PicoScope 5000 Series Data Acquisition Software

Guide d'utilisation et de compréhension

PRÉPARÉ PAR
JOSHUA THEBAUDIN,
STAGIAIRE PREMIERE
ANNÉE BTS SIO.

SILENTSYS
ultralow noise systems

SOMMAIRE

Bienvenue

sur ce guide d'utilisation et de compréhension.

INTRODUCTION

L'application "PicoScope 5000 Data Acquisition Software" a été développé 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.

Veuillez nous excuser des potentiels gènes ou difficulté que vous pourrez rencontrer.

Joshua.

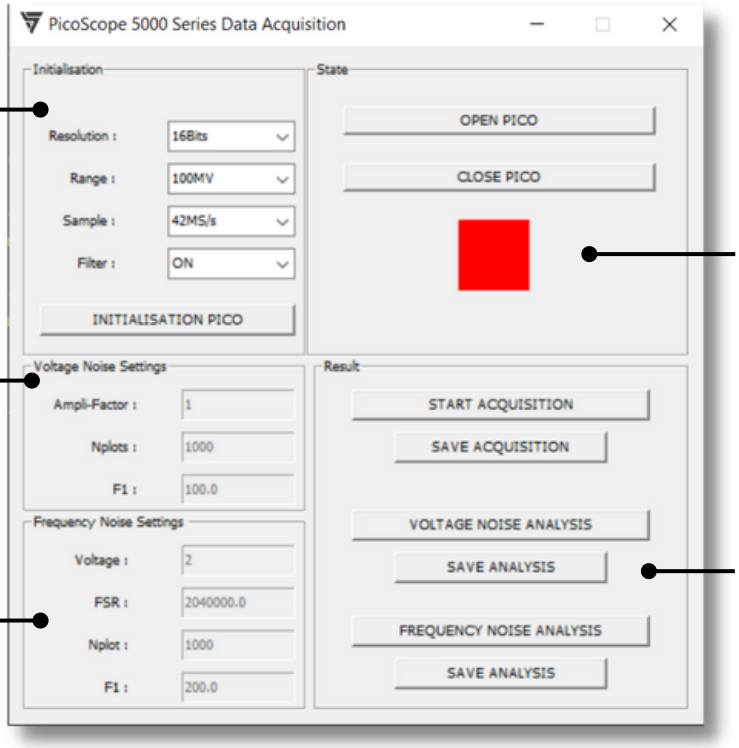
Vision globale de l'application

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 leVoltage 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.

```python
#--------- IMPORTATION LIBRAIRIE ---------#
from tkinter import *
from tkinter.messagebox import showinfo
from tkinter.ttk import Combobox
import tkinter.font
import importlib
D = importlib.import_module('Def_file')
from tkinter import *
import warnings
warnings.filterwarnings('ignore') #just use this if warnings are annoying
```

```python
--------- CLASSE CONTENANT TOUS LES WIDGETS ---------#
class Widget1():
    def __init__(self, parent):
        self.gui(parent)

    def gui(self, parent):
        if parent == 0 :
            self.w1 = Tk()
            self.w1.geometry('490x480')
            self.w1.title('PicoScope 5000 Series Data Acquisition')
            self.w1.resizable(0, 0)

            self.w1.iconbitmap("images/SILENTSYS_BLEU_PICTO_RVB-Web.ico")

        else:
            self.w1 = Frame(parent)
            self.w1.place(x = 0, y = 0, width = 490, height = 480)


        #------ FRAME DU STATE ------#
        self.group2 = LabelFrame(self.w1, text = "State", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow")
        self.group2.place(x = 210, y = 10, width = 270, height = 210)

        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)
        self.button10.place(x = 24, y = 20, width = 220, height = 20)


        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)
        self.button11.place(x = 24, y = 60, width = 220, height = 20)


        D.led = self.state = Frame(self.group2, width = 100, height = 100, bg = "red")

        self.state.place(x = 105, y = 100, width = 50, height = 50)
```

```python
#------ FRAME DE L'INITIALISATION ------#
        self.group3 = LabelFrame(self.w1, text = "Initialisation", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow")
        self.group3.place(x = 10, y = 10, width = 200, height = 210)

        #------ LISTE DEROULANTE RANGE ------#
        D.range = self.combo1 = Combobox(self.group3, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.combo1.place(x = 100, y = 60, width = 90, height = 22)
        self.combo1['values'] = ("10MV","20MV","50MV","100MV","200MV","500MV", "1V","2V","5V","10V")
        self.combo1.current(3)
        self.label1 = Label(self.group3, text = "Range :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label1.place(x = 12, y = 60, width = 70, height = 22)

        #------ LISTE DEROULANTE RESOLUTION ------#
        self.label2 = Label(self.group3, text = "Resolution : ", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label2.place(x = 7, y = 30, width = 70, height = 22)
        D.reso = self.combo2 = Combobox(self.group3, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.combo2.place(x = 100, y = 30, width = 90, height = 22)
        self.combo2['values'] = ("16Bits")
        self.combo2.current(0)

        #------ LISTE DEROULANTE SAMPLE ------#
        self.label3 = Label(self.group3, text = "Sample :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label3.place(x = 10, y = 90, width = 70, height = 22)
        D.sample = self.combo3 = Combobox(self.group3, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.combo3.place(x = 100, y = 90, width = 90, height = 22)
        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")
        self.combo3.current(1)

        #------ LISTE DEROULANTE FILTER ------#
        self.label4 = Label(self.group3, text = "Filter :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label4.place(x = 5, y = 120, width = 90, height = 22)
        D.filter = self.combo4 = Combobox(self.group3, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.combo4.place(x = 100, y = 120, width = 90, height = 22)
        self.combo4['values'] = ("ON", "OFF")
        self.combo4.current(0)
```

Vision globale du code : interface graphique (2)

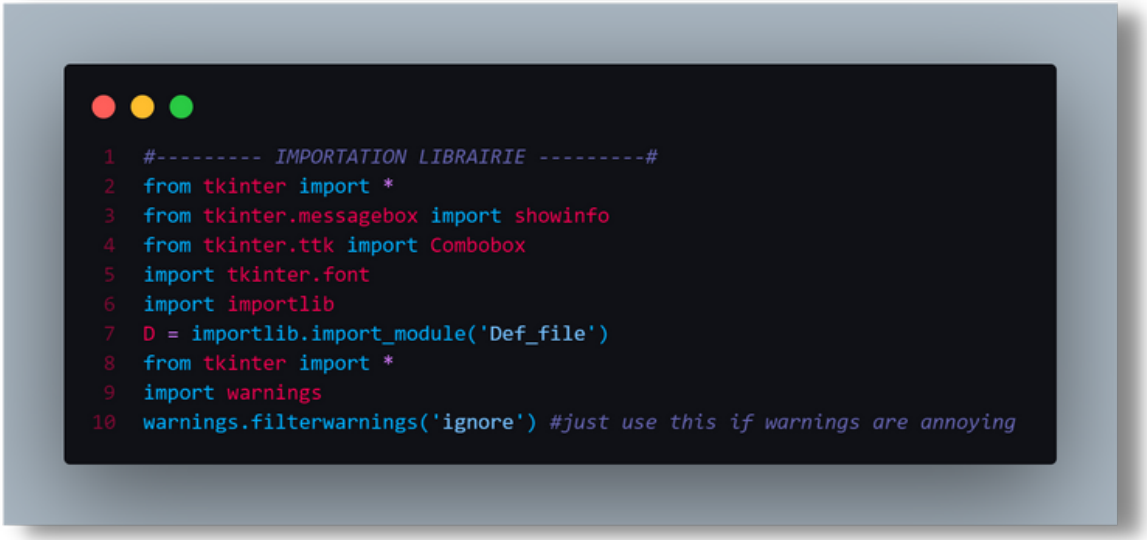
Regardons le code de l'interface graphique.

```
#------ BOUTON INITIALISATION------#
        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)
        self.button5.place(x = 10, y = 160, width = 180, height = 22)

        #------ FRAME DES VOLTAGE NOISE SETTINGS ------#
        self.group6 = LabelFrame(self.w1, text = "Voltage Noise Settings", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow")
        self.group6.place(x = 10, y = 220, width = 200, height = 110)

        #------ AMPLIFACTOR ZONE ------#
        self.label5 = Label(self.group6, text = "Ampli-Factor :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label5.place(x = 4, y = 10, width = 90, height = 22)

        def click(event):                   #PERMET DE METTRE EN PLACE LE PLACEHOLDER DANS LE TEXTE (1), CELA SE REPETERA POUR CHAQUE CHAMP DE TEXTE
            self.ltext5.delete(0, END)

        self.ltext5 = Entry(self.group6, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.ltext5.place(x = 110, y = 10, width = 80, height = 22)

        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 DEFAUT
        self.ltext5.config(state=DISABLED)
        self.ltext5.bind("<Button-1>", click)

        #--- NPLOT ZONE ------#
        self.label6 = Label(self.group6, text = "Nplots :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label6.place(x = 17, y = 40, width = 90, height = 22)

        def click(event):
            self.ltext6.config(state=NORMAL)
            self.ltext6.delete(0, END)

        self.ltext6 = Entry(self.group6, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.ltext6.place(x = 110, y = 40, width = 80, height = 22)

        self.ltext6.insert(0, 1000)         #1000 EN VALEUR PAR DEFAUT
        self.ltext6.config(state=DISABLED)
        self.ltext6.bind("<Button-1>", click)

        #--- F1 ZONE ------#
        self.label7 = Label(self.group6, text = "F1 :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label7.place(x = 25, y = 70, width = 90, height = 22)

        def click(event):
            self.ltext7.config(state=NORMAL)
            self.ltext7.delete(0, END)

        self.ltext7 = Entry(self.group6, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.ltext7.place(x = 110, y = 70, width = 80, height = 22)

        self.ltext7.insert(0,1*1e2)         #1*1e2 EN VALEUR PAR DEFAUT
        self.ltext7.config(state=DISABLED)
        self.ltext7.bind("<Button-1>", click)
```

```
#------ FRAME DU FREQUENCE NOISE SETTINGS ------#
        self.group4 = LabelFrame(self.w1, text = "Frequency Noise Settings ", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow")
        self.group4.place(x = 10, y = 330, width = 200, height = 140)

        #------ VOLTAGE ZONE ------#
        self.label15 = Label(self.group4, text = "Voltage :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label15.place(x = 8, y = 10, width = 90, height = 22)

        def click(event):
            self.ltext1.config(state=NORMAL)
            self.ltext1.delete(0, END)

        self.ltext1 = Entry(self.group4, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")

        self.ltext1.insert(0, 2)            #2 EN VALEUR PAR DEFAUT
        self.ltext1.config(state = DISABLED)
        self.ltext1.bind("<Button-1>", click)

        self.ltext1.place(x = 110, y = 10, width = 80, height = 22)

        #------ FSR ZONE ------#
        self.label17 = Label(self.group4, text = "FSR :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label17.place(x = 15, y = 40, width = 90, height = 22)

        def click(event):
            self.ltext2.config(state=NORMAL)
            self.ltext2.delete(0, END)

        self.ltext2 = Entry(self.group4, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.ltext2.place(x = 110, y = 40, width = 80, height = 22)

        self.ltext2.insert (0, 2.04 * 1e6) #2.04 * 1e6 EN VALEUR PAR DEFAUT
        self.ltext2.config (state=DISABLED)
        self.ltext2.bind ("<Button-1>", click)

        #------ NPLOT ZONE ------#
        self.label18 = Label(self.group4, text = "Nplot : ", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label18.place(x = 15, y = 70, width = 90, height = 22)

        def click(event):
            self.ltext3.config(state=NORMAL)
            self.ltext3.delete(0, END)

        self.ltext3 = Entry(self.group4, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.ltext3.place(x = 110, y = 70, width = 80, height = 22)

        self.ltext3.insert (0, 1000) #1000 EN VALEUR PAR DEFAUT
        self.ltext3.config (state=DISABLED)
        self.ltext3.bind ("<Button-1>", click)

        #------ F1 ZONE ------#
        self.label19 = Label(self.group4, text = "F1 : ", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label19.place(x = 20, y = 100, width = 90, height = 22)

        def click(event):
            self.ltext4.config(state=NORMAL)
            self.ltext4.delete(0, END)

        self.ltext4 = Entry(self.group4, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.ltext4.place(x = 110, y = 100, width = 80, height = 22)

        self.ltext4.insert (0, 2*1e2) #2*1e2 EN VALEUR PAR DEFAUT
        self.ltext4.config (state=DISABLED)
        self.ltext4.bind ("<Button-1>", click)
```

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.



```python
#--------- IMPORTATION LIBRAIRIE ---------#
from tkinter import *
from tkinter.messagebox import showinfo
from tkinter.ttk import Combobox
import tkinter.font
import importlib
D = importlib.import_module('Def_file')
from tkinter import *
import warnings
warnings.filterwarnings('ignore') #just use this if warnings are annoying
```

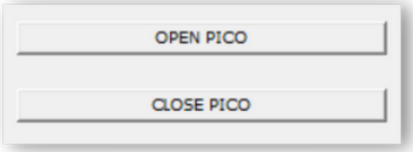
Présente au début du code, **l'importation de librairies est une partie très importante car elle permet de faire fonctionner correctement l'application.** On y retrouve Tkinter et ses modules qui sont très important pour notre interface : **Showinfo** permet de créer des pop-ups, **Combobox** permet 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)

```python
--------- CLASSE CONTENANT TOUS LES WIDGETS ---------#
class Widget1():
    def __init__(self, parent):
        self.gui(parent)

    def gui(self, parent):
        if parent == 0:
            self.w1 = Tk()
            self.w1.geometry('490x480')
            self.w1.title('PicoScope 5000 Series Data Acquisition')
            self.w1.resizable(0, 0)

            self.w1.iconbitmap("images/SILENTSYS_BLEU_PICTO_RVB-Web.ico")

        else:
            self.w1 = Frame(parent)
            self.w1.place(x = 0, y = 0, width = 490, height = 480)


        #------ FRAME DU STATE ------#
        self.group2 = LabelFrame(self.w1, text = "State", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow")
        self.group2.place(x = 210, y = 10, width = 270, height = 210)

        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)
        self.button10.place(x = 24, y = 20, width = 220, height = 20)



        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)
        self.button11.place(x = 24, y = 60, width = 220, height = 20)


        D.led = self.state = Frame(self.group2, width = 100, height = 100, bg = "red")

        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.

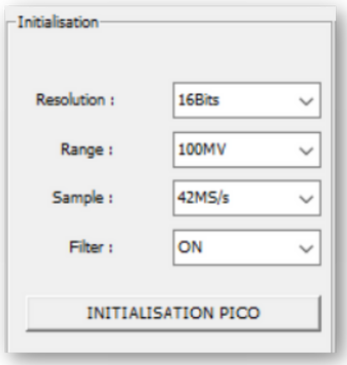


Le D.led correspond au carré rouge que l'on peut voir sur l'interface. Cette variable est relié au fichier des fonctions et de ce fait, selon l'état du PicoScope, la couleur change.

L'INTERFACE GRAPHIQUE (3)

```
#------ FRAME DE L'INITIALISATION ------#
        self.group3 = LabelFrame(self.w1, text = "Initialisation", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow")
        self.group3.place(x = 10, y = 10, width = 200, height = 210)

        #------ LISTE DEROULANTE RANGE ------#
        D.range = self.combo1 = Combobox(self.group3, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.combo1.place(x = 100, y = 60, width = 90, height = 22)
        self.combo1['values'] = ("10MV","20MV","50MV","100MV","200MV","500MV", "1V","2V","5V","10V")
        self.combo1.current(3)
        self.label1 = Label(self.group3, text = "Range :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label1.place(x = 12, y = 60, width = 70, height = 22)

        #------ LISTE DEROULANTE RESOLUTION ------#
        self.label2 = Label(self.group3, text = "Resolution : ", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label2.place(x = 7, y = 30, width = 70, height = 22)
        D.reso = self.combo2 = Combobox(self.group3, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.combo2.place(x = 100, y = 30, width = 90, height = 22)
        self.combo2['values'] = ("16Bits")
        self.combo2.current(0)

        #------ LISTE DEROULANTE SAMPLE ------#
        self.label3 = Label(self.group3, text = "Sample :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label3.place(x = 10, y = 90, width = 70, height = 22)
        D.sample = self.combo3 = Combobox(self.group3, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.combo3.place(x = 100, y = 90, width = 90, height = 22)
        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")
        self.combo3.current(1)

        #------ LISTE DEROULANTE FILTER ------#
        self.label4 = Label(self.group3, text = "Filter :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label4.place(x = 5, y = 120, width = 90, height = 22)
        D.filter = self.combo4 = Combobox(self.group3, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.combo4.place(x = 100, y = 120, width = 90, height = 22)
        self.combo4['values'] = ("ON", "OFF")
        self.combo4.current(0)

        #------ BOUTON INITIALISATION------#
        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)
        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)

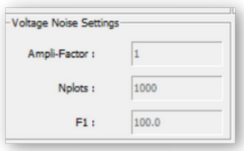
Puis, toutes ces valeurs sont envoyées par le bouton INITIALISATION.

L'INTERFACE GRAPHIQUE (4)

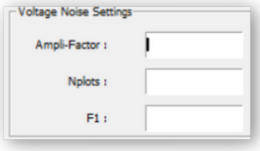
```python
#------ FRAME DES VOLTAGE NOISE SETTINGS ------#
        self.group6 = LabelFrame(self.w1, text = "Voltage Noise Settings", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow")
        self.group6.place(x = 10, y = 220, width = 200, height = 110)

        #------ AMPLIFACTOR ZONE ------#
        self.label5 = Label(self.group6, text = "Ampli-Factor :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label5.place(x = 4, y = 10, width = 90, height = 22)

        def click(event):                   #PERMET DE METTRE EN PLACE LE PLACEHOLDER DANS LE TEXTE (1), CELA SE REPETERA POUR CHAQUE CHAMP DE TEXTE
            self.ltext5.delete(0, END)

        self.ltext5 = Entry(self.group6, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.ltext5.place(x = 110, y = 10, width = 80, height = 22)

        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 DEFAUT
        self.ltext5.config(state=DISABLED)
        self.ltext5.bind("<Button-1>", click)

        #--- NPLOT ZONE ------#
        self.label6 = Label(self.group6, text = "Nplots :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label6.place(x = 17, y = 40, width = 90, height = 22)

        def click(event):
            self.ltext6.config(state=NORMAL)
            self.ltext6.delete(0, END)

        self.ltext6 = Entry(self.group6, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.ltext6.place(x = 110, y = 40, width = 80, height = 22)

        self.ltext6.insert(0, 1000)         #1000 EN VALEUR PAR DEFAUT
        self.ltext6.config(state=DISABLED)
        self.ltext6.bind("<Button-1>", click)

        #--- F1 ZONE ------#
        self.label7 = Label(self.group6, text = "F1 :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label7.place(x = 25, y = 70, width = 90, height = 22)

        def click(event):
            self.ltext7.config(state=NORMAL)
            self.ltext7.delete(0, END)

        self.ltext7 = Entry(self.group6, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.ltext7.place(x = 110, y = 70, width = 80, height = 22)

        self.ltext7.insert(0,1*1e2)         #1*1e2 EN VALEUR PAR DEFAUT
        self.ltext7.config(state=DISABLED)
        self.ltext7.bind("<Button-1>", click)
```

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.

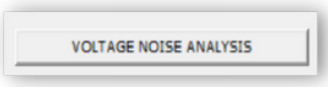
Cependant, ici, il y a des valeurs par défauts qui sont mis en place par le **def click(event) :** on initialise le champ de texte dans un état normal (grisé) qui contient la valeur mise dans le insert (exemple : self.ltext6.insert(0,1000)).



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.



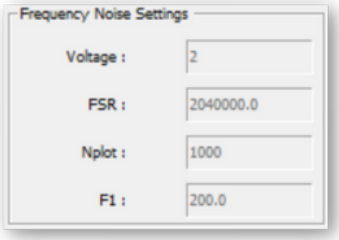
Tous ces paramètres sont envoyés et sont pris en compte grâce au bouton VOLTAGE NOISE ANALYSIS

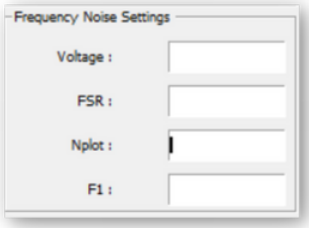


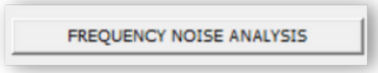
L'INTERFACE GRAPHIQUE (5)

```python
#----- FRAME DU FREQUENCE NOISE SETTINGS ------#
        self.group4 = LabelFrame(self.w1, text = "Frequency Noise Settings ", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow")
        self.group4.place(x = 10, y = 330, width = 200, height = 140)

        #----- VOLTAGE ZONE ------#
        self.label5 = Label(self.group4, text = "Voltage :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label5.place(x = 8, y = 10, width = 90, height = 22)

        def click(event):
            self.ltext1.config(state=NORMAL)
            self.ltext1.delete(0, END)

        self.ltext1 = Entry(self.group4, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")

        self.ltext1.insert(0, 2)            #2 EN VALEUR PAR DEFAUT
        self.ltext1.config(state = DISABLED)
        self.ltext1.bind("<Button-1>", click)

        self.ltext1.place(x = 110, y = 10, width = 80, height = 22)

        #----- FSR ZONE ------#
        self.label7 = Label(self.group4, text = "FSR :", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label7.place(x = 15, y = 40, width = 90, height = 22)

        def click(event):
            self.ltext2.config(state=NORMAL)
            self.ltext2.delete(0, END)

        self.ltext2 = Entry(self.group4, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.ltext2.place(x = 110, y = 40, width = 80, height = 22)

        self.ltext2.insert (0, 2.04 * 1e6) #2.04 * 1e6 EN VALEUR PAR DEFAUT
        self.ltext2.config (state=DISABLED)
        self.ltext2.bind ("<Button-1>", click)

        #----- NPLOT ZONE ------#
        self.label8 = Label(self.group4, text = "Nplot : ", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label8.place(x = 15, y = 70, width = 90, height = 22)

        def click(event):
            self.ltext3.config(state=NORMAL)
            self.ltext3.delete(0, END)

        self.ltext3 = Entry(self.group4, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.ltext3.place(x = 110, y = 70, width = 80, height = 22)

        self.ltext3.insert (0, 1000) #1000 EN VALEUR PAR DEFAUT
        self.ltext3.config (state=DISABLED)
        self.ltext3.bind ("<Button-1>", click)

        #----- F1 ZONE ------#
        self.label9 = Label(self.group4, text = "F1 : ", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.label9.place(x = 20, y = 100, width = 90, height = 22)

        def click(event):
            self.ltext4.config(state=NORMAL)
            self.ltext4.delete(0, END)

        self.ltext4 = Entry(self.group4, font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow", state = "normal")
        self.ltext4.place(x = 110, y = 100, width = 80, height = 22)

        self.ltext4.insert (0, 2*1e2) #2*1e2 EN VALEUR PAR DEFAUT
        self.ltext4.config (state=DISABLED)
        self.ltext4.bind ("<Button-1>", click)
```

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





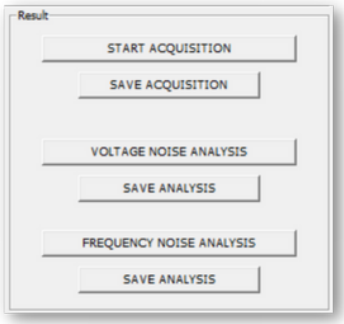


L'INTERFACE GRAPHIQUE (6)

```python
#----- FRAME DES RESULTATS ------#
        self.group5 = LabelFrame(self.w1, text = "Result", font = tkinter.font.Font(family = "MS Shell Dlg 2", size = 7), cursor = "arrow")
        self.group5.place(x = 215, y = 220, width = 265, height = 250)

        #----- START ACQUI BUTTON ------#
        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.startaqui)
        self.button7.place(x = 25, y = 10, width = 210, height = 22)



        #----- SAVE ACQUI BUTTON ------#
        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.saveaqui)
        self.button12.place(x = 55, y = 40, width = 150, height = 22)

        #----- VOLTAGE NOISE BUTTON ------#
        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.ltext5, self.ltext6, self.ltext7))
        self.button8.place(x = 25, y = 95, width = 210, height = 22)

        #----- SAVE ANALYSIS BUTTON ------#
        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.sauvegardevnoise)
        self.button13.place(x = 55, y = 125, width = 150, height = 22)

        #----- FREQUENCY NOISE RESULT ------#
        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.ltext1, self.ltext2, self.ltext3, self.ltext4))
        self.button14.place(x = 25, y = 170, width = 210, height = 22)

        #----- SAVE ANALYSIS BUTTON ------#
        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)
        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 ltext et de les exécuter en même temps.




```python
if __name__ == '__main__':
    print("---------Welcome to the PicoScope 5000 Series Data Acquisition Software---------")

    a = Widget1(0)
    showinfo ("Welcome", "Welcome to the PicoScope 5000 Series Data Acquisition Software")
    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

```python
#----- IMPORTATION (1) -----#
from tkinter import *
from tkinter.ttk import *
import time
from tkinter import N, filedialog, END, Toplevel
from tkinter.messagebox import showinfo
from tkinter.simpledialog import askstring
import importlib


P = importlib.import_module ('PicoScope 5000 Series Data Acquisition Software')

#----GLOBAL VARIABLES-----#
global led, reso, range, sample, filter, noise, status_pico, U_meas, status_init_pico, f_plot1_vnoise, PSD_SSB_plot1_vnoise, U_meas0, v_pico, i_pico, s_pico, vnoise_open, acqui_pico, PSD_SSB_plot1, f_plot1, freq_pico


#-----INITIALIZATION-----#
VnoiseRMS = 0
nbnoise = 0
nbacqui = 0
nbfreq = 0
nbRMS = 0
noise = None
status_pico = False
freq_pico = False
i_pico = False
s_pico = False
save_pico = False
v_pico = False
vnoise_open = False
acqui_pico = False


#-----IMPORTATION (2)-----#
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
import scipy.integrate as spi
import time
import warnings
warnings.filterwarnings ('ignore')  # just use this if warnings are annoying

import ctypes
from picosdk.ps5000a import ps5000a as ps
from picosdk.functions import adc2mV, assert_pico_ok, mV2adc
```

```python
class Picoscope_channelA:
    def __init__(self, Samples_per_channel=int (1e4), triggerlevel_mV=0):

        Samples_per_channel = int (Samples_per_channel)
        self.triggerlevel_mV = triggerlevel_mV
        self.chandle = ctypes.c_int16 ()
        self.status = {}
        self.resolution = ps.PS5000A_DEVICE_RESOLUTION["PS5000A_DR_16BIT"]
        self.status["openunit"] = ps.ps5000aOpenUnit (ctypes.byref (self.chandle), None, self.resolution)
        # PICO_STATUS ps5000aOpenUnit(int16_t * handle,
        #       int8_t * serial,  None means it just takes the first scope found
        #       PS5000A_DEVICE_RESOLUTION resolution)
        assert_pico_ok (self.status["openunit"])

        # print('picoscope is open now')

        # Disable other channels
        # Set up channel B
        # handle = chandle
        self.channelB = ps.PS5000A_CHANNEL["PS5000A_CHANNEL_B"]
        # enabled = 1
        self.coupling_type = ps.PS5000A_COUPLING["PS5000A_DC"]
        #       self.coupling_type = ps.PS5000A_COUPLING["PS5000A_AC"]
        #       typedef enum enPS5000ACoupling{PS5000A_AC,PS5000A_DC} PS5000A_COUPLING
        self.chBRange = ps.PS5000A_RANGE["PS5000A_1V"]  # it was 20V before.
        ##      self.chARange = ps.PS5000A_RANGE["PS5000A_200MV"] # it was 20V before.
        #       typedef enum enPS5000ARange{
        #       PS5000A_10MV,
        #       PS5000A_20MV,
        #       PS5000A_50MV,
        #       PS5000A_100MV,
        #       PS5000A_200MV,
        #       PS5000A_500MV,
        #       PS5000A_1V,
        #       PS5000A_2V,
        #       PS5000A_5V,
        #       PS5000A_10V,
        #       PS5000A_20V,
        #       PS5000A_50V,
        #       PS5000A_MAX_RANGES
        #       } PS5000A_RANGE;
        # Last entry is analogue offset = 0 V
        self.status["setChB"] = ps.ps5000aSetChannel (self.chandle, self.channelB, 0, self.coupling_type, self.chBRange,
                                                      0)
        #       PICO_STATUS ps5000aSetChannel(int16_t handle,PS5000A_CHANNEL channel,int16_t enabled,PS5000A_COUPLING type,
        #                                     PS5000A_RANGE range,float analogueOffset)
        assert_pico_ok (self.status["setChB"])

        # Set up channel C
        # handle = chandle
        self.channelC = ps.PS5000A_CHANNEL["PS5000A_CHANNEL_C"]
        # enabled = 1
        self.coupling_type = ps.PS5000A_COUPLING["PS5000A_DC"]
        #       self.coupling_type = ps.PS5000A_COUPLING["PS5000A_AC"]
        #       typedef enum enPS5000ACoupling{PS5000A_AC,PS5000A_DC} PS5000A_COUPLING
        self.chCRange = ps.PS5000A_RANGE["PS5000A_1V"]  # it was 20V before.
        ##      self.chARange = ps.PS5000A_RANGE["PS5000A_200MV"] # it was 20V before.
        #       typedef enum enPS5000ARange{
        #       PS5000A_10MV,
        #       PS5000A_20MV,
        #       PS5000A_50MV,
        #       PS5000A_100MV,
        #       PS5000A_200MV,
        #       PS5000A_500MV,
        #       PS5000A_1V,
        #       PS5000A_2V,
        #       PS5000A_5V,
        #       PS5000A_10V,
        #       PS5000A_20V,
        #       PS5000A_50V,
        #       PS5000A_MAX_RANGES
        #       } PS5000A_RANGE;
        # Last entry is analogue offset = 0 V
        self.status["setChC"] = ps.ps5000aSetChannel (self.chandle, self.channelC, 0, self.coupling_type, self.chCRange,
                                                      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

```python
class ProcessNoise (object):  # TODO live class
    """
    Class used for a adding live update to a figure. Run `%matplotlib notebook` in jupyter notebook
    for optimal operation of this class. This adds

    Attributes
    ----------
    needs explanation"""

    def __init__(self, init_ps=True, Nsamples=int (2 ** 25)):

        #####
        ##Initialize Picoscope
        #####
        self.Nsamples = Nsamples
        if init_ps:
            try:
                self.picoscope = Picoscope_channelA (Samples_per_channel=self.Nsamples)

            except:
                Picoscope_channelA.close ()  # check if self needed here
                self.picoscope = Picoscope_4channels (Samples_per_channel=self.Nsamples)
                print ('picoscope was already open and was now reinitialized')

            self.picoscope.N = self.Nsamples
            self.picoscope.set_timebase (5)  # for 12BIT 2 is 4ns Sampling interval => 250MS/s; 3 is 8ns => 125MS/s
            # check on page 22 in programmers-guide 3Awith 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,
            # 8 is 20 MS/s, 9 is 18MS/s,
            # 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
            self.picoscope.set_BandwidthFilter (1)  # this sets the 20MHz lowpassfilter for all channels
            self.picoscope.R = 1 / (self.picoscope.timeIntervalns.value * 1e-9)  # sampling rate
            self.timeS = sp.linspace (0, (self.picoscope.N - 1) / self.picoscope.R,
                                      self.picoscope.N)  # check if this will be needed for the future
            self.Noise_start_acquisition ()
            self.picoscope.check_ready ()  # this is needed to write the data to the buffer for the readout.
            # here we could save time to evaluate while writing into the buffer at the same time
            time.sleep (1)  # this is to make sure that the AC coupling has time to stabilize
            self.Noise_start_acquisition ()
            self.picoscope.check_ready ()  # this is needed to write the data to the buffer for the readout.
            self.picoscope.readout_buffer ()

            self.time = sp.linspace (0, (self.picoscope.N - 1) / self.picoscope.R, self.picoscope.N)
            self.signal_A = sp.array (self.picoscope.bufferA)

    #            self.N_FFT = Nsamples
    #            self.dt = 1/self.picoscope.R
    #            self.freq = spf.fftfreq(self.N_FFT,self.dt)
    #            self.signal_FFT1 = sp.zeros(self.N_FFT)
    #            self.signal_FFT1_phase = sp.zeros(self.N_FFT)
    #            self.signal_FFT1_amplitude = sp.ones(self.N_FFT)

    def Noise_start_acquisition(self):
        self.picoscope.start_acquiredata_triggered ()  # this writes the data into the buffer

    def update_data(self):
        self.Noise_start_acquisition ()  # this starts the acquisition
        self.picoscope.check_ready ()  # this is needed to write the data to the buffer for the readout.
        # here we could save time to evaluate while writing into the buffer at the same time
        self.picoscope.readout_buffer ()
        self.signal_A = sp.array (
            adc2mV (self.picoscope.bufferA, self.picoscope.chARange, self.picoscope.maxADC)) * 1e-3
        self.signal_Noise1 = self.signal_A

    def save_data(self, fn):
        data = sp.transpose ([self.time, self.signal_Noise1])
        sp.savetxt (fn, data, fmt='%0.6e', header='time (s), signal A (V)', delimiter=', ', newline='\n')

    def save_dataraw(self, fn):
        data = sp.transpose ([self.time, self.signal_Noise1])
        sp.savetxt (fn, data, fmt='%0.6e', header='time (s), signal A (V)', delimiter=', ', newline='\n')

    def load_rawdata(self, fn):  # speed up this function todo
        self.time, self.signal_A = sp.loadtxt (fn, unpack=True, delimiter=', ', skiprows=1)
```

```python
def open_pico():  # work
    global status_pico, noise
    noise = ProcessNoise (init_ps=True, Nsamples=int (2 ** 25))  # up to 2^25 should be ok for the PS5444D;
    status_pico = True
    if status_pico == True:
        led.config (bg='green')
        return showinfo ('Great','Picoscope is open')


# 2^24 are too many samples for buffering for the PS5442D used for testing.
# execute cell with noise.picoscope.close() below if an error appears

def close_pico():  # work
    global status_pico, s_pico, i_pico, v_pico, vnoise_open, acqui_pico, freq_pico
    noise.picoscope.close ()
    status_pico = False
    if status_pico == False:
        led.config (bg='red')
        return showinfo ('Warning','Picoscope is closed')
    #quand on ferme le picoscope, on réinitialise les popups
    s_pico = False
    i_pico = False
    v_pico = False
    vnoise_open = False
    acqui_pico = False
    freq_pico = False
```

Vision globale du code : les fonctions (3)

Regardons le code des fonctions

```python
def init_pico():
    global i_pico

    if status_pico == False:
        return showinfo ('Error', 'Picoscope is not open')

    range_tab = [
        # tableau associatif pour les valeurs de range
        {'name': '10MV', 'value': 0},
        {'name': '20MV', 'value': 1},
        {'name': '50MV', 'value': 2},
        {'name': '100MV', 'value': 3},
        {'name': '200MV', 'value': 4},
        {'name': '500MV', 'value': 5},
        {'name': '1V', 'value': 6},
        {'name': '2V', 'value': 7},
        {'name': '5V', 'value': 8},
        {'name': '10V', 'value': 10},
        {'name': '20V', 'value': 11},
        {'name': '50V', 'value': 12},
    ]
    timebase_tab = [
        # tableau associatif pour les valeurs de timebase

        {'name': '64MS/s', 'value': 4},
        {'name': '42MS/s', 'value': 5},
        {'name': '32MS/s', 'value': 6},
        {'name': '25MS/s', 'value': 7},
        {'name': '20MS/s', 'value': 8},
        {'name': '18MS/s', 'value': 9},
        {'name': '16MS/s', 'value': 10},
        {'name': '14MS/s', 'value': 11},
        {'name': '12.5MS/s', 'value': 12},
        {'name': '11MS/s', 'value': 13},
        {'name': '10MS/s', 'value': 14},

    ]
    filter_tab = [
        # tableau associatif pour les valeurs de filter
        {'name': 'OFF', 'value': 1},
        {'name': 'ON', 'value': 0},
    ]

    resolution_tab = [
        # tableau associatif pour les valeurs de resolution
        {'name': '16Bits', 'value': 16}
    ]

    for r in resolution_tab:
        if r['name'] == reso.get ():
            getreso = r['value']
            print ("reso : " + str (getreso))

    for i in range_tab:
        if i['name'] == range.get ():
            getrange = i['value']
            print ("range : " + str (getrange))

    for j in timebase_tab:
        if j['name'] == sample.get ():
            gettime = j['value']
            print ("time : " + str (gettime))

    for x in filter_tab:
        if x['name'] == filter.get ():
            getfilter = x['value']
            print ("filter : " + str (getfilter))

    # save the values in the picoscope
    noise.picoscope.set_timebase (gettime)
    noise.picoscope.set_range (getrange)
    noise.picoscope.set_resolution (getreso)
    noise.picoscope.set_BandwidthFilter (getfilter)
    i_pico = True
    showinfo ('Info', 'Picoscope is initialized')
```

```python
def startaqui():
    global U_meas, s_pico, U_meas0, v_pico, acqui_pico, nbacqui

    nbacqui=nbacqui+1

    if status_pico == False:
        return showinfo ('Warning', 'Picoscope is not open, please open it')
    if i_pico == False:
        return showinfo ('Warning', 'Picoscope is not initialized, please initialize the picoscope')
    if i_pico == True:
        showinfo ('Info', 'Acquisition is started. This can take a while. After that you can save the data and launch the Voltage Noise Acquisition. ')

        v_pico = True

    noise.update_data ()

    U_meas0 = noise.signal_Noise1
    U_meas0.max ()

    U_meas0.min ()

    U_meas = (U_meas0 - U_meas0.mean ())

    fig0 = plt.figure ()
    plt.plot (U_meas0)

    plt.autoscale (enable=True, axis='y', tight=True)
    plt.title ('Time Trace ' + str(nbacqui))
    plt.ylabel ('Tension (V)')

    plt.xlabel ('Npts (s)')

    acqui_pico = True
    plt.show ()
```

```python
def saveaqui():
    global U_meas, s_pico, U_meas0, v_pico, acqui_pico

    #si l'acquisition n'est pas Lancée, on ne peut pas sauvegarder
    if v_pico == False:
        return showinfo ('Warning', 'Acquisition is not started, please start the acquisition')

    #si on ne lance pas l'acquisition, on ne peut pas sauvegarder
    if acqui_pico == False:
        return showinfo ('Warning', 'Acquisition is not started, please start the acquisition before saving datas')



    #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 i_pico == False:
        return showinfo ('Warning', 'Picoscope is not initialized, please initialize the picoscope')



    global U_meas0
    path = filedialog.askdirectory (initialdir = "/",title = "Select directory")

    #choose the name of the txt when we are in the folder
    name = askstring ("Name of the txt", "Name of the txt")

    if name == None:
        return showinfo ("Warning", "You did not enter a name")
    if path != None:
        data2 = sp.transpose([U_meas0])
        np.savetxt(path + '/' + name + '.txt', data2, delimiter='\t', fmt='%0.6e')
        showinfo ('Info', 'Txt saved')
```

Vision globale du code : les fonctions (4)

Regardons le code des fonctions

```python
def vnoise(ampfact, 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 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 r_pico == False:
        return showinfo ('Warning', 'Picoscope is not started, please start the acquisition')



    if ampfact.get() == '' and Nplot.get() == '' and f1.get() == '':
        return showinfo ('Warning', 'You did not enter any values')

    if ampfact.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 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_vnoise, PSD_SSB_plot1_vnoise, vnoise_open, nbnoise, VnoiseRMS


    nbnoise = nbnoise + 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


    ampfact = int (ampfact.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 = ampfact  # for the voltage amplification
    dU = 1 / amplificationfactor * U_meas
    dUf = np.absolute (spfft.fft (dU))
    PSD_SSB = dUf[: 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 R&S FSWP
    f = f[: int (N / 2)]

    tic = time.time ()
    # N_plot = 1000  # for points above f1
    # f1 = 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 = (122 / 255, 186 / 255, 76 / 255, 1)
    silentsys_rouge = (254 / 255, 27/ 255, 0/255, 1)
    ax1.loglog (f_plot1_vnoise, np.sqrt(PSD_SSB_plot1_vnoise)*1e9, color=silentsys_rouge)
    #ax1.loglog(f_plot2,np.sqrt(PSD_SSB_plot2)*1e9,color=silentsys_vert)
    ax1.set_xlim (1e0, 1e7)
    ax1.set_ylim (1e1, 1e6)
    ax1.set_ylabel ('Voltage Noise PSD (nV/rHz)')
    ax1.set_xlabel ('Fourier Frequency (Hz)')
    img2 = plt.imread('images/SILENTSYS_VERT_Pantone_10pc.png')
    fig1.tight_layout ()


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

    vnoise_open = True
    plt.show ()
```

```python
#-----------------------------------RMS-----------------------------------------


    startfreq_vnoise = 0.9 #Hz
    stopfreq_vnoise = 1e6 #Hz

    #Integrated noise
    freqNoise = PSD_SSB_plot1_vnoise
    range2 = sp.where((f_plot1_vnoise >= startfreq_vnoise)*(f_plot1_vnoise <= stopfreq_vnoise))
    freq, freqNoise = f_plot1_vnoise[range2], freqNoise[range2]
    RMSVNoise = sp.sqrt(-spi.cumtrapz(2*freqNoise[::-1],freq[::-1],initial=0))[::-1]#the minus is because of the backward integration



    fig2, ax2 = plt.subplots()
    ax2.set_xlabel('Lower limit frequency (Hz)')
    ax2.set_ylabel('RMS Voltage (uV)')

    ax2.set_xlim(startfreq_vnoise, stopfreq_vnoise)
    #axe y autoscale
    ax2.autoscale(enable=True, axis='y', tight=True)
    #ax2.set_ylim(0, plt.autoscale)

    RMSVNoise = RMSVNoise*1e6

    ax2.semilogx(freq, RMSVNoise,color=silentsys_vert, label='RMS Voltage: {0:.2f} µV ({1:.1f} Hz to {2:.1f} MHz)'.format(RMSVNoise[0],freq.min(),freq.max()*1e-6))


    ax2.legend(frameon=False)
    fig2.tight_layout()
    plt.title('Voltage Noise RMS ' + str(VnoiseRMS))
    img = plt.imread('images/SILENTSYS_VERT_Pantone_10pc.png')
    axlogo = plt.axes([0.2, 0.3, 0.7, 0.5])
    axlogo.set_axis_off()
    axlogo.imshow(img)
    plt.show()
```

Vision globale du code : les fonctions (5)

Regardons le code des fonctions

```python
def sauvegardevnoise():


    #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 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', 'Start the Voltage Noise acquisition')
    #si le voltage noise n'est pas ouvert
    if vnoise_open == False:
        return showinfo ('Warning', 'Voltage Noise is not open, please open it')


    global f_plot1_vnoise, PSD_SSB_plot1_vnoise
    path = filedialog.askdirectory (initialdir = "/",title = "Select directory")
    #choose the name of the txt when we are in the folder
    name = askstring ("Name of the txt", "Name of the txt")
    if name == None:
        return showinfo ("Warning", "You did not enter a name")
    if path != None:
        data1 = sp.transpose([f_plot1_vnoise, PSD_SSB_plot1_vnoise])
        np.savetxt(path + '/' + name + '.txt', data1, delimiter='\t', fmt='%0.6e')
        showinfo ('Info', 'Txt saved')
```

```python
def frequency(V, FSR, Nplot, F1):
    #if i_pico == False:
        #return showinfo ('Warning', 'Acquisition is not started, please start the acquisition')
    global nbfreq, nbRMS, f_plot1, PSD_SSB_plot1, freq_pico

    if V.get () == '' and FSR.get () == '' and Nplot.get () == '' and F1.get () == '':
        return showinfo ('Warning', 'Please enter the voltage, the FSR, the number of points and the frequency')
    if V.get () == '':
        return showinfo ('Warning', 'Please enter the voltage')
    if FSR.get () == '':
        return showinfo ('Warning', 'Please enter the FSR')
    if Nplot.get () == '':
        return showinfo ('Warning', 'Please enter the Nplot')
    if F1.get () == '':
        return showinfo ('Warning', 'Please enter F1')



    if status_pico == False:
        return showinfo ('Warning', 'Picoscope is not open, please open it')

    if i_pico == False:
        return showinfo ('Warning', 'Picoscope is not initialized, please initialize the picoscope')


    if v_pico == False:
        return showinfo ('Warning', 'Start the Acquisition')

    if v_pico == True:
        freq_pico == True
        showinfo ('Info', 'The frequency and the RMS are currently being acquired. This may take some time, please wait...')

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



    tension = int (V.get ())
    FSR = float (FSR.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 = tension * 2 * 3.14 / (
        FSR)  # for the conversion from V to Hz - by adjusting optical power to reach +/- "tension" signal
    dU = 1 / amplificationfactor * V_meas
    dUf = np.absolute (spfft.fft (dU))
    PSD_SSB = dUf[int (N / 2):] ** 2 / (
            N * T)  # Power Spectral Density V^2/Hz This is the SSB (single sideband, one-sided only positive frequencies) as displayed and saved in the R&S FSWP
    f = f[:int (N / 2)]

    tic = time.time ()
    # N_plot = 1000  # for points above F1
    # F1 = 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 = np.append (f[f < F1], f_plot)
    PSD_SSB_plot1 = np.append (PSD_SSB[f < F1], PSD_SSB_plot)

    fig1, ax1 = plt.subplots ()
    silentsys_vert = (122 / 255, 186 / 255, 76 / 255, 1)
    ax1.loglog (f_plot1, PSD_SSB_plot1, color=silentsys_vert)
    #ax1.loglog(f_plot2,np.sqrt(PSD_SSB_plot2)*1e9,color=silentsys_vert)
    ax1.set_xlim (1e0, 1e7)
    #ax4.y.autoscale
    ax1.autoscale (enable=True, axis='y', tight=True)
    #ax1.set_ylim (1e-20, 1e-10)
    ax1.set_ylabel ('Frequency Noise PSD (Hz2/Hz)')
    ax1.set_xlabel ('Fourier Frequency (Hz)')
    fig1.tight_layout ()

    plt.title ('Frequency Noise ' + str(nbfreq))
    img = plt.imread('images/SILENTSYS_VERT_Pantone_10pc.png')
    axlogo = plt.axes ([0.2, 0.4, 0.7, 0.5])

    axlogo.set_axis_off ()
    axlogo.imshow (img, zorder=0)
    plt.show ()
```

Vision globale du code : les fonctions (6)

Regardons le code des fonctions

```
#-------------------------------------RMS---------------------------------------


#Limits of intergration
startfreq = 0.9 #Hz
stopfreq = 1e6 #Hz

#Integrated noise
freqNoise = PSD_SSB_plot1
range1 = sp.where((f_plot1 >= startfreq)*(f_plot1 <= stopfreq))
freq, freqNoise = f_plot1[range1], freqNoise[range1]
RMSVNoise = sp.sqrt(-spi.cumtrapz(2*freqNoise[::-1],freq[::-1],initial=0))[::-1]#the minus is because of the backward integration




fig2, ax2 = plt.subplots()
ax2.set_xlabel('Lower limit frequency (Hz)')
ax2.set_ylabel('RMS Voltage (uV)')

ax2.set_xlim(startfreq, stopfreq)
ax2.autoscale(enable=True, axis='y', tight=True)

RMSVNoise = RMSVNoise*1e6
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))


ax2.legend(frameon=False)
fig2.tight_layout()
img = plt.imread('images/SILENTSYS_VERT_Pantone_10pc.png')
axlogo = plt.axes([0.2, 0.3, 0.7, 0.5])
axlogo.set_axis_off()
axlogo.imshow(img)
plt.show()
```

```
def saveFreq():
    global f_plot1, PSD_SSB_plot1, freq_pico

    if status_pico == False:
            return showinfo ('Warning', 'Picoscope is not open, please open it')
    #si le picoscope n'est pas initialisé
    if i_pico == False:
        return showinfo ('Warning', 'Picoscope is not initialized, please initialize the picoscope')
    #si le frequncy noise n'est pas en cours d'acquisition
    if freq_pico == False:
        return showinfo ('Warning', 'Start the Frequency Noise acquisition before saving')


    path = filedialog.askdirectory (initialdir = "/",title = "Select directory")
    #choose the name of the txt when we are in the folder
    name = askstring ("Name of the txt", "Name of the txt")
    if name == None:
        return showinfo ("Warning", "You did not enter a name")
    if path != None:
        data1 = sp.transpose([f_plot1, PSD_SSB_plot1])
        np.savetxt(path + '/' + name + '.txt', data1, delimiter='\t', fmt='%0.6e')
        showinfo ('Info', 'Txt saved')
```

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.

```python
#---- IMPORTATION (1) -----#
from tkinter import *
from tkinter.ttk import *
import time
from tkinter import N, filedialog, END, Toplevel
from tkinter.messagebox import showinfo
from tkinter.simpledialog import askstring
import importlib


P = importlib.import_module ('PicoScope 5000 Series Data Acquisition Software')

#----GLOBAL VARIABLES-----#
global led, reso, range, sample, filter, noise, status_pico, U_meas, status_init_pico, f_plot1_vnoise, PSD_SSB_plot1_vnoise, U_meas0, v_pico, i_pico, s_pico, vnoise_open, acqui_pico, PSD_SSB_plot1, f_plot1, freq_pico


#-----INITIALIZATION-----#
VnoiseRMS = 0
nbnoise = 0
nbacqui = 0
nbfreq = 0
nbRMS = 0
noise = None
status_pico = False
freq_pico = False
i_pico = False
s_pico = False
save_pico = False
v_pico = False
vnoise_open = False
acqui_pico = False


#-----IMPORTATION (2)-----#
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
import scipy.integrate as spi
import time
import warnings
warnings.filterwarnings ('ignore')  # just use this if warnings are annoying

import ctypes
from picosdk.ps5000a import ps5000a as ps
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 commuiquer avec le PicoScope). La librairie importlib est elle aussi présente.**

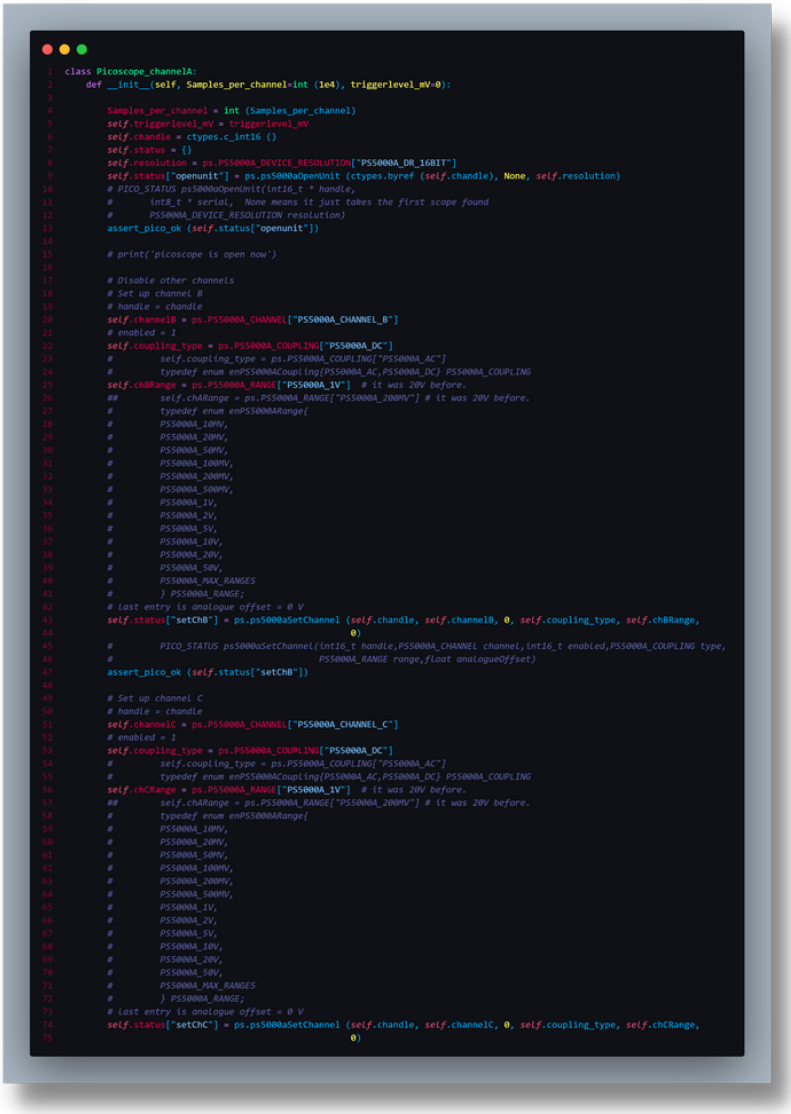
Il y a aussi la partie "global" : global est une instruction qui l'informe que la **variable qui est utilisée à l'intérieur de la fonction est la même que celle qui est définie à l'extérieur de la fonction (dans l'environnement global).**

```
1  global
```

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

```
1  VnoiseRMS = 0
```

LES FONCTIONS (2)



```python
class Picoscope_channelA:
    def __init__(self, Samples_per_channel=int (1e4), triggerlevel_mV=0):

        Samples_per_channel = int (Samples_per_channel)
        self.triggerlevel_mV = triggerlevel_mV
        self.chandle = ctypes.c_int16 ()
        self.status = {}
        self.resolution = ps.PS5000A_DEVICE_RESOLUTION["PS5000A_DR_16BIT"]
        self.status["openunit"] = ps.ps5000aOpenUnit (ctypes.byref (self.chandle), None, self.resolution)
        # PICO_STATUS ps5000aOpenUnit(int16_t * handle,
        #       int8_t * serial,  None means it just takes the first scope found
        #       PS5000A_DEVICE_RESOLUTION resolution)
        assert_pico_ok (self.status["openunit"])

        # print('picoscope is open now')

        # Disable other channels
        # Set up channel B
        # handle = chandle
        self.channelB = ps.PS5000A_CHANNEL["PS5000A_CHANNEL_B"]
        # enabled = 1
        self.coupling_type = ps.PS5000A_COUPLING["PS5000A_DC"]
        #         self.coupling_type = ps.PS5000A_COUPLING["PS5000A_AC"]
        #         typedef enum enPS5000ACoupling{PS5000A_AC,PS5000A_DC} PS5000A_COUPLING
        self.chBRange = ps.PS5000A_RANGE["PS5000A_1V"]  # it was 20V before.
        ##        self.chARange = ps.PS5000A_RANGE["PS5000A_200MV"] # it was 20V before.
        #         typedef enum enPS5000ARange{
        #         PS5000A_10MV,
        #         PS5000A_20MV,
        #         PS5000A_50MV,
        #         PS5000A_100MV,
        #         PS5000A_200MV,
        #         PS5000A_500MV,
        #         PS5000A_1V,
        #         PS5000A_2V,
        #         PS5000A_5V,
        #         PS5000A_10V,
        #         PS5000A_20V,
        #         PS5000A_50V,
        #         PS5000A_MAX_RANGES
        #         } PS5000A_RANGE;
        # last entry is analogue offset = 0 V
        self.status["setChB"] = ps.ps5000aSetChannel (self.chandle, self.channelB, 0, self.coupling_type, self.chBRange,
                                                      0)
        #         PICO_STATUS ps5000aSetChannel(int16_t handle,PS5000A_CHANNEL channel,int16_t enabled,PS5000A_COUPLING type,
        #                                       PS5000A_RANGE range,float analogueOffset)
        assert_pico_ok (self.status["setChB"])

        # Set up channel C
        # handle = chandle
        self.channelC = ps.PS5000A_CHANNEL["PS5000A_CHANNEL_C"]
        # enabled = 1
        self.coupling_type = ps.PS5000A_COUPLING["PS5000A_DC"]
        #         self.coupling_type = ps.PS5000A_COUPLING["PS5000A_AC"]
        #         typedef enum enPS5000ACoupling{PS5000A_AC,PS5000A_DC} PS5000A_COUPLING
        self.chCRange = ps.PS5000A_RANGE["PS5000A_1V"]  # it was 20V before.
        ##        self.chARange = ps.PS5000A_RANGE["PS5000A_200MV"] # it was 20V before.
        #         typedef enum enPS5000ARange{
        #         PS5000A_10MV,
        #         PS5000A_20MV,
        #         PS5000A_50MV,
        #         PS5000A_100MV,
        #         PS5000A_200MV,
        #         PS5000A_500MV,
        #         PS5000A_1V,
        #         PS5000A_2V,
        #         PS5000A_5V,
        #         PS5000A_10V,
        #         PS5000A_20V,
        #         PS5000A_50V,
        #         PS5000A_MAX_RANGES
        #         } PS5000A_RANGE;
        # last entry is analogue offset = 0 V
        self.status["setChC"] = ps.ps5000aSetChannel (self.chandle, self.channelC, 0, self.coupling_type, self.chCRange,
                                                      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)

```python
class ProcessNoise (object):  # TODO live class
    """
    Class used for a adding live update to a figure. Run '%matplotlib notebook' in jupyter notebook
    for optimal operation of this class. This adds

    Attributes
    ----------
    needs explanation"""

    def __init__(self, init_ps=True, Nsamples=int (2 ** 25)):

        #####
        ##Initialize Picoscope
        #####
        self.Nsamples = Nsamples
        if init_ps:
            try:
                self.picoscope = Picoscope_channelA (Samples_per_channel=self.Nsamples)

            except:
                Picoscope_channelA.close ()  # check if self needed here
                self.picoscope = Picoscope_4channels (Samples_per_channel=self.Nsamples)
                print ('picoscope was already open and was now reinitialized')

            self.picoscope.N = self.Nsamples
            self.picoscope.set_timebase (5)  # for 12BIT 2 is 4ns Sampling interval => 250MS/s; 3 is 8ns => 125Ms/s
            # check on page 22 in programmers-guide 3#with 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,
            # 8 is 20 MS/s, 9 is 18MS/s,
            # 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
            self.picoscope.set_BandwidthFilter (1)  # this sets the 20MHz lowpassfilter for all channels
            self.picoscope.R = 1 / (self.picoscope.timeIntervalns.value * 1e-9)  # sampling rate
            self.timeS = sp.linspace (0, (self.picoscope.N - 1) / self.picoscope.R,
                                      self.picoscope.N)  # check if this will be needed for the future
            self.Noise_start_acquisition ()
            self.picoscope.check_ready ()  # this is needed to write the data to the buffer for the readout.
            # here we could save time to evaluate while writing into the buffer at the same time
            time.sleep (1)  # this is to make sure that the AC coupling has time to stabilize
            self.Noise_start_acquisition ()
            self.picoscope.check_ready ()  # this is needed to write the data to the buffer for the readout.
            self.picoscope.readout_buffer ()

            self.time = sp.linspace (0, (self.picoscope.N - 1) / self.picoscope.R, self.picoscope.N)
            self.signal_A = sp.array (self.picoscope.bufferA)

    #             self.N_FFT = Nsamples
    #             self.dt = 1/self.picoscope.R
    #             self.freq = spf.fftfreq(self.N_FFT,self.dt)
    #             self.signal_FFT1 = sp.zeros(self.N_FFT)
    #             self.signal_FFT1_phase = sp.zeros(self.N_FFT)
    #             self.signal_FFT1_amplitude = sp.ones(self.N_FFT)

    def Noise_start_acquisition(self):
        self.picoscope.start_acquiredata_triggered ()  # this writes the data into the buffer

    def update_data(self):
        self.Noise_start_acquisition ()  # this starts the acquisition
        self.picoscope.check_ready ()  # this is needed to write the data to the buffer for the readout.
        # here we could save time to evaluate while writing into the buffer at the same time
        self.picoscope.readout_buffer ()
        self.signal_A = sp.array (
            adc2mV (self.picoscope.bufferA, self.picoscope.chARange, self.picoscope.maxADC)) * 1e-3
        self.signal_Noise1 = self.signal_A

    def save_data(self, fn):
        data = sp.transpose ([self.time, self.signal_Noise1])
        sp.savetxt (fn, data, fmt='%0.6e', header='time (s), signal A (V)', delimiter=', ', newline='\n')

    def save_dataraw(self, fn):
        data = sp.transpose ([self.time, self.signal_Noise1])
        sp.savetxt (fn, data, fmt='%0.6e', header='time (s), signal A (V)', delimiter=', ', newline='\n')

    def load_rawdata(self, fn):  # speed up this function todo
        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.**

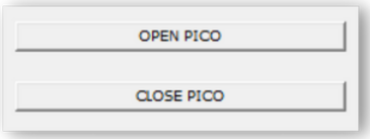
On peut y trouver des fonctions permettant de commencer l'acquisition, de mettre à jour, de sauvegarder..

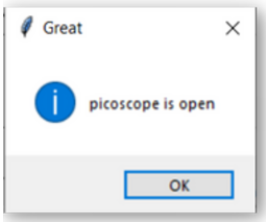
LES FONCTIONS (4)

```python
def open_pico():  # work
    global status_pico, noise
    noise = ProcessNoise (init_ps=True, Nsamples=int (2 ** 25))  # up to 2^25 should be ok for the PS5444D;
    status_pico = True
    if status_pico == True:
        led.config (bg='green')
        return showinfo ('Great','Picoscope is open')


# 2^24 are too many samples for buffering for the PS5442D used for testing.
# execute cell with noise.picoscope.close() below if an error appears

def close_pico():  # work
    global status_pico, s_pico, i_pico, v_pico, vnoise_open, acqui_pico, freq_pico
    noise.picoscope.close ()
    status_pico = False
    if status_pico == False:
        led.config (bg='red')
        return showinfo ('Warning','Picoscope is closed')
    #quand on ferme le picoscope, on réinitialise les popups
    s_pico = False
    i_pico = False
    v_pico = False
    vnoise_open = False
    acqui_pico = False
    freq_pico = False

```

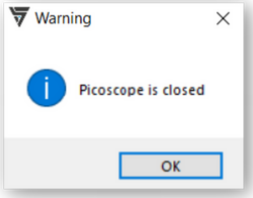
Voici les deux fonctions "open_pico" et "close_pico".

"open_pico" contient le **global** permettant de bien faire marcher les variables, on retrouve aussi ia **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**

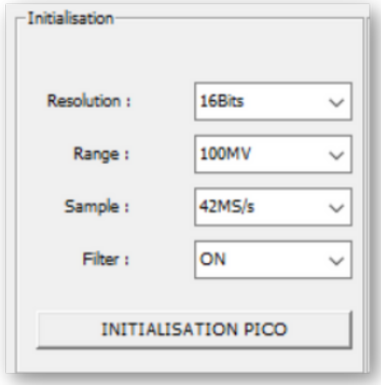
LES FONCTIONS (5)

```python
def init_pico():
    global i_pico

    if status_pico == False:
        return showinfo ('Error', 'Picoscope is not open')

    range_tab = [
        # tableau associatif pour les valeurs de range
        {'name': '10MV', 'value': 0},
        {'name': '20MV', 'value': 1},
        {'name': '50MV', 'value': 2},
        {'name': '100MV', 'value': 3},
        {'name': '200MV', 'value': 4},
        {'name': '500MV', 'value': 5},
        {'name': '1V', 'value': 6},
        {'name': '2V', 'value': 7},
        {'name': '5V', 'value': 8},
        {'name': '10V', 'value': 10},
        {'name': '20V', 'value': 11},
        {'name': '50V', 'value': 12},
    ]
    timebase_tab = [
        # tableau associatif pour les valeurs de timebase

        {'name': '64MS/s', 'value': 4},
        {'name': '42MS/s', 'value': 5},
        {'name': '32MS/s', 'value': 6},
        {'name': '25MS/s', 'value': 7},
        {'name': '20MS/s', 'value': 8},
        {'name': '18MS/s', 'value': 9},
        {'name': '16MS/s', 'value': 10},
        {'name': '14MS/s', 'value': 11},
        {'name': '12.5MS/s', 'value': 12},
        {'name': '11MS/s', 'value': 13},
        {'name': '10MS/s', 'value': 14},

    ]
    filter_tab = [
        # tableau associatif pour les valeurs de filter
        {'name': 'OFF', 'value': 1},
        {'name': 'ON', 'value': 0},
    ]

    resolution_tab = [
        # tableau associatif pour les valeurs de resolution
        {'name': '16Bits', 'value': 16}
    ]

    for r in resolution_tab:
        if r['name'] == reso.get ():
            getreso = r['value']
            print ("reso : " + str (getreso))

    for i in range_tab:
        if i['name'] == range.get ():
            getrange = i['value']
            print ("range : " + str (getrange))

    for j in timebase_tab:
        if j['name'] == sample.get ():
            gettime = j['value']
            print ("time : " + str (gettime))

    for x in filter_tab:
        if x['name'] == filter.get ():
            getfilter = x['value']
            print ("filter : " + str (getfilter))

    # save the values in the picoscope
    noise.picoscope.set_timebase (gettime)
    noise.picoscope.set_range (getrange)
    noise.picoscope.set_resolution (getreso)
    noise.picoscope.set_BandwidthFilter (getfilter)
    i_pico = True
    showinfo ('Info', 'Picoscope is initialized')
```

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 le tableau pour la range, la timebase, le filtre et la résolution.



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 ca 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)

```python
def startaqui():
    global U_meas, s_pico, U_meas0, v_pico, acqui_pico, nbacqui

    nbacqui=nbacqui+1

    if status_pico == False:
        return showinfo ('Warning', 'Picoscope is not open, please open it')
    if i_pico == False:
        return showinfo ('Warning', 'Picoscope is not initialized, please initialize the picoscope')
    if i_pico == True:
        showinfo ('Info', 'Acquisition is started. This can take a while. After that you can save the data and launch the Voltage Noise Acquisition. ')

        v_pico = True

    noise.update_data ()

    U_meas0 = noise.signal_Noise1
    U_meas0.max ()

    U_meas0.min ()

    U_meas = (U_meas0 - U_meas0.mean ())

    fig0 = plt.figure ()
    plt.plot (U_meas0)

    plt.autoscale (enable=True, axis='y', tight=True)
    plt.title ('Time Trace ' + str(nbacqui))
    plt.ylabel ('Tension (V)')

    plt.xlabel ('Npts (s)')

    acqui_pico = True
    plt.show ()
```

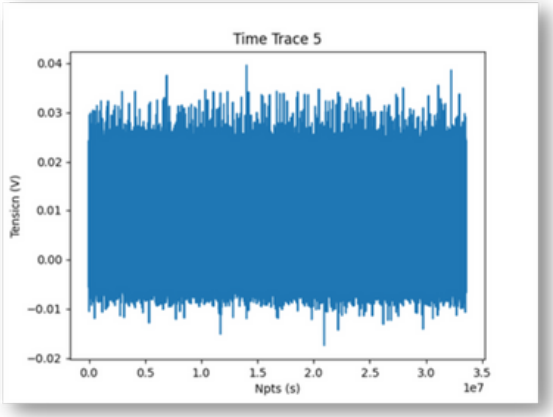
Voici la fonction startacqui (relié au bouton start acquisition) :

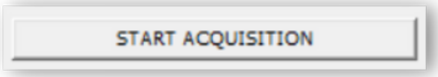
Une **variable nbacqui** est présente, elle a eté initialisée à 0 au début du code. Cette dernière à chaque acquisition se verra incrémenté de 1.

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).**





LES FONCTIONS (7)

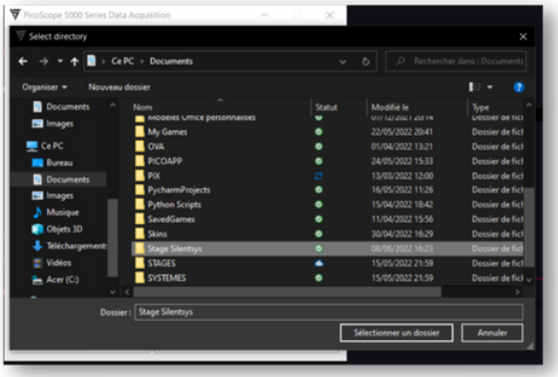
```python
def saveaqui():
    global U_meas, s_pico, U_meas0, v_pico, acqui_pico

    #si l'acquisition n'est pas lancée, on ne peut pas sauvegarder
    if v_pico == False:
        return showinfo ('Warning', 'Acquisition is not started, please start the acquisition')

    #si on ne lance pas l'acquisition, on ne peut pas sauvegarder
    if acqui_pico == False:
        return showinfo ('Warning', 'Acquisition is not started, please start the acquisition before saving datas')



    #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 i_pico == False:
        return showinfo ('Warning', 'Picoscope is not initialized, please initialize the picoscope')



    global U_meas0
    path = filedialog.askdirectory (initialdir = "/",title = "Select directory")

    #choose the name of the txt when we are in the folder
    name = askstring ("Name of the txt", "Name of the txt")

    if name == None:
        return showinfo ("Warning", "You did not enter a name")
    if path != None:
        data2 = sp.transpose([U_meas0])
        np.savetxt(path + '/' + name + '.txt', data2, delimiter='\t', fmt='%0.6e')
        showinfo ('Info', 'Txt saved')
```

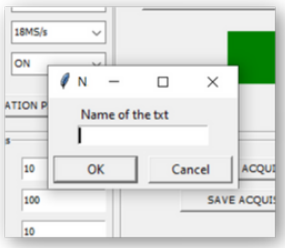
Voici la fonction saveacqui (relié au bouton save acquisition) :

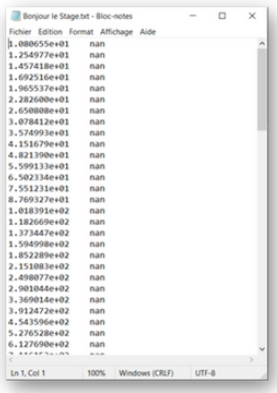
Cette fonction permet de sauvegarder en .txt les données brutes acquises lors de l'acquisition.

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)

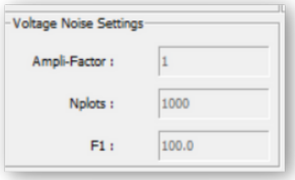
```python
def vnoise(ampfact, Nplot, F1):


    #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 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 ampfact.get() == '' and Nplot.get() == '' and F1.get() == '':
        return showinfo ('Warning', 'You did not enter any values')

    if ampfact.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 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_vnoise, PSD_SSB_plot1_vnoise, vnoise_open, nbnoise, VnoiseRMS


    nbnoise = nbnoise + 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


    ampfact = int (ampfact.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 = ampfact  # for the voltage amplification
    dU = 1 / amplificationfactor * U_meas
    dUf = np.absolute (spfft.fft (dU))
    PSD_SSB = dUf[: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 R&S FSWP
    f = f[:int (N / 2)]

    tic = time.time ()
    # N_plot = 1000  # for points above f1
    # F1 = 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[f < f1], f_plot)
    PSD_SSB_plot1_vnoise = np.append (PSD_SSB[f < f1], PSD_SSB_plot)

    fig1, ax1 = plt.subplots ()
    silentsys_vert = (122 / 255, 186 / 255, 76 / 255, 1)
    silentsys_rouge = (254 / 255, 27/ 255, 0/255, 1)
    ax1.loglog (f_plot1_vnoise, np.sqrt(PSD_SSB_plot1_vnoise)*1e9, color=silentsys_rouge)
    #ax1.loglog(f_plot2,np.sqrt(PSD_SSB_plot2)*1e9,color=silentsys_vert)
    ax1.set_xlim (1e0, 1e7)
    ax1.set_ylim (1e1, 1e6)
    ax1.set_ylabel ('Voltage Noise PSD (nV/√Hz)')
    ax1.set_xlabel ('Fourier Frequency (Hz)')
    img2 = plt.imread('images/SILENTSYS_VERT_Pantone_10pc.png')
    fig1.tight_layout ()



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


    vnoise_open = True
    plt.show ()
```

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

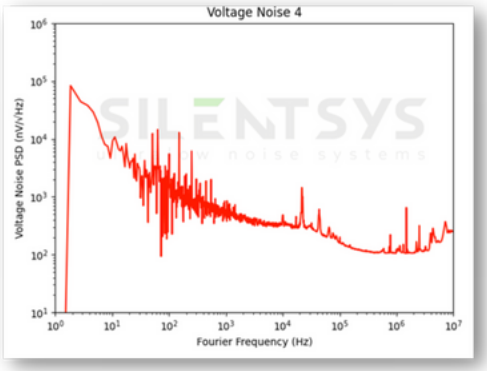
Voici les conditions permettant de guider l'utilisateur.

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

On 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 **fplot_vnoise et le PSD_SSB_plot1_vnoise**



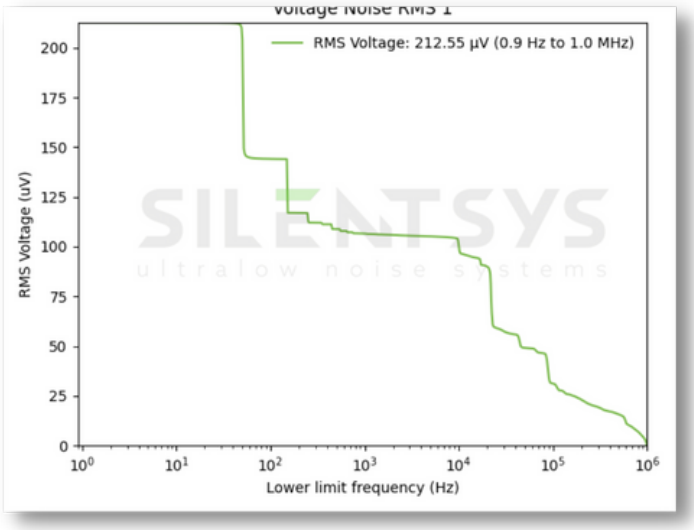
LES FONCTIONS (9)

```python
#-----------------------------------RMS-----------------------------------------


    startfreq_vnoise = 0.9 #Hz
    stopfreq_vnoise = 1e6 #Hz

    #Integrated noise
    freqNoise = PSD_SSB_plot1_vnoise
    range2 = sp.where((f_plot1_vnoise >= startfreq_vnoise)*(f_plot1_vnoise <= stopfreq_vnoise))
    freq, freqNoise = f_plot1_vnoise[range2], freqNoise[range2]
    RMSVNoise = sp.sqrt(-spi.cumtrapz(2*freqNoise[::-1],freq[::-1],initial=0))[::-1]#the minus is because of the backward integration




    fig2, ax2 = plt.subplots()
    ax2.set_xlabel('Lower limit frequency (Hz)')
    ax2.set_ylabel('RMS Voltage (uV)')

    ax2.set_xlim(startfreq_vnoise, stopfreq_vnoise)
    #axe y autoscale
    ax2.autoscale(enable=True, axis='y', tight=True)
    #ax2.set_ylim(0, plt.autoscale)

    RMSVNoise = RMSVNoise*1e6

    ax2.semilogx(freq, RMSVNoise,color=silentsys_vert, label='RMS Voltage: {0:.2f} µV ({1:.1f} Hz to {2:.1f} MHz)'.format(RMSVNoise[0],freq.min(),freq.max()*1e-6))


    ax2.legend(frameon=False)
    fig2.tight_layout()
    plt.title('Voltage Noise RMS ' + str(VnoiseRMS))
    img = plt.imread('images/SILENTSYS_VERT_Pantone_10pc.png')
    axlogo = plt.axes([0.2, 0.3, 0.7, 0.5])
    axlogo.set_axis_off()
    axlogo.imshow(img)
    plt.show()

```

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.**



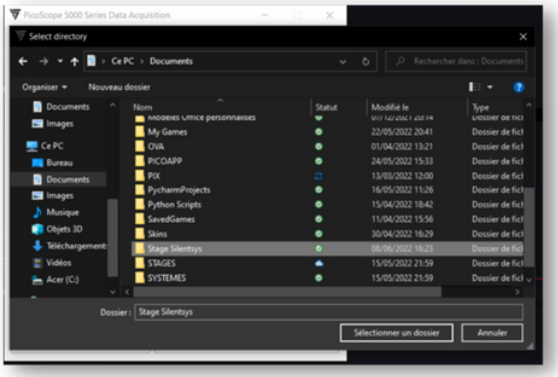
LES FONCTIONS (10)

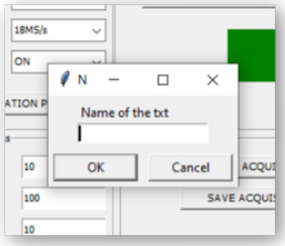
```python
def sauvegardevnoise():


    #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 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', 'Start the Voltage Noise acquisition')
    #si le voltage noise n'est pas ouvert
    if vnoise_open == False:
        return showinfo ('Warning', 'Voltage Noise is not open, please open it')



    global f_plot1_vnoise, PSD_SSB_plot1_vnoise
    path = filedialog.askdirectory (initialdir = "/",title = "Select directory")
    #choose the name of the txt when we are in the folder
    name = askstring ("Name of the txt", "Name of the txt")
    if name == None:
        return showinfo ("Warning", "You did not enter a name")
    if path != None:
        data1 = sp.transpose([f_plot1_vnoise, PSD_SSB_plot1_vnoise])
        np.savetxt(path + '/' + name + '.txt', data1, delimiter='\t', fmt='%0.6e')
        showinfo ('Info', 'Txt saved')
```

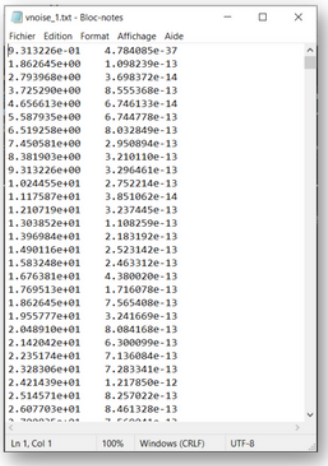
Voici la fonction sauvegardevnoise (relié à son bouton save analysis) :

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

Les seuls choses qui changent sont les valeurs enregistrées (f_plot1_vnoise et PSD_SSB_plot1_vnoise)







LES FONCTIONS (11)

```python
def frequency(V, FSR, Nplot, F1):
    #if s_pico == False:
        #return showinfo ('Warning', 'Acquisition is not started, please start the acquisition')
    global nbfreq, nbRMS, f_plot1, PSD_SSB_plot1, freq_pico

    if V.get () == ''and FSR.get () == ''and Nplot.get () == ''and F1.get () == '':
        return showinfo ('Warning', 'Please enter the voltage, the FSR, the number of points and the frequency')
    if V.get () == '':
        return showinfo ('Warning', 'Please enter the voltage')
    if FSR.get () == '':
        return showinfo ('Warning', 'Please enter the FSR')
    if Nplot.get () == '':
        return showinfo ('Warning', 'Please enter the Nplot')
    if F1.get () == '':
        return showinfo ('Warning', 'Please enter F1')



    if status_pico == False:
         return showinfo ('Warning', 'Picoscope is not open, please open it')

    if i_pico == False:
        return showinfo ('Warning', 'Picoscope is not initialized, please initialize the picoscope')


    if v_pico == False:
        return showinfo ('Warning', 'Start the Acquisition')

    if v_pico == True:
        freq_pico == True
        showinfo ('Info', 'The frequency and the RMS are currently being acquired. This may take some time, please wait...')

    nbfreq = nbfreq + 1
    nbRMS = nbRMS + 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



    tension = int (V.get ())
    FSR = float (FSR.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 = tension * 2 * 3.14 / (
        FSR)  # for the conversion from V to Hz - by adjusting optical power to reach +/- "tension" signal
    dU = 1 / amplificationfactor * U_meas
    dUf = np.absolute (spfft.fft (dU))
    PSD_SSB = dUf[: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 R&S FSWP
    f = f[:int (N / 2)]

    tic = time.time ()
    # N_plot = 1000  # for points above f1
    # f1 = 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 = np.append ([f[f < f1], f_plot)
    PSD_SSB_plot1 = np.append (PSD_SSB[f < f1], PSD_SSB_plot)

    fig1, ax1 = plt.subplots ()
    silentsys_vert = (122 / 255, 186 / 255, 76 / 255, 1)
    ax1.loglog (f_plot1, PSD_SSB_plot1, color=silentsys_vert)
    #ax1.loglog(f_plot2,np.sqrt(PSD_SSB_plot2)*1e9,color=silentsys_vert)
    ax1.set_xlim (1e0, 1e7)
    #axe y autoscale
    ax1.autoscale (enable=True, axis='y', tight=True)
    #ax1.set_ylim (1e-20, 1e20)
    ax1.set_ylabel ('Frequency Noise PSD (Hz2/Hz)')
    ax1.set_xlabel ('Fourier Frequency (Hz)')
    fig1.tight_layout ()

    plt.title ('Frequency Noise ' + str(nbfreq))
    img = plt.imread('images/SILENTSYS_VERT_Pantone_10pc.png')
    axlogo = plt.axes ([0.2, 0.4, 0.7, 0.5])

    axlogo.set_axis_off ()
    axlogo.imshow (img, zorder=0)
    plt.show ()
```

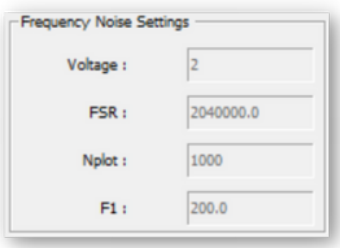
Voici la fonction frequency : Elle permet de faire l'acquisiiton du Frequency Noise (reliée au bouton (Voltage Noise Analysis) cette fonction prend en arguments la tension (V), le FSR, le Nplot et le f1.

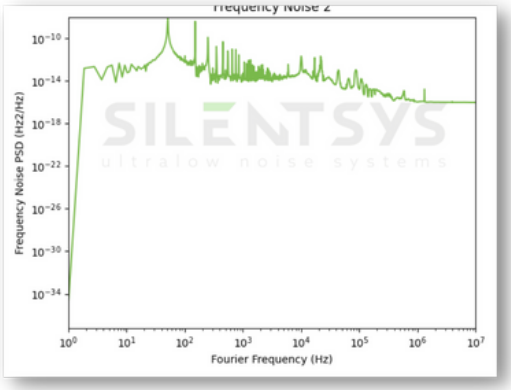
Voici les conditions permettant de guider l'utilisateur.

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**





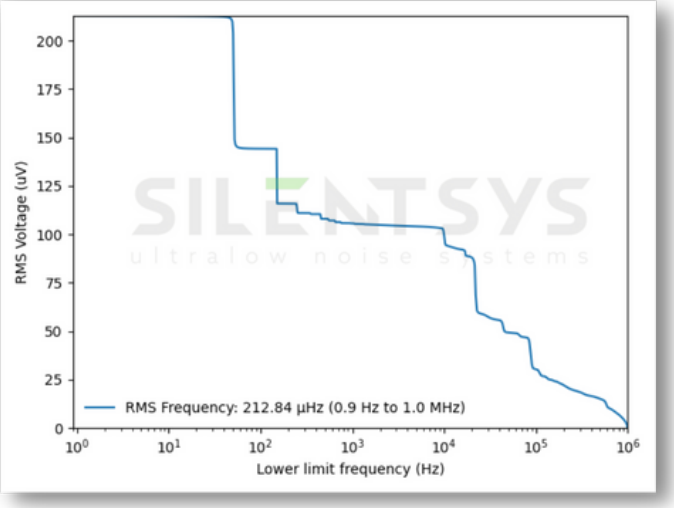
LES FONCTIONS (12)

```python
#-------------------------------------RMS-----------------------------------------


    #Limits of intergration
    startfreq = 0.9 #Hz
    stopfreq = 1e6 #Hz

    #Integrated noise
    freqNoise = PSD_SSB_plot1
    range1 = sp.where((f_plot1 >= startfreq)*(f_plot1 <= stopfreq))
    freq, freqNoise = f_plot1[range1], freqNoise[range1]
    RMSVNoise = sp.sqrt(-spi.cumtrapz(2*freqNoise[::-1],freq[::-1],initial=0))[::-1]#the minus is because of the backward integration




    fig2, ax2 = plt.subplots()
    ax2.set_xlabel('Lower limit frequency (Hz)')
    ax2.set_ylabel('RMS Voltage (uV)')

    ax2.set_xlim(startfreq, stopfreq)
    ax2.autoscale(enable=True, axis='y', tight=True)

    RMSVNoise = RMSVNoise*1e6
    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))


    ax2.legend(frameon=False)
    fig2.tight_layout()
    img = plt.imread('images/SILENTSYS_VERT_Pantone_10pc.png')
    axlogo = plt.axes([0.2, 0.3, 0.7, 0.5])
    axlogo.set_axis_off()
    axlogo.imshow(img)
    plt.show()

```

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)

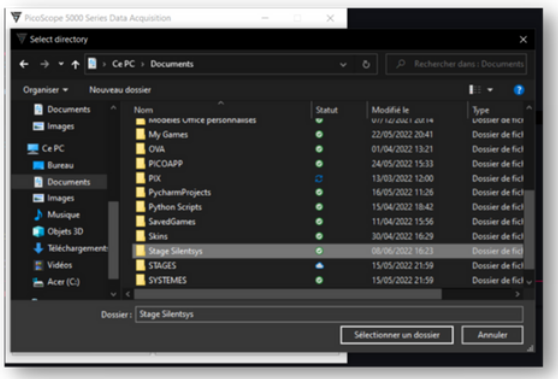
```python
def saveFreq():
    global f_plot1, PSD_SSB_plot1, freq_pico

    if status_pico == False:
            return showinfo ('Warning', 'Picoscope is not open, please open it')
    #si le picoscope n'est pas initialisé
    if i_pico == False:
        return showinfo ('Warning', 'Picoscope is not initialized, please initialize the picoscope')
    #si le frequncy noise n'est pas en cours d'acquisition
    if freq_pico == False:
        return showinfo ('Warning', 'Start the Frequency Noise acquisition before saving')


    path = filedialog.askdirectory (initialdir = "/",title = "Select directory")
    #choose the name of the txt when we are in the folder
    name = askstring ("Name of the txt", "Name of the txt")
    if name == None:
        return showinfo ("Warning", "You did not enter a name")
    if path != None:
        data1 = sp.transpose([f_plot1, PSD_SSB_plot1])
        np.savetxt(path + '/' + name + '.txt', data1, delimiter='\t', fmt='%0.6e')
        showinfo ('Info', 'Txt saved')

```

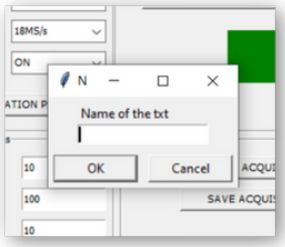
Voici la fonction saveFreq (reliée au bouton save analysis du Frequency Noise) :

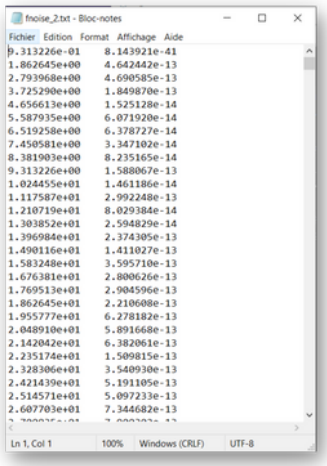
Cette fonction permet de sauvegarder en .txt les données brutes acquises lors de l'acquisition.

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 enregisté.







NOTES

NOTES

SILENTSYS

ultralow noise systems