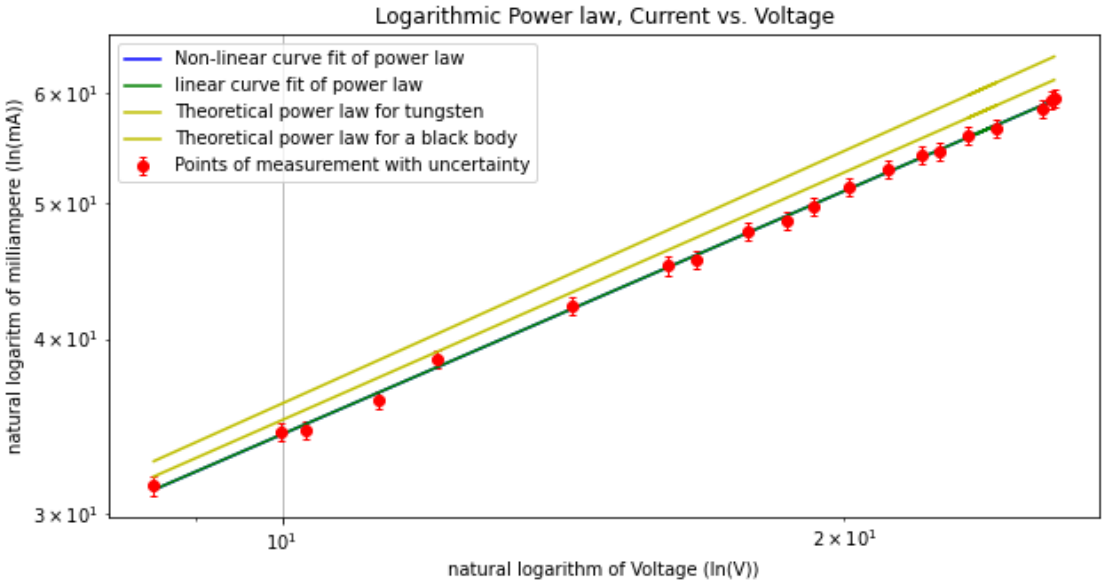


Figure 5: Equal to Figure 4, though with logarithmic axes.

The optimal parameters with variance, estimated by the scipy optimize curve fit function, are:

$[a, b] = [0.57787573\ 2.20117518] \pm [0.00989301\ 0.02841001]$ for the linear model, and:

$[a, b] = [9.03211882\ 0.57799121] \pm [0.25674234\ 0.00989844]$ for the nonlinear model.

The χ^2_{red} values for the nonlinear and linear model respectively are 0.19 and 0.19.

The power laws we found between current and voltage, was, with our non-linear model:

$I(V) = 9.0321 * V^{0.5780}$

For our linear model, we found:

$2.2012 * \exp(0.5779 * V)$,

Where V is the measured voltage (in volts) and I is current (in milliampere).

2.5 Discussion

The uncertainties of the measured current, our dependent variable, were initially estimated by picking the maximum uncertainty caused by accuracy and precision. The uncertainty of accuracy was caused by the ammeter we used in our experiment. The manufacturer of the ammeter writes in its manual that there is an estimated measurement uncertainty of 0.75% of the value measured by the ammeter. The uncertainty of precision was caused by variation over time of the last digit of the measured current of the ammeter. The last digit in our experiment was the first decimal place with milliampere units, so we estimated an uncertainty due to precision of 0.1 milliampere. Our initial estimates of the uncertainty, corresponding to each data point in our experiment, was the maximum of the uncertainty due to precision, and accuracy.

However, with these uncertainties our model curves did not pass through the error bars of each data point in the experiment. To solve this problem, we had to multiply the uncertainty by a factor of 2. The result is shown in Figure 4 and 5 in the Results section.

The χ^2_{red} values calculated for each model, see the Results section, are acceptable values. Thus, the spread (Euclidian distance) between our data points and our two model curves are low. This is good. However, the χ^2_{red} values are perhaps a bit too small. In our next experiment we will consider gathering more data to increase the χ^2_{red} values of our models. We were taught in our labs that a χ^2_{red} value should ideally be approximately between 1 and 10, in experiments such as our own.

As seen in Figure 4 and 5, both models produce approximately the same curve, which fit our data well. However, the theoretical curves describing the power law, current vs. Voltage, for a tungsten lightbulb and a blackbody, both deviate somewhat from our data points and curves. We have programmed the theoretical curves to only differ from the non-linear model curve, by the value of the Voltage exponent.

The exponents of our two models are very close to the theoretical exponents. By the theoretical power laws, with tungsten and blackbody respectively, current is proportional to voltage to the power of 0.5882 and $\frac{3}{5} = 0.6$. While the nonlinear model approximates an exponent of 0.5778 ± 0.51, while the linear model also approximates an exponent of 0.5779 ± 0.17. These uncertainties are calculated as the square root of the parameter variances of our models returned by the scipy curve fit function. Thus, the values of our fitted exponent fall within the range of the blackbody values ($\frac{3}{5} = 0.6$) and the expected value for tungsten (0.5882), with our calculated standard deviation.

These small deviations in model exponents, in comparison to theory exponents, causes the theoretical curves to deviate from our model curves.

2.6 Conclusions

In this exercise we estimated the power law of a lightbulb in a circuit by measuring current and voltage with uncertainties. We successfully followed the instructions for the experiment written in the exercise3.pdf document without issues. Our results show that both our models return approximately the same estimated power law. However, this power law deviates somewhat from the theoretical power laws for a radiating blackbody and a tungsten wire. We have plotted our data, our model and theoretical curves, and calculated each models χ^2_{red} values.

3 Appendix

3.1 Data Tables for Exercise 1

Voltage (V)	Current (mA)
1.2	0.012
2.3	0.024
6.1	0.062
11.4	0.116
14.5	0.148
21.1	0.215
26.1	0.266

Table 2: Readings of Voltage and Current for 100 $k\Omega$ Resistor

Voltage (V)	Current (mA)
1.9	0.122
3.1	0.202
4.3	0.276
5.5	0.354
7	0.447
8.3	0.533
10.4	0.665
12.8	0.818
15.3	0.981

Table 3: Readings of Voltage and Current for Potentiometer

3.2 Data Table for Exercise 3

Voltage (V)	Current (mA)
8.54	31.4
9.99	34.3
10.31	34.4
11.27	36.2
12.12	38.7
14.30	42.2
16.08	45.1
16.65	45.6
17.75	47.7
18.62	48.6
19.24	49.7
20.1	51.4
21.1	52.9
22.0	54.1
22.5	54.4
24.1	56.6
23.3	55.9
25.5	58.4
25.8	59.3
25.9	59.5

Table 4: Readings of Voltage and Current for Exercise 3

3.3 Python Code: Exercise 1

The Python code for this exercise is divided into two files. statslab.py file contains utility methods which we will be frequently using in this course. lab_1_ex_1_code.py file contains the code which analyzes the data for 100 $k\Omega$ resistor and Potentiometer.

3.3.1 statslab.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: Pankaj Patil
"""

import numpy as np
import scipy.optimize as optim
import matplotlib.pyplot as plt

# This file contains various utility methods to do data analysis, curve
# fitting and plotting

# use bigger font size for plots
plt.rcParams.update({'font.size': 20})

def chi2(y_measure,y_predict,errors):
    """Calculate the chi squared value given a measurement with errors and
    prediction"""
    return np.sum( np.power(y_measure - y_predict, 2) / np.power(errors, 2) )

def chi2reduced(y_measure, y_predict, errors, number_of_parameters):
    """Calculate the reduced chi squared value given a measurement with errors
    and prediction, and knowing the number of parameters in the model."""
    return chi2(y_measure, y_predict, errors)/ \
            (y_measure.size - number_of_parameters)

def read_data(filename, skiprows=1, usecols=(0,1)):
    """Load give\n file as csv with given parameters,
    returns the unpacked values"""
    return np.loadtxt(filename,
                      skiprows=skiprows,
                      usecols=usecols,
                      delimiter=",",
                      unpack=True)
```

```
def fit_data(model_func, xdata, ydata, yerrors, guess=None):
    """Utility function to call curve_fit given x and y data with errors"""
    popt, pcov = optim.curve_fit(model_func,
                                 xdata,
                                 ydata,
                                 absolute_sigma=True,
                                 sigma=yerrors,
                                 p0=guess)

    pstd = np.sqrt(np.diag(pcov))
    return popt, pstd

# y = ax+b
def linear_regression(xdata, ydata):
    """Simple linear regression model"""
    x_bar = np.average(xdata)
    y_bar = np.average(ydata)
    a_hat = np.sum( (xdata - x_bar) * (ydata - y_bar) ) / \
            np.sum( np.power((xdata - x_bar), 2) )
    b_hat = y_bar - a_hat  * x_bar
    return a_hat, b_hat

class plot_details:
    """Utility class to store information about plots"""
    def __init__(self, title):
        self.title = title
    def set_errorbar_legend(self, v):
        self.errorbar_legend = v
    def fitted_curve_legend(self, v):
        self.fitted_curve_legend = v
    def x_axis_label(self, v):
        self.x_axis_label = v
    def y_axis_label(self, v):
        self.y_axis_label = v
    def xdata(self, x):
        self.xdata = x
    def ydata(self, y):
        self.ydata = y
    def yerrors(self, y):
        self.yerrors = y
    def xdata_for_prediction(self, x):
        self.xdata_for_prediction = x
    def ydata_predicted(self, y):
        self.ydata_predicted = y
    def legend_position(self, p):
        self.legend_loc = p
```

```
    def chi2_reduced(self, c):
        self.chi2_reduced = c

def plot(plot_details, new_figure=True, error_plot=True):
    """Utility method to plot errorbar and line chart together,
    with given arguments"""
    if new_figure:
        fig  = plt.figure(figsize=(16, 10))
        fig.tight_layout()
        plt.style.use("seaborn-whitegrid")

    # plot the error bar chart
    if error_plot:
        label = None
        if hasattr(plot_details, "errorbar_legend"):
            label = plot_details.errorbar_legend

        plt.errorbar(plot_details.xdata,
                     plot_details.ydata,
                     yerr=plot_details.yerrors,
                     marker="o",
                     label=label,
                     capsize=2,
                     ls="")

    # plot the fitted curve
    plt.plot(plot_details.xdata_for_prediction,
             plot_details.ydata_predicted,
             label=plot_details.fitted_curve_legend)

    # legend and title
    plt.title(plot_details.title)
    plt.xlabel(plot_details.x_axis_label)
    plt.ylabel(plot_details.y_axis_label)

    legend_pos = "upper left"
    if hasattr(plot_details, "legend_loc"):
        legend_pos = plot_details.legend_loc

    plt.legend(loc=legend_pos)
```

3.3.2 lab_1_ex_1_code.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: Pankaj Patil
"""

import numpy as np
import matplotlib.pyplot as plt

# import our utility method library from statslab.py
import statslab as utils

# units for the measured quantities
units = {
    "voltage": "V",
    "current": "mA",
    "resistance": "kiloohm"
}

# assume 2% uncertinity due to connection errors, human factors etc.
uncertainty_correction = 0.02

# compute the uncertainty in current values
def current_uncertainty(current):
    """return the uncertainty in current for given values of current"""
    error_accuracy = 0.0075 * current
    error_precision = 0.01

    # multimeter unceertainty is maxmimum of the above two
    multimeter_uncertainty = max(error_accuracy, error_precision)

    # we additionaly compute uncertainty using 2% uncertainly due to
    # connections and human factors
    return max(multimeter_uncertainty, uncertainty_correction*current)

#  linear model function f(x) = ax + b
def linear_model_function_1(x, a, b):
    return a*x + b

#  linear model function f(x) = ax
def linear_model_function_2(x, a):
    return a*x
```

```
# analyse the data file with linear model function 1, f(x) = ax + b
def analyse_file_model_1(filename, title):
    print("  Linear Model := f(x) = ax + b")

    # read the data from the data file, and collect measured  voltages and
    # currents
    measured_voltages, measured_currents = utils.read_data(filename,
                                                            usecols=(0,1))

    # create error array for the current. use np.vectorize to make the
    # uncertainty function operate on arrays.
    current_errors = np.vectorize(current_uncertainty)(measured_currents)

    # fit the measured data using curve_fit function
    popt, pstd = utils.fit_data(linear_model_function_1,
                          measured_voltages,
                          measured_currents,
                          current_errors)

    # generate data for predicted values using estimated resistance
    # obtained using curve fit model
    voltage_data = np.linspace(0, measured_voltages[-1], 100)
    predicted_currents = linear_model_function_1(voltage_data,
                                                 popt[0],
                                                 popt[1])

    # calculate the chi2reduced for curve fitted model
    chi2r_curve_fit = utils.chi2reduced(measured_currents,
                                  linear_model_function_1(measured_voltages,
                                                          popt[0],
                                                          popt[1]),
                                  current_errors,
                                  2)

    # fill in the plot details for curve fitted
    # model in our plot_details object
    plot_data = utils.plot_details("Current Vs Voltage (%s)" % title)
    plot_data.set_errorbar_legend("Measured Current with Uncertainty (%s)" % units["current
    plot_data.fitted_curve_legend("Linear Fit for Ohm's Law ($f(x) = ax + b$)")
    plot_data.x_axis_label("Voltage (%s)" % units["voltage"])
    plot_data.y_axis_label("Current (%s)" % units["current"])
    plot_data.xdata(measured_voltages)
    plot_data.ydata(measured_currents)
    plot_data.yerrors(current_errors)
    plot_data.xdata_for_prediction(voltage_data)
    plot_data.ydata_predicted(predicted_currents)
    plot_data.legend_position("upper left")
```

```
    plot_data.chi2_reduced(chi2r_curve_fit)

    # plot the data
    utils.plot(plot_data)

    print("    Linear model slope (a) = %.4f" % popt[0])
    print("    Linear model y-intercept (b) := %.4f" % popt[1])
    print("    Linear model slope (a) uncertainty := +/- %.4f" % pstd[0])
    print("    Linear model y-intercept (b) uncertainty := +/- %.4f" % pstd[1])
    print("    Estimated Resistance (1/a) = %.4f %s" % (1/popt[0],
                                                  units["resistance"]))

    print("    Uncertainty in resistance  := +/- %.4f %s" % (pstd[0],
                                                   units["resistance"]))
    print("    chi2reduced (Curve Fit) := %.4f" % chi2r_curve_fit)


# analyse the data file with linear model function 2, f(x) = ax
def analyse_file_model_2(filename, title):
    print("  Linear Model := f(x) = ax")

    # read the data from the data file, and collect measured  voltages and
    # currents
    measured_voltages, measured_currents = utils.read_data(filename,
                                                        usecols=(0,1))

    # create error array for the current. use np.vectorize to make the
    # uncertainty function operate on arrays.
    current_errors = np.vectorize(current_uncertainty)(measured_currents)

    # fit the measured data using curve_fit function
    popt, pstd = utils.fit_data(linear_model_function_2,
                          measured_voltages,
                          measured_currents,
                          current_errors)

    # generate data for predicted values using estimated resistance
    # obtained using curve fit model
    voltage_data = np.linspace(0, measured_voltages[-1], 100)
    predicted_currents = linear_model_function_2(voltage_data,
                                             popt[0])

    # calculate the chi2reduced for curve fitted model
    chi2r_curve_fit = utils.chi2reduced(measured_currents,
                                linear_model_function_2(measured_voltages,
                                                        popt[0]),
                                current_errors,
```

1)

```
    # fill in the plot details for curve fitted
    # model in our plot_details object
    plot_data = utils.plot_details("Current Vs Voltage (%s)" % title)
    plot_data.fitted_curve_legend("Linear Fit for Ohm's Law ($f(x) = ax$)")
    plot_data.x_axis_label("Voltage (%s)" % units["voltage"])
    plot_data.y_axis_label("Current (%s)" % units["current"])
    plot_data.xdata(measured_voltages)
    plot_data.ydata(measured_currents)
    plot_data.yerrors(current_errors)
    plot_data.xdata_for_prediction(voltage_data)
    plot_data.ydata_predicted(predicted_currents)
    plot_data.legend_position("upper left")
    plot_data.chi2_reduced(chi2r_curve_fit)

    # plot the data
    utils.plot(plot_data, new_figure=False)

    print("    Linear model slope (a) = %.4f" % popt[0])
    print("    Linear model slope (a) uncertainty := +/- %.4f" % pstd[0])
    print("    Estimated Resistance (1/a) = %.4f %s" % (1/popt[0],
                                                        units["resistance"]))

    print("    Uncertainty in resistance  := +/- %.4f %s" % (pstd[0],
                                                         units["resistance"]))
    print("    chi2reduced (Curve Fit) := %.4f" % chi2r_curve_fit)

    plt.savefig("lab_1_ex_1_plot_%s.png" % filename[:-4].lower())


# files to analyse
file_titles ={
    "100k.csv": {
        "title": "Resistor: 100 $k\Omega$"
    },
    "Potentiometer.csv": {
        "title": "Potentiometer"
    }
}

for filename, data in file_titles.items():
    print(filename)
    analyse_file_model_1(filename, data["title"])
    analyse_file_model_2(filename, data["title"])
```

3.3.3 Program Output

```
100k.csv
  Linear Model := f(x) = ax + b
    Linear model slope (a) = 0.0102
    Linear model y-intercept (b) := 0.0001
    Linear model slope (a) uncertainty := +/- 0.0004
    Linear model y-intercept (b) uncertainty := +/- 0.0064
    Estimated Resistance (1/a) = 98.1499 kiloohm
    Uncertainty in resistance  := +/- 0.0004 kiloohm
    chi2reduced (Curve Fit) := 0.0009
  Linear Model := f(x) = ax
    Linear model slope (a) = 0.0102
    Linear model slope (a) uncertainty := +/- 0.0003
    Estimated Resistance (1/a) = 98.1189 kiloohm
    Uncertainty in resistance  := +/- 0.0003 kiloohm
    chi2reduced (Curve Fit) := 0.0008
Potentiometer.csv
  Linear Model := f(x) = ax + b
    Linear model slope (a) = 0.0638
    Linear model y-intercept (b) := 0.0020
    Linear model slope (a) uncertainty := +/- 0.0011
    Linear model y-intercept (b) uncertainty := +/- 0.0077
    Estimated Resistance (1/a) = 15.6634 kiloohm
    Uncertainty in resistance  := +/- 0.0011 kiloohm
    chi2reduced (Curve Fit) := 0.0201
  Linear Model := f(x) = ax
    Linear model slope (a) = 0.0641
    Linear model slope (a) uncertainty := +/- 0.0005
    Estimated Resistance (1/a) = 15.6041 kiloohm
    Uncertainty in resistance  := +/- 0.0005 kiloohm
    chi2reduced (Curve Fit) := 0.0257
```

3.4 Python Code: Exercise 3

The Python code for this exercise is divided into two files. Functions.py file contains utility methods which we will be frequently using in this course. exercise_3.py file contains the code which analyzes the data for this experiment.

3.4.1 Functions.py

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 30 09:46:36 2021

@author: Fredrik
"""
import numpy as np
import scipy.optimize as optim
import matplotlib.pyplot as plt

#Defining the function for curve fitting and plotting
def curve_fit_and_plot(model,initial_guess,xdata,ydata,y_uncer,xunit,yunit,
                       plot_title):
    """
    This function uses the scipy curve_fit function to estimate the parameters
    of the model which will minimize the euclidian distance between our data
    points, and the model curve.
    We print these optimal model parameters along with their uncertainty, and
    plot the original data with error bars, along with the curve fit model.

    Parameters
    ----------
    model : function to be used as model
        model(x,a,b,c,...), where we are estimating a,b,c, etc.
    initial_guess : list of guesses for the parameters a,b,c, etc. eg. [2,4,254]
    xdata : list of input points for the model, eg. [2,4,5,7,9,28]
    ydata : list of output points for the model, eg [23,25,26,85,95,104]
    y_uncer : list of uncertaintes associated with the ydata, which the model
        shall output
    xunit : String describing the unit along the x-axis for label when plotting.
        eg. 'Voltage (V)'
    yunit : String describing the unit along the y-axis for label when plotting.
        eg. 'Current (A)'
    plot_title : String describing the title of the plot.
        eg. 'Current vs. Voltage'
```

```
    Returns None
    """
    #Using the scipy curve fit function to find our model parameters
    p_opt , p_cov = optim.curve_fit(model , xdata , ydata, p0 = initial_guess,
                                    sigma = y_uncer, absolute_sigma = True )
    p_std = np.sqrt( np.diag ( p_cov ))

    print("The optimal values for our curve fit model parameters, are:",np.round(p_opt,2))
    print("Their associated uncertainties  are:", np.round(p_std,2))

    #Now we create some data points on the model curve for plotting
    xvalues_for_plot = np.linspace(xdata[0],xdata[-1],1000)
    yvalues_for_plot = []
    for i in xvalues_for_plot:
        yvalues_for_plot.append(model(i,p_opt[0],p_opt[1]))

    #Now we plot the original data with error bars, along with the curve fit model
    plt.figure(figsize=(10,5))
    plt.errorbar(xdata,ydata,y_uncer,c='r', ls='', marker='o',lw=1,capsize=2,
                 label = 'Points of measurement with uncertainty')
    plt.plot(xvalues_for_plot,yvalues_for_plot, c='b',
             label = 'Scipy curve fit')
    plt.title(plot_title)
    plt.xlabel(xunit)
    plt.ylabel(yunit)
    plt.legend()
    plt.grid()
    plt.savefig(plot_title+'.png')
    plt.show()

    return None

def error_plot(model,p_opt,xdata,ydata,y_uncer,xunit,yunit,
                        plot_title):
    #Now we create some data points on the model curve for plotting
    xvalues_for_plot = np.linspace(xdata[0],xdata[-1],1000)
    yvalues_for_plot = []
    for i in xvalues_for_plot:
        yvalues_for_plot.append(model(i,p_opt[0],p_opt[1]))

    #Now we plot the original data with error bars, along with the curve fit model
    plt.figure(figsize=(10,5))
    plt.errorbar(xdata,ydata,y_uncer,c='r', ls='', marker='o',lw=1,capsize=2,
                 label = 'Points of measurement with uncertainty')
    plt.plot(xvalues_for_plot,yvalues_for_plot, c='b',
             label = 'Scipy curve fit')
```

```
    plt.title(plot_title)
    plt.xlabel(xunit)
    plt.ylabel(yunit)
    plt.legend()
    plt.grid()
    plt.savefig(plot_title+'.png')
    plt.show()
    return None

def chi2(y_measure,y_predict,errors):
    """Calculate the chi squared value given a measurement with errors and
    prediction"""
    return np.sum( np.power(y_measure - y_predict, 2) / np.power(errors, 2) )

def chi2reduced(y_measure, y_predict, errors, number_of_parameters):
    """Calculate the reduced chi squared value given a measurement with errors
    and prediction, and knowing the number of parameters in the model."""
    return chi2(y_measure, y_predict, errors)/ \
            (y_measure.size - number_of_parameters)

def read_data(filename, Del, skiprows, usecols=(0,1)):
    """Load give\n file as csv with given parameters,
    returns the unpacked values"""
    return np.loadtxt(filename,
                      skiprows=skiprows,
                      usecols=usecols,
                      delimiter=Del,
                      unpack=True)

def fit_data(model_func, xdata, ydata, yerrors, guess):
    """Utility function to call curve_fit given x and y data with errors"""
    popt, pcov = optim.curve_fit(model_func,
                                 xdata,
                                 ydata,
                                 absolute_sigma=True,
                                 sigma=yerrors,
                                 p0=guess)

    pstd = np.sqrt(np.diag(pcov))
    return popt, pstd
```

3.4.2 exercise_3.py

```
#Importing modules
import numpy as np
import matplotlib.pyplot as plt
import Functions as F

#Defining model function for position prediction
def linear_model(x,a,b):
    return a*x+b

def non_linear_model(x,a,b):
    return a*x**b

#Importing data from csv file
Voltage, Current = np.loadtxt('Data.csv',skiprows=1,
                                        delimiter=', ', unpack=True)

#To find the uncertainty of the current measurements, we choose the larger
#uncertainty of the precission (fluctuations of the last digit) and
#accuracy (instrument error) uncertainty.
def return_uncertainty(A):
    U_acc = 0.0075*A
    U_pre = 0.1

    if U_pre >= U_acc:
        U = U_pre
    else:
        U = U_acc
    return U * 2 #added a factor of 2 to our uncertainty

#Creating a list of uncertainties associated with each
#   y-value/current measurement.
Uncertainty = []
for i in Current:
    Uncertainty.append(return_uncertainty(i))


#Now we fit the parameters of the two models to our data:
popt_linear, pstd_linear = F.fit_data(linear_model,
                                      np.log(Voltage),
                                      np.log(Current),
                                      Uncertainty/Current,
                                      [0.57710934, 2.20315259])
```

```
popt_non_linear, pstd_non_linear = F.fit_data(non_linear_model,
                                      Voltage,
                                      Current,
                                      Uncertainty,
                                      [9.05001068, 0.57722491])
print("The estimated optimal parameters with uncertainty by scipy optimize",
      "curve fit are:",popt_linear, "+-",pstd_linear," for the linear, and:",
      popt_non_linear, "+-", pstd_non_linear, "for the non linear model.")

#Calculating predicted y-values of models:
Power_law_non_linear = np.zeros(len(Voltage))
Power_law_linear = np.zeros(len(Voltage))
#And now we also calculate the y-values predicted by the linear model,
# for the non logarithmic scale. I will use this later on in the plot
Power_law_linear_non_linear = np.zeros(len(Voltage))

for i in range(len(Voltage)):
    Power_law_non_linear[i] = popt_non_linear[0]*Voltage[i]**popt_non_linear[1]
    Power_law_linear[i]= popt_linear[0]*np.log(Voltage[i])+popt_linear[1]
    Power_law_linear_non_linear[i] = np.exp(Power_law_linear[i])

#The Chi squared values for these models are:
chi2_non_linear = F.chi2reduced(Current,Power_law_non_linear,
                          Uncertainty,2)
chi2_linear = F.chi2reduced(np.log(Current),Power_law_linear,
                          Uncertainty/Current,2)
print("\nThe reduced Chi squared values for the non linear and linear model",
      "respectivly",
      "are", np.round(chi2_non_linear,2), "and", np.round(chi2_linear,2))
print("These are good reduced Chi squared values. Perhaps a bit too small.",
      "Next time we may take even more samples to increase the reduced Chi",
      "Squared values. They should ideally be approximately between 1 and 10.")

#output both of the power law relations you calculated.

print("\nThe power law that we found between current and voltage,",
      "was, with our non-linear model: I(V)=", np.round(popt_non_linear[0],4),
      "* V ^", np.round(popt_non_linear[1],4), "And for our linear model: ",
      np.round(popt_linear[1],4),"* exp(", np.round(popt_linear[0],4),"* V )")



#Now we plot these models, the data, and the theoretical curve:
#Now we create some data points on the model curve for plotting

#We calculate the predicted count rates by the theory:
```

```
# Note, theoretical halflife is 2.6 min
Power_law_tungsten = np.zeros(len(Voltage))
Power_law_blackbody = np.zeros(len(Voltage))
for i in range(len(Voltage)):
    Power_law_tungsten[i] = popt_non_linear[0]*Voltage[i]**0.5882
    Power_law_blackbody[i] = popt_non_linear[0]*Voltage[i]**(3/5)

#Now we plot the original data with error bars, along with the curve fit model
plt.figure(figsize=(10,5))
plt.errorbar(Voltage, Current,Uncertainty ,c='r', ls='', marker='o',
             lw=1,capsize=2,
             label = 'Points of measurement with uncertainty')

plt.plot(Voltage,Power_law_non_linear, c='b',
         label = 'Non-linear curve fit of power law')
plt.plot(Voltage,Power_law_linear_non_linear, c='g',
         label = 'linear curve fit of power law')
plt.plot(Voltage,Power_law_tungsten, c='y',
         label = 'Theoretical power law for tungsten')
plt.plot(Voltage,Power_law_blackbody, c='y',
         label = 'Theoretical power law for a black body')

plt.title("Power law, Current vs. Voltage in a Circuit with a Lightbulb")
plt.xlabel("Voltage (V)")
plt.ylabel("milliampere (mA)")
plt.legend()
plt.grid()
plt.savefig("Power_law"+'.png')
plt.show()


#Now we plot the logarithmic version of this:
plt.figure(figsize=(10,5))
plt.errorbar(Voltage, Current,Uncertainty ,c='r', ls='', marker='o',
             lw=1,capsize=2,
             label = 'Points of measurement with uncertainty')

plt.plot(Voltage,Power_law_non_linear, c='b',
         label = 'Non-linear curve fit of power law')
plt.plot(Voltage,Power_law_linear_non_linear, c='g',
         label = 'linear curve fit of power law')
plt.plot(Voltage,Power_law_tungsten, c='y',
         label = 'Theoretical power law for tungsten')
plt.plot(Voltage,Power_law_blackbody, c='y',
         label = 'Theoretical power law for a black body')

plt.title("Logarithmic Power law, Current vs. Voltage")
```

```
plt.xlabel("natural logarithm of Voltage (ln(V))")
plt.ylabel("natural logaritm of milliampere (ln(mA))")
plt.legend()
plt.yscale('log')
plt.xscale('log')
plt.grid()
plt.savefig("Log_Power_law"+'.png')
plt.show()

print("\nWe had to increase our uncertainty by a factor of 2 to",
"make our curve fit graphs pass through the error bars of our",
"measurements.")

print("\nBoth models produce approximately the same curve, which fit our data",
"well. However, the theoretical curves describing the power law, current",
"vs. Voltage, for a tungsten lightbulb and a blackbody, deviate weakly,",
"but consistently",
"from our data points and curves. I have programmed the theoretical",
"curves to only differ from the non-linear model, by their exponent.")

print("\nThe exponents of our two models are very close to the theoretical",
"exponents. The theoretical power laws, with tungsten and blackbody",
"respectively, says that current is proportional",
"to voltage to the power of: 0.5882 and 3/5=0.6, while the nonlinear model",
"approximates an exponent of 0.5778 +-",
np.round(np.sqrt(pstd_non_linear[0]),2),
", while the linear model",
"also approximates an exponent of 0.5779 +-",
np.round(np.sqrt(pstd_linear[1]),2))

print("\nThus, the values of our fitted exponent fell within the range of the",
"blackbody values (3/5=0.6) and the expected value for tungsten (0.5882),",
"with our calculated standard deviation.")

print("\nThese small deviation causes the theoretical curves to",
"deviate from our model curves.")
```

References

[1] Lab Manual - PyLab - Ohm and Power laws - Exerciese 1.

[2] Lab Manual - PyLab - Ohm and Power laws - Exerciese 3.