Fermi National Accelerator Laboratory

SQMS center - Physics Sensing group

in collaboration with

University of Padua

Physics and Astronomy department - "Galileo Galilei"

October 14, 2024

Italian Summer Student Program 2024

Final Report

Preparation for DM searches with high Q SRF cavities

Fabio Castañeda
fabio.castanedarestrepo@studenti.unipd.it
frestrep@fnal.gov

Supervisor: Dr. Bianca Giaccone
giaccone@fnal.gov

Contents

Abstract

This work focuses on the preparatory for two dark matter searches based on high Q SRF cavities. In the context of the SERAPH experiment I participated to experimental work of characterizing the cavity at mK temperature and subsequently analyzed the data collected. For the SHADE experiment, I worked on the preparation of the data analysis starting from the collection of simulated data and built a flexible framework to analyze them.
The goal of this report is to present my contribution to these two experiments that are being investigated at Fermilab by the Physics Sensing group at SQMS.

1 DM search with Haloscopes

Axions are well motivated Dark Matter (DM) candidates [1], that could solve the strong CP problem of QCD [2]. In the GHz regime, top class experiments like ADMX [3] and HAYSTAC [4] have proven that haloscopes can reach the required sensitivity to target the QCD axion.
Haloscopes are resonant cavities equipped with a low noise readout and cryogenic cooling that allow the coherent signal coming from the DM halo to build up into its volume and be detectable as a power excess at the desired cavity mode frequency.[5]
Other DM candidates like dark photons (DPDM), are detectable with Haloscopes with little adaptation of the setup required by an axion search [6]

1.1 two different DM searches

SQMS national center is working at the moment on two experiments based on high quality factor, Q, SRF cavities to search for different dark matter candidates.
SERAPH is sensitive to dark photons and aims to target a wide range of masses through its peculiar plunger design. SHADE targets light axions thanks to a new approach to haloscope searches.

2 SERAPH

SERAPH's Plunger Cavity has an innovative design that allows wide tunability, which we know is a key feature of a well designed DM search experiment.
This comes in addition to the fact that the cavity is made of Niobium, thus is superconductive with a ultra high Q factor ($\sim 10^8$).

The cavity design is far from conventional,

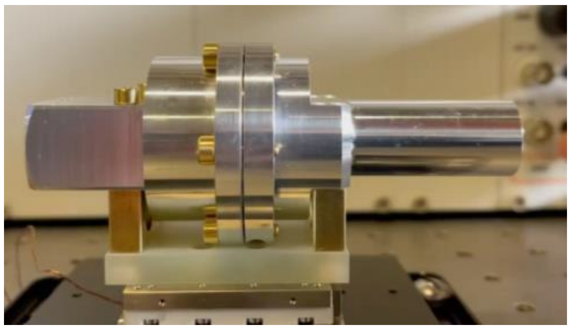
Figure 1: Picture of the SERAPH cavity
Credits: Dr. Cervantes, SQMS

because we're dealing with a cylindrical cavity open on one side where a smaller diameter cylinder guide allows a so called plunger to slide into the cavity volume. The plunger is also made of bulk Niobium and supported by a sapphire rod (Figure 1).
Tuning the cavity consists in moving it with a piezoelectrically actuated motor in order to control the relative insertion of the plunger.

During the time of this internship the cavity was cold in the dilution refrigerator, DR, so a series of calibration and characterization measurements have been conducted. The goal of the measurement session was to ultimately conduct a preliminary DM search, which is the

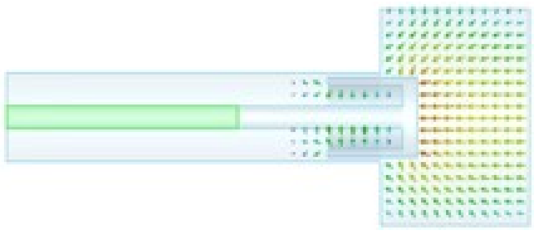
Figure 2: Electric Field profile simulated with a partially inserted plunger,
Credits: Dr. Cervantes, SQMS

only step that wasn't undertaken because, as we will see in the following, considerations from the calibration steps allowed to conclude that the cavity wasn't setup properly to obtain valuable data in terms of sensitivity.
In the following all the steps of the calibration protocol will be briefly explained. I personally assisted Dr. Raphael Cervantes and Daniel Molenaar (graduate intern collaborating with Dr. Cervantes) during the lab operations. In the following, regarding the data analysis it will be made clear which steps have been performed by me.

2.1 Tunability

As we've mentioned, the peculiar design of this cavity allows for a wide tunability range of the TM_{010}-like mode of about 3 GHz, from 4 to 7 GHz. Unfortunately due to mechanical issues, the mode's central frequency could be moved across $\simeq 300\,\mathrm{MHz}$ (from 6.24 GHz to 6.56 GHz, see Figure 5), with acceptable levels of heat injections from the piezo system (characterization of this phenomenon has been postponed to future runs). The issue was probably due to the plunger getting stuck while retracting: the motivations for the limit in tuning range will be investigated once the experiment is warmed up.

2.2 Cryogenic electronics

Let's first briefly describe the experiment's electronic chain (see Figure 3 for the scheme of the cold part of the chain). A3 and A4 ports are the input ports, along these lines cryogenic coaxial cables send the signal through a series of attenuators and filters at various temperature stages. On the other side ports B3 and B4 are output ports along which the signal is amplified by a cryogenic High Electronic Mobility Transistor, HEMT in the 4K stage. Attenuators and isolators are placed along the lines, in particular the output line, to avoid reflection signals from the amplifier to travel backwards on the line and insert additional thermal noise into the cavity.

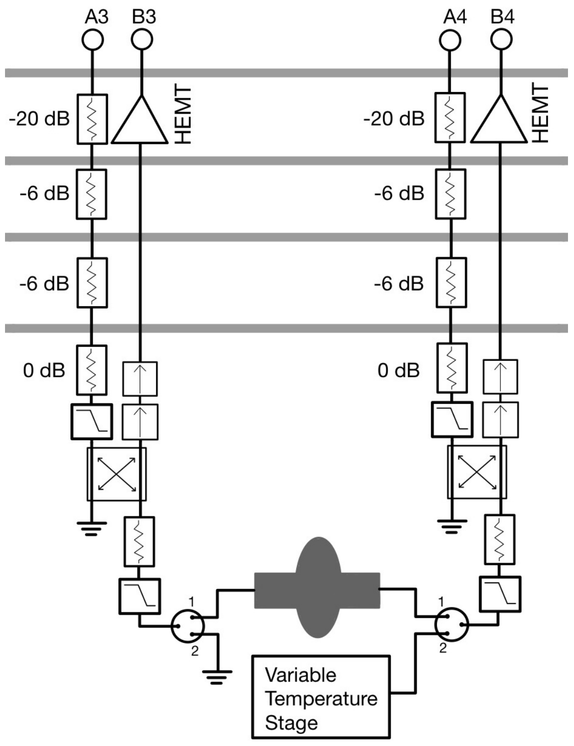

Figure 3: Complete scheme of the cold electronics of the Plunger Cavity experiment

Being the DR equipped to support different experiments at the same time, two series of switches are installed in the mixing chamber, where the cavities are, in order to be able to switch between cavities. Such switches also allow to completely exclude the cavity and instead connect B4 line, the transmission line, to a heatable $50\,\Omega$ load called Variable Temperature Stage, a key component to perform thermal calibration of the line as explained in section 2.6.

2.3 Decay measurements

A fundamental figure of merit of the cavity response is the Q factor. Since the cavity is tunable one needs to estimate its value across the the frequency range of interest. The Q factor extracted from spectral response measurements with a Vector Network Analyzer, VNA is not to be considered reliable for Q factors above a certain limit (the consensus of the community suggests this limit to be $Q \sim 10^7$). This is due to the resolution limits of the instrument, combined with the effect of mechanical vibrations on the cavity resonant frequency. These microphonics, as they're called, broaden the

linewidth of the mode and lower the carrier mode amplitude (see section 2.5).
This is why SRF cavity characterization relies on decay measurements to extract the loaded quality factor, Q_L.
To perform a Decay measurement connect the VNA to all four lines coming out of the DR. Then set up the instrument to send a narrow-band signal precisely at the central frequency of the cavity mode previously measured. We drive the cavity to equilibrium (constant stored energy), then switch off the rf source abruptly and record the transmission response. The transmitted power should decay exponentially with a time constant which is proportional to the loaded quality factor : $Q_L = 2\pi f_0 \tau_L$. This parameter doesn't depend only on the internal cavity response, but also on the coupling to the input and output antennas. For a com-

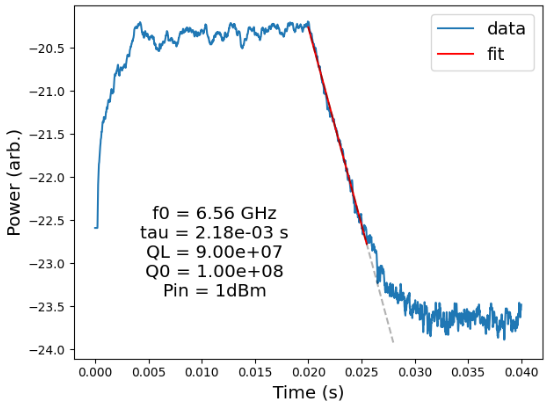

Figure 4: Decay data with fitted exponential function. The estimated parameters are printed on the plot. Y dimension is logaritmic

plete treatment of cavity characterization and testing refer to Padamsee et al.[7]. Here we will only report the relation between Q_L, Q_0, the intrinsic quality factor of the cavity and β_α, the coupling parameters useful to extract Q_0 from decay measurements.

$$Q_0 = Q_L (1 + \beta_i + \beta_t) \tag{1}$$

$$\beta_\alpha := \frac{Q_0}{Q_\alpha}, \quad \alpha = t, i \tag{2}$$

Decay measurements have been taken at three different frequencies to span the whole tuning range accessible during this run (see above section 2.1). For reference Figure 4 shows one of the fitted decays with the extracted parameters (more plots available in the appendix ??), while Figure 5 presents the extracted Q_0 values against frequency. This data was analyzed with python scripts partially adapted and partially developed by me.

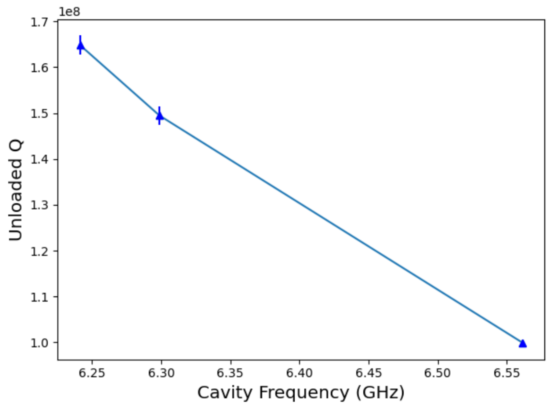

Figure 5: Unloaded quality factor with error bars as a function of frequency

2.4 Coupling parameters

The coupling parameters can be extracted via reflection spectral measurements, S_{11} or S_{22}[8], through a analysis procedure called circular fitting: around the central frequency of the cavity, the reflection coefficient is expected to rotate along a circle in the complex plane, from the radius of such circle the coupling parameter value of each antenna can be extracted[8].
Due to some complications during the data taking procedures and other issues valuable data to perform such analysis couldn't be recorded during this measurement session. The results presented then, rely on a previous estimation of the quality factors of the coupling antennas performed in LHe at $\sim 4K$ (see procedure [9]). Doubts about the current experiment reach arised when we noticed that spectra of the S reflection parameters showed almost no reflection dip at the frequency where the transmitted paramters exhibited a standard lorentzian response. This could be an indication that both lines are probably much undercoupled with respect to what expected and desired. However the antenna weren't change from previous runs, so this apparent undercoupling could just be due to the effect of microphonics changing the frequency of the cavity and the VNA not be-

ing able to follow such frequency and excite the cavity efficiently. The combination of this issue, the impossibility of investigate further where it was coming from and strict time constraints resolved into the decision of limit this run to characterization measurements.
In general, the power deposited by the DPDM candidate if resonant with the cavity mode is expected to be limited by the coupling parameter of the transmission line. In fact, in natural units the dark photon signal power inside the cavity is [10]

$$P_S(f) = P_0 \frac{\beta}{\beta+1} L(f, f_0, Q_L) \tag{3}$$

where

$$P_0 = \eta \chi^2 m_\gamma \rho_{DM} V G Q_{DM} \tag{4}$$

if $Q_L >> Q_{DM}$, quality factor of the DM lineshape predicted by theory ($Q_{DM} \sim 10^6$). L is the lorentzian function that account for the cavity spectral response, η the attenuation factor of the carrier mode due to microphonics, V is the effective volume of the cavity, G the form factor, ρ_{DM} is the Dark matter halo density on earth (equal to 0.45 GeV/cm^3), m_γ the mass of the Dark photon and finally χ^2 the chinetic mixing strenght, the parameter that characterizes the reached sensitivity of the experiment.
This is why an optimal experiment wuold want a port which is extremely undercoupled to use as input line (which is ultimately useful only during characterization) and a critically or over coupled port for the transmitted power.

2.5 Microphonics

In the framework of a DM search, a critical phenomenon that influences the sensitivity is microphonics. This term refers in general to random mechanical vibrations induced from the external environment. Sources could be for example, but not limited to: earthquakes, operators working in the lab around the DR, vacuum pumps, boiling of the LHe.Their effect is to have the modes central frequencies to oscillate in time, meaning that the effective response of the cavity changes from a singular carrier peak for each mode, to a series of sidebands whose amplitude is proportional to the time that the cavity spends at each detuned frequency during it's coherence time. A more detailed model and description of this phenomenon is described by R. Cervantes et al.[11].
Super high Q SRF cavities are deeply affected by microphonics, due to the scale of their bandwidth's ($\lesssim 1\,\text{Hz}$) against the usual detuning observed ($\gtrsim 1\,\text{Hz}$). However, due to the open configuration and the degrees of vibrational freedom that the plunger has, this cavity is affected by $\simeq 1\,\text{kHz}$ microphonics.
Following the method described in [11] a Self Excited Loop, SEL, was implemented in order to then measure with a phase noise analyzer, PNA the time dependant frequency response of the cavity. The SEL[12] is a feedback circuit (see Figure 6) that allows to power up the cavity thanks to the fact that the circuit tracks the frequency of the cavity by itself and amplifies its same signal if the loop phase is shifted in order to match itself after each loop.

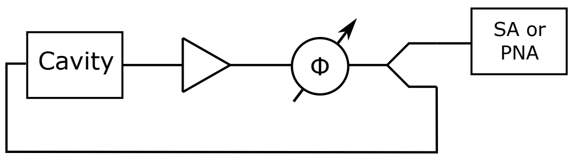

Figure 6: Circuit scheme of a SEL equipped with a power splitter that allows reading through a signal analyzer, SA or phase noise analyzer, PNA.
Figure taken from [11]

The PNA at this point is ready to record data which is plotted in Figure 7 and then Fourier transformed to look at the spectral dependence of the microphonics. Microphonics is essentially caused by random noise meaning that we usually don't expect to have sharp peaks at any frequency. When this happens one can relate this phenomenon to more persistent vibrational effects. Often times this is what happens in the DR, due to the vacuum pumps injecting noise systematically at fixed frequencies. However, in the plots we can see that in this case both liquid Helium, LHe and DR measurements exhibit largely dominant features at similar frequencies.
Such frequencies happen to be close to the ones of the mechanical degrees of freedom of the plunger according to simulations, demonstrat-

ing that in this experiment cavity design is key to limit attenuations that draw power from the carrier mode. My contribution was limited to reproduce these plots using existing scripts.

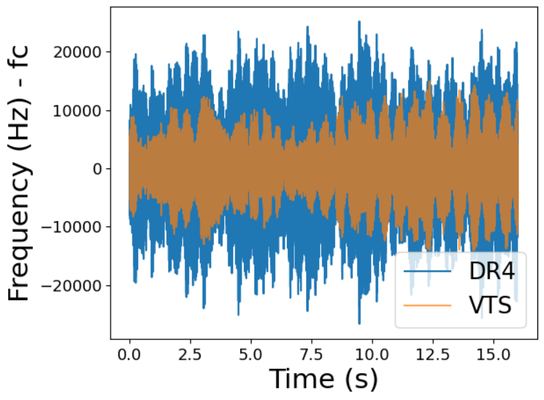

Figure 7: Frequency detuning as function of time recorded by the phase noise analyzer, PNA
The data in blue is measured inside the DR while the one in orange is obtained in the LHe facility (called VTS) at 4K

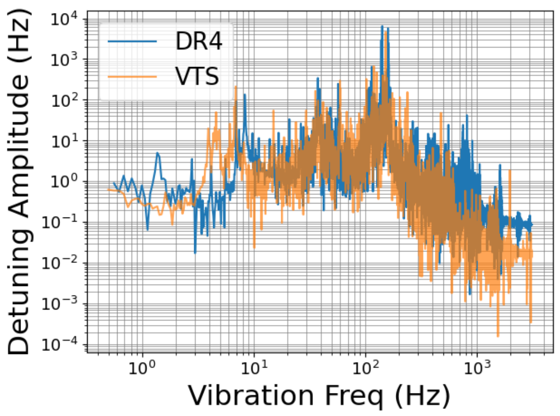

Figure 8: Spectral characterization of microphonics
The data in blue is measured inside the DR while the one in orange is obtained in the LHe facility (called VTS) at 4K

2.6 Y-factor calibration

DM search measurements ultimately aim at the extraction of the SNR that the setup allows to reach. This means that the noise sources have to be characterized with care. Provided that cavity and experiment design are optimized, haloscope searches are then basically limited by thermal noise and amplifier noise of the receiver chain. In this limit the SNR is equal to:

$$SNR = \frac{P_S}{k_b T_n}\sqrt{\frac{t_{int}}{b}} \tag{5}$$

following Nyquist formalism of noise treatment. t_{int} is the total integration time for each tuning step.
Johnson formula, in fact, parametrizes random noise introduced by an amplified line as temperature:

$$P = G k_b b \left(T_{sys} + T_{add}\right) \tag{6}$$

where k_b is the boltzmann constant, b the resolution bandwidth of our measurements, T_{sys} the system temperaure before amplification, G is the gain of the amplifier and T_{add} is the so called amplifier added noise temperature.
G and T_{add} are treated as effective parameters that summarize the influence on the signal of all components in the line (Friis cascade analysis). The advantage of this approach is that Y factor calibration allows to directly measure such effective parameters[13] thanks to the varibale temperature loa mentioned before.

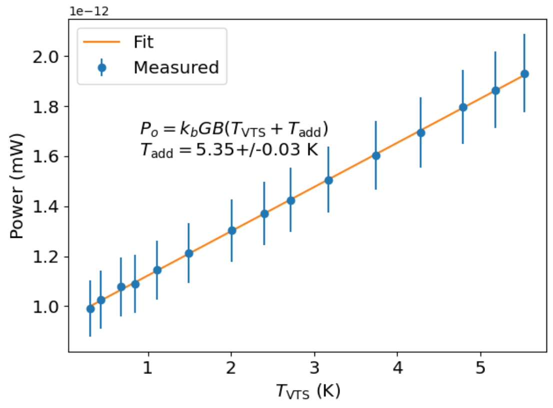

Figure 9: Y factor calibration data end exctracted parameters

In a dark matter search is important to measure T_{add} of the transmission line (B4 in this case) because this allows to compute $T_n \simeq T_{cav} + T_{add}$. In practice, when one excludes the cavity with the cryogenic switch and connects instead the heatable load $T_{sys} = T_{load}$ (up to $\simeq 10\,\mathrm{K}$), extraction of the desired quantities with a linear fit is made possible by measuring noise power against T_{load}. The results are shown in Figure 9 while raw data is in the appendix.
However, interesting considerations can be drawn from studying the spectral dependence

of gain and added temperature, especially to characterize gain variations across the analysis span. My contribution in this part of the analysis was indeed this. Instead of extracting a mean value of temperature and gain over all measured frequency, a python script was developed specifically to extract such value for each measured point (see Figure 10, 11).
What's interesting about these results is that we can clearly see that the temperature variation with frequency is not compatible within its error with the mean value extracted. The temperature variations are between 4K to 7K and hugely dominate system's thermal noise.
Further steps will need to check this results through a cascade analysis of the given line.

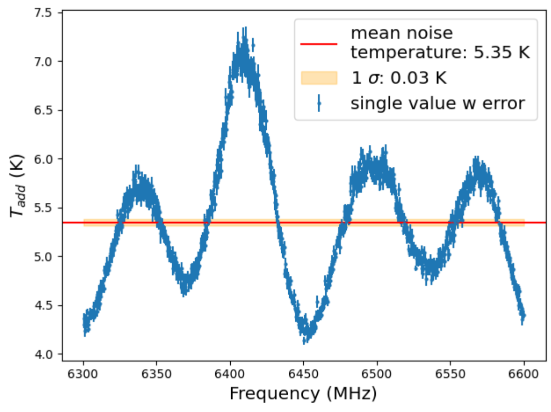

Figure 10: Added amplifier temperature spectral dependence

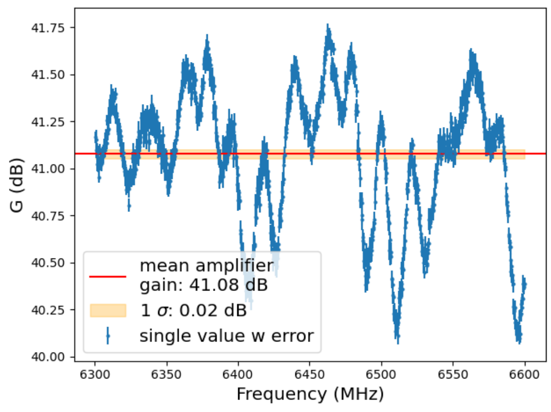

Figure 11: Amplifier gain estimated spectral dependence

3 SHADE

SHADE (Superconducting Heterodyne Axion Dark matter Search) is a dark matter search that leverages ultra high Q SRF cavities and the heterodyne principle, as proposed by Asher Berlin et al[14]. Heterodyne searches bring two main advantages with respect to standard haloscopes:

1. they allow to implement axion searches without the need of an external magnetic field, enabling to leverage ultra high Q SRF cavities for such searches.

2. they extend the target mass range, possibly to extremely low axion masses, limited only by the control of noise sources

Heterodyne detection is a common approach in signal processing and in the case of SRF cavities the conversion will be observed between two cavity modes, called pump ω_p and signal mode ω_{sig}. The pump mode is driven by an external oscillator matched to its frequency and the presence of an axion DM would be detected as a power excess at ω_{sig} only if

$$\omega_{sig} \simeq \omega_p \pm m_a \tag{7}$$

because the DM field resonantly drives power from the pump mode to the signal mode, therefore taking advantage of the pump mode magnetic field B_0 for the inverse Primakoff conversion.
The SHADE experiment will consist on an elliptical Niobium cavity, whose pump and signal mode will be the TM_{020} and TE_{011} respectively.[15] The cavity is designed to have an initial 1 MHz splitting between them and the signal mode to have a frequency of $\omega_{sig} = 2\pi \times 1.41$ GHz with a $Q_{0,sig} \geq 10^{10}$ at 1.4 - 2 K. The cavity is then tunable with a mechanical tuner that would bring the splitting down to about 1 kHz at which point the experiment is expected to be limited by microphonics and leakage noise (see below).
The importance of having high Q SRF cavities is made evident by the advantage brought by them in terms of stored energy in the pump mode (scales with B_0^2), amplification of the excess signal and reduction of noise, in particular at very low axion masses.

Since the experiment is not yet ready to perform data acquisition, in this work some preliminary studies and data analysis on a set of simulated data will be presented. The

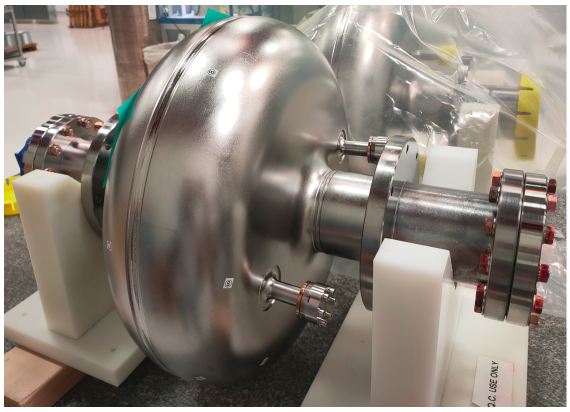

Figure 12: Picture of two cavities commissioned for SHADE experiment

simulated data is taken by terminating a RS-FSW signal analyzer with a 50-Ohm load and measuring background noise spectra at room temperature. The analysis would then automatically renormalize for the expected noise temperature.

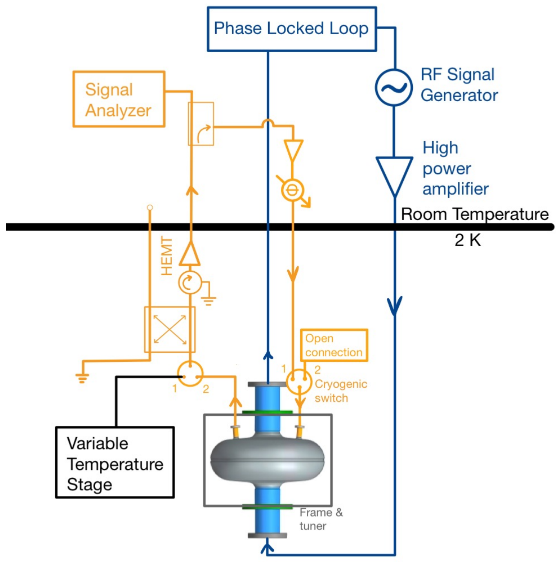

Figure 13: Schematic drawing of the SHADE experiment

The designed experiment setup can be consulted in Figure 13 (see [15] for further details). Here we will briefly mention the presence of:

- two sets of couplers, one for each mode, placed at different positions in order to minimize cross talks and leakage noise. In particular, the coupler ports in yellow placed on the sides are the ones that allow readout of the signal mode;
- a Phase Locked Loop, an active feedback circuit component that allows to lock to the central frequency of the pump mode, to power it up more efficiently;
- a variable temperature stage to perform line calibration;
- a SEL on the signal mode readout lines when microphonics measurements are in order or to measure the frequency of the signal mode;
- a cryogenic HEMT and a signal analyzer to measure the axion power

100 tuning steps have been simulated, each 3 Hz apart and consisting in a 100 times averaged power spectrum spanning 3.2 kHz (analysis bandwith or ABW) around the central frequency with a resolution bandwith, RBW of 0.1 Hz in order to be at least comparable with the linewidth of the cavity, expected to be of the same order of magnitude.

	value	unit
RBW	0.1	Hz
f_c	1.41	GHz
ABW	3.2	kHz
tuning step	3	Hz
N_{avg}	100	
N_{steps}	100	

Table 1: Parameters of the simulated data based on the expected limitations of the experiment

The following analysis steps have been designed by Dr. Bianca Giaccone and me to adapt HAYSTAC[4] and ADMX[3] analyses to an heterodyne search with high Q cavities. What's next is my original contribution to the experiment: the result of my work supervised by Dr. Giaccone. These are preliminary results and yet to be published.

3.1 Spectral baseline

In absence of any DM signal the axion search data should look like random noise around $T_n \simeq T_{cav} + T_{add}$, according to Johnson-Nyquist definition of thermal noise (Equation

6).
However if the ABW is large enough one can see the influence of gain variation of both the IF and RF electronics across the spectrum.
This is why experiments with lower Q cavities, need to subtract both RF and IF spectral baselines, with a corresponding loss of sensitivity due to the application of a Savitzky-Golay low passband filter[16] (any low passband software filter may be applied).

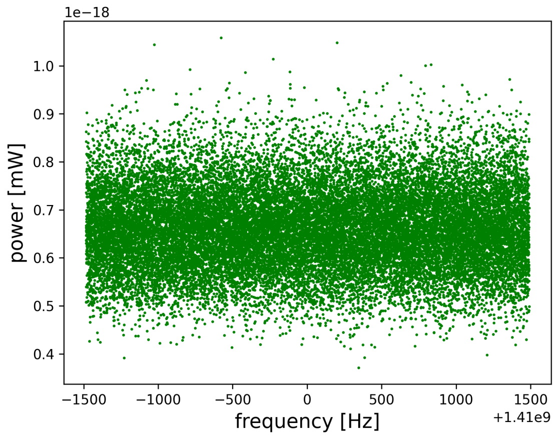

Figure 14: Raw power spectrum recorded for one tuning step

High Q cavities like SHADE and SERAPH can in principle avoid this step and gain in sensitivity because the ABW is narrow enough not to appreciate significant gain variations.
This can be proven by analyzing the distribution of the excess power and confront it with an expected gaussian distribution. Then based on the χ^2 value of the gaussian fit one can discard tuning steps that exceed a certain threshold. Even though χ^2 values have been estimated and attached to the plots, this filtering is not implemented in the code. Statistics can be improved by increasing the number of averages, however a 100 average spectrum with RBW of 0.1 Hz takes about 15 minutes to be recorded by the SA. The effect is that the distribution is better fitted by an asymmetric gaussian, meaning that the underlying χ^2 distribution of each individual power value is still visible. The choice of 100 AVG and 0.1Hz is a compromise between statistics and scan duration with the current experimental setup.
With the current setup it was decided to take the simulated data as it is and approximate it

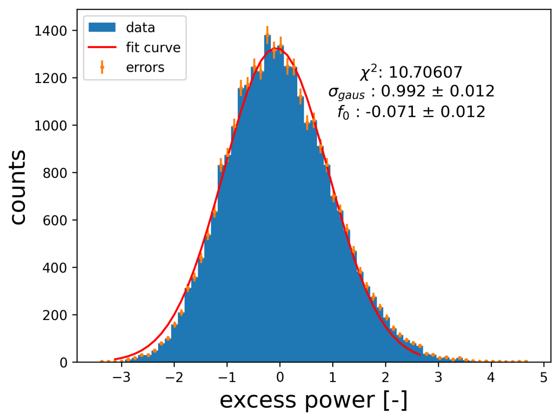

Figure 15: Excess power distribution data of a single tuning step fitted with a symmetric normal function.

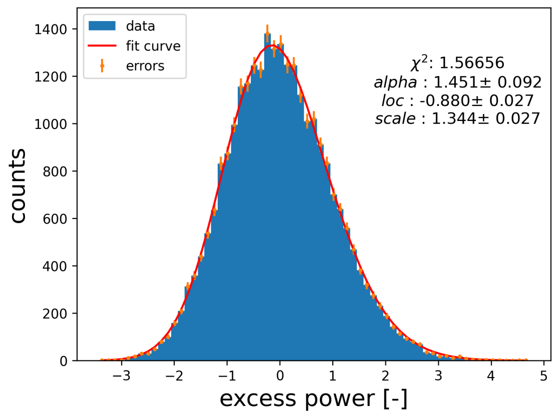

Figure 16: Excess power distribution data of a single tuning step fitted with a skewed normal function. The skew parameter, α is zero for a symmetric normal function. The reduced χ^2 is another indication that this model better fits the data than a symmetric gaussian (see Fig 15)

to gaussian random noise advocating the principles of the central limit theorem.

3.2 Sensitivity estimation

A thorough theoretical treatment of heterodyne haloscope DM searches can be found in the work by A. Berlin et al.[14]. From their calculation one can exctract the complete formula of the axion deposited power in the cavity:

$$P_0 \simeq g_{a\gamma\gamma}^2 \rho_{DM} V \eta_{10}^2 B_0^2 \begin{cases} \frac{\pi Q_a}{m_a}, & \text{if } Q_{L,sig} >> Q_a \\ \frac{Q_{L,sig}}{f_{sig}}, & \text{if } Q_{L,sig} << Q_a \end{cases} \tag{8}$$

The values of the parameters relevant for this analysis are to be found in Table 2, while $g_{a\gamma\gamma}$ is the axion to photon coupling parameter that like χ^2 for DPDM searches parametrizes the sensitivity reach of the experiment

	value	unit
V	25	L
η_{10}	0.193	
B_0	100	mT
β	1	
T_{add}	1.3	K
T_{cav}	2	K
ω_{sig}	$2\pi \times 1.41$	GHz
$Q_{0,sig}$	10^{10}	

Table 2: Relevant parameters of the SHADE experiment. Some are estimated via simulation like the effective cavity volume, V, the coupling parameter η_{10}, some are taken from the datasheets of the electronic components to be used, others are chosen by the operator as reasonable values

As mentioned in the paper by Berlin this formula is valid only in the limit that axion bandwidth is order of magnitudes greater than the cavity bandwidth, which is not the case for this experiment. Therefore, a more correct approach is to integrate equation number (22) of section III[14]. Equation 8 is given as reference to better underline the key magnitudes that play a role in computing the sensitivity, which is then expressed in terms of the parameter $g_{a\gamma\gamma}$:

$$g_{a\gamma\gamma} = g_\gamma \frac{\alpha}{\pi} \frac{m_a}{m_\pi f_\pi} \tag{9}$$

where m_π and f_π are pion mass and decay constant.
Combining equation 8 and 5, which is still valid (see below section 3.3), the expected sensitivity is calculated as:

$$g^2_{a\gamma\gamma} = SNR \frac{k_b T_n}{P_S / g^2_{a\gamma\gamma}} \sqrt{\frac{RBW}{t_{int}}} \tag{10}$$

which results to a value of $2.88 \times 10^{-16}\,\text{GeV}^{-1}$ for SNR = 3.
Future developments of the analysis will have to take care of the numerical computation to estimate a more accurate value of the deposited power.

3.3 Noise sources

In their paper Berlin et al.[14] mention many sources of error that are peculiar of this new detection technique.
All things considered, in the range of masses that SHADE targets the most relevant noise source is still thermal and amplifier noise. The treatment is equivalent to the one presented in section 2.6, so for simulation porpoises we only need to assume a reasonable value of T_{add} and T_{cav} which in this case is 2 (or 1.4) K (see Table 2), because the experiment will be performed in LHe not in a DR.
Further improvements of the analysis will take into account subdominant sources of noise and compute their influence on sensitivity or quantitatively prove that they're negligible.

3.4 Microphonics

Some more words will be dedicated to the characterization of mechanical vibrations effects. Berlin et al.[14] talk about mechanical leakage and model the response of the pump mode.
Preparing for this analysis, the possibility to apply the same response to the signal mode oscillation has been investigated (see Figure 17). This should allow to recover some of the sensitivity lost due to attenuation of the carrier peak, thanks to the sidebands, because for each tuning step the cavity may be sensitive to more than one axion mass.

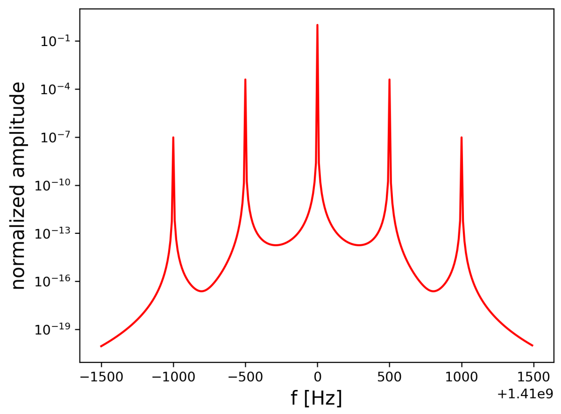

Figure 17: Model of cavity response when microphonics is dominated by two stable in frequency mechanical induced vibrations

3.5 Rescaled spectra

Once the noise sources and the cavity response around the signal mode frequency are established, the next steps of the analysis are mostly consistent with HAYSTAC[4] and ADMX[3].

After extracting the excess power values, δ_p:

$$\delta_p = \frac{P_{raw}}{P_{avg}} - 1 \tag{11}$$

section VI.A in [4] explains how to rescale the excess power in order to:

- account for possible varying parameters, like Temperature, cavity figures of merit and such
- by design relate the error associated to the rescaled excess power to the SNR of such measured data.

In formulas:

$$\delta_r = \frac{P_n \delta_p}{P_S} \tag{12}$$

$$\sigma_r = \frac{P_n \sigma_p}{P_S} = \frac{1}{SNR} \tag{13}$$

where P_n is the total noise power. This is how one ultimately relates the measured data to the limit imposed by it. The result of the rescaling procedure on a single tuning step spectrum can be seen in figure 18

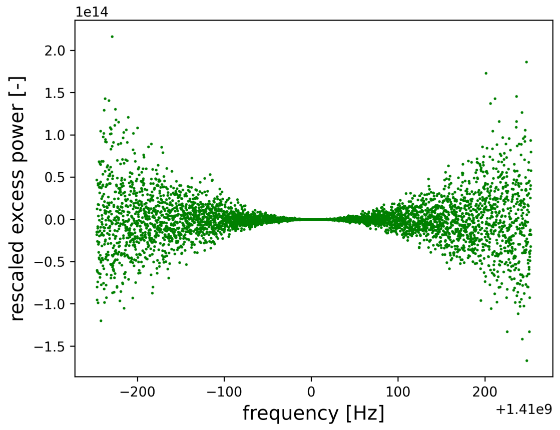

Figure 18: Rescaled power excess spectrum for a single tuning step

In principle computing the rescaled power depends on the value of sensitivity, in terms of $g_{a\gamma\gamma}$, used to compute P_S. However, once the grand spectrum is obtained (section below), the computation of the sensitivity reached by the experiment resolves into computing the ratio between it and the value used to rescale the power (see [4]).
Therefore, the choice of $g_{a\gamma\gamma}$ used for this step is arbitrary (but constant). In this work $g_{a\gamma\gamma}$ was set to the value predicted by KSVZ model[17] for an axion mass of 1 MHz ($1 \times 10^{-19}\,\mathrm{GeV}^{-1}$)

3.6 Convolved spectra and grand spectrum

Since the axion mass distribution has a Q_a factor fixed by theory to a value of 10^6 the axion power signal for each candidate mass could be spread across many bins of the recorded spectra. This is again a key feature of ultra high Q factor cavities even though experiments like HAYSTAC and ADMX still need to take this effect into account.
In order to do so one simply needs to convolve the power data with the expected axion mass distribution which in first approximation can be thought as a simple Maxwell-Boltzmann distribution (see Figure 19).

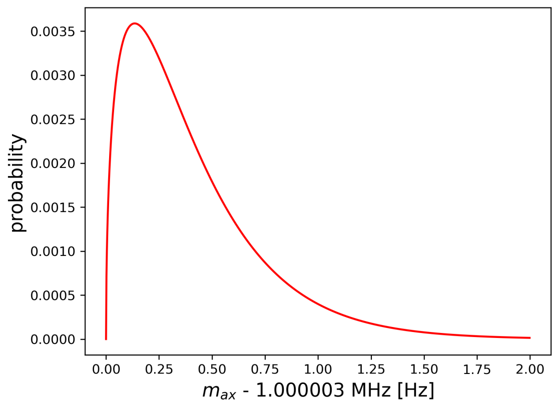

Figure 19: Distribution of axion masses coming from theory

In order to assure that the least amount of sensitivity is lost ADMX analysis[3] and other related papers explain why one needs to perform the convolution as a moving χ^2 minimization fit of the axion lineshape. The result is a convolved spectrum for each tuning step (Figure 20). As final step, the code generates the grand spectrum. The following and final step of the analysis corresponds to the limit extraction. Implementation of this step in the analysis was not possible due to the short duration of this internship.
The grand spectrum simply accounts for the sensitivity contribution of all tuning steps to each axion mass. In a somewhat straightforward manner the grand spectrum bins are the vertical combination of all bins in each tuning step spectrum that correspond to the same axion mass. The combination is performed through a weighted average procedure.

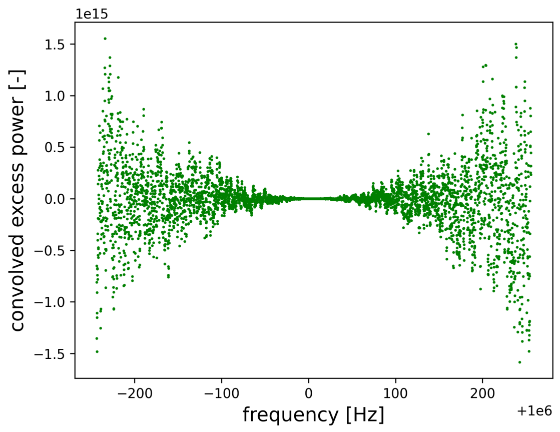

Figure 20: Convolved excess power spectrum for a single tuning step

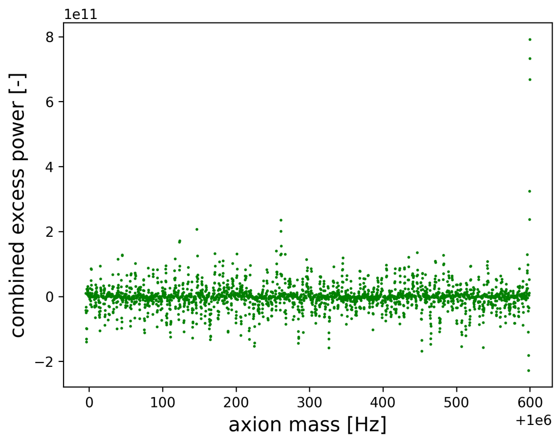

Figure 21: Grand spectrum extracted from the simulated data

At this point the sensitivity limit for each axion mass can be extracted by computing the 90% confidence limit of the underlying gaussian distribution of each combined power excess point in the grand spectrum.
This will be addressed in future endeavours.

3.7 Code

Most of my time during this internship has been dedicated to write the python code to perform the steps of the analysis described just above. The code is implemented as an object-oriented family of six scripts between which the different tasks of the analysis are divided.
The code is flexible and allows to implement the analysis on different sets of data, both regarding DM search data and characterization extracted parameters. In the appendixes one can read through the entirety of the code given for reference.

4 Acknowledgments

In conclusion, I'd like to thank SQMS center for hosting me this summer at Fermilab. This professional experience has set a milestone in my academic path, that couldn't have been so solid without the guidance of Dr. Bianca Giaccone, that supervised my work as the PI of the SHADE experiment. Most sincere thanks and appreciations for her and her work.
Huge thanks to Dr. Raphael Cervantes, who has made space for me to collaborate during the measurement run of the plunger cavity. It's been a pleasure exchange ideas and learn from you.
Last of all I'd like to thank my family: whatever distance divides us, family is home.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, National Quantum Information Science Research Centers, Superconducting Quantum Materials and Systems Center (SQMS) under contract number DE-AC02-07CH11359.

References

[1] Steven Weinberg. A new light boson? *Phys. Rev. Lett.*, 40:223–226, Jan 1978.

[2] R. D. Peccei and Helen R. Quinn. CP conservation in the presence of pseudoparticles. *Phys. Rev. Lett.*, 38:1440–1443, Jun 1977.

[3] C. Bartram, T. Braine, R. Cervantes, N. Crisosto, N. Du, G. Leum, L. J. Rosenberg, G. Rybka, J. Yang, D. Bowring, A. S. Chou, R. Khatiwada, A. Sonnenschein, W. Wester, G. Carosi, N. Woollett, L. D. Duffy, M. Goryachev, B. McAllister, M. E. Tobar, C. Boutan, M. Jones, B. H. LaRoque, N. S. Oblath, M. S. Taubman, John Clarke, A. Dove, A. Eddins, S. R. O'Kelley, S. Nawaz, I. Siddiqi, N. Stevenson, A. Agrawal, A. V. Dixit, J. R. Gleason, S. Jois, P. Sikivie, J. A. Solomon, N. S. Sullivan, D. B. Tanner, E. Lentz, E. J. Daw, M. G. Perry, J. H. Buckley, P. M. Harrington, E. A. Henriksen, and K. W. Murch. Axion dark matter experiment: Run 1b analysis details. *Phys. Rev. D*, 103:032002, Feb 2021.

[4] B. M. Brubaker, L. Zhong, S. K. Lamoreaux, K. W. Lehnert, and K. A. van Bibber. Haystac axion search analysis procedure. *Phys. Rev. D*, 96:123008, Dec 2017.

[5] P. Sikivie. Experimental tests of the "invisible" axion. *Phys. Rev. Lett.*, 51:1415–1417, Oct 1983.

[6] R. Essig, J. A. Jaros, W. Wester, P. Hansson Adrian, S. Andreas, T. Averett, O. Baker, B. Batell, M. Battaglieri, J. Beacham, T. Beranek, J. D. Bjorken, F. Bossi, J. R. Boyce, G. D. Cates, A. Celentano, A. S. Chou, R. Cowan, F. Curciarello, H. Davoudiasl, P. deNiverville, R. De Vita, A. Denig, R. Dharmapalan, B. Dongwi, B. Döbrich, B. Echenard, D. Espriu, S. Fegan, P. Fisher, G. B. Franklin, A. Gasparian, Y. Gershtein, M. Graham, P. W. Graham, A. Haas, A. Hatzikoutelis, M. Holtrop, I. Irastorza, E. Izaguirre, J. Jaeckel, Y. Kahn, N. Kalantarians, M. Kohl, G. Krnjaic, V. Kubarovsky, H-S. Lee, A. Lindner, A. Lobanov, W. J. Marciano, D. J. E. Marsh, T. Maruyama, D. McKeen, H. Merkel, K. Moffeit, P. Monaghan, G. Mueller, T. K. Nelson, G. R. Neil, M. Oriunno, Z. Pavlovic, S. K. Phillips, M. J. Pivovaroff, R. Poltis, M. Pospelov, S. Rajendran, J. Redondo, A. Ringwald, A. Ritz, J. Ruz, K. Saenboonruang, P. Schuster, M. Shinn, T. R. Slatyer, J. H. Steffen, S. Stepanyan, D. B. Tanner, J. Thaler, M. E. Tobar, N. Toro, A. Upadye, R. Van de Water, B. Vlahovic, J. K. Vogel, D. Walker, A. Weltman, B. Wojtsekhowski, S. Zhang, and K. Zioutas. Dark sectors and new, light, weakly-coupled particles, 2013.

[7] Hasan S Padamsee, J. Knobloch, T. Hays, and Perry B. Wilson. Rf superconductivity for accelerators. 1998.

[8] Qi-Ming Chen, Meike Pfeiffer, Matti Partanen, Florian Fesquet, Kedar E. Honasoge, Fabian Kronowetter, Yuki Nojiri, Michael Renger, Kirill G. Fedorov, Achim Marx, Frank Deppe, and Rudolf Gross. Scattering coefficients of superconducting microwave resonators. i. transfer matrix approach. *Phys. Rev. B*, 106:214505, Dec 2022.

[9] O. Melnychuk, A. Grassellino, and A. Romanenko. Error analysis for intrinsic quality factor measurement in superconducting radio frequency resonators. *Review of Scientific Instruments*, 85(12):124705, 12 2014.

[10] Sumita Ghosh, E. P. Ruddy, M. J. Jewell, A. F. Leder, and R. H. Maruyama. Searching for dark photons with existing haloscope data. *Phys. Rev. D*, 104:092016, Nov 2021.

[11] R. Cervantes, J. Aumentado, C. Braggio, B. Giaccone, D. Frolov, A. Gras-

sellino, R. Harnik, F. Lecocq, O. Melnychuk, R. Pilipenko, S. Posen, and A. Romanenko. Deepest sensitivity to wavelike dark photon dark matter with superconducting radio frequency cavities. *Phys. Rev. D*, 110:043022, Aug 2024.

[12] Kenny Fong, M. Laverty, Eric Chojnacki, Si Ping Wang, and Georg H. Hoffstaetter. Self excited operation for a 1.3 ghz 5-cell superconducting cavity. 2011.

[13] Slawomir Simbierowicz, Visa Vesterinen, Joshua Milem, Aleksi Lintunen, Mika Oksanen, Leif Roschier, Leif Grönberg, Juha Hassel, David Gunnarsson, and Russell E. Lake. Characterizing cryogenic amplifiers with a matched temperature-variable noise source. *Review of Scientific Instruments*, 92(3), March 2021.

[14] Asher Berlin, Raffaele Tito D'Agnolo, Sebastian A. R. Ellis, Christopher Nantista, Jeffrey Neilson, Philip Schuster, Sami Tantawi, Natalia Toro, and Kevin Zhou. Axion dark matter detection by superconducting resonant frequency conversion. *Journal of High Energy Physics*, 2020(7), July 2020.

[15] Bianca Giaccone, Asher Berlin, Ivan Gonin, Anna Grassellino, Roni Harnik, Yonatan Kahn, Timergali Khabiboulline, Andrei Lunin, Oleksandr Melnychuk, Alexander Netepenko, Roman Pilipenko, Yuriy Pischalnikov, Sam Posen, Oleg Pronitchev, Alex Romanenko, and Vyacheslav Yakovlev. Design of axion and axion dark matter searches based on ultra high q srf cavities, 2022.

[16] R. W. Schafer. Technical report no. hpl-2010-109, 2011.

[17] Jihn E. Kim. Weak-interaction singlet and strong CP invariance. *Phys. Rev. Lett.*, 43:103–107, Jul 1979.

5 Appendix

5.1 code

5.1.1 main

```python
import sys
import glob
from analysis import models, const, spectrum_analysis, phys_units
from plots import plotting
from spectral_data import spectral_data
from char_data import char_data
from experiment_structure import tuning_step, exp_run


file_patterns = ['simdata/*_1.4100G_*Step*span4000_BW0.1*100.dat']
names  = []
# List all files matching the pattern
for pattern in file_patterns:
    names.extend(glob.glob(pattern))
#print(names)
print(len(names))

ref_m_a_value = 1e6

try:
    run4k = exp_run(names, ["dummyCharFile.csv"], "dummyTheoreticalFile.csv",
    FIXED_m_a = ref_m_a_value)
except Exception as a:
    print(a)
try:
    run4k.filter_bins()
    #run4k.plot_excess_spectra(500,5,250,5)
    run4k.plot_excess_spectra()
except Exception as a:
    print(a)

run4k.verify_gaussian_distribution('excess')
run4k.select_ABW(ABW = 5e2)

run4k.rescaled_spectra()
run4k.compute_expected_sensitivity(SNR=3)
try:
    run4k.ax_lineshape()
    run4k.grand_spectrum()
    for i in range(len(run4k.comb_relevant)):
        print(run4k.comb_relevant.iloc[i])
except Exception as a:
    print(a)
```

5.1.2 structural classes

```
from spectral_data import spectral_data
from char_data import char_data
import pandas as pd
from analysis import spectrum_analysis, const
from plots import plotting
import matplotlib.pyplot as plt
from scipy import constants
from analysis import models, phys_units
import numpy as np
from uncertainties import ufloat, nominal_value
from scipy.spatial import cKDTree


class tuning_step(spectral_data, char_data):
    def __init__(self, file_Spec, file_Char, m_a):
        spectral_data.__init__(self, file_Spec)
        char_data.__init__(self, file_Char)
        self.f_pump = self.f_pump - (self.fc - self.f_sig)
        self.f_sig = self.fc
        self.m_a = m_a
        self.m_a_eV = self.m_a * const.hbar # eV

        #we changed this value to the fc value, this would be temporary, just
    for the simulation stage

    def __repr__(self):
        return f"tuning step: \nfreq={self.fc} \npower={self.spectrum})"

    def rescale_ax_pow(self, eta, V):
        #it could be computed for each axion mass value but maybe it's too
    difficult, there could be problems while redistributing the bins at
    combination step, better use a unique value for all exp run!
        P_ax = self.P_ax(eta, V, self.m_a)
        print(f'axion power: {P_ax:e}')
        wid = self.f_sig / self.Q_sig / 2
        #print(wid)
        Lor_cav_response = models.Lorentzian(self.freq, 1, self.f_sig, wid)
        norm = nominal_value( sum (Lor_cav_response) )
        Lor_cav_response = Lor_cav_response / norm

        self.spectrum['rescaled'] = self.spectrum['excess'] / P_ax /
    Lor_cav_response

    def rescale_noise(self):
        T_noise = constants.k * self.T_eq * self.RBW
        print(f'thermal and amp noise: {T_noise:e}')
        self.spectrum['rescaled'] = self.spectrum['rescaled'] * T_noise
        plotting.power_plot(self.freq, self.spectrum['rescaled'], 'frequency [Hz
]', 'rescaled excess power [-]', f'{self.fc/1e9:0.9f}GHz_rescaledPower.png',
 error = True)


    def compute_expected_sensitivity(self, SNR, eta, V): #check unit of k
        g_agg_ref = const.g_agg(self.m_a_eV)
        print(f'g_agg_ref: {g_agg_ref:e}')
        self.g2_agg = g_agg_ref **2 * SNR * constants.k * self.T_eq * np.sqrt(
self.RBW / self.t_int) / self.P_ax(eta, V, self.m_a)
        print('$g_agg$' + f'= {np.sqrt(self.g2_agg.n)} for tuning step: {self.fc
:0.9f}')
        return self.g2_agg


    def rescale_x_ax_mass(self):
        self.spectrum['f_ax'] = self.freq.copy() - self.f_pump # in Hz



class sim_theory_data:
    def __init__(self, file):
        data_raw = pd.read_csv(file, sep=";", names = ['magnitude', 'value' ,'
unit'], dtype = {'magnitude':str, 'value':float, 'unit': str}).set_index('
magnitude')
        self.store_params(data_raw)

    def store_params(self, data):
        self.form_fac = data.loc['eta_10', 'value']
        self.vol_cav = data.loc['V', 'value']
        self.eps_1d = data.loc['eps_1d', 'value']


class exp_run(sim_theory_data):
    def __init__(self, file_spec_list, file_char_list, file_theory, FIXED_m_a):
        # Check if file_list2 is long enough to handle the number of chunks
        #if len(file_char_list) < (len(file_spec_list) + 9) // 10:
        #    raise ValueError("file_list2 must be long enough to handle all
steps.")
        try:
            self.tuning_steps_series = self.create_tuning_steps(file_spec_list,
file_char_list, m_a = FIXED_m_a)
            sim_theory_data.__init__(self,file_theory)
        except Exception as a:
            #print(a)
            raise Exception(a)

        #if True is disabled
        self.IF = False

    def create_tuning_steps(self, file_list1, file_list2, m_a):
        tuning_steps = []

        for i, file1 in enumerate(file_list1):
            # Determine which file2 to use based on the step
            file2 = file_list2[0]
            #for now we only have the one file with the simulated parameters [i
// 10]  # Change file2 every 10 steps
            tuning_steps.append(tuning_step(file1, file2, m_a))

        return pd.Series(tuning_steps)

    def compute_expected_sensitivity(self, SNR):
        return self.tuning_steps_series.apply(lambda obj: obj.
compute_expected_sensitivity(SNR, self.form_fac, self.vol_cav))

    def plot_excess_spectra(self, WIF = 0 , POIF = 0, WRF = 0 , PORF = 0):
        if not self.IF: #because we decided to perform IF baseline removal
            print("IF analysis on")
            self.init_IF_ana()
        #print(self.IF_spectrum['baseline'])
        #spectrum_analysis.IFbaseline_extraction(self, WIF, POIF)

        #print('over with IF')
        self.tuning_steps_series.apply(lambda obj: spectrum_analysis.calc_excess
(obj, WIF, POIF, WRF, PORF, self.IF_spectrum['baseline']))

    def verify_gaussian_distribution(self, powtype):
        self.tuning_steps_series.apply(lambda obj: obj.verify_gaus(powtype))
        self.tuning_steps_series.apply(lambda obj: obj.verify_skewnorm(powtype))

    def filter_bins(self):

        if not self.IF:
            print("IF analysis on")
            self.init_IF_ana()

        self.tuning_steps_series.apply(lambda obj: obj.spec_filtering(self.
idx_IF_interf))

    def init_IF_ana(self):
        self.IF = True

        self.IF_spectrum = pd.DataFrame()
        self.idx_IF_interf = spectrum_analysis.IFinterferences(self)


        self.IF_spectrum = self.IF_spectrum.copy().iloc[self.idx_IF_interf]
        self.IF_spectrum['baseline'] = self.IF_spectrum['IFmean'] * 0


        #extract the discarded bins
        self.IF_damaged_bins = list(set(list(range(len(self.IF_spectrum)))).
symmetric_difference(set(self.idx_IF_interf)))
        print('list of excluded bins due to IF interferences')
        print(f'{self.IF_damaged_bins}' )
        #for i in range(len(self.IF_damaged_bins)):
        #    print(f' {self.IF_damaged_bins[i]}' )
        print(f'number of bins excluded {len(self.IF_damaged_bins)}')



    def select_ABW(self, ABW):
        self.tuning_steps_series.apply(lambda obj: obj.select_ABW(ABW))

    def rescaled_spectra(self):
        self.tuning_steps_series.apply(lambda obj: obj.rescale_ax_pow(self.
form_fac, self.vol_cav))
        self.tuning_steps_series.apply(lambda obj: obj.rescale_noise())

    def grand_spectrum(self):
        self.mass_column = np.arange(1e6-2e4, 1e6+2e4, 0.2)
        self.combined = pd.DataFrame({'f_ax': self.mass_column})
        self.combine_spectra()

        #print(self.combined)

        # Apply the function row-wise across the DataFrame
        self.combined['weighted_average'] = self.combined.copy().apply(
    phys_units.weighted_sum_ufloats, axis=1)
        #select rows that end up having a non NaN computed value in the weighted
     average column
        self.combined = self.combined[~self.combined['weighted_average'].isna()]

        self.comb_relevant = self.combined[['f_ax', 'weighted_average']]
        self.comb_relevant = self.comb_relevant[(self.combined.f_ax < 1e6+6e2) &
     (self.combined.f_ax > 1e6-0.05e2)]

        plotting.power_plot(self.comb_relevant['f_ax'], self.comb_relevant['
    weighted_average'],
                            'axion mass [Hz]', 'combined excess power [-]',
                            filename = 'grandspectrum.png', error = True)

        #print(self.combined[['f_ax','weighted_sum']])


    def combine_spectra(self):
        for i, step in enumerate(self.tuning_steps_series):
            # Get the masses and powers from the current step
            #here we don't take into account errors in the masses
            step_masses = step.spectrum['f_ax'].apply(lambda x: x.nominal_value)
    .values
            step_powers = step.spectrum['convolved'].values

            # Find the closest mass in the predefined mass_column
            tree = cKDTree(self.mass_column[:, None])
            _, closest_indices = tree.query(step_masses[:, None])

            # Create a power column for this step
            power_column = np.full(len(self.mass_column), np.nan, dtype=object)
            power_column[closest_indices] = step_powers

            # Add the power column to the result DataFrame
            self.combined = self.combined.copy()
            self.combined[f'power_step_{i}'] = power_column


    def ax_lineshape(self):
        self.tuning_steps_series.apply(lambda obj: obj.rescale_x_ax_mass())
        self.tuning_steps_series.apply(lambda obj: obj.convolved_spectra())

    def __repr__(self):
        return f"exp_run(tuning_steps_series={self.tuning_steps_series})"
```

```
import dask.dataframe as dd
import pandas as pd
import numpy as np
from analysis import spectrum_analysis, models
import matplotlib.pyplot as plt
from uncertainties.umath import sqrt
from uncertainties import ufloat, nominal_value, std_dev
from plots import plotting

class spectral_data:
    def __init__(self, file):
        data_raw_dd = dd.read_csv(file, sep=";",
                                  usecols = [0,1], names = ['freq', 'power'])

```

```python
        data_raw = data_raw_dd.compute().iloc[31:].astype('float64')
        params = data_raw_dd.compute().iloc[:31]

        try:
            self.init_params(params, len(data_raw))
        except ImportError:
            #print()
            raise ImportError(f"computed sample rate doesn't make sense. \n see file {file}")
        except ValueError as N:
            #print()
            raise ValueError(f'no match in number of data points. \nexpected {len(data_raw)}, read from file {N}')

        try:
            self.create_spectrum(data_raw)
        except TypeError as typeS:
            #print(typeS)
            raise TypeError(typeS)

    def init_params(self, p, N_points):
        self.fc = float(p.iloc[6,1])
        self.f_range = [ float(p.iloc[8,1]) , float(p.iloc[9,1]) ]
        self.Max_ABW = self.f_range[1] - self.f_range[0]
        if self.Max_ABW != float(p.iloc[16,1]):
            raise ImportError
        self.RBW = float(p.iloc[14,1])
        self.Avg = float(p.iloc[19,1])

        self.N_bins = int(p.iloc[30,1])
        if self.N_bins != N_points:
            raise ValueError(self.N_bins)
        self.bin_w = float(p.iloc[15,1]) / self.N_bins
        self.pow_scale_raw = p.iloc[24,1]

        self.t_int = self.Avg / self.RBW


        self.counts = {}
        self.bins = {}
        self.chi = pd.Series()

    def create_spectrum(self, data):

        freq = data['freq'].copy()
        #select frequencies within ABW, short of 2*500 bins per side for filtering reasons
        ABW_idx = np.where( (freq>self.f_range[0] + 2*500*self.bin_w)
                            & (freq<self.f_range[1]-2*500*self.bin_w) )
        self.freq = freq.iloc[ABW_idx]

        self.spectrum = pd.DataFrame()

        if self.pow_scale_raw == 'LOG':
            self.spectrum['raw_dBm'] = data['power'].copy().iloc[ABW_idx]
            self.spectrum['raw_mW'] = self.spectrum['raw_dBm'].apply(spectrum_analysis.convert_pow_scale, scale = self.pow_scale_raw)
        else:
            if self.pow_scale_raw == 'LIN':
                self.spectrum['raw_mW'] = data['power'].copy().iloc[ABW_idx]
                try:
                    self.spectrum['raw_dBm'] = self.spectrum['raw_mW'].apply(spectrum_analysis.convert_pow_scale, scale = self.pow_scale_raw)
                except TypeError as typeS:
                    #print(typeS)
                    raise TypeError(f"invalid scale: {self.pow_scale_raw}")
            else:
                raise TypeError(f"invalid scale: {self.pow_scale_raw}")
        self.average_lin_pow = self.spectrum['raw_mW'].mean()

    def spec_filtering(self, idx, ax = 1):
        if ax == 0:
            self.spectrum = self.spectrum.copy().iloc[:, idx]
            self.freq = self.freq.copy().iloc[:, idx]
        if ax == 1:
            self.spectrum = self.spectrum.copy().iloc[idx]
            self.freq = self.freq.copy().iloc[idx]

    def select_ABW(self, ABW):
        i_ABW = np.where((self.freq < self.fc + ABW/2) & (self.freq > self.fc -
ABW/2))[0]
        self.spec_filtering(i_ABW)

    def verify_gaus(self, powertype):
        addtoplot = True
        self.counts, self.bins, par, perr, cov = spectrum_analysis.verify_gaus(
self.spectrum, powertype, self.fc, error = True, addtoplot = addtoplot)

        self.chi[powertype] = models.Chi_squared_gaus(self.counts[self.counts
>0], self.counts[self.counts>0], self.bins[self.counts>0], par)

        cov_matrix_formatted = np.vectorize(lambda x: f"{x:.3g}")(cov)

        if addtoplot:
            with pd.option_context('display.float_format', '{:.2f}'.format):
                plt.text(2.5*par[2], max(self.counts) * 8 / 10, f'$\chi^2$: {
self.chi[powertype]:0.5f}\n'
                        + '$\sigma_{gaus}$ : ' + f'{par[2]:0.3f} $\pm$ {perr
[2]:0.3f}\n'
                        + '$f_{0}$ : ' + f'{par[1]:0.3f} $\pm$ {perr[1]:0.3f}\n
'
                        #+ '$cov_{mat}$ : ' + f'{cov_matrix_formatted}\n'
                        , horizontalalignment='center',
                        verticalalignment='center',fontsize = 'large')
            plt.legend(loc = 'best')
            plt.savefig(f'developFolder/img/{self.fc/1e9:0.9f}GHz_{powertype}
_Hist.png', dpi=300, bbox_inches='tight')
            plt.close()
        else:
            print(f'{self.fc} step has chi2: {self.chi[powertype]}')



    def verify_skewnorm(self, powertype):
        addtoplot = True
        self.counts, self.bins, par, perr, cov = spectrum_analysis.
verify_skewnorm(self.spectrum, powertype, self.fc, error = True, addtoplot =
 addtoplot)

        self.chi[powertype + '_skewed'] = models.Chi_squared_skewed_norm(self.
counts[self.counts>0], self.counts[self.counts>0], self.bins[self.counts>0],
 par)

        cov_matrix_formatted = np.vectorize(lambda x: f"{x:.3g}")(cov)

        if addtoplot:
            plt.text(2.5*par[2], max(self.counts) * 8 / 10,
                            f'$\chi^2$: {self.chi[powertype + '_skewed']:0.5f}\n'
                            + '$alpha$ : ' + f'{par[3]:0.3f}$\pm$ {perr[3]:0.3f}\n'
                            + '$loc$ : ' + f'{par[1]:0.3f}$\pm$ {perr[1]:0.3f}\n'
                            + '$scale$ : ' + f'{par[2]:0.3f}$\pm$ {perr[2]:0.3f}\n'
                            #+ '$cov_{mat}$ : ' + f'{cov_matrix_formatted}\n',
                            ,horizontalalignment='center',
                            verticalalignment='center',fontsize = 'large')
            plt.legend(loc = 'best')
            plt.savefig(f'developFolder/img/{self.fc/1e9:0.9f}GHz_{powertype}
    _Hist_SkewedNorm.png', dpi=300, bbox_inches='tight')
            plt.close()
        else:
            print(f'{self.fc} step has chi2: {self.chi[powertype]}')


    def convolved_spectra(self):
        def kernel(data, x_0):
            #print(x_0)
            k = data.apply(lambda x:  models.MB_ax(x.n, x_0.n))
            return k / sum(k)

        def A(row_new):
            spec = pd.DataFrame( {'rescaled': self.spectrum.rescaled, 'f_ax' :
    self.spectrum.f_ax })


            spec = spec[spec.f_ax > row_new.f_ax]
            #print(len(spec))
            end = min(len(spec), 25)
            if end == 0:
                return ufloat(np.nan, np.nan)
            spec = spec.copy().iloc[:end]

            spec['k'] = kernel(spec.f_ax, row_new.f_ax)

            #print(spec)
            A_num = sum (spec.apply(lambda row: nominal_value( row.rescaled ) *
    row.k  / std_dev(row.rescaled) **2, axis = 1))
            sigma_2 = sum (spec.apply(lambda row: 0.5 *  (row.k  / row.rescaled.
    s ) **2 , axis = 1)) **-1
            #print(sigma_2)
            sigma = np.sqrt(sigma_2)
            del spec
            A =  ufloat( A_num * 2 * sigma_2, sigma)
            #print(A)
            if type(A).__name__ != 'Variable':
                return ufloat(np.nan, np.nan)
            return A

        #A_list = []
        #for i in range(len(self.spectrum)):
        #    A_list.append(A(self.spectrum.iloc[i]))

        #print(A_list)
        self.spectrum['convolved'] = self.spectrum.apply(lambda row: A(row),
    axis=1)

        # Check the types and values before assignment
        #print("Convolved Values:")
        #print(self.spectrum['convolved'])

        plotting.power_plot(self.spectrum.f_ax, self.spectrum['convolved'], '
    frequency [Hz]', 'convolved excess power [-]', f'{self.fc/1e9:0.9f}GHz_convolvedspectrum.png', error = True)
```

```
import pandas as pd
from analysis import const
from scipy import constants
from uncertainties import ufloat
from uncertainties.umath import *
import numpy as np
class char_data:
    def __init__(self, file):
        data_raw = pd.read_csv(file, sep=";", names = ['magnitude', 'value', 'error' ,'unit'], dtype = {'magnitude':str, 'value':float, 'error':float, 'unit': str}).set_index('magnitude')
        self.store_params(data_raw)

    def store_params(self, data):
        self.T_add = ufloat ( data.loc['T_add', 'value'] , data.loc['T_add', 'error'] )
        self.G_rc = ufloat ( data.loc['G_rc', 'value'] , data.loc['G_rc', 'error'] )
        self.b_yfac = ufloat ( data.loc['bw', 'value'] , data.loc['bw', 'error'] )
        self.eta_mic = ufloat ( data.loc['eta', 'value'] , data.loc['eta', 'error'] )
        self.f_pump = ufloat ( data.loc['f_0', 'value'] , data.loc['f_0', 'error'] )
        self.Q_pump = ufloat ( data.loc['Q_0', 'value'] , data.loc['Q_0', 'error'] )
        self.beta_pump = ufloat ( data.loc['beta_0', 'value'] , data.loc['beta_0', 'error'] )
        self.f_sig = ufloat ( data.loc['f_1', 'value'] , data.loc['f_1', 'error'] )
        self.Q_sig = ufloat ( data.loc['Q_1', 'value'] , data.loc['Q_1', 'error'] )
        self.beta_sig = ufloat ( data.loc['beta_1', 'value'] , data.loc['beta_1', 'error'] )
        self.B_0 = ufloat ( data.loc['B_0', 'value'] , data.loc['B_0', 'error'] )
        self.T_cav = ufloat ( data.loc['T_cav', 'value'] , data.loc['T_cav', 'error'] )

        #one cannot define an axion mass for each tuning step!! There's always a range of masses to which each tuning step is sensitive
        #self.m_a = abs(self.f_pump - self.f_sig)
        #self.m_a_eV = self.m_a * const.hbar # eV
        self.T_eq = self.T_add + self.T_cav

    def P_ax(self, form_fac, V, m_a):
        g_agg_val = const.g_agg(m_a * const.hbar)
        #print(f'gagg: {g_agg_val:e}')
        return 0.25e7 * np.pi * const.rho_DM * (form_fac * self.B_0 * g_agg_val)**2 * V * const.Q_a * self.beta_sig / m_a / (self.beta_sig + 1)
```

5.1.3 custom libraries

```
from scipy import constants
import numpy as np
from plots import plotting
import matplotlib.pyplot as plt
import statistics as stat
import scipy
from scipy.stats import skewnorm
import pandas as pd
```

```python
import dask.dataframe as dd
from scipy.signal import savgol_filter as SG
from uncertainties import ufloat


class models:
    def Gaussian(x, A, x0, sigma):
        return A * np.exp(-(x - x0) ** 2 / (2 * sigma ** 2))

    def Skenorm(x, A, x0, scale, alpha):
        return A * skewnorm.pdf(x, alpha, loc = x0, scale = scale )

    def Lorentzian(x, amp, cen, wid):
        return amp*wid**2/((x-cen)**2+wid**2)

    def MB_ax(x,f_a):
        if x < f_a:
            return 0
        else:
            return 2 * np.sqrt((x-f_a)/np.pi) * (3* const.v2_ax / f_a /
    constants.c **2 ) **(3/2) * np.exp(-3 * (x - f_a) * constants.c **2 / f_a /
    const.v2_ax)

    def Chi_squared_gaus(y, w2, x, par):
        DoF = len(x) - len(par)
        #print(w2)
        #print(DoF)
        return sum( (y - models.Gaussian(x, *par)) **2 / w2) / DoF

    def Chi_squared_skewed_norm(y, w2, x, par):
        DoF = len(x) - len(par)
        #print(w2)
        #print(DoF)
        return sum( (y - models.Skenorm(x, *par)) **2 / w2) / DoF


class const:
    rho_DM = 0.45e3 # GeV / dm-3
    Q_a = 1e6 #?
    Lambda = 77.6 # MeV
    g_gamma = -0.97
    hbar = constants.hbar / constants.eV # eV/Hz
    mu_0 = constants.mu_0 # T2 * m3 / J
    k = constants.k / constants.eV # eV/K
    m_pi = 135 # MeV
    f_pi = 93 # MeV
    v2_ax = 270e3 **2 # m/s  [rms velocity of the dark matter halo]

    def g_agg (m_a_eV):
        return const.g_gamma * m_a_eV *1e-3 * constants.alpha / np.pi / const.
    m_pi / const.f_pi

class phys_units:
    def p_tomW(x):
        return 10 ** (x/10)

    # Function to compute the weighted sum across rows
    def weighted_sum_ufloats(row):

        # Filter out NaN values directly, whether they are in ufloat or plain
    NaNs
        ufloat_values = [u for u in row if not isinstance(u, float)]

        # If no valid values remain
        if not ufloat_values:
            #print("created a row with no values!!!")
            return np.nan

        # For valid ufloat values, proceed with the weighted sum calculation
        weights = [1 / (u.s**2) for u in ufloat_values]

        # Compute the weighted sum of nominal values
        weighted_nominal_sum = sum(u.n * w for u, w in zip(ufloat_values, weights))

        # Compute the total weight
        total_weight = sum(weights)

        # Calculate the weighted mean nominal value
        weighted_mean_nominal = weighted_nominal_sum / total_weight

        # Calculate the uncertainty (standard deviation) of the weighted sum
        weighted_std = np.sqrt(1 / total_weight)

        return ufloat(weighted_mean_nominal, weighted_std)



class spectrum_analysis:

    def convert_pow_scale(x, scale):
        if(scale == 'LOG'):
            return 10 ** (x/10)
        if(scale == 'LIN'):
            return 10 * np.log10(x)
        else:
            raise TypeError(f"invalid scale: {scale}")

    def calc_excess(obj, WIF, POIF, WRF, PORF, IFbase = []):
        plotting.power_plot(obj.freq, obj.spectrum['raw_mW'], 'frequency [Hz]', 'power [mW]', f'{obj.fc/1e9:0.9f}GHz_rawPower.png')
        spectrum_analysis.baseline_removal_average(obj)
        #change to another method if Gain variations are to be taken into account and any filter is to be applied to the data

        #spectrum_analysis.IFbaseline_removal(obj, IFbase)
        #RFbaseline = spectrum_analysis.RFbaseline_removal(obj, WRF, PORF)
        plotting.power_plot(obj.freq, obj.spectrum['excess'], 'frequency [Hz]', 'excess power [-]', incomplete = True)



        #plt.plot(obj.freq, RFbaseline, 'b', ms = 2, label = 'RF gain variations')
        plt.legend(loc = 'best')
        plt.savefig(f'developFolder/img/{obj.fc/1e9:0.9f}GHz_normalizedPower_baseline.png', dpi=300, bbox_inches='tight')
        plt.close()

        obj.spectrum['excess'] = obj.spectrum['excess'] - 1
        err = obj.spectrum['excess'].std()
        print(f'excess noise std: {err}')
        obj.spectrum['excess'] = obj.spectrum['excess'].copy().apply(lambda x: ufloat(x, err))
        plotting.power_plot(obj.freq, obj.spectrum['excess'], 'frequency [Hz]', 'excess power [-]', f'{obj.fc/1e9:0.9f}GHz_excessPower.png', error = True)



    def baseline_removal_average(obj):
        obj.spectrum['excess'] = obj.spectrum['raw_mW'].copy() / obj.average_lin_pow

    def IFbaseline_removal(obj, IFbase):
        if len(obj.spectrum) == len(IFbase):
            obj.spectrum['excess'] = obj.spectrum['raw_mW'].copy() / IFbase
        else:
            raise ValueError(f'IF baseline doesnt match data length: should be {len(obj.spectrum)}, insteas it is {len(IFbase)}')

    def RFbaseline_removal(obj, W, P0):
        RFbase = spectrum_analysis.RFbaseline_extraction(obj, W, P0)
        obj.spectrum['excess'] = obj.spectrum['excess'].copy() / RFbase
        return RFbase

    def verify_gaus(spectrum, powertype, fc, error = False, addtoplot = False):
        if error:
            spectral_values = spectrum[powertype].apply(lambda x: x.n)
            AMP = spectrum[powertype].apply(lambda x: x.s).iloc[0]
        else:
            spectral_values = spectrum[powertype]
            AMP = stat.stdev(spectral_values)

        if powertype == 'excess':
            counts, bins, bars = plotting.hist_plot(spectral_values/AMP,f'{powertype} power [-]', incomplete = True)
            startAMP = 1
        else:
            counts, bins, bars = plotting.hist_plot(spectral_values,f'{powertype} power [-]', incomplete = True)
            startAMP = AMP

        #shift bins to be centered
        nbins = len(bins) - 1
        binw = bins[1] - bins[0]
        bins = bins + binw/2
        #delete last one
        bins = bins[:nbins]

        idx_max = np.argmax(counts)

        MAX = counts[idx_max]
        CENTRE = bins[idx_max]

        filtering_idx = np.where( (bins < CENTRE + 3 * startAMP ) & (bins > CENTRE - 3 * startAMP ) )

        filt_bins = bins[filtering_idx]
        filt_count = counts[filtering_idx]


        par, pcov = scipy.optimize.curve_fit(models.Gaussian,
                                             filt_bins, filt_count,
                                             p0=[MAX,CENTRE,startAMP])
        perr = np.sqrt(np.diag(pcov))

        #print(par)
        plt.plot(filt_bins, models.Gaussian(filt_bins, *par),
                 'r', ms = 2, label = 'fit curve')
        plt.legend(loc = 'best')
        plt.savefig(f'developFolder/img/{fc/1e9:0.9f}GHz_{powertype}_Hist_log.png', dpi=300, bbox_inches='tight')
        plt.close()

        if powertype == 'excess':
            plotting.hist_plot(spectral_values/AMP,f'{powertype} power [-]', incomplete = True, log = False)
        else:
            plotting.hist_plot(spectral_values,f'{powertype} power [-]', incomplete = True, log = False)
        plt.plot(filt_bins, models.Gaussian(filt_bins, *par),
                  'r', ms = 2, label = 'fit curve')
        if not addtoplot:

            plt.legend(loc = 'best')
            plt.savefig(f'developFolder/img/{fc/1e9:0.9f}GHz_{powertype}_Hist.png', dpi=300, bbox_inches='tight')
            plt.close()

        return counts, bins, par, perr, pcov

    def verify_skewnorm(spectrum, powertype, fc, error = False, addtoplot = False):
        if error:
            spectral_values = spectrum[powertype].apply(lambda x: x.n)
            AMP = spectrum[powertype].apply(lambda x: x.s).iloc[0]
        else:
            spectral_values = spectrum[powertype]
            AMP = stat.stdev(spectral_values)
        if powertype == 'excess':
            counts, bins, bars = plotting.hist_plot(spectral_values/AMP,f'{powertype} power [-]', incomplete = True)
            startAMP = 1
        else:
            counts, bins, bars = plotting.hist_plot(spectral_values,f'{powertype} power [-]', incomplete = True)
            startAMP = AMP

        #shift bins to be centered
        nbins = len(bins) - 1
        binw = bins[1] - bins[0]
        bins = bins + binw/2
        #delete last one
        bins = bins[:nbins]

        idx_max = np.argmax(counts)

        MAX = counts[idx_max]
        CENTRE = bins[idx_max]

        filtering_idx = np.where( (bins < CENTRE + 3) & (bins > CENTRE - 3) )

        filt_bins = bins[filtering_idx]
        filt_count = counts[filtering_idx]


        par, pcov = scipy.optimize.curve_fit(models.Skenorm,
                                             filt_bins, filt_count,
                                             p0=[MAX,CENTRE,startAMP, 1 ])
        perr = np.sqrt(np.diag(pcov))

        #print(par)
        plt.plot(bins, models.Skenorm(bins, *par),
                 'r', ms = 2, label = 'fit curve')
        plt.legend(loc = 'best')
        plt.savefig(f'developFolder/img/{fc/1e9:0.9f}GHz_{powertype}_Hist_SkewedNorm_log.png', dpi=300, bbox_inches='tight')
        plt.close()


        if powertype == 'excess':
            plotting.hist_plot(spectral_values/AMP,f'{powertype} power [-]', incomplete = True, log = False)
        else:
            plotting.hist_plot(spectral_values,f'{powertype} power [-]', incomplete = True, log = False)
        plt.plot(bins, models.Skenorm(bins, *par),
                 'r', ms = 2, label = 'fit curve')
        if not addtoplot:
            plt.legend(loc = 'best')
            plt.savefig(f'developFolder/img/{fc/1e9:0.9f}GHz_{powertype}_Hist_SkewedNorm.png', dpi=300, bbox_inches='tight')
            plt.close()

        return counts, bins, par, perr, pcov

    def IFinterferences(obj):

        power = pd.DataFrame()
        #power = dd.DataFrame()
        for i in range(len(obj.tuning_steps_series)):
            newcol = obj.tuning_steps_series[i].spectrum['raw_mW'].copy().rename(f'step{i:n}')
            power = pd.concat([power.copy(),newcol], axis = 1)

        #print(len(obj.tuning_steps_series))
        obj.IF_spectrum['IFmean'] = power.mean(axis = 1)
        #print(power.mean(axis = 1))

        c, b, MeanPar, MeanPerr, MeanCov = spectrum_analysis.verify_gaus(obj.IF_spectrum, 'IFmean', fc = 0)

        f_central = ufloat (MeanPar[1], MeanPerr[1])
        sigma = ufloat (MeanPar[2], MeanPerr[2])

        match len(power):
            case _ if len(power) < 1e4:
                tolerance = 4.5
            case _ if 1e4 <= len(power) < 1e5:
                tolerance = 5
            case _ if 1e5 <= len(power) <= 1e6:
                tolerance = 5.5
            case _ if 1e6 <= len(power) <= 1e7:
                tolerance = 6
            case _:
                print("sample size out of range (too large), go check!")

        return np.where((obj.IF_spectrum['IFmean'] < f_central.n + tolerance * sigma.n) & (obj.IF_spectrum['IFmean'] > f_central.n - tolerance * sigma.n))[0]

    def IFbaseline_extraction(obj, HWL, PO):
        obj.IF_spectrum['baseline'] = SG(obj.IF_spectrum['IFmean'],
                                         window_length= 2*HWL+1, polyorder = PO,
                                          mode = 'nearest')
```

```python


        plotting.power_plot(obj.idx_IF_interf, obj.IF_spectrum['IFmean'], 'IF bins', 'average IF response (mW)', incomplete = True)
        plt.plot(obj.idx_IF_interf, obj.IF_spectrum['baseline'], 'b', ms = 1, label = 'IF baseline')
        plt.legend(loc = 'best')
        plt.savefig('developFolder/img/IFInterferences_mean_baseline.png', dpi=300, bbox_inches='tight')
        plt.close()

        return obj.IF_spectrum['baseline']

    def RFbaseline_extraction(obj, HWL, PO):
        return SG(obj.spectrum['excess'],
              window_length= 2*HWL+1, polyorder = PO,
             mode = 'nearest')
```

```python
import matplotlib.pyplot as plt
from uncertainties import ufloat
from uncertainties.unumpy import nominal_values, std_devs
import numpy as np

class plotting:
    def power_plot(x, y, x_lab, y_lab, filename = 'dummyname.png', new = True, incomplete = False, error = False):
        if new:
            plt.figure()

        if error:
            x_val = nominal_values(x)
            y_val = nominal_values(y)
            y_err = std_devs(y)
            #plt.errorbar(x_val, y_val, yerr = y_err,
            #             marker = 'o', ms = 0.5, color = 'yellow',
            #             label = "errors")
            plt.plot(x_val, y_val,
                     'go', ms = 1,
                     label = "data")
        else:
            plt.plot(x, y,
                     'go', ms = 1,
                     label = "data")
        plt.xlabel(x_lab, fontsize='x-large')
        plt.ylabel(y_lab, fontsize='x-large')

        if not incomplete:
            #plt.legend(loc = 'best')
            plt.savefig('developFolder/img/'+filename, dpi=300, bbox_inches='tight')
            plt.close()

    def hist_plot(x, x_lab, filename = 'dummyname.png', new = True, incomplete = False, log = True):
        if new:
            plt.figure()
        if log:
            counts, bins, bars = plt.hist(x, bins = len(x)//400, log = True, label = "data")
        else:
            counts, bins, bars = plt.hist(x, bins = len(x)//400, label = "data")


        bin_centers = 0.5 * (bins[1:] + bins[:-1])

        # Errors (e.g., Poisson errors: sqrt of the counts)
        errors = np.sqrt(counts)

        # Overlay the error bars
        plt.errorbar(bin_centers, counts, yerr=errors, fmt='o', color='C1', label='errors', ms = 2)
        plt.xlabel(x_lab, fontsize='xx-large')
        plt.ylabel('counts', fontsize='xx-large')
        if not incomplete:
            plt.legend(loc = 'best')
            plt.savefig('developFolder/img/'+filename, dpi=300, bbox_inches='tight')
            plt.close()
        return counts, bins, bars
```