PyLab - Ohm and Power laws

Fredrik Dahl Bråten, Pankaj Patil

September 23, 2021

1 Exercise 1: Introduction to fitting methods

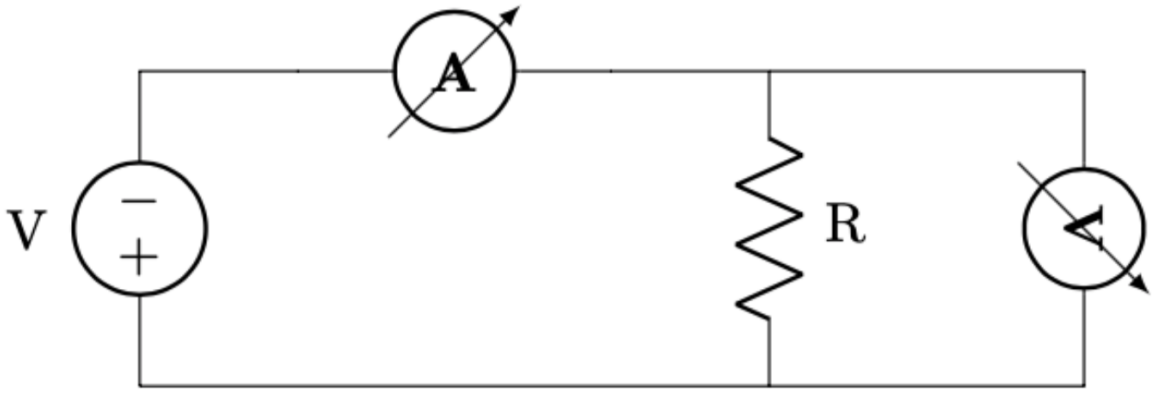
1.1 Introduction

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

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 (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.



Folowing observations were made for the instruments,

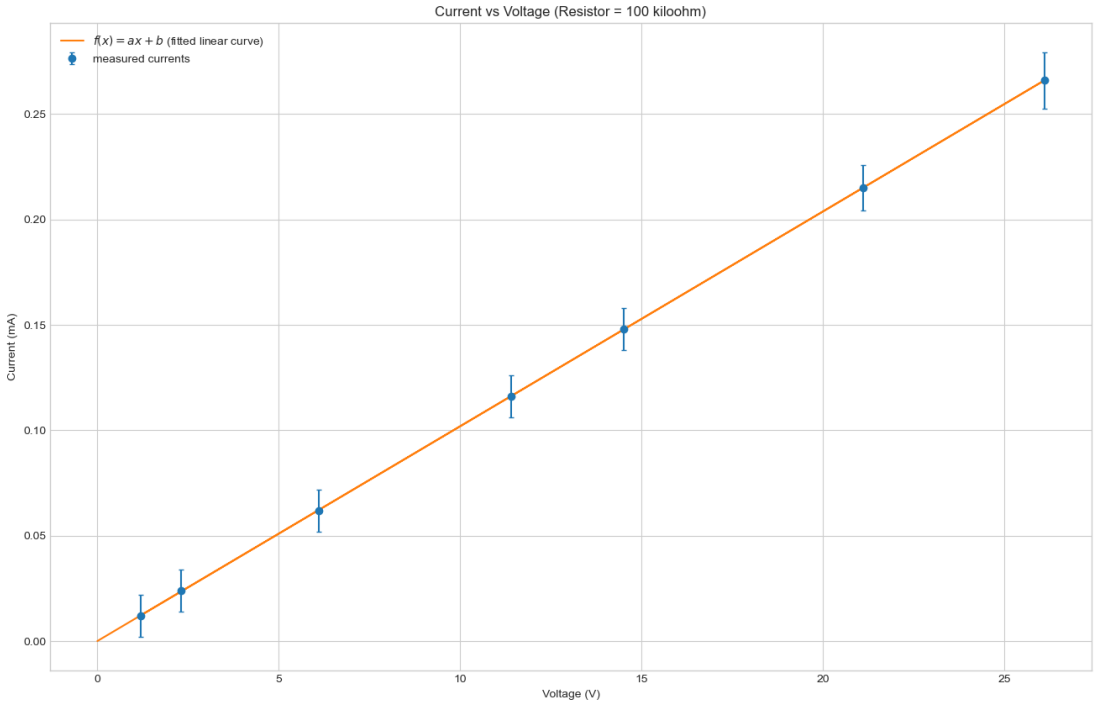
1. One of the multimeter was faulty, and we discarded the readings takes from that multimeter.
2. We accidently changed the potentiometer value, so we had to repeat the set of observations for it.

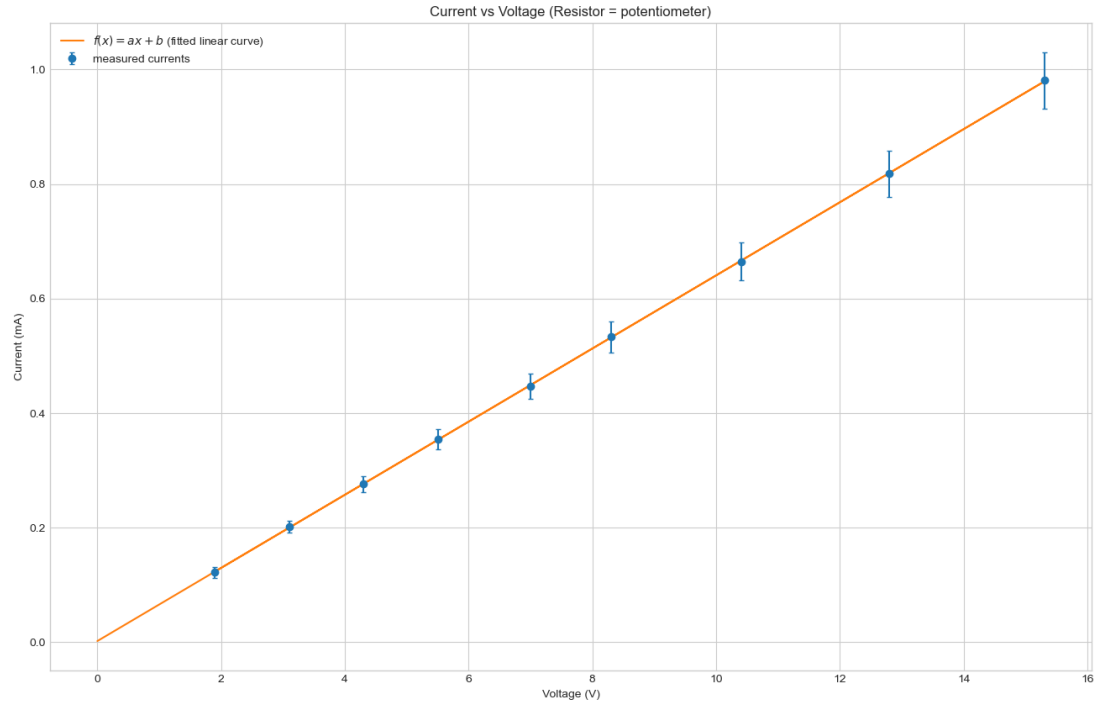
Voltage (V)	Current (mA)
1.2	0.012
2.3	0.024
6.1	0.062
11.4	0.116
14.5	0.148
21.1	0.215
26.1	0.266

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

1.3 Results

1.4 Analysis & Discussion





1.5 Conclusions

2 Exercise 3: Nonlinear fitting methods II

2.1 Introduction

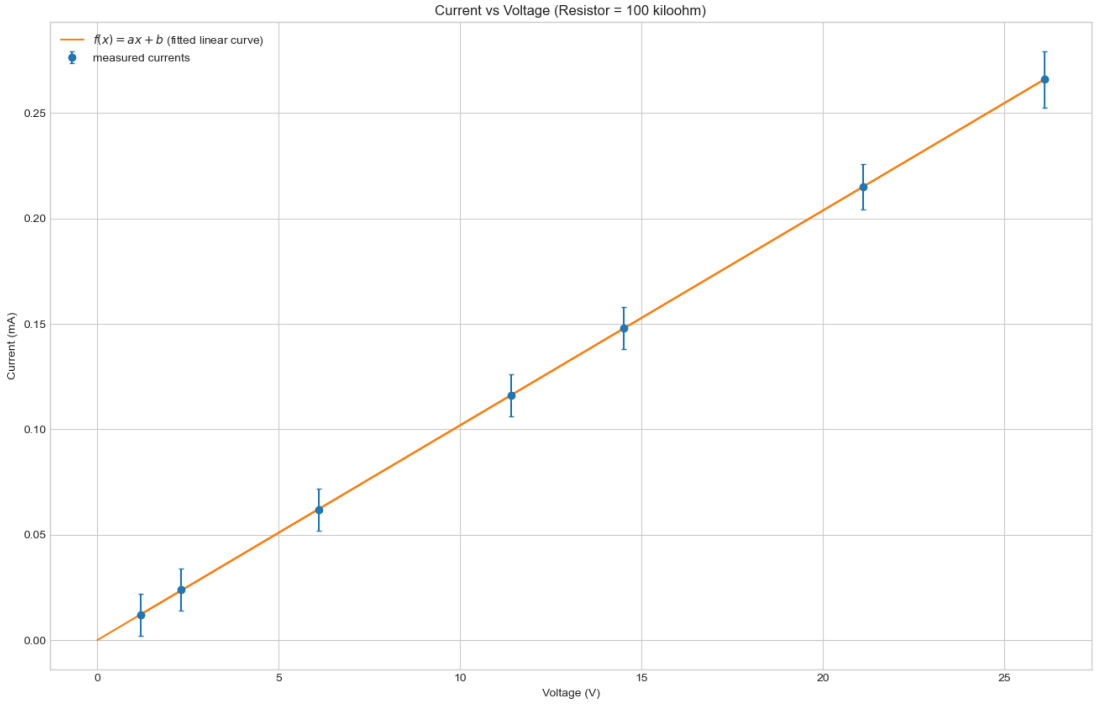
Voltage (V)	Current (mA)
1.2	0.012
2.3	0.024
6.1	0.062
11.4	0.116
14.5	0.148
21.1	0.215
26.1	0.266

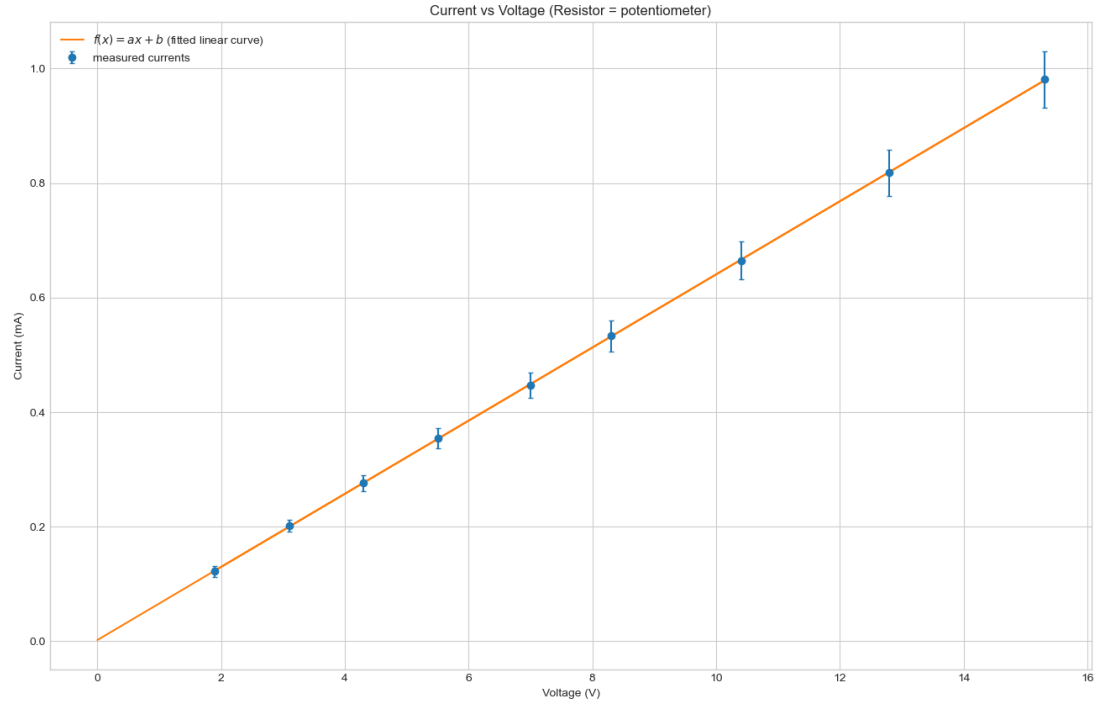
Voltage (V)	Current (mA)
1.9	0.122
3.1	0.202
4.3	0.276
5.5	0.354
7	0.447
8.3	0.533
10.4	0.665
12.8	0.818
15.3	0.981

2.2 Methods

2.3 Results

2.4 Analysis & Discussion





2.5 Conclusions

Appendix

Python Code: Exercise 1

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

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)

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

# uncertainity of connections

def current_uncertainity(current):
    """return the uncertainity in current for given values of current"""
    multimeter_uncertainity = 0.0
    if current > 100:
        multimeter_uncertainity = 1
    elif current > 10:
        return max(0.1, multimeter_uncertainity*current)
        multimeter_uncertainity = 0.1
    else:
        multimeter_uncertainity = 0.01

    return max(multimeter_uncertainity, setup_uncetainity*current)


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

def analyse_file(filename, title):
    # load the csv file as txt
    measured_voltages, measured_currents = np.loadtxt(filename,
                                                      skiprows=1,
                                                      usecols=(0,1),
```

```
                                                delimiter=",",
                                                unpack=True)

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

# do the curve fitting
popt, pcov = optim.curve_fit(linear_model_function,
                             measured_voltages,
                             measured_currents,
                             absolute_sigma=True,
                             sigma=current_errors)

pvar = np.diag(pcov)

# new figure for this file
plt.figure(figsize=(16, 10))
plt.style.use("seaborn-whitegrid")

# plot the error bar chart
plt.errorbar(measured_voltages,
             measured_currents,
             yerr=current_errors,
             marker="o",
             label="measured currents",
             capsize=2,
             ls="")

# plot the fitted curve
# add 0 to the measured data set
measured_voltages_with_0 = np.append(measured_voltages, 0)
plt.plot(measured_voltages_with_0,
         linear_model_function(measured_voltages_with_0, popt[0], popt[1]),
         label='$f(x) = ax + b$ (fitted linear curve)')

# legend and title
plt.title("Current vs Voltage (Resistor = %s)" % title)
plt.xlabel("Voltage (V)")
plt.ylabel("Current (mA)")
plt.legend(loc="upper left")
plt.savefig("lab_1_ex_1_plot_%s.png" % filename[:-4].lower())

chi2r = chi2reduced(measured_currents,
                    linear_model_function(measured_voltages,
                                          popt[0],
                                          popt[1]),
                    current_errors,
```

```
                        2)

    print("filename %s" % filename)
    print("\tlinear fit model gives a=%.2f, b=%.2f" % (popt[0], popt[1]))
    print("\tfitted (average) resistance = %.3f kiloohm" % (1/popt[0]))
    print("\terror in fitted resistance = %.5f kiloohm" % np.sqrt(pvar[0]))
    print("\tmodel chi2r = %.3f" % chi2r)

# files to analyse
file_titles ={
    "100k.csv": "100 kiloohm",
    "Potentiometer.csv": "potentiometer"
}

for filename, title in file_titles.items():
    analyse_file(filename, title)
```

Python Code: Exercise 3

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

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)

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

# uncertainity of connections

def current_uncertainity(current):
    """return the uncertainity in current for given values of current"""
    multimeter_uncertainity = 0.0
    if current > 100:
        multimeter_uncertainity = 1
    elif current > 10:
        return max(0.1, multimeter_uncertainity*current)
        multimeter_uncertainity = 0.1
    else:
        multimeter_uncertainity = 0.01

    return max(multimeter_uncertainity, setup_uncetainity*current)


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

def analyse_file(filename, title):
    # load the csv file as txt
    measured_voltages, measured_currents = np.loadtxt(filename,
                                                      skiprows=1,
                                                      usecols=(0,1),
                                                      delimiter=",",
                                                      unpack=True)
```

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

# do the curve fitting
popt, pcov = optim.curve_fit(linear_model_function,
                             measured_voltages,
                             measured_currents,
                             absolute_sigma=True,
                             sigma=current_errors)

pvar = np.diag(pcov)

# new figure for this file
plt.figure(figsize=(16, 10))
plt.style.use("seaborn-whitegrid")

# plot the error bar chart
plt.errorbar(measured_voltages,
             measured_currents,
             yerr=current_errors,
             marker="o",
             label="measured currents",
             capsize=2,
             ls="")

# plot the fitted curve
# add 0 to the measured data set
measured_voltages_with_0 = np.append(measured_voltages, 0)
plt.plot(measured_voltages_with_0,
         linear_model_function(measured_voltages_with_0, popt[0], popt[1]),
         label='$f(x) = ax + b$ (fitted linear curve)')

# legend and title
plt.title("Current vs Voltage (Resistor = %s)" % title)
plt.xlabel("Voltage (V)")
plt.ylabel("Current (mA)")
plt.legend(loc="upper left")
plt.savefig("lab_1_ex_1_plot_%s.png" % filename[:-4].lower())

chi2r = chi2reduced(measured_currents,
                    linear_model_function(measured_voltages,
                                          popt[0],
                                          popt[1]),
                    current_errors,
                    2)

print("filename %s" % filename)
```

```
    print("\tlinear fit model gives a=%.2f, b=%.2f" % (popt[0], popt[1]))
    print("\tfitted (average) resistance = %.3f kiloohm" % (1/popt[0]))
    print("\terror in fitted resistance = %.5f kiloohm" % np.sqrt(pvar[0]))
    print("\tmodel chi2r = %.3f" % chi2r)

# files to analyse
file_titles ={
    "100k.csv": "100 kiloohm",
    "Potentiometer.csv": "potentiometer"
}

for filename, title in file_titles.items():
    analyse_file(filename, title)
```

References

[1]