Use this document as a template

My PhD Thesis

Customise this page according to your needs

Tobias Hangleiter*

April 10, 2025

* A LaTeX lover/hater

The kaobook class

Disclaimer
You can edit this page to suit your needs. For instance, here we have a no copyright statement, a colophon and some other information. This page is based on the corresponding page of Ken Arroyo Ohori's thesis, with minimal changes.

Colophon
This document was typeset with the help of KOMA-Script and LaTeX using the kaobook class.
The source code of this book is available at:
https://github.com/fmarotta/kaobook
(You are welcome to contribute!)

Publisher
First printed in May 2019 by

The harmony of the world is made manifest in Form and Number, and the heart and soul and all the poetry of Natural Philosophy are embodied in the concept of mathematical beauty.

– D'Arcy Wentworth Thompson

Contents

Part I

A Flexible Python Tool For Fourier-Transform Noise Spectroscopy

1 Introduction

Noise is ubiquitous in condensed matter physics experiments, and in mesoscopic systems in particular it can easily drown out the sought-after signal. Hence, characterizing (and subsequently mitigating) noise is an essential task for the experimentalist. But noise comes in as many different forms as there are types of signal sources and detectors, whether it be a voltage source or a photodetector, and while some instruments have built-in solutions for noise analysis, they vary in functionality and capability. Moreover, the measured signal often does not directly correspond to the noisy physical quantity of interest, making it desirable to be able to manipulate the raw data before processing.

Theory of spectral noise estimation

2

THERE exists a multitude of methods for estimating noise properties.

lay out some others

If the noisy process $x(t)$[1] has Gaussian statistics, meaning that the value at a given point in time follows a normal distribution with some mean μ and variance σ^2 over multiple realizations of the process, it can be fully described by the power spectral density (PSD) $S(\omega)$.[2] For the purpose of noise estimation, the assumption of Gaussianity is a rather weak one as the noise typically arises from a large ensemble of individual fluctuators and is therefore well approximated by a Gaussian distribution by the central limit theorem.[3] Even if the process $x(t)$ is not perfectly Gaussian, non-Gaussian contributions can be seen as higher-order contributions if viewed from the perspective of perturbation theory, and therefore the PSD still captures a significant part of the statistical properties. For this reason, the PSD is the central quantity of interest in noise spectroscopy and I will discuss some of its properties in the following.

1: We discuss only classical noise here, meaning $x(t)$ commutes with itself at all times. For descriptions of and spectroscopy protocols for quantum noise refer to Refs. 1 and 2, for example.

2: The term *power spectrum* is often used interchangably. I will do so as well, but emphasize at this point that in digital signal processing in particular, the *spectrum* is a different quantity from the *spectral density*. See also sidenote 14 in Chapter 3.

maybe a classical signal processing ref?

3: As an example, consider electronic devices, where voltage noise arises from a large number of defects and other charge traps in oxides being populated and depopulated at certain rates γ. The ensemble average over these so-called two-level fluctuators (TLFs) then yields the well-known $1/f$-like noise spectra (at least for a large density [3]).

flesh out this sidenote?

For real signals $x(t) \in \mathbb{R}$, $S(\omega)$ is an even function and one therefore distinguishes the *two-sided* PSD $S^{(2)}(\omega)$ defined over $\mathbb{R}$ from the *one-sided* PSD $S^{(1)}(\omega) = 2S^{(2)}(\omega)$ defined only over $\mathbb{R}^+$. Complex signals $x(t) \in \mathbb{C}$ such as those generated by Lock-in amplifiers after demodulation in turn have asymmetric, two-sided PSDs.

2.1 Spectrum estimation from time series

flesh out

To see how the PSD may be estimated from time-series data, consider a continuous wide-sense stationary[4] signal in the time domain $x(t) \in \mathbb{C}$ that is observed for some time T. We define the windowed Fourier transform of $x(t)$ and its inverse by[5]

$$\hat{x}_T(\omega) = \int_0^T \mathrm{d}t\, x(t) \mathrm{e}^{-\mathrm{i}\omega t} \tag{2.1}$$

$$\text{and} \quad x(t) = \int_{-\infty}^{\infty} \frac{\mathrm{d}\omega}{2\pi} \hat{x}_T(\omega) \mathrm{e}^{\mathrm{i}\omega t}, \tag{2.2}$$

i. e., we assume that outside of the window of observation $x(t)$ is zero. The auto-correlation function of $x(t)$ is given by

$$C(\tau) = \langle x(t)^* x(t+\tau) \rangle \tag{2.3}$$

$$= \lim_{T\to\infty} \frac{1}{T} \int_0^T \mathrm{d}t\, x(t)^* x(t+\tau), \tag{2.4}$$

where $\langle \cdot \rangle$ is the ensemble average over multiple realizations of the process and the last equality holds true for ergodic processes. Expressing $x(t)$ in terms of its Fourier representation (Equation 2.1) and reordering the integrals, we get[6]

4: For a wide-sense stationary (also called weakly stationary) process $x(t)$, the mean is constant and the auto-correlation function $C(t,t') = \langle x(t)^* x(t') \rangle$ is given by $\langle x(t)^* x(t+\tau) \rangle = \langle x(0)^* x(\tau) \rangle$ with $\tau = t' - t$. That is, it is a function of only the time lag τ and not the absolute point in time. For Gaussian processes as discussed here, this also implies stationarity [4]. The property further implies that $C(\tau)$ is an even function.

sketch of auto-correlation function?

5: In this chapter we will always denote the Fourier transform of some quantity ξ using the same symbol with a hat, $\hat{\xi}$.

6: Mathematicians might at this point argue the integrability of $x(t)$, but as we deal with physical processes with finite bandwidth (and have no shame), we do not.

$$C(\tau) = \lim_{T\to\infty} \frac{1}{T}\int_0^T \mathrm{d}t \int_{-\infty}^{\infty} \frac{\mathrm{d}\omega}{2\pi}\hat{x}_T(\omega)^* \mathrm{e}^{-\mathrm{i}\omega t} \int_{-\infty}^{\infty} \frac{\mathrm{d}\omega'}{2\pi}\hat{x}_T(\omega')\mathrm{e}^{\mathrm{i}\omega'(t+\tau)} \tag{2.5}$$

$$= \lim_{T\to\infty} \frac{1}{T}\int_{-\infty}^{\infty} \frac{\mathrm{d}\omega}{2\pi} \int_{-\infty}^{\infty} \frac{\mathrm{d}\omega'}{2\pi}\hat{x}_T(\omega)^*\hat{x}_T(\omega')\mathrm{e}^{\mathrm{i}\omega'\tau} \int_0^T \mathrm{d}t\, \mathrm{e}^{\mathrm{i}t(\omega'-\omega)} \tag{2.6}$$

7: Note that, because $x(t)$ is wide-sense stationary, we may shift the limits of integration $\int_0^T \to \int_{-T/2}^{+T/2}$.

The innermost integral approaches a δ-function for large T,[7] allowing us to further simplify this under the limit as

$$C(\tau) = \lim_{T\to\infty} \frac{1}{T}\int_{-\infty}^{\infty} \frac{\mathrm{d}\omega}{2\pi} \int_{-\infty}^{\infty} \frac{\mathrm{d}\omega'}{2\pi}\hat{x}_T(\omega)^*\hat{x}_T(\omega')\mathrm{e}^{\mathrm{i}\omega'\tau}\delta(\omega'-\omega) \tag{2.7}$$

$$= \lim_{T\to\infty} \frac{1}{T}\int_{-\infty}^{\infty} \frac{\mathrm{d}\omega}{2\pi}|\hat{x}_T(\omega)|^2\mathrm{e}^{\mathrm{i}\omega\tau} \tag{2.8}$$

$$= \int_{-\infty}^{\infty} \frac{\mathrm{d}\omega}{2\pi}S(\omega)\mathrm{e}^{\mathrm{i}\omega\tau} \tag{2.9}$$

with the PSD

$$S(\omega) = \lim_{T\to\infty} \frac{1}{T}|\hat{x}_T(\omega)|^2 \tag{2.10}$$

$$= \int_{-\infty}^{\infty} \mathrm{d}\tau C(\tau)\mathrm{e}^{-\mathrm{i}\omega\tau} \tag{2.11}$$

Equation 2.9 is the Wiener-Khinchin theorem that states that the autocorrelation function $C(\tau)$ and the PSD $S(\omega)$ are Fourier-transform pairs [4]. Furthermore, defining the latter through Equation 2.10 gives us an intuitive picture of the PSD if we recall Parseval's theorem,

$$\int_{-\infty}^{\infty} \frac{\mathrm{d}\omega}{2\pi}\frac{1}{T}|\hat{x}_T(\omega)|^2 = \frac{1}{T}\int_{-\infty}^{\infty} \mathrm{d}t\,|x(t)|^2. \tag{2.12}$$

That is, the total power P contained in the signal $x(t)$ is given by integrating over the PSD. Similarly, the power contained in a band of frequencies $[\omega_1, \omega_2]$ is given by

$$P(\omega_1,\omega_2) = \mathrm{RMS}_S(\omega_1,\omega_2)^2 \tag{2.13}$$

$$= \int_{\omega_1}^{\omega_2} \frac{\mathrm{d}\omega}{2\pi}S(\omega) \tag{2.14}$$

where $\mathrm{RMS}_S(\omega_1,\omega_2)$ is the root mean square within this frequency band. These relations are helpful when analyzing noise PSDs to gauge the relative weight of contributions from different frequency bands to the total noise power.

8: We only discuss the problem of equally spaced samples here. Variants for spectral estimation of time series with unequal spacing exist [5, 6].

Equation 2.10 represents the starting point for the experimental spectrum estimation procedure. Instead of a continuous signal $x(t), t \in [0, T]$, consider its discretized version[8]

$$x_n, \quad n \in \{0, 1, \ldots, N-1\} \tag{2.15}$$

defined at times $t_n = n\Delta t$ with $T = N\Delta t$ and where $\Delta t = f_\mathrm{s}^{-1}$ is the sampling interval (the inverse of the sampling frequency f_s). Invoking the ergodic theorem, we can replace the long-term average in Equation 2.10 by the ensemble average over M realizations i of the noisy signal $x_n^{(m)}$ and

write

$$S_n = \frac{1}{M}\sum_{i=0}^{M-1}\left|\hat{x}_n^{(m)}\right|^2 \tag{2.16}$$

$$= \frac{1}{M}\sum_{i=0}^{M-1} S_n^{(m)} \tag{2.17}$$

where $\hat{x}_n^{(m)}$ is the discrete Fourier transform of $x_n^{(m)}$, we defined the *periodogram* of $x_n^{(m)}$ by

$$S_n^{(m)} = \left|\hat{x}_n^{(m)}\right|^2, \tag{2.18}$$

and S_n is an *estimate* of the true PSD sampled at the discrete frequencies $\omega_n = 2\pi n/T \in 2\pi \times \{-f_s/2\,, \ldots, f_s/2\}$.[9] Equation 2.16 is known as Bartlett's method [7] for spectrum estimation.[10]

To better understand the properties of this estimate, let us take a look at the parameters Δt, N, and M. The sampling interval Δt defines the largest resolvable frequency by the Nyquist sampling theorem,

$$f_{\max} = \frac{f_s}{2} = \frac{1}{2\Delta t}. \tag{2.19}$$

In turn, the number of samples N determines the frequency resolution Δf, or smallest resolvable frequency,

$$f_{\min} = \Delta f = \frac{1}{T} = \frac{1}{N\Delta t} = \frac{f_s}{N}. \tag{2.20}$$

Lastly, M determines the variance of the set of periodograms $\{S_n^{(m)}\}_{i=0}^{M-1}$ and hence the accuracy of the estimate S_n.

In practice, the ensemble realizations i are of course obtained sequentially, implying that one acquires a time series of data $x_n, n \in \{0, 1, \ldots, NM-1\}$ and partitions these data into M sequences of length N. It becomes clear, then, that the Bartlett average (Equation 2.16) trades spectral resolution (larger N) for estimation accuracy (larger M) given the finite acquisition time $T = NM\Delta t$.

An improvement in data efficiency can be obtained using Welch's method [8]. To see how, we first need to discuss spectral windowing.

9: We blithely disregard integer algebra issues occuring here for conciseness and leave it as an exercise for the reader to figure out what the exact bounds of the set of ω_n are.

10: By taking the limit $M \to \infty$ one recovers the true PSD,

$$\lim_{M\to\infty} S_n = S(\omega_n).$$

The continuum limit is as always obtained by sending $\Delta t \to 0, N \to \infty, N\Delta t = \text{const}$.

2.2 Window functions

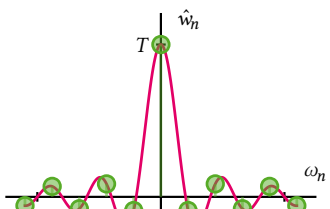
Partitioning a signal x_n into M sections $x_n^{(m)}$ of length N is mathematically equivalent to multiplying the signal with the rectangular *window function* given by[11]

$$w_n^{(m)} = \begin{cases} 1 & \text{if} \quad (m-1)N \le n < mN \quad \text{and} \\ 0 & \text{else} \end{cases} \tag{2.21}$$

so that $x_n^{(m)} = x_n w_n^{(m)}$.

11: This window is also known as the boxcar or Dirichlet window.

add $\omega_n^{(m)}$, scaled ticks



Now recall that multiplication and convolution are duals under the Fourier

transform, implying that

$$\hat{x}_n^{(m)} = \hat{x}_n * \hat{w}_n^{(m)}. \tag{2.22}$$

where the Fourier representation of the rectangular window[12]

$$\hat{w}_n^{(m)} = \hat{w}_n \mathrm{e}^{-\mathrm{i}(m-1/2)\omega_n T}, \tag{2.23}$$

$$\hat{w}_n = T\,\mathrm{sinc}\left(\frac{\omega_n T}{2}\right). \tag{2.24}$$

12: $\mathrm{sinc}(x) = \sin(x)/x$.

Figure 2.1 shows the unshifted rectangular window $\hat{w}_n$ in Fourier space. We can hence understand the Fourier spectrum of $x_n^{(m)}$ as sampling $\hat{x}_n$ with the probe $\hat{w}_n^{(m)}$. However, while in the continuum limit (*c.f.* sidenote 10) Equation 2.24 tends towards $\delta(\omega_n)$ and thus will produce a faithful reconstruction of the true spectrum, the finite frequency spacing Δf of discrete signals introduces a finite bandwidth of the probe as well as *sidelobes*. These effects induce what is known as *spectral leakage* [4, 9] and lead to artifacts and deviations of the spectrum estimator S_n from the true spectrum $S(\omega_n)$.

For this reason, a plethora of *window functions* have been introduced to mitigate the effects of spectral leakage. Key properties of a window are the spectral bandwidth (center lobe width) and sidelobe amplitude between which there typically is a tradeoff.[13]

13: Wikipedia gives a good overview of existing window functions [10].

add $\omega_n^{(m)}$, scaled ticks

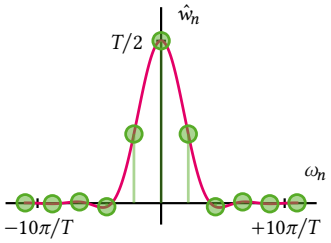


Figure 2.2: The Fourier representation of the Hann window in continuous time.

A window frequently used in spectral analysis is the Hann window [11],

$$w_n^{(m)} = \begin{cases} \cos^2\left(\frac{\pi n}{N}\right) & \text{if} \quad (m-1)N \le n < mN \quad \text{and} \\ 0 \quad \text{else} \end{cases} \tag{2.25}$$

with the Fourier representation of the unshifted window,

$$\hat{w}_n = T\,\mathrm{sinc}\left(\frac{\omega_n T}{2}\right) \times \frac{1}{2(1-\omega_n T/2\pi)(1+\omega_n T/2\pi)}, \tag{2.26}$$

shown in Figure 2.2. The favorable properties of the Hann window are apparent when compared to the rectangular window in Figure 2.1; the sidelobes are quadratically suppressed while the center lobe is only slightly broadened.

Another favorable property of the Hann window is that $w_0^{(0)} = w_{N-1}^{(0)} = 0$. This suppresses detrimental effects arising from a possible discontinuity ($x_0^{(0)} \neq x_{N-1}^{(0)}$) at the edge of a data segment related to the discrete Fourier transform, which assumes periodic data.[14]

14: Although this can usually also be achieved by detrending the data before performing the Fourier transform, which is a good idea in any case.

2.3 Welch's method

Contemplating Equation 2.25, one might come to the conclusion that using a window such as this is not very data efficient in the sense that a large fraction of samples located at the edge of the window is strongly suppressed and hence does not contribute significantly to the spectrum estimate. To alleviate this lack of efficiency, one can introduce an overlap between adjacent data windows. That is, instead of partitioning the data x_n into M non-overlapping sections of length N, one shifts the mth window forward by $-mK$ with $K > 0$ the overlap. Finally, the periodogram

(Equation 2.18) is computed for each window and subsequently averaged to obtain the spectrum estimator (Equation 2.16).

This method of spectrum estimation is known as Welch's method [8]. One can show [8] that the correlation between the periodograms of adjacent, overlapping windows is sufficiently small to avoid a biased estimate. The overlap naturally depends on the choice of window; a typical value for the Hann window $K = N/2$ with which one would obtain $M = 2L/N - 1$ windows for data of length L.[15]

15: Again neglecting integer arithmetic issues.

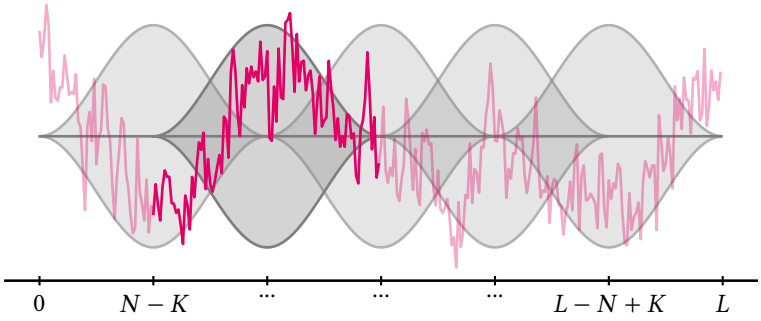


Figure 2.3: Illustration of Welch's method for spectrum estimation. The data (pink) of length L is partitioned into $K = 2L/N-1$ segments of length N. Each segment is multiplied with a window function (gray) which reduces spectral leakage and other artifacts. A finite overlap K between adjacent windows (gray) ensures efficient sample use.

Figure 2.3 conceptually illustrates Welch's method for a trace of $1/f$ noise with $L = 300$ samples in total. Choosing the Hann window and an overlap of 50 % results in $M = 5$ segments for a window length of $N = 100$. The data in the second window is highlighted.

2.3.1 Parameters

We are now in a position to discuss how the various parameters of a time series relate to both to physical parameters of the resulting spectrum estimate and to each other. To this end, we will go through the typical procedure of acquiring a spectrum estimate using Welch's method chronologically.

To acquire data using some form of (digital) data acquisition device (DAQ), one usually needs to specify two parameters first: the total number of samples to be acquired, L, and the sample rate, f_s. This results in a measurement of duration $T = L\Delta t$ where $\Delta t = f_s^{-1}$ as previously mentioned. The choice of f_s already induces an upper bound on the first parameter characterizing the PSD estimate: the largest resolvable frequency $f_{max} \leq f_s/2$ (*c.f.* Equation 2.19, but note that we allow f_{max} to be smaller than half the sample rate in anticipation of hardware constraints). Next, we choose a number of Welch averages, M, *i.e.*, data partitions, and their overlap, K. In doing so, one fixes the number of samples per partition N and thereby induces the lower bound on the second parameter characterizing the PSD estimate: the frequency spacing $\Delta f = 1/N \leq f_{min}$ (*c.f.* Equation 2.20).[16]

Table 2.1: Overview of spectrum estimation parameters. The parameters can be assigned into three groups: 1. DAQ parameters configuring the data acquisition device, 2. Welch parameters specifying the periodogram averaging, and 3. Spectrum properties induced by the above.

1. DAQ parameters	
L	Total number of samples
f_s	Sample rate
2. Welch parameters	
K	Number of overlap samples
N	Number of segment samples
M	Number of Welch segments
3. Spectrum parameters	
f_{min}	Smallest resolvable frequency
f_{max}	Largest resolvable frequency

16: Technically, the smallest resolvable frequency in a fast Fourier transform (FFT) is zero, of course. But as data is typically detrended (a constant or linear trend subtracted) before computation of the periodogram, the smallest *meaningful* frequency is given by f_{min}.

D(A)G for parameter interdependencies?

The python_spectrometer software package

3

In this chapter, I will lay out the design and functionality of the python_spectrometer Python package.[1]

3.1 Package design and implementation

The python_spectrometer package provides a central class, Spectrometer, that users interact with to perform data acquisition, spectrum estimation, and plotting. It is instantiated with an instance of a child class of the DAQ base class that implements an interface to various DAQ hardware devices. New spectra are obtained by calling the Spectrometer.take() method with all acquisition and metadata settings.

In the following, I will go over the the design of these aspects of the package in more detail.

3.1.1 Data acquisition

The daq module contains on the one hand the declaration of the DAQ abstract base class and its child class implementations, and on the other the settings module, which defines the DAQSettings class. This class is used in the background to validate data acquisition settings both for consistency (*c.f.* Subsection 2.3.1) and hardware constraints.

To better understand the necessity of this functionality, consider the typical scenario of a physicist[2] in the lab. Alice has wired up her experiment, performed a first measurement, and to her dismay discovered that the data is too noisy to see the sought-after effect. She sets up the python_spectrometer code to investigate the noise spectrum of her measurement setup. From her noisy data she could already estimate the frequency of the most harrowing noise, so she knows the frequency band $[f_{\text{min}}, f_{\text{max}}]$ she is most interested in. But because she is lazy,[3] she does not want to do the mental gymnastics to convert f_{min} to the parameter that her DAQ device understands, L (see Table 3.1), especially considering that L depends on the number of Welch averages and the overlap. Furthermore, while she could just about do the conversion from f_{max} to the other relevant DAQ parameter, f_s, in her head, her device imposes hardware constraints on the allowed sample rates she can select! The DAQSettings class addresses these issues. It is instantiated with any subset of the parameters listed in Table 3.1[4] and attempts to resolve the parameter interdependencies lined out in Subsection 2.3.1 upon calling DAQSettings.to_consistent_dict().[5] This either infers those parameters that were not given from those that were or, if not possible, uses a default value. Child classes of the DAQ class can subclass DAQSettings to implement hardware constraints such as a finite set of allowed sampling rates or a maximum number of samples per data buffer.

For instance, Alice might want to measure the noise spectrum in the frequency band [1.5 Hz, 72 kHz]. Although she would not have to do this explicitly,[6] she could inspect the parameters after resolution using the code shown in Listing 3.1.

1: The package repository is hosted on GitLab. Its documentation is automatically generated and hosted on GitLab as well. Releases are automatically published to PyPI and allow the package to be installed using `pip install python-spectrometer`.

Table 3.1: Variable names used in Chapter 2 and their corresponding parameter names as used in python_spectrometer and scipy.signal.welch() [12].

Variable	Parameter
L	n_pts
f_s	fs
K	noverlap
N	nperseg
M	n_seg
f_{min}	f_min
f_{max}	f_max

2: Let's call her Alice.

3: Physicists generally are.

4: DAQSettings inherits from the builtin dict and as such can contain arbitrary other keys besides those listed in Table 3.1. However, automatic validation of parameter consistency is only performed for these special keys.

5: Since the graph spanned by the parameters is not acyclic, this only works *most* of the time.

6: Settings are automatically parsed when passed to the take() method of the Spectrometer class.

Listing 3.1: `DAQSettings` example showcasing automatic parameter resolution. `n_avg` determines the number of outer averages, *i. e.*, the number of data buffers acquired and processed individually.

```
>>> from python_spectrometer.daq import DAQSettings
>>> settings = DAQSettings(f_min=1.5, f_max=7.2e4)
>>> settings.to_consistent_dict()
{'f_min': 1.5,
 'f_max': 72000.0,
 'fs': 144000.0,
 'df': 1.5,
 'nperseg': 96000,
 'noverlap': 48000,
 'n_seg': 5,
 'n_pts': 288000,
 'n_avg': 1}
```

```
{'f_min': 14.30511474609375,
 'f_max': 72000.0,
 'fs': 234375.0,
 'df': 14.30511474609375,
 'nperseg': 16384,
 'noverlap': 0,
 'n_seg': 1,
 'n_pts': 16384,
 'n_avg': 1}
```

Listing 3.2: Resolved settings for the same input parameters as in Listing 3.1 but for the `ZurichInstrumentsMFLIScope` backend with hardware constraints on `n_pts` and `fs`.

If the instrument she'd chosen for data acquisition had been a Zurich Instruments MFLI's "Scope" module [13], the same requested settings would have resolved to those shown in Listing 3.2.[7] This is because the Scope module constrains $L \in [2^{12}, 2^{14}]$ and $f_s \in 60\,\mathrm{MHz} \times 2^{[-16,0]} \approx \{915.5\,\mathrm{Hz}, \ldots, 30\,\mathrm{MHz}, 60\,\mathrm{MHz}\}$.

As already mentioned, the `DAQ` base class implements a common interface for different hardware backends, allowing the `Spectrometer` class to be hardware agnostic. That is, changing the instrument that is used to acquire the data does not necessitate adapting the code used to interact with the instrument. To enable this, different instruments require small wrapper drivers that map the functionality of their actual driver onto the interface dictated by the `DAQ` class. This is achieved by subclassing `DAQ` and implementing the `DAQ.setup()` and `DAQ.acquire()` methods. Their functionality is best illustrated by the internal workflow. When acquiring a new spectrum, all settings supplied by the user are first fed into the `setup()` method where instrument configuration takes place. The method returns the actual device settings,[8] which are then forwarded to the `acquire()` generator function. Here, the instrument is armed (if necessary), and subsequently data is fetched from the device and yielded to the caller `n_avg` times, where `n_avg` is the number of outer averages. Listing 3.3 represents the data acquisition workflow as pseudocode.

DAQ pseudocode?

Introduce Bob?

3.1.2 Data processing

Once time series data has been acquired using a given `DAQ` backend, it could in principle immediately be used to estimate the PSD following Equation 2.16. However, it is often desirable to transform, or process, the data in some fashion. This can include simple transformations such as accounting for the gain of a transimpedance amplifier (TIA) and convert

Listing 3.3: DAQ workflow pseudocode. A `SomeDAQ` object (representing the instrument `Some`) is instantiated with a driver object (for instance a `QCoDeS Instrument`). The instrument is configured with the given `user_settings`. Calling the generator function `daq.acquire()` with the actual device settings returns a generator, iterating over which yields one data buffer per iteration. The data buffers can then be passed to further processing functions (the PSD estimator in our example).

```
daq = SomeDAQ(driver)
parsed_settings = daq.setup(**user_settings)
acquisition_generator = daq.acquire(**parsed_settings)
for data_buffer in acquisition_generator:
    do_something_with(data_buffer)
```

[13]: (n.d.), *Scope Module - LabOne API User Manual*

7: And issued a warning to inform the user their requested settings could not be matched.

8: Which might differ from the requested settings as outlined above.

the voltage back to a current,[9] or more complex ones such as applying calibrations. In particular, since the process of computing the PSD already involves Fourier transformation, the processing can also be performed in frequency space.

In `python_spectrometer`, this can be done using a `procfn` (in the time domain) or `fourier_procfn` (in the Fourier domain). The former is specified as an argument directly to the `Spectrometer` constructor. It is a callable with signature `(x, **kwargs) -> xp`, that is, takes the time series data as its first (positional) argument and arbitrary settings that are passed through from the `take()` method as keyword arguments, and returns the processed data. Listing 3.4 shows a simple function that accounts for the gain of an amplifier.

```
def compensate_gain(x, gain=1.0):
    return x / gain
```

Listing 3.4: A simple `procfn`, which converts amplified data back to the level before amplification. More complex processing chains can concisely be defined with `qutil.functools.FunctionChain` that pipes the output of one function into the input of the next.

The latter is specified in the `psd_estimator` argument of the `Spectrometer` constructor. This argument allows the user to specify a custom estimator for the PSD, in which case a callable is expected. Otherwise, it should be a mapping containing parameters for the default PSD estimator, `scipy.signal.welch()` [12]. Here, the keyword `fourier_procfn` should be a callable with signature `(xf, f, **kwargs) -> (xfp, fp)`.[10] That is, it should take the frequency-space data, the corresponding frequencies, and arbitrary keyword arguments and return a tuple of the processed data and the corresponding frequencies. The latter are required in case the function modifies the frequencies.[11] A simple example for a processing function in Fourier space is shown in Listing 3.5, which computes the (anti-)derivative of the data using the fact that

$$\frac{\partial^n}{\partial t^n} \xrightarrow{\text{F.T.}} (\mathrm{i}\omega)^n \tag{3.1}$$

under the Fourier transform. In Part II, I discuss more complex use-cases of the processing functionality included in `python_spectrometer` in the context of vibration spectroscopy.

```
def derivative(xf, f, n=0):
    return xf / (2j * pi * f)**n
```

Listing 3.5: A simple `fourier_procfn`, which calculates the (anti-)derivative.

ref

3.2 Feature overview

überleitung

3.2.1 Sequential spectrum acquisition

Now that we have a basic understanding of the design choices underlying `python_spectrometer`, let us discuss the typical workflow of using the package. The default mode for spectrum acquisition using `python_spectrometer` revolves around the `take()` method. Key to this workflow is the idea that each acquired spectrum can be assigned a comment that allows to easily identify a spectrum in the main plot. For instance, this comment could contain information about the particular settings that were active when the spectrum was recorded, or where a particular cable was placed.

Consider as an example the procedure of "noise hunting", *i.e.*, debugging a noisy experimental setup. The experimentalist,[12] having discovered that his data is noisier than expected, sets up the `Spectrometer` class with an instance of the `DAQ` subclass for the DAQ instrument connected to his sample. Choosing the frequency bounds, say $f_{\min} = 10\,\mathrm{Hz}$ and $f_{\max} = 100\,\mathrm{kHz}$, and using the sensible defaults for the remaining spectrum parameters, Charlie first grounds the input of his DAQ to record a

9: Although it is of course less than trivial to discriminate between current and voltage noise in a TIA.

[12]: (n.d.), *Welch — SciPy v1.15.2 Manual*

10: *I.e.*, the `psd_estimator` argument would be `{"fourier_procfn": fn}`.

11: One example is the `octave_band_rms()` function from the `qutil.signal_processing` module [14]

12: Let's call him Charlie.

baseline spectrum. Thus far, his code would hence look something like that shown in Listing 3.6.

Listing 3.6: Setup and workflow using the `python_spectrometer` package. `session` and `device` are Application Programming Interface (API) objects of the `zhinst.toolkit` driver package. It is therefore possible to simply use the driver objects that are already in use in the measurement setup. The `procfn` and `processed_unit` arguments help converting raw data into a more human-friendly unit.

```
from python_spectrometer import Spectrometer, daq
from qutil.functools import scaled

mfli_daq = daq.ZurichInstrumentsMFLIDAQ(session, device)
spect = Spectrometer(mfli_daq, procfn=scaled(1e6),
                     processed_unit='µV')
spect.take('baseline', f_min=1e1, f_max=1e5, daq='grounded')
```

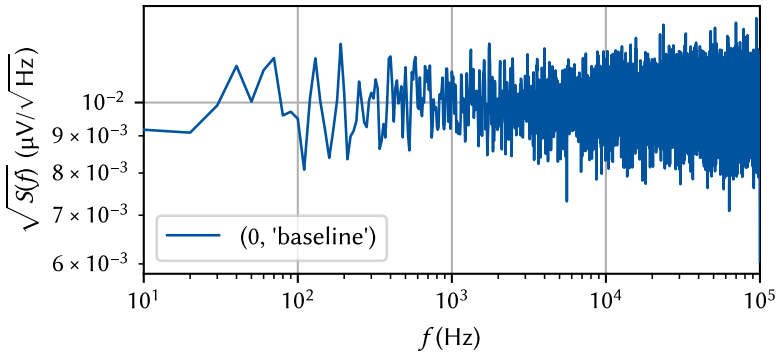


Figure 3.1: The `python_spectrometer` plot after acquiring the (synthetic) baseline spectrum. By default, the amplitude spectral density (ASD) = √PSD is displayed in the main plot. Each spectrum is assigned a unique identifier key consisting of an incrementing integer and the user comment, and can be refered to by either (or both) when interacting with the object.

After acquiring the baseline, he next ungrounds the DAQ to obtain a representative spectrum of the noise in an actual measurement. He then proceeds by tweaking things on his setup, testing out different parameters, *etc.* Every time he changes something, he acquires another spectrum using `take()`, labeling each with a meaningful comment for identification.

Listing 3.7: Code to acquire additional spectra. Arbitrary key-value pairs can be passed to the `take()` method, which are stored as metadata if they do not apply to any functions downstream in the data processing chain.

```
settings = {'f_min': 1e1, 'f_max': 1e5, 'daq': 'connected'}
spect.take('connected', **settings)
spect.take('lifted cable', cable='lifted', **settings)
spect.take('jumped', **settings)
```

This leaves him with the spectrometer plot as shown in Figure 3.2. While working, Charlie realizes he'd like see the signal in the time-domain as well. He easily achieves this by setting `spect.plot_timetrace = True`, which adds an oscilloscope subplot to spectrometer figure as shown in Figure 3.3.

Charlie now observes that the noise spectra he has acquired display many sharp peaks in particular at high frequencies while the $1/f$ noise floor seems pretty consistent across different measurements. This makes it harder for him to evaluate whether any of his changes are actually an improvement or not. The `python_spectrometer` package allows addressing this by plotting the integrated spectra in another subplot. Charlie's spectrometer figure after setting `spect.plot_cumulative = True` is shown in Figure 3.4. In the case that `spect.plot_amplitude == True`, this new subplot shows the root mean square (RMS) in the band $[f_{\text{min}}, f]$,

$$\text{RMS}_S(f) \equiv \text{RMS}_S(f_{\text{min}}, f), \tag{3.2}$$

and the band power (Equation 2.13) otherwise.

The cumulative RMS plot already helps, but Charlie would like a more quantitative comparison of relative spectral powers. Therefore, he rescales

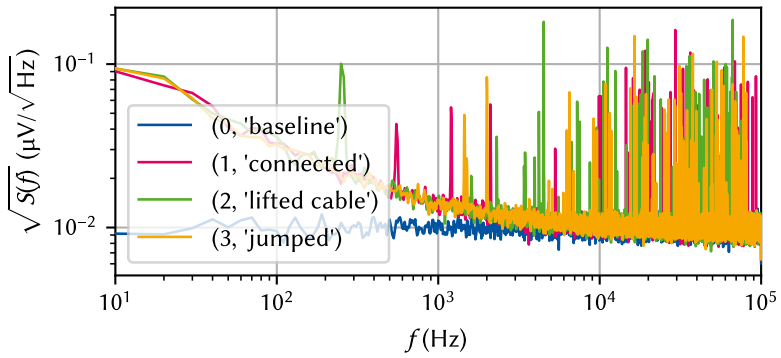


Figure 3.2: The `python_spectrometer` plot after acquiring additional (synthetic) spectra. Each spectrum is uniquely identified by a two-tuple of `(index, comment)`.

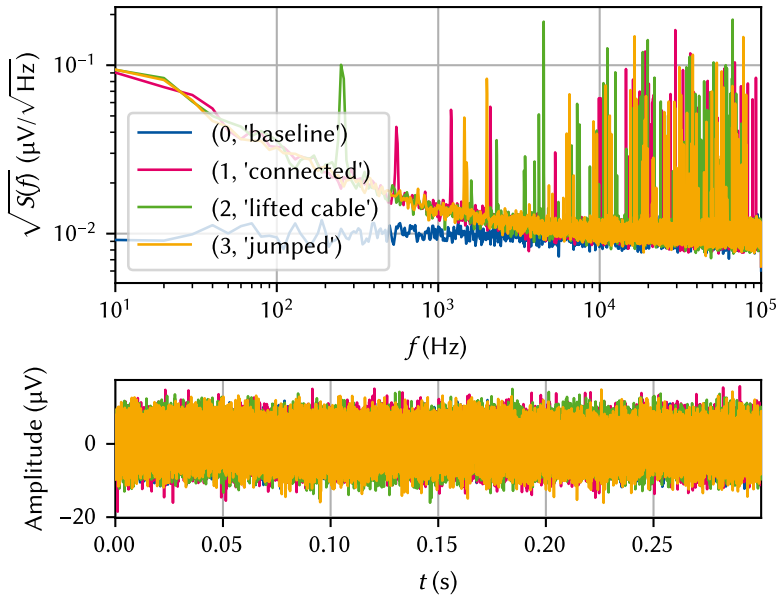


Figure 3.3: The `python_spectrometer` plot shown in Figure 3.2 when setting `spect.plot_timetrace = True`. This adds a subplot that shows the time series data from which the PSD was computed akin to what an oscilloscope would show. Note that this is the entire time series, *i.e.*, the data of length L, which is (by default, using Welch's method) segmented for spectrum estimation.

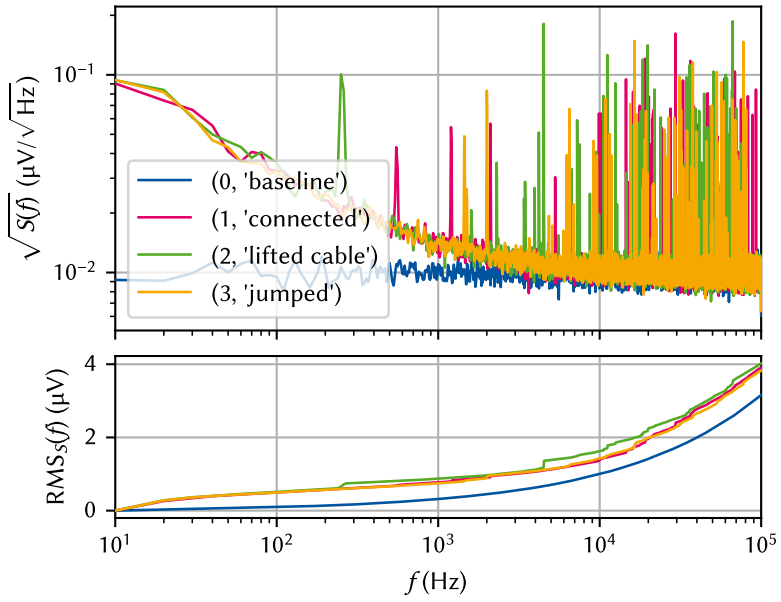


Figure 3.4: The `python_spectrometer` plot shown in Figure 3.2 when setting `spect.plot_cumulative = True`. This adds a subplot that shows the RMS (*c.f.* Equation 2.13) which can be helpful in evaluating the contribution of individual peaks in the spectrum to the total noise power. Both the oscilloscope subplot (Figure 3.3) and the RMS subplot can also be shown at the same time.

the spectra in terms of their relative powers expressed in dB[13] by applying the following settings, which produces the plot shown in Figure 3.5:

```
spect.plot_dB_scale = True
spect.plot_amplitude = False
spect.plot_density = False
```

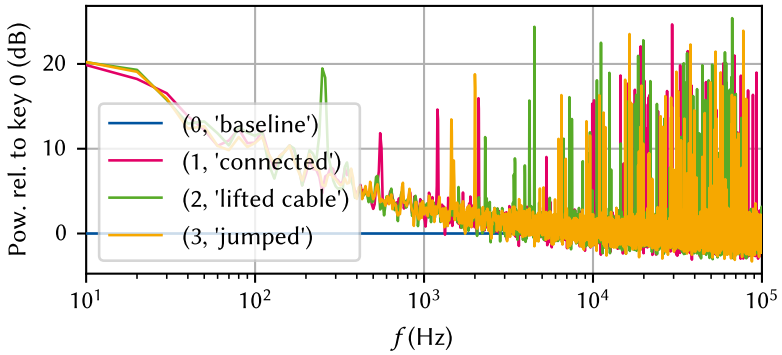


Figure 3.5: The `python_spectrometer` plot in relative mode. Starting from the state in Figure 3.2, we set `spect.plot_dB_scale = True` and `spect.plot_amplitude = False` to compare the relative noise powers with respect to the baseline.

The attribute `plot_density` controls whether the *power spectral density* or the *power spectrum.*[14] Scaling the data to the power spectrum instead of the density, Charlie can get an estimate of the RMS at a single frequency by reading off the peak height. Additionally displaying the data in dB then gives insight into relative noise powers of different spectra.

Charlie carries on with his enterprise and continues to acquire spectra until, finally, he finds the source of his noise! Alas, his spectrometer plot is now overflowing with plotted data when really he just wants to compare the baseline, the original, noisy state, and the final, clean spectrum. He simply calls

```
spect.hide(*range(2, 10))
```

to hide the eight spectra of unsuccessful debugging, leaving him with a plot as shown in Figure 3.6.

Finally, happy with the results, Charlie serializes the state of the spectrometer to disk, allowing him to pick up where he left off at a later point in time:

```
# do some stuff
spect.serialize_to_disk('2032-12-24_noise_hunting')
```

The next week, Charlie is asked by his team about his progress on debugging the noise in their setup. Even though he is working from home that day and does not have access to the lab computer, Charlie simply uses

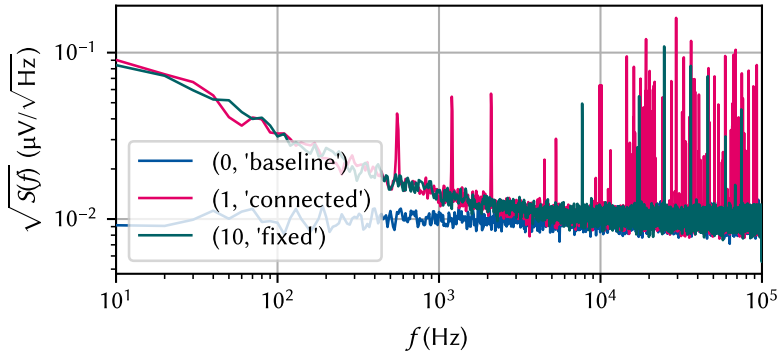


Figure 3.6: The `python_spectrometer` plot after multiple additional spectra were acquired and hidden. Hiding spectra that one is not interested in anymore is achieved through `spect.hide(*range(2, 10))`. This is reversed by `spect.show(*range(2, 10))`. Data can also be dropped (`spect.drop(key)`) or deleted (`spect.delete(key)`) from the internal cache and disk, respectively.

13: Recall that the decibel is defined by the ratio L_P of two powers P_1, P_2 as [15]

$$L_P = 10 \log_{10}\left(\frac{P_1}{P_2}\right) \text{ dB}.$$

14: At this point, we *do* need to distinguish between the PSD and the power spectrum counter to sidenote 2 in Chapter 2. The PSD and power spectrum are related by the equivalent noise bandwidth (ENBW),

$$\text{Spectral density} \xrightarrow{\times \text{ENBW}} \text{Spectrum},$$

which is itself a function of the sampling rate and the properties of the spectral window [9],

$$\text{ENBW} = f_s \frac{\sum_n \hat{w}_n^2}{\left[\sum_n \hat{w}_n\right]^2}. \tag{3.3}$$

his laptop computer and pulls up the `Spectrometer` session stored on the server, allowing them to interactively discuss the spectra:

```
# Read-only instance because no DAQ attached
file = '2032-12-24_noise_hunting'
spect = Spectrometer.recall_from_disk(savepath / file)
```

This opens up the plot shown in Figure 3.6 again. While they cannot acquire new spectra in this state,[15] they can still use all the plotting features like showing or hiding spectra, or changing plot types as discussed above.

15: They could of course always attach a `DAQ` instance to the spectrometer and continue as they were.

Part II

Characterization and Improvements of a Millikelvin Confocal Microscope

Part III

Electrostatic Trapping of Excitons in Semiconductor Membranes

Part IV

A Filter-Function Formalism For Unital Quantum Operations

Appendix

Bibliography

[1] A. A. Clerk et al. "Introduction to Quantum Noise, Measurement, and Amplification." In: *Rev. Mod. Phys.* 82.2 (Apr. 15, 2010), pp. 1155–1208. DOI: 10.1103/RevModPhys.82.1155. (Visited on 01/19/2022) (cited on page 5).

[2] Gerardo A. Paz-Silva, Leigh M. Norris, and Lorenza Viola. "Multiqubit Spectroscopy of Gaussian Quantum Noise." In: *Phys. Rev. A* 95.2 (Feb. 23, 2017), p. 022121. DOI: 10.1103/PhysRevA.95.022121 (cited on page 5).

[3] M. Mehmandoost and V. V. Dobrovitski. "Decoherence Induced by a Sparse Bath of Two-Level Fluctuators: Peculiar Features of $1/F$ Noise in High-Quality Qubits." In: *Phys. Rev. Res.* 6.3 (Aug. 15, 2024), p. 033175. DOI: 10.1103/PhysRevResearch.6.033175. (Visited on 08/20/2024) (cited on page 5).

[4] Lambert Herman Koopmans. *The Spectral Analysis of Time Series*. 2nd ed. Vol. 22. Probability and Mathematical Statistics. San Diego: Academic Press, 1995 (cited on pages 5, 6, 8).

[5] N. R. Lomb. "Least-Squares Frequency Analysis of Unequally Spaced Data." In: *Astrophys Space Sci* 39.2 (Feb. 1, 1976), pp. 447–462. DOI: 10.1007/BF00648343. (Visited on 04/10/2025) (cited on page 6).

[6] J. D. Scargle. "Studies in Astronomical Time Series Analysis. II. Statistical Aspects of Spectral Analysis of Unevenly Spaced Data." In: *Astrophys. J.* 263 (Dec. 1, 1982), pp. 835–853. DOI: 10.1086/160554. (Visited on 04/10/2025) (cited on page 6).

[7] M. S. Bartlett. "Smoothing Periodograms from Time-Series with Continuous Spectra." In: *Nature* 161.4096 (May 1948), pp. 686–687. DOI: 10.1038/161686a0. (Visited on 03/26/2025) (cited on page 7).

[8] P. Welch. "The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging over Short, Modified Periodograms." In: *IEEE Trans. Audio Electroacoustics* 15.2 (June 1967), pp. 70–73. DOI: 10.1109/TAU.1967.1161901 (cited on pages 7, 9).

[9] F.J. Harris. "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform." In: *Proc. IEEE* 66.1 (Jan. 1978), pp. 51–83. DOI: 10.1109/PROC.1978.10837. (Visited on 03/27/2025) (cited on pages 8, 16).

[10] *Window Function*. In: *Wikipedia*. Mar. 20, 2025. (Visited on 03/27/2025) (cited on page 8).

[11] A. Nuttall. "Some Windows with Very Good Sidelobe Behavior." In: *IEEE Trans. Acoust. Speech Signal Process.* 29.1 (Feb. 1981), pp. 84–91. DOI: 10.1109/TASSP.1981.1163506. (Visited on 03/27/2025) (cited on page 8).

[12] *Welch — SciPy v1.15.2 Manual*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.welch.html (visited on 03/31/2025) (cited on pages 11, 13).

[13] *Scope Module - LabOne API User Manual*. URL: https://docs.zhinst.com/labone_api_user_manual/modules/scope/index.html (visited on 04/02/2025) (cited on page 12).

[14] *Octave_band_rms — Qutil 2025.3.1 Documentation*. URL: https://qutech.pages.rwth-aachen.de/qutil/_autogen/qutil.signal_processing.fourier_space.octave_band_rms.html (visited on 04/03/2025) (cited on page 13).

[15] David M. Pozar. *Microwave Engineering*. 3. ed. Hoboken, NJ: Wiley, 2005. 700 pp. (cited on page 16).

Special Terms

A
API Application Programming Interface. 14
ASD amplitude spectral density. 14

D
DAQ data acquisition device. 9, 11–14

E
ENBW equivalent noise bandwidth. 16

F
FFT fast Fourier transform. 9

P
PSD power spectral density. 5–7, 9, 12–16

R
RMS root mean square. 6, 14–16

T
TIA transimpedance amplifier. 12, 13
TLF two-level fluctuator. 5