Use this document as a template

Millikelvin Confocal Microscopy of Semiconductor Membranes and Filter Functions for Unital Quantum Operations

Customise this page according to your needs

Tobias Hangleiter*

September 7, 2025

* A LaTeX lover/hater

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

Summary

Zusammenfassung

Acknowledgements

Tobias Hangleiter
Aachen, September 2025

Preface

It's been a journey.

Tobias Hangleiter
Aachen, September 2025

How to read this thesis

Contents

Publications

[1] Yaiza Aragonés-Soria, René Otten, Tobias Hangleiter, Pascal Cerfontaine, and David Gross. "Minimising Statistical Errors in Calibration of Quantum-Gate Sets." June 7, 2022. DOI: 10.48550/arXiv.2206.03417. arXiv: 2206.03417 [quant-ph]. Pre-published.

[2] Pascal Cerfontaine, Tobias Hangleiter, and Hendrik Bluhm. "Filter Functions for Quantum Processes under Correlated Noise." In: *Physical Review Letters* 127.17 (Oct. 18, 2021), p. 170403. DOI: 10.1103/PhysRevLett.127.170403.

[3] Thomas Descamps, Feng Liu, Sebastian Kindel, René Otten, Tobias Hangleiter, Chao Zhao, Mihail Ion Lepsa, Julian Ritzmann, Arne Ludwig, Andreas D. Wieck, Beata E. Kardynał, and Hendrik Bluhm. "Semiconductor Membranes for Electrostatic Exciton Trapping in Optically Addressable Quantum Transport Devices." In: *Physical Review Applied* 19.4 (Apr. 28, 2023), p. 044095. DOI: 10.1103/PhysRevApplied.19.044095.

[4] Thomas Descamps, Feng Liu, Tobias Hangleiter, Sebastian Kindel, Beata E. Kardynał, and Hendrik Bluhm. "Millikelvin Confocal Microscope with Free-Space Access and High-Frequency Electrical Control." In: *Review of Scientific Instruments* 95.8 (Aug. 9, 2024), p. 083706. DOI: 10.1063/5.0200889.

[5] Denny Dütz, Sebastian Kock, Tobias Hangleiter, and Hendrik Bluhm. "Distributed Bragg Reflectors for Thermal Isolation of Semiconductor Spin Qubits." In preparation.

[6] Sarah Fleitmann, Fabian Hader, Jan Vogelbruch, Simon Humpohl, Tobias Hangleiter, Stefanie Meyer, and Stefan van Waasen. "Noise Reduction Methods for Charge Stability Diagrams of Double Quantum Dots." In: *IEEE Transactions on Quantum Engineering* 3 (2022), pp. 1–19. DOI: 10.1109/TQE.2022.3165968.

[7] Fabian Hader, Jan Vogelbruch, Simon Humpohl, Tobias Hangleiter, Chimezie Eguzo, Stefan Heinen, Stefanie Meyer, and Stefan van Waasen. "On Noise-Sensitive Automatic Tuning of Gate-Defined Sensor Dots." In: *IEEE Transactions on Quantum Engineering* 4 (2023), pp. 1–18. DOI: 10.1109/TQE.2023.3255743.

[8] Tobias Hangleiter, Pascal Cerfontaine, and Hendrik Bluhm. "Filter-Function Formalism and Software Package to Compute Quantum Processes of Gate Sequences for Classical Non-Markovian Noise." In: *Physical Review Research* 3.4 (Oct. 18, 2021), p. 043047. DOI: 10.1103/PhysRevResearch.3.043047.

[9] Tobias Hangleiter, Pascal Cerfontaine, and Hendrik Bluhm. "Erratum: Filter-function Formalism and Software Package to Compute Quantum Processes of Gate Sequences for Classical Non-Markovian Noise [Phys. Rev. Research 3, 043047 (2021)]." In: *Physical Review Research* 6.4 (Oct. 16, 2024), p. 049001. DOI: 10.1103/PhysRevResearch.6.049001.

[10] Isabel Nha Minh Le, Julian D. Teske, Tobias Hangleiter, Pascal Cerfontaine, and Hendrik Bluhm. "Analytic Filter-Function Derivatives for Quantum Optimal Control." In: *Physical Review Applied* 17.2 (Feb. 2, 2022), p. 024006. DOI: 10.1103/PhysRevApplied.17.024006.

[11] Paul Surrey, Julian D. Teske, Tobias Hangleiter, Pascal Cerfontaine, and Hendrik Bluhm. "Data-Driven Qubit Characterization and Optimal Control Using Deep Learning." In preparation.

[12] Kui Wu, Sebastian Kindel, Thomas Descamps, Tobias Hangleiter, Jan Christoph Müller, Rebecca Rodrigo, Florian Merget, Beata E. Kardynal, Hendrik Bluhm, and Jeremy Witzens. "Modeling an Efficient Singlet-Triplet-Spin-Qubit-to-Photon Interface Assisted by a Photonic Crystal Cavity." In: *Physical Review Applied* 21.5 (May 24, 2024), p. 054052. DOI: 10.1103/PhysRevApplied.21.054052.

[13] Kui Wu, Sebastian Kindel, Thomas Descamps, Tobias Hangleiter, Jan Christoph Müller, Rebecca Rodrigo, Florian Merget, Hendrik Bluhm, and Jeremy Witzens. "An Efficient Singlet-Triplet Spin Qubit to Fiber Interface Assisted by a Photonic Crystal Cavity." In: *The 25th European Conference on Integrated Optics*. Ed. by Jeremy Witzens, Joyce Poon, Lars Zimmermann, and Wolfgang Freude. Cham: Springer Nature Switzerland, 2024, pp. 365–372. DOI: 10.1007/978-3-031-63378-2_60.

Software

The following open-source software packages were developed (at least partially) during the work on this thesis.

[1] Tobias Hangleiter, Isabel Nha Minh Le, and Julian D. Teske, *Filter_functions* version v1.1.3, May 14, 2024. Zenodo. DOI: 10.5281/ZENODO.4575000,

[2] Tobias Hangleiter, *Lindblad_mc_tools* Aug. 8, 2025. URL: https://git.rwth-aachen.de/tobias.hangleiter/lindblad_mc_tools.

[3] Tobias Hangleiter, *Mjolnir* Aug. 8, 2025. URL: https://git-ce.rwth-aachen.de/qutech/python-mjolnir.

[4] Tobias Hangleiter, Simon Humpohl, Max Beer, and René Otten, *Python-Spectrometer* version 2024.11.1, Nov. 21, 2024. Zenodo. DOI: 10.5281/ZENODO.13789861,

[5] Tobias Hangleiter, Simon Humpohl, Paul Surrey, and Han Na We, *Qutil* version 2024.11.1, Nov. 21, 2024. Zenodo. DOI: 10.5281/ZENODO.14200303,

Part I

A Flexible Python Tool For Fourier-Transform Noise Spectroscopy

1 Introduction

Author contributions — *This part of the present thesis has benefited from discussions with several people as part of a course taught during the winter semester of 2022/23. The software package presented here was developed by me and has received contributions from Simon Humpohl,*[a] *Max Beer,*[a] *Paul Surrey,*[a] *and René Otten.*[b]

a: RWTH Aachen University and Forschungszentrum Jülich GmbH.

b: Then at RWTH Aachen University and Forschungszentrum Jülich.

Noise is ubiquitous in condensed matter physics experiments, and in mesoscopic systems in particular it can easily drown out the sought-after signal. In solid-state quantum technology research, devices on the length scale of the Fermi wavelength – say tens of nanometers – are embedded in a crystalline matrix of 10^{23} atoms. These devices host single quantum objects such as electrons or quasiparticles—collective many-body excitations. They are controlled and measured by classical signals routed to the device through macroscopic connections like cables and fibers. The signals, in turn, are generated and analyzed by electronic (*e.g.*, transistors) or optical instruments (*e.g.*, lasers) that, in all likelihood, are again based on solid-state technology driven by the first quantum revolution [1, 2]. Ultimately, then, the experiments take place in an environment full of external influences from trams passing by, shaking the ground, to cosmic rays creating electron-hole or breaking up Cooper pairs.

All of these different layers to such experiments are inherently – and in fact often fundamentally[1] – *noisy*, and it is the physicist's challenge to measure their desired effects in spite of this noise. A well-thought-out experimental setup is hence one that has been designed taking the various noise sources into consideration, and state-of-the-art experiments often need to push the frontier in order to be successful. From material choice in the sample to the signal path and the specs of the measurement equipment, many aspects need to be optimized and, in particular, characterized in order to assess the noise. Indeed, the assessment of noise might even be the entire *goal* of the experiment, for instance to evaluate material properties. In this case, the quantity being measured might not be the same quantity whose noise one is interested in but rather some function of it, and the resulting data still needs to be transformed before one is able to make any practical statements about those properties.

1: Consider Johnson-Nyquist noise in a resistor, for example.

Noise, in the sense that we are concerned with in the present thesis,[2] is a stochastic process, meaning that we cannot predict with certainty a dynamical system's time evolution; it *fluctuates* randomly around its noise-free value.[3] Only by obtaining statistics, *i.e.*, repeated observations, either by preparing the system in the same initial state or measuring for a certain amount of time, can we make any statements about the underlying stochastic process. Two questions are key to assessing these statistics: first, what is the amplitude of the fluctuations? And second, at which frequency do the fluctuations occur? If the amplitude is small enough, we might not need to care, and if the frequency is large enough, we might be able to average out the fluctuations while if it is small enough, it might appear *quasistatic* and we might be able to subtract them. Although numerous other techniques for measuring and estimating a noise's properties exist, analyzing noise in frequency space by means of the Fourier transform imposes itself when considering these two notions.

2: Quantum noise does not fall under this scope as it may – disregarding vacuum fluctuations for the sake of argument – be considered *emergent*; it arises from a system entangled with a (not necessarily large) number of quantum degrees of freedom being observed, *i.e.*, tracing out the environment's degrees of freedom. Our lack of knowledge about this environment then appears as noise in our observations. See Reference 3 for a comprehensive review of the quantum case.

3: Quantum measurements are also noisy in this sense as we cannot predict the outcome of a single-shot measurement, but here the stochasticity is in the outcomes of an ensemble rather than the sequence of values in time. The two are, however, closely related through ergodicity, which we will require in Section 2.1.

This approach is the central topic of this part of the present thesis. However, we will not be too much concerned with the theoretical side of the subject matter. Rather, I will focus on making these techniques readily and easily accessible to experimentalists in the lab. Given the arguments laid out above, noise spectroscopy should be an essential item in an experimentalists toolbelt. In practice, though, we are faced with several challenges. First, different experiments require different data acquisition (DAQ) hardware, all of which have both varying capabilities and software interfaces. Hence, transferring a spectroscopy code from one device or setup to another is a non-trivial task and can inhibit adoption of the technique. Albeit some instruments come with built-in spectroscopy solutions, they vary in functionality and are not easily transferred to different systems. Second, while probably everyone finding themselves in the situation is capable of computing the noise spectrum when presented with a set of time series data, inferring the correct parameters for data acquisition given the desired parameters of the resulting spectrum can, while not difficult, be cumbersome to do.[4] Lastly, noise spectroscopy is most effective when proper visualization tools are employed. Again, this is not a difficult task per se, but such things always incur overhead costs that can deter users from employing these essential techniques.

4: Although it is of course a good exercise, and, as a physicist, one should always strive to understand the tools one is using and the underlying principles at work. This part of the present thesis is intended to provide a starting point for that.

Here, I address these points by introducing a Python software package, `python_spectrometer` [4], that tackles the entire processing chain of practical noise spectroscopy in a physics laboratory. By abstracting DAQ hardware into a unified interface, it is portable between different setups. With the goal to make noise spectroscopy as accessible as possible and lower the entry barrier, it automatically handles parameter inference and hardware constraints. Finally, it provides a comprehensive plotting solution that allows for interactively analyzing the data using various different data visualization methods.

The rest of this part is structured as follows. In Chapter 2, I review the mathematical groundwork underpinning noise spectroscopy by means of Fourier transforms of time series and discuss parameters and properties of the central quantity of interest, the power spectral density (PSD). In Chapter 3, I then present the software package by going over its design choices and giving a walkthrough of its features using a typical workflow as an example. I conclude by giving an outlook on future directions in Chapter 4.

Theory of spectral noise estimation

2

There exist various methods for estimating the properties of noise in a classical signal $x(t)$.[1] A simple metric quantifying the average noise amplitude of the signal observed from time $t = 0$ to $t = T$ is the root mean square (RMS) [6],

$$\mathrm{RMS}_x = \sqrt{\frac{1}{T}\int_0^T \mathrm{d}t\,|x(t)|^2}. \tag{2.1}$$

While this already tells us *something* about the noise, it is evident that a single number does not provide many clues if we were to attempt to mitigate the noise or say something qualitative about it beyond "small" and "large". In cases such as these, physics has often turned to the *spectral* representation of the function of interest. Knowing the frequency content of a function gives access to a wealth of information about the underlying contributing processes. But how can we learn how a system behaves as a function of frequency?

Consider an electrical black box – some device under test (DUT) – with two leads connected to a lock-in amplifier (LIA) as sketched in Figure 2.1. Assuming the DUT is conducting, we could simply measure the conductance $G(t) = I(t)/V(t)$ through the device for some time T with a given lock-in modulation frequency f_i, subtract the constant offset,[2] and calculate the RMS using Equation 2.1. Repeating this procedure for different modulation frequencies $\{f_i\}_i$, we would collect a set of RMS-values that we could assign to the modulation frequencies at which they were measured and thus sample the noise amplitude spectrum[3] of the conductance, $S_G(f_i)$.[4] This method has the advantage that we can choose the frequencies at which the spectrum should be sampled—in particular they do not have to be evenly spaced. However, it has two significant shortcomings. First, it is rather inefficient and therefore time-consuming. For N frequency sample points, it takes a total time of NT to acquire all data, where T needs to be chosen such that the variance of RMS_G is sufficiently small. Second, and crucially, it is not always possible to excite the system at a certain frequency and measure its response. For example, it is generally considered hard[5] to – deliberately – excite vibrations of a specific frequency in a cryostat, yet we might still be interested in the displacement spectrum inside of it. A related method is available if one has access to a frequency-tunable probe, that is, a physical system whose behavior – typically its time evolution or some measurable property after letting the system evolve – under noise is well understood within a given noise model, and that can be controlled to be sensitive to a certain frequency. One example for such methods is dynamical decoupling (DD)-based noise spectroscopy using a qubit [7–10], a protocol based on insights gained from the filter-function formalism—see Part IV. These protocols are especially well-suited for high but less so for low frequencies as they rely on observing the coherence of a qubit and hence the visibility on long time scales.

In a sense the most straight-forward way of measuring the noise spectrum presents itself as a consequence of the Wiener-Khinchin theorem [11, 12] that relates the noise spectrum to the Fourier transform of a time-dependent signal. This makes it possible to estimate the noise spectrum by simply observing a signal for a certain amount of time and perform-

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 Refences 3 and 5, for example.

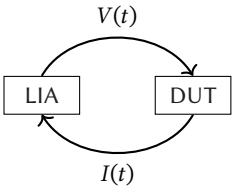

Figure 2.1: Measuring the conductance through a device under test (DUT) using a lock-in amplifier (LIA).

2: The constant part $G_0 = G(t) - \delta G(t)$ of course also holds some information about the system, for example about its bandwidth.

3: We will properly define this term below.

4: See Subsection 3.2.1 for a discussion on how the RMS at a certain frequency relates to other quantities discussed in this part.

5: And also unpopular with colleagues.

ing some post-processing![6] This method will be the topic of the present chapter. It is laid out as follows. I will first derive the Wiener-Khinchin theorem for continuous stochastic processes and introduce the central quantity of interest in noise spectroscopy; the PSD. Then, I will discretize the method and discuss resulting side-effects before describing an efficient way of obtaining the spectrum from a finite amount of data. I will conclude by elaborating on relevant parameters and properties.

6: An interesting combination of the Fourier-transform-based and the qubit-based spectroscopy methods was recently proposed in Reference 13, where the authors compute the spectrum by Fourier-transforming the second derivative of the coherence function.

2.1 Spectrum estimation from time series

To see how spectral noise properties may be estimated from time series data, consider a continuous wide-sense stationary[7] 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[8]

$$\hat{x}_T(\omega) = \int_0^T \mathrm{d}t\, x(t) e^{-i\omega t} \tag{2.2}$$

$$x(t) = \int_{-\infty}^{\infty} \frac{\mathrm{d}\omega}{2\pi} \hat{x}_T(\omega) e^{i\omega t}, \tag{2.3}$$

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

$$C(\tau) = \langle x^*(t) x(t+\tau) \rangle \tag{2.4}$$

$$= \lim_{T\to\infty} \frac{1}{T} \int_0^T \mathrm{d}t\, x^*(t) x(t+\tau), \tag{2.5}$$

where $\langle \bullet \rangle$ 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 2.2) and reordering the integrals, we get

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

$$= \lim_{T\to\infty} \frac{1}{T} \int_{-\infty}^{\infty} \frac{\mathrm{d}\omega}{2\pi} \int_{-\infty}^{\infty} \frac{\mathrm{d}\omega'}{2\pi} \hat{x}_T^*(\omega) \hat{x}_T(\omega') e^{i\omega'\tau} \int_0^T \mathrm{d}t\, e^{it(\omega'-\omega)} \tag{2.7}$$

The innermost integral approaches a δ-function for large T, allowing us to further simplify this under the limit as[9]

$$C(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-\infty}^{\infty} \frac{\mathrm{d}\omega}{2\pi} \int_{-\infty}^{\infty} \frac{\mathrm{d}\omega'}{2\pi} \hat{x}_T^*(\omega) \hat{x}_T(\omega') e^{i\omega'\tau} \delta(\omega' - \omega) \tag{2.8}$$

$$= \lim_{T\to\infty} \frac{1}{T} \int_{-\infty}^{\infty} \frac{\mathrm{d}\omega}{2\pi} |\hat{x}_T(\omega)|^2 e^{i\omega\tau} \tag{2.9}$$

$$= \int_{-\infty}^{\infty} \frac{\mathrm{d}\omega}{2\pi} S(\omega) e^{i\omega\tau} \tag{2.10}$$

with the power spectral density (PSD)[10]

$$S(\omega) = \lim_{T\to\infty} \frac{1}{T} |\hat{x}_T(\omega)|^2 \tag{2.11}$$

$$= \int_{-\infty}^{\infty} \mathrm{d}\tau C(\tau) e^{-i\omega\tau} \tag{2.12}$$

7: For a wide-sense stationary (also called weakly stationary) process $x(t)$, the mean is constant and the autocorrelation function $C(t,t') = \langle x^*(t)x(t')\rangle$ simplifies to $\langle x^*(t)x(t+\tau)\rangle = \langle x^*(0)x(\tau)\rangle$ with $\tau = t' - t$ and where $\langle \bullet \rangle$ is the ensemble average. 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 [14]. The property further implies that $C(\tau)$ is an even function.

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

9: We normalize the δ-function such that $1 = \int \frac{\mathrm{d}\omega}{2\pi} \delta(\omega)$.

10: The term *power spectrum* is often used interchangeably. 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 20 in Chapter 3.

Equation 2.10 is the Wiener-Khinchin theorem [11, 12] that states that the autocorrelation function $C(\tau)$ and the PSD $S(\omega)$ are Fourier-transform pairs [14]. Furthermore, defining the latter through Equation 2.11 gives us an intuitive picture of the PSD if we recall Parseval's theorem,

$$\int_{-\infty}^{\infty} \frac{\mathrm{d}\omega}{2\pi} \frac{1}{T} |\hat{x}_T(\omega)|^2 = \frac{1}{T} \int_0^T \mathrm{d}t\, |x(t)|^2. \tag{2.13}$$

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

$$P(\omega_1, \omega_2) = \mathrm{RMS}_S(\omega_1, \omega_2)^2 \tag{2.14}$$

$$= \int_{\omega_1}^{\omega_2} \frac{\mathrm{d}\omega}{2\pi} S(\omega) \tag{2.15}$$

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.

To become familiar with the quantities $C(\tau)$ and $S(\omega)$, consider the Ornstein-Uhlenbeck process [15], the only stationary Gaussian Markovian stochastic process [16]. The autocorrelation function of the Ornstein-Uhlenbeck process is given by

$$C(\tau) = \sigma^2 e^{-\tau/\tau_c}, \tag{2.16}$$

with σ the RMS and τ_c the correlation time of the process. The PSD in turn is the Lorentzian function

$$S(\omega) = \frac{2\sigma^2 \tau_c}{1 + (\omega\tau_c)^2}. \tag{2.17}$$

For a given discretization time step Δt and hence bandwidth $\omega \in [0, \pi f_\mathrm{S}]$, the Ornstein-Uhlenbeck process interpolates between perfectly uncorrelated, white noise ($\Delta t/\tau_c \to \infty, S(\omega) = 2\tau_c\sigma^2$), correlated, Brownian noise ($\Delta t/\tau_c \gg 1, S(\omega) = 2\sigma^2/\tau_c\omega^2$), and perfectly correlated, quasistatic noise ($\Delta t/\tau_c \to 0, S(\omega) = \sigma^2\delta(\omega)$), although I note that it is still Markovian in all cases. Figure 2.2 depicts simulated data and its autocorrelation function and PSD for exemplary parameters: in the white noise limit ($\tau_c = 10^{-2}\Delta t$, blue), in the intermediate regime ($\tau_c = \Delta t$, magenta), and in the correlated regime ($\tau_c = 10^2\Delta t$, green). From the time series plot at the top it becomes clear that the RMS alone is insufficient to describe the properties of noisy signals as the curves differ significantly despite being normalized to their RMS. The autocorrelation functions averaged over 10^3 realizations of the noisy signals as well as their theoretical (continuous) value, Equation 2.16, are plotted in the middle panel, normalized to $\tau_c = \Delta t$ and $\sigma^2 = \Delta t/4$. For the white noise limit (blue), correlations are too short to be resolved with the given time discretization. The correlations decay to e^{-1} at $\tau/\tau_c = 10^{-2}, 1, 10^2$, respectively. Finally, the bottom panel shows the PSD, Equation 2.17, and its periodogram estimate, again averaged over 10^3 realizations of the signal and normalized to $\tau_c = \Delta t$ and $\sigma^2 = \Delta t/4$. The cross-over from white to Brownian PSD occurs at $\omega = \tau_c$. While the simulated data for $\tau_c = 10^{-2}\Delta t$ appears perfectly white, that for $\tau_c = 10^2\Delta t$ appears perfectly $1/f$-like because the spectrum is only white below the smallest resolvable frequency $\Delta f = T^{-1}$.

Having gotten an intuition for the quantities $C(\tau)$ and $S(\omega)$, let us move

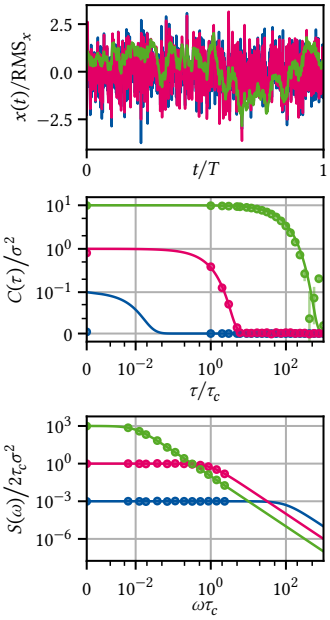

Figure 2.2: Ornstein-Uhlenbeck process. Simulated time traces, autocorrelation function, and PSD of the Ornstein-Uhlenbeck process. Top: Simulated time traces using the algorithm presented in Section A.4. The data are normalized to the computed RMS (equal to σ in the continuous case). Middle: Theoretical autocorrelation function (Equation 2.16, solid lines) computed from the simulated data averaged over 10^3 traces (circles, subset of points). Error bars indicate the standard error of the mean, axes are scaled with respect to the parameters of the magenta data, and data are plotted on an asinh-scale. Bottom: Theoretical PSD (Equation 2.17, solid lines) and periodograms computed from the simulated data using `scipy.signal.periodogram()`, *cf.* Equation 2.21, averaged over 10^3 traces (circles, subset of points). Axes are again scaled with respect to the parameters of the magenta data and plotted on an asinh-scale. Parameters are $\tau_c = \Delta t \times \{10^{-2}, 1, 10^2\}$ and $\sigma^2 = \sqrt{\tau_c}/4$ for blue, magenta, and green data, respectively.

on to see how the latter may be obtained from time series data. Equation 2.11 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[11]

$$x_n, \quad n \in \{0, 1, \dots, N-1\} \tag{2.18}$$

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,[12] we can replace the long-term average in Equation 2.11 by the ensemble average over M realizations of the noisy signal, $\left\{x_n^{(m)}\right\}_m$, and write

$$S_n = \frac{1}{M}\sum_{m=0}^{M-1}\left|\hat{x}_n^{(m)}\right|^2 \tag{2.19}$$

$$= \frac{1}{M}\sum_{m=0}^{M-1} S_n^{(m)} \tag{2.20}$$

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

$$S_n^{(m)} = \left|\hat{x}_n^{(m)}\right|^2, \tag{2.21}$$

and S_n is an *estimate* of the true PSD sampled at the discrete frequencies $\omega_n = 2\pi n/T \in 2\pi\times\{-f_\mathrm{s}/2\,, \dots,\, f_\mathrm{s}/2\}$. Equation 2.19 is known as Bartlett's method [19] for spectrum estimation.[13]

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,

$$f_\mathrm{max} = \frac{f_\mathrm{s}}{2} = \frac{1}{2\Delta t}. \tag{2.22}$$

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

$$f_\mathrm{min} = \Delta f = \frac{1}{T} = \frac{1}{N\Delta t} = \frac{f_\mathrm{s}}{N}. \tag{2.23}$$

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 $x_n, n \in \{0, 1, \dots, NM-1\}$ and partitions these data into M sequences of length N. It becomes clear, then, that the Bartlett average (Equation 2.19) 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 achieved using Welch's method [20]. To see how, we first need to discuss spectral windowing.

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*

11: We only discuss the problem of equally spaced samples here. Variants for spectral estimation of time series with unequal spacing exist [17, 18].

12: Note that the limit of perfectly correlated noise, $S(\omega) \propto \delta(\omega)$, technically does *not* correspond to an ergodic process because $C(\tau) = \text{const.}\ \forall\tau \in (-\infty, \infty)$. In practice, this is always a mathematical idealization and the spectrum is actually better described by a nascent δ-function with a small but finite width.

13: 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 $\Delta t \to 0, N \to \infty, N\Delta t = \text{const.}$

function given by[14]

$$w_n^{(m)} = \begin{cases} 1 & \text{if} \quad (m-1)N \leq n < mN \quad \text{and} \\ 0 & \text{else} \end{cases} \tag{2.24}$$

so that $x_n^{(m)} = x_n w_n^{(m)}$. Now recall that multiplication and convolution are duals under the Fourier transform, implying that

$$\hat{x}_n^{(m)} = \hat{x}_n * \hat{w}_n^{(m)}, \tag{2.25}$$

where the Fourier representation of the rectangular window is given by[15]

$$\hat{w}_n^{(m)} = \hat{w}_n e^{-i(m-1/2)\omega_n T}, \tag{2.26}$$

$$\hat{w}_n = T\,\mathrm{sinc}\left(\frac{\omega_n T}{2}\right). \tag{2.27}$$

Figure 2.3 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, whereas in the continuum limit (see Sidenote 13) Equation 2.27 tends towards $\delta(\omega_n)$ and thus will produce a faithful reconstruction of the true spectrum, the finite sample rate f_s of discrete signals and observation length T induce finite frequency sampling and bandwidth of the probe as well as *sidelobes*.[16] These effects incur what is known as *spectral leakage* and *scalloping loss*, respectively, and lead to artifacts and deviations of the spectrum estimator S_n from the true spectrum $S(\omega_n)$ [14, 21].

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.[17] A window frequently used in spectral analysis is the Hann window [23],

$$w_n^{(m)} = \begin{cases} \sin^2\left(\frac{\pi n}{N}\right) & \text{if} \quad (m-1)N \leq n < mN \quad \text{and} \\ 0 & \text{else,} \end{cases} \tag{2.28}$$

with the Fourier representation of the unshifted window,

$$\hat{w}_n = \frac{T}{2}\,\mathrm{sinc}\left(\frac{\omega_n T}{2}\right)\frac{1}{1-(\omega_n T/2\pi)^2} \tag{2.29}$$

shown in Figure 2.4. The favorable properties of the Hann window are apparent when compared to the rectangular window in Equation 2.27 and Figure 2.3; the sidelobes are quadratically suppressed while the center lobe is broadened by a factor of two.

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.[18]

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

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

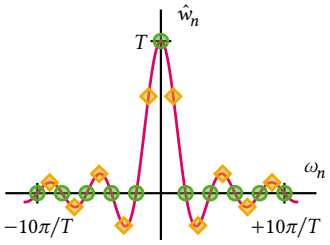

Figure 2.3: The Fourier representation of the rectangular window in continuous time (solid line) and for discrete frequencies $\omega_n = 2\pi n/T$ (circles). Introducing a phase shift, that is, shifting the window with respect to the signal in time, effectively shifts $\omega_n \to \omega_{n+\eta}$ as indicated for $\eta = 1/2$ (diamonds). This incurs scalloping loss.

16: For the optically inclined: this is akin to Fraunhofer diffraction at an aperture.

17: Wikipedia gives a good overview of existing window functions [22].

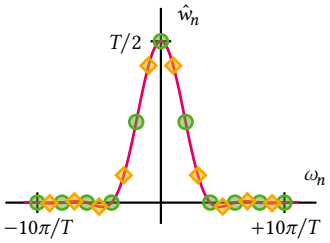

Figure 2.4: The Fourier representation of the Hann window in continuous time (solid line) and for discrete frequencies ω_n (circles). Diamonds indicate discrete sampling when the window completely out of phase with the signal (*cf.* Figure 2.3).

18: Although this can usually also be achieved approximately by detrending

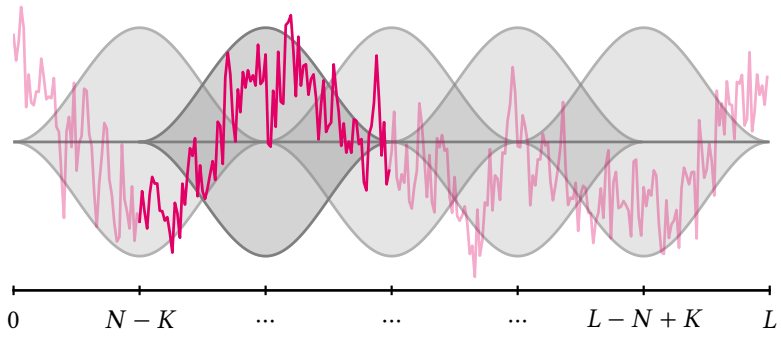

Figure 2.5: Illustration of Welch's method for spectrum estimation. The data (magenta) of length L is partitioned into $K = 2L/N - 1$ segments of length N. Each segment is multiplied with a window function (gray) which reduces spectral leakage and other artifacts. A finite overlap K between adjacent windows (gray) ensures efficient sample use.

2.3 Welch's method

Contemplating Equation 2.28, 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 backward by mK with $K > 0$ the overlap. Finally, the periodogram (Equation 2.21) is computed for each window and subsequently averaged to obtain the spectrum estimator (Equation 2.19).

This method of spectrum estimation is known as Welch's method [20]. One can show [20] 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.[19]

Figure 2.5 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.

19: Again neglecting integer arithmetic issues.

2.4 Parameters & Properties of the PSD

We are now in a position to discuss how the various parameters of a time series relate to both the 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. Table 2.1 gives an overview of all relevant parameters.

To acquire data using some form of (digital) 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 $\Delta t = f_\mathrm{s}^{-1}$ 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 $f_\mathrm{max} \leq f_\mathrm{s}/2$ (*cf.* Equation 2.22, 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

Table 2.1: Overview of spectrum estimation parameters. The parameters can be assigned into four groups: 1. DAQ parameters configuring the data acquisition device, 2. Welch parameters specifying the periodogram averaging, 3. Spectrum properties induced by the above, and 4. External parameters unrelated to the others.

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
4. Miscellaneous parameters	
O	Number of outer averages

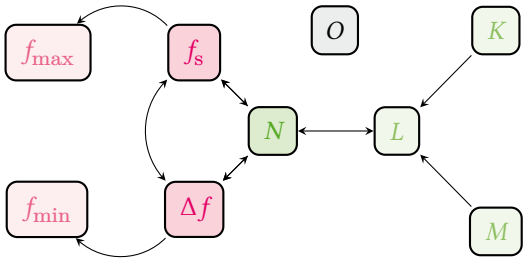

Figure 2.6: Relationships of data acquisition parameters (*cf.* Tables 2.1 and 3.1, with arrows indicating dependencies. N, f_s, and Δf are the central quantities defining the estimated spectrum's properties (darker shades). From f_s and Δf follow (bounds for) $f_{\max}$ and $f_{\min}$ (shaded red, rational numbers). From N, together with K and M, follows L, the total number of samples per data batch (shaded green, integers).

the lower bound on the second parameter characterizing the PSD estimate: the frequency spacing $\Delta f = 1/N \leq f_{\min}$ (*cf.* Equation 2.23).[20] Finally, we can introduce a number of *outer* averages O, that is, the number of data batches that are acquired. While not directly related to Welch's method, choosing $O > 1$ can, for instance, help achieve a certain variance if the number of samples per batch, L, is limited by the data acquisition hardware, or simply allow for updating the spectrum estimate as data is being acquired. Figure 2.6 shows the relationships of the various parameters among each other. In Subsection 3.1.1, I lay out how these inter-dependencies are implemented in software.

20: 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 $f_{\min}$.

To conclude this chapter, let us discuss some of the properties of stochastic processes and their autocorrelation function and PSD. Consider again the process $x(t)$. We say $x(t)$ is *Gaussian* if $x(t) \sim \mathcal{N}(\mu, \sigma^2)\, \forall t$, meaning that the value of $x(t)$ at a given point in time follows a normal distribution with some mean μ and variance σ^2 over multiple realizations of the process. In this case, its statistical properties are fully described by the autocorrelation function $C(\tau)$ and PSD $S(\omega)$. This is because only the first two cumulants of a Gaussian distribution are nonzero [24]. 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 [25].[21] Even if $x(t)$ 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. Let us just note at this point that techniques to estimate higher-order spectra (or *polyspectra*) exist [8, 29–31].

21: As an example, consider electronic devices, where voltage noise is thought to arise 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 [26, 27] (at least for a large density [28]).

For real signals $x(t) \in \mathbb{R}$, the autocorrelation function $C(\tau)$ is an even function, while for $x(t) \in \mathbb{C}$ its real part is even and its complex part odd. From this it immediately follows that for real $x(t)$ $S(\omega)$ is also 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 $x(t)$ such as those generated by LIAs after demodulation in turn have asymmetric, two-sided PSDs. In this chapter so far, we have implicitly employed the two-sided definition, but in the software package presented in Chapter 3, two-sided spectra are used only for complex data since they contain redundant information for real data.

The `python_spectrometer` software package

3

In this chapter, I introduce the `python_spectrometer` Python package[1] [4] and lay out its design and functionality. This software package was developed to make it easier for experimentalists to transfer the mathematical machinery introduced in Chapter 2 to the lab. While in principle the entire process of spectrum estimation from a given array of time series data is already covered by the `welch()` routine in `SciPy` [32], obtaining the data array is not standardized. Different DAQ instruments have different capabilities, both on the hardware and the software level, and different driver interfaces to communicate with them. This implies that custom data acquisition code is required for every instrument, introducing a significant entry barrier to spectral analysis. The `python_spectrometer` package implements a simple interface to different hardware instruments that allows for changing the hardware backend without having to adapt the user-facing code and also incorporates different hardware constraints.

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

What is more, noise spectroscopy tends to be a visual endeavor in practice; it is hard to compare different noise spectra based on quantitative reasoning alone. Data visualization is hence an integral part of noise spectroscopy, but plotting is not just plotting. Do we want the data to be shown on a log-log scale?[2] Do we want to show the relative magnitude of different data sets? Do we want to inspect the time traces as well? The `python_spectrometer` package addresses these questions by allowing users to interactively change features of the main plot window to adapt it to the form best suited to the situation at hand.

2: The short answer is yes, but it comes with visual side-effects that demand other ways of plotting data at times. The long answer is therefore yes, *and* …

Moreover, when concerned with noise spectrum estimation, we are typically more interested in specifying parameters of the resulting PSD rather than parameters of the underlying time series data. The `python_spectrometer` package approaches data acquisition from the inverse direction: rather than inferring the spectrum properties from the time series data, users specify the properties they would like the resulting spectrum to have and the package chooses the correct parameters for data acquisition accordingly.

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.[3] 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.

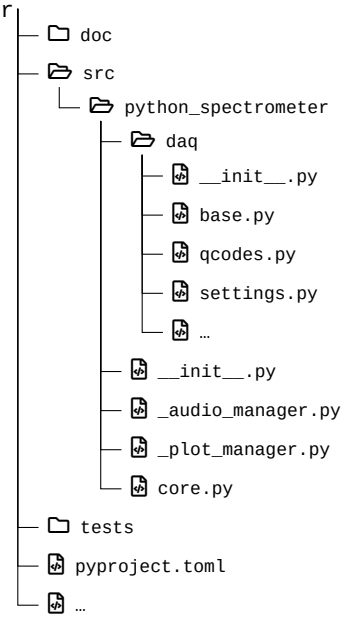

Figure 3.1: Source tree structure of the `python_spectrometer` package. Driver wrappers are placed in the `daq` subpackage. `core.py` exports the `Spectrometer` class.

3: Actually *drivers* to be more precise.

3.1.1 Data acquisition

Figure 3.1 shows the directory structure of the source code. The `daq` subpackage contains on the one hand the declaration of the `DAQ` abstract base class (`base.py`) and its child class implementations (`qcodes.py`, *etc.*), and on the other the `settings.py` module, which defines the `DAQSettings`

class. This class is used in the background to validate data acquisition settings both for consistency (see Section 2.4) and hardware constraints.

To better understand the necessity of this functionality, consider the typical scenario of a physicist[4] 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 $[f_{\mathrm{min}}, f_{\mathrm{max}}]$ she is most interested in. But because she is lazy, 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[5] and attempts to resolve the parameter interdependencies laid out in Section 2.4 and depicted in Figure 2.6 upon calling `DAQSettings.to_consistent_dict()`.[6] 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.

Table 3.1: Variable names used in Chapter 2 and their corresponding parameter names as used in `python_spectrometer` and `scipy.signal.welch()` [32].

Variable	Parameter
L	`n_pts`
N	`nperseg`
K	`noverlap`
M	`n_seg`
O	`n_avg`
f_{s}	`fs`
f_{max}	`f_max`
f_{min}	`f_min`

4: Let's call her Alice.

5: `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.

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

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,[7] she could inspect the parameters after resolution using the code shown in Listing 3.1.

```
>>> 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}
```

7: 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.

If the instrument she'd chosen for data acquisition had been a Zurich Instruments MFLI's Scope module,[8] the same requested settings would have resolved to those shown in Listing 3.2.[9] This is because the Scope module constrains $L \in [2^{12}, 2^{14}]$ and $f_{\mathrm{s}} \in 60\,\mathrm{MHz} \times 2^{[-16,0]} \approx \{915.5\,\mathrm{Hz}, \ldots, 30\,\mathrm{MHz}, 60\,\mathrm{MHz}\}$.

```
{'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`.

8: https://docs.zhinst.com/labone_api_user_manual/modules/scope/index.html

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 `Spectrometer`. 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()` meth-

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

ods. Their functionality is best illustrated by the internal workflow as representatively shown in Listing 3.3.

```
daq = MyDAQ(driver_handle)

parsed_settings = daq.setup(**user_settings)
acquisition_generator = daq.acquire(**parsed_settings)

for data_buffer in acquisition_generator:
    estimate_psd(data_buffer)

# Finalize, clean up, return to user code
...
```

Listing 3.3: DAQ workflow pseudocode. A `MyDAQ` object (representing the instrument `My`) 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).

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,[10] 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.[11] An exemplary implementation of a `DAQ` subclass for a fictitious instrument is shown in Listing 3.4. In addition to the methods to configure the instrument and perform data acquisition, it is possible to override the `DAQSettings` property to implement instrument-specific hardware constraints such as, in this example, the number of samples per buffer being constrained to the discrete interval $[1, 2048]$. Leveraging the `qutil.domains` module [33], more complex constraints such as sample rates restricted to an internal clock rate divided by a power of two[12] can be specified.

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

11: *I. e.*, the number of time series data batches acquired, as opposed to the number of Welch averages `n_seg` within one batch.

12: See for example the implementation of the AlazarTech ATS9440 digitizer card.

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 2.19. 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 the voltage back to a current,[13] 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.

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

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`, taking as arguments the time series data and arbitrary settings passed through from the `take()` method, and returns the processed data. Listing 3.5 shows a simple function that accounts for the gain of an amplifier. 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()` [32]. Here, the keyword `fourier_procfn` should be a callable with signature `(xf, f, **kwargs) -> (xfp, fp)`.[14] That is, it should

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

```python
# daq/mydaq.py
import dataclasses
from qutil.domains import DiscreteInterval
from .base import DAQ
from .settings import DAQSettings

@dataclasses.dataclass
class MyDAQ(DAQ):
    handle: mydriver.DeviceHandle

    @property
    def DAQSettings(self) -> type[DAQSettings]:
        class MyDAQSettings(DAQSettings):
            ALLOWED_NPERSEG = DiscreteInterval(1, 2048)
        return MyDAQSettings

    def setup(self, **settings) -> dict:
        settings = self.DAQSettings(settings)
        parsed_settings = settings.to_consistent_dict()
        self.handle.configure(parsed_settings)
        return parsed_settings

    def acquire(self, n_avg: int, *, **settings) -> Generator:
        self.handle.arm(n_avg)
        for _ in range(n_avg):
            self.handle.wait_for_trigger()
            yield self.handle.fetch()
        return self.handle.metadata
```

Listing 3.4: Exemplary code for a `DAQ` implementation of some instrument with given driver class `DeviceHandle` in the package `mydriver`. The `MyDAQ` class is instantiated with a `DeviceHandle` instance. Optionally, the `DAQSettings` property can be overridden to implement hardware constraints or default values for data acquisition parameters. For this, the `qutil.domains` module provides several classes that represent bounded domains and sets. The `setup()` method parses the given acquisition settings and configures the instrument through the external driver interface `handle.configure()`. The `acquire()` method arms the instrument (if necessary) and loops over the number of outer averages, `n_avg`. In the body of the loop, it can wait for external triggers (or send software triggers) before yielding a batch of data fetched from the external driver interface. Once acquisition is done, the method can return arbitrary metadata to the `Spectrometer` object to attach to the stored data.

take the frequency-space data, the corresponding frequencies, and arbitrary keyword arguments and return a tuple of the processed data and the corresponding frequencies.

A simple example for a processing function in Fourier space is shown in Listing 3.6, which computes the (anti-)derivative of the data using the fact that

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

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

```
def comp_gain(x, gain=1.0, **_):
    return x / gain
```

Listing 3.5: A simple `procfn`, which converts amplified data back to the level before amplification. Note the token `**_` variable keyword argument that ensures no errors arise from other parameters being passed to the function. 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.

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

Listing 3.6: A simple `fourier_procfn`, which calculates the (anti-)derivative.

3.2 Feature overview

Now that we have a basic understanding of the design choices underlying `python_spectrometer`, let us discuss the typical workflow of using the package. Two modes of operation are to be distinguished: first, "serial" mode, in which users record new spectra manually, and second, "live" mode, in which new data is continuously being acquired. The former is well suited to a structured approach to noise spectroscopy where data is retained persistently and discrete changes are made to the system in between subsequent data acquisitions. The latter is aimed at a more fluent workflow in which data is not retained and data acquisition runs in the background.

3.2.1 Serial spectrum acquisition

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 be assigned a comment that allows to easily identify it 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,[15] having discovered that his data are noisier than expected, sets up the `Spectrometer` class with an instance of the `DAQ` subclass for the DAQ instrument connected to his sample, a Zurich Instruments MFLI.[16] Choosing to work with the demodulated data to benefit from the corresponding DAQ module's larger flexibility, he recognizes that the resulting PSD will be the two-sided version because the data returned by the LIA is complex. Since he is interested in the physical frequencies[17] he sets the lock-in's modulation frequency to 0 Hz and disables plotting the negative part of the frequency spectrum as it contains only redundant data in this case. Selecting the frequency bounds, say $f_{\text{min}} = 10\,\text{Hz}$ and $f_{\text{max}} = 100\,\text{kHz}$, and using the sensible defaults for the remaining spectrum parameters, Bob first grounds the input of his DAQ to record a *baseline* spectrum. Thus far, his code would hence look something like that shown in Listing 3.7, which produces the plot shown in Figure 3.2.

The noise spectrum he obtains is white up until approximately 20 kHz where it starts falling off $\propto f^{-n}$. This is because the LIA low-pass filters the signal to suppress aliasing.[18] 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

15: Let's call him Bob.

16: For the MFLI, `DAQ` subclasses for both the Scope and the DAQ module are implemented. The former gives access to the signal before and the latter to the signal after demodulation.

17: A discussion of lock-in amplification is beyond the scope of this chapter. Here I will simply note that, for finite modulation frequencies f_m, LIAs will measure the PSD in the up-converted frequency band $[-f_{\text{max}} + f_m, f_m + f_{\text{max}}]$ rather than the baseband.

18: Aliasing effects arise from finite sampling according to the Nyquist-Shannon sampling theorem [34–36]. It states that for a given physical bandwidth, the sampling rate f_s must be at least twice as large to faithfully reconstruct the signal in order to avoid aliasing, *i.e.*, the reverse of the argument we have made for the largest resolvable frequency f_{max}. Some DAQ devices perform internal aliasing rejection while others do not.

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

mfli_daq = daq.ZurichInstrumentsMFLIDAQ(session, device)
spect = Spectrometer(mfli_daq, procfn=scaled(1e6),
                     plot_negative_frequencies=False,
                     processed_unit='µV')

settings = {'f_min': 1e1, 'f_max': 1e5, 'freq': 0}
spect.take('baseline', daq='grounded', **settings)
```

Listing 3.7: Setup and serial 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.

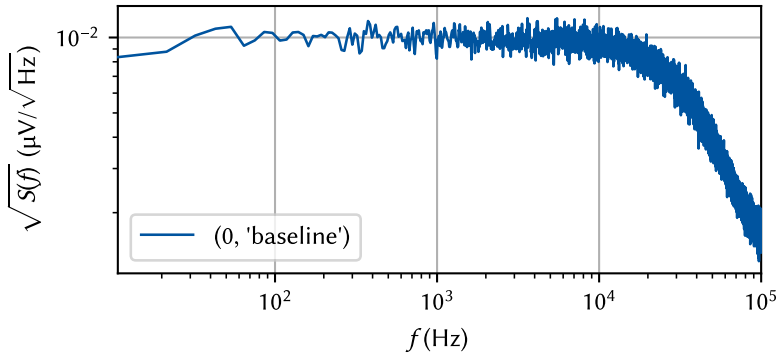

Figure 3.2: The `python_spectrometer` plot after acquiring the (here: synthetic) baseline spectrum. By default, the amplitude spectral density (ASD) = √PSD is displayed in the main plot. Each spectrum is assigned a unique identifier key consisting of an incrementing integer and the user comment, and can be referred to by either (or both) when interacting with the object.

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. The code shown in Listing 3.8 would then leave him with the spectrometer plot as shown in Figure 3.3. While working, Bob 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 the spectrometer figure as shown in Figure 3.4. Since his DAQ returns complex data, the absolute value $R = X + iY$ is plotted.

```python
settings['daq'] = 'connected'
spect.take('connected', **settings)
spect.take('lifted cable', cable='lifted', **settings)
spect.take('jumped', **settings)
```

Listing 3.8: 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.

Bob now observes that the noise spectra he has recorded 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. Bob's spectrometer figure after setting `spect.plot_cumulative = True` is shown in Figure 3.5. In the case that `spect.plot_amplitude == True`, this new subplot shows the RMS in the band $[f_{\min}, f]$,

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

and the band power (Equation 2.14) otherwise.

The cumulative RMS plot already helps, but Bob would like a more quantitative comparison of relative spectral powers. Therefore, he rescales the spectra in terms of their relative powers expressed in dB[19] by applying the following settings, which changes the plot to the layout shown

19: Recall that the decibel is defined by the ratio L_P of two powers P_1, P_2 as [37]

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

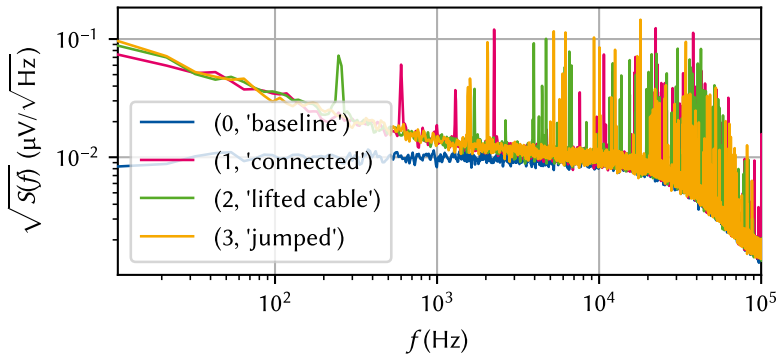

Figure 3.3: The `python_spectrometer` plot after acquiring additional (synthetic) spectra. Each spectrum is uniquely identified by a two-tuple of `(index, comment)`.

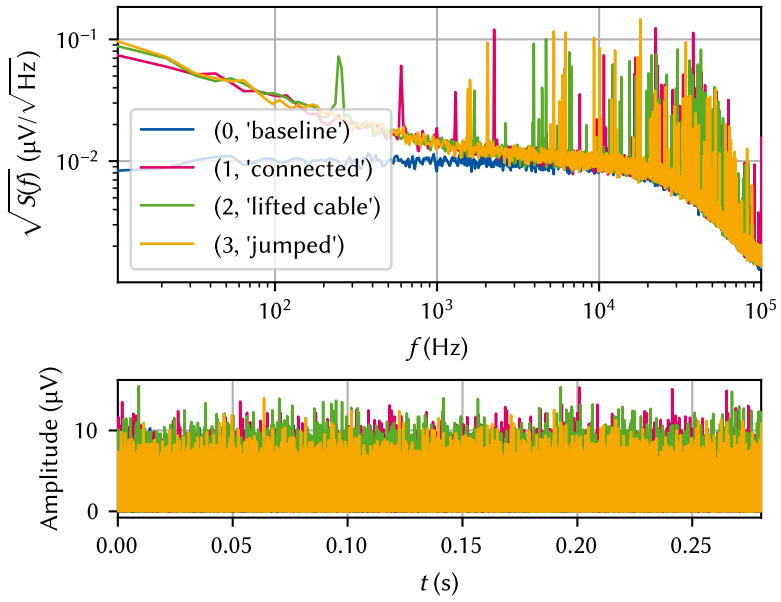

Figure 3.4: The `python_spectrometer` plot shown in Figure 3.3 when setting `spect.plot_timetrace = True`. This adds a subplot that shows the time series data from which the PSD was computed akin to what an oscilloscope would show. For complex time series, the absolute value $R = X + iY$ is plotted. 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.

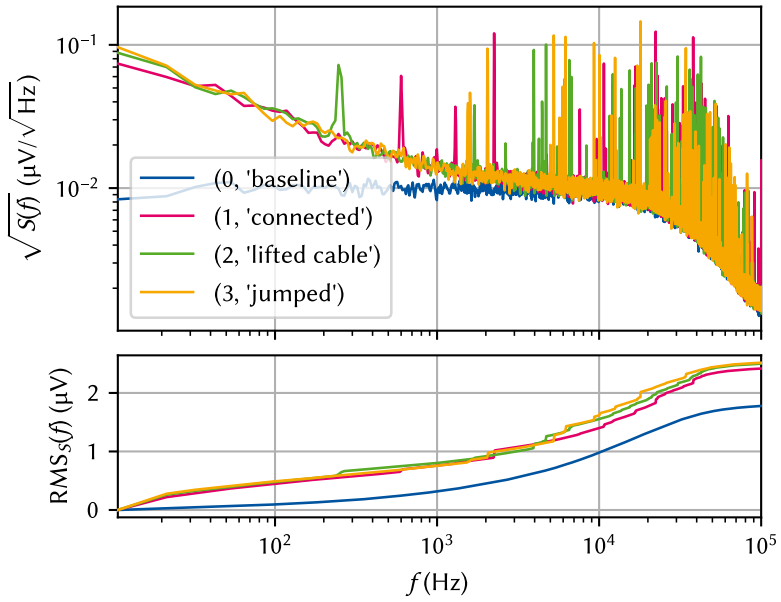

Figure 3.5: The `python_spectrometer` plot shown in Figure 3.3 when setting `spect.plot_cumulative = True`. This adds a subplot that shows the RMS (see Equation 2.14) which can be helpful in evaluating the contribution of individual peaks in the spectrum to the total noise power. Both the oscilloscope subplot (Figure 3.4) and the RMS subplot can also be shown at the same time.

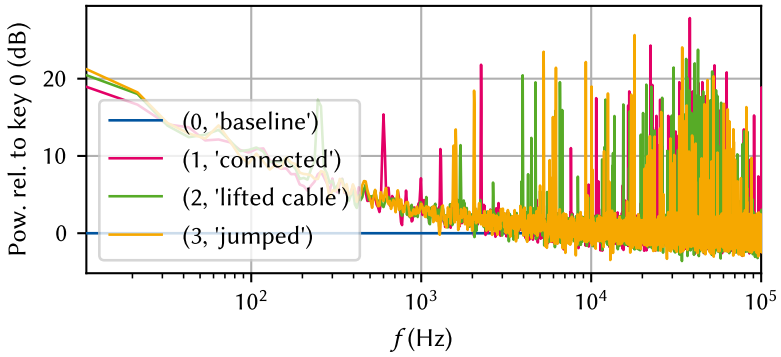

Figure 3.6: The `python_spectrometer` plot in relative mode. Starting from the state in Figure 3.3, we set `spect.plot_dB_scale = True` as well as `spect.plot_amplitude = False` and `spect.plot_density = False` to compare the relative noise powers with respect to the baseline.

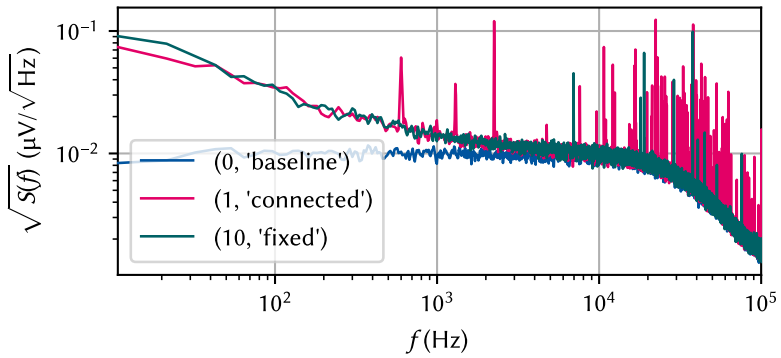

Figure 3.7: The `python_spectrometer` plot after multiple additional spectra were acquired and hidden. Hiding spectra that one is not interested in anymore is achieved through `spect.hide(*range(2, 10))`. This is reversed by `spect.show(*range(2, 10))`. Data can also be dropped (`spect.drop(key)`) or deleted (`spect.delete(key)`) from the internal cache and disk, respectively.

in Figure 3.6:

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

The attribute `plot_density` controls whether the *power spectral density* or the *power spectrum* is plotted.[20] Scaling the data to the power spectrum instead of the density, Bob 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.

Bob 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 3.7.

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

```
spect.serialize_to_disk('2032-12-24_noise_hunting')
```

The next week, Bob 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, Bob simply uses his laptop

20: At this point, we *do* need to distinguish between the PSD and the power spectrum counter to Sidenote 10 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 [21],

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

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

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

This opens up the plot shown in Figure 3.7 again. While they cannot acquire new spectra in this state,[21] they can still use all the plotting features like showing or hiding spectra, or changing plot types as discussed above.

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

3.2.2 Live spectrum acquisition

Manually recording spectra in the workflow outlined in Subsection 3.2.1 becomes tedious at some point, and experimenters tend to become negligent with keeping metadata and comments up to date as they continue to change settings. Once a certain number of spectra has been obtained, the spectrometer plot also becomes crowded, and hiding old spectra manually is cumbersome. Moreover, each time a spectrum is captured, data is saved to disk, potentially accruing large amounts of storage space. For these reasons, the python_spectrometer package also offers a non-persistent live mode for displaying spectra continuously.

This mode is facilitated by the qutil.plotting.live_view module that provides asynchronous plotting functionality based on matplotlib. The live_view module supports both the multithreading and multiprocessing paradigms for concurrency in order to keep the interpreter responsive.[22] In the former case, the window hosting the figure runs in the main thread and is kept responsive using matplotlib's graphical user interface (GUI) event loop mechanisms. In the latter, the plotting takes place in a separate process, resulting in true parallelism.

22: Note that technically, data is also recorded in a background thread in serial mode by default.

The live mode is started with the Spectrometer.live_view() method. Data is continuously acquired in a background thread using the same DAQ interface as the serial mode. Instead of saving the data on disk and managing plotting from python_spectrometer, however, the data is fed into a queue.[23] A live_view.IncrementalLiveView2D object then retrieves the data from the queue and handles plotting in the GUI event loop. To start a spectroscopy session with the same parameters as in Subsection 3.2.1 with a given Spectrometer object (see Listing 3.7), we would call

23: A queue is a concurrency mechanism for exchanging data between multiple threads or processes.

```
view = spect.live_view(f_min=1e1, f_max=1e5, in_process=True)
```

which would open a figure window such as that shown in Figure 3.8. Similar to take(), the data acquisition parameters are passed to live_view() as keyword arguments.[24] The in_process argument specifies if multiprocessing (True) or multithreading (False) is used. Dictionaries with customization parameters for the live_view object can further be passed to the method. Functionality to take snapshots to retain live data is currently not implemented but would be a valuable addition to the noise-hunting workflow.

24: The difference is, of course, that we do not need to specify a comment since no data is retained.

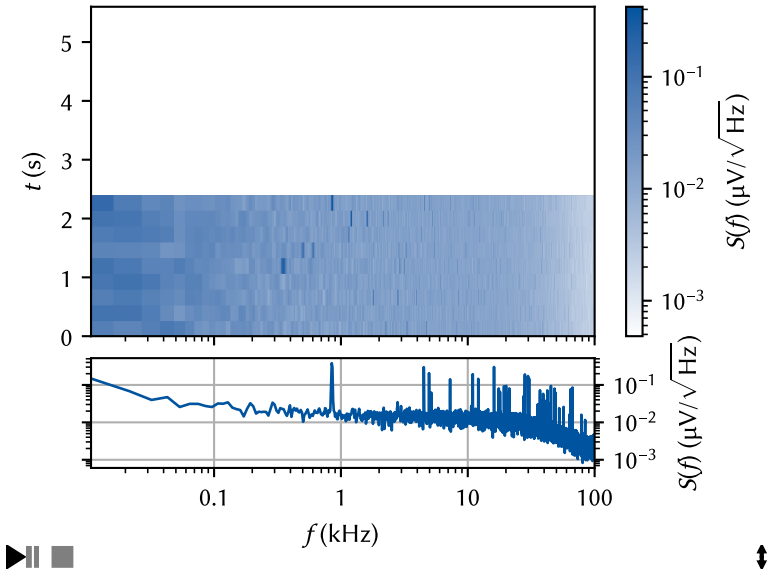

Figure 3.8: Spectrometer live view. The bottom plot shows the most recently acquired spectrum, while the top plot shows a waterfall plot of the most recent ones. Data acquisition runs in a background thread, keeping the interpreter responsive and available to interact with instruments, for example. The icons in the bottom left and right corners allow interacting with the live view.

Conclusion and outlook 4

In this part of the present thesis, I introduced the `python_spectrometer` Python package for interactive, backend-agnostic noise spectroscopy. I first presented the theory behind spectral noise estimation based on time series analysis in Chapter 2. There, I showed how, by the Wiener-Khinchin theorem, the PSD $S(\omega)$ is related to a time-dependent signal by the Fourier transform of its autocorrelation function $C(\tau)$. After discretizing the problem, I discussed how the arising spectral windowing introduces distortions in the estimated PSD through effects such as spectral leakage and scalloping loss. The introduction of windowing naturally led to the need for a more efficient method of spectrum estimation to avoid data wastefulness. This is addressed by Welch's method, which trades spectral resolution for estimation accuracy for a given input size by dividing the input into segments and letting them overlap. Finally, I discussed the various parameters that influence the properties of the spectrum estimate produced by Welch's method.

In Chapter 3, I then introduced the `python_spectrometer` package that facilitates noise spectroscopy based on the theory developed in the previous chapter. While in principle agnostic to the specific technique (simple periodogram, Bartlett's method, *etc.*), it employs Welch's method by default to perform efficient estimation of noise power spectral densities. To make noise spectroscopy as approachable as possible, the package provides a unified interface to various hardware and software backends to manage data acquisition. Furthermore, it incorporates rich plotting features to ease data analysis and thereby expedite the feedback loop with the lab. I first outlined the design choices underlying the source code and explained the mechanism employed to infer DAQ settings from, for example, the physical parameters of the resulting spectrum estimate. I described the interface to drivers for different hardware instruments that makes it possible to use the same user code for spectrum acquisition independent of the specific instrument used for data acquisition while at the same time keeping the amount of required driver code at a minimum. Finally, I showcased the interactive features by means of a typical workflow. Here, the serial approach of recording single spectra that are saved to disk at a time stands in contrast to the non-persistent live mode, where data is continuously acquired and displayed in a separate thread or process.

There are several possible avenues for future development of the `python_spectrometer` package. An obvious case is adding support for more DAQ hardware instruments by implementing `DAQ` interfaces. Modular devices such as those offered by QBLOX[1] and Quantum Machines[2] are on track to become the new standard in quantum technology labs. Implementing drivers for these instruments would benefit the adoption of both the instruments and the `python_spectrometer` package. Another valuable addition would be supporting generic instruments abstracted by `QuMADA` [38]. `QuMADA` is a `QCoDeS`-based measurement framework that provides a unified interface to instruments in order to – in a similar spirit to `python_spectrometer`'s `DAQ` interface – abstract away internals of individual instruments and provide users with a standardized way to interact with them. This approach should naturally lend itself to a single implementation of the `DAQ` class supporting various instruments through the unified `QuMADA` interface.

1: https://www.qblox.com/research

2: https://www.quantum-machines.co/

Next, incorporating noise spectroscopy into the standard measurement workflow of quantum device experiments would allow experimentalists to quickly gauge noise levels as they are performing measurements. If for some reason the noise changed[3] the experimentalist could quickly obtain insight into the noise by analyzing the spectrum. In a client-server architecture, which is inherently asynchronous, such as Zurich Instrument's LabOne[4] software, this is already possible using the web interface. But of course the strength of the `python_spectrometer` package stems from its capacity to be utilized in conjunction with any hardware instrument.

One way to implement such functionality would be to introduce a proxy `DAQ` subclass to be used together with the live mode presented in Subsection 3.2.2. This proxy class would serve as an interface to external measurement software and expose two attributes; first, a data queue, into which the external code could place arbitrary time series data that was obtained during some measurement, and second, a shared dictionary to hold acquisition parameters as these might change between measurements. Because the live view mode runs in the background, the external measurement framework could push data to the queue whenever new data was taken without obstructing the measurement workflow.

Listing 4.1 shows a template design for such a `TeeDAQ` class.[5] The `setup()` method ignores the input parameters and instead obtains the current settings from the shared `settings` proxy. Similarly, instead of fetching data from an instrument itself, the `acquire()` method attempts to fetch data from the shared queue and blocks the thread if no data is present, thereby efficiently idling and consuming no resources unless triggered by the external caller. A measurement framework would then interact with the `TeeDAQ` object as exemplarized by the following code:

```
daq = TeeDAQ(...)
spect = Spectrometer(daq)
view = spect.live_view()
...
data = measure(fs, n_pts)
daq.settings.update(fs=fs, n_pts=n_pts)
daq.queue.put(data)
```

Measurement frameworks integrating with this interface could thus provide experimentalists live feedback on current noise levels with negligible overhead and minimal code adaptation.

Finally, it might be useful to not only allow estimating PSDs but also cross power spectral densities (CSDs) or *cross-spectra*. The cross-spectrum[6] is the Fourier transform of not the autocorrelation but the cross-correlation function $C(\tau)$ (Equation 2.4) between two random processes. Take a set of processes $\{x_1(t), x_2(t), \dots, x_n(t)\}$ that correspond to noise measured at different locations in a sample. The cross-correlation function between variables x_i and x_j is then given by[7]

$$C_{ij}(\tau) = \left\langle x_i(t)^* x_j(t+\tau) \right\rangle . \tag{4.1}$$

This function (and its Fourier pair the cross-spectrum $S_{ij}(\omega)$) quantifies the degree of correlation between noise at site i and noise at site j. Unlike the *auto*-spectrum (or self-spectrum), the cross-spectrum is always a complex quantity, even for real $x_i(t)$. It is not hard to see that for quantum processors, for example, these kinds of correlations could have significant impact on operation, and on error correction in particular [39–41]. To incorporate cross-spectra in the `python_spectrometer` package, only small changes should be necessary.

3: As it happens often, unfortunately.

4: https://www.zhinst.com/ch/en/instruments/labone/labone-instrument-control-software

```
# daq/tee.py
import dataclasses
import threading
import multiprocessing as mp

from .base import DAQ

@dataclasses.dataclass
class TeeDAQ(DAQ):
    settings:
     ↪ mp.managers.DictProxy
    queue: mp.JoinableQueue
    event: threading.Event

    def setup(self, **_):
        settings = self.
         ↪ DAQSettings(self.
         ↪ settings)
        return settings.
         ↪ to_consistent_dict()

    def acquire(self, **_):
        while not
         ↪ self.event.is_set():
            yield self.queue.
             ↪ get(block=True)
```

Listing 4.1: Template design for a proxy `DAQ` implementation to stream noise spectra from an external measurement framework. The `settings` attribute is a dictionary proxy shared between processes and used to pass acquisition parameters from the measurement framework to `python_spectrometer`.

5: The UNIX shell command `data = $(measure(**settings) | tee daq)` should give an indication as to the naming.

6: Again, we use the two terms interchangably unless otherwise indicated, see Sidenote 10 in Chapter 2.

7: Again assuming wide-sense stationary processes.

First, the data acquisition logic would need to be adapted. Two possible routes suggest themselves here; first, specialized `DAQ` classes could be implemented that, in place of yielding one batch of time series data, yield two batches each time they are queried. This approach first of all requires instruments with multiple channels,[8] which is not necessarily given. Furthermore, it would incur additional coding efforts by having to re-implement each `DAQ` class for cross-spectra. On the other hand, it would arguably make synchronization between channels easier to achieve.

8: If one sticks to single `DAQ` instances managing single instruments.

A less involved path would adapt the `Spectrometer` class to work with multiple `DAQ`s. This would not involve additional driver work[9] and allow the `Spectrometer` object to ensure synchronization between the different `DAQ`s. The internal workflow shown in Listing 3.3 would then need to be slightly adapted to the code shown in Listing 4.2. The downside of this approach is that synchronization of different instruments or channels would need to be taken care of externally.

9: Except possibly ensuring thread safety and timing synchronization if multiple `DAQ` objects communicate with the same physical instrument.

```
daq_1 = MyDAQ(driver_handle, channel=1)
# Or MyOtherDAQ(driver_handle_2) if another instrument
daq_2 = MyDAQ(driver_handle, channel=2)
daqs = (daq_1, daq_2)

parsed_settings = [daq.setup(**user_settings) for daq in daqs]
assert all_equal(parsed_settings), "DAQ settings do not match"

acquisition_generators = [daq.acquire(**parsed_settings[0])
                          for daq in daqs]
for data_buffers in zip(*acquisition_generators):
    estimate_csd(*data_buffers)
```

Listing 4.2: Proposed `DAQ` workflow for estimating cross-spectra. Each hardware channel (same or different instruments) is assigned to a `DAQ` object. After instrument configuration, it is asserted that the parameters match. Finally, data is fetched from both channels and fed into a CSD estimator. Note that triggering would need to be implemented externally.

Further code adaptations would involve minor changes such as replacing the spectrum estimator with `scipy.signal.csd()` [42] for CSD estimation[10] and making the plotting conform to complex data.[11]

10: In practice, working with the normalized CSD, or correlation coefficient [43, 44]

$$r_{ij}(\omega) = \frac{S_{ij}(\omega)}{\sqrt{S_{ii}(\omega) S_{jj}(\omega)}} \tag{4.2}$$

with $S_{ii}(\omega)$ the PSD of process x_i would likely be more favorable.

11: It could be worthwile to add a subplot to display both the magnitude and phase of the complex quantity $S_{ij}(\omega)$.

Part II

Characterization and Improvements of a Millikelvin Confocal Microscope

Part III

Optical Measurements of Electrostatic Exciton Traps in Semiconductor Membranes

Part IV

A Filter-Function Formalism For Unital Quantum Operations

Appendix

Bibliography

[1] Jonathan P. Dowling and Gerard J. Milburn. "Quantum Technology: The Second Quantum Revolution." In: *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 361.1809 (Aug. 15, 2003). Ed. by A. G. J. MacFarlane, pp. 1655–1674. DOI: 10.1098/rsta.2003.1227 (cited on page 2).

[2] Alain Aspect. *Einstein and the Quantum Revolutions*. Trans. by David Kaiser Translated by Teresa Lavender Fagan. The France Chicago Collection. Chicago, IL: University of Chicago Press, Oct. 2024. 112 pp. (cited on page 2).

[3] A. A. Clerk et al. "Introduction to Quantum Noise, Measurement, and Amplification." In: *Reviews of Modern Physics* 82.2 (Apr. 15, 2010), pp. 1155–1208. DOI: 10.1103/RevModPhys.82.1155 (cited on pages 2, 4).

[4] Tobias Hangleiter et al., *Python-Spectrometer* version 2024.11.1, Nov. 21, 2024. Zenodo. DOI: 10.5281/ZENODO.13789861, (cited on pages 3, 11).

[5] Gerardo A. Paz-Silva, Leigh M. Norris, and Lorenza Viola. "Multiqubit Spectroscopy of Gaussian Quantum Noise." In: *Physical Review A* 95.2 (Feb. 23, 2017), p. 022121. DOI: 10.1103/PhysRevA.95.022121 (cited on page 4).

[6] Eric W. Weisstein. *Root-Mean-Square*. URL: https://mathworld.wolfram.com/Root-Mean-Square.html (visited on 04/21/2025) (cited on page 4).

[7] Gonzalo A. Álvarez and Dieter Suter. "Measuring the Spectrum of Colored Noise by Dynamical Decoupling." In: *Physical Review Letters* 107.23 (Nov. 30, 2011), p. 230501. DOI: 10.1103/PhysRevLett.107.230501 (cited on page 4).

[8] P Szańkowski et al. "Environmental Noise Spectroscopy with Qubits Subjected to Dynamical Decoupling." In: *Journal of Physics: Condensed Matter* 29.33 (Aug. 23, 2017), p. 333001. DOI: 10.1088/1361-648X/aa7648 (cited on pages 4, 10).

[9] O. E. Dial et al. "Charge Noise Spectroscopy Using Coherent Exchange Oscillations in a Singlet-Triplet Qubit." In: *Physical Review Letters* 110.14 (Apr. 5, 2013), p. 146804. DOI: 10.1103/PhysRevLett.110.146804 (cited on page 4).

[10] Elliot J. Connors et al. "Charge-Noise Spectroscopy of Si/SiGe Quantum Dots via Dynamically-Decoupled Exchange Oscillations." In: *Nature Communications* 13.1 (Dec. 2022), p. 940. DOI: 10.1038/s41467-022-28519-x (cited on page 4).

[11] Norbert Wiener. "Generalized Harmonic Analysis." In: *Acta Mathematica* 55 (none Jan. 1930), pp. 117–258. DOI: 10.1007/BF02546511 (cited on pages 4, 6).

[12] A. Khintchine. "Korrelationstheorie der stationären stochastischen Prozesse." In: *Mathematische Annalen* 109.1 (Dec. 1, 1934), pp. 604–615. DOI: 10.1007/BF01449156 (cited on pages 4, 6).

[13] Arian Vezvaee et al. "Fourier Transform Noise Spectroscopy." In: *npj Quantum Information* 10.1 (May 17, 2024), pp. 1–12. DOI: 10.1038/s41534-024-00841-w (cited on page 5).

[14] 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).

[15] G. E. Uhlenbeck and L. S. Ornstein. "On the Theory of the Brownian Motion." In: *Physical Review* 36.5 (Sept. 1, 1930), pp. 823–841. DOI: 10.1103/PhysRev.36.823 (cited on page 6).

[16] N. G Van Kampen. "Stochastic Differential Equations." In: *Physics Reports* 24.3 (Mar. 1, 1976), pp. 171–228. DOI: 10.1016/0370-1573(76)90029-6 (cited on page 6).

[17] N. R. Lomb. "Least-Squares Frequency Analysis of Unequally Spaced Data." In: *Astrophysics and Space Science* 39.2 (Feb. 1, 1976), pp. 447–462. DOI: 10.1007/BF00648343 (cited on page 7).

[18] J. D. Scargle. "Studies in Astronomical Time Series Analysis. II. Statistical Aspects of Spectral Analysis of Unevenly Spaced Data." In: *The Astrophysical Journal* 263 (Dec. 1, 1982), pp. 835–853. DOI: 10.1086/160554 (cited on page 7).

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

[20] 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 Transactions on Audio and Electroacoustics* 15.2 (June 1967), pp. 70–73. DOI: 10.1109/TAU.1967.1161901 (cited on pages 7, 9).

[21] F.J. Harris. “On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform.” In: *Proceedings of the IEEE* 66.1 (Jan. 1978), pp. 51–83. DOI: 10.1109/PROC.1978.10837 (cited on pages 8, 18).

[22] Wikipedia contributors. *Window Function*. Wikipedia, The Free Encyclopedia. Mar. 20, 2025. URL: https://en.wikipedia.org/w/index.php?title=Window_function&oldid=1281514999 (visited on 03/27/2025) (cited on page 8).

[23] A. Nuttall. “Some Windows with Very Good Sidelobe Behavior.” In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 29.1 (Feb. 1981), pp. 84–91. DOI: 10.1109/TASSP.1981.1163506 (cited on page 8).

[24] Ronald Forrest Fox. “Gaussian Stochastic Processes in Physics.” In: *Physics Reports* 48.3 (Dec. 1978), pp. 179–283. DOI: 10.1016/0370-1573(78)90145-X (cited on page 10).

[25] Jan Krzywda and Łukasz Cywiński. “Adiabatic Electron Charge Transfer between Two Quantum Dots in Presence of 1/f Noise.” In: *Physical Review B* 101.3 (Jan. 15, 2020), p. 035303. DOI: 10.1103/PhysRevB.101.035303. arXiv: 1909.11780 (cited on page 10).

[26] Josef Schriefl et al. “Decoherence from Ensembles of Two-Level Fluctuators.” In: *New Journal of Physics* 8 (Jan. 20, 2006), pp. 1–1. DOI: 10.1088/1367-2630/8/1/001 (cited on page 10).

[27] Félix Beaudoin and W. A. Coish. “Microscopic Models for Charge-Noise-Induced Dephasing of Solid-State Qubits.” In: *Physical Review B* 91.16 (Apr. 29, 2015), p. 165432. DOI: 10.1103/PhysRevB.91.165432. arXiv: 1412.5536 (cited on page 10).

[28] 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: *Physical Review Research* 6.3 (Aug. 15, 2024), p. 033175. DOI: 10.1103/PhysRevResearch.6.033175 (cited on page 10).

[29] C.L. Nikias and J.M. Mendel. “Signal Processing with Higher-Order Spectra.” In: *IEEE Signal Processing Magazine* 10.3 (July 1993), pp. 10–37. DOI: 10.1109/79.221324 (cited on page 10).

[30] V. Chandran and S. Elgar. “A General Procedure for the Derivation of Principal Domains of Higher-Order Spectra.” In: *IEEE Transactions on Signal Processing* 42.1 (Jan. 1994), pp. 229–233. DOI: 10.1109/78.258147 (cited on page 10).

[31] Leigh M. Norris, Gerardo A. Paz-Silva, and Lorenza Viola. “Qubit Noise Spectroscopy for Non-Gaussian Dephasing Environments.” In: *Physical Review Letters* 116.15 (Apr. 14, 2016), p. 150503. DOI: 10.1103/PhysRevLett.116.150503 (cited on page 10).

[32] *welch — SciPy v1.15.2 Manual*. URL: https://docs.scipy.org/doc/scipy-1.15.2/reference/generated/scipy.signal.welch.html (visited on 03/31/2025) (cited on pages 11–13).

[33] Tobias Hangleiter et al., *Qutil* version 2024.11.1, Nov. 21, 2024. Zenodo. DOI: 10.5281/ZENODO.14200303, (cited on page 13).

[34] E. T. Whittaker. “XVIII.—On the Functions Which Are Represented by the Expansions of the Interpolation-Theory.” In: *Proceedings of the Royal Society of Edinburgh* 35 (Jan. 1915), pp. 181–194. DOI: 10.1017/S0370164600017806 (cited on page 15).

[35] H. Nyquist. “Certain Topics in Telegraph Transmission Theory.” In: *Transactions of the American Institute of Electrical Engineers* 47.2 (Apr. 1928), pp. 617–644. DOI: 10.1109/T-AIEE.1928.5055024 (cited on page 15).

[36] C.E. Shannon. “Communication in the Presence of Noise.” In: *Proceedings of the IRE* 37.1 (Jan. 1949), pp. 10–21. DOI: 10.1109/JRPROC.1949.232969 (cited on page 15).

[37] David M. Pozar. *Microwave Engineering*. Fourth edition. Hoboken, NJ: John Wiley & Sons, Inc, 2012. 1 p. (cited on page 16).

[38] Till Huckemann. PhD thesis. Aachen: RWTH Aachen University, 2025 (cited on page 21).

[39] Dorit Aharonov, Alexei Kitaev, and John Preskill. "Fault-Tolerant Quantum Computation with Long-Range Correlated Noise." In: *Physical Review Letters* 96.5 (Feb. 7, 2006), p. 050504. DOI: 10.1103/PhysRevLett.96.050504 (cited on page 22).

[40] Naomi H. Nickerson and Benjamin J. Brown. "Analysing Correlated Noise on the Surface Code Using Adaptive Decoding Algorithms." In: *Quantum* 3 (Apr. 8, 2019), p. 131. DOI: 10.22331/q-2019-04-08-131. arXiv: 1712.00502 (cited on page 22).

[41] B. D. Clader et al. "Impact of Correlations and Heavy Tails on Quantum Error Correction." In: *Physical Review A* 103.5 (May 24, 2021), p. 052428. DOI: 10.1103/PhysRevA.103.052428 (cited on page 22).

[42] *csd — SciPy v1.15.2 Manual*. URL: https://docs.scipy.org/doc/scipy-1.15.2/reference/generated/scipy.signal.csd.html (visited on 04/15/2025) (cited on page 23).

[43] J.S. Rojas-Arias et al. "Spatial Noise Correlations beyond Nearest Neighbors in ^{28}Si/Si-Ge Spin Qubits." In: *Physical Review Applied* 20.5 (Nov. 9, 2023), p. 054024. DOI: 10.1103/PhysRevApplied.20.054024 (cited on page 23).

[44] J. Yoneda et al. "Noise-Correlation Spectrum for a Pair of Spin Qubits in Silicon." In: *Nature Physics* (Oct. 9, 2023). DOI: 10.1038/s41567-023-02238-6 (cited on page 23).

Glossary

A
API application programming interface. 16
ASD amplitude spectral density. 16

C
CSD cross power spectral density. 22, 23

D
DAQ data acquisition. 3, 9–13, 15, 16, 21
DD dynamical decoupling. 4
DUT device under test. 4

E
ENBW equivalent noise bandwidth. 18

F
FFT fast Fourier transform. 10

G
GKSL Gorini-Kossakowski-Sudarshan-Lindblad. ix
GRAPE gradient ascent pulse engineering. ix
GUI graphical user interface. 19

L
LIA lock-in amplifier. 4, 10, 15

P
PSD power spectral density. viii, 3, 5–7, 9–11, 13, 15–18, 21–23

Q
QFT quantum Fourier transform. ix

R
RMS root mean square. 4, 6, 16–18

T
TIA transimpedance amplifier. 13
TLF two-level fluctuator. 10

Figure source files and parameters

Declaration of Authorship

I, Tobias Hangleiter, declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.

I do solemnly swear that:

1. This work was done wholly or mainly while in candidature for the doctoral degree at this faculty and university;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this university or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others or myself, this is always clearly attributed;
4. Where I have quoted from the work of others or myself, the source is always given. This thesis is entirely my own work, with the exception of such quotations;
5. I have acknowledged all major sources of assistance;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published before as:

[1] Pascal Cerfontaine, Tobias Hangleiter, and Hendrik Bluhm. "Filter Functions for Quantum Processes under Correlated Noise." In: *Physical Review Letters* 127.17 (Oct. 18, 2021), p. 170403. DOI: 10.1103/PhysRevLett.127.170403.

[2] Thomas Descamps, Feng Liu, Tobias Hangleiter, Sebastian Kindel, Beata E. Kardynał, and Hendrik Bluhm. "Millikelvin Confocal Microscope with Free-Space Access and High-Frequency Electrical Control." In: *Review of Scientific Instruments* 95.8 (Aug. 9, 2024), p. 083706. DOI: 10.1063/5.0200889.

[3] Tobias Hangleiter, Pascal Cerfontaine, and Hendrik Bluhm. "Filter-Function Formalism and Software Package to Compute Quantum Processes of Gate Sequences for Classical Non-Markovian Noise." In: *Physical Review Research* 3.4 (Oct. 18, 2021), p. 043047. DOI: 10.1103/PhysRevResearch.3.043047.

Aachen, September 7, 2025.