

# PicoScope 5000 Series Data Acquisition Software

Guide d'utilisation et de compréhension

PRÉPARÉ PAR  
JOSHUA THEBAUDIN,  
STAGIAIRE PREMIÈRE  
ANNÉE BTS SIO.



# SOMMAIRE

- 01.** INTRODUCTION
- 02.** VISION GLOBALE DE L'APPLICATION
- 03.** VISION GLOBALE DU CODE (INTERFACE GRAPHIQUE)
- 04.** L'INTERFACE GRAPHIQUE
- 05.** VISION GLOBALE DU CODE (FONCTIONS)
- 06.** LES FONCTIONS
- 07.** NOTES

# Bienvenue

**sur ce guide d'utilisation et de compréhension.**

L'application "PicoScope 5000 Data Acquisition Software" a été développée afin de rendre plus simple et facile d'utilisation l'acquisition de données par le biais du PicoScope. À travers ce guide, nous permettrons aux utilisateurs de mieux comprendre l'application pour une utilisation optimale.

Ce guide traitera de l'application d'une façon globale, mais aussi d'une façon plus précise lorsque le besoin se fera sentir.

Cette dernière a été développée dans le cadre d'un stage, de ce fait, elle peut ne pas être parfaitement stable ou optimisée.

Veillez nous excuser des potentiels gênes ou difficulté que vous pourrez rencontrer.

Joshua.

# INTRODUCTION

# Vision globale de l'application

Regardons globalement l'application.

**Cette partie permet d'initialiser le PicoScope et donc son acquisition.**

On peut y paramétrer sa résolution, sa range, son sample et s'il y aura un filtre ou non. (détails plus loin dans ce guide.)

**Cette partie permet de paramétrer le Voltage Noise,**

il y a 3 paramètres : L'Amplification Factor, le Nplot et le F1 (les paramètres seront détaillés plus loin dans ce guide.)

**Cette partie permet de paramétrer le Frequency Noise,**

il y a 4 paramètres : Le Voltage, le FSR, le Nplot et le F1 (les paramètres seront détaillés plus loin dans ce guide.)



**Voici la partie où nous démarrons le PicoScope.** Le carré change de couleur en fonction de l'état du PicoScope (Vert signifie qu'il est allumé et rouge signifie le contraire).

**C'est ici que se font les différents calculs et affichages des graphiques** comme celui du Voltage Noise ainsi que son RMS. Des boutons de sauvegarde des données brutes sont aussi présents afin de pouvoir mieux travailler sur les données recueillies.

Cette application a été réalisée en **langage Python avec la librairie Tkinter**. Cette librairie permet de réaliser l'interface graphique que nous détaillons plus loin.

# Vision globale du code : interface graphique (1)

Regardons le code de l'interface graphique.

```
1 #----- IMPORTATION LIBRAIRIE -----#
2 from tkinter import *
3 from tkinter.messagebox import showinfo
4 from tkinter.ttk import Combobox
5 import tkinter.font
6 import importlib
7 D = importlib.import_module('Def_file')
8 from tkinter import *
9 import warnings
10 warnings.filterwarnings('ignore') #just use this if warnings are annoying
```

```
1 #----- CLASSE CONTENANT TOUS LES WIDGETS -----#
2 class Widget1():
3     def __init__(self, parent):
4         self.gul(parent)
5
6     def gul(self, parent):
7         if parent == 0:
8             self.wd = Tk()
9             self.wd.geometry('490x480')
10            self.wd.title('Picoscope 5000 Series Data Acquisition')
11            self.wd.resizable(0, 0)
12            self.wd.iconbitmap("Images/SILENTSYS_BLEU_PICTO_RVB-Web.ico")
13        else:
14            self.wd = Frame(parent)
15            self.wd.place(x = 0, y = 0, width = 490, height = 480)
16
17        #----- FRAME DU STATE -----#
18        self.group2 = LabelFrame(self.wd, text = "State", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow")
19        self.group2.place(x = 210, y = 10, width = 270, height = 210)
20
21        self.button0 = Button(self.group2, text = "OPEN PICO", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal", command=0.open_pico)
22        self.button0.place(x = 24, y = 20, width = 220, height = 20)
23
24        self.button1 = Button(self.group2, text = "CLOSE PICO", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal", command=0.close_pico)
25        self.button1.place(x = 24, y = 60, width = 220, height = 20)
26
27        D.led = self.state = Frame(self.group2, width = 100, height = 100, bg = "red")
28        self.state.place(x = 105, y = 100, width = 50, height = 50)
```

```
1 #----- FRAME DE L'INITIALISATION -----#
2 self.group3 = LabelFrame(self.wd, text = "Initialisation", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow")
3 self.group3.place(x = 10, y = 10, width = 200, height = 210)
4
5 #----- LISTE DEROULANTE RANGE -----#
6 D.range = self.combo1 = Combobox(self.group3, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
7 self.combo1.place(x = 100, y = 60, width = 90, height = 22)
8 self.combo1['values'] = ('10mV', '20mV', '50mV', '100mV', '200mV', '500mV', '1V', '2V', '5V', '10V')
9 self.combo1.current(0)
10 self.label1 = Label(self.group3, text = "Range :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
11 self.label1.place(x = 12, y = 60, width = 70, height = 22)
12
13 #----- LISTE DEROULANTE RESOLUTION -----#
14 self.label2 = Label(self.group3, text = "Resolution :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
15 self.label2.place(x = 7, y = 30, width = 70, height = 22)
16 D.reso = self.combo2 = Combobox(self.group3, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
17 self.combo2.place(x = 100, y = 30, width = 90, height = 22)
18 self.combo2['values'] = ('16Bits')
19 self.combo2.current(0)
20
21 #----- LISTE DEROULANTE SAMPLE -----#
22 self.label3 = Label(self.group3, text = "Sample :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
23 self.label3.place(x = 10, y = 90, width = 70, height = 22)
24 D.samp = self.combo3 = Combobox(self.group3, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
25 self.combo3.place(x = 100, y = 90, width = 90, height = 22)
26 self.combo3['values'] = ('64MS/s', '42MS/s', '32MS/s', '25MS/s', '20MS/s', '18MS/s', '16MS/s', '14MS/s', '12.5MS/s', '11MS/s', '10MS/s')
27 self.combo3.current(1)
28
29 #----- LISTE DEROULANTE FILTER -----#
30 self.label4 = Label(self.group3, text = "Filter :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
31 self.label4.place(x = 5, y = 120, width = 90, height = 22)
32 D.filter = self.combo4 = Combobox(self.group3, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
33 self.combo4.place(x = 100, y = 120, width = 90, height = 22)
34 self.combo4['values'] = ('ON', 'OFF')
35 self.combo4.current(0)
```

# Vision globale du code : interface graphique (2)

Regardons le code de l'interface graphique.

```
1 #----- BOUTON INITIALISATION -----#
2 self.buttons = Button(self.group, text = "INITIALISATION PICO", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal", command=self.init_pico)
3 self.buttons.place(x = 10, y = 160, width = 180, height = 22)
4
5 #----- FRAME DES VOLTAGE NOISE SETTINGS -----#
6 self.group = LabelFrame(self.w, text = "Voltage Noise Settings", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow")
7 self.group.place(x = 10, y = 220, width = 200, height = 110)
8
9 #----- AMPLIFACTOR ZONE -----#
10 self.label5 = Label(self.group, text = "Ampli-Factor :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
11 self.label5.place(x = 4, y = 10, width = 90, height = 22)
12
13 def click(event):
14     self.text5.delete(0, END)
15
16 self.text5 = Entry(self.group, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
17 self.text5.place(x = 110, y = 10, width = 80, height = 22)
18
19 self.text5.insert(0, 1)
20 self.text5.config(state=DISABLED)
21 self.text5.bind("<button-1>", click)
22
23 #----- NPLOT ZONE -----#
24 self.label6 = Label(self.group, text = "Nplots :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
25 self.label6.place(x = 17, y = 40, width = 90, height = 22)
26
27 def click(event):
28     self.text6.config(state=NORMAL)
29     self.text6.delete(0, END)
30
31 self.text6 = Entry(self.group, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
32 self.text6.place(x = 110, y = 40, width = 80, height = 22)
33
34 self.text6.insert(0, 1000)
35 self.text6.config(state=DISABLED)
36 self.text6.bind("<button-1>", click)
37
38 #----- F1 ZONE -----#
39 self.label7 = Label(self.group, text = "F1 :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
40 self.label7.place(x = 25, y = 70, width = 90, height = 22)
41
42 def click(event):
43     self.text7.config(state=NORMAL)
44     self.text7.delete(0, END)
45
46 self.text7 = Entry(self.group, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
47 self.text7.place(x = 110, y = 70, width = 80, height = 22)
48
49 self.text7.insert(0, 1162)
50 self.text7.config(state=DISABLED)
51 self.text7.bind("<button-1>", click)
52
```

```
1 #----- FRAME DE FREQUENCE NOISE SETTINGS -----#
2 self.group = LabelFrame(self.w, text = "Frequency Noise Settings", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow")
3 self.group.place(x = 10, y = 330, width = 200, height = 140)
4
5 #----- VOLTAGE ZONE -----#
6 self.label10 = Label(self.group, text = "Voltage :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
7 self.label10.place(x = 8, y = 10, width = 90, height = 22)
8
9 def click(event):
10     self.text10.config(state=NORMAL)
11     self.text10.delete(0, END)
12
13 self.text10 = Entry(self.group, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
14 self.text10.place(x = 110, y = 10, width = 80, height = 22)
15
16 self.text10.insert(0, 2)
17 self.text10.config(state=DISABLED)
18 self.text10.bind("<button-1>", click)
19
20 self.text10.place(x = 110, y = 10, width = 80, height = 22)
21
22 #----- FSR ZONE -----#
23 self.label17 = Label(self.group, text = "FSR :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
24 self.label17.place(x = 15, y = 40, width = 90, height = 22)
25
26 def click(event):
27     self.text17.config(state=NORMAL)
28     self.text17.delete(0, END)
29
30 self.text17 = Entry(self.group, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
31 self.text17.place(x = 110, y = 40, width = 80, height = 22)
32
33 self.text17.insert(0, 2.04 * 1e6)
34 self.text17.config(state=DISABLED)
35 self.text17.bind("<button-1>", click)
36
37 #----- NPLOT ZONE -----#
38 self.label18 = Label(self.group, text = "Nplot :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
39 self.label18.place(x = 15, y = 70, width = 90, height = 22)
40
41 def click(event):
42     self.text18.config(state=NORMAL)
43     self.text18.delete(0, END)
44
45 self.text18 = Entry(self.group, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
46 self.text18.place(x = 110, y = 70, width = 80, height = 22)
47
48 self.text18.insert(0, 1000)
49 self.text18.config(state=DISABLED)
50 self.text18.bind("<button-1>", click)
51
52 #----- F2 ZONE -----#
53 self.label19 = Label(self.group, text = "F2 :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
54 self.label19.place(x = 20, y = 100, width = 90, height = 22)
55
56 def click(event):
57     self.text19.config(state=NORMAL)
58     self.text19.delete(0, END)
59
60 self.text19 = Entry(self.group, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
61 self.text19.place(x = 110, y = 100, width = 80, height = 22)
62
63 self.text19.insert(0, 2162)
64 self.text19.config(state=DISABLED)
65 self.text19.bind("<button-1>", click)
66
```

```
1 #----- NPLOT ZONE -----#
2 self.label18 = Label(self.group, text = "Nplot :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
3 self.label18.place(x = 15, y = 70, width = 90, height = 22)
4
5 def click(event):
6     self.text18.config(state=NORMAL)
7     self.text18.delete(0, END)
8
9 self.text18 = Entry(self.group, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
10 self.text18.place(x = 110, y = 70, width = 80, height = 22)
11
12 self.text18.insert(0, 1000)
13 self.text18.config(state=DISABLED)
14 self.text18.bind("<button-1>", click)
15
16 #----- F2 ZONE -----#
17 self.label19 = Label(self.group, text = "F2 :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
18 self.label19.place(x = 20, y = 100, width = 90, height = 22)
19
20 def click(event):
21     self.text19.config(state=NORMAL)
22     self.text19.delete(0, END)
23
24 self.text19 = Entry(self.group, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
25 self.text19.place(x = 110, y = 100, width = 80, height = 22)
26
27 self.text19.insert(0, 2162)
28 self.text19.config(state=DISABLED)
29 self.text19.bind("<button-1>", click)
30
```

# L'INTERFACE GRAPHIQUE (1)

L'interface graphique à été réalisée sur Tkinter. Dans cette partie nous détaillerons le code permettant de la faire. L'interface se trouve dans un fichier python dédié.

```
1 #----- IMPORTATION LIBRAIRIE -----#
2 from tkinter import *
3 from tkinter.messagebox import showinfo
4 from tkinter.ttk import Combobox
5 import tkinter.font
6 import importlib
7 D = importlib.import_module('Def_file')
8 from tkinter import *
9 import warnings
10 warnings.filterwarnings('ignore') #just use this if warnings are annoying
```

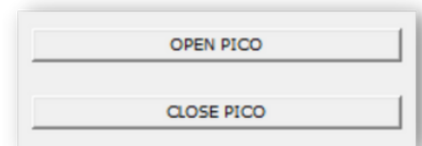
Présente au début du code, **l'importation de librairies est une partie très importante car elle permet de faire fonctionner correctement l'application.** On y retrouve Tkinter et ses modules qui sont très important pour notre interface : **Showinfo** permet de créer des pop-ups, **Combobox** permet de faire des listes déroulantes. On trouve aussi la librairie **importlib** permettant de faire le lien entre plusieurs fichiers python. **Ici la variable D contient l'importation du fichier des fonctions**

# L'INTERFACE GRAPHIQUE (2)

```
1 ----- CLASSE CONTENANT TOUS LES WIDGETS -----#
2 class Widget1():
3     def __init__(self, parent):
4         self.gui(parent)
5
6     def gui(self, parent):
7         if parent == 0:
8             self.w1 = Tk()
9             self.w1.geometry('490x480')
10            self.w1.title('PicoScope 5000 Series Data Acquisition')
11            self.w1.resizable(0, 0)
12
13            self.w1.iconbitmap("images/SILENTSYS_BLEU_PICTO_RVB-Web.ico")
14
15        else:
16            self.w1 = Frame(parent)
17            self.w1.place(x = 0, y = 0, width = 490, height = 480)
18
19        #----- FRAME DU STATE -----#
20        self.group2 = LabelFrame(self.w1, text = "State", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow")
21        self.group2.place(x = 210, y = 10, width = 270, height = 210)
22
23        self.button10 = Button(self.group2, text = "OPEN PICO", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal", command=D.open_pico)
24        self.button10.place(x = 24, y = 20, width = 220, height = 20)
25
26
27
28        self.button11 = Button(self.group2, text = "CLOSE PICO", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal", command=D.close_pico)
29        self.button11.place(x = 24, y = 60, width = 220, height = 20)
30
31
32
33        D.led = self.state = Frame(self.group2, width = 100, height = 100, bg = "red")
34
35        self.state.place(x = 105, y = 100, width = 50, height = 50)
```

La **def gui** permet de configurer la fenêtre de l'application et les éléments en son sein, on y définit sa taille, son titre, si on peut l'agrandir ou non ainsi que l'icône de l'application.

Le **frame du state** correspond à la partie **State** de l'application. On y retrouve les boutons OPEN et CLOSE PICO. Dans ses boutons, il y a la partie "command:..." suivie de "D.nom de la fonction correspondante". Cela correspond à la liaison avec le fichier des fonctions.



Le **D.led** correspond au carré rouge que l'on peut voir sur l'interface. Cette variable est relié au fichier des fonctions et de ce fait, selon l'état du PicoScope, la couleur change.





# L'INTERFACE GRAPHIQUE (3)

```
1 #----- FRAME DE L'INITIALISATION -----#
2 self.group3 = LabelFrame(self.w1, text = "Initialisation", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow")
3 self.group3.place(x = 10, y = 10, width = 200, height = 210)
4
5 #----- LISTE DEROULANTE RANGE -----#
6 D.range = self.combo1 = Combobox(self.group3, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
7 self.combo1.place(x = 100, y = 60, width = 90, height = 22)
8 self.combo1['values'] = ("10MV", "20MV", "50MV", "100MV", "200MV", "500MV", "1V", "2V", "5V", "10V")
9 self.combo1.current(3)
10 self.label1 = Label(self.group3, text = "Range :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
11 self.label1.place(x = 12, y = 60, width = 70, height = 22)
12
13 #----- LISTE DEROULANTE RESOLUTION -----#
14 self.label2 = Label(self.group3, text = "Resolution :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
15 self.label2.place(x = 7, y = 30, width = 70, height = 22)
16 D.reso = self.combo2 = Combobox(self.group3, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
17 self.combo2.place(x = 100, y = 30, width = 90, height = 22)
18 self.combo2['values'] = ("16Bits")
19 self.combo2.current(0)
20
21 #----- LISTE DEROULANTE SAMPLE -----#
22 self.label3 = Label(self.group3, text = "Sample :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
23 self.label3.place(x = 10, y = 90, width = 70, height = 22)
24 D.sample = self.combo3 = Combobox(self.group3, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
25 self.combo3.place(x = 100, y = 90, width = 90, height = 22)
26 self.combo3['values'] = ("64MS/s", "42MS/s", "32MS/s", "25MS/s", "20MS/s", "18MS/s", "16MS/s", "14MS/s", "12.5MS/s", "11MS/s", "10MS/s")
27 self.combo3.current(1)
28
29 #----- LISTE DEROULANTE FILTER -----#
30 self.label4 = Label(self.group3, text = "Filter :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
31 self.label4.place(x = 5, y = 120, width = 90, height = 22)
32 D.filter = self.combo4 = Combobox(self.group3, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
33 self.combo4.place(x = 100, y = 120, width = 90, height = 22)
34 self.combo4['values'] = ("ON", "OFF")
35 self.combo4.current(0)
36
37 #----- BOUTON INITIALISATION -----#
38 self.button5 = Button(self.group3, text = "INITIALISATION PICO", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal", command=D.init_pico)
39 self.button5.place(x = 10, y = 160, width = 180, height = 22)
```

Dans cet extrait, on établit la frame de l'initialisation. On met en place 4 listes déroulantes (réalisées par le module ComboBox).

Dans self.combo1 et dans les autres on définit entre crochets les différentes valeurs des différents paramètres.

Le self.combo1.current et les autres, on définit la valeur par défaut par son index dans le tableau (exemple : le 3 dans la première liste correspond à 100Mv)

Puis, toutes ces valeurs sont envoyées par le bouton INITIALISATION.



# L'INTERFACE GRAPHIQUE (4)

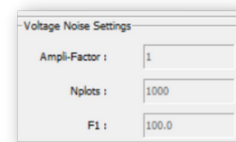
```
1 #----- FRAME DES VOLTAGE NOISE SETTINGS -----#
2 self.group6 = LabelFrame(self.w1, text = "Voltage Noise Settings", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow")
3 self.group6.place(x = 10, y = 220, width = 200, height = 110)
4
5 #----- AMPLIFACTOR ZONE -----#
6 self.label5 = Label(self.group6, text = "Ampli-Factor :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
7 self.label5.place(x = 4, y = 10, width = 90, height = 22)
8
9 def click(event): #PERMET DE METTRE EN PLACE LE PLACEHOLDER DANS LE TEXTE (1), CELA SE REPETERA POUR CHAQUE CHAMP DE TEXTE
10     self.ltext5.delete(0, END)
11
12 self.ltext5 = Entry(self.group6, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
13 self.ltext5.place(x = 110, y = 10, width = 80, height = 22)
14
15 self.ltext5.insert(0,1) #PERMET DE METTRE EN PLACE LE PLACEHOLDER DANS LE TEXTE (2), CELA SE REPETERA POUR CHAQUE CHAMP DE TEXTE, 1 EN VALEUR PAR DEFAULT
16 self.ltext5.config(state=DISABLED)
17 self.ltext5.bind("<Button-1>", click)
18
19 #--- NPLOT ZONE -----#
20 self.label6 = Label(self.group6, text = "Nplots :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
21 self.label6.place(x = 17, y = 40, width = 90, height = 22)
22
23 def click(event):
24     self.ltext6.config(state=NORMAL)
25     self.ltext6.delete(0, END)
26
27 self.ltext6 = Entry(self.group6, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
28 self.ltext6.place(x = 110, y = 40, width = 80, height = 22)
29
30 self.ltext6.insert(0, 1000) #1000 EN VALEUR PAR DEFAULT
31 self.ltext6.config(state=DISABLED)
32 self.ltext6.bind("<Button-1>", click)
33
34 #--- F1 ZONE -----#
35 self.label7 = Label(self.group6, text = "F1 :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
36 self.label7.place(x = 25, y = 70, width = 90, height = 22)
37
38 def click(event):
39     self.ltext7.config(state=NORMAL)
40     self.ltext7.delete(0, END)
41
42 self.ltext7 = Entry(self.group6, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
43 self.ltext7.place(x = 110, y = 70, width = 80, height = 22)
44
45 self.ltext7.insert(0,1*1e2) #1*1e2 EN VALEUR PAR DEFAULT
46 self.ltext7.config(state=DISABLED)
47 self.ltext7.bind("<Button-1>", click)
48
49
```

Ici, on met en place la frame des paramètres pour le Voltage Noise. C'est une simple zone de texte dans laquelle l'utilisateur rentre ce qu'il veut.

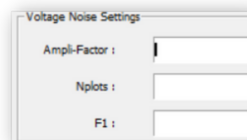
Cependant, ici, il y a des valeurs par défauts qui sont mis en place par le **def click(event)** : on initialise le champ de texte dans un état normal (grisé) qui contient la valeur mise dans le insert (exemple : `self.ltext6.insert(0,1000)`).

Mais lorsque l'on clique sur la zone de texte, la valeur par défaut est supprimé et l'utilisateur peut donc rentrer la valeur qu'il souhaite.

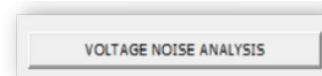
Tous ces paramètres sont envoyés et sont pris en compte grâce au bouton VOLTAGE NOISE ANALYSIS



Voltage Noise Settings	
Ampli-Factor :	1
Nplots :	1000
F1 :	100.0



Voltage Noise Settings	
Ampli-Factor :	
Nplots :	
F1 :	



# L'INTERFACE GRAPHIQUE (5)

```
1 #----- FRAME DU FREQUENCY NOISE SETTINGS -----#
2 self.group4 = LabelFrame(self.w1, text = "Frequency Noise Settings ", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow")
3 self.group4.place(x = 10, y = 330, width = 200, height = 140)
4
5 #----- VOLTAGE_ZONE -----#
6 self.label5 = Label(self.group4, text = "Voltage :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
7 self.label5.place(x = 0, y = 10, width = 90, height = 22)
8
9 def click(event):
10     self.ltext1.config(state=NORMAL)
11     self.ltext1.delete(0, END)
12
13 self.ltext1 = Entry(self.group4, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
14
15 self.ltext1.insert(0, 2) #2 EN VALEUR PAR DEFAULT
16 self.ltext1.config(state = DISABLED)
17 self.ltext1.bind("<Button-1>", click)
18
19 self.ltext1.place(x = 110, y = 10, width = 80, height = 22)
20
21 #----- FSR_ZONE -----#
22 self.label7 = Label(self.group4, text = "FSR :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
23 self.label7.place(x = 15, y = 40, width = 90, height = 22)
24
25 def click(event):
26     self.ltext2.config(state=NORMAL)
27     self.ltext2.delete(0, END)
28
29 self.ltext2 = Entry(self.group4, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
30 self.ltext2.place(x = 110, y = 40, width = 80, height = 22)
31
32 self.ltext2.insert(0, 2.04 * 1e6) #2.04 * 1e6 EN VALEUR PAR DEFAULT
33 self.ltext2.config(state=DISABLED)
34 self.ltext2.bind("<Button-1>", click)
35
36 #----- NPLOT_ZONE -----#
37 self.label8 = Label(self.group4, text = "Nplot :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
38 self.label8.place(x = 15, y = 70, width = 90, height = 22)
39
40 def click(event):
41     self.ltext3.config(state=NORMAL)
42     self.ltext3.delete(0, END)
43
44 self.ltext3 = Entry(self.group4, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
45 self.ltext3.place(x = 110, y = 70, width = 80, height = 22)
46
47 self.ltext3.insert(0, 1000) #1000 EN VALEUR PAR DEFAULT
48 self.ltext3.config(state=DISABLED)
49 self.ltext3.bind("<Button-1>", click)
50
51 #----- F1_ZONE -----#
52 self.label9 = Label(self.group4, text = "F1 :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
53 self.label9.place(x = 20, y = 100, width = 90, height = 22)
54
55 def click(event):
56     self.ltext4.config(state=NORMAL)
57     self.ltext4.delete(0, END)
58
59 self.ltext4 = Entry(self.group4, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
60 self.ltext4.place(x = 110, y = 100, width = 80, height = 22)
61
62 self.ltext4.insert(0, 2*1e2) #2*1e2 EN VALEUR PAR DEFAULT
63 self.ltext4.config(state=DISABLED)
64 self.ltext4.bind("<Button-1>", click)
65
66
```

Pour le Frequency Noise, le système fonctionne de la même manière et les paramètres sont envoyés au bouton Frequency Noise Analysis.

Frequency Noise Settings	
Voltage :	<input type="text" value="2"/>
FSR :	<input type="text" value="2040000.0"/>
Nplot :	<input type="text" value="1000"/>
F1 :	<input type="text" value="200.0"/>

Frequency Noise Settings	
Voltage :	<input type="text"/>
FSR :	<input type="text"/>
Nplot :	<input type="text"/>
F1 :	<input type="text"/>

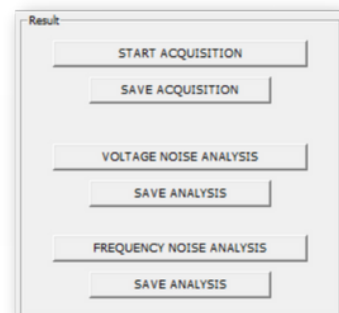
FREQUENCY NOISE ANALYSIS

# L'INTERFACE GRAPHIQUE (6)

```
1 #----- FRAME DES RESULTATS -----#
2 self.group5 = LabelFrame(self.w, text = "Result", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow")
3 self.group5.place(x = 215, y = 220, width = 265, height = 250)
4
5 #----- START ACQUI BUTTON -----#
6 self.button7 = Button(self.group5, text = "START ACQUISITION", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal", command=D.startacqui)
7 self.button7.place(x = 25, y = 10, width = 210, height = 22)
8
9
10
11 #----- SAVE ACQUI BUTTON -----#
12 self.button12 = Button(self.group5, text = "SAVE ACQUISITION", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal", command=D.saveacqui)
13 self.button12.place(x = 55, y = 40, width = 150, height = 22)
14
15 #----- VOLTAGE NOISE BUTTON -----#
16 self.button8 = Button(self.group5, text = "VOLTAGE NOISE ANALYSIS", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal", command=lambda: D.vnoise(self.itext5, self.itext6, self.itext7))
17 self.button8.place(x = 25, y = 95, width = 210, height = 22)
18
19 #----- SAVE ANALYSIS BUTTON -----#
20 self.button13 = Button(self.group5, text = "SAVE ANALYSIS", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal", command=D.sauvegardevoie)
21 self.button13.place(x = 55, y = 125, width = 150, height = 22)
22
23 #----- FREQUENCY NOISE RESULT -----#
24 self.button14 = Button(self.group5, text = "FREQUENCY NOISE ANALYSIS", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal", command=lambda: D.frequency(self.itext1, self.itext2, self.itext3, self.itext4))
25 self.button14.place(x = 25, y = 170, width = 210, height = 22)
26
27 #----- SAVE ANALYSIS BUTTON -----#
28 self.button15 = Button(self.group5, text = "SAVE ANALYSIS", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal", command=D.savefreq)
29 self.button15.place(x = 55, y = 200, width = 150, height = 22)
```

Dans cette partie du code, nous mettons en place les différents boutons appelant les différentes fonctions associées.

Pour le bouton Voltage Noise et Frequency Noise, command est différente car elles dépendent de champs de texte. Le **lambda**: permet de récupérer toutes les valeurs dans les itext et de les exécuter en même temps.



```
1 if __name__ == '__main__':
2     print("-----Welcome to the PicoScope 5000 Series Data Acquisition Software-----")
3
4     a = Widget1(0)
5     showinfo("Welcome", "Welcome to the PicoScope 5000 Series Data Acquisition Software")
6     a.w1.mainloop()
```

Cette partie du code permet d'afficher toute cette interface graphique.

# Vision globale du code : les fonctions (1)

Regardons le code des fonctions

```
1 #---- IMPORTATION (1) ----#
2 from tkinter import *
3 from tkinter.ttk import *
4 import time
5 from tkinter import N, filedialog, END, TOPLEVEL
6 from tkinter.messagebox import showinfo
7 from tkinter.ttk import Label, Entry, Text, Button
8 import importlib
9
10 P = importlib.import_module('PicoScope 5000 Series Data Acquisition Software')
11
12 #----GLOBAL VARIABLES----#
13 global led, reso, range, sample, filter, noise, status_pico, U_meas, status_init_pico, f_plot1_vnoise, PSD_SSR_plot1_vnoise, U_meas0, v_pico, i_pico, s_pico, vnoise_open, acqui_pico, PSD_SSR_plot1, f_plot1, freq_pico
14
15 #----INITIALIZATION----#
16 noise0 = 0
17 nbaqui = 0
18 nbfreq = 0
19 nbs00 = 0
20 noise = None
21 status_pico = False
22 freq_pico = False
23 i_pico = False
24 s_pico = False
25 save_pico = False
26 v_pico = False
27 vnoise_open = False
28 acqui_pico = False
29
30 #----IMPORTATION (2)----#
31 import matplotlib.pyplot as plt
32 import numpy as np
33 import scipy as sp
34 import scipy.integrate as spi
35 import time
36 import warnings
37 warnings.filterwarnings('ignore') # Just use this if warnings are annoying
38
39 import ctypes
40 from picosdk.pico5000 import ps5000a as ps
41 from picosdk.functions import adc2mv, assert_pico_ok, HV2dc
```

```
1 class PicoScope_ChannelA:
2     def __init__(self, Samples_per_channel=int(1e4), trigger_level_mv=0):
3         Samples_per_channel = int(Samples_per_channel)
4         self.trigger_level_mv = trigger_level_mv
5         self.channel = ctypes.c_int16(0)
6         self.status = {}
7         self.resolution = ps.PSS000A_DEVICE_RESOLUTION["PSS000A_OR_16BIT"]
8         self.status["openunit"] = ps.ps5000aOpenUnit(ctypes.byref(self.channel), None, self.resolution)
9         # PICO_STATUS_PSS000A_OPENUNIT_SUCCESS == handle
10         # int16_t serial, None means it just takes the first scope found
11         # PSS000A_DEVICE_RESOLUTION
12         assert_pico_ok(self.status["openunit"])
13
14         # print('PicoScope is open now')
15
16         # Disable other channels
17         # Set up channel B
18         # handle = channel
19         self.channelB = ps.PSS000A_CHANNEL["PSS000A_CHANNEL_B"]
20         # enabled = 1
21         self.coupling_type = ps.PSS000A_COUPLING["PSS000A_DC"]
22         # self.coupling_type = ps.PSS000A_COUPLING["PSS000A_AC"]
23         # Typed? enum psPSS000ACoupling(PSS000A_AC, PSS000A_DC) PSS000A_COUPLING
24         self.chRange = ps.PSS000A_RANGE["PSS000A_1V"] # It was 20V before
25         # self.chRange = ps.PSS000A_RANGE["PSS000A_200mV"] # It was 20V before
26         # Typed? enum psPSS000ARange(
27         #     PSS000A_1mV,
28         #     PSS000A_200mV,
29         #     PSS000A_500mV,
30         #     PSS000A_100mV,
31         #     PSS000A_200mV,
32         #     PSS000A_500mV,
33         #     PSS000A_1V,
34         #     PSS000A_2V,
35         #     PSS000A_5V,
36         #     PSS000A_10V,
37         #     PSS000A_20V,
38         #     PSS000A_50V,
39         #     PSS000A_100V,
40         #     PSS000A_200V,
41         #     PSS000A_500V,
42         #     PSS000A_1000V,
43         #     PSS000A_RANGE)
44         # Last entry is analogue offset = 0 V
45         self.status["setCHB"] = ps.ps5000aSetChannel(self.channel, self.channelB, 0, self.coupling_type, self.chRange,
46         #     0,
47         #     PICO_STATUS_PSS000aSetChannel(int16_t handle, PSS000A_CHANNEL channel, int16_t enabled, PSS000A_COUPLING type,
48         #     PSS000A_RANGE range, float analogueOffset)
49         # assert_pico_ok(self.status["setCHB"])
50
51         # Set up channel C
52         # handle = channel
53         self.channelC = ps.PSS000A_CHANNEL["PSS000A_CHANNEL_C"]
54         # enabled = 1
55         self.coupling_type = ps.PSS000A_COUPLING["PSS000A_DC"]
56         # self.coupling_type = ps.PSS000A_COUPLING["PSS000A_AC"]
57         # Typed? enum psPSS000ACoupling(PSS000A_AC, PSS000A_DC) PSS000A_COUPLING
58         self.chRange = ps.PSS000A_RANGE["PSS000A_1V"] # It was 20V before
59         # self.chRange = ps.PSS000A_RANGE["PSS000A_200mV"] # It was 20V before
60         # Typed? enum psPSS000ARange(
61         #     PSS000A_1mV,
62         #     PSS000A_200mV,
63         #     PSS000A_500mV,
64         #     PSS000A_100mV,
65         #     PSS000A_200mV,
66         #     PSS000A_500mV,
67         #     PSS000A_1V,
68         #     PSS000A_2V,
69         #     PSS000A_5V,
70         #     PSS000A_10V,
71         #     PSS000A_20V,
72         #     PSS000A_50V,
73         #     PSS000A_100V,
74         #     PSS000A_200V,
75         #     PSS000A_500V,
76         #     PSS000A_1000V,
77         #     PSS000A_RANGE)
78         # Last entry is analogue offset = 0 V
79         self.status["setCHC"] = ps.ps5000aSetChannel(self.channel, self.channelC, 0, self.coupling_type, self.chRange,
80         #     0,
```

Cette partie du code étant longue, nous ne mettrons qu'une partie, nous y reviendrons après pour y définir son rôle juste après.

# Vision globale du code : les fonctions (2)

Regardons le code des fonctions

```
1 class ProcessNoise(object): # TODO live class
2     """
3     Class used for a adding live update to a figure. Run 'matplotlib notebook' in jupyter notebook
4     for optimal operation of this class. This adds
5
6     Attributes
7     """
8     needs explanation"""
9
10 def __init__(self, init_ps=True, Nsamples=int(2**25)):
11     """
12     """
13     #Initialize Picoscope
14     """
15     self.Nsamples = Nsamples
16     if init_ps:
17         try:
18             self.picoscope = Picoscope_channelA(Samples_per_channel=self.Nsamples)
19
20         except:
21             Picoscope_channelA.close() # check if self needed here
22             self.picoscope = Picoscope_4channels(Samples_per_channel=self.Nsamples)
23             print('Picoscope was already open and was now reinitialized.')
24
25     self.picoscope.N = self.Nsamples
26     self.picoscope.set_timebase(5) # for 128T 2 is 4ns Sampling interval => 250MS/s, 3 is 8ns => 125MS/s
27     # check on page 22 in programmers-guide 3Width 15 bit 3 is 125MS/s, 4 is 64MS/s, 5 is 42 MS/s, 6 is 32 MS/s, 7 is 25 MS/s,
28     # 8 is 20 MS/s, 9 is 16MS/s
29     # 10 is 16 MS/s, 11 is 14 MS/s, 12 is 12.5 MS/s, 13 is 11 MS/s, 14 is 10 MS/s, 17 is 7 MS/s, 18 is 6 MS/s, 19 is 5 MS/s
30     self.picoscope.set_bandwidthfilter(1) # this sets the 20MHz lowpassfilter for all channels
31     self.picoscope.N = 1 if (self.picoscope.timeinterval.value * 1e-9) # sampling rate
32     self.times = sp.linspace(0, (self.picoscope.N - 1) / self.picoscope.R,
33                             self.picoscope.N) # check if this will be needed for the future
34
35     self.Noise_start_acquisition() # check if this will be needed for the future
36     self.picoscope.check_ready() # this is needed to write the data to the buffer for the readout.
37     # here we could save time to evaluate while writing into the buffer at the same time
38     time.sleep(1) # this is to make sure that the AC coupling has time to stabilize
39     self.Noise_start_acquisition()
40     self.picoscope.check_ready() # this is needed to write the data to the buffer for the readout.
41     self.picoscope.readout_buffer()
42
43     self.time = sp.linspace(0, (self.picoscope.N - 1) / self.picoscope.R, self.picoscope.N)
44     self.signal_A = sp.array(self.picoscope.bufferA)
45
46     # self.N_FFT = Nsamples
47     # self.dt = 1/self.picoscope.R
48     # self.freq = sp.fftfreq(self.N_FFT, self.dt)
49     # self.signal_FFT = sp.zeros(self.N_FFT)
50     # self.signal_FFT_phase = sp.zeros(self.N_FFT)
51     # self.signal_FFT_ampitude = sp.ones(self.N_FFT)
52
53 def Noise_start_acquisition(self):
54     self.picoscope.start_acquiredata_triggered() # this writes the data into the buffer
55
56 def update_data(self):
57     self.Noise_start_acquisition() # this starts the acquisition
58     self.picoscope.check_ready() # this is needed to write the data to the buffer for the readout.
59     # here we could save time to evaluate while writing into the buffer at the same time
60     self.picoscope.readout_buffer()
61     self.signal_A = sp.array(
62         sp.zeros(self.picoscope.bufferA, self.picoscope.chRange, self.picoscope.maxADC) * 1e-3)
63     self.signal_NoiseI = self.signal_A
64
65 def save_data(self, fn):
66     data = sp.transpose([self.time, self.signal_NoiseI])
67     sp.savetxt(fn, data, fmt='%0.6e', header='time (s), signal A (V)', delimiter=',', newline='\n')
68
69 def save_dataram(self, fn):
70     data = sp.transpose([self.time, self.signal_NoiseI])
71     sp.savetxt(fn, data, fmt='%0.6e', header='time (s), signal A (V)', delimiter=',', newline='\n')
72
73 def load_rawdata(self, fn): # speed up this function todo
74     self.time, self.signal_A = sp.loadtxt(fn, unpack=True, delimiter=',', skiprows=1)
```

```
1 def open_pico(): # work
2     global status_pico, noise
3     noise = ProcessNoise(init_ps=True, Nsamples=int(2**25)) # up to 2^25 should be ok for the PS5444D;
4     status_pico = True
5     if status_pico == True:
6         led.config(bg='green')
7         return showinfo('Great', 'Picoscope is open')
8
9
10 # 2^24 are too many samples for buffering for the PS5442D used for testing.
11 # execute cell with noise.picoscope.close() below if an error appears
12
13 def close_pico(): # work
14     global status_pico, s_pico, i_pico, v_pico, vnoise_open, acqui_pico, freq_pico
15     noise.picoscope.close()
16     status_pico = False
17     if status_pico == False:
18         led.config(bg='red')
19         return showinfo('Warning', 'Picoscope is closed')
20     # quand on ferme le picoscope, on réinitialise les pupups
21     s_pico = False
22     i_pico = False
23     v_pico = False
24     vnoise_open = False
25     acqui_pico = False
26     freq_pico = False
27
```

# Vision globale du code : les fonctions (3)

Regardons le code des fonctions

```
1 def init_pico():
2     global i_pico
3
4     if status_pico == False:
5         return showinfo ('Error', 'Picoscope is not open')
6
7     range_tab = [
8         # tableau associatif pour les valeurs de range
9         ('name': '10mV', 'value': 0),
10        ('name': '20mV', 'value': 1),
11        ('name': '50mV', 'value': 2),
12        ('name': '100mV', 'value': 3),
13        ('name': '200mV', 'value': 4),
14        ('name': '500mV', 'value': 5),
15        ('name': '1V', 'value': 6),
16        ('name': '2V', 'value': 7),
17        ('name': '5V', 'value': 8),
18        ('name': '10V', 'value': 10),
19        ('name': '20V', 'value': 11),
20        ('name': '50V', 'value': 12),
21    ]
22    timebase_tab = [
23        # tableau associatif pour les valeurs de timebase
24
25        ('name': '64MS/s', 'value': 4),
26        ('name': '42MS/s', 'value': 5),
27        ('name': '32MS/s', 'value': 6),
28        ('name': '25MS/s', 'value': 7),
29        ('name': '20MS/s', 'value': 8),
30        ('name': '18MS/s', 'value': 9),
31        ('name': '16MS/s', 'value': 10),
32        ('name': '14MS/s', 'value': 11),
33        ('name': '12.5MS/s', 'value': 12),
34        ('name': '11MS/s', 'value': 13),
35        ('name': '10MS/s', 'value': 14),
36    ]
37
38    filter_tab = [
39        # tableau associatif pour les valeurs de filter
40        ('name': 'OFF', 'value': 1),
41        ('name': 'ON', 'value': 0),
42    ]
43
44    resolution_tab = [
45        # tableau associatif pour les valeurs de resolution
46        ('name': '16bits', 'value': 16)
47    ]
48
49    for r in resolution_tab:
50        if r['name'] == reso.get():
51            getreso = r['value']
52            print ("reso : " + str (getreso))
53
54    for i in range_tab:
55        if i['name'] == range.get():
56            getrange = i['value']
57            print ("range : " + str (getrange))
58
59    for j in timebase_tab:
60        if j['name'] == sample.get():
61            gettime = j['value']
62            print ("time : " + str (gettime))
63
64    for x in filter_tab:
65        if x['name'] == filter.get():
66            getfilter = x['value']
67            print ("filter : " + str (getfilter))
68
69    # save the values in the picoscope
70    noise.picoscope.set_timebase (gettime)
71    noise.picoscope.set_range (getrange)
72    noise.picoscope.set_resolution (getreso)
73    noise.picoscope.set_bandwidthfilter (getfilter)
74    i_pico = True
75    showinfo ('Info', 'Picoscope is initialized')
76
77
```

```
1 def startacqui():
2     global U_meas, s_pico, U_meas0, v_pico, acqui_pico, nbaacqui
3
4     nbaacqui=nbaacqui+1
5
6     if status_pico == False:
7         return showinfo ('Warning', 'Picoscope is not open, please open it')
8     if i_pico == False:
9         return showinfo ('Warning', 'Picoscope is not initialized, please initialize the picoscope')
10    if i_pico == True:
11        showinfo ('Info', 'Acquisition is started. This can take a while. After that you can save the data and launch the Voltage Noise Acquisition. ')
12
13        v_pico = True
14
15        noise.update_data ()
16
17        U_meas0 = noise.signal_noise1
18        U_meas0.max ()
19
20        U_meas0.min ()
21
22        U_meas = (U_meas0 - U_meas0.mean ())
23
24        fig0 = plt.figure ()
25        plt.plot (U_meas0)
26
27        plt.autoscale (enable=True, axis='y', tight=True)
28        plt.title ("Time Trace " + str(nbaacqui))
29        plt.ylabel ("Tension (V)")
30
31        plt.xlabel ("Npts (s)")
32
33        acqui_pico = True
34        plt.show ()
```

```
1 def saveacqui():
2     global U_meas, s_pico, U_meas0, v_pico, acqui_pico
3
4     #si l'acquisition n'est pas lancée, on ne peut pas sauvegarder
5     if v_pico == False:
6         return showinfo ('Warning', 'Acquisition is not started, please start the acquisition')
7
8     #si on ne lance pas l'acquisition, on ne peut pas sauvegarder
9     if acqui_pico == False:
10        return showinfo ('Warning', 'Acquisition is not started, please start the acquisition before saving datas')
11
12
13    #si le picoscope n'est pas ouvert
14    if status_pico == False:
15        return showinfo ('Warning', 'Picoscope is not open, please open it')
16    #si le picoscope n'est pas initialisé
17    if i_pico == False:
18        return showinfo ('Warning', 'Picoscope is not initialized, please initialize the picoscope')
19
20
21
22    global U_meas0
23    path = filedialog.askdirectory (initialdir = "/", title = "Select directory")
24
25
26    #choose the name of the txt when we are in the folder
27    name = askstring ("Name of the txt", "Name of the txt")
28
29    if name == None:
30        return showinfo ("Warning", "You did not enter a name")
31    if path != None:
32        data2 = sp.transpose([U_meas0])
33        np.savetxt(path + '/' + name + '.txt', data2, delimiter='\t', fmt='%0.6e')
34        showinfo ('Info', 'Txt saved')
```

# Vision globale du code : les fonctions (4)

Regardons le code des fonctions

```
def noise(amplfact, Nplot, f1):  
    # si le picoscope n'est pas ouvert  
    if status_pico == False:  
        return showInfo('Warning', 'Picoscope is not open, please open it')  
    # si le picoscope n'est pas initialisé  
    if _f_pico == False:  
        return showInfo('Warning', 'Picoscope is not initialized, please initialize the picoscope')  
    # si le picoscope n'est pas en cours d'acquisition  
    if _capture == False:  
        return showInfo('Warning', 'Picoscope is not started, please start the acquisition')  
  
    if amplfact.get() == '' and Nplot.get() == '' and f1.get() == '':  
        return showInfo('Warning', 'You did not enter any values')  
    if amplfact.get() == '':  
        return showInfo('Warning', 'You did not enter a value for the amplification factor')  
    # si le champ Nplot est vide  
    if Nplot.get() == '':  
        return showInfo('Warning', 'You did not enter a value for the number of points')  
    # si f1.get() == '':  
        return showInfo('Warning', 'You did not enter a value for the frequency')  
    # si tous les champs sont vides  
    # si le voltage noise est lancé  
    if _v_pico == True:  
        showInfo('Info', 'Voltage Noise Analysis and RMS is started, this can take a while. After that you can save the data...')  
  
    global f_plot1_noise, PSD_SSB_plot1_noise, noise_open, noise, noiseRMS  
  
    noise = noise + 1  
    noiseRMS = noiseRMS + 1  
    N = noise_picoscope.N  
    n = noise_picoscope.n  
    # t = 1 / f # Time between 2 samples [s]  
    # t = 1 / f # Total measurement time [s]  
    # f = 1 / t # Frequency sampling [Hz]  
    f = np.arange(1, N + 1) # f / axis  
    #  
  
    amplfact = int(amplfact.get())  
    Nplot = int(Nplot.get())  
    f1 = float(f1.get())  
  
    # amplificationfactor = cension * 2 * 3.14 / (2.84 * 10^4) # for the conversion from V to Hz - by adjusting optical power to reach +/- "tension" signal  
    # amplificationfactor = amplfact # for the voltage amplification  
    # W = 1 / amplificationfactor * U_rms  
    # W = np.absolute(coeff_err * 60)  
    PSD_SSB = abs(int((N / 2) ** 2) / ( # Power Spectral Density (W/Hz) This is the SSB (single sidedband, one-sided only positive frequencies) as displayed and saved in the RMS PSD  
    f = [int(N / 2)]  
  
    TIC = Time.time()  
    # N_plot = 2000 # for points above f1  
    # f1 = 2 # Hz  
    # N_plot = Nplot # for points below f1  
    f1 = f1  
    f_max = np.logspace(np.log10(f1), np.log10(f_max()), N_plot + 1)  
    f_plot = (f_max[1] + f_max[-1]) / 2  
    PSD_SSB_plot = np.zeros_like(f_plot)  
    for i1, f_plot_1 in enumerate(f_plot):  
        PSD_SSB_plot[i1] = PSD_SSB(f = f_max[i1]) * (f_max[i1 + 1] - f_max[i1])  
    toc = Time.time() - tic  
    print(toc)  
  
    f_plot1_noise = np.append(f1, f_plot)  
    PSD_SSB_plot1_noise = np.append(PSD_SSB(f = f1), PSD_SSB_plot)  
  
    fig1_ax1 = plt.subplots(1)  
    silentsys_ver = '322 / 255 / 186 / 255 / 76 / 255 / 3'  
    silentsys_range = '254 / 255 / 27 / 255 / 0 / 255 / 1'  
    ax1.legend([f'Plot1:noise', np.next(PSD_SSB_plot1_noise) * 10^9, color=silentsys_range])  
    ax1.set_xlabel('f [Hz]', color=silentsys_ver)  
    ax1.set_ylabel('PSD [dBm/Hz]', color=silentsys_ver)  
    ax1.set_xlim(100, 10^4)  
    ax1.set_ylim(140, 160)  
    ax1.set_xlabel('Voltage Noise PSD (dBm/Hz)')  
    ax1.set_ylabel('Fourier Frequency (Hz)')  
    img = plt.imread('images/SILENTSYS_VERT_Pantone_18pc.png')  
    fig1.tight_layout()  
  
    plt.title('Voltage Noise + str(noise)')  
    noise = plt.axes([0.2, 0.4, 0.7, 0.5])  
    ax1log = noise.twinx()  
    ax1log.set_ylabel('RMS Voltage (uV)')  
    ax1log.set_xlabel('startfreq_noise, stopfreq_noise')  
    ax1log.autoscale()  
    ax1log.autoscale(enable=True, axis='y', tight=True)  
    ax1log.set_xlabel('f1, f2')  
  
    RMSNoise = RMSNoise + 10^6  
  
    ax2 = plt.subplots(1)  
    ax2.set_xlabel('Lower limit frequency (Hz)')  
    ax2.set_ylabel('RMS Voltage (uV)')  
    ax2.set_xlim(startfreq_noise, stopfreq_noise)  
    ax2.y.autoscale()  
    ax2.autoscale(enable=True, axis='y', tight=True)  
    ax2.set_xlabel('f1, f2')  
  
    RMSNoise = RMSNoise + 10^6  
    ax2.semilogx(freq, RMSNoise, color=silentsys_ver, label='RMS Voltage: (R:2f) uV ((1:1f) Hz to (2:1f) MHz)'.format(RMSNoise[0], freq.min(), freq.max()*1e-6))  
  
    ax2.legend(frameon=False)  
    fig1.tight_layout()  
    plt.title('Voltage Noise RMS + str(noiseRMS)')  
    img = plt.imread('images/SILENTSYS_VERT_Pantone_18pc.png')  
    ax1log = plt.axes([0.2, 0.3, 0.7, 0.5])  
    ax1log.set_ylabel('RMS Voltage (uV)')  
    ax1log.set_xlabel('startfreq_noise, stopfreq_noise')  
    ax1log.autoscale()  
    plt.show()
```

```
1 #----- RMS -----  
2  
3 startfreq_noise = 0.9 #Hz  
4 stopfreq_noise = 100 #Hz  
5  
6 #Integrated noise  
7 freqnoise = PSD_SSB_plot1_noise  
8 range2 = np.where(f_plot1_noise >= startfreq_noise)(f_plot1_noise <= stopfreq_noise)  
9 freq, freqnoise = f_plot1_noise[range2], freqnoise[range2]  
10 RMSNoise = np.sqrt(np.cumtrapz(freqnoise**2, freq[1:-1], initial=0))[:-1] #the minus is because of the backward integration  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100
```





# Vision globale du code : les fonctions (6)

Regardons le code des fonctions

```
1 #-----RMS-----
2
3
4 #Limits of integration
5 startfreq = 0.9 #Hz
6 stopfreq = 1e6 #Hz
7
8 #Integrated noise
9 freqNoise = PSD_SSB_plot1
10 rangel = sp.where((f_plot1 >= startfreq)*(f_plot1 <= stopfreq))
11 freq, freqNoise = f_plot1[rangel], freqNoise[rangel]
12 RMSVNoise = sp.sqrt(-spi.cumtrapz(2*freqNoise[::-1],freq[::-1],initial=0))[:-1]#the minus is because of the backward integration
13
14
15
16
17 fig2, ax2 = plt.subplots()
18 ax2.set_xlabel('Lower limit frequency (Hz)')
19 ax2.set_ylabel('RMS Voltage (uV)')
20
21 ax2.set_xlim(startfreq, stopfreq)
22 ax2.autoscale(enable=True, axis='y', tight=True)
23
24 RMSVNoise = RMSVNoise*1e6
25 ax2.semilogx(freq, RMSVNoise, label='RMS Frequency: {0:.2f} µHz to {1:.1f} Hz to {2:.1f} MHz'.format(RMSVNoise[0],freq.min(),freq.max())*1e-6)
26
27
28 ax2.legend(frameon=False)
29 fig2.tight_layout()
30 img = plt.imread('images/SILENTSYS_VERT_Pantone_10pc.png')
31 axlogo = plt.axes([0.2, 0.3, 0.7, 0.5])
32 axlogo.set_axis_off()
33 axlogo.imshow(img)
34 plt.show()
35
```

```
1 def saveFreq():
2     global f_plot1, PSD_SSB_plot1, freq_pico
3
4     if status_pico == False:
5         return showinfo ('Warning', 'Picoscope is not open, please open it')
6     #si le picoscope n'est pas initialise
7     if i_pico == False:
8         return showinfo ('Warning', 'Picoscope is not initialized, please initialize the picoscope')
9     #si le frequency noise n'est pas en cours d'acquisition
10    if freq_pico == False:
11        return showinfo ('Warning', 'Start the Frequency Noise acquisition before saving')
12
13
14    path = filedialog.askdirectory (initialdir = "/",title = "Select directory")
15    #choose the name of the txt when we are in the folder
16    name = askstring ("Name of the txt", "Name of the txt")
17    if name == None:
18        return showinfo ("Warning", "You did not enter a name")
19    if path != None:
20        data1 = sp.transpose([f_plot1, PSD_SSB_plot1])
21        np.savetxt(path + '/' + name + '.txt', data1, delimiter='\t', fmt='%0.6e')
22        showinfo ('Info', 'Txt saved')
23
```

# LES FONCTIONS (1)

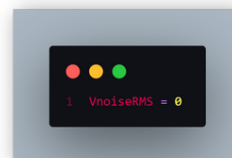
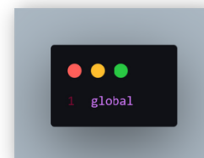
Les fonctions sont dans un fichier à part et sont reliées au fichier de l'interface. C'est dans ce fichier que se passent toutes les actions déclenchées par les boutons.

```
1 #---- IMPORTATION (1) ----#
2 from tkinter import *
3 from tkinter.ttk import *
4 import time
5 from tkinter import N, filedialog, END, Toplevel
6 from tkinter.messagebox import showinfo
7 from tkinter.simpledialog import askstring
8 import importlib
9
10
11 P = importlib.import_module('PicoScope 5000 Series Data Acquisition Software')
12
13 #----GLOBAL VARIABLES----#
14 global led, reso, range, sample, filter, noise, status_pico, U_meas, status_init_pico, f_plot1_vnoise, PSD_S58_plot1_vnoise, U_meas0, v_pico, i_pico, s_pico, vnoise_open, acqui_pico, PSD_S58_plot1, f_plot1, freq_pico
15
16
17 #----INITIALIZATION----#
18 VnoiseRMS = 0
19 nbnoise = 0
20 nbacqui = 0
21 nbfreq = 0
22 nbRMS = 0
23 noise = None
24 status_pico = False
25 freq_pico = False
26 l_pico = False
27 s_pico = False
28 save_pico = False
29 v_pico = False
30 vnoise_open = False
31 acqui_pico = False
32
33
34 #----IMPORTATION (2)----#
35 import matplotlib.pyplot as plt
36 import numpy as np
37 import scipy as sp
38 import scipy.integrate as spi
39 import time
40 import warnings
41 warnings.filterwarnings('ignore') # just use this if warnings are annoying
42
43 import ctypes
44 from picosdk.ps5000a import ps5000a as ps
45 from picosdk.functions import adc2mV, assert_pico_ok, mv2adc
```

Comme sur le fichier de l'interface graphique, **les importations sont présentes, mais elles sont aussi nombreuses**. On importe dans ce fichier des librairies en plus comme **Matplotlib (librairie de graphiques)** ou encore **picosdk (module nécessaire pour commuier avec le PicoScope)**. La librairie **importlib** est elle aussi présente.

Il y a aussi la partie "global" : global est une instruction qui l'informe que la **variable qui est utilisée à l'intérieur de la fonction est la même que celle qui est définie à l'extérieur de la fonction (dans l'environnement global)**.

Enfin, dans cet extrait, **des variables sont initialisées**. Ces dernières sont importantes pour la suite du code.



# LES FONCTIONS (2)

```
1 class Picoscope_channelA:
2     def __init__(self, Samples_per_channel:int (104), triggerlevel_mv:0):
3
4         Samples_per_channel = int (Samples_per_channel)
5         self.triggerlevel_mv = triggerlevel_mv
6         self.channel = ctypes.c_int16 (1)
7         self.status = 1
8         self.resolution = ps.PS5000A_DEVICE_RESOLUTION["PS5000A_DR_16BIT"]
9         self.status["openunit"] = ps.ps5000aOpenUnit (ctypes.byref (self.channel), None, self.resolution)
10        # PICO_STATUS ps5000aOpenUnit(int16_t * handle,
11        #     int16_t * serial, None means it just takes the first scope found
12        #     PS5000A_DEVICE_RESOLUTION resolution)
13        assert_pico_ok (self.status["openunit"])
14
15        # print('picoscope is open now')
16
17        # Disable other channels
18        # Set up channel B
19        # handle = channel
20        self.channelB = ps.PS5000A_CHANNEL["PS5000A_CHANNEL_B"]
21        # enabled = 1
22        self.coupling_type = ps.PS5000A_COUPLING["PS5000A_DC"]
23        # self.coupling_type = ps.PS5000A_COUPLING["PS5000A_AC"]
24        # typedef enum enPS5000ACoupling(PS5000A_AC, PS5000A_DC) PS5000A_COUPLING
25        self.chRange = ps.PS5000A_RANGE["PS5000A_1V"] # it was 20V before.
26        ## self.chRange = ps.PS5000A_RANGE["PS5000A_200mV"] # it was 20V before.
27        # typedef enum enPS5000ARange(
28        #     PS5000A_10mV,
29        #     PS5000A_20mV,
30        #     PS5000A_50mV,
31        #     PS5000A_100mV,
32        #     PS5000A_200mV,
33        #     PS5000A_500mV,
34        #     PS5000A_1V,
35        #     PS5000A_2V,
36        #     PS5000A_5V,
37        #     PS5000A_10V,
38        #     PS5000A_20V,
39        #     PS5000A_50V,
40        #     PS5000A_MAX_RANGES
41        # ) PS5000A_RANGE;
42        # Last entry is analogue offset = 0 V
43        self.status["setChB"] = ps.ps5000aSetChannel (self.channel, self.channelB, 0, self.coupling_type, self.chRange,
44        #     0)
45        # PICO_STATUS ps5000aSetChannel(int16_t handle, PS5000A_CHANNEL channel, int16_t enabled, PS5000A_COUPLING type,
46        #     PS5000A_RANGE range, float analogueOffset)
47        assert_pico_ok (self.status["setChB"])
48
49        # Set up channel C
50        # handle = channel
51        self.channelC = ps.PS5000A_CHANNEL["PS5000A_CHANNEL_C"]
52        # enabled = 1
53        self.coupling_type = ps.PS5000A_COUPLING["PS5000A_DC"]
54        # self.coupling_type = ps.PS5000A_COUPLING["PS5000A_AC"]
55        # typedef enum enPS5000ACoupling(PS5000A_AC, PS5000A_DC) PS5000A_COUPLING
56        self.chRange = ps.PS5000A_RANGE["PS5000A_1V"] # it was 20V before.
57        ## self.chRange = ps.PS5000A_RANGE["PS5000A_200mV"] # it was 20V before.
58        # typedef enum enPS5000ARange(
59        #     PS5000A_10mV,
60        #     PS5000A_20mV,
61        #     PS5000A_50mV,
62        #     PS5000A_100mV,
63        #     PS5000A_200mV,
64        #     PS5000A_500mV,
65        #     PS5000A_1V,
66        #     PS5000A_2V,
67        #     PS5000A_5V,
68        #     PS5000A_10V,
69        #     PS5000A_20V,
70        #     PS5000A_50V,
71        #     PS5000A_MAX_RANGES
72        # ) PS5000A_RANGE;
73        # Last entry is analogue offset = 0 V
74        self.status["setChC"] = ps.ps5000aSetChannel (self.channel, self.channelC, 0, self.coupling_type, self.chRange,
75        #     0)
```

Voici l'une des classes importantes pour faire fonctionner tout ce code :  
"Picoscope\_channelA".

Elaborée sur Jupyter dans un premier temps, nous l'avons repris ici. Cette classe contient toutes les fonctions permettant d'acquérir les datas, de mettre au point les différents channels de l'appareil, la résolution, la range...

De par sa longueur et du fait que rien n'a été changé, nous ne montrerons que l'un des extraits de cette partie.

# LES FONCTIONS (3)

```
1 class ProcessNoise (object): # TODO Live class
2 ...
3 Class used for a adding live update to a figure, Run "Matplotlib notebook" in jupyter notebook
4 for optimal operation of this class. This adds
5
6 Attributes
7 -----
8 needs explanation""
9
10 def __init__(self, init_ps=True, Nsamples=int(2 ** 25)):
11
12     #####
13     #Initialize Picoscope
14     #####
15     self.Nsamples = Nsamples
16     if init_ps:
17         try:
18             self.picoscope = Picoscope_channelA (Samples_per_channel=self.Nsamples)
19
20         except:
21             Picoscope_channelA.close () # check if self needed here
22             self.picoscope = Picoscope_4channels (Samples_per_channel=self.Nsamples)
23             print ("Picoscope was already open and was now reinitialized")
24
25     self.picoscope.N = self.Nsamples
26     self.picoscope.set_timebase (5) # for 12BIT 2 is 4ns Sampling Interval => 250MS/s; 3 is 8ns => 125MS/s
27     # check on page 22 in programmers-guide 3width 15 bit 3 is 125MS/s, 4 is 64MS/s, 5 is 42 MS/s, 6 is 32 MS/s, 7 is 25 MS/s,
28     # 8 is 20 MS/s, 9 is 18MS/s,
29     # 10 is 16 MS/s, 11 is 14 MS/s, 12 is 12.5 MS/s, 13 is 11 MS/s, 14 is 10 MS/s, 127 is 1 MS/s, 52 is 2.5 MS/s
30     self.picoscope.set_bandwidthfilter (1) # this sets the 200MHz lowpassfilter for all channels
31     self.picoscope.R = 1 / (self.picoscope.timeintervals.value * 1e-9) # sampling rate
32     self.times = sp.linspace (0, (self.picoscope.N - 1) / self.picoscope.R,
33                               self.picoscope.N) # check if this will be needed for the future
34     self.Noise_start_acquisition ()
35     self.picoscope.check_ready () # this is needed to write the data to the buffer for the readout.
36     # here we could save time to evaluate while writing into the buffer at the same time
37     time.sleep (1) # this is to make sure that the AC coupling has time to stabilize
38     self.Noise_start_acquisition ()
39     self.picoscope.check_ready () # this is needed to write the data to the buffer for the readout.
40     self.picoscope.readout_buffer ()
41
42     self.time = sp.linspace (0, (self.picoscope.N - 1) / self.picoscope.R, self.picoscope.N)
43     self.signal_A = sp.array (self.picoscope.bufferA)
44
45     # self.N_FFT = Nsamples
46     # self.dt = 1/self.picoscope.R
47     # self.freq = self.fftfreq(self.N_FFT, self.dt)
48     # self.signal_FFT1 = sp.zeros(self.N_FFT)
49     # self.signal_FFT1_phase = sp.zeros(self.N_FFT)
50     # self.signal_FFT1_amplitude = sp.ones(self.N_FFT)
51
52 def Noise_start_acquisition(self):
53     self.picoscope.start_acqldata_triggered () # this writes the data into the buffer
54
55 def update_data(self):
56     self.Noise_start_acquisition () # this starts the acquisition
57     self.picoscope.check_ready () # this is needed to write the data to the buffer for the readout.
58     # here we could save time to evaluate while writing into the buffer at the same time
59     self.picoscope.readout_buffer ()
60     self.signal_A = sp.array (
61         adc2mV (self.picoscope.bufferA, self.picoscope.chARange, self.picoscope.maxADC)) * 1e-3
62     self.signal_Noise1 = self.signal_A
63
64 def save_data(self, fn):
65     data = sp.transpose ([self.time, self.signal_Noise1])
66     sp.savetxt (fn, data, fmt='%0.6e', headers='time (s), signal A (V)', delimiter=',', newline='\n')
67
68 def save_dataraw(self, fn):
69     data = sp.transpose ([self.time, self.signal_Noise1])
70     sp.savetxt (fn, data, fmt='%0.6e', headers='time (s), signal A (V)', delimiter=',', newline='\n')
71
72 def load_rawdata(self, fn): # speed up this function todo
73     self.time, self.signal_A = sp.loadtxt (fn, unpack=True, delimiter=',', skiprows=1)
```

Voici la deuxième classe importante pour faire fonctionner tout ce code :  
"ProcessNoise".

Cette classe permet de faire une **mise à jour en temps réel des graphiques**.

On peut y trouver des fonctions permettant de commencer l'acquisition, de mettre à jour,  
de sauvegarder..

# LES FONCTIONS (4)

```
1 def open_pico(): # work
2     global status_pico, noise
3     noise = ProcessNoise (init_ps=True, Nsamples=int (2 ** 25)) # up to 2^25 should be ok for the PS5444D;
4     status_pico = True
5     if status_pico == True:
6         led.config (bg='green')
7         return showinfo ('Great','Picoscope is open')
8
9
10 # 2^24 are too many samples for buffering for the PS5442D used for testing.
11 # execute cell with noise.picoscope.close() below if an error appears
12
13 def close_pico(): # work
14     global status_pico, s_pico, i_pico, v_pico, vnoise_open, acqui_pico, freq_pico
15     noise.picoscope.close ()
16     status_pico = False
17     if status_pico == False:
18         led.config (bg='red')
19         return showinfo ('Warning','Picoscope is closed')
20 #quand on ferme le picoscope, on réinitialise Les popups
21     s_pico = False
22     i_pico = False
23     v_pico = False
24     vnoise_open = False
25     acqui_pico = False
26     freq_pico = False
27
```

Voici les deux fonctions "open\_pico" et "close\_pico".

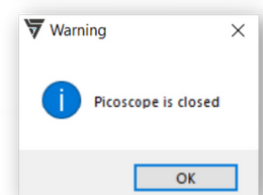
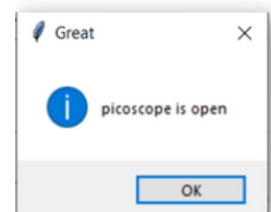
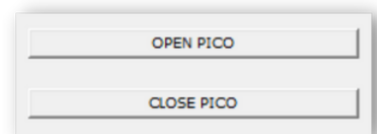
"open\_pico" contient le **global** permettant de bien faire marcher les variables, on retrouve aussi la **variable noise** qui permet de **démarrer le PicoScope avec un sample définit**.

Une **mise à jour de la variable status\_pico** est aussi là, elle passe de False à True.

Enfin, on retrouve une **condition "if"** qui permet de changer la couleur du carré en vert et d'afficher un pop-up notifiant l'utilisateur de l'ouverture.

"close\_pico" est similaire, elle contient un **global**, une **ligne permettant de faire fermer le PicoScope**, une **mise à jour des variables** plus grande, car elle remet à zéro tous les boutons puis enfin une **condition** accompagnée d'un **changement de statut** et un **pop-up notifiant la fermeture**.

Ces deux fonctions sont reliés aux boutons "OPEN PICO" et "CLOSE PICO" grâce à command: D."*nom de la def*" présent dans le fichier de l'interface

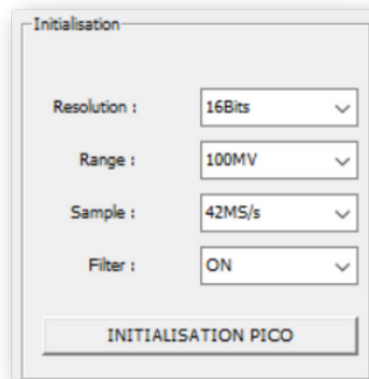


# LES FONCTIONS (5)

Voici la fonction "init\_pico" (reliée au bouton INITIALISATION PICO)

Cette fonction contient elle aussi un global et une condition en cas d'oubli de l'ouverture du PicoScope.

Mais elle contient aussi **4 tableaux associatifs** permettant d'établir une liste des valeurs possibles dans les listes déroulantes. Il y a le tableau pour la range, la timebase, le filtre et la résolution.



```
1 def init_pico():
2     global i_pico
3
4     if status_pico == False:
5         return showinfo ('Error', 'Picoscope is not open')
6
7     range_tab = [
8         # tableau associatif pour les valeurs de range
9         {'name': '10MV', 'value': 0},
10        {'name': '20MV', 'value': 1},
11        {'name': '50MV', 'value': 2},
12        {'name': '100MV', 'value': 3},
13        {'name': '200MV', 'value': 4},
14        {'name': '500MV', 'value': 5},
15        {'name': '1V', 'value': 6},
16        {'name': '2V', 'value': 7},
17        {'name': '5V', 'value': 8},
18        {'name': '10V', 'value': 10},
19        {'name': '20V', 'value': 11},
20        {'name': '50V', 'value': 12},
21    ]
22    timebase_tab = [
23        # tableau associatif pour les valeurs de timebase
24
25        {'name': '64MS/s', 'value': 4},
26        {'name': '42MS/s', 'value': 5},
27        {'name': '32MS/s', 'value': 6},
28        {'name': '25MS/s', 'value': 7},
29        {'name': '20MS/s', 'value': 8},
30        {'name': '18MS/s', 'value': 9},
31        {'name': '16MS/s', 'value': 10},
32        {'name': '14MS/s', 'value': 11},
33        {'name': '12.5MS/s', 'value': 12},
34        {'name': '11MS/s', 'value': 13},
35        {'name': '10MS/s', 'value': 14},
36    ]
37
38    filter_tab = [
39        # tableau associatif pour les valeurs de filter
40        {'name': 'OFF', 'value': 1},
41        {'name': 'ON', 'value': 0},
42    ]
43
44    resolution_tab = [
45        # tableau associatif pour les valeurs de resolution
46        {'name': '16Bits', 'value': 16}
47    ]
48
49    for r in resolution_tab:
50        if r['name'] == reso.get ():
51            getreso = r['value']
52            print ("reso : " + str (getreso))
53
54    for i in range_tab:
55        if i['name'] == range.get ():
56            getrange = i['value']
57            print ("range : " + str (getrange))
58
59    for j in timebase_tab:
60        if j['name'] == sample.get ():
61            gettime = j['value']
62            print ("time : " + str (gettime))
63
64    for x in filter_tab:
65        if x['name'] == filter.get ():
66            getfilter = x['value']
67            print ("filter : " + str (getfilter))
68
69    # save the values in the picoscope
70    noise.picoscope.set_timebase (gettime)
71    noise.picoscope.set_range (getrange)
72    noise.picoscope.set_resolution (getreso)
73    noise.picoscope.set_BandwidthFilter (getfilter)
74    i_pico = True
75    showinfo ('Info', 'Picoscope is initialized')
76
77
```

Ces tableaux sont tous parcourus par une boucle faisant la chose suivante :

si le "name" de l'un des choix est égal au choix sélectionné alors la variable getrange (par exemple) est égale à la valeur en adéquation avec le "name" choisi (exemple : on sélectionne dans la liste de la range "100Mv", c'est donc égal au "name" 100Mv dans le tableau donc getrange prend la valeur 3 car "value" : 3 dans le tableau associatif)

Ces valeurs sont envoyées au PicoScope :

**noise.picoscope.set\_"nom de la valeur" (get"variable correspondante")** permet d'envoyer tous ça au PicoScope et donc de l'initialiser grâce au bouton "INITIALISATION PICO". Une variable est mise à jour et elle vous permettra de lancer la suite de l'application. L'utilisateur sera mis au courant du succès de l'initialisation.

# LES FONCTIONS (6)

```
1 def startacqui():
2     global U_meas, s_pico, U_meas0, v_pico, acqui_pico, nbacqui
3
4     nbacqui=nbacqui+1
5
6     if status_pico == False:
7         return showinfo ('Warning', 'Picoscope is not open, please open it')
8     if i_pico == False:
9         return showinfo ('Warning', 'Picoscope is not initialized, please initialize the picoscope')
10    if i_pico == True:
11        showinfo ('Info', 'Acquisition is started. This can take a while. After that you can save the data and launch the Voltage Noise Acquisition. ')
12
13        v_pico = True
14
15        noise.update_data ()
16
17        U_meas0 = noise.signal_Noise1
18        U_meas0.max ()
19
20        U_meas0.min ()
21
22        U_meas = (U_meas0 - U_meas0.mean ())
23
24        fig0 = plt.figure ()
25        plt.plot (U_meas0)
26
27        plt.autoscale (enable=True, axis='y', tight=True)
28        plt.title ('Time Trace ' + str(nbacqui))
29        plt.ylabel ('Tension (V)')
30
31        plt.xlabel ('Npts (s)')
32
33        acqui_pico = True
34        plt.show ()
```

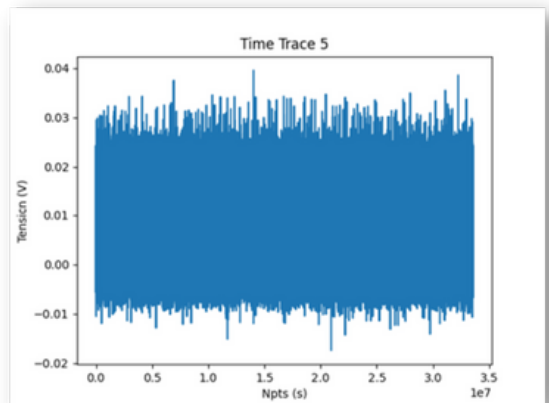
Voici la fonction **startacqui** (relié au bouton start acquisition) :

Une **variable nbacqui** est présente, elle a été initialisée à 0 au début du code. Cette dernière à chaque acquisition se verra incrémenté de 1.

Après les conditions, on peut apercevoir le **noise.update\_data()** permettant de mettre à jour le noise.

La **variable U\_meas0** est égal au signal émit, on récupère le **min()** et le **max()**. Puis on écrit **une variable U\_meas** égal à **U\_meas0** et la **moyenne**.

Puis on plot les données en un graphique (**plt.plot(U\_meas0)**). On met **l'axe y en automatique**, on définit le titre des axes puis on l'affiche (**plt.show**).



START ACQUISITION



# LES FONCTIONS (7)

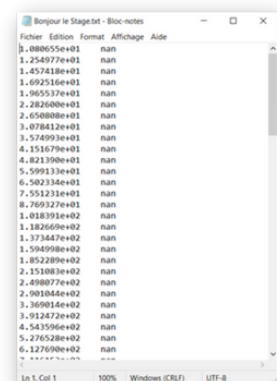
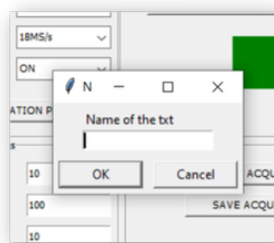
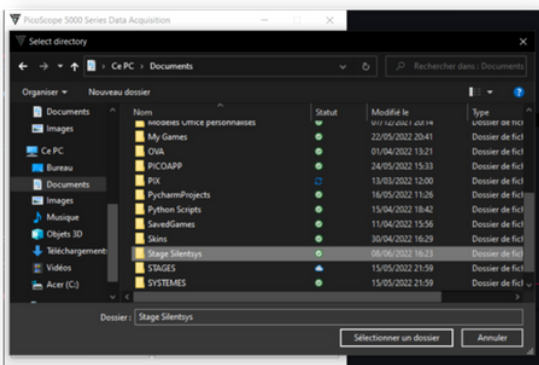
```
1 def saveacqui():
2     global U_meas, s_pico, U_meas0, v_pico, acqui_pico
3
4     #si l'acquisition n'est pas lancée, on ne peut pas sauvegarder
5     if v_pico == False:
6         return showinfo ('Warning', 'Acquisition is not started, please start the acquisition')
7
8     #si on ne lance pas l'acquisition, on ne peut pas sauvegarder
9     if acqui_pico == False:
10        return showinfo ('Warning', 'Acquisition is not started, please start the acquisition before saving datas')
11
12
13
14     #si le picoscope n'est pas ouvert
15     if status_pico == False:
16         return showinfo ('Warning', 'Picoscope is not open, please open it')
17     #si le picoscope n'est pas initialisé
18     if i_pico == False:
19         return showinfo ('Warning', 'Picoscope is not initialized, please initialize the picoscope')
20
21
22
23     global U_meas0
24     path = filedialog.askdirectory (initialdir = "/",title = "Select directory")
25
26     #choose the name of the txt when we are in the folder
27     name = askstring ("Name of the txt", "Name of the txt")
28
29     if name == None:
30         return showinfo ("Warning", "You did not enter a name")
31     if path != None:
32         data2 = sp.transpose([U_meas0])
33         np.savetxt(path + '/' + name + '.txt', data2, delimiter='\t', fmt='%0.6e')
34         showinfo ('Info', 'Txt saved')
```

**Voici la fonction saveacqui (relié au bouton save acquisition) :**

Cette fonction permet de sauvegarder en .txt les données brutes acquises lors de l'acquisition.

La variable path permet d'ouvrir une fenetre de choix pour le dossier. Une fois que c'est fait, on utilise **askstring de Tkinter** afin de donner un nom au fichier.

Si le chemin et le nom sont rentrés, on sauvegarde tout cela (**np.savetxt**) et un pop-up apparait notifiant l'utilisateur que cela est bien enregistré.



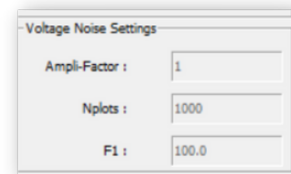
# LES FONCTIONS (8)

Voici la fonction `vnoise` : Elle permet de faire l'acquisition du Vnoise (relié au bouton (Voltage Noise Analysis))

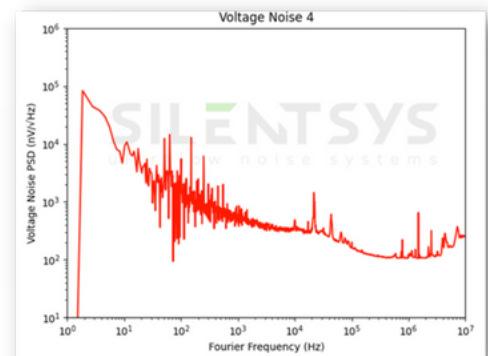
Voici les conditions permettant de guider l'utilisateur.

Et voici les **début des calculs** correspondant aux différentes valeurs que l'on va rentrer. (Nplot, F1, Ampli-Factor)

On crée des variables permettant de récupérer les différentes données inscrites dans les zones de texte.



Puis on affiche tout cela dans un graphique prenant en valeurs le `fplot_vnoise` et le `PSD_SSB_plot1_vnoise`



```
def vnoise(amplfact, Nplot, F1):
    # si le picoscope n'est pas ouvert
    if status_pico == False:
        return showinfo('Warning', 'Picoscope is not open, please open it')
    # si le picoscope n'est pas initialise
    if i_pico == False:
        return showinfo('Warning', 'Picoscope is not initialized, please initialize the picoscope')
    # si le picoscope n'est pas en cours d'acquisition
    if v_pico == False:
        return showinfo('Warning', 'Picoscope is not started, please start the acquisition')

    if amplfact.get() == '' and Nplot.get() == '' and F1.get() == '':
        return showinfo('Warning', 'You did not enter any values')
    if amplfact.get() == '':
        return showinfo('Warning', 'You did not enter a value for the amplification factor')
    # si le champ Nplot est vide
    if Nplot.get() == '':
        return showinfo('Warning', 'You did not enter a value for the number of points')
    # si le champ F1 est vide
    if F1.get() == '':
        return showinfo('Warning', 'You did not enter a value for the frequency')
    # si tous les champs sont vides
    # si le voltage noise est lance
    if v_pico == True:
        showinfo('Info', 'Voltage Noise Analysis and RMS is started, this can take a while. After that you can save the data.')

    global f_plot1_vnoise, PSD_SSB_plot1_vnoise, vnoise_open, nnoise, VnoiseRMS

    nnoise = nnoise + 1
    VnoiseRMS = VnoiseRMS + 1
    R = noise.picoscope.R
    N = noise.picoscope.N
    dt = 1 / R # time between 2 samples [s]
    t = dt * np.arange(N) # t axis np.arange(N)=[0,1,...,N-2,N-1]
    T = N / R # total measurement time [s]
    df = 1 / T # frequency spacing [Hz]
    f = df * np.arange(1, N + 1) # f axis
    T

    amplfact = int(amplfact.get())
    Nplot = int(Nplot.get())
    F1 = float(F1.get())

    # amplificationfactor = tension * 2 * 3.14 / (2.04 * 1e6) # for the conversion from V to Hz - by adjusting optical power to reach +/- "tension" signal
    amplificationfactor = amplfact # for the voltage amplification
    R0 = 1 / amplificationfactor # 0.2meas
    df0 = np.absolute(coeff.fet(R0))
    PSD_SSB = df0[ int(N/2) ] ** 2 / (
        N * R) # Power Spectral Density v^2/Hz This is the SSB (single sideband, one-sided only positive frequencies) as displayed and saved in the RMS FSNP
    f = f[ int(N/2) ]

    tic = time.time()
    # N_plot = 1000 # for points above f1
    # f2 = 2 * 1e2
    N_plot = Nplot # for points below f1
    f1 = F1
    f_temp = np.logspace(np.log10(f1), np.log10(f.max()), N_plot + 1)
    f_plot = (f_temp[-1] + f_temp[1]) / 2
    PSD_SSB_plot = np.zeros_like(f_plot)
    for ii, f_plot_i in enumerate(f_plot):
        PSD_SSB_plot[ii] = PSD_SSB[(f > f_temp[ii]) * (f < f_temp[ii + 1])].mean()
    toc = time.time() - tic
    print(toc)

    f_plot1_vnoise = np.append([f < f1], f_plot)
    PSD_SSB_plot1_vnoise = np.append(PSD_SSB[f < f1], PSD_SSB_plot)

    fig1, ax1 = plt.subplots()
    silentsys_vert = (22 / 255, 186 / 255, 76 / 255, 1)
    silentsys_rouge = (254 / 255, 22 / 255, 0 / 255, 1)
    ax1.loglog(f_plot1_vnoise, np.sqrt(PSD_SSB_plot1_vnoise)*1e9, color=silentsys_rouge)
    #ax1.loglog(f_plot1_vnoise, np.sqrt(PSD_SSB_plot1_vnoise)*1e9, color=silentsys_vert)
    ax1.set_ylim(1e0, 1e7)
    ax1.set_xlim(1e1, 1e7)
    ax1.set_ylabel('Voltage Noise PSD (nV/√Hz)')
    ax1.set_xlabel('Fourier Frequency (Hz)')
    img2 = plt.imread('images/SILENTSYS_VERT_Pantone_18pc.png')
    fig1.tight_layout()

    plt.title('Voltage Noise '+ str(nnoise))
    axlogo = plt.axes([0.2, 0.4, 0.7, 0.5])
    axlogo.set_axis_off()
    axlogo.imshow(img2)

    vnoise_open = True
    plt.show()
```

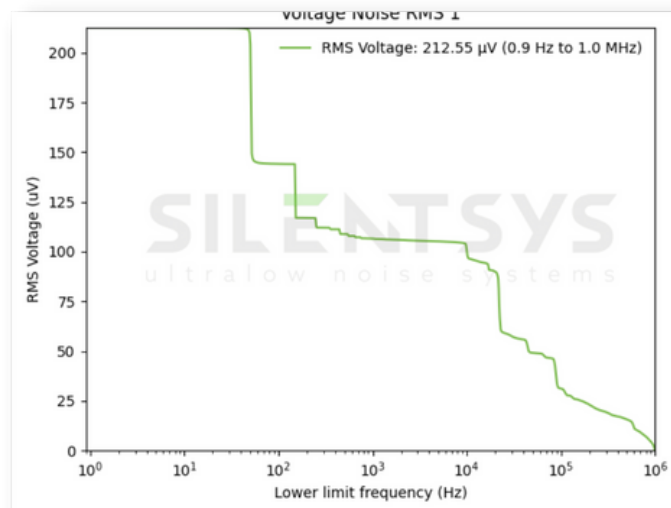
# LES FONCTIONS (9)

```
1 #-----RMS-----
2
3
4 startfreq_vnoise = 0.9 #Hz
5 stopfreq_vnoise = 1e6 #Hz
6
7 #Integrated noise
8 freqNoise = PSD_SSB_plot1_vnoise
9 range2 = sp.where((f_plot1_vnoise >= startfreq_vnoise)*(f_plot1_vnoise <= stopfreq_vnoise))
10 freq, freqNoise = f_plot1_vnoise[range2], freqNoise[range2]
11 RMSVnoise = sp.sqrt(-spi.cumtrapz(2*freqNoise[:-1],freq[:-1],initial=0))[:-1]#the minus is because of the backward integration
12
13
14
15
16
17
18 fig2, ax2 = plt.subplots()
19 ax2.set_xlabel('Lower limit frequency (Hz)')
20 ax2.set_ylabel('RMS Voltage (uV)')
21
22 ax2.set_xlim(startfreq_vnoise, stopfreq_vnoise)
23 #axe y autoscale
24 ax2.autoscale(enable=True, axis='y', tight=True)
25 #ax2.set_ylim(0, plt.autoscale())
26
27 RMSVnoise = RMSVnoise*1e6
28
29 ax2.semilogx(freq, RMSVnoise,color=silentsys_vert, label='RMS Voltage: {0:.2f} uV ({1:.1f} Hz to {2:.1f} MHz)'.format(RMSVnoise[0],freq.min(),freq.max()*1e-6))
30
31
32 ax2.legend(frameon=False)
33 fig2.tight_layout()
34 plt.title('Voltage Noise RMS ' + str(VnoiseRMS))
35 img = plt.imread('images/SILENTSYS_VERT_Pantone_10pc.png')
36 axlogo = plt.axes([0.2, 0.3, 0.7, 0.5])
37 axlogo.set_axis_off()
38 axlogo.imshow(img)
39 plt.show()
40
```

**Voici la partie de la précédente fonction permettant de faire le RMS (elle aussi reliée au bouton Voltage Noise Analysis):**

On définit plusieurs variables qui sont égales à différents calculs.

On met en **place la création du graphique qui affichera le RMS.**



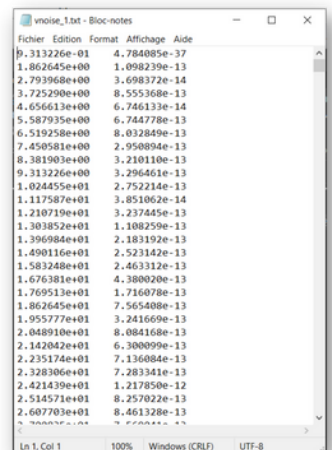
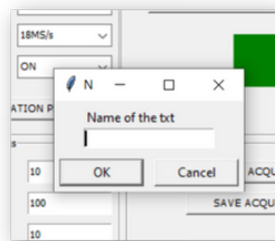
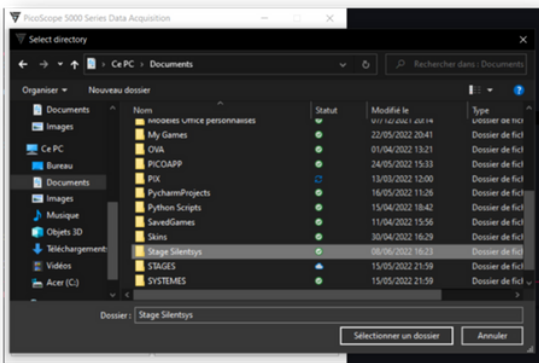
# LES FONCTIONS (10)

```
1 def sauvegardevnoise():
2
3
4     #si le picoscope n'est pas ouvert
5     if status_pico == False:
6         return showinfo ('Warning', 'Picoscope is not open, please open it')
7     #si le picoscope n'est pas initialisé
8     if i_pico == False:
9         return showinfo ('Warning', 'Picoscope is not initialized, please initialize the picoscope')
10    #si le picoscope n'est pas en cours d'acquisition
11    if v_pico == False:
12        return showinfo ('Warning', 'Start the Voltage Noise acquisition')
13    #si le voltage noise n'est pas ouvert
14    if vnoise_open == False:
15        return showinfo ('Warning', 'Voltage Noise is not open, please open it')
16
17
18
19    global f_plot1_vnoise, PSD_SSB_plot1_vnoise
20    path = filedialog.askdirectory (initialdir = "/",title = "Select directory")
21    #choose the name of the txt when we are in the folder
22    name = askstring ("Name of the txt", "Name of the txt")
23    if name == None:
24        return showinfo ("Warning", "You did not enter a name")
25    if path != None:
26        data1 = sp.transpose([f_plot1_vnoise, PSD_SSB_plot1_vnoise])
27        np.savetxt(path + '/' + name + '.txt', data1, delimiter='\t', fmt='%0.6e')
28        showinfo ('Info', 'Txt saved')
```

**Voici la fonction sauvegardevnoise (relié à son bouton save analysis):**

Cette fonction permet de sauvegarder en .txt les données brutes acquises lors de l'acquisition comme pour la fonction save acquisition.

Les seules choses qui changent sont les valeurs enregistrées (f\_plot1\_vnoise et PSD\_SSB\_plot1\_vnoise)



# LES FONCTIONS (11)

```

1 def frequency(V, FSR, Nplot, F1):
2     #if status_pico == False:
3         return showinfo('Warning', 'Acquisition is not started, please start the acquisition')
4     global nbFreq, nbRMS, f_plot1, PSD_SSB_plot1, freq_pico
5     if V.get() == '' and FSR.get() == '' and Nplot.get() == '' and F1.get() == '':
6         return showinfo('Warning', 'Please enter the voltage, the FSR, the number of points and the frequency')
7     if V.get() == '':
8         return showinfo('Warning', 'Please enter the voltage')
9     if FSR.get() == '':
10        return showinfo('Warning', 'Please enter the FSR')
11    if Nplot.get() == '':
12        return showinfo('Warning', 'Please enter the Nplot')
13    if F1.get() == '':
14        return showinfo('Warning', 'Please enter F1')
15
16
17
18
19 if status_pico == False:
20     return showinfo('Warning', 'Picoscope is not open, please open it')
21
22 if f_pico == False:
23     return showinfo('Warning', 'Picoscope is not initialized, please initialize the picoscope')
24
25 if v_pico == False:
26     return showinfo('Warning', 'Start the Acquisition')
27
28 if v_pico == True:
29     freq_pico = True
30     showinfo('Info', 'The frequency and the RMS are currently being acquired. This may take some time, please wait...')
31
32 nbFreq = nbFreq + 1
33 nbRMS = nbRMS + 1
34 R = noise.picoscope.R
35 N = noise.picoscope.N
36 dt = 1 / R # time between 2 samples [s]
37 t = dt * np.arange(N) # t axis: arange(N)= [0,1,...,N-2,N-1]
38 T = N / R # total measurement time [s]
39 df = 1 / T # frequency spacing [Hz]
40 f = df * np.arange(1, N + 1) # f axis
41
42
43
44
45 tension = int(V.get())
46 FSR = float(FSR.get())
47 Nplot = int(Nplot.get())
48 F1 = float(F1.get())
49
50 # amplificationFactor = tension * 2 * 3.14 / (2.04 * 1e6) # for the conversion from V to Hz - by adjusting optical power to reach +/- "tension" signal
51 amplificationFactor = tension * 2 * 3.14 / (
52     FSR) # for the conversion from V to Hz - by adjusting optical power to reach +/- "tension" signal
53 dB = 1 / amplificationFactor * 0.0001
54 dB = np.absolute(np.fft.fft(f))
55 PSD_SSB = dB[(int(N / 2)) ** 2 : ]
56     N * R) # Power Spectral Density V^2/Hz This is the SSB (single sideband, one-sided only positive frequencies) as displayed and saved in the RMS FSAP
57 f = [int(N / 2)]
58
59 tic = time.time()
60 # N_plot = 1000 # for points above f1
61 # f1 = 2 * 1e2
62 N_plot = Nplot # for points below f1
63 f1 = F1
64 f_temp = np.logspace(np.log10(f1), np.log10(f.max()), N_plot + 1)
65 f_plot = (f_temp[-1] + f_temp[1]) / 2
66 PSD_SSB_plot = np.zeros_like(f_plot)
67 for ii, f_plot_1 in enumerate(f_plot):
68     PSD_SSB_plot[ii] = PSD_SSB[(f >= f_temp[ii]) * (f < f_temp[ii + 1])].mean()
69
70 tic = time.time() - tic
71 print(tic)
72
73 f_plot1 = np.append([f < f1], f_plot)
74 PSD_SSB_plot1 = np.append(PSD_SSB[f < f1], PSD_SSB_plot)
75
76 fig1, ax1 = plt.subplots()
77 silentsys_vert = (122 / 255, 186 / 255, 76 / 255, 1)
78 ax1.imshow(f_plot1, PSD_SSB_plot1, color=silentsys_vert)
79 ax1.loglog(f_plot1, PSD_SSB_plot1, color=silentsys_vert)
80 ax1.set_xlim(1e0, 1e7)
81 ax1.set_ylim(1e0, 1e7)
82 #ax1.yaxis.set_label('Frequency Noise PSD (Hz^2/Hz)')
83 ax1.set_xlabel('Fourier Frequency (Hz)')
84 fig1.tight_layout()
85
86 plt.title('Frequency Noise ' + str(nbFreq))
87 img = plt.imread('images/SILENTSYS_VERT_Pantone_18pc.png')
88 axlogo = plt.axes([0.2, 0.4, 0.7, 0.5])
89
90 axlogo.set_axis_off()
91 axlogo.imshow(img, zorder=0)
92 plt.show()

```

Frequency Noise Settings

Voltage :

FSR :

Nplot :

F1 :

Voici la fonction frequency : Elle permet de faire l'acquisition du Frequency Noise (reliée au bouton (Voltage Noise Analysis) cette fonction prend en arguments la tension (V), le FSR, le Nplot et le f1.

Voici les conditions permettant de guider l'utilisateur.

Et voici les début des calculs correspondant aux différentes valeurs que l'on va rentrer. (Nplot, F1, Voltage, FSR)

On créer des variables permettant de récupérer les différentes données inscrites dans les zones de texte.

Puis on affiche tout cela dans un graphique prenant en valeurs le f\_plot1 et PSD\_SSB\_plot1



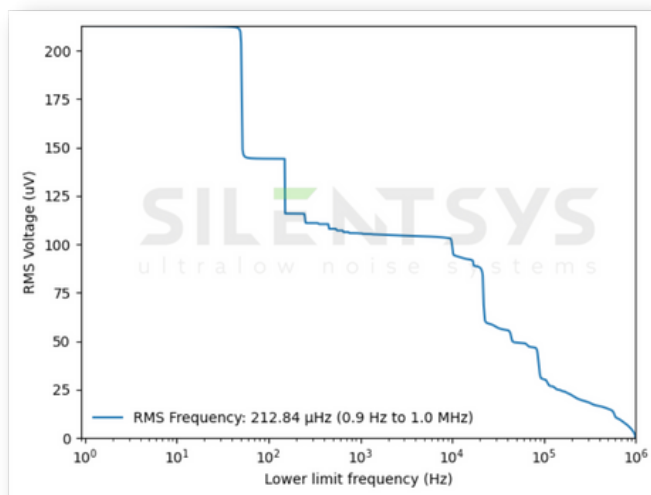
# LES FONCTIONS (12)

```
1 #-----RMS-----
2
3
4 #Limits of intergration
5 startfreq = 0.9 #Hz
6 stopfreq = 1e6 #Hz
7
8 #Integrated noise
9 freqNoise = PSD_SSB_plot1
10 range1 = sp.where((f_plot1 >= startfreq)*(f_plot1 <= stopfreq))
11 freq, freqNoise = f_plot1[range1], freqNoise[range1]
12 RMSVNoise = sp.sqrt(-spi.cumtrapz(2*freqNoise[::-1],freq[::-1],initial=0))[:-1])#the minus is because of the backward integration
13
14
15
16
17 fig2, ax2 = plt.subplots()
18 ax2.set_xlabel('Lower limit frequency (Hz)')
19 ax2.set_ylabel('RMS Voltage (uV)')
20
21 ax2.set_xlim(startfreq, stopfreq)
22 ax2.autoscale(enable=True, axis='y', tight=True)
23
24 RMSVNoise = RMSVNoise*1e6
25 ax2.semilogx(freq, RMSVNoise, label='RMS Frequency: {0:.2f} μHz ({1:.1f} Hz to {2:.1f} MHz)'.format(RMSVNoise[0],freq.min(),freq.max()*1e-6))
26
27
28 ax2.legend(frameon=False)
29 fig2.tight_layout()
30 img = plt.imread('images/SILENTSYS_VERT_Pantone_10pc.png')
31 axlogo = plt.axes([0.2, 0.3, 0.7, 0.5])
32 axlogo.set_axis_off()
33 axlogo.imshow(img)
34 plt.show()
35
```

Voici la partie de la précédente fonction permettant de faire le RMS (elle aussi reliée au bouton Frequency Noise Analysis):

On définit plusieurs variables qui sont égales à différents calculs.

On met en place la création du graphique qui affichera le RMS.



# LES FONCTIONS (13)

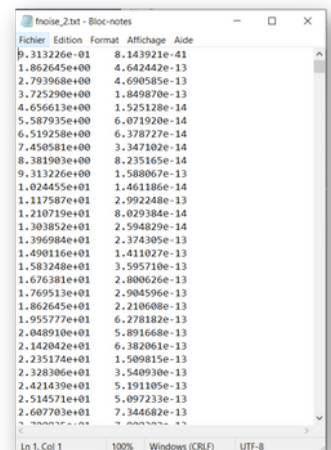
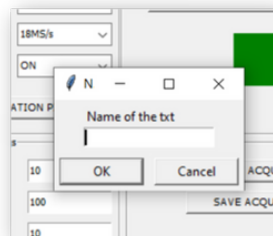
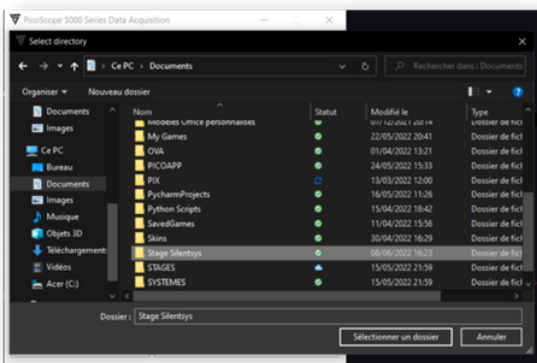
```
1 def saveFreq():
2     global f_plot1, PSD_SSB_plot1, freq_pico
3
4     if status_pico == False:
5         return showinfo ('Warning', 'Picoscope is not open, please open it')
6     #si Le picoscope n'est pas initialise
7     if i_pico == False:
8         return showinfo ('Warning', 'Picoscope is not initialized, please initialize the picoscope')
9     #si Le frequency noise n'est pas en cours d'acquisition
10    if freq_pico == False:
11        return showinfo ('Warning', 'Start the Frequency Noise acquisition before saving')
12
13
14    path = filedialog.askdirectory (initialdir = "/",title = "Select directory")
15    #choose the name of the txt when we are in the folder
16    name = askstring ("Name of the txt", "Name of the txt")
17    if name == None:
18        return showinfo ("Warning", "You did not enter a name")
19    if path != None:
20        data1 = sp.transpose([f_plot1, PSD_SSB_plot1])
21        np.savetxt(path + '/' + name + '.txt', data1, delimiter='\t', fmt='%0.6e')
22        showinfo ('Info', 'Txt saved')
23
```

Voici la fonction saveFreq (reliée au bouton save analysis du Frequency Noise):

Cette fonction permet de sauvegarder en .txt les données brutes acquises lors de l'acquisition.

La variable path permet d'ouvrir une fenetre de choix pour le dossier. Une fois que c'est fait, on utilise **askstring de Tkinter** afin de donner un nom au fichier.

Si le chemin et le nom sont rentrés, on sauvegarde tout cela (**np.savetxt**) et un pop-up apparait notifiant l'utilisateur que cela est bien enregistré.



# NOTES



# NOTES



**SILENTSYS**  
ultralow noise systems