PyLab - Ohm and Power laws

Fredrik Dahl Bråten, Pankaj Patil

September 24, 2021

1 Exercise 1: Introduction to fitting methods

1.1 Introduction

In this exercise, we fit the experimental data to a linear function [1].

1.2 Methods

We followed the method descibed in the lab manual to setup the circuit. Following steps were performed,

1. Connect the ameter, voltmeter, and power supply to the resistor. The setup of the circuit is shown in Figure 1.
2. Vary the voltage on the power supply.
3. Record the voltage and current from the multimeters, along with uncertainty.
4. Change the voltage, and record the new values. This step was repeated to record more data points.
5. Observations were recorded in csv files, separate file for 100 kiloohm and Potentiometer.
6. After performing all the above measurements, disconnected the power, and switch the voltmeter to resistance. The resistance value was noted for reference.

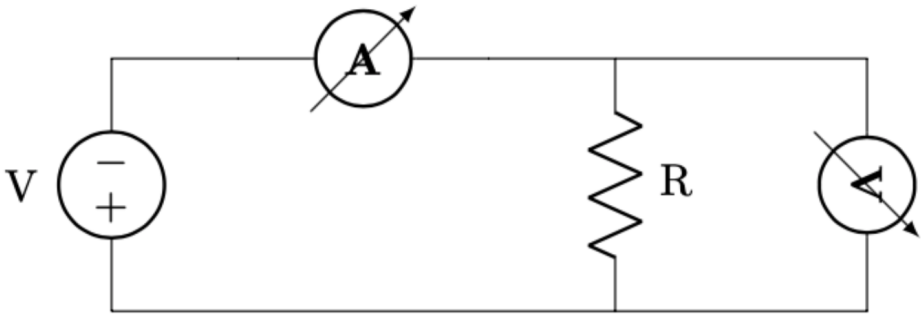

Figure 1: Electrical Circuit Setup

Folowing observations were made for the instruments,

1. One of the multimeter was faulty, and we discarded the readings takes from that multimeter.
2. We accidently changed the potentiometer value, so we had to repeat the set of observations for it.

Below are the table containing the data collected for 100 kiloohm resistor (Table 1) and Potentiometer (Table 2) for Voltage, measured in Volts, and Current, measured in milli Ampere.

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 1: Readings of Voltage and Current for 100 kiloohm Resistor

Voltage (mV)	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 2: Readings of Voltage and Current for Potentiometer

Following table records the reference values measured for both the resistors.

Resistor	Reference Resistance
100 kiloohm	98.8 +/- 0.5 kiloohm
Potentiometer	15.64 +/- 0.08 ohm

Table 3: Reference resistance using multimeter

1.3 Results

1.4 Analysis & Discussion

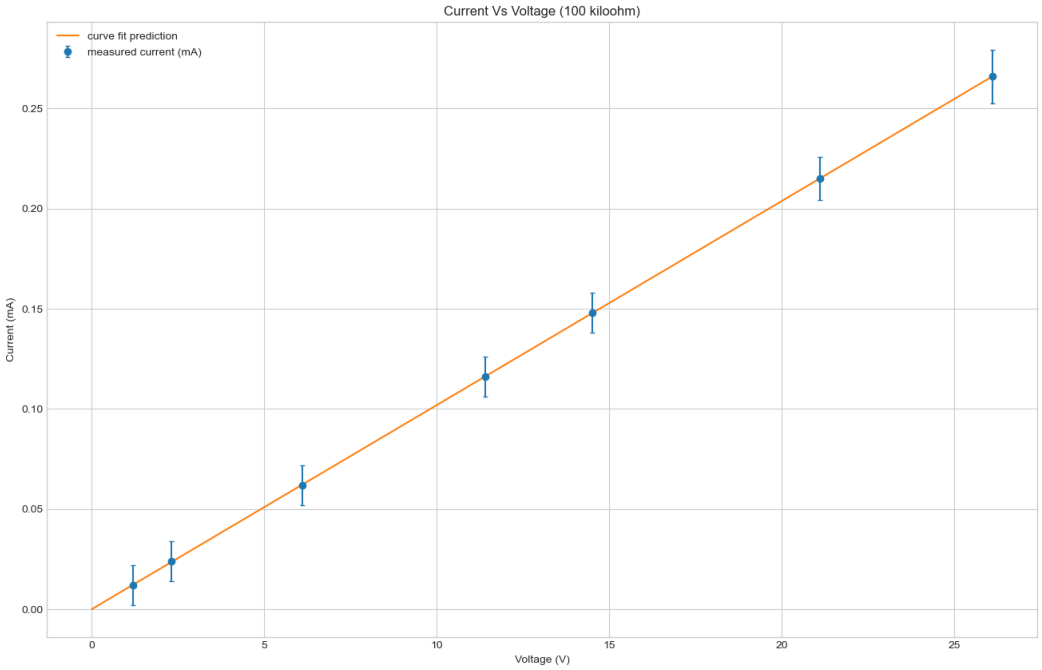

Figure 2: 100 kiloohm data curve fitting

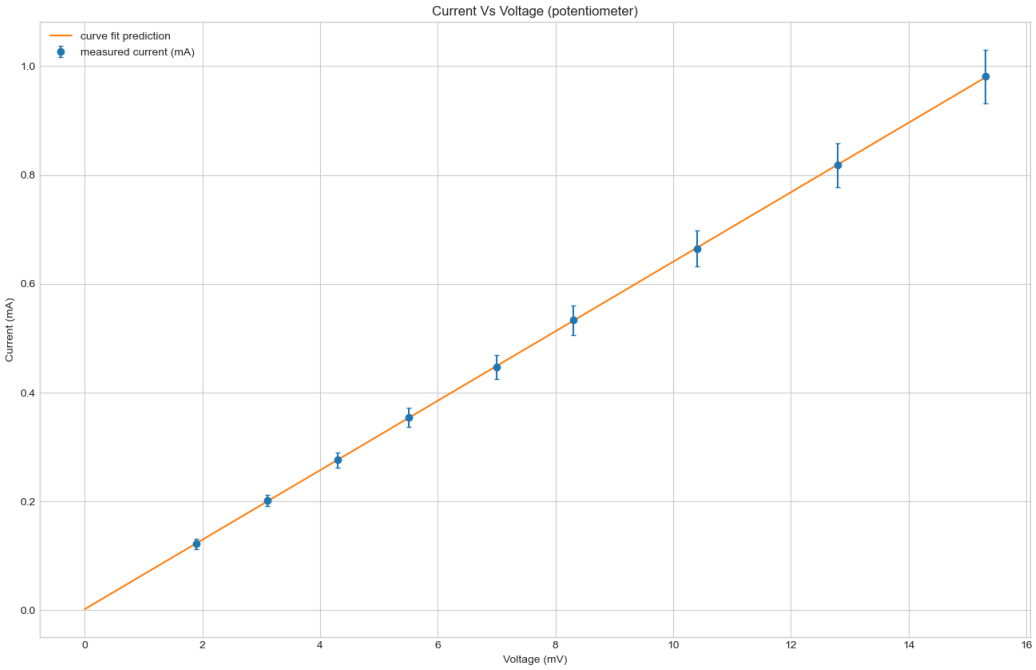

Figure 3: Potentiometer data curve fitting

1.5 Conclusions

2 Exercise 3: Nonlinear fitting methods II

2.1 Introduction

2.2 Methods

2.3 Results

2.4 Analysis & Discussion

2.5 Conclusions

Appendix

Python Code: statslab.py Utility Module

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


##############################################################################
# Utility Methods Library
#
# This file contains some utility method which are common to our data analysis.
# This library also contains customized plotting methods.
##############################################################################

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),
    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 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:
        plt.errorbar(plot_details.xdata,
                     plot_details.ydata,
```

```
                yerr=plot_details.yerrors,
                marker="o",
                label=plot_details.errorbar_legend,
                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)
```

Python Code: Exercise 1

```
import numpy as np
import matplotlib.pyplot as plt

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

# assume 5% uncertinity due to connection errors, human factors etc.
setup_uncetainty = 0.05

def current_uncertainty(current):
    """return the uncertainty in current for given values of current"""
    multimeter_uncertainty = 0.0
    if current > 100:
        multimeter_uncertainty = 1
    elif current > 10:
        multimeter_uncertainty = 0.1
    else:
        multimeter_uncertainty = 0.01

    return max(multimeter_uncertainty, setup_uncetainty*current)

#  model function
def linear_model_function(x, a, b):
    return a*x + b

def analyse_file(filename, title, units):
    print(filename)

    measured_voltages, measured_currents = utils.read_data(filename,
                                                            usecols=(0,1))

    # create error array for the current
    current_errors = np.vectorize(current_uncertainty)(measured_currents)

    # fit the measured data using curve_fit function
    popt, pstd = utils.fit_data(linear_model_function,
                          measured_voltages,
                          measured_currents,
                          current_errors,
                          guess=(1/100, 0))

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

```
    predicted_currents = linear_model_function(voltage_data,
                                               popt[0],
                                               popt[1])

    # calculate the chi2reduced for curve fitted model
    chi2r_curve_fit = utils.chi2reduced(measured_currents,
                                linear_model_function(measured_voltages,
                                                      popt[0],
                                                      popt[1]),
                                current_errors,
                                3)

    # 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.errorbar_legend("measured current (%s)" % units["current"])
    plot_data.fitted_curve_legend("curve fit prediction")
    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("\tLinear model slope (a) = %.2f" % popt[0])
    print("\tLinear model y-intercept (b) := %.2f" % popt[1])
    print("\tEstimated Resistance (1/a) = %.2f %s" % (1/popt[0], units["resistance"]))

    print("\tchi2reduced (Curve Fit) := %.3f" % chi2r_curve_fit)
    print("\tUncertainty in resistance  := %.3f %s" % (pstd[0], units["resistance"]))

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

# files to analyse
file_titles ={
    "100k.csv": {
        "title": "100 kiloohm",
        "units": {
            "voltage": "V",
            "current": "mA",
            "resistance": "kiloohm"
        }
```

```
    },
    "Potentiometer.csv": {
        "title": "potentiometer",
        "units": {
                    "voltage": "mV",
                    "current": "mA",
                    "resistance": "ohm"
                }
        }
}

for filename, data in file_titles.items():
    analyse_file(filename, data["title"], data["units"])
```

Python Code: Exercise 3

References

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

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