Obtaining current-voltage characteristic with Keithley 2602B through GPIB and RS232 on Python (with Numerical Analysis)

M. R. Soubkovsky

Master's in Applied Physics and Physics Engineering
University of Lorraine and CentraleSupélec

Abstract

The goal of this study was to obtain a current-voltage characteristic with the SMU Keithley 2602B through GPIB and RS232 protocols using Python.

1 Device features and commands used

1.1 Keithley 2602B

1.2 Technical Specifications

Keithley 2602B (Fig. 1) is a **Source Measure Unit**[1], in other words it is an equipment that allows the user to use it as a source and measure at the same time.

Figure 1: Keithley 2602B. (Source:`https://assets.tequipment.net`)

The Keithley 2602B is composed by two input/output (called smu a and smu b) channels that can be simultaneously used. The user is able to communicate with the device through either USB, GPIB, RS232.

1.3 Coding Interface

1.3.1 General Commands

The general commands in the Keithley 2602B follow the IEEE Std 488.2. These common commands that are supported by the SMU are listed in Table 1. Although commands are shown in uppercase, common commands are not case sensitive and either uppercase or lowercase can be used. Note that although these commands are essentially the same as those defined by the IEEE Std 488.2 standard, the Series 2600B does not strictly conform to that standard.[2]

As an example, on Python the code for obtaining the device identification **IDN?* would look like:

```python
print(keithley.query("*IDN?"))
```

Code	Name	Description
*IDN?	Identification query	Gives the identification tag of the device
*RST	Reset command	Returns the Series 2600B to default conditions
*TST?	Self-test query	Returns a 0
*CLS	Clear status	Clears all event registers and Error Queue
*TRG	Trigger command	Generates the trigger.EVENT_ID trigger event for use with the trigger model.
*OPC	Operation complete command	Set the Operation Complete bit in the Standard Event Register after all pending commands, including overlapped commands, have completed

Table 1: General Commands

1.3.2 Device Specific Commands

The specific commands that were used to operate the equipment are listed in Table 2.

2 First tests and Communication through MAX

In order to assure that the connection with the device is well established, a series of primary tests is performed. For these tests we used the software Measurement & Automation Explorer (MAX) provided by Natural Instruments (Fig. 2).

MAX Visa Test Panel (Fig. 3) provides the user with a simple GUI to connect with the device and execute the first series of tests and check if the connection is well established.

3 Python Program

In order to explain the whole program, first, a short description of the libraries and the functions will be given.

3.1 Libraries used

A list of libraries that were imported in Python and necessary to the execution of the code that follows is shown below and a brief description of the library is given. Libraries:

- **matplotlib.pyplot**: Provides a MATLAB-like plotting framework inside matplotlib
- **numpy**: adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays
- **visa**: enables you to control all kinds of measurement devices independently of the interface (e.g. GPIB, RS232, USB, Ethernet)
- **tkinter**: standard Python interface to the Tk GUI toolkit
- **time**: provides various time-related functions

Code	Description
beeper.enable=0	deactivate machine beep
smuX.reset()	reset all the input/output channels (smu)
smuX.source.output = smuX.OUTPUT_ON	turn on the smu X as an output (a or b)
smuX.source.output = smuX.OUTPUT_OFF	turn off the smu X as an output (a or b)
smuX.measure.count = NB_OF_MEASUREMENTS	specifies the number of measurements to be stored in the buffer
smuX.source.levelv = VOLTAGE	adjust voltage of smu X to the desired level
print(smuX.measure.i())	return value of current I measured on smu X
smuX.source.func = smuX.OUTPUT_DCVOLTS	set smu X as DC voltage output

Table 2: Keithley 2602B Specific Commands. Source:[2]

3.2 Functions

The following list of functions lists the functions that are part of the program with its parameters and specifications.

- **connexion_choice(connection)**: enables the user to choose bewteen GPIB, RS232, USB and connects with the device.

  connection: *string*, contains the identifier of the connection

- **close_all()**: close any connection
- **reset()**: resets both the smuX's to the default state
- **switchON(smux[, onoff=False])**: allows the user to turn on or off the smuX chosen

  smux: *string*, chosen smuX ('a' or 'b')

  onoff: *string*, [Optional] True to turn on and False to Turn off. Default: False

- **measurement(smux,volts_min,volts_max,nb)**: sends the voltage to the device and measures the current

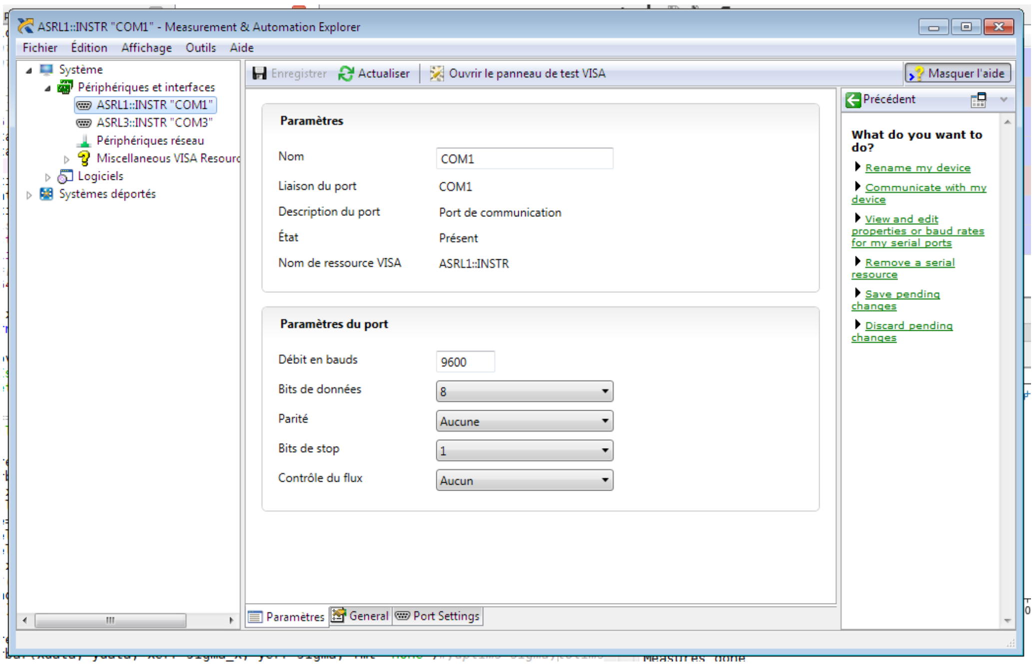

Figure 2: MAX Interface

smux:	*string*,	chosen smuX ('a' or 'b')
volts_min:	*float*,	initial output voltage that is sent to the circuit
volts_max:	*float*,	final output voltage that is sent to the circuit
nb:	*int*,	number of measurements
RETURNS:	*float*	measure of current in Amps

- **complete_measure(smux,volts_min,volts_max,nb)**: calls other functions in order to automatically make all the measurements

smux:	*string*,	chosen smuX ('a' or 'b')
volts_min:	*float*,	initial output voltage that is sent to the circuit
volts_max:	*float*,	final output voltage that is sent to the circuit
nb:	*int*,	number of measurements
RETURNS:	*list*	[input voltage(*float*), measure of current in Amps(*float*)]

3.3 Code

First of all, the libraries described beforehand need to be imported

```python
import visa
import numpy as np
import matplotlib.pyplot as plt
from tkinter import Button, Tk, Frame, Entry, Label, Checkbutton, BooleanVar, StringVar
import time
```

Then we use a series of functions to perform individual tasks. These functions are thoroughly described in the Section 3.2.

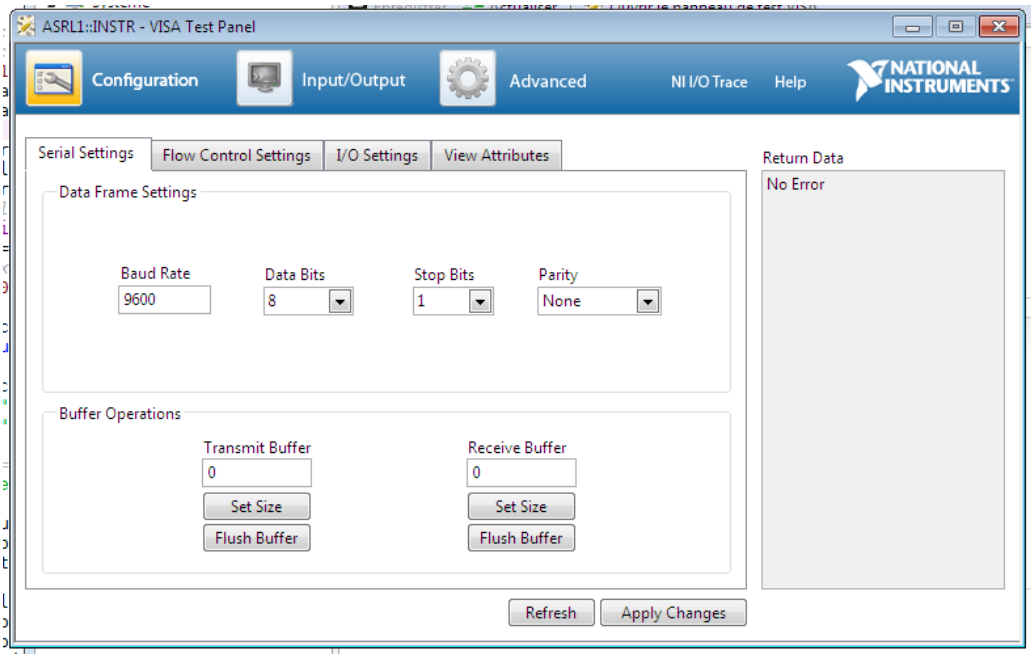

Figure 3: MAX - VISA Test Panel

```python
    def connexion_choice(connexion):
        """Permet de choisir la connexion
        \nAllow to choose the connexion"""
        global keithley
        try:
            rm = visa.ResourceManager() #import visa
            rm.list_resources() #import visa
            keithley = rm.open_resource(connexion)
        except:
            print("Connexion error, check the connexion (GPIB,RS232,USB,Ethernet) and it's number")
            raise StopIteration ("Erreur de connexion. Verifier la connexion (GPIB,RS232,USB,Ethernet) et son numeros \
                                 \nConnexion error, check the connexion (GPIB,RS232,USB,Ethernet) and it's number")
    #-------------------------------------------------------------------------
    def close_all():
        """Coupe la connexion
        \nClose the connexion"""
        try:
            reset() #reset de fin
            keithley.write("beeper.enable=0") #desactive le beep
            keithley.close() #ferme la connexion
            print("Connexion closed")
        except:
            print("Closing error")
            raise StopIteration ("Erreur de fermeture \
                                 \nClosing error")
    #-------------------------------------------------------------------------
    def reset():
```

```python
        """reset les smu"""
        try:
            keithley.write("smua.reset()")
            keithley.write("smub.reset()")
        except:
            print("Reset error")
            raise StopIteration ("Erreur de reset \
                                 \nReset error")
    #---------------------------------------------------------------------------
    def switchON(smux, onoff=False):
        """Active/desactive le smu
        \nTurn on/off the smu"""
        try:
            keithley.write(("smu%s.source.output = smu%s.OUTPUT_ON" if onoff else "smu%s.source.output = smu%s.OUTPUT_OFF") % (smux, smux))
            print("Source on" if onoff else "Source off")
        except:
            print("Source can't be turn on/off")
            raise StopIteration ("Erreur de changement d'etats \
                                 \nSource can't be turn on/off")
    #---------------------------------------------------------------------------
    def measurement(smux,volts_min,volts_max,nb,delay=0):
        """Envoi des tensions et mesure des courants. Enregistre les mesures dans la variable measure
        \nSend tensions and measure currents. Save measures in measure variable"""
        measure=[] #creat array for futur measures
        tension_input=np.linspace(volts_min,volts_max,nb) #creat an array with all tensions needed for measurement
        print('U(V)\t I(A)\t\t points\ttemps(s)') #display tension,current,points,time
        time_begin=time.time()
        i=0
        for x in tension_input: #loop on tensions values
            i+=1
            keithley.write("smu%s.source.levelv = %f" % (smux,x))  #send the voltage x to the smu
            time.sleep(delay) #delay not needed

            y=float(keithley.query("print(smu%s.measure.i())" % smux)) #read current value I(A)
            time_end=time.time()
            print ("%3.3f\t%s\t%s/%s\t%3.3f" % (x,y,i,nb,time_end-time_begin)) #display tension,current,points,time
            if y>0.2:
                close_all()
                print("current too high")
                raise ValueError ("current too high") #used as safety if the volt protection fail (Redundancy)
            measure.append(y) #add the current value to a array regrouping all measurements

        return tension_input,measure #return the voltage and current array
    #---------------------------------------------------------------------------
```

```
def complete_measure(smux,volts_min,volts_max,nb,delay=0):
    """Fonction principale prennant les valeurs de tkinter en entrer. Trace les mesures et les sauvegardes.
    Principal fonction taking tkinter value as input. Plot measures and save them."""
    switchON(smux, True) #active smua

    keithley.write("smu%s.source.func = smu%s.OUTPUT_DCVOLTS" % (smux,smux)) #smua devient source de tension (et donc ne peut être que mesure de courant)
    tension_input,measure=measurement(smux,volts_min,volts_max,nb) #envoi les tensions choisis et mesure les courants associés
    keithley.write("smu%s.source.levelv = 0" % smux)

    plt.figure(num='Diode '+smux+' Characteristic') #plot differents figure according to a specific name
    plt.clf() #clear the graph to avoir superposing data from the same set (can be deactivated if need to superpose)
    plt.title("Diode "+smux+" Characteristic")
    plt.ylabel('I(A)')
    plt.xlabel('U(V)')
    plt.plot(tension_input,measure, '+', label='Diode '+smux) #display current(input_tension) with dots
    plt.legend() #add legend to the graph (take label from plot)
    plt.savefig('Diode %s Characteristic I(U).svg' %smux, format='svg', dpi=1000, bbox_inches='tight') #save the graph in a vector file
    plt.show() #plot data

    np.savetxt('Diode %s Characteristic I(U).csv' % smux,np.transpose((tension_input,measure)),delimiter="\t") #newline='\n'
    switchON(smux, False) #deactivate smu
```

After defining the functions, we attribute values to the variables, raising an error if the voltage passes the chosen limit:

```
connexion='GPIB0::26::INSTR'
connexion='COM1'
volts_min=0  #min voltage
nb=101       #nb of measurements
volts_max=1   # max voltage
delay=0  #time between measurements
smux='a'

if abs(volts_max)>10:
    close_all()
    raise ValueError ("ERROR: Too much voltage.")

connexion_choice(connexion)

reset() #reset both smu
keithley.write("beeper.enable=0") #deactivate the beep
```

And finally we execute the TKinter code:

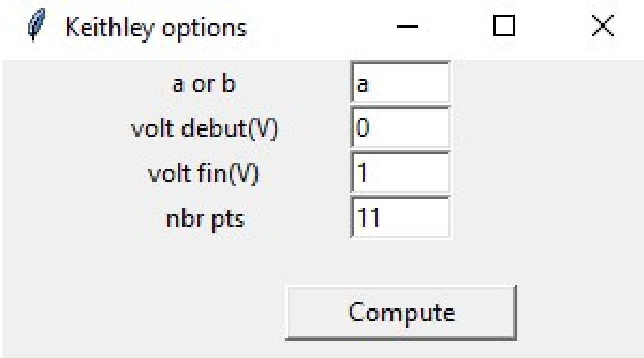

Figure 4: GUI TKinter with measurement options

```python
def compute():
    global delay
    message1["text"] = ""
    message2["text"] = ""
    message3["text"] = ""
    message4["text"] = ""
    message5["text"] = ""
    message6["text"] = ""
    message7["text"] = ""
    try:
        smux=str(smux_entry.get())
        if smux != 'a' and smux != 'b':
            raise ValueError
        try:
            volts_min=float(volt_min_entry.get())  #min voltage
            volts_max=float(volt_max_entry.get())   # max voltage
            if abs(volts_max)>10 or abs(volts_min)>10:
                texte5="abs(volt) <=10"
                message5["text"] = texte5
                print(texte5)
            else:
                nb=int(point_number_entry.get())       #nb de mesures
                if nb<1:
                    texte4="nb>0"
                    message4["text"] = texte4
                    print(texte4)
                else:
    #                delay=float(delay_entry.get())  #temps entre mesures en secondes
                    print(smux,volts_min,volts_max,nb,delay)
                    try:
                        ""
                            tension_input,measure=complete_measure(smux,volts_min,volts_max,nb)
#                            close_all()
                        texte7="Measures done"
                        message7["text"] = texte7
                        print(texte7)
```

```
                    except:
                        reset() #give float error if can't close but actualy is connection error
                        texte6="Unknown error detected"
                        message6["text"] = texte6
                        print(texte6)
        except:
            texte2="floats"
            message2["text"] = texte2
            print(texte2)
    except:
        texte="a or b"
        message1["text"] = texte
        print(texte)

root = Tk()
frame = Frame(root)
root.title("Keithley options")
frame.pack()

L0 = Label(frame, text="a or b")
L0.grid(row=0, column=0)
smux_entry = Entry(frame, textvariable=StringVar(frame, value='a'), bd =2, width=7)
smux_entry.grid(row=0, column=1)

message1 = Label(frame, text="")   #allow to display message when activate [text]
message1.grid(row=0, column=3)

L1 = Label(frame, text="volt debut(V)")
L1.grid(row=1, column=0)
volt_min_entry = Entry(frame, textvariable=StringVar(frame, value=0), bd =2, width=7)
volt_min_entry.grid(row=1, column=1)

message3 = Label(frame, text="")
message3.grid(row=1, column=3)

L2 = Label(frame, text="volt fin(V)")
L2.grid(row=2, column=0)
volt_max_entry = Entry(frame, textvariable=StringVar(frame, value=1), bd =2, width=7)
volt_max_entry.grid(row=2, column=1)

message5 = Label(frame, text="")
message5.grid(row=2, column=3)

L3 = Label(frame, text="nbr pts")
L3.grid(row=3, column=0)
point_number_entry = Entry(frame, textvariable=StringVar(frame, value=11), bd =2, width=7)
point_number_entry.grid(row=3, column=1)

message4 = Label(frame, text="")
```

```python
    message4.grid(row=3, column=3)

    message2 = Label(frame, text="")
    message2.grid(row=4, column=3)

    message6 = Label(frame, text="")
    message6.grid(row=5, column=3)

    compute_button = Button(frame, text="Compute", width=14, command=compute)
    compute_button.grid(row=5, column=1)

    message7 = Label(frame, text="")
    message7.grid(row=5, column=3)


    root.mainloop()
    close_all()
```

The GUI can be seen on Fig. 4.
After the clicking on "Compute", a csv file with the data for the voltage and current, and a pdf chart are saved on the same folder as the python code; and the chart is shown with matplotlib.pyplot.

4 Numerical Analysis

In order to exploit the data, we studied the current-voltage characteristic of a PN diode. The set-up consists of a 100 Ohms resistance and a PN diode connected in series. Both ends are connected to the keithley. The code for the data acquisition is on Section 3.3 and the code for numerical analysis is seen further on Section 4.4.

4.1 Data obtained from a PN diode

First of all, the raw data is collected from the Keithley 2602B. The raw data includes the resistance that is in series, giving origin to a decreasing slope after voltage passes a certain point. The curve in logarithmic scale can be seen on Figure 5.

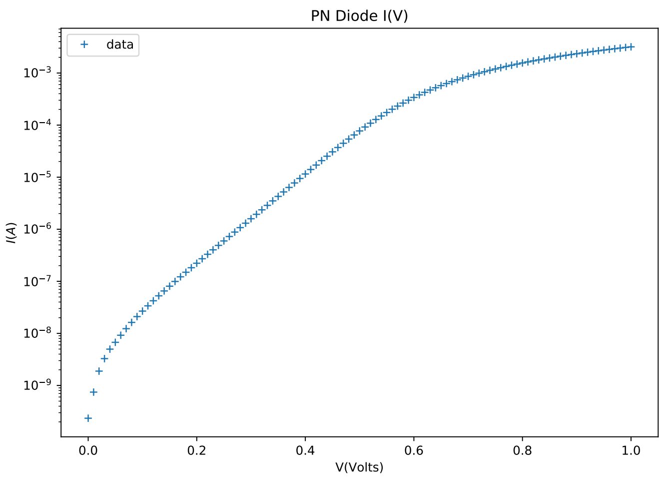

Figure 5: PN Diode Raw Data in logarithmic scale

Knowing that the resistance value is 100 Ohms, value that can also be found comparing experimental data to the Shockley's model, the precise value for the voltage can be found.The data is then corrected using the following formula:

$$V_{\text{diode}} = V_{\text{measured}} - RI_{\text{measured}}$$

The result seen on Figure 6 shows the corrected values. Now we can proceed to apply Shockley's model through a curve-fit and verify the curve fitness through a χ^2 Distribution.

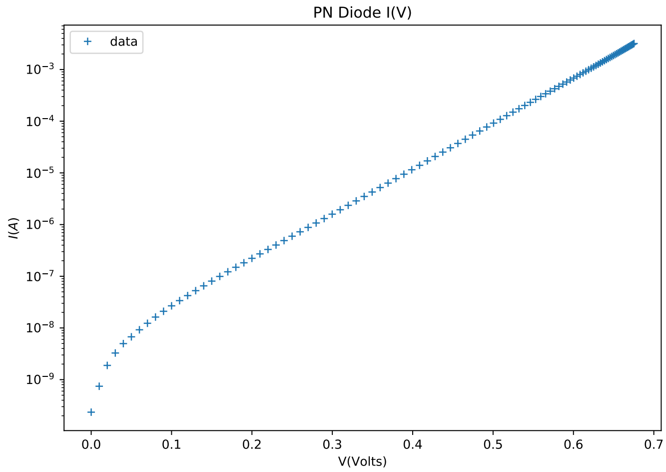

Figure 6: PN Diode Corrected Data in logarithmic scale

4.2 Curve-fit Optimization

The Python library that is used here to perform a curve-fit is **scipy.optimize.curve_fit**. It uses non-linear least squares to fit a function, f, to data.[3] The function requires only three parameters, but has several optional. The ones that are passed here are briefly explained on Table 3.

Parameter	Description
f : callable	The model function, f(x, ...). It must take the independent variable as the first argument and the parameters to fit as separate remaining arguments
xdata	The independent variable where the data is measured.
ydata	The dependent data

Table 3: **curve_fit** parameters that are used. Source:[3]

Now, in order to perform a curve-fit we need to select a stable region that follows the desired model on the curve. The model used here is the **Shockley's model**, whose relation is:

$$I = I_s(e^{\frac{V_d}{\eta V_T}} - 1)$$

where V_d is the tension through the diode (previously noted V_{diode}, η is the ideality factor, I_s the saturation current and V_T the thermal voltage (25.85mV at 300K).
After establishing the model, the curve_fit is ready to be performed. Using the parameters specified in Section 4.4 we obtained the curve shown in Figure7.

4.3 χ^2 Method

The χ^2 distribution can be expressed in the following form:

$$\chi^2 = \sum_{j=1}^{N} \left(\frac{y_j - y(t_j, a, ...)}{\Delta y_j} \right)^2$$

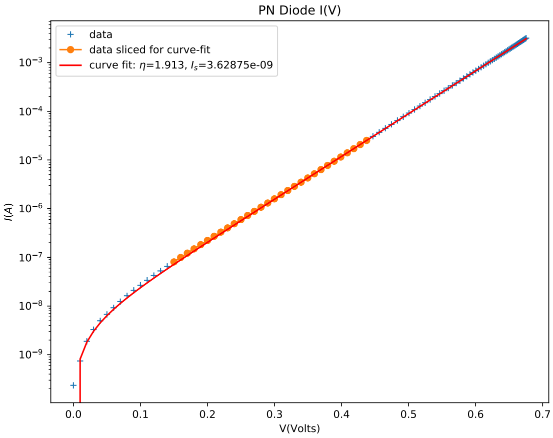

Figure 7: **curve_fit** on corrected data obtained from the PN diode

where N is the number of points, y_j is the obtained data, $y(t_j, a, ...)$ is the curve with parameters $t_j, a, ...$ and r is the number of parameters.[4]
In the above given relation Δy_j is the absolute uncertainty. In the code both relative and absolute uncertainties are taken into account through a selection parameter. The function for the χ^2 Method includes the module **scipy.stats.chi2** and **numpy**.

```
from scipy.stats import chi2
import numpy as np

def chi2_calc(f_exp,f_th,popt=None,f_exp_uncertainty=0.05,confiance=0.95,uncert_type=1,talkative=0):

        if uncert_type==1:                      #percentage uncertainty
                chi_squared=np.sum(((f_exp-f_th)/(f_exp_uncertainty*f_exp))**2)                 #percent
                ↪ uncertainty
        elif uncert_type==0:
                chi_squared=np.sum(((f_exp-f_th)/f_exp_uncertainty)**2)
                ↪ uncertainty
        else:
                raise ValueError('Uncertainty type not accepted. \n Type uncert_type=1 for percentage uncertainty, and uncert_type=0 for absolute uncertainty.')


        if popt is not None:
                degreesoffreedom=len(f_exp)-len(popt)
                if uncert_type==1:
                        confiance=1-f_exp_uncertainty
                chi2_lim=chi2.ppf(confiance,degreesoffreedom)
                if talkative:
                        print("Degrees of freedom : ",degreesoffreedom)
                        print("chi2 ppf (according to degrees of freedom) : ",chi2_lim)
                        print("chi2 calculated : ",chi_squared)
                if chi2_lim>chi_squared:
                        if talkative:
                                print("Model is acceptable.")
```

```python
                        acceptance=True
                else:
                        if talkative:
                                print("Model is not acceptable.")
                        acceptance=False

        return acceptance,chi_squared
```

The χ^2 Method allowed us to test our curve fitness. A constant uncertainty of 5% was used. The results can be seen on Figure 8 and the code on Section 4.4 .

```
eta=1.912993e+00     Is=3.628749e-09
Degrees of freedom :  28
chi2 ppf (according to degrees of freedom) :  41.33713815142739
chi2 calculated :  35.40992440849461
Model is acceptable.
[Finished in 1.4s]
```

Figure 8: Results on command prompt for the χ^2 calculation.

4.4 Code

```python
import matplotlib.pyplot as plt
import numpy as np
import math
from scipy.optimize import curve_fit
from scipy.stats import chi2


data=np.loadtxt("Diode a Characteristic I(U).csv")

xdata=data[:,0]
ydata=data[:,1]

resistance=102

xdata=xdata-resistance*ydata                                    #correction v=v_bi-R*I

start=15                                                                            #split data for curve fit
interval=30
x=xdata[start:start+interval]
y=ydata[start:start+interval]


start_line=98                                                                     #split data for tracing a straight line
interval_line=1
x_line=xdata[start_line:start_line+interval_line]
y_line=ydata[start_line:start_line+interval_line]


logarithm=True
```

```python

if logarithm:

    plt.semilogy(xdata,ydata,'+', label="data")
    plt.semilogy(x,y,'-o', label="data sliced for curve-fit")
    # plt.errorbar(xdata, ydata, xerr=0.05, yerr=0.05)
else:
    plt.plot(xdata,ydata,'+', label="data")
    plt.plot(x,y,'o', label="data")
    # plt.errorbar(xdata, ydata, xerr=0.05, yerr=0.05)

def current(V, eta=10,Is=10E-10):

    If=[]
    Vt=25.85E-3
    for i in V:
        x=(i)/(eta*Vt)

        If.append(Is*(math.exp(x)-1))


    return If #return If:final-current


p0=(10,10E-10)
sigma=0.95*np.ones((len(x),))
popt, pcov = curve_fit(current, x,y,p0=p0,sigma=sigma)
print("eta=%e \t Is=%e"%(popt[0],popt[1]))

# def horizontal_line(x,b):                               #used for tracing a straight horizontal line (graphic calculus of resistance)
#         b=np.ones((len(x),))*b
#         return 0*x+b

# y2=horizontal_line(xdata,ydata[100])

if logarithm:
    # plt.semilogy(xdata,y2)                              #straight line
    plt.semilogy(xdata, current(xdata, *popt), 'r-',label=r'curve fit: $\eta$=%5.3f, $I_s$=%g' % tuple(popt))
else:
    # plt.plot(xdata,y2)                                  #straight line
    plt.plot(xdata, current(xdata, *popt), 'r-',label=r'curve fit: $\eta$=%5.3f, $I_s$=%g' % tuple(popt))


def chi2_calc(f_exp,f_th,popt=None,f_exp_uncertainty=0.05,confiance=0.95,uncert_type=1,talkative=0):

    if uncert_type==1:                      #percentage uncertainty
```

```
        chi_squared=np.sum(((f_exp-f_th)/(f_exp_uncertainty*f_exp))**2)            #percent
        ↪ uncertainty
    elif uncert_type==0:
        chi_squared=np.sum(((f_exp-f_th)/f_exp_uncertainty)**2)
        ↪ uncertainty
    else:
        raise ValueError('Uncertainty type not accepted. \n Type uncert_type=1 for
        ↪ percentage uncertainty, and uncert_type=0 for absolute uncertainty.')


    if popt is not None:
        degreesoffreedom=len(f_exp)-len(popt)
        if uncert_type==1:
            confiance=1-f_exp_uncertainty
        chi2_lim=chi2.ppf(confiance,degreesoffreedom)
        if talkative:
            print("Degrees of freedom : ",degreesoffreedom)
            print("chi2 ppf (according to degrees of freedom) : ",chi2_lim)
            print("chi2 calculated : ",chi_squared)
        if chi2_lim>chi_squared:
            if talkative:
                print("Model is acceptable.")
            acceptance=True
        else:
            if talkative:
                print("Model is not acceptable.")
            acceptance=False

    return acceptance,chi_squared

acceptance,chi_squared=chi2_calc(y,current(x,*popt),popt=popt,f_exp_uncertainty=0.05,uncert_type=1,
↪ talkative=1)


plt.xlabel("V(Volts)")
# plt.title("Schottky Diode Characteristic : I(V)")
plt.title("PN Diode I(V)")
plt.ylabel(r"$I(A)$")

plt.legend()
plt.show()

# plt.savefig("m-schottky-curve%s.pdf"%(text))
```

5 GitHub Repository

This project was published on GitHub and is public under MIT License. The repository is accessible through this link https://github.com/marcelrsoub/keithley-visa-measurements.

References

[1] Source measure unit, August 2018. Page Version ID: 854854432.

[2] Series 2600b System SourceMeter Manual.

[3] scipy.optimize.curve_fit — SciPy v1.2.1 Reference Guide.

[4] George B. Arfken and Hans-Jurgen Weber. *Mathematical Methods for Physicists*. Elsevier, Acad. Press, Amsterdam, 6. ed., internat. ed., [5. nachdr.] edition, 2008. OCLC: 551222393.