GUlogo.pdf

Project Report

MSc Semester 3

Unit 505: Project Work

Ved Rudani Roll number: **54**
Vijay Panchal Roll number: **55**

GUlogo.pdf

Project Report

Comprehensive analysis of electronic noise and their noise spectra of zener diode

MSc Semester 3

Unit 505: Project Work

Ved Rudani Roll number: **54**
Vijay Panchal Roll number: **55**

Abstract

This is our semester project in which we studied noise from source *zener diode* specifically BZX55C5V1. We looked for low frequency flicker noise, johnson noise and shot noise traces. In this project we used a lock-in amplifier specifically SR830 which is relatively accurate and low noise. We used the relative capability of SR830 that of minimum detection around 10nV/Hz signal. We made open source community with making python library *pyinstro* for controlling and interfacing instruments like SR830. This program is made to be very flexible and highly extensible for use in every SCPI supported interface like GPIB, RS232, USB, LAN with capabilities like built-in file writer (.csv) and some CLI argument parsing.

Contents

1. Introduction

Regulated power sources are extremely important in day to day lab work. Zener diodes and passive elements share an integral part of the overall circuit of regulated power supply. When we are dealing with precision measurement and study we need the most precise power sources to work with but because of 'Noise' of components of zener diodes and passive elements it inherits noise internally. Since, this noise will be infested in precision work we are doing in the lab. It is better to study the known structure of noise in these devices to address methodic treatment to our data and circuits. With all this in mind we are doing noise measurement and studying the noise spectrum of the zener diode.

For this semester we had radical plans to try but it evolved into more mature or downgraded in a way. First tried as shot noise to generate a random number generator which could possibly be a true random generator with little transformations. Then we eyed the more on general idea of studying noise theoretically and doing analysis experimentally. Which is exactly what we are doing right now but change is that at start we are working with photodiode and now with zener diode. Thanks to Dr. U S Joshi sir who guided us to try different diodes against photodiode. In this report we are having the following parts in order. First we are studying theoretically components, then we will discuss methods and tools that we used included all instrumentation, data acquisition, data analysis etc., we will conclude with our results and discuss it. we took help from Electronic noise and interfering signals: principles and applications.[**?**] The foundational work in thermal noise was done by J B johnson in his paper johnson1928thermal. Johanson also gave first experimental evidence of frequency dependent noise.[**?**]

2. Theoretical compilation

This section will deal with theoretical components from our project. Here, linear circuit analysis gives noise and output voltage relationship.

2.1. Linear circuit analysis

We have a voltage regulator circuit from a zener diode which regulates voltages at specific voltage known as zener voltage V_z. The fluctuation from these regulated voltages is what we call noise. Since ideal regulators only give pure DC voltages at output, this fluctuation is completely unwanted and only be resultant of intrinsic noise of this regulator circuit. We limit ourselves with only noise coming from zener diode which is not quite good practice. Since, noise can be added from extra resistors, wires and even the power supply itself. The resistor noise can be neglected because of their low values as we used 10k in series and 100k in parallel to output. We will see this later.

Let's take a basic voltage regulator circuit as shown in figure 1.

`./images//circuit-20231005-2243.png`

Figure 1: Simple voltage regulator circuit made from zener diode

As you can see we have a zener diode parallel to the power supply, which regulates at a certain degree. Since this is a linear circuit output voltage can be easily derived.

Applying kirchhoff current low in the figure 1,

$$\begin{aligned}I_z &= I_{R_s} - I_L \\ &= \frac{V_s - V_o}{R_s} - \frac{V_o}{R_L} \\ &= -V_o(\frac{1}{R_s} + \frac{1}{R_L}) + \frac{V_s}{R_s} \\ &= -V_o A' + B'\end{aligned}$$

Here, $A' = (\frac{1}{R_s} + \frac{1}{R_L})$ and $B' = \frac{V_s}{R_s}$.

We can write $I_z = \frac{V_z}{R_z}$, where V_z and R_z are respectively zener voltages and impedance. This relation is quite linear in the breakdown region as you can see in the figure 2. [1]

./images//thzenerIV.png

Figure 2: theoretical current and voltage relation for zener diode

Here we can assume equivalent circuit of **??** as figure **??**

Further, simplifying the circuit,

This circuit is further simplified as we take $V_z = V_{DC} + V_n$ where V_n is the noise voltage of the zener diode. If we neglect noise from other sources like resistors and power supply then from figure 4,

$$\frac{V_z}{I_z} = -V_o A' + B'$$

$$\frac{V_{DC} + V_n}{I_z} = -V_o A' + B'$$

$$V_n = -V_o A + B$$

[1]Image is taken from Electronics Devices and Circuit Theory by Robert Boylsted

./images//circuit-20231005-2244.png

Figure 3: Equivalent circuit of figure 1

./images//circuit-20231005-2245.png

Figure 4: Equivalent circuit of figure 3

$$V_o = -\frac{V_n}{A} + \frac{B}{A} \tag{1}$$

So, we can conclude that here as $V_o V_n$. This will be the main focus of this project. Here we are neglecting V_{DC} and will be totally okay when we read data from the LOCK IN amplifier, since the DC component has zero frequency which can't be read from the LOCK IN amplifier.

2.2. Different noises in the circuit

The noise voltage V_n is made from different types of noise source which can act as a symbol voltage source. So, V_n can be broken into sub noise sources such as $V_n = V_{flicker} + V_{thermal} +$

$V_{shot} + \cdots$. We will see this noise source and its origin then we will derive its respective distribution and equations.

noiseeq.png

Figure 5: Equivalent noise sources

2.2.1. Flicker Noise

Flicker noise is also known as 1/f noise in view of the fact that power density decreases with increasing frequency. This implies that at lower frequencies, the flicker noise dominates. This type of noise is found almost in any electronic device which is able to operate at lower frequencies.The main source of this type of noise is D.C supply. Its first evidence was given by J. B. Johnson [**?**]. Its first 1/f form is derived by beck and spruit [**?**]. Now the form is given as

$$S(f) = \frac{\gamma}{f^{\alpha}} \tag{2}$$

Here, γ and α determine the nature of flicker noise. α determine relations with other noise elements.

1. ($\alpha > 0$): This means that white noise is dominating the flicker noise as frequency increases.
2. ($\alpha = 0$): This means that only white noise is exists
3. ($\alpha < 0$): This means noise is increasing as frequency. Also, shows that noise will be persistent with a higher range of frequencies. Typically white noise dominates traditional flicker noise.

We can see noise levels as from figure **??**.

2.2.2. Shot noise

Shot noise is a form of noise that arises because of the discrete nature of the charges carried by charge carriers, electrons or holes or photons hitting the surface. Shot noise is analogous to the rainfall in which raindrop hitting the surface can be considered as discrete. The sound of rainfall is very similar to noise we hear from speakers when we are considering shot noise. Foundational studies in shot noise done my campbell.citecampbell1909study

Since, shot noise is a phenomenon for discrete charge passing through a junction, it can be modelled by poisson distribution. Suppose that In the time interval the τ Q charge passes through a junction in a semiconductor device (in present context zener diode). This gives rise to discrete probability distribution,

$$P(N) = \frac{e^{-\lambda\tau}(\lambda\tau)^N}{N!} \tag{3}$$

If $N = 0$ charge passes in time interval τ then $P(N)$will be,

$$P(0) = e^{-\lambda\tau} \tag{4}$$

Now suppose, probability of one and only one charge passing through junction in time τ,

$$P(\tau)d\tau = (P_\tau(0))(P_\tau(1))$$

From equation **??**,

$$P(\tau)d\tau = (e^{-I_0\tau})(e^{-I_0 d\tau} I_0 d\tau)$$

$$P(\tau) = (e^{-I_0(\tau+d\tau)})I_0$$

We can write this equation in frequency domain and by,

$$P(f) = Sdf \tag{5}$$

Where S is the spectral density of noise.

Here we can write specific form for shot noise in equation **??** [**?**].

$$\langle V_{shot}^2 \rangle = 2eI_0 df \tag{6}$$

Here, e is electron charge,
I_0 is average current,
df is ENBW = Equivalent Noise Bandwidth

$$S(f) = 2eI_0 \tag{7}$$

./images//thnoise.png

Figure 6: Equivalent noise sources

This spectral density gives independence to frequency, which is called white noise.

2.2.3. Avalanche or zener noise

avalanche noise often considers the device's operating characteristics in the avalanche breakdown region. It is a major problem where the device is working in avalanche breakdown regions. It is multiplicative noise where chains of electrons crossing from junction rise to noise behaviour. It is very similar to shot noise and we can use that model and just use a multiplicative element in it. In our circuit this is significant since we are dealing with zener diode. With potential gradient inside the zener diode, if any hole and electron pair generates, it gets dragged by potential and hits the other lattice. This creates chain reaction and very high amplitude noise measured.

$$\langle V_{avalanche}^2 \rangle = M \langle V_{shot}^2 \rangle$$

$$\langle V_{avalanche}^2 \rangle = 2eMI_0 df \tag{8}$$

So, the spectral density $S(f)$ of this noise will be nearly white.

Here, we can combine this both avalanche and shot noise to make one noise source,

$$\langle V_s^2 \rangle = \langle V_{shot}^2 \rangle + \langle V_{avalanche}^2 \rangle$$

$$= 2eI_0 df + 2eMI_0 df$$

$$= (M+1)2eI_0 df$$

And spectral density will be $S(f) = (M + 1)2eI_0$

Since this noise is white noise we can measure at every frequency. This is what we are going to do in the next chapter.

2.2.4. Thermal noise

Thermal noise, also called Johnson–Nyquist noise is the electronic noise generated by random motion of charge carriers. This charge carrier is generated by the thermal agitation inside an electrical conductor at equilibrium, which happens regardless of any applied voltage. Because of their random motion it can be said that they have a mean value at zero. This reason says that we can't model this noise by poisson distribution but have to model with normal or gaussian distribution. In 1936, J B Johnson first gave an idea about thermal noise in thermionic valves. [1]

The noise amplitude is very similar to that of shot noise and given as,

$$\langle V_{thermal}^2 \rangle = 4K_B R df \tag{9}$$

Here, K_B is boltzmann constant,
R is resistance of device or component,
df is ENBW.

$$S(f) = 4K_B R \tag{10}$$

By equation 10 we can see that thermal noise in an ideal resistor is approximately white, meaning that the power spectral density is nearly constant throughout the frequency spectrum. But practically it does decay to zero at extremely high frequencies (terahertz for room temperature). Also, we are neglecting quantum effects.

Total noise in the circuit will bre frequency dependent noise and white noise,

$$S(f) \approx \frac{\gamma}{f^{\alpha}} + (2e(M+1)I_0 + 4K_bR) \qquad (11)$$

which is main derivation of our project.

3. Methodology

3.1. Our voltage regulation circuit

Our purpose was to regulate voltages and also study noise related to the circuit. If we choose a complicated circuit for voltage voltage regulation then analysis of noise will be relatively complicated. So, we used a very basic voltage regulator circuit from a zener diode. Supply was given as DC power supply with voltage V_s. This voltage is decided by the zener voltage at hand.

The noise in the circuit will be relatively higher at the zener breakdown region. As we discussed from the theoretical part, noise power will be proportional to current flowing in the zener diode (here, we are assuming that noise from other parts is almost zero). To prepare a zener diode (BZX55C5V1) to break down the region we choose 5.4V. This is calculated from For our purpose we utilised a general purpose zener diode with breakdown region between 4.8V to 5.4V with current of μA order. We first did the Current and voltage characteristics of zener diodes. The useful information we got from there is source voltages, zener voltages and current that we particularly needed in our project. Our aim was to never exceed the LOCK IN amplifier's input limits. Current and voltage characteristics are down in figure 8. The zener diode we used had its datasheets, which you can see from Appendix. Its power rating is ...

./images//zener.png

Figure 7: Our zener diode

The zener diode was given proper voltages to work in reverse bias, specifically in the breakdown region. The overall circuit was identical to that of voltage regulator by zener diode. We gave particularly 5.0 V, 5.5V in two different runs from the powersource. The Zener diode regulated around 4.9 V.

Now, what we need is that fluctuation over the regulated DC voltage. These fluctuations have to be some function in the frequency domain as we assumed. This function must be made of different harmonics of sinusoidal waves with different phases and frequencies as thought by Fourier and his analysis. So basically we needed a system to measure different amplitudes of these harmonics at different frequencies to model our fluctuations. We needed a complete frequency spectrum at the particular bandwidth we are looking for in this analysis. The LOCK IN amplifier gives exactly that.

3.2. Measuring instrument: LOCK IN amplifier

LOCK IN amplifiers came in the 1930s and became very important in signal extraction from given frequency and phase. It is very helpful in

./images//zenerIV.png

Figure 8: current and voltage characterists of zener diode

measuring signals in a very noisy environment. It takes two inputs, one which is being measured and one which is given as a reference mono frequency signal. Reference signal gets multiplied with input signal and gives output through a process called Phase sensitive detection in which it uses homodyne detection scheme and filters out signal as DC component. We will see in a bit.

3.2.1. Phase sensitive detection

In nutshell it uses frequency multiplication and generates double side bands which then pass through a low pass filter to extract signal. In figure **??** you can see a signal first goes into a low noise differential amplifier which strengthens the signal. Signal Gets multiplied by another reference signal. This gives rise to two bands which pass through a low pass filter which cancels higher degree signal and only left is low frequency signal.

./images//PSD.png

Figure 9: basic phase sensitive detector

If we take signal $V_s(t)$ with frequency w_s, amplitude A and phase θ.

$$V_s(t) = A\cos(w_s t + \theta)$$

$$= \frac{A}{2}(e^{i(w_s t+\theta)} + e^{-i(w_s t+\theta)})$$

Reference signal can be taken as following,

$$V_r(t) = B(e^{-i(w_r t+\phi)})$$

In common settings, $\phi = 0$ and $B = 1$,

$$V_r(t) = e^{i(-w_r t)}$$

Together after mixing the signals we have,

$$Z(t) = V_s(t)_r(t)$$

$$= \frac{A}{2}(e^{i[(w_s-w_r)t+\theta]} + e^{-i[(w_s+w_r)t+\theta]})$$

$$= X(t) + Y(t)$$

Making $w_s = w_r$ which makes subtraction vanishes and only one term with higher frequency lefts. Passing this signal through a low pass filter with very low cutoff gives only DC components and rejects noise even from neighbouring frequencies.

$$Z(t) = \frac{A}{2}(e^{i\theta})$$

Two component $X(t)$ and $Y(t)$ becomes,

$$X(t) = (Z(t))$$

$$= \frac{A}{2}\cos(\theta)$$

And,

$$Y(t) = (Z(t))$$

$$= \frac{A}{2}\sin(\theta)$$

So, Amplitude and Phase becomes,

$$R = \sqrt{X(t)^2 + Y(t)^2}$$

$$= \sqrt{(\frac{A}{2}\cos(\theta))^2 + (\frac{A}{2}\sin(\theta))^2}$$

$$= \frac{A}{2}$$

$$\Theta = \arctan(\frac{Y}{X})$$

So, the final product in PSD is the absolute amplitude of the signal and its phase.

3.2.2. Time Constant

3.2.3. ENBW

3.2.4. Sensitivity

3.2.5. LOCK IN amplifier over traditional measuring device/system

For noise analysis LOCK IN amplifiers are the optimal choice. Traditional approaches deal with the first measurement of a small signal in the time domain. This signal gets amplified with additional noise from the amplifier. Also, amplifiers attenuates signals with its limited bandwidth which is a measure of concern for certain use case scenarios. This attenuated signal gets into some detector. For signal analysis, this signal must go into other processes like analog to digital conversion then Fourier transformation. This whole process gives too much concerned noise which is not related to devices being analysed in our case the voltage regulator circuit. Alternative approach is to go with a LOCK IN amplifier.

Which cancels out most burdens of traditional measurement steps. This whole combined help in reducing internal noise and increasing S/N ratio.

Pros of LOCK IN amplifier:

- LOCK In amplifiers reduces attenuation of signal with increasing frequency since it does not measure signal in the whole frequency spectrum.
- Increase S/N ratio over traditional amplifier circuit
- Gives direct data into frequency domain

Cons of LOCK IN amplifier:

- Relatively expensive
- Does not give information in time domain
- Relatively slow for whole analysis of frequency domain (low but accurate resolution of frequency domain)

3.3. SR830

We used a LOCK IN amplifier from Stanford Research Systems. It is used to detect low amplitude signals as low as $10\frac{nV}{Hz}$ and frequency as low as $100mV$. and measure very small AC signals - upto few nanovolts. Accurate measurements may be made even when the small signal is obscured by noise sources many thousands of times larger.

Internal block diagram of SR830,

3.3.1. Inputs

3.3.2. Outputs

3.3.3. Interfacing

Lock-in amplifiers use a technique known as phase-sensitive detection to single out the component of the signal at a specific reference frequency and phase. Noise signals at frequencies other than the reference frequency are rejected and do not affect the measurements.

Now , on the basis of frequencies/ frequency levels and its origin, there are different types of noises are present in our environment. Some of them are discussed below:

4. Results and Analysis

We have surveyed voltage regulated circuits we made with zener diodes and found some satisfactory results. We take different results for different bandwidths. For context this is row data from different bandwidth. Here, for each set of frequencies in bandwidth we took almost 50 to 100 readings.

4.1. Low frequency: up to 10k hertz

We are mainly focused low frequency results since we are only intrested in regulated power supply applications. We had original assumption that in low frequency flicker noise is higly dominating.

`./images//raw_1000_100us.png`

Figure 10: This is row data for 1k to 10k frequency band TIME CONSTANT = $100\mu s$ and $12dB/oct$

Here initial results were quite random, which means we have to filter our data a bit. For this reason we used a basic filtering method. In this method the data are sorted out as minimum deviation from their minimum then we took the upper 5 to 10 results and took the mean of it. The basic implementation is as following,

Let, $X(f)$ as data point for specific f and $Y(f)$

be sorted data with n number of results,

$$\langle X(f)\rangle = \sum_{i=0}^{N-1} X^{(i)}(f)$$
$$Y_n(f) = min(t\; such\; that\; \#\{s = |X(f) - \langle X(f)\rangle|\;|s \geq t\} = n)$$
$$\langle Y(f)\rangle = \sum_{i=0}^{n} Y^{(i)}(f)$$

This is how we implemented it with python. If you wanna checkout whole code then it is in the appendix.

```
for datapoint in data:
    count+=1
    if datapoint[0]==index:
        temp_erray.append(float(datapoint[1]))
    else:
        nlist = array(temp_erray,dtype=float)
        m = mean(nlist)
        sorted_deviation = argsort(abs(nlist - m))
        filtered_nlist = nlist[sorted_deviation[:points]]
        indexonelist = array([index]*points)
        final_list= column_stack((indexonelist,filtered_nlist))

```

Also, as discussed from the previous section we have to correct these terms with ENBW. Time constant (T)= 100 and roll-off =12 dB/oct

$$ENBW = \frac{1}{8T} = 1250Hz$$

Here we can see some traces of flicker noise. Parameterized as follows,

$$S(f) = \frac{\gamma}{f^{\alpha}}$$

Here, γ and α determine the nature of flicker noise. Results are from theoritical section.

./images//final_1000_100us.png

Figure 11: final analysed data for 1k to 10k hertz frequency with TIME CONSTANT = $100\mu s$ and $12dB/oct$

$$S_{flicker}(f) = 5.90923491\frac{1}{f^{1.34677368}} \tag{12}$$
$$S_{white}(f) = 2.0644381e - 05 \tag{13}$$

4.2. Very low frequency: upto 1 hertz

Let's take a second data set where we have sub hertz frequency data. This data set have each frequency with corresponding 50 values. Raw data looks like figure 12.

For analysing data ENBW is calculated for TIME CONSTANT = $100ms$ and roll of factor $12dB/oct$,

$$ENBW = \frac{1}{8T} = 1.25Hz$$

Final data will be look like this,

Here data feeted at negative $-alpha$,

$$S_{flicker}(f) = \frac{-0.64836618}{f^{-0.64836618}} \tag{14}$$

Here two things is important $\alpha < 0$ and γ is small.

./images//raw_1_100us.png

Figure 12: This is row data for sub one hertz band with TIME CONSTANT = $100ms$ and roll of factor $12dB/oct$

./images//final_1_100ms.png

Figure 13: final analysed data for sub one hertz band with TIME CONSTANT = $100ms$ and roll of factor $12dB/oct$

./images//raw10000100ms.png

Figure 14: This is row data for high frequency band with TIME CONSTANT = $100ms$ and roll of factor $12dB/oct$

./images//final_10000_100ms.png

Figure 15: final analysed data for high frequency band with TIME CONSTANT = $100ms$ and roll of factor $12dB/oct$

4.3. High frequency data: up to 100k hertz

Same as we discussed previously raw data is given here, we have up to 50khz frequency domaoin data,

same as previous sections, we sorted data and take ENBW calculations at here.

the noise spectral density is as following.

$$S_{flicker}(f) = \frac{0.13707877}{f^{0.92287006}} \tag{15}$$

$$S_{white}(f) = 1.20525914e - 05 \tag{16}$$

Here this parameters are similar to that of first data set aka low frequency data.

5. Conclutions

In 1k to 10k data, we have specified flicker noise power AKA $(\frac{1}{f})^{\alpha}$ as $\alpha = 1.34677368$, which is close to flicker noise theoretical values for maximum flicker noise which is equal to 1. white noise level in μV region, which is expected (at $2.0644381e-05$).

The most mysterious results came from second data sets, very low frequency data sets. Here $\alpha = -0.64836618$ which is negative. This means that noise increases as frequency increases, this case already discussed in theoritical part. Our assumtion here is that as frequency increases the white noise gets saturated to it's related power. This means that white noise is not quite white as it seems. Assumption is also be made that frequency dependence of flicker noise is not quite like that of distrubution as theory discussed. Well, we have to dig deep into that.

Last data set is quite normal, it satisfy first data sets and closely correlate with the theoritical model. We have white noise level is similar to that of first data sets with different is of $2.0644381e-05-1.20525914e-05 = 0.85917896e-05$. which is very small.

Further studies can be conducted to anamolous behaviour at very low frequency noise $\approx 1Hz$. This can be important in regulated power supply usage since 1Hz is very near to DC level.

We hope that our contribution to Open source community is well recevied and gets more contribution to our package **PyInstro**.

References

[1] John Bertrand Johnson. Thermal agitation of electricity in conductors. *Physical review*, 32(1):97, 1928.

Appendix

PyInstro

PyInstro is a package we made to communicate, control and data logging to any scientific instrument easily. The main work of it is giving utility for data logging and ease SCPI. Also, it does streamline instruments after extending it. It is just a cover for the PYVISA backend but it gives instrument specific tools. whole package in the following link

This is code for just the GPIB connection which we used. (it is also extensible to the RS232, LAN and USB)

GPIB.py

```
import pyvisa
import sys
from termcolor import cprint

class GPIB:
    def __init__(self) -> None:
        try:                        # GPIB connection check
            cprint("-----------checking GPIB connections--------",color="yellow")
            resources = pyvisa.ResourceManager()
            interface = None
            resourceslist = resources.list_resources()
            cprint(resourceslist,'blue',attrs=['bold'])
            if resourceslist==():
                cprint("ERROR: please check GPIB connection", "red")
                sys.exit()
            else:
                while True:
                    try:
                        choise = int(input("please, choose your device from this list: "))-
                        if choise>len(resourceslist):
                            TypeError
                        interface = resourceslist[choise]
                        cprint("-------------chose resource-----------------",color="green'
                        cprint("-------following device is connected--------",color="green'
                        cprint(interface)
                        break
                    except:
                        cprint("choose with interger and from following...","red")

            self.interface =  resources.open_resource(interface)
        except:
            cprint("ERROR in detecting GPIB, there must be problem with setup of pyvisa or
            sys.exit()

    def ping(self)-> None:
        self.interface.write("*IDN?\n")

    def read(self)-> None:
        self.interface.read()

    def reset(self)-> None:
        self.interface.write("*RST\n")

    def clear_status(self)-> None:
        self.interface.write("*CLS\n")

    def close(self)->None:
        self.interface.close()

    def std_event(self)->None:
        pass
```

This is SR830 device commands,
SR830.py

```
from pyinstro.utils import sysarg
from pyinstro.utils import datafile

new_instance = sysarg.CLI()

if new_instance.get_connection()=="GPIB":
    from pyinstro.interfaces import gpib

    class SR830(gpib.GPIB):
        def __init__(self) -> None:
            super().__init__()

            file_init = datafile.Get_File(new_instance.get_file())

            self.get_levels = new_instance.get_levels
            self.get_partitions = new_instance.get_partitions
            self.writerow = file_init.writerow
            self.longwriterow = file_init.longwriterow
            self.fmin = new_instance.get_fmin
            self.fmax = new_instance.get_fmax
            self.freq = new_instance.get_freq

        def local_defaults(self)-> None:
            pass

        def local_arguments(self)-> None:
            new_instance.argparser.add_argument('-fl','--fmin', metavar='', type=float, def
            new_instance.argparser.add_argument('-fr','--freq', metavar='', type=float, def
            new_instance.argparser.add_argument('-fh','--fmax', metavar='', type=float, def

        def set_frequency(self, value, errdelay = 3) -> None:
            """change reference frequency"""
            self.interface.write("FREQ "+"{:.4E}".format(value))
            pass

        def autogain(self)->None:
            self.interface.write("AGAN")

        def set_phase(self,value) -> None:
            self.interface.write("PHAS "+str(value))
            pass

        def time_constant(self,choise) -> None:
            self.interface.write("OFLT "+str(choise))
            pass

        def sensitivity(self,choise) -> None:
            self.interface.write("SENS "+str(choise))
            pass

        def set_sample_rate(self, choise)->None:
            self.interface.write("SRAT "+str(choise))

        def start_data_acquision(self) -> None:
            self.interface.write("STRT")
            pass

        def pause_data_acquision(self) -> None:
            self.interface.write("PAUS")
            pass

        def reset_data_acquision(self) -> None:
            self.interface.write("REST")
            pass

        def get_data(self) -> None:
            pass

        def get_data_explicitly(self, data_variable=3, errdelay=3):
            """
            two params, give resource object and the second params is parameter to variable
            default to data_variable = 3 which is equievalent to reading R.
            as SR830manual,
            data_variable = 1 => X,
            data_variable = 2 => Y,
            data_variable = 3 => R,
            data_variable = 4 => phase
            """
            return self.interface.query("OUTP? "+str(data_variable))


else:

    from pyinstro.interfaces import rs232

    class SR830(rs232.RS232):
```

This is some utilities to ease control of scientific instruments,
FIlewrite simple data logger: *getfile.py*

```
from pyinstro.utils import getpath



import csv
import os



class Get_File:
    """
    INFO: just to write file, must be CSV
    """
    def __init__(self,file) -> None:
        _project_dir_path_abs = getpath.getpath()

        if os.path.exists(os.path.join(_project_dir_path_abs,"data")):
            _data_dir_path_abs = os.path.join(_project_dir_path_abs,"data")
        else:
            os.mkdir(os.path.join(_project_dir_path_abs,"data"))
            _data_dir_path_abs = os.path.join(_project_dir_path_abs,"data")

        if file=='default':
            file = "auto0.csv"
            count = 0
            while os.path.exists(os.path.join(_data_dir_path_abs,f"auto{count}.csv")):
                count+=1
                file = f"auto{count}.csv"

        file = os.path.join(_data_dir_path_abs,file)

        # i did not used re module down here
        if not ((file[len(file)-1]=='v')and(file[len(file)-2]=='s')and(file[len(file)-3]=='
            file = file+".csv"
        else:
            pass

        self.filepath = file
        self.firsttime = True

    def writerow(self, data)-> None:
        """
        open file one time ad write it
        """
        if self.firsttime:
            self.file = open(self.filepath,'w',newline='')
            self.writer = csv.writer(self.file)
            self.writer.writerow(data)
            self.firsttime = False
            print(self.filepath)
        else:
            self.writer.writerow(data)


    def longwriterow(self,data)->None:
        """
        for long data, i think it is suitable to write file each time open and close
        """
        with open(self.filepath,'a',newline="") as datafile:
            self.writer = csv.writer(datafile)
            self.writer.writerow(data)
```

DEFAULT setting: *defaults.py*

```
from pyinstro.utils import getpath

import os
import configparser

class DefaultParams:
    """
    specify default parameters

    for more info:
    refer to SR830 manual for more info.
    """
    time_constant = 5
    sensitivity = 5
```

```
# filter_slope =

baud_rate = 9600
sample_rate = 10
gpib_address = 1
time_delay = 1

connection = 1          # means GPIB, 1: GPIB, 2: RS232, 3: USB, 4: LAN
connections = {1:"GPIB", 2:"RS232", 3:"USB", 4:"LAN"}

fmin = 01E+3
fmax = 01E+5

partitions = 4
levels = 4

data= 3

def __init__(self) -> None:
    self.defaults_params_list= [attr for attr in dir(self) if not callable(getattr(self
    #defaults_params= dict(zip(defaults_params_list,list(" "*len(defaults_params_list))

    self.target_path = getpath.getpath()
    config_file = os.path.join(self.target_path,"config.ini")

    print("checking config.ini file")

    if os.path.exists(config_file):
        config = configparser.ConfigParser()
        config.read(config_file)
        config_file_dict = config.defaults()
        if len(config_file_dict)==len(self.defaults_params_list):
            for keys in config_file_dict:
                if (config_file_dict[keys].isspace() or not config_file_dict[keys]):
                    pass
                else:
                    setattr(self, keys, config_file_dict[keys])
                    print(keys+":  "+config_file_dict[keys])
        else:
            for keys in self.defaults_params_list:
                if not (keys in config_file_dict):
                    with open(config_file, "w") as _conf_file:
                        config.set("DEFAULT",keys," ")
                        config.write(_conf_file)
                else:
                    if (config_file_dict[keys].isspace() or config_file_dict[keys] =="
                        pass
                    else:
                        setattr(self, keys, config_file_dict[keys])
                        print(keys+":  "+config_file_dict[keys])

    else:
        pass

def makeconfig(self):
    config_file =os.path.join(self.target_path,"config.ini")
    config = configparser.ConfigParser()
    if os.path.exists(config_file):
        config.read(config_file)
        config_file_dict = config.defaults()
        if len(config_file_dict)==len(self.defaults_params_list):
            pass
        else:
            for keys in self.defaults_params_list:
                if not (keys in config_file_dict):
                    with open(config_file, "w") as _conf_file:
                        config.set("DEFAULT",keys," ")
                        config.write(_conf_file)

                else:
                    pass
            print("I had appened to full option to config file!")
    else:
        with open(config_file,'w') as config_file:
            config_file.write("[DEFAULT]\n")
            for params in self.defaults_params_list:
                config_file.write(params+" = \n")
        print("I had made config file in present directory !")
```

Code for data analysis

This is code for where I made my tools for data analysis. This tools presents with me file opening, file reading, taking mean over single frequency data, sorting my data with deviation method, plotting single data points form points etc. all the data and code is at following github link hrefhttps://github.com/vijaypanchalr3/shotnoisehttps://github.com/vijaypanchalr3/shotnoise.

tools.py

```
import csv
import re
from numpy import array,mean,abs,split,vstack,argsort,column_stack

__all__=[
    "files"
    ]
```

```
class files:
    """
    PARAMETER: filename as Relative path to __file__
    RETURN: nil
    FUNCTION: read files named filename, write other files with data
    """
    def __init__(self,filename:str,datatype=float) -> None:
        self.datatype = datatype
        self.filename = filename

        with open(filename, 'r',) as newfile:

            self.fobject = list(csv.reader(newfile))

            # omit first member
            try:
                datatype(self.fobject[0][0])
                self.header = None
            except ValueError:
                self.header = self.fobject.pop(0)

            self.length = sum(1 for row in self.fobject)

    def file_add(self,nameaddition:str="extra"):
        _finalfile = re.split("\/",self.filename)
        finalfile = re.split("\.", _finalfile[len(_finalfile)-1])
        self.finalfile = "/".join(str(_finalfile[i]) for i in range(len(_finalfile)-1))+"/'

    def write_another(self,data:list)-> None:
        with open(self.finalfile, 'wxs', newline="") as cleandata:
            writer = csv.writer(cleandata)
            writer.writerow(data)

    def get_mean(self):
        data = array(self.fobject)
        try:
            first_array,second_array = split(data,2,axis=1)
        except IndexError:
            print("from get_mean()::empty array from data")

        extra_array,_first_array,_second_array = [],[],[]
        count = 0
        _first_array.append(first_array[count][0])
        for i in range(0,len(data)):
            if first_array[i][0]==_first_array[count]:
                extra_array.append(second_array[i][0])
            else:
                _second_array.append(mean(array(extra_array,dtype=float)))
                count+=1
                _first_array.append(first_array[i][0])
                extra_array=[]

            if i==len(first_array)-1:
                _second_array.append(mean(array(extra_array,dtype=float)))

        return vstack((array(_first_array,dtype=float),array(_second_array,dtype=float))).T

    def sort_on_deviation(self, points=5):
        data = array(self.fobject, dtype=float)
        filtered_data=[]
        temp_erray = []
        index = float(data[0][0])
        count = 0
        for datapoint in data:
            count+=1
            if datapoint[0]==index:
                temp_erray.append(float(datapoint[1]))
            else:
                nlist = array(temp_erray,dtype=float)
                m = mean(nlist)
                sorted_deviation = argsort(abs(nlist - m))
                filtered_nlist = nlist[sorted_deviation[:points]]
                indexonelist = array([index]*points)
                final_list= column_stack((indexonelist,filtered_nlist))
                for member in final_list:
                    filtered_data.append(member)
                index = float(datapoint[0])
                temp_erray = []

            if count == len(data)-1:
                nlist = array(temp_erray,dtype=float)
                m = mean(nlist)
                sorted_deviation = argsort(abs(nlist - m))
                filtered_nlist = nlist[sorted_deviation[:points]]
                indexonelist = array([index]*points)
                final_list= column_stack((indexonelist,filtered_nlist))
                for member in final_list:
                    filtered_data.append(member)
        return array(filtered_data)

    def shady_plot(self, color="Blues"):
        """
```

```
        """
        data = array(self.fobject,dtype=float)

        # [here] can be better memort handling !
        common_array = [[]]
        index = float(data[0][0])
        count = 0
        for datapoint in data:
            if float(datapoint[0])==index:
                try:
                    common_array[count].append([float(datapoint[0]),float(datapoint[1])])
                    count+=1
                except IndexError:
                    common_array.append([])
                    common_array[count].append([float(datapoint[0]),float(datapoint[1])])
                    count+=1
            else:
                count=0
                index = float(datapoint[0])

        delete = []
        common_array.pop()
        for i in range(len(common_array)-1):
            if len(common_array[len(common_array)-1])<len(common_array[i]):
                common_array[i].pop()
        common_array = array(common_array)
        from matplotlib import cm
        colormap = cm.get_cmap(color, len(common_array))
        return common_array,colormap

    def point_mean(self,data):
        freq = []
        vo = []
        freq.append(float(data[0][0]))
        vo.append(float(data[0][1]))
        count = 1
        for i in range(1,len(data)):
            lenth = len(freq)
            if float(data[i][0])==freq[lenth-1]:
                vo[lenth-1]+=float(data[i][1])
                count+=1
            else:
                vo[lenth-1]=vo[lenth-1]/count
                count=1
                freq.append(float(data[i][0]))
                vo.append(float(data[i][1]))
        freq.pop()
        vo.pop()
        return array(freq,dtype=float),array(vo,dtype=float)

if __name__=="__main__":
    print("No Error")
```