Documentation on python scripts for TL, LM-OSL and CW-OSL deconvolution

Prevezanou Konstantina

Contents

1 Introduction

The following scripts aim to deconvolve stimulated luminescence curves, using the Python programming language. No previous programming knowledge is required.

Note: during the analysis only the code for TL will be presented. The only differences that occur between the different scripts are in the equations solely.

2 Conditions

To be able to run the scripts you will need to make the necessary installations on your computer. These installations are:

- Python 3,
- A compiler to run the scripts,
- Python libraries:
  - numpy
  - csv
  - matplotlib.pyplot
  - scipy
  - easygui
  - lmfit
  - pandas
  - pybroom
  - os
  - tkinter

3 Program analysis

3.1 Libraries

The first code lines are dedicated to import the libraries we need through out the program. For specific libraries some additional actions have been taken:

- numpy: has been introduced with the initials np for easier use in the program, likewise for matplotlib.pyplot, pandas and pybroom.
- From some libraries we have imported only specific sub-libraries. This helps us with the writing of the code as we do not need to mark the sublibrary in every report.

```
import numpy as np
import csv
import matplotlib.pyplot as plt
import scipy.constants
from scipy.special import lambertw, exp1
from lmfit import Parameters, report_fit, Parameter, Model
import pybroom as br
import pandas as pd
```

```
from easygui import *
from tkinter import *
from tkinter import filedialog,ttk
import os
from os.path import exists
```

3.2 Definition of functions

We define the functions that will be used in the rest of the script. We have:

1. Theoretical function for TL,
2. Function for the frequency factor,
3. Function for the model creation using the theoretical function and the parameters,
4. Function to make lists for each parameter,
5. Function to check if a file with the same name exists, it adds a number at the end.

```
k=scipy.constants.physical_constants['Boltzmann constant in eV/K']
K=k[0]

def tl(x,Im,Tm,E,R):
  '''Theoretical function for the TL phenomenon'''
  c=(1-R)/R
  F=x*np.exp(-E/(K*x))+(E/K)*(- np.real(exp1(E/(K*x))))
  Z=1/c-np.log(c)+(E*np.exp(E/(K*Tm))*F)/(K*(Tm**2)*(1-1.05*(R**1.26)))
  #W=np.real(lambertw(np.exp(Z)))
  Fm=Tm*np.exp(-E/(K*Tm))+(E/K)*(- np.real(exp1(E/(K*Tm))))
  Zm=1/c-np.log(c)+(E*np.exp(E/(K*Tm))*Fm)/(K*(Tm**2)*(1-1.05*(R**1.26)))
  Wm=np.real(lambertw(np.exp(Zm)))

  for i in range(len(Z)):
    if Z[i]<710:
      W=np.real(lambertw(np.exp(Z)))
    else:
      W=Z-np.log(Z)
    return Im*((Wm+Wm**2)/(W+W**2))*np.exp((E*(x-Tm))/(K*x*Tm))


def freq_f(Tm,E,R):
  '''Function for the computation of the frequency factor'''
  c=(1-R)/R
  F_m=((K*(Tm**2))/E)*np.exp(-E/(K*Tm))*(1-((2*K*Tm)/E))
  z_m=1/c-np.log(c)+(E*np.exp(E/(K*Tm))*F_m)/(K*(Tm**2)*(1-1.05*(R**1.26))
   )
  w=np.real(lambertw(np.exp(z_m)))
  return ((1-R)*(1+w)**2*heat_rate*E)/((1+2*w)*(K*Tm**2*np.exp(-E/(K*Tm)))
   )


def make_model(num):
  '''We define the model using the theoretical function and define the
   parameters'''
  pref = "tl{}_".format(num)
  peak = list_peaks[num]
  model = Model(tl, prefix = pref)
  model.set_param_hint(pref+'Im', value = a[0][num], min = a[1][num], max
   = a[2][num])
```

```
  model.set_param_hint(pref+'Tm', value = b[0][num], min = b[1][num], max
   = b[2][num])
  model.set_param_hint(pref+'E', value = c[0][num], min = c[1][num], max =
    c[2][num])
  model.set_param_hint(pref+'R', value = d[0][num], min = d[1][num], max =
    d[2][num])
  return model


def loop_init(name1,name2,name3,num):
  '''Function to make the list for each parameter including the initial
   parameters'''
  name1=[]
  name2=[]
  name3=[]
  for i in range(num,len(datap),5):
    name1.append(datap[i][0])
    name2.append(datap[i][1])
    name3.append(datap[i][2])
  name1=np.array(name1,dtype=float)
  name2=np.array(name2,dtype=float)
  name3=np.array(name3,dtype=float)
  return name1,name2,name3


def file_exist(f):
  '''Function to check if a file exists and if it does it adds a number at
    the end of the name'''
  fnew = f
  root, ext = os.path.splitext(f)
  i = 0
  while os.path.exists(fnew):
    i += 1
    fnew = '%s_%i%s' % (root, i, ext)
  return fnew
```

3.3 Script information and files input

This is the first pop-up box where important information about the script can be found. Also, here the user will give the experimental data as well as the initial parameters file.

The file with the experimental data contain in the first column the number of the line, in the second column the x-axis data and in the third column the y-axis data in 'txt (tab delimited)' formatting.

For the parameter file the format should be three columns - first contains the values (Im, Tm, E, R), second the minimum and third the maximum for each value (again in 'txt (tab delimited)' formatting).

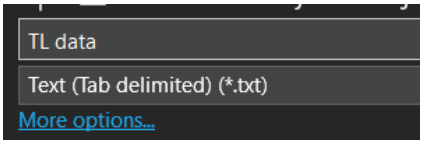
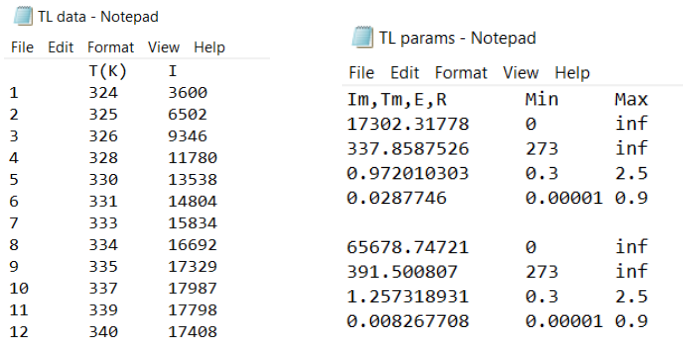


Figure 1: Tab delimited txt file



(a) Experimental data file

(b) Initial parameters file

Figure 2: Example of input files

```
root=Tk()
root.title('Data for the deconvolution')
root.geometry('700x370')

global b_sequence, files, fileswithoutpath
b_sequence=[]
files=[]
fileswithoutpath=[]


def addfile():
  '''Function to go through computer files'''
  filename=filedialog.askopenfilename(initialdir=os.getcwd(),title="Select
    File",filetypes=(('TXT files','.txt'),('All files','.')))
  files.append(filename)
  filenamewithout=os.path.split(filename)[1]
  fileswithoutpath.append(filenamewithout)

def which_button(t):
  '''Function to save the order of button push'''
  b_sequence.append(t)

def clickonce(k):
  '''Function to allow the user to press the button only once'''
  k.configure(state=DISABLED)

notice=Label(root,text="What you should know to run the script properly:",
   font=('Arial',10, 'underline'))
notice.pack()

info=Label(root,text="\n1. You need one file with experimental data (first
    column index in ascending order, second column x-axis data,\nthird
   column y-axis data) in  txt  (tab delimited)  formating.\n\n2. You
   need a second file with the initial guess of the parameters in  txt
   (tab delimited)  formating.\nThe format should be: three columns -
   first contains the values, second the minimum and third the maximum
   for each value.\nAlso, for each peak the parameters (Im, Tm, E, R)
   will be given and between the different peaks should be a blank line.\
   n\n3. Read carefully the instructions of each pop-up box to correctly
   complete the gaps.\n\n4. At the end, fitted parameters will be stored
   in a new directory,  Fitted  parameters ,\nand theoretical data in
   new directory,  Theoretical  data .\n\n5. To save the graph just
   click on  Save  (the button is on the down left corner of the
   graph).\n\n")
```

```
info.pack()

out=Button(root,text='Continue',padx=5,pady=4,fg='white',bg='black',
    command=root.destroy ,font=('Arial', 11))
out.pack(side=BOTTOM)

frame1=Frame(root)
frame1.place(relwidth=0.41,relheight=0.1,relx=0.3,rely=0.73)

expdata=Button(frame1,text='Experimental data',padx=6,pady=5,fg='white',bg
    ='black',command=lambda:[addfile(),which_button('exp'),clickonce(
    expdata)],font=('Arial', 11))
expdata.pack(side=LEFT)

expparams=Button(frame1,text='Initial parameters',padx=6,pady=5,fg='white'
    ,bg='black',command=lambda:[addfile(),which_button('para'),clickonce(
    expparams)],font=('Arial', 11))
expparams.pack(side=RIGHT)

root.mainloop()
```

3.4 Data to lists

Here the data from the files are gathered and put into individual lists. Also, the name for the output files is made using the name of the experimental data file.

```
if b_sequence[0]=='exp':
  #Import experimental data
  with open (files[0], 'r') as file:
    data=list(csv.reader(file,delimiter='\t'))

  usedata=np.array(data[1:], dtype=float)
  x=usedata[:,1]
  y=usedata[:,2]


  #Import parameters
  with open (files[1], 'r') as filep:
    datap=list(csv.reader(filep,delimiter='\t'))

  a = loop_init('Iinit','minIm','maxIm',1)
  b = loop_init('Tinit','minTm','maxTm',2)
  c = loop_init('Einit','minE','maxE',3)
  d = loop_init('Rinit','minR','maxR',4)

  '''We create the names of the output files'''
  file1=fileswithoutpath[0].replace('.txt','')

  newfile1=file1+'_out.txt'
  newfile2=file1+'_param.txt'
else:
  #Import experimental data
  with open (files[1], 'r') as file:
    data=list(csv.reader(file,delimiter='\t'))

  usedata=np.array(data[1:], dtype=float)
  x=usedata[:,1]
  y=usedata[:,2]


```

```
  #Import parameters
  with open (files[0], 'r') as filep:
    datap=list(csv.reader(filep,delimiter='\t'))

  a = loop_init('Iinit','minIm','maxIm',1)
  b = loop_init('Tinit','minTm','maxTm',2)
  c = loop_init('Einit','minE','maxE',3)
  d = loop_init('Rinit','minR','maxR',4)

  '''We create the names of the output files'''
  file1=fileswithoutpath[1].replace('.txt','')

  newfile1=file1+'_out.txt'
  newfile2=file1+'_param.txt'
```

3.5 Extra data for deconvolution

In the second pop-up box, the user gives the number of peaks, the start and end for the deconvolution process and the heating rate which is only used to calculate the frequency factor.

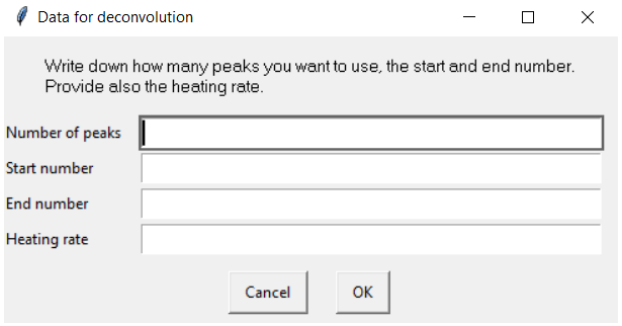


Figure 3: Second pop-up box

```
msg = "Write down how many peaks you want to use, the start and end number
    .\nProvide also the heating rate."
title = "Data for deconvolution"
fieldNames = ['Number of peaks',"Start number",'End number','Heating rate'
    ]
fieldValues = []
fieldValues = multenterbox(msg,title,fieldNames)

peaks=int(fieldValues[0])
list_peaks=range(peaks)
startfit = int(fieldValues[1])
endfit = int(fieldValues[2])
heat_rate=float(fieldValues[3])

x=np.array(x[startfit:endfit])
y=np.array(y[startfit:endfit])
```

3.6 Deconvolution

Firstly, the model runs through a loop in order to calculate the total peak. After the user defines the optimization method through a third pop-up box, the deconvolution process takes place.

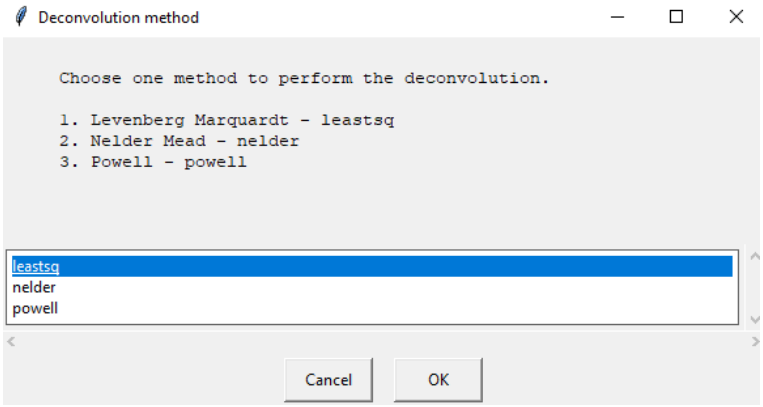
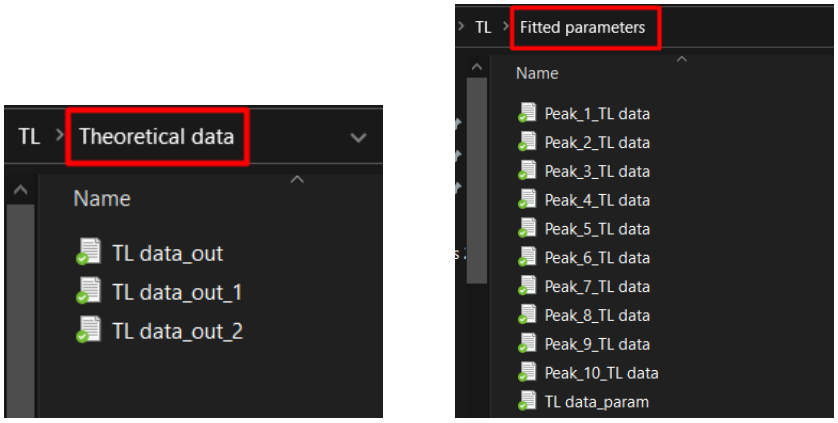


Figure 4: Third pop-up box

```
mod = None
for i in range(len(list_peaks)):
  this_mod = make_model(i)
  if mod is None:
    mod = this_mod
  else:
    mod = mod + this_mod

msg = 'Choose one method to perform the deconvolution.\n\n1. Levenberg
   Marquardt - leastsq\n2. Nelder Mead - nelder\n3. Powell - powell'
title = "Deconvolution method"
choices = ['leastsq','nelder','powell']
choice = choicebox(msg, title, choices)

howtofit=choice

'''Deconvolution'''
out=mod.fit(y, x=x, method=howtofit)
```

3.7 Output files

Firstly two new directories are created, one for the theoretical data and one for the fitted parameters.



(a) Theoretical data directory (b) Fitted parameters directory

Figure 5: Directories

Then, two files are created, one with the theoretical data and one with the fitted parameters. Simultaneously, the frequency factor and the FOM are calculated.

	x	data	best_fit	residual	Model(tl, prefix='tl0_')	Model(tl, prefix='tl1_')	Model(tl, prefix='tl2_')	Model(tl, prefix='tl3_')	Model(tl, prefix='tl4_')	Model(tl, prefix='tl5_')
0	325	6502	9550.328	3048.328	9466.152	83.003	1.169	0.006	0	0
1	326	9346	10334.09	988.087	10237.471	95.253	1.357	0.007	0	0
2	328	11780	11961.24	181.238	11834.28	125.126	1.823	0.009	0	0
3	330	13538	13603.17	65.169	13436.897	163.818	2.441	0.013	0	0
4	331	14804	14397.68	-406.324	14207.629	187.21	2.821	0.016	0	0
5	333	15834	15851.7	17.704	15604.028	243.895	3.758	0.023	0	0
6	334	16692	16473.02	-218.979	16190.62	278.043	4.332	0.027	0	0
7	335	17329	16996.38	-332.619	16674.643	316.718	4.989	0.032	0	0

(a) Theoretical data file

	Peaks	Imax	Tmax	E	R	s	Total Fom
0	Peak_1	17302.32	337.859	0.972	0.029	3.04E+13	1.893
1	Peak_2	65678.75	391.501	1.257	0.008	1.45E+15	
2	Peak_3	64231.62	430.066	1.362	0.044	7.5E+14	
3	Peak_4	72037.33	461.681	1.64	0	7.13E+16	
4	Peak_5	192966.8	487.976	2.2	0.018	5.56E+21	
5	Peak_6	2526.236	525	1.735	0.14	3.05E+15	

(b) Fitted parameters file

Figure 6: Output files

```
da = br.augment(out)
da=da.round(3)
da.to_csv(file_exist("Theoretical data/{}".format(newfile1)),sep='\t')

residual = da['residual'].to_numpy()
yfitdata= da['best_fit'].to_numpy()
fom=round(100*(np.sum(abs(residual))/(np.sum(yfitdata))),3)
Fom=['' for i in range(peaks)]
Fom[0]=fom

dt=br.tidy(out)
values=dt['value'].to_numpy()

If=[]
Tf=[]
Ef=[]
Rf=[]
name_peaks=[]

for i in range(1,peaks+1):
  name_peaks.append('Peak_{}'.format(i))
for i in range(0,len(dt),4):
  Ef.append(values[i])
for i in range(1,len(dt),4):
  If.append(values[i])
for i in range(2,len(dt),4):
  Rf.append(values[i])
for i in range(3,len(dt),4):
  Tf.append(values[i])

frequency_factor=[]
for i in range(peaks):
  s=freq_f(Tf[i],Ef[i],Rf[i])
  frequency_factor.append(s)


df = pd.DataFrame(list(zip(name_peaks,If,Tf,Ef,Rf,frequency_factor,Fom)),
    columns =['Peaks', 'Imax','Tmax','E','R','s','Total Fom'])
```

```
df=df.round(3)
df.to_csv(file_exist("Fitted parameters/{}".format(newfile2)),sep='\t')
```

Moreover, there is the possibility for the user to choose if he wants to save each peak's parameters in separate files adding a condition (ex. Dose).

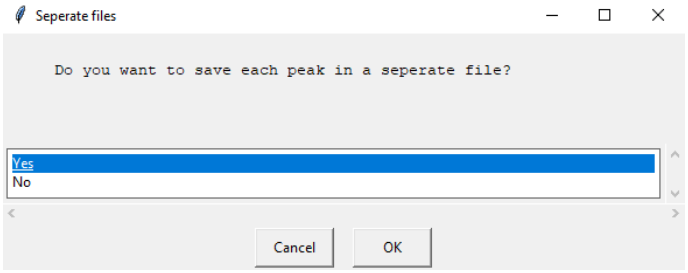
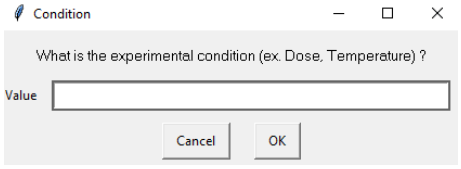


Figure 7: Fourth pop-up box

If the user clicks on 'Yes', another pop-up box will appear in order to give the value of the extra condition.



(a) Fifth pop-up box

	Peaks	Imax	Tmax	E	R	s	Condition
0	Peak_1	17302.318	337.859	0.972	0.029	3.044302e+13	854

(b) One peak data file

Figure 8: One peak data files

```
dk=pd.DataFrame(list(zip(name_peaks,If,Tf,Ef,Rf,frequency_factor)),columns
    =['Peaks', 'Imax','Tmax','E','R','s'])
dk=dk.round(3)

msg = 'Do you want to save each peak in a seperate file?'
title = 'Seperate files'
choices = ['Yes','No']
choice = choicebox(msg, title, choices)

seper_file=choice

if seper_file=='No':
  pass
else:
  '''Extra condition to import'''
  msg = "What is the experimental condition (ex. Dose, Temperature) ?"
  title = "Condition"
  fieldNames = ['Value']
  fieldValues = []
  fieldValues = multenterbox(msg,title,fieldNames)

  column_vl=fieldValues[0]

  dk['Condition'] = column_vl

  '''Seperate files creation'''
  for i in range(peaks):
    if exists('Fitted parameters/Peak_{}_{}.txt'.format(i+1,file1))==True:
      with open ('Fitted parameters/Peak_{}_{}.txt'.format(i+1,file1), 'a'
   ) as filep:
```

```
        filep.write('\n')
        filep.write('{}'.format(i)+'  '+str(dk.loc[i,'Peaks'])+'  '+str(dk
    .loc[i,'Imax'])+'  '+str(dk.loc[i,'Tmax'])+'  '+str(dk.loc[i,'E'])+'
    '+str(dk.loc[i,'R'])+'  '+str(dk.loc[i,'s'])+'  '+str(dk.loc[i,'
    Condition']))
     else:
       with open ('Fitted parameters/Peak_{}_{}.txt'.format(i+1,file1), 'w'
    ) as filep:
        filep.write(str(dk.loc[[i]]))
```

3.8 Plot and print

We print the total FOM, the fitted data and the plot of the data.

```
Total FOM is:  1.893  %.

Name        Value      Min      Max Init_Value
tl0_E       0.972      0.3      2.5     0.972
tl0_Im   1.73e+04        0      inf  1.73e+04
tl0_R     0.02877    1e-05      0.9   0.02877
tl0_Tm      337.9      273      inf     337.9
tl1_E       1.257      0.3      2.5     1.257
tl1_Im   6.568e+04        0      inf  6.568e+04
tl1_R    0.008268    1e-05      0.9  0.008268
tl1_Tm      391.5      273      inf     391.5
```

Figure 9: Printed data

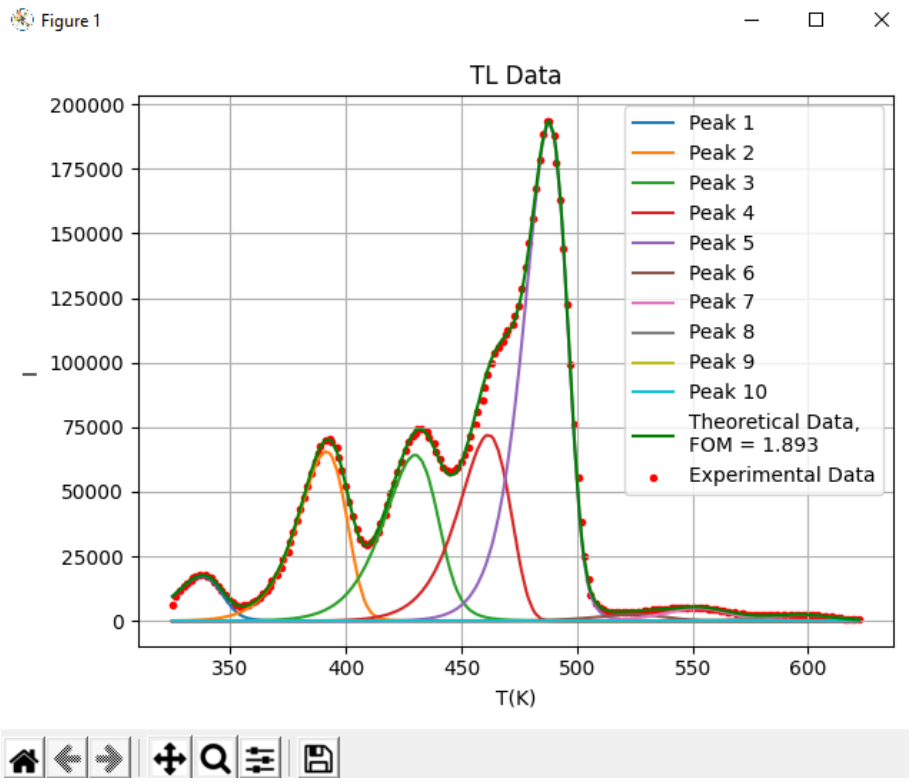


Figure 10: Plot

```
print('\nTotal FOM is: ', fom, ' %.\n')
out.params.pretty_print(columns=['value', 'min', 'max','init_value'])

i=0
```

```
for i in range(peaks):
  ytl=da["Model(tl, prefix='tl{}_')".format(i)].to_numpy()
  plt.plot(x,ytl,label='Peak {}'.format(i+1))


plt.plot(x, out.best_fit,label='Theoretical Data, \nFOM = {}'.format(fom),
    color='g')
plt.scatter(x,y,label='Experimental Data',color='r',marker='.')
plt.title('TL Data')
plt.xlabel(data[0][0])
plt.ylabel(data[0][1])
plt.grid(True)
plt.tight_layout()
plt.legend()
plt.show()
```

4 Appendix

The scripts conducted were tested on Python 3.9.1, Sublime Text 3 and

- altgraph, 0.17
- appdirs, 1.4.4
- asteval, 0.9.23
- auto-py-to-exe, 2.9.0
- backcall, 0.2.0
- bottle, 0.12.19
- bottle-websocket, 0.2.9
- certifi, 2021.5.30
- cffi, 1.14.5
- chardet, 4.0.0
- colorama, 0.4.4
- cycler, 0.10.0
- decorator, 5.0.9
- distlib, 0.3.2
- docutils, 0.17.1
- easygui, 0.98.2
- Eel, 0.12.4
- emcee, 3.0.2
- filelock, 3.0.12
- future, 0.18.2
- future-fstrings, 1.2.0
- gevent, 21.1.2
- gevent-websocket, 0.10.1
- greenlet, 1.1.0
- idna, 2.10
- ipython, 7.24.1
- ipython-genutils, 0.2.0
- jedi, 0.18.0
- Jinja2, 3.0.1
- kiwisolver, 1.3.1
- line-profiler, 3.3.0
- lmfit, 1.0.2
- MarkupSafe, 2.0.1
- matplotlib, 3.3.4
- matplotlib-inline 0.1.2
- mpmath, 1.2.1
- numpy, 1.20.1
- pandas, 1.2.2
- parso, 0.8.2
- pefile, 2021.5.24
- pexpect, 4.8.0
- pickleshare, 0.7.5
- Pillow, 8.1.0
- pip, 21.1.2
- prettytable, 2.0.0
- prompt-toolkit, 3.0.19
- ptyprocess, 0.7.0
- pybroom, 0.2
- pycparser, 2.20
- Pygments, 2.9.0
- pyinstaller, 4.3
- pyinstaller-hooks-contrib 2021.1
- pyparsing, 2.4.7
- pypiwin32, 223
- python-dateutil, 2.8.1
- pytz, 2021.1
- pywin32, 301
- pywin32-ctypes, 0.2.0
- requests, 2.25.1
- scipy, 1.6.0
- selenium, 3.141.0
- setuptools, 49.2.1
- sh, 1.14.2
- six, 1.15.0
- sympy, 1.7.1
- texttable, 1.6.3
- tk, 0.1.0
- traitlets, 5.0.5
- uncertainties, 3.1.5
- urllib3, 1.26.4
- virtualenv, 20.4.7
- wcwidth, 0.2.5
- wheel, 0.36.2
- whichcraft, 0.6.1
- zope.event, 4.5.0
- zope.interface, 5.4.0