Mesures temporelles large bande résolues en phase du bruit de grenaille photoexcité et statistique de photons d'un amplificateur paramétrique Josephson

par

Jean Olivier Simoneau

Thèse présentée au département de physique

en vue de l'obtention du grade de docteur ès science (Ph.D.)

FACULTÉ des SCIENCES

UNIVERSITÉ de SHERBROOKE

Sherbrooke, Québec, Canada, 30 janvier 2021

Le 30 janvier 2021

le jury a accepté la thèse de M. Jean Olivier Simoneau dans sa version finale.

Membres du jury

Professeur Bertrand Reulet
Directeur de recherche
Département de physique

Professeur Claude Bourbonnais
Membre interne
Département de physique

Julien Gabelli, PhD, Chercheur CRNS
Membre externe
Université Paris–Saclay

Professeur Michel Pioro-Ladrière
Président rapporteur
Département de physique

À Audrey-Ann et Wilo

Sommaire

On présente dans cette thèse deux applications du traitement des signaux quantiques à l'étude des fluctuations qui émanent de composantes mésoscopiques. En premier lieu, on s'intéresse à l'étude des fluctuations électroniques en sortie d'un amplificateur paramétrique Josephson du point de vue de la statistique de photons. En second lieu, on présente une exploration des fluctuations émises en bande large par une jonction tunnel dans le régime photoexcité à travers des mesures dans le domaine temporel et un traitement numérique des données. On porte une attention particulière à la calibration et au traitement numérique des données, qui sont fondamentaux à la probité des résultats.

Les résultats obtenus quant à la statistique de photons de l'amplificateur paramétrique Josephson permettent de vérifier quelles prédictions théoriques sont représentatives de la situation expérimentale typique en plus de mettre en valeur l'effet de la procédure de détection sur les résultats obtenus.

Du côté de l'étude dans le domaine temporel de la jonction tunnel photoexcitée, on met à profit l'équipement micro-onde à la fine pointe de la technologie et la grande puissance de calcul disponible pour développer des méthodes de traitement du signal polyvalentes. En particulier, celles-ci nous permettent d'introduire et de mesurer expérimentalement la densité spectrale de puissance résolue en fréquence et en phase dans différentes conditions expérimentales. Cette quantité, d'une richesse surprenante, nous donne à son tour accès à une pléthore de corrélations courant–courant. Elle nous permet aussi de définir la réponse harmonique des fluctuations à l'excitation, qui met en lumière les effets de retard et de temps caractéristique au sein de la jonction tunnel.

Remerciements

Je tiens d'abord à remercier Bertrand Reulet pour toute l'aide, le soutien, l'encadrement et la motivation qu'il m'a apportés tout au long de mon parcours académique. Bertrand est toujours disponible pour ses étudiants, que ce soit pour discuter des résultats, prévoir des manips, explorer des données fraîches interactivement, ou simplement pour déguster un bon café en discutant de tout et de rien. L'ambiance amicale et décontractée qu'il a su instaurer dans son groupe de recherche permet d'amplifier les bons moments et d'atténuer les passages plus difficiles qui sont inévitables en recherche. Je le remercie en particulier pour sa confiance et son temps pendant toutes ces années passées sous sa direction.

Je remercie de plus les membres de mon jury de thèse — Claude Bourbonnais, Julien Gabelli, Michel Pioro-Ladrière et Bertrand Reulet — d'avoir eu la patience de lire les *quelques pages* formant cette thèse et d'avoir participé au processus d'évaluation et de soutenance de thèse en pleine pandémie de COVID–19, alors qu'ils jonglent tous avec de multiples responsabilités.

Je suis aussi énormément reconnaissant envers Christian Lupien pour toutes les connaissances qu'il m'a transmises. De l'électronique à l'informatique, au fonctionnement des cryostats à dilution, en passant par le contrôle et fonctionnement des appareils de laboratoire ; Christian aura toujours su trouver les bonnes questions pour répondre aux miennes. La variété et la profondeur de ses compétences techniques et scientifiques font de lui un collaborateur sans pareil et une source de savoir intarissable.

Un grand merci à Stéphane Virally avec qui j'ai poursuivi la collaboration entamée lors de mes travaux de maîtrise, collaboration qui a abouti aux résultats de la première partie de la présente thèse et font l'objet d'un article soumis pour publication. J'en profite pour remercier Samuel Boutin pour les discussions à ce sujet et ses commentaires sur le manuscrit de l'article.

Je veux remercier Eva Dupont-Ferrier, Max Hofheinz et Clément Godfrin de m'avoir prêter la source micro-onde utilisée dans le cadre de la deuxième partie de ma thèse. Sans

ce prêt de *quelques semaines* qui s'est tranquillement transformé en prêt de *plusieurs mois*, je n'aurais jamais pu effectuer les mesures résolues en phases qui forment une grande partie des résultats originaux de mes travaux.

Je veux aussi remercier tous les membres du département de Physique — autant le corps professoral que les étudiants gradués, le personnel technique et le personnel de soutient — qui y font un travail exceptionnel et créent un endroit où il fait bon étudier. Un merci particulier à Guy Bernier, Jeffrey Quilliam, Yves Bérubé-Lauzière, Édouard Pinsolle et Michel Pioro-Ladrière avec qui j'ai eu la chance de travailler au niveau de l'enseignement.

Merci également aux nombreux membres du groupe Reulet que j'ai côtoyés au cours des années : Édouard, Karl, Sam Houle, Pierre, Vishnu, Mathieu, Simon, Laurine, Charles, Louis, Alexandre, Emmanuel, Dominique, Marc, Fatou, Jean-Charles, Sébastien, Sam Bateau, Maxime, Jonathan, Gasse, Keyan, Anthoni, Kevin, ainsi que les stagiaires qui se sont joints à nous au cours des années. Si l'ambiance est si bonne dans le groupe, c'est grâce à vous ! Je veux aussi souligner mes partenaires de squash Steve, Mathieu, Édouard et Sébastien, avec qui j'ai pu me changer les idées le temps de foncer dans les murs et briser des raquettes sur l'heure du midi.

Je tiens aussi à remercier mes amis avec qui le temps passé à jouer à des jeux de société et à organiser de bons soupers qui finissent en soirées arrosées aura permis de mettre le stress des études de côté de temps en temps. Un merci particulier à Fred, Marco et Guill pour les niaiseries, les gun-à-patates, le snow, le skate, la trampoline, et les blessures à travers les années.

Merci aussi à mes parents, Pauline et Sylvain, de m'avoir toujours soutenu dans mes études et encouragé à être curieux et à apprendre. Je veux aussi remercier ma tante Jocelyne Simoneau qui a su éveiller en moi un intérêt pour la lecture et les sciences dès mon enfance.

Enfin, je ne saurais trouver les mots pour adéquatement rendre hommage à celle avec qui je partage ma vie depuis bientôt quinze ans. Faire un doctorat est déjà assez ardu en soi, vivre avec un doctorant occupé, distrait et noctambule est encore plus difficile ; et toi tu fais les deux ! Audrey, je t'ai dédié mon mémoire, mais pour la thèse tu devras te contenter de partager l'honneur avec Wilo.

Je crois qu'au fond c'est encore mieux.

Table des matières

Liste des figures

Chapitre 1

Introduction

1.1 Mesure et traitement de signaux quantiques

Les technologies quantiques s'imposent de plus en plus comme un domaine d'avenir. En effet, les propriétés contre-intuitives [1–3], mais indubitablement réelles [4–6], de la physique quantique permettent de pousser la métrologie et le traitement de l'information aux limites des possibilités des lois de la nature [7–14]. Or, les propriétés quantiques sont subtiles et fragiles ; les observer expérimentalement requiert donc une attention particulière à la fois du côté de l'élaboration des échantillons ou systèmes d'intérêts, mais aussi quant aux méthodes utilisées pour en faire la mesure.

On s'intéresse ici à l'aspect de la mesure, plus spécifiquement à l'extraction de l'information associée aux propriétés quantiques du système d'étude, information qui se retrouve souvent noyée parmi les fluctuations classiques. On se concentre en particulier sur les signaux quantiques micro-ondes, en insistant sur l'information contenue au sein des fluctuations [15]. Dans ce domaine, l'amplification nécessaire à l'observation des signaux quantiques faibles vient en effet s'ajouter aux contributions d'intérêts ; il faut alors utiliser des techniques de calibration et de traitement des signaux pour isoler les contributions recherchées. Un traitement du signal judicieux peut aussi donner accès à des quantités qui nous éluderaient autrement.

Deux sujets de ce domaine sont abordés dans cette thèse ; chacun étant introduit plus en détail à sa partie respective.

Le premier sujet est l'étude de l'amplificateur paramétrique Josephson du point de vue de la statistique de photons. Pour cette partie, on s'intéresse à l'interprétation en termes d'optique quantique du champ électromagnétique en sortie d'un amplificateur paramétrique Josephson, ainsi qu'à l'importance de la procédure de détection sur les résultats expérimentaux obtenus.

Le second sujet adopte une perspective négligée du domaine, soit les mesures dans le domaine temporel en bande large. En effet, pour des raisons de praticités, la majorité des expériences traitant des signaux quantiques dans les micro-ondes sont faites dans le domaine fréquentiel. Ce choix est naturel lorsque les systèmes étudiés ont une échelle de fréquence définie par un résonateur ou bien la transition entre des niveaux d'énergie discrets, par exemple. Toutefois, certains systèmes qui ne comportent pas de référence fréquentielle intrinsèque sont aussi typiquement étudiés du point de vue des fréquences. On propose donc ici d'étudier un tel échantillon — une jonction tunnel sans échelle de fréquence naturelle — à l'aide de mesures temporelles et d'une bonne dose de traitement numérique du signal.

1.2 Structure de la thèse

Cette thèse est séparée en deux parties principales indépendantes, respectivement associées aux deux projets qu'elle couvre. Chaque section comporte sa propre mise en contexte et sa conclusion spécifique. La structure de la thèse est hybride ; la première partie étant construite autour du manuscrit d'un article scientifique alors que la seconde partie est traitée de manière traditionnelle.

La première partie, qui comprend les chapitres 2 à 5, traite des mesures de la statistique de photons de l'amplificateur paramétrique Josephson. La deuxième partie, couverte par les chapitres 6 à 13, présente quant à elle l'étude des fluctuations émises en large bande par une jonction tunnel photoexcitée à

l'aide de mesures effectuées dans le domaine temporel. Le chapitre 14 présente finalement une conclusion générale.

Première partie

Statistique de photons : JPA

Chapitre 2

Amplificateur paramétrique Josephson, compression d'état et statistique de photons

Une des composantes les plus utiles de l'informatique quantique expérimentale est la jonction Josephson [16–18]. Cette composante relativement simple — essentiellement deux supraconducteurs en interraction indirecte à travers un isolant ou un métal normal — est étonnament polyvalente. Avec les bons paramètres et le bon environement, elle permet de réaliser des détecteurs de rayonnement électromagnétique [19–21], des sondes de champs magnétique extrêment sensibles [10, 22, 23], des circulateurs cryogéniques [24, 25], des amplificateurs opérant à la limite quantique [11, 26–32] et tout un assortiment de qubits supraconducteurs [8, 12–14] ; entre autres. C'est en quelque sorte l'archétype de la composante électronique purement quantique.

Plus spécifiquement, on s'intéresse ici à l'une des applications de la jonction Josephson des plus utiles en traitement des signaux quantiques, soit l'amplificateur paramétrique Josephson — ci-après paramp. En opérant, en théorie, à la limite quantique, le paramp permet d'amplifier les faibles signaux émanant des composantes en régime quantique en leur ajoutant aussi peu de bruit que la nature le permet, augmentant à la fois la fiabilité et la rapidité des mesures [7 ; 33, §3].

Le paramp est souvent utilisé comme premier élément d'une chaîne d'amplification de manière assez pragmatique, idéalisée ; sans trop porter attention à son fonctionnement interne [34]. Cette approche est tout à fait adéquate lorsque seul un gain raisonnable et un minimum de bruit ajouté son requis. Cependant, si le paramp est ajusté avec un gain trop optimiste, des effets d'ordres supérieurs peuvent devenir appréciables et modifier le comportement du paramp de manière *a priori* inatendue, ce qui peut affecter les mesures [35–38]. Il est donc essentiel de bien comprendre le comportement des paramps réels utilisés expérimentalement pour bien saisir leurs limitations actuelles et en aiguiller le développement et la recherche.

Le paramp pouvant être considéré comme un *compresseur d'état* [1], il est naturellement associé à la génération de paires de photons corrélés, voire intriqués [39 ; 40, §3.7 ; 41, §9.4 ; 42, §10.6.3]. Ainsi, si son entrée est le signal le plus simple imaginable — le vide — la statistique de photons attendue à sa sortie est celle du vide comprimé, c'est-à-dire [2] $\langle \delta n^2 \rangle = 2\langle n \rangle \left(\langle n \rangle + 1\right)$. Or, une publication théorique récente [43] prédit un résultat différent, soit $\langle \delta n^2 \rangle = 2\langle n \rangle \left(8\langle n^2 \rangle + 5\langle n \rangle + 1\right)$. Il est donc pertinent de vérifier expérimentalement laquelle de ces prévisions est observée dans une mesure typique en laboratoire et de bien cerner les causes de la disparité de ces prédictions.

Il se trouve que le paramp est au coeur de la chaîne d'amplification de mes travaux de maîtrise sur le lien entre les fluctuations électromagnétiques et la statistique de photons d'une jonction tunnel générant des paires de photons corrélés [44–46]. Il est donc relativement simple d'adapter le montage expérimental de ces études, non pas pour utiliser le paramp comme un simple amplificateur, mais bien pour l'étudier comme échantillon d'intérêt.

Pour mesurer la statistique de photons d'un paramp, il suffit donc conceptuellement de retirer la jonction tunnel du montage de [44, §3.1 ; 46]. Bien sûr, la situation n'est pas si simple en pratique et le retrait de la jonction tunnel présente des défis quant à la calibration de la mesure.

Les travaux effectués sur ce sujet font l'objet d'un article en collaboration

1. Traduction libre de l'anglais *squeezer*.
2. Notons que $\langle \delta n^2 \rangle = \langle n^2 \rangle - \langle n \rangle^2$ est la variance du nombre de photons.

avec Stéphane Virally. Le manuscrit de l'article est présenté au chapitre 3 ; il est aussi disponible sous forme de prépublication arXiv [47]. Le chapitre 4 présente ensuite des détails supplémentaires quant à la calibration des mesures ainsi que l'ensemble des résultats obtenus pour toutes les conditions expérimentales étudiées. Finalement, le chapitre 5 fait la synthèse du projet et de ses conclusions.

Chapitre 3

Statistique de photons du paramp et importance de la procédure de détection

Ce chapitre présente le manuscrit de l'article « Photocount statistics of the Josephson parametric amplifier : a question of detection » [47]. Pour ce qui est de la contribution principale de chacun des auteurs, en ordre d'attribution : j'ai conçu et implémenté le montage expérimental, les méthodes de calibration et les scripts d'acquisition de données, en plus d'effectuer le traitement et l'analyse de ces dernières ; Stéphane Virally a développé la modélisation théorique du paramp et a participé à l'élaboration de la méthode expérimentale lors de travaux antérieurs [44–46] ; Christian Lupien a développé le système d'acquisition des données et est responsable des systèmes cryogéniques ; et Bertrand Reulet a encadré le projet en tant que directeur de recherche. Tous les auteurs ont participé à la planification du projet, à la discussion et à l'interprétation des résultats. Le manuscrit a été rédigé par Stéphane Viraly, Bertrand Reulet et moi-même.

Suit le manuscrit de l'article.

Photocount statistics of the Josephson parametric amplifier: a question of detection

Jean Olivier Simoneau,[1,*] Stéphane Virally,[1,2] Christian Lupien,[1] and Bertrand Reulet[1]

[1] *Institut Quantique, Département de Physique, Université de Sherbrooke, Sherbrooke, Québec J1K 2R1, Canada*
[2] *Département de Génie Physique, Polytechnique Montréal, Montréal, Québec H3T 1J4, Canada*

(Dated: January 19, 2021)

Parametric amplifiers are known to squeeze the vacuum state of the electromagnetic field, which results in predictable statistics of the photocounts at their output. However, recent theoretical work [1] predicts a very different statistical distribution for an amplifier based on a Josephson junction. We test the hypothesis experimentally and recover the expected squeezed vacuum statistics. We explain this discrepancy by showing theoretically how the photocount statistics is dictated by the detection process, from single mode (our experiment) to multimode, fully resolved in frequency (as in [1]).

Introduction. Photons truly reveal themselves as particles only when interacting with the matter field. From the experimental perspective, a photon is best described as two causally linked events, a creation and an annihilation. The statistics of photocounts must then depend both on the emission and the detection modes, and predictions about statistics of photons emitted by any system should always specify the detection setup.

This fact becomes an important factor in some experiments. Consider, for instance, the output of a Josephson parametric amplifier (JPA) [2–6]. This type of device is very much at the forefront of quantum optics in microwaves, as it constitutes a quantum-limited amplifier in this band and as such is likely to be used in all quantum computing and measurement schemes. Understanding the noise characteristics of those devices is critical for these typically small signal applications. In the most fundamental case, the input of a JPA is simply the electromagnetic vacuum. A theoretical paper [1] predicts the full counting statistics of photocounts emitted by such a system. But the results appear to contradict the model of parametric amplifiers as "vacuum squeezers", a well-studied quantum optics fact. The squeezing operator generates pairs of photons, and this is reflected in the photocount variance, which reads $\langle \delta n^2 \rangle = 2\langle n \rangle (\langle n \rangle + 1)$. In contrast, the theoretical predictions of Ref. [1] is $\langle \delta n^2 \rangle = 2\langle n \rangle \left(8\langle n \rangle^2 + 5\langle n \rangle + 1\right)$[7][8, 9]. In both cases, at very small signal, $\langle \delta n^2 \rangle \simeq 2\langle n \rangle$, twice the classical value. This reflects the emission of pairs of photons in the squeezing process. However, the variance predicted in Ref. [1] dramatically increases as $\langle n \rangle^3$ for higher signals. This is a strong departure form the expected squeezed vacuum behavior.

In this paper, we show that the apparent discrepancy is due to the choice of detection scheme in the theoretical reference. Indeed, the detector is assumed to have infinite frequency resolution. Of course, real measurements are limited both in time and bandwidth. They inherently possess finite frequency resolution. When the detection bandwidth closely resembles the natural mode of the amplifier's cavity, a single quantum mode is observed and we show both theoretically and experimentally that the 'correct' squeezed vacuum statistics is recovered.

The paper is organized as follows. We present an experiment with limited frequency resolution at the output of a JPA with vacuum input. The discrete photocount statistics is recovered from continuous voltage measurements [10]. We show that after careful calibration, we recover a variance and third-order photocount moment equal to those predicted for a squeezed vacuum, and not those predicted by Ref. [1]. These measurements are well captured by a simple input-output [11–13] model of the JPA, followed by a single-mode (non frequency-resolved) detector. In contrast, a variant of the model, using the same input-output relations for the JPA but a multimode (frequency-resolved) detector, leads to the statistics predicted by Ref. [1]. The distinction is lost in narrow-band experiments, but we anticipate that it will play a crucial role in the results of future experiments using the new generation of wide bandwidth JPAs [14, 15].

Experimental setup. The experimental setup is presented in Fig. 1. We study the signal emitted by a commercial Josephson parametric amplifier (paramp), similar to that of Ref. [16], placed in a dilution refrigerator at $\sim 7\,\mathrm{mK}$ and driven by two (phase-locked) sinusoidal pumps of frequencies $f_1 = 4.5\,\mathrm{GHz}$ and $f_2 = 7.5\,\mathrm{GHz}$. The output signal is measured in a small frequency band centered around $(f_1+f_2)/2 = 6.0\,\mathrm{GHz}$. The dual-pump operation mode [17] is selected to avoid residual pump signal in the measurement band, so that the input of the paramp in the measured bandwidth can be considered as the vacuum.

The paramp resonance frequency can be tuned by a current bias through a superconducting flux coil in the vicinity of the paramp SQUID loop (omitted on schematic for clarity). A 4–8 GHz band pass filter protects the paramp from radiation outside of its operation range. Circulators are used to separate the input and output fields of the paramp and to isolate it from the noise of the 3 K and 300 K stages. A microwave switch is used to swap a $50\,\Omega$ resistor in place of the paramp for calibration purposes.

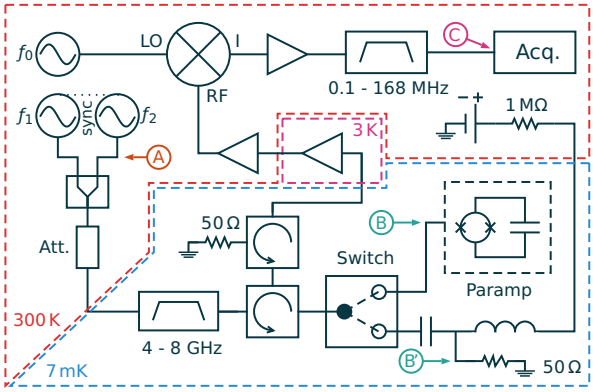


FIG. 1. Experimental setup used for detection. The flux bias coil of the paramp is omitted for clarity. Circled letters are calibration reference points. See text for details.

The paramp output signal is amplified and conveyed to 300 K, where it is downconverted by an IQ mixer with a local oscillator (LO) at frequency $f_0 \approx 6.0\,\mathrm{GHz}$. The LO is *not* phase-locked with the pumps. The downconverted signal is then filtered by a $0.1 - 168\,\mathrm{MHz}$ bandpass filter and sampled by a fast acquisition card with 14-bit resolution and 400 MSa/s rate. The effective photocount integration time is the inverse of the full detected bandwidth ($2 \times 168\,\mathrm{MHz}$). Histograms of the measured signal and their six first cumulants are computed on the fly during the data acquisition.

Calibration. Proper calibration is essential to compare experimental results with theory. Three calibrations are required: that of the ac power at the sample level, that of the absolute photon numbers that are detected, calculated for the measured voltage cumulants C_j, and that of the paramp resonance frequency vs. current in the flux bias coil.

To calibrate the attenuation between the excitation at room temperature (circle reference A in Fig. 1) and the input port of the paramp (B ≈ B'), we use a macroscopic $R = 50\,\Omega$ resistor in place of the paramp (using the cryogenic switch). We can heat that resistor using either a known dc current or an ac bias and observe the temperature increase by the increased noise it emits. Thus we can map which dc current is needed to heat the resistor as much as a given ac voltage, as in [18]. The linear relation we observe between them provides us with the A–B attenuation, 24.96 dB.

To calibrate the effective gain between the output of the paramp (B) and the data acquisition (C), we measure the A–C gain by adjusting the paramp DC flux line to put it out of resonance such that it totally reflects an incoming test tone signal of known amplitude, and subtract the previously obtained A–B attenuation. We find the B–C gain to be 87.67 dB.

The paramp resonance frequency, which is controlled by the current applied to the flux bias coil, is calibrated by measuring the reflected phase on the paramp using a vector network analyser in the absence of a pump signal.[16, 19]

Measurements. In order to probe the photon statistics of the paramp for different regimes of operations, we explore its parameter space, flux bias and pump power, for a fixed measurement frequency $f_0 = (f_1 + f_2)/2 = 6.0\,\mathrm{GHz}$. Experimentally, we first select a pump power yielding a maximum gain of approximately 10 dB and adjust the paramp at this operation point. Then, we sweep the flux bias current and the pump power around the initial values while measuring the cumulants of the voltage fluctuations generated by the paramp. From these we compute the moments of the photocount distribution $\langle n\rangle, \langle \delta n^2\rangle, \langle \delta n^3\rangle$, shown in Fig. 2 using the precedure developped in [10, 20]. We show in Fig. 3 the variance and in Fig. 4 the skwness of the photocount distribution as a function of the average photon number. There are many combinations of flux bias and pump power that give the same average photon number $\langle n\rangle$, each providing a different value of $\langle \delta n^2\rangle$ and $\langle \delta n^3\rangle$. As a consequence, Figs. 3 and 4 exhibit clouds of experimental points and not just single curves. A particular subset of points corresponds to the maximum gain of the paramp, i.e. the largest value of $\langle n\rangle$ for each pump power. Those are the best operating points for the paramp used as an amplifier; they are represented as blue solid points in Fig. 2 (a). Reporting these points in Figs. 2 (b) and (c), we observe that they are close the the maximum of the fourth cumulant C_4 and correspond to a vanishing C_6. The same points are highlighted in Figs. 3 and 4 (open circles). We find that these specific points closely follow the expected relations for a squeezed vacuum, represented by dashed lines in Figs. 3 and 4.

Theory. To model the JPA, we apply the input-output formalism [11–13] to a single-ended, frequency-symmetric single-mode cavity. The intra-cavity Hamiltonian is assumed to be the squeezing Hamiltonian that is characteristic of parametric amplifiers [21–24]. The output electromagnetic modes can be written as a function of the free input modes $\boldsymbol{b}_\nu$, in the frame rotating at ν_0 and up to a constant phase, as

$$\boldsymbol{B}_{\mathrm{out}}(\nu) = \cosh[\eta(\nu)]\boldsymbol{b}_\nu + e^{i\phi}\sinh[\eta(\nu)]\boldsymbol{b}^\dagger_{-\nu}, \qquad (1)$$

with ϕ defining a squeezing direction, and

$$\eta(\nu) = \frac{1}{2}\ln\left[\frac{\sqrt{(\Gamma^2+\nu^2+|\xi|^2-\delta^2)^2+4\delta^2\Gamma^2}+2\Gamma|\xi|}{\sqrt{(\Gamma^2+\nu^2+|\xi|^2-\delta^2)^2+4\delta^2\Gamma^2}-2\Gamma|\xi|}\right]. \qquad (2)$$

In this expression, Γ is the cavity coupling parameter, inversely proportional to the decay time of the cavity, and ξ is the nonlinear intra-cavity 2-photon coupling parameter (in the two-pump scheme, $|\xi| \propto \sqrt{P_1 P_2}$, the geometric

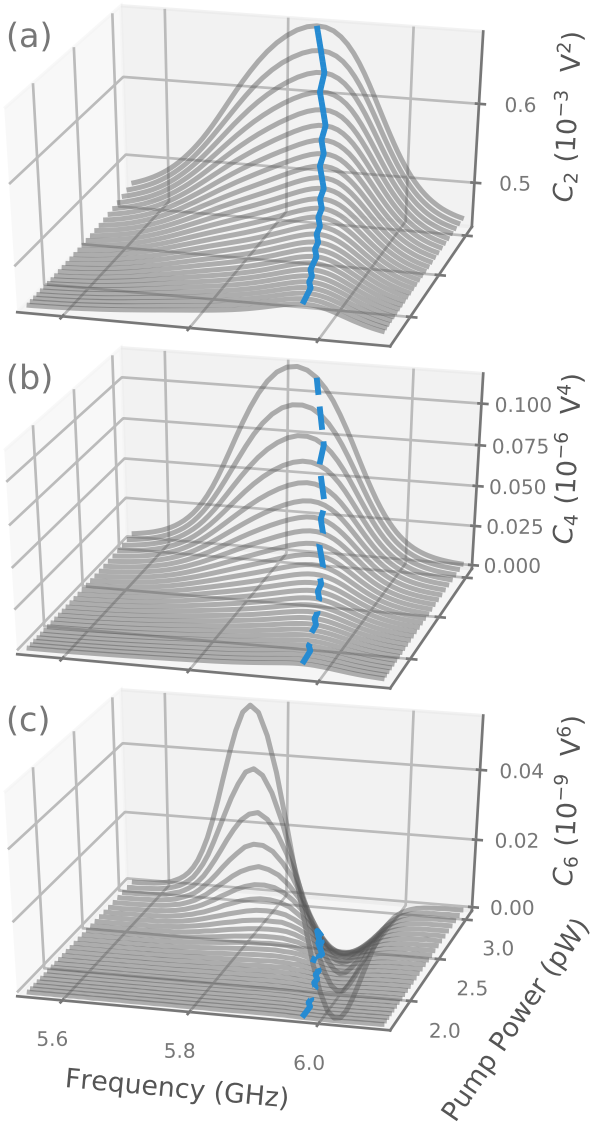


FIG. 2. Measured cumulants of voltage fluctuations generated by the paramp as a function of its resonance frequency and pump power. The *Frequency* axis is controlled by the flux bias. The thick blue solid line in (a) corresponds to the ridge of C_2. The dashed blue line in (b) and (c) corresponds to frequency and pump power that correspond to the blue line of (a).

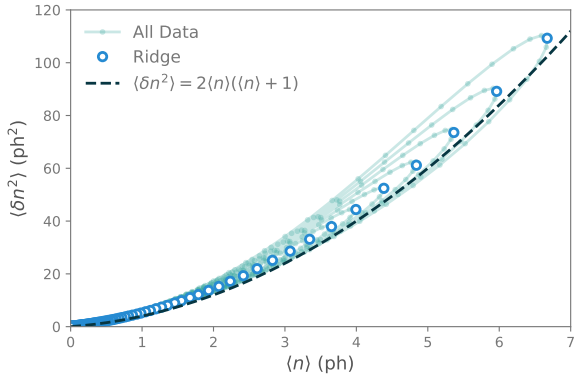


FIG. 3. Variance of the photocounts $\langle\delta n^2\rangle$ as a function of the average photon number $\langle n\rangle$. Dots are experimental data and each line represents a given pump power. Open circles correspond to the maximum of the paramp gain, i.e the blue line in Fig. 2. The dashed line corresponds to the theoretical prediction for squeezed vacuum.

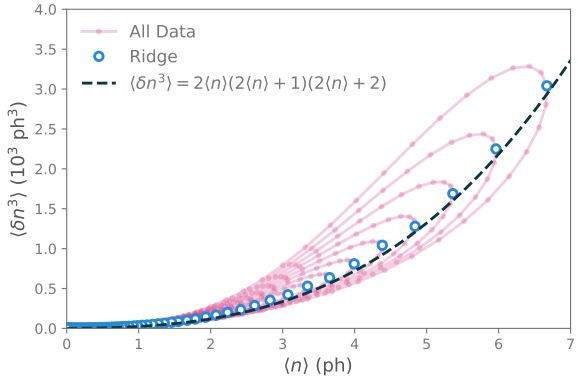


FIG. 4. Skewness of the photocounts $\langle\delta n^3\rangle$ as a function of the average photon number $\langle n\rangle$. Dots are experimental data and each line represents a given pump power. Open circles correspond to the maximum of the paramp gain, i.e the blue line in Fig. 2. The dashed line corresponds to the theoretical prediction for squeezed vacuum.

average of both pump powers). The photon-photon interaction Hamiltonian also shifts the position of the center peak of the cavity mode proportionally to P_1+P_2, as seen in Fig. 2. The peak can be brought back to the center of the measurement window by adjusting the magnetic flux. This is captured by $\delta = \phi + |\xi|(P_1+P_2)/\sqrt{P_1\,P_2}$, where ϕ is the frequency shift induced by the magnetic flux. The 'ridge' (maximum) of C_2 observed in Fig. 2 corresponds to $\delta = 0$.

Using this model, we calculate the moments of the statistics of the photon flux per unit of frequency and time, for a detector with normalized response function $h(\nu)$,

$$\langle n^k\rangle = \int \mathrm{d}\nu_1\cdots\mathrm{d}\nu_{2k}\; h^*(\nu_1)\,h(\nu_2)\cdots h^*(\nu_{2k-1})\,h(\nu_{2k}) \\ \langle\phi_\mathrm{i}|\,\boldsymbol{B}^\dagger_\mathrm{out}(\nu_1)\boldsymbol{B}_\mathrm{out}(\nu_2)\cdots\boldsymbol{B}^\dagger_\mathrm{out}(\nu_{2k-1})\boldsymbol{B}_\mathrm{out}(\nu_{2k})\,|\phi_\mathrm{i}\rangle\,, \tag{3}$$

where $|\phi_\mathrm{i}\rangle$ is the input state of the JPA. In the case of an electromagnetic vacuum input, $|\phi_\mathrm{i}\rangle = |\mathrm{vac}\rangle$, we find the

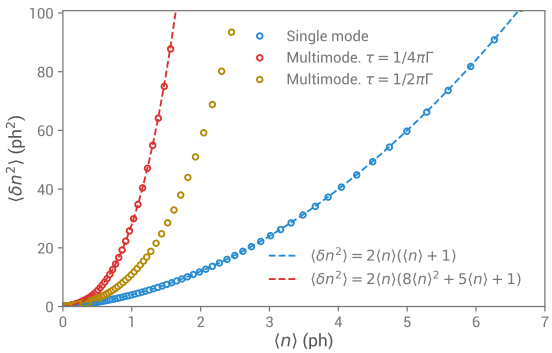


FIG. 5. Comparison between single mode counting, as in our experiment, and multimode counting with unit cavity coupling strength, as in Ref. [1]. The parameter τ is the time of accumulation of photocounts for each data point [25].

expected statistics of a squeezed vacuum on the 'ridge' of C_2. In particular, for η in the unit range or above, [25]

$$\langle n\rangle = \int \mathrm{d}\nu\, |h(\nu)|^2\, n(\nu), \tag{4}$$

with $n(\nu) = \sinh^2[\eta(\nu)]$, and

$$\left\langle \delta n^2 \right\rangle = 2\,\langle n\rangle\,(\langle n\rangle + 1). \tag{5}$$

The Fano factor $\mathcal{F} = \left\langle \delta n^2 \right\rangle / \langle n\rangle$ is thus simply

$$\mathcal{F} = 2\,(\langle n\rangle + 1). \tag{6}$$

This result is at odds with the predictions of Ref. [1]. The reason is that the theoretical framework of the reference uses a different detection scheme, where the signal is resolved in frequency. For a frequency resolution Δ, the measured moments per unit time are,

$$\left\langle n^k \right\rangle_\Delta = \int \mathrm{d}\nu_1 \cdots \mathrm{d}\nu_{2k}\; \delta_\Delta(\nu_1-\nu_2)\cdots\delta_\Delta(\nu_{2k-1}-\nu_{2k}) \\ \langle\phi_\mathrm{i}|\, \boldsymbol{b}_\mathrm{o}^\dagger(\nu_1)\boldsymbol{b}_\mathrm{o}(\nu_2)\cdots\boldsymbol{b}_\mathrm{o}^\dagger(\nu_{2k-1})\boldsymbol{b}_\mathrm{o}(\nu_{2k})\,|\phi_\mathrm{i}\rangle\,, \tag{7}$$

where δ_Δ are peaked functions of width Δ and unit integrated value. They tend to the true Dirac delta distribution as $\Delta \to 0$ [25]. In the same limit, the Fano factor $\mathcal{F}_\Delta \equiv \left\langle \delta n^2 \right\rangle_\Delta / \langle n\rangle_\Delta$ behaves as

$$\lim_{\Delta\to 0} \mathcal{F}_\Delta = \frac{\int \mathrm{d}\nu\, 2\, n(\nu)[n(\nu)+1]}{\int \mathrm{d}\nu\, n(\nu)}. \tag{8}$$

This result is very different from that of Eq. (5). It corresponds to summing many independent modes, resolved in frequency. Each mode is a squeezed vacuum with a Fano factor of the form of Eq. (6). But the behavior of the overall Fano factor is dependent on the detector bandwidth (the inverse of the time τ spent accumulating photocounts for each data point [25]). For a detector with bandwidth $4\pi\Gamma$ (the coupling strength between the inside and outside of the cavity), the Fano factor corresponds to that found in Ref. [1]. However, if another bandwidth is chosen, the relation between $\left\langle \delta n^2 \right\rangle$ and $\langle n\rangle$ is different, as shown in Fig. 5.

Discussion. Our theoretical analysis clearly explains the importance of the detection scheme and thus why the predictions of [1] differ from that of the expected squeezed vacuum statistics. It also predicts that with our present setup we should observe the photon statistics of squeezed vacuum. We indeed observe the 'right' statistics on the 'ridge' of C_2, that is on the optimal functioning points of the paramp.

However, the theory fails to explain why we observe clouds of points for $\langle\delta n^2\rangle$ and $\langle\delta n^3\rangle$ vs. $\langle n\rangle$ in Figs. 3 and 4. It does predict that we can observe a photocount variance lower than that of the squeezed vacuum, but never higher [25]. One can easily understand that if the measurement bandwidth is finite and not centered on the resonance (i.e., off the ridge) there might be photon pairs which are detected as single photons (the other photon of the pair being outside the detection bandwidth). This leads to a mixture between squeezed vacuum and thermal state and thus leads to a decrease of $\langle\delta n^2\rangle$. In contrast, we observe that experimental points off the ridge lie both below and above the variance of squeezed vacuum.

In an attempt to correct the theory, we considered the effect of wideband detection [25, 26]. We do find a small cloud of points, but they all lie beneath the theoretical maximum variance. In addition, we find a non-zero sixth-cumulant C_6 outside of the ridge, as featured in Fig. 2. This is an interesting feature, as the narrow-band theory predicts $C_6 = 0$ everywhere, just as it is zero on the 'ridge' of our experiments. However, the amplitude of the corrected C_6 is too small by one order of magnitude compared to the experiments. Hence, the wideband correction is insufficient to explain experimental data.

Another potential shortcoming of the theoretical model is the fact that the nonlinear coupling of the Josephson junction is cut at the second order in our Hamiltonian. This is also the case for the reference motivating this text [1], so we did not attempt to expand the Hamiltonian to higher orders. However, the Josephson parametric amplifier can be highly nonlinear, and this simplification is likely to fail at higher powers, starting at the single digit photon number. This has been studied in details in [24] for the case $\phi = 0$. In Figs. 3 and 4, we linked all the points corresponding to the same pump power by a single line. We clearly see that excursions away from the theoretical values increase dramatically with pump power. We also see that the ridge is defined as the maximum of $\langle n\rangle$ vs. flux bias for any given pump power, i.e. $\left(\frac{\partial\langle n\rangle}{\partial\Phi}\right)_P = 0$. It is straightforward to show

that it also corresponds to the minimum pump power for a given $\langle n \rangle$, i.e. $\left(\frac{\partial P}{\partial \Phi}\right)_{\langle n \rangle} = 0$. As a consequence, we show that we recover the squeezed vacuum photocount distribution only for a pump power close to the optimum gain (even though our bichromatic pumping scheme is the one that leads to the least nonlinearities [24]). In addition, half the points lie above, and half the points lie below the theoretical curve. Thus we expect that a successful theory would take into account the sign of the flux bias (i.e. it should feature odd terms in the flux bias, which our theory fails to do).

Conclusion. We have performed an experimental and theoretical investigation of the photon statistics of the microwave radiation generated by a Josephson parametric amplifier. We have observed that with a wideband, single-mode detection scheme, the statistics is that of squeezed vacuum when the pump of the paramp is kept to its lowest value for a given average photon number and strongly departs from it at higher power. Our theoretical analysis shows how the photocount statistics crucially depends on the detection bandwidth, from a time-resolved, wideband amplifier (our setup) to that of frequency-resolved photodetection[1]. Our results, which are valid for any kind of paramp, are of great interest both to the development of quantum limited amplifiers with optimal photon statistics as well as for the development of sources of radiation with non-classical statistics. As a matter of fact, instead of playing with the source, we show that one can play with the detector, in a similar way as quantum computation may require non-Gaussian states of light if measurements are performed with linear detectors whereas Gaussian states are enough if one uses single photon detectors[27]. More theoretical and experimental works are needed to explore the path we have paved, in particular to understand how high order terms in the Hamiltonian affect the photcount distribution for an arbitrary detection bandwidth.

We thank G. Laliberté for technical help and S. Boutin for fruitful discussions. This work was supported by the Canada Excellence Research Chair program, the NSERC, the Canada First Research Excellence Fund, the MDEIE, the FRQNT, the INTRIQ, the Université de Sherbrooke, and the Canada Foundation for Innovation.

* Jean.Olivier.Simoneau@USherbrooke.ca

[1] C. Padurariu, F. Hassler, and Y. V. Nazarov, Physical Review B **86**, 054514 (2012).
[2] M. Castellanos-Beltran, K. Irwin, L. Vale, G. Hilton, and K. Lehnert, IEEE Transactions on Applied Superconductivity **19**, 944 (2009).
[3] N. Bergeal, F. Schackert, M. Metcalfe, R. Vijay, V. E. Manucharyan, L. Frunzio, D. E. Prober, R. J. Schoelkopf, S. M. Girvin, and M. H. Devoret, Nature **465**, 64 (2010).
[4] X. Zhou, V. Schmitt, P. Bertet, D. Vion, W. Wustmann, V. Shumeiko, and D. Esteve, Physical Review B **89**, 214517 (2014).
[5] C. Eichler, *Experimental Characterization of Quantum Microwave Radiation and its Entanglement with a Superconducting Qubit*, Ph.D. thesis, Eidgenössische Technische Hochschule Zürich (2013).
[6] S. Jebari, F. Blanchet, A. Grimm, D. Hazra, R. Albert, P. Joyez, D. Vion, D. Estève, F. Portier, and M. Hofheinz, Nature Electronics **1**, 223 (2018).
[7] For a specific value of the photocount integration time, namely the inverse of the JPA's cavity bandwidth.
[8] R. Vyas and S. Singh, Physical Review A **40**, 5147 (1989).
[9] A. L. De Brito and A. A. D. Gomes, Physica A **232**, 563 (1996).
[10] S. Virally, J. O. Simoneau, C. Lupien, and B. Reulet, Physical Review A **93**, 043813 (2016).
[11] C. W. Gardiner and M. J. Collett, Physical Review A **31**, 3761 (1985).
[12] M. Collett, R. Loudon, and C. Gardiner, Journal of Modern Optics **34**, 881 (1987).
[13] C. W. Gardiner and P. Zoller, *Quantum Noise*, 3rd ed. (Springer Berlin / Heidelberg, 2004).
[14] C. Macklin, K. O'Brien, D. Hover, M. E. Schwartz, V. Bolkhovsky, X. Zhang, W. D. Oliver, and I. Siddiqi, Science **350**, 307 (2015).
[15] U. C. Mendes, S. Jezouin, P. Joyez, B. Reulet, A. Blais, F. Portier, C. Mora, and C. Altimiras, Physical Review Applied **11**, 034035 (2019).
[16] M. Hatridge, R. Vijay, D. H. Slichter, J. Clarke, and I. Siddiqi, Physical Review B **83**, 134501 (2011).
[17] A. Kamal, A. Marblestone, and M. Devoret, Phys. Rev. B **79**, 184301 (2009).
[18] D. F. Santavicca, B. Reulet, B. S. Karasik, S. V. Pereverzev, D. Olaya, M. E. Gershenson, L. Frunzio, and D. E. Prober, Applied Physics Letters **96**, 083505 (2010).
[19] J. Y. Mutus, T. C. White, E. Jeffrey, D. Sank, R. Barends, J. Bochmann, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, J. Kelly, A. Megrant, C. Neill, P. J. J. O'Malley, P. Roushan, A. Vainsencher, J. Wenner, I. Siddiqi, R. Vijay, A. N. Cleland, and J. M. Martinis, Applied Physics Letters **103**, 122602 (2013).
[20] J. O. Simoneau, S. Virally, C. Lupien, and B. Reulet, Physical Review B **95**, 060301(R) (2017).
[21] B. Mollow and R. J. Glauber, Physical Review **160**, 1076 (1967).
[22] B. Mollow and R. J. Glauber, Physical Review **160**, 1097 (1967).
[23] M. J. Collett and R. Loudon, Journal of the Optical Society of America B **4**, 1525 (1987).
[24] S. Boutin, D. M. Toyli, A. V. Venkatramani, A. W. Eddins, I. Siddiqi, and A. Blais, Physical Review Applied **8**, 054030 (2017).
[25] See supplementary material.
[26] S. Virally and B. Reulet, Physical Review A **100**, 023833 (2019).
[27] E. Knill, R. Laflamme, and G. J. Milburn, Nature **409**, 46 (2001).

Chapitre 4

Détails supplémentaires

Ce chapitre présente plus en profondeur certains aspects qui ne sont que survolés, par souci de concision, dans l'article présenté au chapitre 3, en particulier en ce qui a trait à la calibration du montage.

4.1 Calibration

Lors des mesures antérieures de la statistique de photons via les fluctuations électroniques de [44–46], on utilisait un paramp comme simple amplificateur et on utilisait la jonction tunnel elle-même pour calibrer les effets du montage. Ici, la jonction tunnel est retirée du montage et on s'intéresse directement à la statistique de photons du paramp. Bien que cela puisse paraître, de prime abord, plus simple, l'impossibilité d'utiliser le paramp lui-même comme référence de calibration complique la mesure expérimentale. En effet, pour que les résultats représentent bien la statistique de photons du paramp, sans les contributions des amplificateurs et autres sources de bruit au sein du montage, il faut développer une stratégie de calibration nouvelle.

Les points d'intérêts pour la calibration sont identifiés sur le montage de la figure 1 de l'article par les lettres A, B, B' et C. Ils correspondent respectivement à la sortie des sources micro-ondes, le port entrée–sortie du paramp, l'entrée de la résistance de 50 Ω et l'entrée de la carte d'acquisition.

In fine, on cherche à extraire le facteur d'atténuation entre la puissance de sortie des pompes du paramp et la puissance qu'il reçoit, ainsi que le gain entre la sortie du paramp et la carte d'acquisition ; respectivement l'atténuation A–B et le gain B–C. Pour extraire ces quantités, on procède en deux étapes : la détermination de l'atténuation A–B, détaillée à la section 4.1.1, et celle du gain A–C, décrite à la section 4.1.2. Le gain B–C est alors simplement la combinaison de ces deux valeurs.

Les mesures de statistique de photons sont effectuées sur une bande de $\approx 336\,\mathrm{MHz}$ centrée autour de $6\,\mathrm{GHz}$ par conversion vers de bas et détection sur la bande 0.1–$168\,\mathrm{MHz}$. Dans ce qui suit, on s'applique donc à calibrer le montage autour de $6\,\mathrm{GHz}$, cette calibration étant représentative de toute la bande.

4.1.1 Atténuation A–B

La calibration de l'atténuation A–B s'effectue en remplaçant le paramp par une résistance cryogénique de $50\,\Omega$ à l'aide du relais de la figure 1 de l'article. La résistance est placée directement sur un T de polarisation qui est lui-même placé directement à la sortie du relais. Il est donc possible de biaiser la résistance en tension alternative à travers le relais avec les sources de pompe du paramp, ou de la biaiser en tension continue via de T de polarisation. Conceptuellement, la calibration consiste à profiter du fait que l'on connaisse très bien le courant de polarisation en tension continue, et donc la puissance DC que reçoit la résistance, pour calibrer la puissance AC. On considère, pour les besoins de la calibration, que les points B et B' sont équivalents.

Lors de l'acquisition des données, on soumet tour à tour la résistance à une série de biais en puissance continue P_{DC} et alternative P_{AC} ; les P_{DC} étant directement celles subies par la résistance et les P_{AC} étant définies en sortie de la source micro-onde. Pendant ces balayages, on mesure le *bruit* C_2, soit le second cumulant des fluctuations de tension, à la carte d'acquisition. Pour obtenir des données probantes, on moyenne chaque mesure ~ 880 fois en alternant chaque mesure polarisée avec une référence sans biais de manière à éliminer les dérives

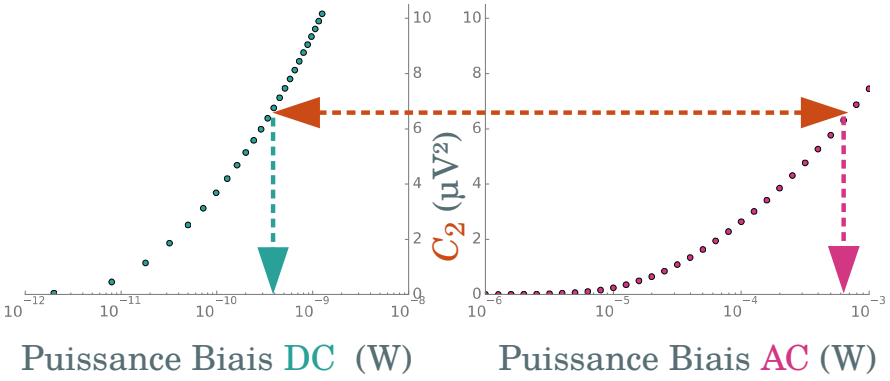


FIGURE 4.1 – Données pour calibration A–B. La puissance DC est celle incidente sur la résistance alors que la puissance AC est celle fournie par la source micro-onde. Les flèches schématisent le principe de la calibration.

lentes lors de l'acquisition. De plus, on prend soin de choisir une fréquence de mesure différente de celle d'excitation — respectivement 4.5 GHz et 6 GHz — en plus de bien filtrer les amplificateurs en dehors du cryostat pour ne pas qu'ils soient affectés par l'excitation. On fixe l'excitation à 6 GHz, puisque c'est la fréquence pour laquelle on veut calibrer l'atténuation. Quant à elle, la valeur exacte de la fréquence de mesure importe peu, ne servant qu'à faire le lien entre P_{AC} et P_{DC}.

La figure 4.1 présente les résultats de cette procédure. De là, on établit la relation $P_{\mathrm{AC}} \leftrightarrow P_{\mathrm{DC}}$ à travers C_2 en considérant qu'un certain niveau de bruit correspond à une puissance de polarisation donnée, indépendamment de sa source — les flèches de la figure 4.1 en schématisent le principe. En pratique, il faut interpoler les données, puisqu'il est fort improbable d'obtenir des valeurs de C_2 identiques, mais les courbes sont assez lisses pour que cela ne cause pas de problèmes.

Une fois la relation entre les puissances AC et DC établie, on peut estimer l'atténuation α en modélisant la situation comme

$$P_{\mathrm{DC}} = \alpha P_{\mathrm{AC}}. \tag{4.1}$$

On peut alors estimer α via le ratio $P_{\mathrm{DC}}/P_{\mathrm{AC}}$, en faisant un lissage linéaire sur

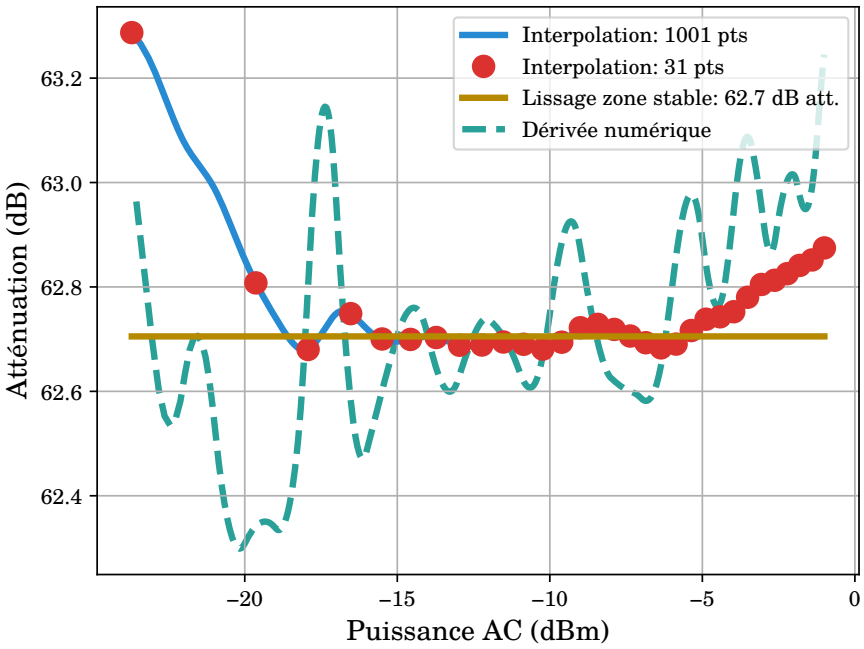


FIGURE 4.2 – Atténuation estimée via la résistance de 50 Ω. Les interpolations correspondent au ratio $P_{\mathrm{DC}}/P_{\mathrm{AC}}$, le traitillé est la dérivée numérique de $P_{\mathrm{DC}}\,(P_{\mathrm{AC}})$ et le trait jaune est un lissage linéaire sur la zone centrale où les ratios $P_{\mathrm{DC}}/P_{\mathrm{AC}}$ sont stables.

(4.1) ou bien en prenant sa dérivée numérique. Ces résultats sont disponibles à la figure 4.2. On y voit qu'à basse puissance, le ratio $P_{\mathrm{DC}}/P_{\mathrm{AC}}$ est instable, probablement dû à la pente quasi nulle des puissances de la figure 4.1 dans ce régime. Aux plus hautes puissances, l'estimation de l'atténuation devient aussi instable, possiblement à cause d'effets de saturation ou de légères non-linéarités des amplificateurs [1].

On se concentre donc sur la zone centrale où $\alpha \sim$ cte. On fait un lissage linéaire sur cette zone, soit environ de $-15\,\mathrm{dBm}$ à $-5\,\mathrm{dBm}$, pour extraire une valeur d'atténuation de $\alpha = -62.7\,\mathrm{dB}$. C'est donc la valeur de l'atténuation A–B', qui est équivalente par hypothèse à l'atténuation A–B recherchée. Notons que la valeur obtenue est raisonnable considérant l'atténuation des câbles, les composantes du montage et les atténuateurs placés entre les étages thermiques

1. Ils peuvent aussi voir une partie de l'excitation AC, via l'isolation imparfaite des circulateurs ou à travers des réflexions dans le montage.

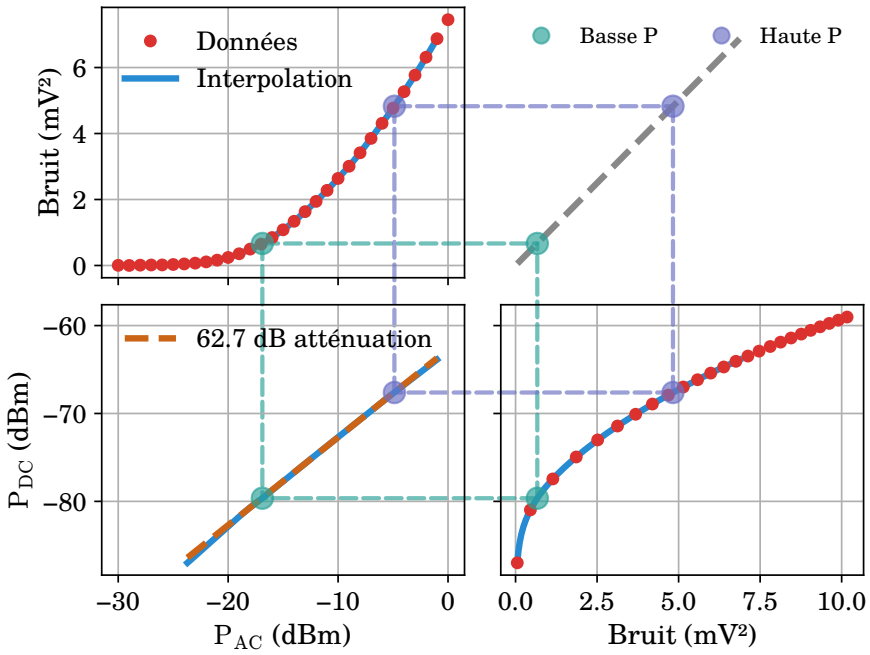


FIGURE 4.3 – Résumé de la calibration de l'atténuation A–B.

du cryostat sur les lignes d'excitation [48, §2.3.2]. La procédure complète de calibration est aussi résumée à la figure 4.3.

4.1.2 Gains A–C et B–C

La calibration du gain A–C est conceptuellement assez simple. D'abord, avec le relais en position *paramp*, on désajuste la fréquence de résonance du paramp de manière à ce qu'il réfléchisse tout signal incident. Ensuite, on génère une tonalité de fréquence $f = 6\,\mathrm{GHz} + \delta f$ et d'amplitude connue en A à l'aide d'une des sources micro-ondes servant normalement de pompe pour le paramp. Ce signal va faire son chemin jusqu'au paramp, y être réfléchi pour ensuite être dirigé en dehors du cryostat par les circulateurs et amplificateurs, être converti vers le bas par le mélangeur et finalement être numérisé par la carte d'acquisition. Il va ainsi subir les effets de tout le montage entre les points A et C, mais — crucialement — sans la contribution du paramp.

À la carte d'acquisition, on mesure la densité spectrale de puissance du

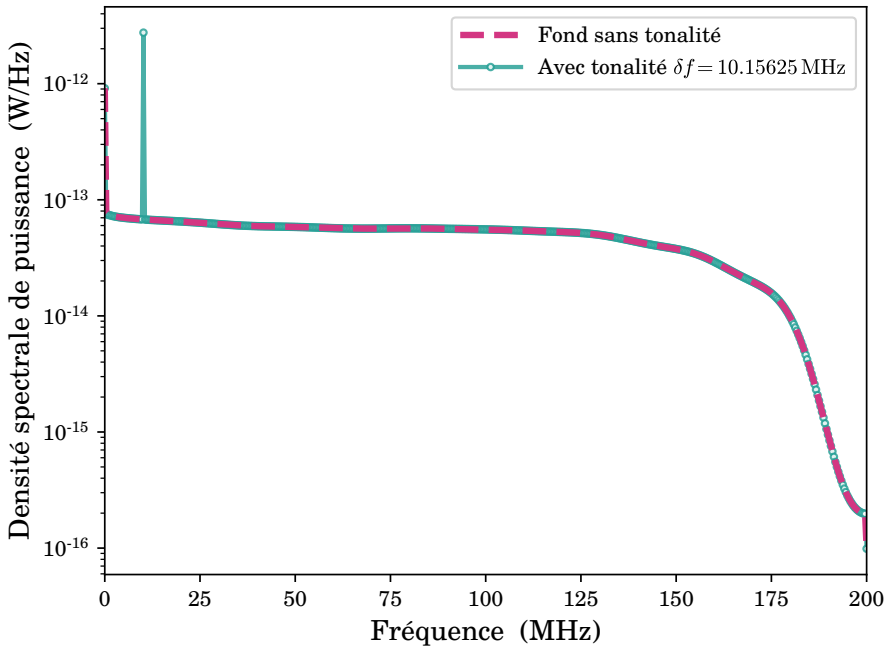


FIGURE 4.4 – Densité spectrale de bruit et tonalité de référence.

signal en présence ou non de la tonalité, comme montré à la figure 4.4. On choisit $\delta f = 10.15625\,\mathrm{MHz}$ pour correspondre exactement à une fréquence résolue par les spectres. La puissance de la tonalité à l'entrée de la carte s'obtient alors en faisant la différence des spectres de la figure 4.4 et en intégrant le résultat sur le seul point à δf.

On répète alors cette procédure en balayant la puissance de la tonalité et en mesurant sa puissance transmise de A à C. La figure 4.5 en présente le résultat. La tendance est linéaire à basse puissance, comme attendu pour un gain A–C indépendant du signal d'entrée, mais ce n'est plus le cas à plus haute puissance. Il semble qu'une tonalité trop forte fasse saturer les amplificateurs. On fait donc un lissage linéaire sur les résultats à basse excitation, pour extraire une valeur de gain A–C de 25.0 dB.

Le gain B–C s'obtient alors simplement en considérant que le signal subit l'atténuation A–B avant d'être réfléchi sur le paramp et d'être soumis au gain B–C ; le gain A–C contient ces deux contributions. Avec une atténuation A–B de $-62.7\,\mathrm{dB}$ et un gain A–C de 25.0 dB, on trouve donc un gain B–C de 87.7 dB

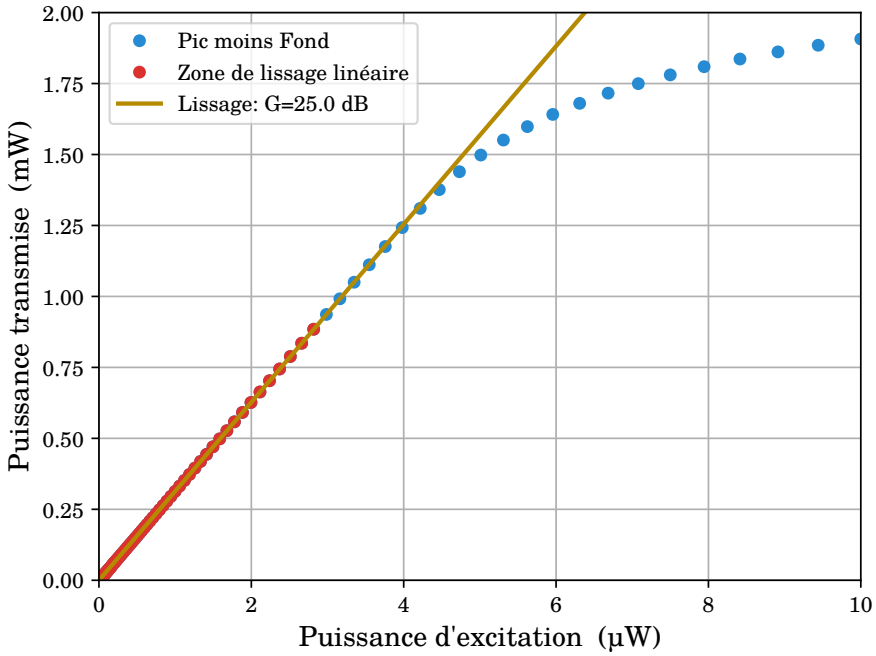


FIGURE 4.5 – Balayage de la puissance de la tonalité. Le lissage est effectué sur la zone linéaire à plus basse puissance d'excitation.

entre la sortie du paramp et la mesure à la carte d'acquisition.

4.1.3 Courant de biais de flux et fréquence de résonance

La figure 2 de l'article présente les différents cumulants mesurés en fonction de la puissance de pompe incidente sur le paramp et de sa fréquence ce résonance. Or, on ne peut pas ajuster la fréquence de résonance directement. On ajuste plutôt le courant dans une boucle de flux qui, à son tour, vient changer la fréquence de résonance du paramp. Pour présenter les données en fonction de la fréquence de résonance, il faut donc relier celle-ci au courant de biais de flux.

À cette fin, on mesure la phase en réflexion du paramp à l'aide d'un analyseur vectoriel en balayant le courant de biais de flux, similairement à [29, 44]. Le résultat de cette mesure est présenté à la figure 4.6, et un grossissement de la région autour de 6 GHz est disponible à la figure 4.7. Ces données ont été obtenues avec le même paramp que celui utilisé ici et sont une gracieuseté de

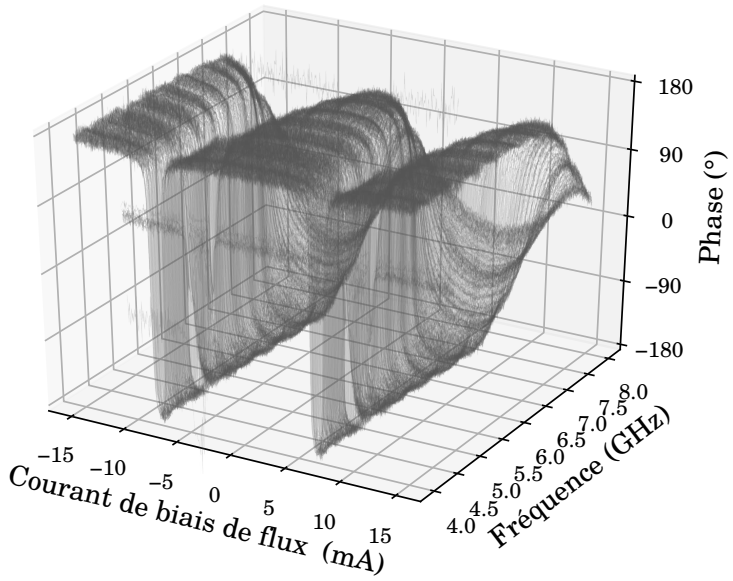


FIGURE 4.6 – Phase en réflexion du paramp à différents biais en flux. Les données sont une gracieuseté de Pierre Février et Charles Marseille.

Pierre Février et Charles Marseille [2].

À fréquence fixe, la résonance du paramp correspond au centre de la région où la phase tourne rapidement de 360°. Pour faciliter l'extraction de ce point, on exprime la phase entre $-180°$ et $180°$ en y ajoutant une constante additive choisie de manière à ce que les phases de part et d'autre de la résonance soient de grandeurs égales et de signes opposés. La résonance correspond alors simplement au point où la phase est nulle, et ce, pour chaque fréquence.

La relation *courant–fréquence* ainsi obtenue est représentée à la figure 4.7 par un traitillé bleu. C'est ce résultat qui est utilisé pour convertir le courant de biais de flux vers la fréquence de résonance du paramp aux axes de la figure 2 de l'article.

2. L'axe de biais de courant de flux, qui est décalé par une constante aléatoire à chaque refroidissement, est recalé sur nos données en utilisant les paramètres optimaux d'ajustement du paramp à 6 GHz pour une puissance de pompe très faible.

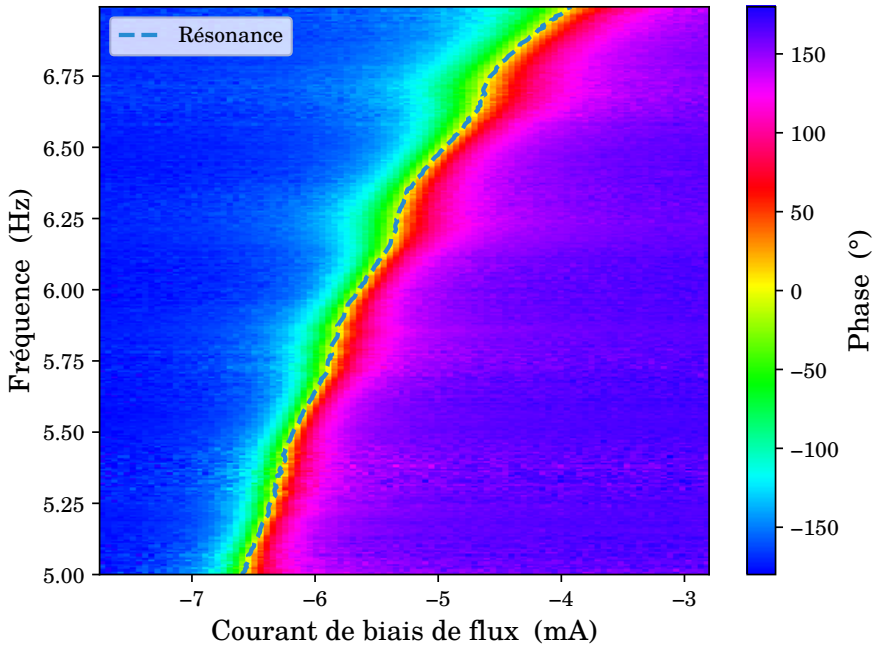


FIGURE 4.7 – Grossissement de la figure 4.6 autour de 6 GHz. La résonance correspond à la phase nulle — le centre jaune de l'*arc-en-ciel* — et est marquée d'un traitillé bleu.

4.1.4 Unités : de volts à nombre de photons

Les résultats de la statistique de photons sont exprimés termes de nombre de photons. Or, les mesures brutes des cumulants ont plutôt des unités en termes de volts, soit V^k pour le k^{e} cumulant. Lors des études précédentes sur la statistique de photons de la jonction tunnel, l'échantillon lui-même permettait de calibrer aisément le montage et d'absorber le facteur de conversion d'unités, ci-après α, dans un gain effectif. Ici, comme on n'utilise plus de jonction tunnel pour la calibration, il faut prendre soin de bien faire la conversion d'unité manuellement.

On modélise les cumulants obtenus sur la bande passante Δf centrée à f avec un gain G_A et un bruit d'amplification S_A comme

$$C_{k,\mathrm{m}} = (\alpha\, G_A)^{\frac{k}{2}} \left(C_k + S_A \delta_{k,2}\right), \tag{4.2}$$

où k est l'ordre du cumulant d'intérêt et α est le facteur de conversion d'unité ;

le cumulant intrinsèque C_k étant exprimé en ph^k. Pour se débarasser de S_A et pour éviter les dérives lentes lors des mesures, on prend une mesure de $C^{(0)}_{k,\mathrm{m}}$, le cumulant brut à polarisation nulle [3] entre chaque mesure de $C_{k,\mathrm{m}}$ à polarisation finie. On s'intéresse alors à leur différence, soit

$$C'_{k,\mathrm{m}} = C_{k,\mathrm{m}} - C^{(0)}_{k,\mathrm{m}} \tag{4.3}$$

$$= (\alpha\, G_A)^{\frac{k}{2}} \left(C_k - C^{(0)}_k \right). \tag{4.4}$$

Ces $C'_{k,\mathrm{m}}$ sont donc exprimés en unités de V^2, et sont intégrés sur une bande passante Δf.

Pour les convertir en unités de photons, qu'on représente par [4] ph, on fait l'hypothèse bande-étroite selon laquelle les densités spectrales à l'ordre k sont constantes sur la bande de largeur Δf. On peut alors utiliser la bande passante de la mesure et l'impédance d'entrée de la carte d'acquisition ($50\,\Omega$) pour convertir le cumulant mesuré en densité de puissance exprimée en $\mathrm{W/Hz} = \mathrm{J}$. Cette densité de puissance est ensuite convertie en termes de ph à l'aide de l'énergie hf que contient un photon de fréquence f.

Dans le cas le plus intuitif, $k = 2$, on a donc

$$C_2 = \overbrace{\underbrace{\frac{C'_{2,\mathrm{m}}}{50\,\Omega}}_{\mathrm{W=J/s}} \cdot \frac{1}{\Delta f} \cdot \underbrace{\frac{1}{hf}}_{\mathrm{ph/J}} \cdot \frac{1}{G_A}}^{\mathrm{J/s/Hz}}_{\mathrm{ph/s/Hz=ph}} + \frac{1}{2}, \tag{4.5}$$

avec le terme $1/2$ correspondant au fait que les fluctuations du vide ont comme seul cumulant non nul $C_2 = 1/2$; autrement dit, $C^{(0)}_k = \frac{1}{2}\delta_{k,2}$.

En se référant à l'équation (4.4) pour trouver α et en généralisant à toutes

3. Sans polarisation continue ni photoexcitation.

4. Les cumulants exprimés en termes de nombre de photons sont techniquement sans unité, le symbole ph est utilisé par souci de clarté.

valeurs de k, on trouve donc

$$C_k = \frac{C'_{k,\mathrm{m}}}{(\alpha\, G_A)^{\frac{k}{2}}} + \frac{1}{2}\delta_{k,2}, \tag{4.6}$$

avec le facteur de conversion d'unité

$$\alpha = 50\,\Omega \cdot \Delta f \cdot h f. \tag{4.7}$$

Ce sont ces deux dernières équations qui permettent de convertir les mesures brutes en unités de voltage vers les unités de photons nécessaire pour l'obtention de la statistique de photons.

4.2 Résultats à plus haute puissance

Les résultats présentés dans l'article ne représentent qu'une petite partie de ceux obtenus. En effet, l'exploration de la puissance de pompe du paramp a été effectuée jusqu'à de très hautes puissances. Alors que les résultats présentés ci-haut correspondent à des valeurs de $\langle n \rangle$ allant jusqu'à ~ 7 photons, des mesures ont été prises avec des puissances de pompes générant jusqu'à ~ 300 photons.

Des versions des figures 3 et 4 de l'article incluant toutes les puissances balayées sont disponibles aux figures 4.8 et 4.9. Les non-linéarités évoquées à la section résultats de l'article y sont frappantes. On observe en effet un écart plus marqué entre la crête et les prévisions théoriques, avec des comportements à bas $\langle n \rangle$ drastiquement différents de la théorie pour les hautes puissances de pompes, mais avec le paramp désajusté en fréquence. Clairement, les effets d'ordres supérieurs deviennent importants lorsque la puissance de la pompe est substantielle [35–38]. Cependant, aux plus hautes puissances, on remarque que la crête diminue éventuellement avec $\langle n \rangle$ croissant, ce qui est probablement un artefact. Il est effectivement possible que cette structure reflète des effets de saturation dans le montage, similairement à la saturation observée à la figure

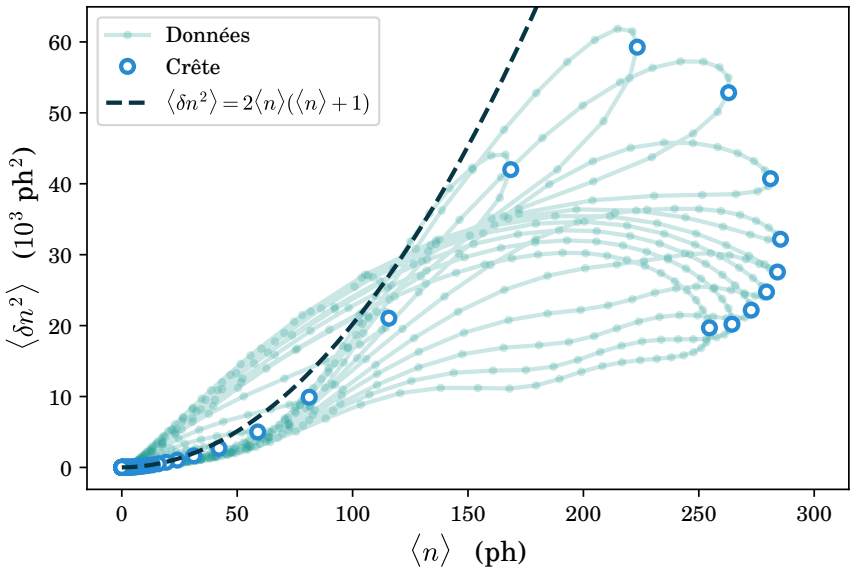


FIGURE 4.8 – Résultats complets de $\langle \delta n^2 \rangle$ vs $\langle n \rangle$. Figure associée à la figure 3 de l'article. Les lignes correspondent à des puissances de pompes données.

4.5 pour les plus hautes puissances. Une certaine prudence est donc de mise avant d'attribuer les résultats à grand $\langle n \rangle$ purement au paramp. Malgré tout, il va sans dire que le signal en sortie du paramp exhibe une statistique de photons des plus intéressante qui mériterait d'être étudiée plus en détail.

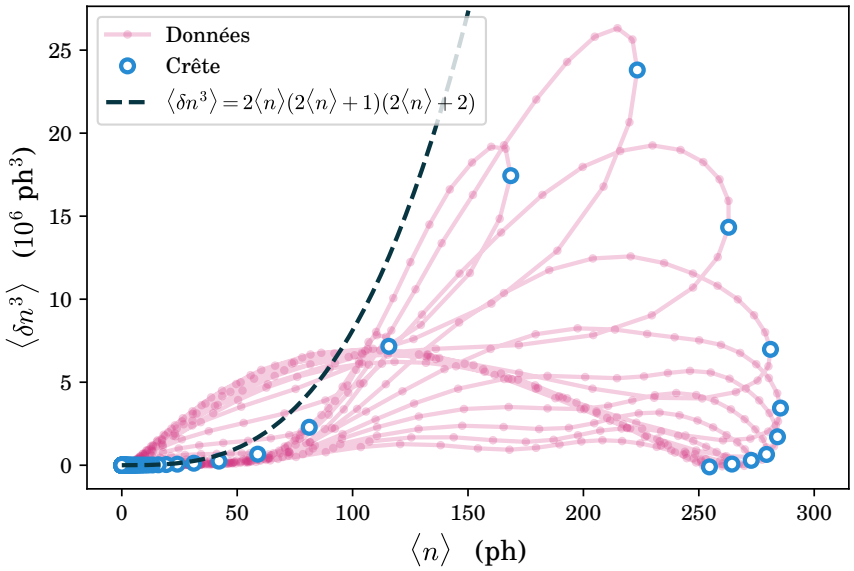


FIGURE 4.9 – Résultats complets de $\langle \delta n^3 \rangle$ vs $\langle n \rangle$. Figure associée à la figure 4 de l'article. Les lignes correspondent à des puissances de pompes données.

Chapitre 5

Synthèse : statistique de photons du paramp

En résumé, on a étudié l'amplificateur paramétrique Josephson en regardant les fluctuations électroniques qu'il émet lorsque son signal d'entrée est simplement le vide. La technique de conversion entre les fluctuations électroniques et la statistique de photons développée dans le cadre de travaux antérieurs [44–46] nous a permis d'obtenir les trois premiers moments de la statistique de photons à partir des six premiers cumulants des fluctuations électroniques. Des méthodes de calibration originales permettent d'isoler la contribution du paramp des autres sources de bruit du montage ; elles sont au coeur de la probité des résultats. Une panoplie de conditions expérimentales ont ainsi été étudiées en balayant tour à tour la fréquence de résonance et la puissance de pompe du paramp.

Ces résultats expérimentaux, de pair avec une modélisation théorique du paramp et de la procédure de détection [1], démontrent bien que — pour une détection intégrée sur la bande de détection — le paramp génère du vide comprimé à sa sortie lorsqu'il est ajusté à son point d'opération optimal. En dehors de celui-ci, cependant, des effets d'ordres supérieurs viennent modifier la statistique de photons observée de manière substantielle.

Ces travaux permettent aussi de mettre en lumière l'importance de la pro-

1. Traduction libre de l'anglais *detection scheme*.

cédure de détection sur le résultat obtenu pour la statistique de photons. En particulier, les prédictions théoriques de Padurariu et al. [43] ne concordent pas avec nos résultats, puisque la modélisation de la détection sur laquelle elles se basent ne correspond pas à la situation expérimentale typique. En effet, alors qu'on fait normalement des mesures intégrées sur une bande étroite de fréquences, sans discerner les fréquences à l'intérieur de celle-ci, cettedite référence présume plutôt que les fluctuations sont mesurées à chacune des fréquences incluses dans la bande passante[2]. Autrement dit, alors que nos résultats expérimentaux consistent — pour chaque condition expérimentale — en un seul ensemble de cumulants $\{C_2, C_4, C_6\}$ chacun intégré sur une largeur de bande Δf, la référence [43] se base sur la connaissance de $\{C_2(f), C_4(f), C_6(f)\}$ pour toutes les fréquences f résolues au sein de Δf.

Sommes toutes, les résultats présentés ici permettent de confirmer dans quelles conditions expérimentales et via quelle procédure de détection la statistique de photons à la sortie d'un paramp correspond au vide comprimé, tout en permettant de caractériser ce dernier dans toutes conditions expérimentales. L'approche adoptée ici est donc tout indiquée pour caractériser un amplificateur paramétrique Josephson et vérifier sa statistique de photons, que ce soit pour s'assurer que les effets d'ordre supérieurs soient négligeables ou encore pour les étudier directement. Comme la méthode expérimentale ne dépend pas de la composante observée, elle ouvre aussi la porte à l'étude en termes de statistique de photons de tout échantillon ayant une prédiction théorique intéressante.

Il serait aussi pertinent d'adapter la procédure de détection pour reproduire les prédictions de Padurariu et al. [43] et pour étudier l'utilité des différentes procédures de détections possibles, similairement à ce qui est fait en optique quantique [49]. Notamment, un traitement numérique du signal, inspiré de la partie II de la présente thèse, devrait permettre d'obtenir la résolution en fréquence sur laquelle les travaux de [43] s'appuient. On pourrait, par exemple, appliquer à la trace expérimentale une série de filtres indépendants étroits et centrés en différentes fréquences f, avant de calculer les cumulants pour chacun

2. Avec une certaine résolution de fréquence qui dépend en pratique des conditions expérimentales.

de ces filtres [3].

L'approche présentée ici quant à l'étude des fluctuations électroniques en termes de fluctuations de photons permet donc non seulement d'étudier des composantes aux propriétés intéressantes ; elle pave aussi la voie à l'étude de l'impact — et potentiellement de l'utilité — de la procédure de détection lors de mesures micro-ondes de signaux quantiques.

3. Il serait en théorie possible de mesurer directement les corrélateurs généraux à 2, 4 et 6 corps sur toute la bande ; essentiellement la généralisation à plus de deux temps des $\tilde{S}(t_1, t_2)$ discutées à la partie II. Cependant, cela nécessiterait *a priori* une puissance de calcul prohibitive.

Deuxième partie

Mesures de bruit ultrarapides

Chapitre 6

Fluctuations, fréquences, temps

L'étude des fluctuations électroniques — communément appelées le *bruit* électronique — est typiquement faite dans le domaine fréquentiel, ce qui est particulièrement judicieux lorsqu'on s'intéresse au régime quantique [15]. En effet, ce régime est typiquement atteint lorsque l'échelle d'énergie hf de la fréquence d'observation est supérieure aux autres échelles d'énergies [50] en jeu, comme par exemple celles associées à la température électronique $k_\mathrm{B}T_\mathrm{e}$ et à la polarisation en tension continue eV_dc. Il est donc tout à fait naturel d'adopter le domaine fréquentiel pour des mesures de bruit, d'autant plus que l'électronique micro-ondes est particulièrement appropriée à cette approche.

Or, le domaine temporel est tout aussi riche que le monde des fréquences ; il offre un point de vue différent et complémentaire à l'étude des fluctuations, en plus d'être mieux adapté à certaines situations expérimentales que son pendant fréquentiel. Notamment, le bruit d'un conducteur mésoscopique a une expression particulièrement élégante dans le domaine temporel [51], qui est valide aussi bien à l'équilibre que dans la situation hors équilibre et qui ne dépend que de la polarisation du conducteur au cours du temps. Aussi, des travaux récents se sont attardés au domaine temporel pour mettre en évidence simultanément les principes d'exclusion de Pauli et d'incertitude d'Heisenberg au sein de la jonction tunnel [52, 53]. Ces résultats ont cependant nécessité plusieurs mesures séparées en bande étroite dans le domaine fréquentiel pour obtenir des résultats temporels par analyse de Fourier.

En fait, une approche large bande dans le domaine temporel sérait particulièrement bien adaptée — et plus naturelle — pour ce type d'étude. Une grande résolution temporelle et une large bande passante sont cependant nécessaires pour de telles mesures [54, 55], ce qui présente un défi technique important. Cette approche permettrait aussi l'étude des corrélations temporelles au sein d'un signal à des échelles de temps plus courtes que la période d'excitation, similairement à ce qui est fait en optique quantique sous-cycle [1] [56, 57]. Les mesures dans le domaine temporel, bien qu'elles présentent un défi technique certain, promettent donc de donner accès à des propriétés autrement inaccessibles, ou à tout le moins extrêmement difficiles à étudier.

C'est pourquoi, dans le but d'élucider la physique régissant les corrélations temporelles présentes au sein des fluctuations électroniques, on s'applique ici à l'étude d'une jonction tunnel photoexcitée à l'aide de mesures ultrarapides en bande large. La jonction tunnel est un sujet d'étude idéal puisqu'elle est conceptuellement simple, qu'elle exhibe de riches corrélations et que son régime classique se prête bien à la calibration des effets du montage [53, 58–66]. On adopte ici une approche de traitement de signaux quantiques utilisant une carte d'acquisition ultrarapide large bande de pair avec une grande puissance de calcul, ce qui nous permet d'obtenir les corrélations temporelles au sein du signal pour des temps très courts, en plus d'en déduire des résultats large bande résolus en fréquence par analyse de Fourier. Synchroniser l'excitation de la jonction tunnel avec l'acquisition numérique nous permet aussi d'effectuer des mesures résolues en phase et d'ainsi étudier les corrélations sous-cycle au sein des fluctuations.

On présente donc, dans ce qui suit, une étude des fluctuations bande large émises par une jonction tunnel photoexcitée à l'aide de mesures effectuées dans le domaine temporel. Une attention particulière est portée à la calibration des effets du montage et de la chaîne d'amplification, incluant un traitement original dans le cas résolu en phase. On cherche *in fine* à explorer la réponse de la jonction tunnel à l'excitation à travers les fluctuations électroniques qui en émanent, autant pour mieux comprendre les phénomènes quantiques qui y sont en jeux que pour vérifier la prédiction de [51] et ses conséquences.

1. Traduction libre de l'anglais *subscycle quantum optics*.

Afin de bien définir les concepts et la terminologie [2] utilisés tout au long de la thèse, le chapitre 7 fait un survol théorique des quantités étudiées en s'attardant en particulier aux corrélateurs courant–courant et aux densités spectrales qu'on mesure expérimentalement. La section 7.6 de ce chapitre introduit quant à elle notre définition du spectre de bruit résolu en phase et de ses propriétés. Le chapitre 8 présente ensuite le montage expérimental et le système d'acquisition des données, en plus de présenter la technique d'ajustement de phase de la mesure. Le prétraitement numérique des données à la volée, qui est au coeur même de la probité des résultats, est décrit au chapitre 9. La calibration et les résultats pour les mesures non résolues en phase sont présentés de paire au chapitre 10, alors qu'ils sont respectivement répartis dans les chapitres 11 et 12 dans le cas des mesures résolues en phase. Finalement, le chapitre 13 fait la synthèse des résultats principaux du projet.

2. Une panoplie de conventions différentes sont couramment utilisées dans le domaine du traitement de signal, on s'efforce donc de bien définir celles qui sont adoptées dans le cadre des présents travaux, par souci de clarté et de complétude.

Chapitre 7

Aspects théoriques

7.1 Corrélations

7.1.1 Fonction de corrélation

La fonction de corrélation entre deux signaux est une mesure de la similarité entre ceux-ci. Comme son nom l'indique, ce n'est pas un simple scalaire ; c'est plutôt une fonction du décalage temporel τ entre les signaux, ce qui permet de quantifier la similitude entre des signaux qui serait désynchronisés ou dont une partie aurait subi des réflexions ou délais électriques. Conceptuellement, cela revient à faire *glisser* le second signal sur le premier et, pour chaque décalage, à utiliser le produit scalaire pour quantifier combien ceux-ci se ressemblent. Comme on s'intéresse *in fine* à des mesures de tension au cours du temps, une quantité réelle, on se limitera ici au cas des signaux d'entrée réels ; lorsque les conjugés sont utilisés, c'est par souci de généralité.

La fonction de corrélation [1] $r_{f,g}(t_1, t_2)$ entre deux signaux $f(t)$ et $g(t)$ est définie par

$$r_{f,g}(t_1, t_2) = \langle f(t_1)\, g^*(t_2) \rangle \tag{7.1}$$

1. Aussi appelée corrélation croisée ou *cross-correlation* en anglais.

où t_1 et t_2 sont deux temps de référence et où $\langle \cdot \rangle$ dénote l'espérance mathématique [67, §10 ; 68, §6.7]. Comme on s'intéresse à des signaux réels, on a

$$h^*(t) = h(t) \quad \forall\, h \in \{f, g\}. \tag{7.2}$$

Alors, par simple changement de variable $t = t_1 = t_2 - \tau$, on obtient [2]

$$r_{f,g}(t\,, t+\tau) = \langle f(t)\, g(t+\tau) \rangle\,, \tag{7.3}$$

ce qui est essentiellement, pour chaque valeur de τ, le produit scalaire entre $f(t)$ et $g_\tau(t) \equiv g(t+\tau)$. De plus, si leur valeur moyenne respective ne dépend pas du temps et que leur corrélation ne dépend que de l'écart entre les temps de référence t_1 et t_2, on dira que $f(t)$ et $g(t)$ sont *stationnaires au sens large*. Formellement, cela correspond à dire que, $\forall\, h \in \{f, g\}$,

$$\begin{aligned} \langle h(t) \rangle &= \langle h \rangle \quad \forall t \\ r_{h,h}(t_1\,, t_2) &= r_{h,h}(|t_2 - t_1|) \end{aligned} \tag{7.4}$$

et la fonction de corrélation $r_{f,g}(t_1\,, t_2)$ ne dépendra que de $|\tau| \equiv |t_2 - t_1|$. Ainsi,

$$r_{f,g}(t\,, t+\tau) = r_{f,g}(\tau) = r_{f,g}(-\tau)\,, \tag{7.5}$$

et l'équation (7.1) devient simplement [67, Theorem 6.8.1]

$$\boxed{r_{f,g}(\tau) = \langle f(t)\, g(t+\tau) \rangle}\,. \tag{7.6}$$

7.1.2 Autocorrelation

La fonction d'autocorrélation est, quant-à-elle, un cas particulier de la fonction de corrélation lorsqu'un signal est comparé à lui-même ; c'est essentiellement une généralisation du deuxième moment.

2. Plusieurs conventions existent dans la littérature avec différents signes pour $\pm\tau$ ou appliquant le décalage temporel sur $f(t)$ ou $g(t)$. Ces choix ne sont pas critiques dans la majorité des cas, mais il importe d'en tenir compte pour comparer des résultats de sources variées.

L'autocorrélation $r_f(t_1, t_2) \equiv r_{f,f}(t_1, t_2)$ est la fonction de corrélation du signal $f(t)$ avec lui-même. En général, l'équation (7.7) prend donc la forme

$$r_f(t_1, t_2) = \langle f(t_1)\, f(t_2) \rangle \,. \tag{7.7}$$

Dans le cas d'un signal réel et stationnaire au sens large, $f(t)$ va respecter les équations (7.2) et (7.4). Selon le même raisonnement qu'à la section 7.1.1 l'équation (7.7) revient alors à

$$\boxed{r_f(\tau) = \langle f(t)\, f(t+\tau) \rangle} \,. \tag{7.8}$$

7.1.3 Covariance et autocovariance

La covariance [3] et l'autocovariance sont les versions centrées de la corrélation et de l'autocorrélation [67, §6.7.3–5]. C'est l'analogue de la variance d'un processus aléatoire X, soit son second cumulant $\langle\langle X^2 \rangle\rangle = \langle (X - \langle X \rangle)^2 \rangle = \langle X^2 \rangle - \langle X \rangle^2$. Avec les hypothèses de signaux $f(t)$ et $g(t)$ réels et stationnaires de la section 7.1.1, la covariance $c_{f,g}(\tau)$ est donnée par [68, §10.2]

$$\begin{aligned} c_{f,g}(\tau) &= \big\langle \{f(t) - \langle f(t) \rangle\} \{g(t+\tau) - \langle g(t+\tau) \rangle\} \big\rangle & (7.9) \\ &= \langle f(t)\, g(t+\tau) \rangle - \langle f \rangle \langle g \rangle \,, & (7.10) \end{aligned}$$

via (7.8) on obtient

$$\boxed{c_{f,g}(\tau) = r_{f,g}(\tau) - \langle f \rangle \langle g \rangle} \tag{7.11}$$

et l'autocovariance $c_f(\tau) \equiv c_{f,f}(\tau)$ est simplement le cas $g = f$

$$\boxed{c_f(\tau) = r_f(\tau) - \langle f \rangle^2} \,. \tag{7.12}$$

3. Aussi appelée covariance croisée ou *cross-covariance* en anglais.

On voit donc que la variance en est le cas particulier à $\tau = 0$

$$c_f(0) = \left\langle f(t)^2 \right\rangle - \left\langle f(t) \right\rangle^2 \tag{7.13}$$

$$c_f(0) = \left\langle\!\left\langle f^2(t) \right\rangle\!\right\rangle . \tag{7.14}$$

On remarque aussi que, pour des signaux centrés, soit si $\langle f \rangle = \langle g \rangle = 0$, alors les covariances et corrélations coïncident, c'est-à-dire que $c_{f,g}(\tau) = r_{f,g}(\tau)$ et $c_f(\tau) = r_f(\tau)$.

En pratique, les signaux traités expérimentalement sont souvent de moyenne nulle et les covariances sont communément appelées corrélations par abus de langage. La confusion de nomenclature entre autocorrélation et autocovariance est d'ailleurs répandue dans la littérature des domaines de la statistique et du traitement de signal en ingénierie [69, §1.1].

7.2 Spectre de bruit

Bien que les fonctions de corrélation soient intéressantes en elles-mêmes, on s'intéresse typiquement plutôt à une quantité qui y est reliée : la densité spectrale de puissance. La section 7.2.3 décrit le lien entre l'autocorrélation et la densité spectrale, mais on se concentre pour l'instant à décrire cette dernière.

Pour éviter la confusion avec un simple changement de variable, on adopte la convention selon laquelle le tilde $\tilde{\cdot}$ dénote qu'une fonction est représentée dans le domaine fréquentiel, si bien que $\tilde{g}(\omega) = \mathcal{F}[g(t)](\omega)$ pour une fonction $g(t)$ arbitraire.

La densité spectrale de puissance $\tilde{S}(\omega)$ d'un signal en courant — aussi appelée densité spectrale de bruit, spectre de bruit ou spectre de puissance — décrit la participation de chaque composante fréquentielle d'un signal à sa puissance totale P. Elle est définie par [68, §10.5]

$$P = R \int_{-\infty}^{\infty} \tilde{S}(\omega) \, \frac{\mathrm{d}\omega}{2\pi} , \tag{7.15}$$

où $\omega = 2\pi f$ est la fréquence angulaire et où R est l'impédance de l'échantillon mesuré. Notons qu'elle est définie ici en A^2/Hz, alors que P est en Watts.

Remarquons que les mesures expérimentales de $\tilde{S}(\omega)$ sont quant à elles souvent présentées en température équivalente de bruit, donc en Kelvin, par comparaison avec la température T_N d'une résistance classique générant un bruit thermique $\tilde{S}_{\text{Therm.}}$ de niveau équivalent à celui mesuré, soit [70, §2.4.1]

$$\tilde{S}_{\text{Therm.}}^{[\mathrm{A}^2/\mathrm{Hz}]} = 2k_{\mathrm{B}}T_N/R \qquad \Longrightarrow \qquad \tilde{S}^{[\mathrm{K}]} = \tilde{S}^{[\mathrm{A}^2/\mathrm{Hz}]}\frac{R}{2k_{\mathrm{B}}}, \tag{7.16}$$

où les crochets en exposant dénotent les unités des différentes densités spectrales. Dans la littérature, le facteur $2k_{\mathrm{B}}$ est souvent remplacé par $4k_{\mathrm{B}}$ lorsque seules les fréquences positives sont considérées, ce qui est entre autres valide pour les signaux stationnaires au sens large avec $\tilde{S}(-\omega) = \tilde{S}(\omega)$.

La représentation de $\tilde{S}$ en Kelvin a l'avantage d'être exprimée en unités plus tangibles que des A^2/Hz et de ne pas dépendre de R dans le cas de la jonction tunnel. L'équation (7.16) montre d'ailleurs que cette représentation correspond simplement à la substitution $R \to 2k_{\mathrm{B}}$ dans (7.15).

7.2.1 Mesure expérimentale

La densité spectrale de puissance de l'équation (7.15) est une quantité définie sous une intégrale. Pour la mesurer, il faudrait donc avoir une résolution fréquentielle infinie, correspondant à une trace temporelle du signal de longueur infinie, ce qui est irréaliste en pratique. De plus, les mesures expérimentales sont typiquement intégrées dans le temps, ce qui ajoute une moyenne temporelle qui a un effet important sur la mesure. C'est pourquoi on introduit $\hat{\tilde{S}}(\omega)$, qui modélise l'approche de la mesure.

Dans ce qui suit, on utilise la notation $\hat{\cdot}$ pour spécifier une quantité telle qu'intégrée sur le temps de mesure, comparativement à sa prédiction théorique intrinsèque.

Soit le courant $i(t)$, un signal ayant une transformée de Fourier bien définie,

tel que

$$i(t) = \mathcal{F}^{-1}\left[\tilde{\imath}(\omega)\right](t) = \int_{-\infty}^{\infty} \tilde{\imath}(\omega)\, \mathrm{e}^{\mathrm{i}\omega t}\, \frac{\mathrm{d}\omega}{2\pi} \tag{7.17}$$

et

$$\tilde{\imath}(\omega) = \mathcal{F}\left[i(t)\right](\omega) = \int_{-\infty}^{\infty} i(t)\, \mathrm{e}^{-\mathrm{i}\omega t}\, \mathrm{d}t\,, \tag{7.18}$$

où i dénote l'unité imaginaire et où $\omega = 2\pi f$ est la fréquence angulaire. Ces définitions ne représentent pas ce qui est réellement accessible en laboratoire. En pratique, les mesures sont intégrées sur une période T et le signal accessible est plutôt [68, §10.9]

$$i_T(t) = \begin{cases} i(t) & \text{si} \quad -\frac{T}{2} \leq t \leq \frac{T}{2} \\ 0 & \text{sinon} \end{cases} \tag{7.19}$$

et, par définition,

$$\tilde{\imath}_T(\omega) = \int_{-\infty}^{\infty} i_T(t)\, \mathrm{e}^{-\mathrm{i}\omega t}\, \mathrm{d}t = \int_{-\frac{T}{2}}^{\frac{T}{2}} i(t)\, \mathrm{e}^{-\mathrm{i}\omega t}\, \mathrm{d}t. \tag{7.20}$$

Pour s'assurer d'avoir la bonne forme de $\hat{\tilde{S}}(\omega)$, la densité spectrale telle que mesurée en laboratoire, définissons-la à travers les deux définitions équivalentes suivantes de la puissance moyenne mesurée $\hat{P}$ d'un signal, soit

$$\hat{P} = R \int_{-\infty}^{\infty} \hat{\tilde{S}}(\omega)\, \frac{\mathrm{d}\omega}{2\pi} \tag{7.21}$$

et

$$\hat{P} = R \left\langle \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} \left|i_T(t)\right|^2\, \mathrm{d}t \right\rangle, \tag{7.22}$$

avec R la résistance de l'échantillon et où $i_T(t)$ est en ampères. Notons que, bien que l'on prenne ici la limite infinie, ce sont les bornes de cette intégrale

qui viennent limiter la résolution fréquentielle atteignable expérimentalement. Via (7.19), on peut pousser les bornes de cette intégrale à l'infini sans perte de généralité et utiliser l'identité de Parseval–Plancherel [71 ; 68, §10.9]

$$\int_{-\infty}^{\infty} |f(t)|^2 \, \mathrm{d}t = \int_{-\infty}^{\infty} \left|\tilde{f}(\omega)\right|^2 \frac{\mathrm{d}\omega}{2\pi} \tag{7.23}$$

pour obtenir

$$\hat{P} = R\left\langle \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} |i_T(t)|^2 \, \mathrm{d}t \right\rangle \tag{7.24}$$

$$= R\left\langle \lim_{T\to\infty} \frac{1}{T} \int_{-\infty}^{\infty} |i_T(t)|^2 \, \mathrm{d}t \right\rangle \tag{7.25}$$

$$= R\left\langle \lim_{T\to\infty} \frac{1}{T} \int_{-\infty}^{\infty} |\tilde{\imath}_T(\omega)|^2 \frac{\mathrm{d}\omega}{2\pi} \right\rangle. \tag{7.26}$$

On peut alors changer l'ordre de la limite et de l'intégrale grâce au théorème de convergence dominée de Lebesgue [72 ; 73, §5.6], qui est valide pourvu que $|\tilde{\imath}_T(\omega)|^2$ converge simplement vers $|\tilde{\imath}(\omega)|^2$ et qu'il $\exists\ g(\omega)$ t.q. $\left||\tilde{\imath}_T(\omega)|^2\right| \leq g(\omega)$, et ce pour tout T. En termes plus simples, c'est valide si $\tilde{\imath}_T(\omega)$ tend vers $\tilde{\imath}(\omega)$ en ne divergeant pour aucune valeur de T. La définition (7.19) respectant ces conditions, on obtient

$$\hat{P} = R \int_{-\infty}^{\infty} \underbrace{\lim_{T\to\infty} \frac{1}{T} \left\langle |\tilde{\imath}_T(\omega)|^2 \right\rangle}_{\hat{\tilde{S}}(\omega)} \frac{\mathrm{d}\omega}{2\pi}, \tag{7.27}$$

où on identifie $\hat{\tilde{S}}(\omega)$ via (7.21). Comme on s'intéresse à des courants réels $i_T(t) \in \mathbb{R}$, (7.20) implique $\tilde{\imath}_T^*(\omega) = \tilde{\imath}_T(-\omega)$ et (7.27) donne

$$\boxed{\hat{\tilde{S}}(\omega) = \lim_{T\to\infty} \frac{1}{T} \left\langle \tilde{\imath}_T(-\omega)\, \tilde{\imath}_T(\omega) \right\rangle}. \tag{7.28}$$

7.2.2 Mesure bande étroite

Lors d'une mesure de bruit typique en bande étroite, on vient sonder la densité spectrale sur une bande de largeur Δf centrée en f_0 tel que [4]

$$\hat{\tilde{S}}(f_0, \Delta f, \dots) = \frac{1}{\Delta f} \int_{f_0 - \frac{\Delta f}{2}}^{f_0 + \frac{\Delta f}{2}} \tilde{S}(f, \dots) \, \mathrm{d}f, \tag{7.29}$$

où $\hat{S}$ est le bruit expérimental et où $\dots$ représente les autres paramètres pertinents de l'expérience – que l'on omet dans la suite pour alléger le texte. On remarque que, dans la limite bande étroite idéale,

$$\hat{\tilde{S}}(f_0) = \lim_{\Delta f \to 0} \int_{f_0 - \frac{\Delta f}{2}}^{f_0 + \frac{\Delta f}{2}} \frac{\tilde{S}(f)}{\Delta f} \, \mathrm{d}f \tag{7.30}$$

$$= \int_{-\infty}^{\infty} \delta(f - f_0) \, \tilde{S}(f) \, \mathrm{d}f, \tag{7.31}$$

et donc

$$\hat{\tilde{S}}(f_0) = \tilde{S}(f_0). \tag{7.32}$$

C'est pourquoi $\hat{\tilde{S}}(f_0)$ mesurée expérimentalement est souvent appelée *densité spectrale* bien qu'elle ait techniquement été intégrée autour de f_0.

7.2.3 Mesure bande large : Wiener-Khintchine

En bande large, il est certainement possible d'intégrer sur les fréquences comme cela est fait en bande étroite, voir la section 7.2.2. Cependant, les mesures dans le domaine temporel offrent en réalité une plus grande richesse. En effet, le théorème de Wiener–Khintchine [74, 75] stipule que la densité spectrale en puissance $\tilde{S}(f)$ d'un signal correspond à la transformée de Fourier de son autocorrélation [67, §8.1.2 ; 68, §10.11].

4. Pour une mesure idéale faisant fi des détails expérimentaux. On ajouterait typiquement l'effet d'un amplificateur au minimum.

Pour démontrer ce résultat, prenons (7.28) et développons à l'aide de (7.20) pour obtenir

$$\hat{\tilde{S}}(\omega) = \lim_{T\to\infty} \frac{1}{T} \left\langle \int_{-\frac{T}{2}}^{\frac{T}{2}} \int_{-\frac{T}{2}}^{\frac{T}{2}} i(t_1) i(t_2)\, \mathrm{e}^{-\mathrm{i}\omega(t_1-t_2)}\, \mathrm{d}t_1\, \mathrm{d}t_2 \right\rangle. \tag{7.33}$$

Afin d'étendre l'intégrale centrale à $\mathbb{R}$, posons

$$\mathbb{1}_{[a,b]}(t) = \begin{cases} 1 & \text{si} \quad a \le t \le b \\ 0 & \text{sinon}, \end{cases} \tag{7.34}$$

si bien que

$$\hat{\tilde{S}}(\omega) = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} \int_{-\infty}^{\infty} \mathbb{1}_{\left[-\frac{T}{2},\frac{T}{2}\right]}(t_1) \underbrace{\langle i(t_1) i(t_2) \rangle}_{r_i(t_1,t_2)} \mathrm{e}^{-\mathrm{i}\omega(t_1-t_2)}\, \mathrm{d}t_1\, \mathrm{d}t_2, \tag{7.35}$$

où $r_i(t_1, t_2)$ découle de (7.7). Par changement de variable $t_1 = \tau + t_2$ dans l'intégrale centrale, l'expression devient

$$\hat{\tilde{S}}(\omega) = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} \int_{-\infty}^{\infty} \underbrace{\mathbb{1}_{\left[-\frac{T}{2},\frac{T}{2}\right]}(t_2+\tau)}_{\mathbb{1}_{\left[-\frac{T}{2}-\tau,\frac{T}{2}-\tau\right]}(t_2)} r_i(t_2+\tau, t_2)\, \mathrm{e}^{-\mathrm{i}\omega\tau}\, \mathrm{d}\tau\, \mathrm{d}t_2. \tag{7.36}$$

De plus, en posant $t = t_2$, en prenant la limite $T \gg \tau$ tel que $\pm T - \tau \approx \pm T$ et en changeant l'ordre des opérations [5], on obtient

$$\hat{\tilde{S}}(\omega) = \int_{-\infty}^{\infty} \mathrm{e}^{-\mathrm{i}\omega\tau} \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} \underbrace{\mathbb{1}_{\left[-\frac{T}{2},\frac{T}{2}\right]}(t)}_{1\ \forall\, t\in\left[\frac{-T}{2},\frac{T}{2}\right]} r_i(t+\tau, t)\, \mathrm{d}t\, \mathrm{d}\tau \tag{7.37}$$

$$= \int_{-\infty}^{\infty} \mathrm{e}^{-\mathrm{i}\omega\tau} \left(\lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} r_i(t+\tau, t)\, \mathrm{d}t \right) \mathrm{d}\tau. \tag{7.38}$$

On remarque que l'expression entre parenthèses ne dépendra pas de t. C'est en

5. Permis par le théorème de convergence dominée de Lebesgue [73, §5.6], voir la discussion sous l'équation (7.26).

fait la modélisation de l'autocorrélation intégrée sur le temps de mesure, qu'on note

$$\hat{r}_i(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} r_i(t+\tau, t)\,\mathrm{d}t. \tag{7.39}$$

L'intégrale temporelle vient donc, en quelque sorte, *stationnariser* l'autocorrélation, ce qui permet d'obtenir — via (7.38) — la densité spectrale de puissance

$$\hat{\tilde{S}}(\omega) = \int_{-\infty}^{\infty} \hat{r}_i(\tau)\,\mathrm{e}^{-\mathrm{i}\omega\tau}\,\mathrm{d}\tau \tag{7.40}$$

sans égards aux propritétés du signal d'entré. Cela mène directement au résultat du théorème de Wiener–Khintchine, soit

$$\hat{\tilde{S}}(\omega) = \mathcal{F}\left[\hat{r}_i(\tau)\right](\omega), \tag{7.41}$$

avec le corollaire

$$\hat{r}_i(\tau) = \mathcal{F}^{-1}\left[\hat{\tilde{S}}(\omega)\right](\tau). \tag{7.42}$$

En pratique, lorsqu'on étudie les fluctuations, on pose $\hat{S} \equiv \hat{r}_i$ et les équations modélisant la mesure sont

$$\boxed{\hat{S}(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} S(t+\tau, t)\,\mathrm{d}t} \tag{7.43}$$

et

$$\boxed{\hat{\tilde{S}}(\omega) = \mathcal{F}\left[\hat{S}(\tau)\right](\omega)}. \tag{7.44}$$

Ce résultat est valide peu importe les propriétés de stationnarité du signal d'entrée $i(t)$. En fait, aucune présomption n'est faite sur les propriétés du signal,

sinon qu'il est réel est que sa transformée de Fourier existe[6]. Le moyennage dans le temps implémenté par l'intégrale sur t vient ici éliminer, ou tracer, l'un des deux degrés de liberté de la forme générale de l'autocorrélation (7.7), ce qui permet de définir la densité spectrale de puissance peu importe les propriétés du signal. C'est une approche pragmatique très utile en laboratoire, puisqu'on ne connaît pas a priori les propriétés du signal étudié. De plus, il n'est pas évident d'envisager une approche qui permettrait de mesurer correctement $\hat{S}(t_1, t_2)$ en général, sans devoir conserver la trace complète de $i(t)$ faute de référence temporelle.

Le fait que la densité spectrale de puissance et le corrélateur courant–courant sont reliés par la transformée de Fourier permet de déduire certaines propriétés de symétries intéressantes. En particulier, si le signal mesuré est réel, alors l'autocorrélation le sera aussi et la densité spectrale sera par conséquent hermitienne [76, §4.1.2.1] pour respecter (7.44). Conceptuellement,

$$i_T(t) \in \mathbb{R} \quad \Rightarrow \quad \hat{S}(\tau) \in \mathbb{R} \quad \Rightarrow \quad \hat{\tilde{S}}^*(\omega) = \hat{\tilde{S}}(-\omega)\,. \tag{7.45}$$

De plus, comme l'autocorrélation expérimentale telle que définie ici est réelle et symétrique[7] selon τ, on peut arbitrairement changer le signe de τ dans (7.40) et remarquer que la densité spectrale sera symétrique en ω. En termes plus mathématiques,

$$\hat{S}(-\tau) = \hat{S}(\tau) \quad \text{et} \quad \hat{S}(\tau) \in \mathbb{R} \quad \Rightarrow \quad \hat{\tilde{S}}(\omega) = \hat{\tilde{S}}(-\omega)\,. \tag{7.46}$$

Ces propriétés seront particulièrement importantes à la section 9.1.2 pour optimiser le traitement des données à la volée pendant les acquisitions.

Remarquons aussi que, si $i(t)$ est stationnaire au sens large — voir la section 7.1.1 — alors $S(t+\tau, t) = S(\tau)$ et (7.43) implique donc $\hat{S}(\tau) = S(\tau)$. Dans ce cas, les mesures à $T \to \infty$ coïncident avec la formulation théorique de la densité spectrale, c'est-à-dire que les *chapeaux* sont caduques et que $\hat{\tilde{S}}(\omega) = \tilde{S}(\omega)$. Bien sûr, en pratique, on aura plutôt $\hat{S}(\tau) \approx S(\tau)$ et $\hat{\tilde{S}}(\omega) \approx \tilde{S}(\omega)$; le délai temporel, la

6. Ce qui est toujours vrai pour un signal numérisé, la transformée de Fourier numérique — ou FFT, de l'anglais *Fast Fourier Transform* — remplaçant la transformée de Fourier analytique.
7. Ce qui n'est pas le cas des autocorrélations résolues en phase — voir la section 7.6.

résolution spectrale et le ratio signal sur bruit étant contraints par les limites expérimentales[8].

Les résultats des équations (7.43) et (7.44) rendent l'autocorrélation très attrayante pour des mesures en bande large. Plutôt que d'intégrer sur toute la bande de mesure — comme on le fait typiquement en bande étroite — l'autocorrélation donne accès à la même information que plusieurs mesures en bande étroite de $\tilde{S}(f)$, sous respect des limites expérimentales[8].

7.3 Bruit à l'équilibre

On considère qu'une jonction tunnel est à l'équilibre si elle n'est soumise à aucun biais externe en tension ou courant, c'est-à-dire que $V(t) = 0$, et qu'elle est à l'équilibre thermique ; c'est-à-dire que ses deux contacts ont la même température stable.

7.3.1 Densité spectrale

Sous les conditions d'équilibre énoncées ci-haut, la densité spectrale de bruit d'une jonction tunnel de résistance R est donnée par [70, §3.2.1 ; 51]

$$\tilde{S}_{\mathrm{eq}}(\omega) = \frac{\hbar\omega}{R} \coth\left(\frac{\hbar\omega}{2k_{\mathrm{B}}T_{\mathrm{e}}}\right), \tag{7.47}$$

qu'on peut réexprimer

$$\boxed{\tilde{S}_{\mathrm{eq}}(\omega) = \frac{2k_{\mathrm{B}}T_{\mathrm{e}}}{R} \frac{\hbar\omega}{2k_{\mathrm{B}}T_{\mathrm{e}}} \coth\left(\frac{\hbar\omega}{2k_{\mathrm{B}}T_{\mathrm{e}}}\right)}, \tag{7.48}$$

avec T_{e} la température des électrons participant au transport et $\omega = 2\pi f$ la fréquence angulaire associée à la fréquence f de la mesure.

8. Soit, de manière non exhaustive, le temps d'intégration fini, le $\tau_{\max}$ fini dans le calcul des autocorrélations, les limites de la transformée de Fourier [54, 55], les appareils non idéaux, le bruit parasite, etc.

L'autocorrélation à l'équilibre est donc une fonction réelle et paire, telle que

$$\tilde{S}_{\mathrm{eq}}(\omega) \in \mathbb{R} \qquad \text{et} \qquad \tilde{S}_{\mathrm{eq}}(-\omega) = \tilde{S}_{\mathrm{eq}}(\omega)\,. \tag{7.49}$$

De plus, sachant que

$$\lim_{x\to 0} x\coth(x) = 1 \tag{7.50}$$

$$x\coth(x) \approx |x| \quad \text{si} \quad x \gg 1\,, \tag{7.51}$$

on peut aisément calculer les limites de (7.48). Dans la limite fréquence nulle $\hbar\omega \ll k_{\mathrm{B}}T_{\mathrm{e}}$, l'expression donne le résultat attendu pour le bruit thermique, soit [77–79]

$$\tilde{S}_{\mathrm{eq}}\left(\omega \ll \frac{k_{\mathrm{B}}T_{\mathrm{e}}}{\hbar}\right) = \frac{2k_{\mathrm{B}}T_{\mathrm{e}}}{R}\,, \tag{7.52}$$

alors que dans la limite quantique $\hbar\omega \gg k_{\mathrm{B}}T_{\mathrm{e}}$, on obtient plutôt le bruit quantique de point zéro [79]

$$\tilde{S}_{\mathrm{eq}}\left(\omega \gg \frac{k_{\mathrm{B}}T_{\mathrm{e}}}{\hbar}\right) = \frac{|\hbar\omega|}{R}\,. \tag{7.53}$$

Ce dernier résultat est encore plus frappant lorsqu'exprimé en Kelvin, tel que décrit à l'équation 7.16, soit

$$\tilde{S}_{\mathrm{eq}}\left(\omega \gg \frac{k_{\mathrm{B}}T_{\mathrm{e}}}{\hbar}\right) \times \frac{R}{2k_{\mathrm{B}}} = \frac{1}{2}\frac{|\hbar\omega|}{k_{\mathrm{B}}}\,, \tag{7.54}$$

ce qui correspond au demi–photon des fluctuations du vide.

Ainsi, le bruit à l'équilibre décrit les contributions des fluctuations du vide et du bruit thermique, avec des amplitudes associées qui dépendent de la grandeur relative des échelles d'énergies en jeu.

7.3.2 Corrélateur courant–courant

On obtient le corrélateur courant–courant dans le domaine temporel, soit l'autocorrélation $S_{\mathrm{eq}}(\tau)$, en prenant la transformée de Fourier inverse de $\tilde{S}_{\mathrm{eq}}(\omega)$, c'est-à-dire

$$S_{\mathrm{eq}}(\tau) = \mathcal{F}^{-1}\left[\tilde{S}_{\mathrm{eq}}(\omega)\right](\tau) \tag{7.55}$$

$$= \frac{2k_{\mathrm{B}}T_{\mathrm{e}}}{R}\int_{-\infty}^{\infty}\frac{\hbar\omega}{2k_{\mathrm{B}}T_{\mathrm{e}}}\coth\left(\frac{\hbar\omega}{2k_{\mathrm{B}}T_{\mathrm{e}}}\right)\mathrm{e}^{\mathrm{i}\omega\tau}\frac{\mathrm{d}\omega}{2\pi}, \tag{7.56}$$

ce qui donne [51]

$$\boxed{S_{\mathrm{eq}}(\tau) = -\frac{\pi\left(k_{\mathrm{B}}T_{\mathrm{e}}\right)^2}{R\hbar}\frac{1}{\sinh^2\left(\frac{\pi k_{\mathrm{B}}T_{\mathrm{e}}}{\hbar}\tau\right)}}, \tag{7.57}$$

voir le code *Mathematica* à l'annexe D.2.1. [9] Ainsi, à l'instar de la densité spectrale à l'équilibre, l'autocorrélation est réelle et symétrique en τ :

$$S_{\mathrm{eq}}(\tau) \in \mathbb{R} \qquad \text{et} \qquad S_{\mathrm{eq}}(-\tau) = S_{\mathrm{eq}}(\tau). \tag{7.58}$$

Puisque $\left\{T_{\mathrm{e}}, R, \sinh^2(x)\right\} \geq 0 \ \forall\, x$, on aura l'intéressante propriété

$$S_{\mathrm{eq}}(\tau) \leq 0 \qquad \forall\ \tau, \tag{7.59}$$

qui est simplement l'expression du principe d'exclusion de Pauli [80]. Ce dernier étant d'autant plus flagrant dans la limite à temps nul

$$\lim_{\tau\to 0} S_{\mathrm{eq}}(\tau) = -\infty, \tag{7.60}$$

9. Notons que les équations (7.48) et (7.57) ne se comportent pas particulièrement bien au sens de la transformée de Fourier et qu'on néglige probablement ici des termes constants ou associés au delta de Dirac. Cependant, comme discuté aux sections 10.1 et 11.1.1, la procédure de calibration de la mesure implique qu'on s'intéresse expérimentalement à des bruits en excès. Ces termes se voient donc absorbés dans le bruit d'amplification et peuvent donc être omis.

où l'anticorrélation infinie traduit le fait qu'il est *impossible* que deux électrons sautent en même temps à travers la barrière tunnel vers le même canal de conduction. Autrement dit, puisque les électrons obéissent à une statistique de Fermi–Dirac [81, §9.1], au moment où un électron subit un événement tunnel à travers la jonction, un autre électron *ne peut absolument pas* effectuer un saut vers le même micro-état d'arrivée. Évidemment, par les mêmes arguments, le corollaire est

$$\lim_{\tau\to\infty} S_{\mathrm{eq}}(\tau) = 0; \tag{7.61}$$

les électrons ne sont pas du tout corrélés à grand temps.

On remarque aussi qu'à basse température, soit $k_{\mathrm{B}}T_{\mathrm{e}} < h/\tau$, $S_{\mathrm{eq}}(\tau)$ va en $\frac{-1}{\tau^2}$, spécifiquement

$$\lim_{T_{\mathrm{e}}\to 0} S_{\mathrm{eq}}(\tau) = \frac{-1}{\tau^2}\frac{\hbar}{\pi R}, \tag{7.62}$$

ce qui est cohérent avec la limite $T_{\mathrm{e}} \ll \hbar\omega/k_{\mathrm{B}}$ de (7.53), puisque, la transformée de Fourier inverse de la fonction valeur absolue est en général $\mathcal{F}^{-1}\left[|p|\right](x) = \frac{-1}{\pi x^2}$, avec x et p des variables conjugées [82, p. A-6].

7.4 Bruit de grenaille hors-équilibre

7.4.1 Forme générale

À l'aide du formalisme de Keldysh [83], la référence [51] dérive la forme générale des fluctuations $S(t_1, t_2)$ attendues pour un conducteur mésoscopique hors équilibre en fonction de la tension $V(t)$ à laquelle il est soumis. Le modèle utilisé correspondant à un fil mésoscopique [84], le résultat contient donc un facteur de Fano de $\mathfrak{F} = \frac{1}{3}$. En général, pour un facteur de Fano $\mathfrak{F}$ donné, l'expression du corrélateur courant–courant sera

$$S(t_1, t_2) = S_{\mathrm{eq}}(t_2 - t_1)\Big(1 - \mathfrak{F}\Big[\big(1 - \cos\left[\phi(t_1, t_2)\right]\big)\Big]\Big), \tag{7.63}$$

avec

$$\phi(t_1\,,t_2) = \frac{e}{\hbar}\int_{t_1}^{t_2} V(t)\,\mathrm{d}t \tag{7.64}$$

et où $S_{\mathrm{eq}}(t)$ est le bruit à l'équilibre décrit à la section 7.3.2. Puisqu'on s'intéresse ici à une jonction tunnel [70, §2.4.3], on prend $\mathfrak{F} = 1$. On peut exprimer les équations (7.63) et (7.64) en fonction d'un temps de référence t et d'un décalage τ par changement de variable $t = t_1 = t_2 - \tau$, soit

$$\boxed{S(t\,,t+\tau) = S_{\mathrm{eq}}(\tau)\cos\left(\frac{e}{\hbar}\int_{t}^{t+\tau} V(t')\,\mathrm{d}t'\right)}\,. \tag{7.65}$$

En utilisant la modélisation de la mesure expérimentale intégrée sur le temps, soit l'équation (7.43) :

$$\hat{S}(\tau) = \lim_{T\to\infty}\frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} S\left(t\,,t+\tau\right)\,\mathrm{d}t\,, \tag{7.66}$$

on trouve

$$\boxed{\hat{S}(\tau) = \lim_{T\to\infty}\frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} S_{\mathrm{eq}}(\tau)\cos\left(\frac{e}{\hbar}\int_{t}^{t+\tau} V(t')\,\mathrm{d}t'\right)\mathrm{d}t}\,. \tag{7.67}$$

Ce résultat est particulièrement intéressant, puisqu'il implique que $\hat{S}(\tau)$ ne dépend que de la moyenne de l'intégrale de $V(t)$ sur l'intervalle τ, indépendamment de la forme de $V(t)$ ou de son historique.

7.4.2 Mesure du bruit photoexcité

Dans le cas d'une jonction tunnel soumise à la fois à un biais DC d'amplitude V_{dc} et à une photoexcitation sinusoïdale AC d'amplitude V_{ac}, la tension au cours

du temps prend la forme

$$V(t) = V_{\mathrm{ac}} \cos(\Omega t) + V_{\mathrm{dc}}\,, \tag{7.68}$$

où Ω est la fréquence de photoexcitation et où on a pris, à $t = 0$, la phase de référence de l'excitation $\Delta\phi = 0$ pour simplifier. Ce choix correspond en fait simplement à une translation $t \to t + \varphi/\Omega$ dans le temps. Il est valide pour la démarche qui suit si $2\pi/\Omega \ll T$ dans (7.67), ou bien si la mesure est incohérente de manière à moyenner le cosinus à 0 et d'avoir $\langle V(t)\rangle = V_{\mathrm{dc}}$.

On appelle le *bruit photoexcité* $\hat{S}^{\mathrm{pa}}$ le bruit attendu aux bornes d'une jonction tunnel soumise à l'excitation (7.68) et dont la mesure est modélisée par (7.67). De ces équations, on trouve

$$\hat{S}^{\mathrm{pa}}(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} S_{\mathrm{eq}}(\tau) \cos\left\{ \frac{eV_{\mathrm{ac}}}{\hbar} \int_t^{t+\tau} \cos\left(\Omega t'\right) \mathrm{d}t' + \frac{eV_{\mathrm{dc}}}{\hbar} \int_t^{t+\tau} \mathrm{d}t' \right\} \mathrm{d}t \tag{7.69}$$

$$= \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} S_{\mathrm{eq}}(\tau) \cos\left\{ z\big[\sin(\Omega t + \Omega\tau) - \sin(\Omega t)\big] + \nu\tau \right\} \mathrm{d}t\,, \tag{7.70}$$

avec $z = \frac{eV_{\mathrm{ac}}}{\hbar\Omega}$ et $\nu = \frac{eV_{\mathrm{dc}}}{\hbar}$. Pour développer cette expression, on utilisera $\cos\theta = \left(\mathrm{e}^{\mathrm{i}\theta} + \mathrm{e}^{-\mathrm{i}\theta}\right)/2$ et l'expansion de Jacobi–Anger [85, Éq. (17.1.7) avec $\varphi = \theta + \frac{\pi}{2}$]

$$\mathrm{e}^{\mathrm{i}z\sin\varphi} = \sum_{n=-\infty}^{\infty} J_n(z)\,\mathrm{e}^{\mathrm{i}n\varphi}\,, \quad n \in \mathbb{Z}, \tag{7.71}$$

faisant intervenir les fonctions de Bessel du premier type $J_n(z)$. L'expression prend alors la forme

$$\hat{S}^{\mathrm{pa}}(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} \frac{S_{\mathrm{eq}}(\tau)}{2} \Big(\mathrm{e}^{\mathrm{i}[z\sin(\Omega t+\Omega\tau) - z\sin(\Omega t) + \nu\tau]} + \mathrm{e}^{-\mathrm{i}[z\sin(\Omega t+\Omega\tau) - z\sin(\Omega t) + \nu\tau]} \Big) \mathrm{d}t \tag{7.72}$$

$$= \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} \frac{S_{\text{eq}}(\tau)}{2} \left(\sum_n \sum_m J_n(z) J_m(z)\, \mathrm{e}^{\mathrm{i}(n-m)\Omega t}\, \mathrm{e}^{\mathrm{i}(n\Omega+\nu)\tau} \right.$$
$$\left. + \sum_k \sum_l J_k(z) J_l(z)\, \mathrm{e}^{-\mathrm{i}(k-l)\Omega t}\, \mathrm{e}^{-\mathrm{i}(k\Omega+\nu)\tau} \right) \mathrm{d}t\,, \qquad (7.73)$$

où les sommes sont considérées sur l'intervalle $]-\infty, \infty[$ implicitement. Si on se concentre uniquement sur les termes participant à l'intégrale, on obtiendra deux termes de la forme

$$\lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} \mathrm{e}^{\mathrm{i}(n-m)\Omega t}\, \mathrm{d}t = \lim_{T\to\infty} \operatorname{sinc}\left(\pi\,(n-m)\, \frac{\Omega}{2\pi} T \right). \qquad (7.74)$$

Or, il se trouve que c'est la forme du sinus cardinal normalisé tel que [86, §4]

$$\operatorname{sinc}(\pi\kappa) = \delta_{\kappa,0} \qquad \kappa \in \mathbb{Z} \qquad (7.75)$$
$$\lim_{\kappa\to\infty} \operatorname{sinc}(\pi\kappa) = 0. \qquad (7.76)$$

avec $\delta_{a,b}$ le delta de Kronecker, valide pour $\{a\,, b\} \in \mathbb{Z}$, valant 1 si $a = b$ et 0 sinon. Dans notre situation $\kappa = (n-m)\,\frac{\Omega}{2\pi} T$ et $T \to \infty$, si bien que l'équation (7.74) sera non-nulle seulement si $\kappa = 0$, auquel cas $\kappa \in \mathbb{Z}$ et l'équation vaudra 1. Cette condition sera respectée dans deux situations, soit $\Omega = 0$ et $n - m = 0$. On ignore le premier cas, puisqu'on s'intéresse au bruit photoexcité avec $\Omega \neq 0$ par hypothèse [10]. On peut donc identifier

$$\delta_{n,m} = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} \mathrm{e}^{\mathrm{i}(n-m)\Omega t}\, \mathrm{d}t \qquad \text{avec} \qquad \{n\,, m\} \in \mathbb{Z}\,, \qquad (7.77)$$

et, réduisant les sommes et profitant du fait que leurs indices sont muets, (7.73) devient

$$\hat{S}^{\text{pa}}(\tau) = \frac{S_{\text{eq}}(\tau)}{2} \left(\sum_n \sum_m J_n(z) J_m(z) \delta_{n,m}\, \mathrm{e}^{\mathrm{i}(n\Omega+\nu)\tau} \right.$$

10. Si on persiste à prendre $\Omega = 0$, V_{dc} sera simplement renormalisé dans (7.68) et la suite du développement sera indépendante de t, si bien que le sinc et le $\delta_{n,m}$ n'émergeront jamais.

$$+ \sum_k \sum_l J_k(z) J_l(z) \delta_{k,l} \, \mathrm{e}^{-\mathrm{i}(k\Omega+\nu)\tau} \Bigg) \tag{7.78}$$

$$= \frac{S_{\mathrm{eq}}(\tau)}{2} \left(\sum_n J_n^2(z) \left(\mathrm{e}^{\mathrm{i}(n\Omega+\nu)\tau} + \mathrm{e}^{-\mathrm{i}(n\Omega+\nu)\tau} \right) \right). \tag{7.79}$$

Comme on s'intéresse à la densité spectrale $\hat{\tilde{S}}^{\mathrm{pa}}(\omega) = \mathcal{F}\left[\hat{S}^{\mathrm{pa}}(\tau)\right](\omega)$, on prend la transformée de Fourier de l'équation précédente pour obtenir

$$\hat{\tilde{S}}^{\mathrm{pa}}(\omega) = \int_{-\infty}^{\infty} \hat{S}^{\mathrm{pa}}(\tau) \, \mathrm{e}^{-\mathrm{i}\omega\tau} \, \mathrm{d}\tau \tag{7.80}$$

$$= \sum_n \frac{J_n^2(z)}{2} \left(\int_{-\infty}^{\infty} S_{\mathrm{eq}}(\tau) \, \mathrm{e}^{\mathrm{i}(n\Omega+\nu-\omega)\tau} \, \mathrm{d}\tau + \int_{-\infty}^{\infty} S_{\mathrm{eq}}(\tau) \, \mathrm{e}^{-\mathrm{i}(n\Omega+\nu+\omega)\tau} \, \mathrm{d}\tau \right). \tag{7.81}$$

En posant $\omega_n^{\pm} = \pm(n\Omega + \nu \pm \omega)$, les intégrales prennent la forme de transformées de Fourier et

$$\hat{\tilde{S}}^{\mathrm{pa}}(\omega) = \sum_n \frac{J_n^2(z)}{2} \left(\int_{-\infty}^{\infty} S_{\mathrm{eq}}(\tau) \, \mathrm{e}^{-\mathrm{i}\omega_n^- \tau} \, \mathrm{d}\tau + \int_{-\infty}^{\infty} S_{\mathrm{eq}}(\tau) \, \mathrm{e}^{-\mathrm{i}\omega_n^+ \tau} \, \mathrm{d}\tau \right) \tag{7.82}$$

$$= \sum_n J_n^2(z) \left(\frac{\tilde{S}_{\mathrm{eq}}(\omega_n^- \tau) + \tilde{S}_{\mathrm{eq}}(\omega_n^+ \tau)}{2} \right) \tag{7.83}$$

$$= \sum_n J_n^2(z) \left(\frac{\tilde{S}_{\mathrm{eq}}(\omega - n\Omega - \nu) + \tilde{S}_{\mathrm{eq}}(\omega + n\Omega + \nu)}{2} \right). \tag{7.84}$$

Le carré de l'identité de la référence [85, Éq. (17.1.5)], soit $J_{-n}^2(z) = J_n^2(z)$, et la parité du bruit à l'équilibre, voir la section 7.3, impliquent que seul le signe relatif entre ω et ν importe dans l'argument de $\tilde{S}_{\mathrm{eq}}$. On utilise donc cette propriété pour obtenir

$$\hat{\tilde{S}}^{\mathrm{pa}}(\omega) = \sum_n J_n^2(z) \underbrace{\left\{ \frac{\tilde{S}_{\mathrm{eq}}\Big(\nu - (\omega + n\Omega)\Big) + \tilde{S}_{\mathrm{eq}}\Big(\nu + (\omega + n\Omega)\Big)}{2} \right\}}_{\equiv \tilde{S}^{\mathrm{dc}}(\omega + n\Omega, \nu)}, \tag{7.85}$$

où

$$\tilde{S}^{\mathrm{dc}}(\omega\,,\,\nu) = \frac{\tilde{S}_{\mathrm{eq}}\Big(\nu-\omega\Big) + \tilde{S}_{\mathrm{eq}}\Big(\nu+\omega\Big)}{2} \tag{7.86}$$

n'est autre que le bruit attendu d'une jonction tunnel soumise à un biais $V_{\mathrm{dc}} = \nu\hbar/e$ et mesurée à la fréquence $f = \pm\omega/(2\pi)$ [87]. On obtient ainsi le bruit photoassisté en fonction du bruit sous biais dc tel que décrit dans la littérature, soit [58, 61, 88, 89]

$$\boxed{\hat{\tilde{S}}^{\mathrm{pa}}(\omega) = \sum_{n=-\infty}^{\infty} J_n^2\,(z)\,\tilde{S}^{\mathrm{dc}}(\omega + n\Omega\,,\,\nu)}\,. \tag{7.87}$$

C'est donc dire que le spectre de bruit photoexcité consiste conceptuellement en une somme infinie, pondérée par les $J_n^2(z)$, de contributions de la forme de $\tilde{S}^{\mathrm{dc}}(\omega_n\,,\,\nu)$ avec différents ω_n translatés de $n\Omega$.

Il est aussi possible de choisir le signe devant n et de regrouper les termes pour montrer que $\tilde{S}^{\mathrm{dc}}(\omega+n\Omega\,,\,\nu) = \tilde{S}^{\mathrm{dc}}(\omega\,,\,\nu+n\Omega)$; la photoexcitation correspond donc pareillement à prendre $\tilde{S}^{\mathrm{dc}}(\omega)$ et à y ajouter des répliques de lui-même avec ν décalé de $n\Omega$, selon une pondération modulée par les $J_n^2(z)$.

7.5 Domaine temporel : Bruits en excès

7.5.1 Motivation

Les spectres de bruits discutés jusqu'à présent se comportent tous comme des fonctions valeurs absolues pour les très grandes fréquences. En effet, dans cette limite — bruit du vide oblige — la densité spectrale sera toujours $\tilde{S}\left(\omega \gg \frac{\mathcal{E}}{\hbar}\right) = \frac{2|\hbar\omega|}{R}$, avec $\mathcal{E}$ représentant toutes les échelles d'énergie pertinentes autres que la fréquence de mesure $f = \omega/2\pi$. Ce résultat n'est pas surprenant ; on mesure toujours le bruit du vide si l'échelle d'énergie de la fréquence de mesure est beaucoup plus grande que celles des autres phénomènes en jeu.

Toutefois, la bande passante d'une mesure est toujours finie ; les mesures expérimentales de densités spectrales se voient donc limitées à une fréquence maximale $f_{\max}$. Dans le cas d'une mesure bien calibrée, ce sera donc simplement l'équivalent de l'application d'un filtre passe–bas, ou filtre carré, à la fréquence $f_{\max}$. Dans le cas d'un spectre de bruit dans le domaine fréquentiel, cette situation est tout-à-fait appropriée, mais elle est inadéquate si on veut plutôt étudier l'autocorrélation dans le domaine temporel. La multiplication par une fonction *porte rectangulaire* dans le domaine fréquentiel vient effectivement convoluer le résultat temporel avec une fonction sinus cardinal qui brouille les détails de celui-ci [86, §4] ; c'est simplement la conséquence du théorème de convolution.

Si on s'intéresse à des résultats dans le domaine temporel, il importe donc de regarder des quantités qui se comportent bien à $f \gtrsim f_{\max}$; typiquement via des soustractions judicieuses.

Deux telles soustractions sont présentées aux sections 7.5.2 et 7.5.3 dans le cas d'une jonction tunnel soumise à un biais en tension continue, soit respectivement les contributions de la tension de biais et de la température au bruit total émis par la jonction.

7.5.2 Bruit en excès DC

Une quantité respectant les critères établis à la section 7.5.1 pour les mesures dans le domaine temporel est le *bruit en excès DC*. Il s'agit de la contribution du biais en tension continue au bruit total d'une jonction tunnel soumise à celui-ci. Concrètement cela consiste simplement à soustraire le bruit mesuré à $V_{\mathrm{dc}} = 0$ de celui pour $V_{\mathrm{dc}} \neq 0$.

Soit $\hat{S}^{\mathrm{dc}}(\tau, V_{\mathrm{dc}})$, le corrélateur courant–courant attendu pour le signal émit par une jonction tunnel soumise au biais $V(t) = V_{\mathrm{dc}}$. Via (7.67), on calcule

$$\hat{S}^{\mathrm{dc}}(\tau, V_{\mathrm{dc}}) = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} S_{\mathrm{eq}}(\tau) \cos\left(\frac{e}{\hbar} \int_{t}^{t+\tau} V_{\mathrm{dc}}\, \mathrm{d}t'\right) \mathrm{d}t \tag{7.88}$$

$$= \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} S_{\text{eq}}(\tau) \cos\left(\frac{e}{\hbar} V_{\text{dc}} \tau\right) \mathrm{d}t \tag{7.89}$$

$$= \underbrace{\left(\lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} \mathrm{d}t\right)}_{1} S_{\text{eq}}(\tau) \cos\left(\frac{e}{\hbar} V_{\text{dc}} \tau\right). \tag{7.90}$$

Puisque la moyenne temporelle est disparue — le signal étant indubitablement stationnaire au sens large — le *chapeau* est caduc et on obtient

$$\boxed{S^{\text{dc}}(\tau, V_{\text{dc}}) = S_{\text{eq}}(\tau) \cos\left(\frac{e V_{\text{dc}} \tau}{\hbar}\right)}. \tag{7.91}$$

Posons maintenant le bruit en excès DC défini

$$\Delta S^{\text{dc}}(\tau, V_{\text{dc}}) = S^{\text{dc}}(\tau, V_{\text{dc}}) - S^{\text{dc}}(\tau, 0) \tag{7.92}$$

$$= S_{\text{eq}}(\tau) \left(\cos\left(\frac{e V_{\text{dc}} \tau}{\hbar}\right) - 1\right) \tag{7.93}$$

et utilisons l'identité trigonométrique $\sin^2(\theta) = \frac{1-\cos(2\theta)}{2}$ pour le réexprimer

$$\boxed{\Delta S^{\text{dc}}(\tau, V_{\text{dc}}) = -2\, S_{\text{eq}}(\tau) \sin^2\left(\frac{e V_{\text{dc}} \tau}{2\hbar}\right)}. \tag{7.94}$$

À τ fixe, le bruit en excès DC correspond donc à une autocorrélation qui oscille selon V_{dc}. Autrement dit, la corrélation au temps τ est contrôlée par V_{dc}.

Bien que le signe devant $S_{\text{eq}}(\tau)$ puisse être inattendu de prime abord, la propriété $S_{\text{eq}}(\tau) \leq 0 \ \forall\, \tau$ force le bruit en excès DC à être positif[11], soit

$$\Delta S^{\text{dc}}(\tau, V_{\text{dc}}) \geq 0 \quad \forall \tau. \tag{7.95}$$

11. Voir la section 7.3.2 pour les propriétés de $S_{\text{eq}}(\tau)$.

7.5.3 Oscillations de Pauli–Heisenberg

À l'instar du bruit en excès DC présenté à la section 7.5.2 il est intéressant d'isoler la contribution de la température au bruit total d'une jonction tunnel soumise à un biais en tension continue, et ce, dans le but de respecter les critères de la section 7.5.1 et d'ainsi avoir une mesure temporelle probante. On appelle l'autocorrélation résultante dans le domaine temporel le *bruit en excès thermique* ou *les oscillations de Pauli–Heisenberg* [52, 53].

Définissons le bruit en excès thermique dans le domaine temporel en ajoutant explicitement la dépendance en température dans (7.91) et en soustrayant les expressions à température finie et à température nulle, soit,

$$\Delta S^{\mathrm{PH}}(\tau, V_{\mathrm{dc}}, T_{\mathrm{e}}) = S^{\mathrm{dc}}(\tau, V_{\mathrm{dc}}, T_{\mathrm{e}}) - S^{\mathrm{dc}}(\tau, V_{\mathrm{dc}}, 0) \tag{7.96}$$

$$= \left(S_{\mathrm{eq}}(\tau, T_{\mathrm{e}}) - S_{\mathrm{eq}}(\tau, 0)\right) \cos\left(\frac{eV_{\mathrm{dc}}\tau}{\hbar}\right). \tag{7.97}$$

Posons ensuite $\Delta S^{\mathrm{PH}}_{\mathrm{eq}}(\tau, T_{\mathrm{e}}) = \Delta S^{\mathrm{PH}}(\tau, 0, T_{\mathrm{e}}) = \left(S_{\mathrm{eq}}(\tau, T_{\mathrm{e}}) - S_{\mathrm{eq}}(\tau, 0)\right)$ pour obtenir [12]

$$\boxed{\Delta S^{\mathrm{PH}}(\tau, V_{\mathrm{dc}}, T_{\mathrm{e}}) = \Delta S^{\mathrm{PH}}_{\mathrm{eq}}(\tau, T_{\mathrm{e}}) \cos\left(\frac{eV_{\mathrm{dc}}\tau}{\hbar}\right)}. \tag{7.98}$$

Cette équation a la forme d'une oscillation harmonique ayant comme enveloppe une fonction décroissante d'origine thermique ; ce sont les oscillations de Pauli-Heisenberg [52, 53].

Le nom *Pauli–Heisenberg* provient de l'interprétation de la partie oscillante comme la conséquence combinée du principe d'exclusion de Pauli [80] — qui empêche deux électrons de sauter à travers la barrière tunnel vers le même niveau d'énergie en même temps — et le principe d'incertitude d'Heisenberg [90, 91] — qui décrit l'incertitude minimale sur les niveaux d'énergie étant donnée l'incertitude temporelle des sauts.

12. On ne présente pas la forme de $\Delta S^{\mathrm{PH}}_{\mathrm{eq}}(\tau, T_{\mathrm{e}})$ ici, car elle est inélégante et peu pertinente à la discussion, mais elle est simple à obtenir en soustrayant les équations (7.57) et (7.62).

L'enveloppe $\Delta S_{\mathrm{eq}}^{\mathrm{PH}}(\tau, T_{\mathrm{e}})$ a un comportement assez intéressant. En effet, bien que $S_{\mathrm{eq}}(\tau)$ diverge en $\tau = 0$, voir l'équation (7.60), l'expansion de Taylor de (7.98) autour de $\tau = 0$ vaut, via (7.57) et (7.62),

$$\Delta S_{\mathrm{eq}}^{\mathrm{PH}}(\tau \approx 0\,, T_{\mathrm{e}}) \approx \frac{\pi (k_{\mathrm{B}} T_{\mathrm{e}})^2}{3R\hbar} - \frac{\pi^3 (k_{\mathrm{B}} T_{\mathrm{e}})^4}{15R\hbar^3}\tau^2 + \mathcal{O}\left(\tau^4\right). \tag{7.99}$$

Ainsi, l'enveloppe a une ordonnée à l'origine en T_{e}^2 qui est d'autant plus grande que la température est élevée ; conséquence de sa définition comme contribution thermique au bruit. Cependant, bien qu'elle décroisse toujours en $1/\tau^2$, sa décroissance est accélérée en T_{e}^4 par la température ; conséquence du *jitter* [13] thermique accru. On remarque aussi que la conséquence de (7.61) est

$$\lim_{\tau \to \infty} \Delta S^{\mathrm{PH}}(\tau, T_{\mathrm{e}}) = 0\,, \tag{7.100}$$

c'est-à-dire qu'à long temps, le *jitter* thermique vient masquer la partie oscillante et les deux signaux soustraits sont donc équivalents.

Un compromis est donc nécessaire pour bien mesurer cet effet : il faut une température assez faible pour que l'enveloppe ne décroisse pas trop rapidement dans le temps, mais assez élevée pour que l'autocorrélation soit tout de même aisément mesurable. Par chance, les températures les plus froides atteignables expérimentalement avec les cryostats à dilution, soit de l'ordre de $\sim 10\,\mathrm{mK}$, se prêtent bien à cette mesure — voir les résultats de la référence [52], avec le cryostat à $8\,\mathrm{mK}$ et $T_{\mathrm{e}} = 30\,\mathrm{mK}$.

7.6 Spectre de bruit résolu en phase

7.6.1 Principe

L'approche préconisée aux sections 7.2.1 et 7.2.3 pour transformer l'autocorrélation à deux temps en une autocorrélation ne dépendant que du décalage τ est d'effectuer un moyennage temporel. Cela revient en quelque sorte à prétendre

13. Mot anglais sans traduction élégante dans le contexte ; équivalent à *gigue* ou *remuage*.

que le signal est stationnaire au sens large, peu importe s'il l'est ou non, et permet de définir la densité spectrale mesurée indépendamment des propriétés du signal. C'est un choix judicieux lorsqu'une référence de temps absolue n'est pas disponible.

Cependant, dans certaines conditions expérimentales, il est possible de définir une telle référence temporelle. C'est notamment le cas si l'expérience comprend au moins une périodicité, ce qui est à la base du principe de la cyclostationnarité [92, §1 ; 93 ; 94]. On s'intéresse ici plus spécifiquement à la situation où l'expérience comprend une excitation périodique et où la mesure est effectuée de manière synchrone avec celle-ci, c'est-à-dire verrouillée en phase. Dans cette situation, chaque *point* mesuré expérimentalement dans le domaine temporel peut-être associé à une *phase* de l'excitation variant de 0 à 2π rad, ou de 0° à 360°, à une phase globale près selon les considérations expérimentales. Il n'y a toujours pas de temps absolu de référence, mais plutôt une équivalence entre un temps quelconque et tous les autres temps qui sont séparés de celui-ci d'un nombre entier de périodes d'excitation. Ainsi, chaque temps peut être associé à une phase de référence bien précise, périodiquement.

À phase fixe, l'autocorrélation ne dépend alors que d'un temps — le décalage par rapport à la phase choisie — et on peut en prendre la transformée de Fourier pour obtenir une densité spectrale ; c'est ce qu'on appellera le *spectre de bruit résolu en phase*.

7.6.2 Corrélateurs résolus en phase

On s'intéresse ici à obtenir les expressions générales pour le corrélateur courant-courant résolu en phase, qu'on définit $S_\phi(\tau)$, et pour la densité spectrale de puissance résolue en phase associée à celui-ci, qu'on dénote $\tilde{S}_\phi(\omega)$, elle aussi exprimée en termes de corrélateurs de courants. On considère une référence de phase périodique [14] de fréquence Ω qui permet de faire l'équivalence temps–phase $\phi = \Omega t$.

14. Typiquement une photoexcitation, mais seule l'existence de la référence est requise.

Pour obtenir les corrélateurs courant–courant résolus en phase, on prend comme point de départ le corrélateur bande large général de (7.43) et la modélisation de (7.19) pour le signal mesuré pendant un temps T. On réexprime le corrélateur en fonction de la période $P = 2\pi/\Omega$ du signal de référence en posant $T = MP$, où M est le nombre de périodes P sur lesquelles le signal est intégré. On a alors

$$i_M(t) \equiv \begin{cases} i(t) & \text{si} \quad -\frac{MP}{2} \leq t \leq \frac{MP}{2} \\ 0 & \text{sinon} \end{cases}, \tag{7.101}$$

d'où on obtient

$$\hat{S}(\tau) = \lim_{M\to\infty} \frac{1}{MP} \int_{-\frac{MP}{2}}^{\frac{MP}{2}} \langle i_M(t+\tau)\, i_M(t)\rangle \, \mathrm{d}t. \tag{7.102}$$

On peut alors simplement décomposer l'intégrale sur les périodes, c'est-à-dire

$$\hat{S}(\tau) = \lim_{M\to\infty} \frac{1}{MP} \sum_{m=-\frac{M}{2}}^{\frac{M}{2}} \int_{-\frac{P}{2}}^{\frac{P}{2}} \langle i_M(t+mP+\tau)\, i_M(t+mP)\rangle \, \mathrm{d}t, \tag{7.103}$$

et utiliser l'équivalence temps–phase $\phi = \Omega t$ pour poser

$$t_{\phi,m} = t + mP \tag{7.104}$$

$$= \frac{\phi}{\Omega} + mP, \tag{7.105}$$

soit l'ensemble des temps associés à la phase ϕ, indexés par la période m.

En substituant dans l'intégrale on trouve

$$\hat{S}(\tau) = \lim_{M\to\infty} \frac{1}{MP} \sum_{m=-\frac{M}{2}}^{\frac{M}{2}} \int_{-\pi}^{\pi} \left\langle i_M\left(t_{\phi,m}+\tau\right)\, i_M\left(t_{\phi,m}\right)\right\rangle \frac{\mathrm{d}\phi}{\Omega} \tag{7.106}$$

$$= \lim_{M\to\infty} \frac{1}{M} \sum_{m=-\frac{M}{2}}^{\frac{M}{2}} \int_{-\pi}^{\pi} \left\langle i_M\left(t_{\phi,m}+\tau\right)\, i_M\left(t_{\phi,m}\right)\right\rangle \frac{\mathrm{d}\phi}{2\pi}, \tag{7.107}$$

où on a utilisé $P\Omega = 2\pi$, qui n'est autre que la conséquence de la périodicité de la phase, et $\mathrm{d}t = \mathrm{d}\phi/\Omega$. Il est ainsi possible d'exprimer le résultat sous la forme

$$\hat{S}(\tau) = \int_{-\pi}^{\pi} \left(\lim_{M\to\infty} \frac{1}{M} \sum_{m=-\frac{M}{2}}^{\frac{M}{2}} \left\langle i_M\left(t_{\phi,m}+\tau\right)\, i_M\left(t_{\phi,m}\right)\right\rangle \right) \frac{\mathrm{d}\phi}{2\pi}. \tag{7.108}$$

On remarque qu'il est possible — dans la situation verrouillée en phase — de simplement mesurer l'intégrande plutôt que l'intégrale en tant que telle. La définition du corrélateur courant–courant résolu en phase et mesuré à la phase ϕ s'impose alors d'elle-même, soit

$$\boxed{S_\phi(\tau) = \lim_{M\to\infty} \frac{1}{M} \sum_{m=-\frac{M}{2}}^{\frac{M}{2}} \left\langle i_M\left(t_{\phi,m}+\tau\right)\, i_M\left(t_{\phi,m}\right)\right\rangle}, \tag{7.109}$$

une forme semblable à l'équation (7.43). Notons que, bien qu'il soit ici moyenné sur les périodes, $S_\phi(\tau)$ n'est pas complètement intégré sur le temps associé à la phase [15] ; c'est fondamentalement un corrélateur à deux temps dont l'un des deux est périodique. Bien entendu, on retrouve le corrélateur général intégré sur une des dépendances temporelles en intégrant sur ϕ,

$$\hat{S}(\tau) = \int_{-\pi}^{\pi} S_\phi(\tau) \frac{\mathrm{d}\phi}{2\pi}. \tag{7.110}$$

On remarque aussi que (7.109) n'est qu'une reformulation utile de

$$S_\phi(\tau) = \langle i(\phi/\Omega+\tau)\, i(\phi/\Omega)\rangle \tag{7.111}$$

avec la moyenne sur les périodes faite de manière explicite.

On peut alors obtenir le spectre résolu en phase par simple transformée de Fourier, à l'instar de la densité spectrale en puissance régulière [67, 68],

15. D'où l'omission du $\hat{\cdot}$ sur $S_\phi(\tau)$.

c'est-à-dire

$$\tilde{S}_\phi(\omega) = \mathcal{F}\left[S_\phi(\tau)\right](\omega) \tag{7.112}$$

$$= \int_{-\infty}^{\infty} S_\phi(\tau)\, \mathrm{e}^{-\mathrm{i}\omega\tau}\, \mathrm{d}\tau. \tag{7.113}$$

Or, tel que discuté ci-haut, $S_\phi(\tau)$ est périodique en ϕ par hypothèse. Puisque (7.113) est valide pour chaque ϕ indépendamment, il va de soi que $\tilde{S}_\phi(\omega)$ sera aussi une fonction périodique en ϕ.

Conséquemment, on peut exprimer $\tilde{S}_\phi(\omega)$ comme la série de Fourier en ϕ

$$\tilde{S}_\phi(\omega) = \sum_{n=-\infty}^{\infty} \tilde{\beta}_n(\omega)\, \mathrm{e}^{\mathrm{i}n\phi}, \tag{7.114}$$

avec les coefficients de Fourier

$$\tilde{\beta}_n(\omega) = \int_{-\pi}^{\pi} \tilde{S}_\phi(\omega)\, \mathrm{e}^{-\mathrm{i}n\phi} \frac{\mathrm{d}\phi}{2\pi}. \tag{7.115}$$

Via (7.113) et (7.109), on explicite

$$\tilde{\beta}_n(\omega) = \int_{-\pi}^{\pi} \left(\lim_{M\to\infty} \frac{1}{M} \sum_{m=-\frac{M}{2}}^{\frac{M}{2}} \int_{-\infty}^{\infty} \left\langle i_M\left(t_{\phi,m}+\tau\right)\, i_M\left(t_{\phi,m}\right)\right\rangle \mathrm{e}^{-\mathrm{i}\omega\tau}\, \mathrm{d}\tau \right) \mathrm{e}^{-\mathrm{i}n\phi} \frac{\mathrm{d}\phi}{2\pi}. \tag{7.116}$$

Exprimons maintenant $i_M\left(t_{\phi,m}+\tau\right)$ selon sa transformée de Fourier, soit

$$i_M\left(t_{\phi,m}+\tau\right) = \left(\int_{-\infty}^{\infty} \tilde{\imath}_M\left(\omega'\right)\, \mathrm{e}^{\mathrm{i}\omega'\left(t_{\phi,m}+\tau\right)} \frac{\mathrm{d}\omega'}{2\pi} \right) \tag{7.117}$$

$$= \left(\int_{-\infty}^{\infty} \tilde{\imath}_M\left(\omega'\right)\, \mathrm{e}^{\mathrm{i}\omega'\tau} \frac{\mathrm{d}\omega'}{2\pi} \right) \mathrm{e}^{\mathrm{i}\omega' t_{\phi,m}}. \tag{7.118}$$

En se concentrant sur les termes en τ de (7.116), on trouve alors

$$\int_{-\infty}^{\infty} \left\langle i_M\left(t_{\phi,m}+\tau\right)\, i_M\left(t_{\phi,m}\right)\right\rangle \mathrm{e}^{-\mathrm{i}\omega\tau}\, \mathrm{d}\tau$$

$$= \int_{-\infty}^{\infty} \left\langle \tilde{\imath}_M(\omega')\, i_M\left(t_{\phi,m}\right)\right\rangle \underbrace{\int_{-\infty}^{\infty} \mathrm{e}^{\mathrm{i}(\omega'-\omega)\tau}\,\mathrm{d}\tau}_{2\pi\delta(\omega'-\omega)} \mathrm{e}^{\mathrm{i}\omega' t_{\phi,m}} \frac{\mathrm{d}\omega'}{2\pi} \tag{7.119}$$

$$= \left\langle \tilde{\imath}_M(\omega)\, i_M\left(t_{\phi,m}\right)\right\rangle \mathrm{e}^{\mathrm{i}\omega t_{\phi,m}} \tag{7.120}$$

$$= \left\langle \tilde{\imath}_M(\omega)\, i_M\left(t_{\phi,m}\right)\right\rangle \mathrm{e}^{\mathrm{i}\omega\phi/\Omega}\,\mathrm{e}^{\mathrm{i}\omega mP}, \tag{7.121}$$

d'où

$$\tilde{\beta}_n(\omega) = \lim_{M\to\infty} \frac{1}{M} \sum_{m=-\frac{M}{2}}^{\frac{M}{2}} \int_{-\pi}^{\pi} \left\langle \tilde{\imath}_M(\omega)\, i_M\left(t_{\phi,m}\right)\right\rangle \mathrm{e}^{\mathrm{i}\omega\phi/\Omega}\,\mathrm{e}^{\mathrm{i}\omega mP}\,\mathrm{e}^{-\mathrm{i}n\phi} \frac{\mathrm{d}\phi}{2\pi}. \tag{7.122}$$

On peut combiner les exponentielles en utilisant (7.105) et en notant que $\Omega P = 2\pi$, soit

$$\mathrm{e}^{-\mathrm{i}n\phi}\,\mathrm{e}^{\mathrm{i}\omega\phi/\Omega}\,\mathrm{e}^{\mathrm{i}\omega mP} = \mathrm{e}^{-\mathrm{i}n\Omega\left(t_{\phi,m}-mP\right)}\,\mathrm{e}^{\mathrm{i}\omega(\phi/\Omega+mP)} \tag{7.123}$$

$$= \mathrm{e}^{-\mathrm{i}n\Omega t_{\phi,m}} \underbrace{\mathrm{e}^{\mathrm{i}nm\Omega P}}_{\mathrm{e}^{\mathrm{i}nm2\pi}=1\ \ \forall\ n,m\in\mathbb{Z}} \mathrm{e}^{\mathrm{i}\omega t_{\phi,m}} \tag{7.124}$$

$$= \mathrm{e}^{-\mathrm{i}(-\omega+n\Omega)t_{\phi,m}}. \tag{7.125}$$

En substituant dans (7.122) et en exprimant l'intégrale sur $t_{\phi,m}$, on trouve

$$\tilde{\beta}_n(\omega) = \lim_{M\to\infty} \frac{1}{M} \sum_{m=-\frac{M}{2}}^{\frac{M}{2}} \int_{-\frac{P}{2}+mP}^{\frac{P}{2}+mP} \left\langle \tilde{\imath}_M(\omega)\, i_M\left(t_{\phi,m}\right)\right\rangle \mathrm{e}^{-\mathrm{i}(-\omega+n\Omega)t_{\phi,m}} \frac{\mathrm{d}t_{\phi,m}}{P}, \tag{7.126}$$

qui n'est autre que la somme sur toutes les périodes de l'intégrale sur cesdites périodes. On peut donc combiner la somme et l'intégrale, sachant que $M \gg 1$, donc

$$\tilde{\beta}_n(\omega) = \lim_{M\to\infty} \frac{1}{MP} \int_{-\frac{MP}{2}}^{\frac{MP}{2}} \left\langle \tilde{\imath}_M(\omega)\, i_M\left(t_{\phi,m}\right)\right\rangle \mathrm{e}^{-\mathrm{i}(-\omega+n\Omega)t_{\phi,m}}\,\mathrm{d}t_{\phi,m}. \tag{7.127}$$

Comme $i_M\left(t_{\phi,m}\right) = 0$ si $\left|t_{\phi,m}\right| > MP/2$, on peut étendre l'intégrale à l'infini, et,

en regroupant les termes en $t_{\phi,m}$, on obtient

$$\tilde{\beta}_n(\omega) = \lim_{M\to\infty} \frac{1}{MP} \left\langle \underbrace{\int_{-\infty}^{\infty} i_M\left(t_{\phi,m}\right) \mathrm{e}^{-\mathrm{i}(-\omega+n\Omega)t_{\phi,m}} \,\mathrm{d}t_{\phi,m}}_{\mathcal{F}\left[i_M\left(t_{\phi,m}\right)\right](-\omega+n\Omega)} \tilde{\imath}_M(\omega) \right\rangle \tag{7.128}$$

$$= \lim_{M\to\infty} \frac{1}{MP} \left\langle \tilde{\imath}_M(-\omega+n\Omega)\, \tilde{\imath}_M(\omega) \right\rangle . \tag{7.129}$$

Finalement, en rappelant que $T = MP$, on obtient la forme finale des coefficients de Fourier

$$\boxed{\tilde{\beta}_n(\omega) = \lim_{T\to\infty} \frac{1}{T} \left\langle \tilde{\imath}_T(-\omega+n\Omega)\, \tilde{\imath}_T(\omega) \right\rangle} , \tag{7.130}$$

si bien que (7.114) prend une forme rappelant (7.28), c'est-à-dire

$$\boxed{\tilde{S}_\phi(\omega) = \lim_{T\to\infty} \frac{1}{T} \sum_{n=-\infty}^{\infty} \left\langle \tilde{\imath}_T(-\omega+n\Omega)\, \tilde{\imath}_T(\omega) \right\rangle \mathrm{e}^{\mathrm{i}n\phi}} . \tag{7.131}$$

Il est intéressant de remarquer que la seule dépendance en ϕ se retrouve dans l'exponentielle, de sorte qu'on peut poser

$$\tilde{S}_{\langle\phi\rangle}(\omega) \equiv \frac{1}{2\pi} \int_{-\pi}^{\pi} \tilde{S}_\phi(\omega)\, \mathrm{d}\phi \tag{7.132}$$

$$= \sum_{n=-\infty}^{\infty} \tilde{\beta}_n(\omega) \underbrace{\frac{1}{2\pi} \int_{-\pi}^{\pi} \mathrm{e}^{\mathrm{i}n\phi}(\omega)\, \mathrm{d}\phi}_{\mathrm{sinc}(n\pi)=\delta_{n,0}\ \forall\ n\in\mathbb{Z}} \tag{7.133}$$

$$= \tilde{\beta}_0(\omega) \tag{7.134}$$

$$= \lim_{T\to\infty} \frac{1}{T} \left\langle \tilde{\imath}_T(-\omega)\, \tilde{\imath}_T(\omega) \right\rangle \tag{7.135}$$

où on a utilisé la définition du sinus cardinal normalisé comme pour (7.75)

[86, §4]. Or, (7.135) n'est autre que (7.28), de manière à ce que

$$\tilde{S}_{\langle\phi\rangle}(\omega) = \hat{\tilde{S}}(\omega), \tag{7.136}$$

sans grande surprise.

De plus, il va sans dire que, expérimentalement, la somme sur n dans (7.114) et (7.131) sera tronquée de manière à ce qu'il y ait autant de valeurs de n que de phases résolues ; le résultat théorique correspondant à une résolution infinie. Conceptuellement, ϕ et n sont des variables conjugées par la série de Fourier au même titre que des variables — typiquement τ et ω — qui seraient conjuguées par la transformée de Fourier. Ainsi, n est en essence la *fréquence en phase*, ou l'harmonique, ce qui transparaît à l'identité

$$\tilde{\beta}_0(\omega) = \hat{\tilde{S}}(\omega) \tag{7.137}$$

découlant de (7.134) Le terme $n = 0$ de la série de Fourier, l'équivalent en phase de la fréquence nulle — le terme constant — correspond donc à la moyenne sur la phase de $\tilde{S}_\phi(\omega)$.

7.6.3 Spectre photoexcité résolu en phase

On se concentre ici sur le cas d'une jonction tunnel photoexcitée avec l'excitation de période Ω décrite en (7.68) et mesurée de manière synchrone. Considérant que $S^{\text{pa}}(t_1, t_2)$ à deux temps est l'autocorrélation générale dans la situation photoexcitée, les équations (7.66) et (7.70) nous permettent d'écrire

$$S^{\text{pa}}(t, t+\tau) = S_{\text{eq}}(\tau)\cos\Big(z\sin(\Omega t + \Omega\tau) - z\sin(\Omega t) + \nu\tau\Big), \tag{7.138}$$

où on rappelle que $z = \frac{eV_{\text{ac}}}{\hbar\Omega}$ et $\nu = \frac{eV_{\text{dc}}}{\hbar}$. Tel que discuté plus haut, on peut utiliser l'excitation comme référence en posant

$$\phi = \Omega t \qquad \text{avec} \qquad \phi \% 2\pi \Leftrightarrow \phi, \tag{7.139}$$

où % dénote l'opération *modulo* définie dans [95, §1.2.4], pour ensuite utiliser la notation $S_\phi^{\mathrm{pa}}(\tau) \equiv S^{\mathrm{pa}}(\phi/\Omega, \phi/\Omega + \tau)$ afin d'obtenir l'*autocorrélation résolue en phase*

$$S_\phi^{\mathrm{pa}}(\tau) = S_{\mathrm{eq}}(\tau) \cos\Big(z \sin(\phi + \Omega\tau) - z \sin(\phi) + \nu\tau\Big). \tag{7.140}$$

Bien que S_ϕ^{pa} ne dépende que de τ, l'indice ϕ traduit la seconde dépendance temporelle, qui est périodique pour les raisons discutées ci-haut. Comme on s'intéresse *in fine* au spectre résolu en phase, on définit

$$\tilde{S}_\phi^{\mathrm{pa}}(\omega) = \mathcal{F}\Big[S_\phi^{\mathrm{pa}}(\tau)\Big](\omega). \tag{7.141}$$

Puisque la transformée de Fourier du cosinus d'une somme de sinus semble embêtante à calculer, on pose

$$F_\phi(\tau) = S_{\mathrm{eq}}(\tau)\, \mathrm{e}^{\mathrm{i}(z \sin(\phi+\Omega\tau) - z \sin(\phi) + \nu\tau)} \tag{7.142}$$

tel que

$$S_\phi^{\mathrm{pa}}(\tau) = \Re\left[F_\phi(\tau)\right] = \frac{F_\phi(\tau) + F_\phi^*(\tau)}{2}, \tag{7.143}$$

et on utilise l'expansion de Jacobi–Anger, voir (7.71), pour simplifier à

$$F_\phi(\tau) = S_{\mathrm{eq}}(\tau)\, \mathrm{e}^{-\mathrm{i}z \sin\phi} \sum_{n=-\infty}^{\infty} \mathrm{e}^{\mathrm{i}n\phi} J_n(z)\, \mathrm{e}^{\mathrm{i}(n\Omega+\nu)\tau}. \tag{7.144}$$

Le spectre de bruit résolu en phase vaut alors, de (7.141) et (7.143),

$$\tilde{S}_\phi^{\mathrm{pa}}(\omega) = \frac{1}{2} \int_{-\infty}^{\infty} \Big(F_\phi(\tau) + F_\phi^*(\tau)\Big)\, \mathrm{e}^{-\mathrm{i}\omega\tau}\, \mathrm{d}\tau \tag{7.145}$$

$$= \frac{1}{2}\Big(\mathcal{F}\big[F_\phi(\tau)\big] + \mathcal{F}\big[F_\phi^*(\tau)\big]\Big). \tag{7.146}$$

Calculons donc

$$\mathcal{F}\big[F_\phi(\tau)\big](\omega) \equiv \tilde{F}_\phi(\omega) \tag{7.147}$$

$$= \mathrm{e}^{-\mathrm{i}z\sin\phi} \sum_{n=-\infty}^{\infty} \mathrm{e}^{\mathrm{i}n\phi} J_n(z) \int_{-\infty}^{\infty} S_{\mathrm{eq}}(\tau)\, \mathrm{e}^{-\mathrm{i}(\omega - n\Omega - \nu)\tau}\, \mathrm{d}\tau \tag{7.148}$$

et identifions la parenthèse en exposant à une fréquence effective [16] pour obtenir

$$\tilde{F}_\phi(\omega) = \mathrm{e}^{-\mathrm{i}z\sin\phi} \sum_{n=-\infty}^{\infty} \mathrm{e}^{\mathrm{i}n\phi} J_n(z)\ \tilde{S}_{\mathrm{eq}}(\omega - n\Omega - \nu) \tag{7.149}$$

$$= \mathrm{e}^{-\mathrm{i}z\sin\phi} \sum_{n=-\infty}^{\infty} \mathrm{e}^{\mathrm{i}n\phi} J_n(z)\ \tilde{S}_{\mathrm{eq}}(-\omega + n\Omega + \nu)\,, \tag{7.150}$$

où on a utilisé la parité de $\tilde{S}_{\mathrm{eq}}(\omega)$, voir (7.49). Calculons maintenant, de manière semblable,

$$\mathcal{F}\left[F_\phi^*(\tau)\right] = \mathrm{e}^{\mathrm{i}z\sin\phi} \sum_{n=-\infty}^{\infty} \mathrm{e}^{-\mathrm{i}n\phi} J_n(z) \int_{-\infty}^{\infty} S_{\mathrm{eq}}(\tau)\, \mathrm{e}^{-\mathrm{i}(\omega + n\Omega + \nu)\tau}\, \mathrm{d}\tau \tag{7.151}$$

$$= \mathrm{e}^{\mathrm{i}z\sin\phi} \sum_{n=-\infty}^{\infty} \mathrm{e}^{-\mathrm{i}n\phi} J_n(z)\ \tilde{S}_{\mathrm{eq}}(\omega + n\Omega + \nu) \tag{7.152}$$

$$= \left[\mathrm{e}^{-\mathrm{i}z\sin\phi} \sum_{n=-\infty}^{\infty} \mathrm{e}^{\mathrm{i}n\phi} J_n(z)\ \tilde{S}_{\mathrm{eq}}(\omega + n\Omega + \nu)\right]^* \tag{7.153}$$

$$= \tilde{F}_\phi^*(-\omega)\,, \tag{7.154}$$

si bien que (7.146) prend la forme

$$\tilde{S}_\phi^{\mathrm{pa}}(\omega) = \frac{1}{2}\left[\tilde{F}_\phi(\omega) + \tilde{F}_\phi^*(-\omega)\right]. \tag{7.155}$$

Finalement, via (7.150) et (7.154), on trouve l'expression finale représentant le spectre de bruit résolu en phase attendue pour une jonction tunnel photoexcitée,

16. Ce qui est la version courte de la démarche utilisée pour passer de l'équation (7.81) à l'équation (7.84).

soit

$$\tilde{S}_{\phi}^{\mathrm{pa}}(\omega) = \frac{1}{2}\Bigg\{ \mathrm{e}^{-\mathrm{i}z\sin\phi} \sum_{n=-\infty}^{\infty} \mathrm{e}^{+\mathrm{i}n\phi} J_n(z)\ \tilde{S}_{\mathrm{eq}}(-\omega + n\Omega + \nu) + \mathrm{e}^{+\mathrm{i}z\sin\phi} \sum_{n=-\infty}^{\infty} \mathrm{e}^{-\mathrm{i}n\phi} J_n(z)\ \tilde{S}_{\mathrm{eq}}(+\omega + n\Omega + \nu) \Bigg\} . \quad (7.156)$$

Propriétés de symétrie

Par substitutions dans (7.156), on remarque certaines propriétés de symétrie de $\tilde{S}_{\phi}^{\mathrm{pa}}(\omega)$. En particulier, $\tilde{S}_{\phi}^{\mathrm{pa}}(\omega)$ est hermitien avec une restriction supplémentaire sur la phase

$$\tilde{S}_{\phi}^{\mathrm{pa}\,*}(\omega) = \tilde{S}_{\phi}^{\mathrm{pa}}(-\omega) = \tilde{S}_{-\phi}^{\mathrm{pa}}(\omega) , \quad (7.157)$$

mais n'est pas symétrique

$$\tilde{S}_{\phi}^{\mathrm{pa}}(\omega) \neq \tilde{S}_{\phi}^{\mathrm{pa}}(-\omega) , \quad (7.158)$$

ce qui signifie que $S_{\phi}^{\mathrm{pa}}(\tau) \neq S_{\phi}^{\mathrm{pa}}(-\tau)$. En effet, on peut utiliser (7.140), la parité du cosinus et l'équivalence $\phi \Leftrightarrow \phi\%2\pi$ pour montrer que

$$S_{\phi}^{\mathrm{pa}}(-\tau) = S_{(\phi-\Omega\tau)\%2\pi}^{\mathrm{pa}}(\tau) , \quad (7.159)$$

ce qui est tout à fait logique considérant qu'un décalage de la phase correspond à un décalage temporel via Ω. C'est donc dire que, en autant d'acquérir l'ensemble des phases, toute l'information de l'autocorrélation résolue en phase se retrouve dans les délais positifs. Ce résultat sera crucial à l'optimisation du traitement numérique lors de la mesure.

Enfin, la parité de $\tilde{S}_{\mathrm{eq}}(\omega)$ et la propriété [85, Éq. (17.1.5)] des fonctions de Bessel du premier type $J_{-n}(z) = (-1)^n J_n(z)$ permet de démontrer

$$\tilde{S}_{\phi}^{\mathrm{pa}}(\omega, -\nu) = \tilde{S}_{\phi+\pi}^{\mathrm{pa}}(\omega, \nu) , \quad (7.160)$$

ce qui s'explique par le fait que changer le signe de ν change de π la phase relative entre les biais AC et DC alors que la phase globale n'est pas pertinente.

Limite à grande tension continue

Dans la limite à grande polarisation en tension continue — soit $|\nu| \gg |\mathcal{E}|/\hbar$, avec $\mathcal{E}$ représentant toutes les autres échelles d'énergie — on peut utiliser le résultat (7.54) pour obtenir

$$\tilde{S}_{\phi}^{\mathrm{pa}}(\hbar|\nu| \gg |\mathcal{E}|) = \frac{\hbar}{2R}\Bigg\{ \mathrm{e}^{-\mathrm{i}z\sin\phi} \sum_{n=-\infty}^{\infty} \mathrm{e}^{+\mathrm{i}n\phi} J_n(z) \,|-\omega + n\Omega + \nu| + \mathrm{e}^{+\mathrm{i}z\sin\phi} \sum_{n=-\infty}^{\infty} \mathrm{e}^{-\mathrm{i}n\phi} J_n(z) \,|+\omega + n\Omega + \nu| \Bigg\}. \tag{7.161}$$

Si on prend $\nu > 0$ et que l'on remarque que les seules valeurs de n pour lesquelles $n\Omega \gtrsim \nu$ sont celles où les $J_n(z)$ sont négligeables — comparativement aux contributions à la somme de n plus près de zéro — on peut alors simplement remplacer la valeur absolue par des parenthèses. On note maintenant que

$$\frac{1}{\mathrm{i}}\frac{\mathrm{d}}{\mathrm{d}\phi} \sum_{n=-\infty}^{\infty} \mathrm{e}^{+\mathrm{i}n\phi} J_n(z) = \sum_{n=-\infty}^{\infty} n\,\mathrm{e}^{+\mathrm{i}n\phi} J_n(z) \tag{7.162}$$

et on utilise l'expansion de Jacobi–Anger, soit l'équation (7.71), pour obtenir

$$\frac{1}{\mathrm{i}}\frac{\mathrm{d}}{\mathrm{d}\phi} \sum_{n=-\infty}^{\infty} \mathrm{e}^{+\mathrm{i}n\phi} J_n(z) = \frac{1}{\mathrm{i}}\frac{\mathrm{d}}{\mathrm{d}\phi}\,\mathrm{e}^{\mathrm{i}z\sin\phi} \tag{7.163}$$

$$= z\cos\phi\,\mathrm{e}^{\mathrm{i}z\sin\phi}. \tag{7.164}$$

Les équations (7.162) et (7.164), ainsi que leurs conjugés complexes, impliquent donc

$$\sum_{n=-\infty}^{\infty} n J_n(z)\,\mathrm{e}^{\pm\mathrm{i}n\phi} = z\cos\phi\,\mathrm{e}^{\pm\mathrm{i}z\sin\phi}. \tag{7.165}$$

On utilise alors (7.71) et (7.165) pour simplifier (7.161) tel que

$$\tilde{S}^{\rm pa}_{\phi}\left(\hbar\nu \gg |\mathcal{E}|\right) = \frac{\hbar}{2R}\Big\{ \quad \mathrm{e}^{-\mathrm{i}z\sin\phi}\left(-\omega + z\Omega\cos\phi + \nu\right)\mathrm{e}^{+\mathrm{i}z\sin\phi} \\ + \mathrm{e}^{-\mathrm{i}z\sin\phi}\left(+\omega + z\Omega\cos\phi + \nu\right)\mathrm{e}^{-\mathrm{i}z\sin\phi}\Big\}, \tag{7.166}$$

et donc

$$\tilde{S}^{\rm pa}_{\phi}\left(\hbar\nu \gg |\mathcal{E}|\right) = \frac{\hbar}{R}\left(\nu + z\Omega\cos\phi\right). \tag{7.167}$$

Ce résultat est beaucoup plus frappant lorsqu'exprimé en termes des tensions $V_{\rm dc} = \hbar\nu/e$ et $V_{\rm ac} = \hbar z\Omega/e$ appliquées à la jonction, soit

$$\boxed{\tilde{S}^{\rm pa}_{\phi}\left(eV_{\rm dc} \gg |\mathcal{E}|\right) = \frac{e}{R}\left(V_{\rm dc} + V_{\rm ac}\cos\phi\right)}. \tag{7.168}$$

Notons que dans le cas $\nu < 0$, on aurait

$$\tilde{S}^{\rm pa}_{\phi}\left(-eV_{\rm dc} \gg |\mathcal{E}|\right) = \frac{e}{R}\left(-V_{\rm dc} - V_{\rm ac}\cos\phi\right) \tag{7.169}$$

$$= \frac{e}{R}\left(-V_{\rm dc} + V_{\rm ac}\cos\left(\phi+\pi\right)\right), \tag{7.170}$$

en accord avec (7.160).

Il est aussi intéressant de remarquer que ces résultats se reformulent, via (7.68) et $\phi = \Omega t$, sous les formes

$$\tilde{S}^{\rm pa}_{\phi}\left(eV_{\rm dc} \gg |\mathcal{E}|\right) = +\frac{e}{R}V(t) \qquad \text{si} \quad V_{\rm dc} > 0 \tag{7.171}$$

$$\tilde{S}^{\rm pa}_{\phi}\left(-eV_{\rm dc} \gg |\mathcal{E}|\right) = -\frac{e}{R}V(t) \qquad \text{si} \quad V_{\rm dc} < 0, \tag{7.172}$$

si bien que

$$\boxed{\tilde{S}^{\rm pa}_{\phi}\left(e\,|V_{\rm dc}| \gg |\mathcal{E}|\right) = \frac{e}{R}\,|V(t)|}. \tag{7.173}$$

7.6.4 Limite photoexcitée habituelle

Pour confirmer que notre approche est cohérente, on peut prendre la moyenne sur les phases de $\tilde{S}^{\mathrm{pa}}_{\phi}(\omega)$. Si tous les temps de références sont bel et bien équivalents modulo $2\pi/\Omega$, la moyenne sur les phases sera équivalente à celle sur le temps et on obtiendra alors le même résultat que pour le bruit photoassisté de l'équation (7.87), soit $\hat{\tilde{S}}^{\mathrm{pa}}(\omega)$. Posons donc,

$$\tilde{S}^{\mathrm{pa}}_{\langle\phi\rangle}(\omega) = \frac{1}{2\pi}\int_{-\pi}^{\pi} \tilde{S}^{\mathrm{pa}}_{\phi}(\omega)\,\mathrm{d}\phi, \tag{7.174}$$

et intéressons-nous d'abord seulement aux termes en ϕ, c'est-à-dire

$$\frac{1}{2\pi}\left(\int_{-\pi}^{\pi} \mathrm{e}^{-\mathrm{i}z\sin\phi}\,\mathrm{e}^{\mathrm{i}n\phi}\,\mathrm{d}\phi + \int_{-\pi}^{\pi} \mathrm{e}^{\mathrm{i}z\sin\phi}\,\mathrm{e}^{-\mathrm{i}n\phi}\,\mathrm{d}\phi\right) \tag{7.175}$$

$$= \frac{1}{2\pi}\left(\sum_{m=-\infty}^{\infty} J_m(z)\int_{-\pi}^{\pi} \mathrm{e}^{\mathrm{i}(n-m)\phi}\,\mathrm{d}\phi + \sum_{m=-\infty}^{\infty} J_m(z)\int_{-\pi}^{\pi} \mathrm{e}^{\mathrm{i}(m-n)\phi}\,\mathrm{d}\phi\right), \tag{7.176}$$

où on a encore utilisé l'expansion (7.71) de Jacobi–Anger. En posant $\kappa \in \mathbb{Z}$, les sommes contiendront des termes de la forme

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} \mathrm{e}^{\mathrm{i}\kappa\phi}\,\mathrm{d}\phi = \frac{1}{2\pi}\frac{\mathrm{e}^{\mathrm{i}\kappa\phi}}{\mathrm{i}\kappa}\bigg|_{-\pi}^{\pi} \tag{7.177}$$

$$= \frac{\mathrm{e}^{\mathrm{i}\kappa\pi} - \mathrm{e}^{-\mathrm{i}\kappa\pi}}{2\mathrm{i}\kappa\pi} \tag{7.178}$$

$$= \mathrm{sinc}\,(\kappa\pi) \tag{7.179}$$

$$= \delta_{\kappa,0}, \tag{7.180}$$

ce résultat étant strictement équivalent à (7.75). Ainsi, puisque $\delta_{a,b} = \delta_{b,a}$,

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} \mathrm{e}^{\mathrm{i}(n-m)\phi}\,\mathrm{d}\phi = \frac{1}{2\pi}\int_{-\pi}^{\pi} \mathrm{e}^{\mathrm{i}(m-n)\phi}\,\mathrm{d}\phi = \delta_{n,m}. \tag{7.181}$$

En propageant ces deltas dans (7.176) puis dans (7.174), en passant par (7.156), on trouve

$$\tilde{S}^{\mathrm{pa}}_{\langle\phi\rangle}(\omega) = \frac{1}{2}\Bigg\{ \sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty} J_n(z)\,J_m(z)\,\delta_{n,m}\,\tilde{S}_{\mathrm{eq}}(-\omega + n\Omega + \nu) + \sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty} J_n(z)\,J_m(z)\,\delta_{n,m}\,\tilde{S}_{\mathrm{eq}}(+\omega + n\Omega + \nu) \Bigg\}. \tag{7.182}$$

On peut ensuite utiliser les $\delta_{n,m}$ pour réduire des sommes, exploiter le fait que les indices des sommes sont muets, se prévaloir de la propriété [85, Éq. (17.1.5)] des fonctions de Bessel du premier type $J^2_{-n}(z) = J^2_n(z)$ et profiter de la parité de $\tilde{S}_{\mathrm{eq}}(\omega)$ pour obtenir

$$\tilde{S}^{\mathrm{pa}}_{\langle\phi\rangle}(\omega) = \sum_{n=-\infty}^{\infty} J^2_n(z)\left\{\frac{\tilde{S}_{\mathrm{eq}}\Big(\nu - (\omega + n\Omega)\Big) + \tilde{S}_{\mathrm{eq}}\Big(\nu + (\omega + n\Omega)\Big)}{2}\right\} \tag{7.183}$$

$$= \sum_{n=-\infty}^{\infty} J^2_n(z)\,\tilde{S}^{\mathrm{dc}}(\omega + n\Omega\,,\,\nu), \tag{7.184}$$

et finalement

$$\boxed{\tilde{S}^{\mathrm{pa}}_{\langle\phi\rangle}(\omega) = \hat{\tilde{S}}^{\mathrm{pa}}(\omega)} \tag{7.185}$$

tel qu'attendu. L'approche résolue en phase est donc cohérente avec la mesure habituelle du bruit photoexcité.

Expérimentalement, on fera une moyenne d'ensemble pour chaque phase, avec un échantillon par période par délai τ. La moyenne sur les phases convertira la moyenne d'ensemble vers la moyenne temporelle habituelle ; mesurer $\tilde{S}^{\mathrm{pa}}_{\phi}(\omega)$ pour toutes les phases accessibles donne donc aussi accès à $\hat{\tilde{S}}^{\mathrm{pa}}(\omega)$.

Chapitre 8

Aspects Expérimentaux

On cherche à mesurer le signal de bruit aux bornes d'une jonction tunnel dans différentes conditions expérimentales de polarisation en tension continue et d'excitation sinusoïdale dans le but d'en extraire — numériquement et à la volée — les propriétés d'intérêt pendant l'acquisistion. La section 8.1 décrit le schéma du montage expérimental utilisé pour ces mesures ainsi que son principe de fonctionnement. La section 8.3 présente la carte d'acquisition qui permet de faire des mesures large bande ultrarapides. Les spécificités liées aux mesures résolues en phases sont discutées à la section 8.4, alors que la méthode de verrouillage de phase implémentée pour stabiliser la phase relative entre le taux d'acquisition et la fréquence de photoexcitation lors ces mesures est traitée à la section 8.5. Le traitement des données à la volée, un aspect de la mesure très important pour la validité des résultats, est quant à lui présenté au chapitre 9.

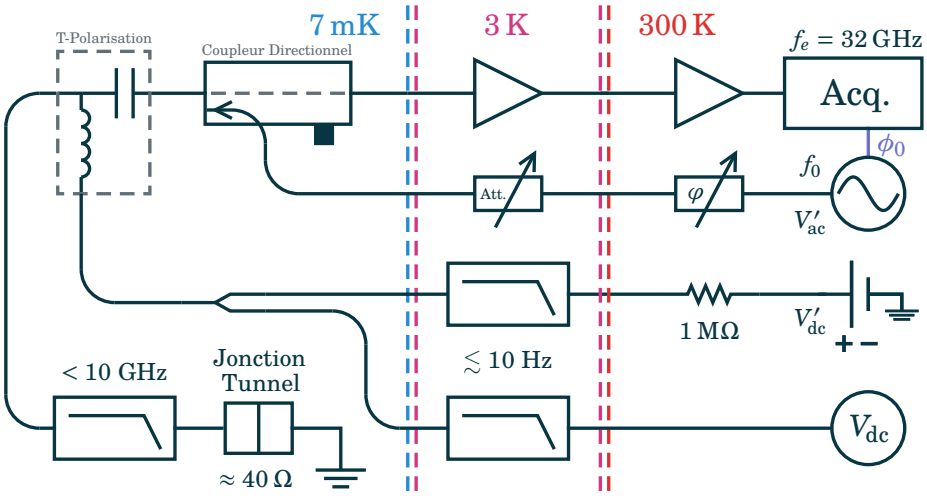


FIGURE 8.1 – Montage expérimental pour les mesures ultrarapides. Un montage simple permet de polariser et de photoexciter une jonction tunnel avant d'en mesurer le bruit numériquement à haute vitesse. Le traitement numérique de la trace temporelle permet une grande flexibilité en ce qui a trait aux quantités mesurées.

8.1 Montage expérimental

8.1.1 Schéma

La figure 8.1 présente le schéma du montage expérimental utilisé pour l'acquisition des données.

Une jonction tunnel de résistance $R \approx 40.38\,\Omega$ est placée sur l'étage froid d'un réfrigérateur à dilution de manière à la maintenir à une température $\lesssim 7$ mK. Elle est polarisée en courant continu à travers la branche inductive d'un *T de polarisation* [1] (Anritsu K250) à l'aide d'une source (Yokogawa GS200) de tension V'_{dc} en série avec une résistance. Un multimètre digital (Keysight/Agilent 34410A) permet de mesurer la tension V_{dc} aux bornes de la jonction [2]. La photoexcitation est quant à elle appliquée à travers un coupleur directionnel (Krytar

1. Communément appelé *Bias-Tee* en anglais.
2. Cette situation de *mesure à trois fils* est cependant sensible aux résistances parasites entre la jonction tunnel et la mise à la terre ainsi qu'entre le point où les lignes des lignes basses fréquence se rejoignent et la jonction — voir la discussion de la section 10.1.4.

102040016K) et la branche capacitive du *T de polarisation* par une source de tension alternative (Rohde&Schwarz SGS100A et SGU100A) de fréquence f_0 et d'amplitude V'_{ac}. Un atténuateur variable (Agilent J7211C) et une ligne à délai micro-onde (Colby PDL-100A) — respectivement dénotés par « Att. » et φ sur le schéma — permettent d'étendre la plage de puissance de la source et d'ajuster la phase de l'excitation.

Du côté de la détection, le bruit aux bornes de la jonction traverse la branche capacitive du *T de polarisation* pour être acheminé vers la sortie du cryostat — à $\approx 300\,\text{K}$ — par une chaîne d'amplification (LNF-LNC1_12A sn021B et MITEQ AFS4-00101200-18-10P-4), avant d'être numérisé à la fréquence d'échantillonnage $f_e = 32\,\text{GHz}$ par une carte d'acquisition ultrarapide Guzik ADP-7104 ayant une bande passante analogique de 10 GHz et une résolution de 10 bit. Il est possible de synchroniser l'échantillonnage et la photoexcitation en partageant une référence d'horloge entre les appareils, ils ont alors la même référence de phase ϕ_0.

Pour s'assurer que la tension appliquée aux bornes de la jonction soit stable, et pour éliminer le bruit électronique pouvant provenir des appareils de mesure, on filtre les lignes basses fréquences avec des filtres résistifs passe-bas de fréquence de coupure $\lesssim 10\,\text{Hz}$. Un filtre passe-bas de fréquence de coupure 10 GHz est aussi placé devant la jonction tunnel pour l'isoler de l'environnement aux fréquences hors de la bande passante de la mesure.

Ce montage relativement élémentaire met à profit la carte d'acquisition ultrarapide et la puissance de calcul offerte par la station de travail qui y est connectée pour offrir une grande flexibilité expérimentale malgré sa simplicité. En effet, puisqu'on a ici accès directement à la trace expérimentale — en large bande et avec une résolution temporelle importante — toute opération mathématique imaginable peut en théorie y être appliquée. Il est aussi possible d'effectuer plusieurs traitements de données différents sur une seule et même trace — indépendamment les uns des autres ou bien de manière subséquentes — même si ceux-ci seraient normalement incompatibles ou très fastidieux à implémenter à l'aide d'un montage micro-onde standard. De plus, le poste de travail utilisé pour contrôler les appareils et recevoir les données prodigue une grande puissance de

calcul qui permet de traiter l'important flot de données généré au moment même de l'acquisition, sans devoir enregistrer les traces brutes sur disque. La section 8.3 décrit plus techniquement la carte d'acquisition et le système de mesure et de traitement des données.

8.2 Jonction Tunnel

Comme décrit à la section 8.1.1, l'échantillon étudié est une jonction tunnel. Or, les jonctions tunnels typiquement utilisées dans le cadre de mesures en bande étroite ne sont pas nécessairement adaptées à la situation en bande large.

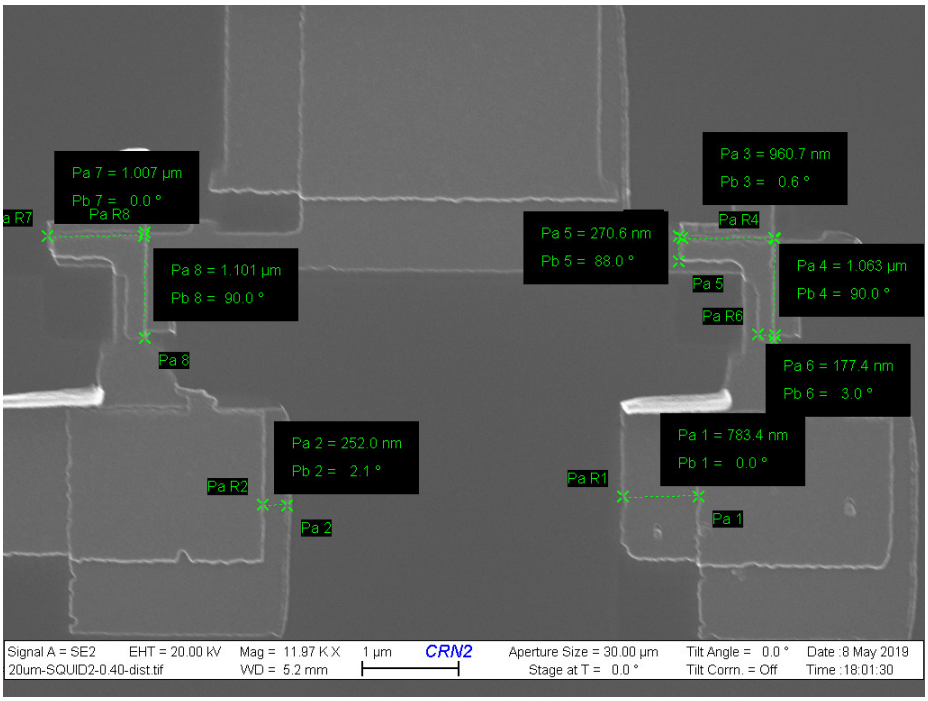


FIGURE 8.2 – Image par microscopie électronique d'une jonction similaire à celle utilisée. L'image est une gracieuté de Sébastien Jezouin et Charles Paradis.

C'est pourquoi l'échantillon sélectionné pour les mesures en bande large est en réalité une jonction Josephson [16] conçue pour être utilisée en tant qu'amplificateur à large bande [32]. Un aimant permanent placé à proximité de l'échantillon permet d'y détruire la supraconductivité de sorte que l'échantillon

soit à toutes fins pratiques une jonction tunnel. On profite toutefois toujours du circuit micro-ondes large bande au sein duquel la jonction est intégrée ainsi que du porte échantillon large bande spécifiquement conçu à cet effet.

Une image par microscope électronique d'un échantillon nominalement identique à celui utilisé est disponible à la figure 8.2. L'échantillon est une gracieuseté de Sébastien Jezouin, il a été fabriqué dans le cadre de la référence [32].

8.3 Système d'acquisition ultrarapide

Les deux éléments essentiels à l'acquisition de données ultrarapides sont une carte d'acquisition ultrarapide et un ordinateur assez puissant pour traiter les données dans un temps raisonnable. En effet, plus le taux d'échantillonnage est élevé, moins il est réaliste d'enregistrer les traces expérimentales sur disques mécaniques — même sur disque à semi-conducteurs [3] — que ce soit dû au volume des données requises pour une expérience ou à l'importance du flux de données généré. Il importe donc de pouvoir traiter les données directement en mémoire vive — à la volée — au cours de l'acquisition et de n'enregistrer que l'information importante, distillée des détails superflus de chaque réalisation. De plus, peu importe la rapidité de l'échantillonnage, un traitement de données en mémoire trop long vient diminuer, voire annuler, l'avantage des mesures ultrarapides ; le traitement des données devient rapidement le facteur limitant la vitesse des mesures.

La carte d'acquisition ultrarapide utilisée au sein du montage, une Guzik ADP-7104 dans un châssis Keysight M9505A, est présentée à la figure 8.3. Elle possède quatre canaux et permet d'en échantillonner un ou deux à 32 GSa/s et jusqu'à quatre à 16 GSa/s ; 128 GB de mémoire vive à même la carte emmagasine les échantillons pour transfert éventuel vers le poste de travail. La résolution est configurable à 8 ou 10 bit, les données étant respectivement présentées dans des entiers non signés `uint8` ou dans des entiers signés `int16` dont les 6 bits les moins significatifs sont nuls.

3. Communément appelé *SSD*, de l'anglais *solid state drive*.

FIGURE 8.3 – Guzik ADP7104 dans châssis Keysight M9505A. Les quatre ports SMA identifiés par des fonds colorés correspondent aux canaux d'acquisitions. Les ports de synchronisation « 1 GHz » et « Sync Clk » sont visibles en haut à gauche du panneau de la carte. La communication avec le poste de travail est assurée par le câble double branché au panneau inférieur.

Une option d'égalisation permet d'utiliser le FPGA [4] interne à la carte d'acquisition pour appliquer un filtre numérique aux données dans le but de compenser les non-linéarités et inhomogéités des convertisseurs analogique–numérique, d'éliminer le signal en dehors de leur bande passante nominale de 10 GHz et d'appliquer un filtre d'antirepliement [5]. Lorsque ce filtre est activé avec des données 10 bit, les bits les moins significatifs des données sont mis à profit pour minimiser les erreurs de rastérisation ; elles sont donc présentées avec 16 bit de résolutions bien qu'elles proviennent de mesures 10 bit. La carte est reliée au poste de travail, à travers le châssis, par un bus PCIe x8 de seconde génération qui sert à la fois au contrôle de la carte et au transfert des données. Le taux de transfert empirique entre la carte d'acquisition et la mémoire vive du système est d'environ 1.5 GB/s, ce qui correspond à 1.5 et 0.75 GSa/s pour une résolution

4. De l'anglais *Field Programmable Gate Array*, un type de circuit logique programmable.
5. Plus connu sous le terme anglais de filtre d'*antialiasing*.

de 8 ou 10 bit, respectivement.

Dans le but de la synchroniser avec les différents appareils micro-ondes du montage, il est possible de fournir une référence d'horloge de 50, 100 ou 200 MHz à la carte à l'aide de son port « Sync Clk », ou encore de lui fournir une référence de 1 GHz au port étiqueté « 1 GHz » — voir la section 8.4 pour plus de détails.

FIGURE 8.4 – Topologie du poste de travail utilisé pour le traitement des données à la volée. Obtenue via l'utilitaire `hwloc-info` [96].

Quant à lui, le poste de travail utilisé pour le traitement de données est un HP Z840 disposant de 128 GB de mémoire vive et contenant deux processeurs centraux Intel Xeon E5-2697 v4 avec 45 MB de cache chacun et supportant l'*hyper-threading*. On a donc accès à 36 coeurs physiques, 72 coeurs logiques, pour une puissance de calcul parallèle substantielle.

Notons que *Windows 10 Pro for Workstation* est le système d'exploitation utilisé et que, bien de ce dernier supporte jusqu'à 256 coeurs logiques, ceux-ci sont répartis en groupes d'au plus 64 unités [97]. Le poste de travail est donc configuré avec deux groupes, soit un groupe de 36 coeurs associé à chacun des processeurs ; la figure 8.4 en présente la topologie telle que déterminée par `hwloc` [96]. Lors de la rédaction de code `C++` parallèle visant la performance optimale [6], on utilise la fonction maison `manage_thread_affinity`, disponible à

6. Développé via l'interface de programmation `OpenMP` [98].

l'annexe B.1.3, pour optimiser l'utilisation des processeurs logiques. Autrement, le code n'utilise que le premier groupe de processeurs et n'a donc accès qu'à la moitié de la puissance de calcul.

Bien que le poste de travail contienne aussi une carte graphique puissante permettant d'accélérer les opérations parallèles à virgule flottante — une NVIDIA Quadro P5000 — les processeurs centraux offrent une puissance de calcul suffisante pour les besoins des expériences présentées ici, tout en étant beaucoup plus simples à programmer. On n'utilise donc pas la carte graphique dans ce qui suit.

8.4 Mesures résolues en phase

Lors des mesures résolues en phase on synchronise — ou verrouille — la source micro-onde et la carte d'acquisition en prenant le signal de synchronisation $f_{\text{sync}} = 1\,\text{GHz}$ [7] de la source et en le fournissant au port de référence 1 GHz de la carte d'acquisition. L'excitation et l'acquisition sont alors synchronisées et, pourvu que la fréquence d'excitation soit un multiple entier de f_{sync}, le premier échantillon de chaque acquisition de données correspondra à la même phase ϕ_0 de l'excitation — celle-ci pouvant être considérée comme une phase globale.

Notons que, si la fréquence d'excitation f_0 n'est pas un multiple entier de f_{sync}, la différence de phase entre chaque échantillon restera constante, mais le premier échantillon de chaque acquisition pourra prendre différentes valeurs. Par exemple, pour $f_0 = 4.5\,\text{GHz}$ et $f_e = 32\,\text{GHz}$, les échantillons seront séparés d'un temps correspondant à une phase de $2\pi f_0/f_e = \frac{9}{32}\pi\,\text{rad}$ de l'excitation. Cependant, comme l'acquisition est synchronisée sur $f_{\text{sync}} = 1\,\text{GHz}$, la phase du premier échantillon de chaque acquisition pourra être de ϕ_0 ou $\phi_0 + \pi\,\text{rad}$, selon le moment précis auquel l'acquisition est lancée. Comme ce temps n'est pas contrôlé précisément par le programme d'acquisition, il n'est pas possible de prévoir la phase initiale de chaque acquisition de donnée.

7. D'autres fréquences de synchronisation sont aussi disponibles ; on utilise 1 GHz parce qu'elle devrait minimiser la dérive de phase entre la source et la carte d'acquisition.

Ce qui précède vaut pour toute fréquence demi-entière de 1 GHz. Pour une fréquence quart-entière, il y aurait quatre phases possibles — et ainsi de suite, tendant selon toute vraisemblance vers une phase complètement aléatoire si la fréquence s'approche d'une valeur irrationnelle. Par souci de simplicité, on se restreint donc à la situation où $f_0/f_{\text{sync}} \in \mathbb{N}$ et $f_e/f_0 \in \mathbb{N}$. Typiquement, on utilise $f_0 = 4\,\text{GHz}$ et $f_e = 32\,\text{GHz}$ de manière à ce que les échantillons soient séparés de $\frac{\pi}{4}$ rad, que les $f_e/f_0 = 8$ phases résolues soient balayées en une période $1/f_0$ et que la phase de départ de chaque acquisition soit toujours ϕ_0.

Le verrouillage de phase détaillé ci-haut est excellent pour une échelle de temps courte ; lors de l'acquisition d'une série de données par exemple. Cependant, lors de longues mesures pouvant s'étaler sur plusieurs jours, des facteurs externes peuvent faire dériver la phase. Par exemple, un léger déplacement de câble, la contraction thermique des câbles et des composantes ou encore une dérive lente entre les horloges des appareils pourraient, entre autres, participer à la dérive de la référence de phase. Pour éviter que cet effet vienne brouiller les mesures longues, il faut périodiquement réajuster la phase relative entre l'excitation et l'acquisition à l'aide du déphaseur. La méthode de stabilisation de la phase développée dans le cadre de ce travail est expliquée en détail à la section 8.5.

8.5 Stabilisation de la phase

8.5.1 Partie déterministe du signal

Lors des mesures de bruit photoexcité, une partie du signal de photoexcitation est typiquement mêlée au signal d'intérêt à la sortie du cryostat. En effet, comme la jonction tunnel n'a pas une impédance parfaitement adaptée au circuit micro-onde 50 Ω — et comme plusieurs désaccords d'impédance et réflexions mineures peuvent avoir lieu au sein du montage expérimental — une partie de l'excitation y est réfléchie et se mêle aux fluctuations. Ce signal résiduel venant ainsi polluer l'information recherchée, il est typiquement éliminé des mesures en choisissant une fréquence d'excitation hors de la bande de mesure [46, 61, 64, 65, 99] ou

en l'éliminant des fluctuations par conversion vers le bas [61, 63] à la même fréquence.

Néanmoins, dans le cas de mesures synchrones et commensurables avec l'excitation — tel que décrit à la section 8.4 — le signal d'excitation résiduel s'avère en pratique plutôt utile. Dans cette situation, l'excitation résiduelle a une contribution à toutes fins pratiques identique à chacune de ses périodes, contrairement au bruit qui est aléatoire. On appelle cette contribution la *partie détermiste* du signal, puisqu'il ne s'agit pas de fluctuations ; elle est conceptuellement prédictible.

L'extraction de cette *partie déterministe* du signal est détaillée à la section 9.3. On se concentre ici sur son application à l'extraction et la stabilisation de la phase de la mesure.

8.5.2 Extraction de la phase

La partie déterministe n'étant autre que le signal résiduel de la photoexcitation, elle devrait avoir la même forme que celle-ci. Puisque qu'on photoexcite la jonction tunnel avec un signal monofréquence, on s'attend donc à obtenir un cosinus [8] avec un déphasage ϕ_0 dépendant des détails expérimentaux et ajustable via le déphaseur. Il est dès lors possible d'associer ce cosinus à une phase de plusieurs manières, que ce soit par lissage, transformée de Fourier, algorithme de Goertzel [100] ou autre heuristique.

Par souci de simplicité, on choisit de simplement prendre la tranformée de Fourier discrète de la partie déterministe et de regarder la phase de la composante de Fourier associée à la fréquence f_0 de photoexcitation. En pratique, pour tout signal périodique en $1/f_0$, cela correspondra à l'index 1 — le second élément — de la transformée de Fourier discrète.

Soit donc l'élément de vecteur $\bar{x}_i$ avec $0 \leq i < f_e/f_0$ et $i \in \mathbb{Z}$, la partie

8. On choisit le cosinus pour que $\phi_0 = 0$ corresponde à un ventre de l'excitation.

déterministe du signal d'intérêt. On obtient la phase ϕ_0 associée à celle-ci via

$$\phi_0 = \arctan\left(\frac{\Re\left[\tilde{\bar{x}}_1\right]}{\Im\left[\tilde{\bar{x}}_1\right]}\right) \tag{8.1}$$

avec $\tilde{\bar{x}}_j = F\left[\bar{x}_i\right]_j$ et où F dénote la transformée de Fourier discrète.

Il est important de remarquer que la valeur de cette phase en elle-même n'est pas importante ; elle est valide à une phase globale près. Par exemple, si un câble était remplacé par un autre câble de longueur différente, la valeur de la phase ainsi obtenue changerait, même si la synchronisation entre la source micro-onde et l'acquisition était parfaite et constante, mais cela n'affecterait pas autrement les résultats. La phase de la partie déterministe permet donc de définir la référence de phase requise pour effectuer des mesures résolues en phase comme discuté à la section 7.6.

8.5.3 Ajustement de la phase

Tel que discuté aux sections 8.4 et 8.5.2, des facteurs externes incontrôlables [9] peuvent venir changer la référence de phase au cours du temps même pour une situation expérimentale nominalement constante. Lors de mesures moyennées sur de longues périodes — jusqu'à plusieurs jours — il importe donc de ne pas moyenner ensemble des résultats correspondant à des phases différentes afin de ne pas compromettre la probité des résultats.

Pour ce faire, on s'assure que la phase ϕ_0 de chaque mesure corresponde à la phase visée avant de poursuivre le traitement des données. Lors de chaque acquisition, on extrait la phase via la partie déterministe — voir la section 8.5.1 — et, si celle-ci s'éloigne de la valeur visée d'un écart plus grand que la tolérance demandée, on utilise la ligne à délai pour rectifier la situation. L'implémentation en `PyHegel` de cette procédure, utilisée lors des acquisitions, est disponible à l'annexe C.1 ; on en présente ici le principe.

9. Incluant sans s'y limiter : les variations de température, les perturbations de câbles et les dérives internes lentes des appareils de mesures.

Soit la phase de référence θ mesurée sur une série de données, la phase visée ϕ_0 et la tolérance $\Delta\phi$ définie positivement. Si $|\theta - \phi_0| > \Delta\phi$, on juge que la phase n'est pas adéquate et on l'ajuste avec la ligne à délai. La conversion entre la différence de phase et le délai temporel d'ajustement est donnée par [10]

$$\Delta t = \frac{\theta - \phi_0}{360^\circ} \cdot \frac{1}{f_0} \tag{8.2}$$

et le delai est ajusté en conséquence. Puisque la précision de la ligne à délai n'est pas parfaite, on vérifie ensuite que la nouvelle phase est adéquate. Dans le cas contraire, on répète cette procédure — extraction et ajustement — jusqu'à l'obtention d'une phase acceptable. Une fois la phase acceptée, le protocole expérimental peut suivre son cours.

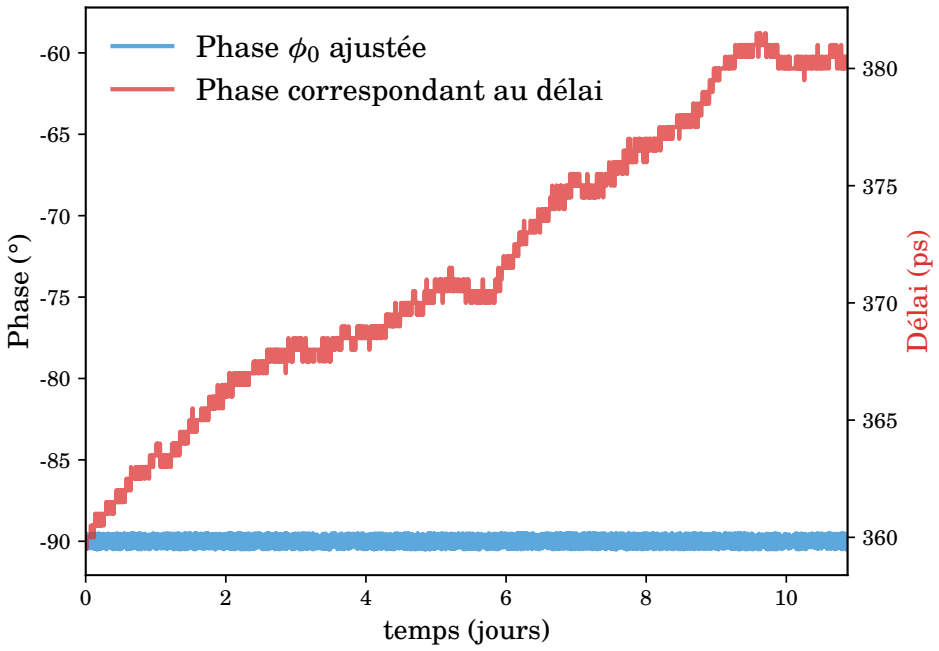


FIGURE 8.5 – Stabilisation de la phase sur plusieurs jours. La phase est stabilisée à $-90^\circ \pm 0.5^\circ$. La courbe bleue correspond à la phase mesurée après ajustement pour chaque mesure (axe de gauche). La courbe rouge correspond à la fois à la valeur de la ligne à délai (axe de droite) et à la phase qui aurait été obtenue sans ajustement (axe de gauche).

10. On travaille en degrés parce qu'il est plus simple d'identifier les valeurs numériques d'angles intéressants en degrés qu'en radian ; il est plus simple de viser 45° que ≈ 0.78539 rad.

La figure 8.5 montre la variation de la phase lors d'une longue acquisition de données. On y voit entre autres que l'ajustement de phase permet effectivement de maintenir ϕ_0 autour de la cible de $-90° \pm 0.5°$ pendant toute la durée de l'acquisition.

Bien qu'il soit impossible de mesurer la dérive de la phase tout en l'ajustant, on peut estimer quelle eut été la phase non-stabilisée en utilisant la valeur du délai après ajustement. En effet, comme discuté ci-haut, on peut voir ce délai de compensation comme un déphasage égal et opposé à la dérive de phase. Les deux axes verticaux de la figure sont donc ajustés pour que la courbe de *Phase correspondant au délai* soit valide à la fois en termes de phase et de délai temporel.

On voit clairement que, même si la phase est assez stable à court terme, celle-ci varie de près de 30° sur une période d'une dizaine de jours sans le réajustement constant, ce qui correspond à un délai d'une vingtaine de picosecondes.

Chapitre 9

Traitement de données à la volée

En donnant accès à la trace temporelle directement, la carte d'acquisition ultrarapide décrite à la section 8.3 permet une flexibilité expérimentale incomparable aux mesures analogiques. Toute opération mathématique imaginable peut effectivement être appliquée au signal d'entrée, avec comme seule restriction les limites de la numérisation à la carte d'acquisition.

Cependant, le prix à payer pour cette flexibilité est le volume astronomique de données à traiter pour une expérience. En effet, à 32 GSa/s, avec des échantillons dans des contenants 16 bit, une seule seconde de données correspond à 64 gigaoctets. Bien que le flux de données entrantes soit en réalité limité par le transfert entre la carte d'acquisition et l'ordinateur de contrôle, une seule expérience peut engendrer plusieurs centaines de téraoctets. Il n'est donc pas réaliste — ni d'un point de vue pratique, ni d'un point de vue économique — d'enregistrer les traces temporelles brutes telles quelles. En fait, relire des traces de cette taille sur disques mécaniques serait typiquement plus lent que l'acquisition elle-même.

La solution est donc de traiter les données en mémoire *à la volée*, et de n'enregistrer sur disque que le résultat de ce traitement préliminaire. On garde ainsi l'information pertinente contenue dans les données et on jette les détails propres aux réalisations individuelles, qui n'ont pas d'impact sur les résultats finaux. On utilise le terme *à la volée* pour décrire le traitement fait sur les données en mémoire au fil de l'expérience, mais il est important de spécifier que

ce traitement n'est pas nécessairement fait en temps réel dépendamment de sa complexité ou des informations demandées. Idéalement, le traitement à la volée n'aura pas d'impact notable sur la durée de l'expérience ; certaines stratégies et optimisations sont utilisées ici pour minimiser cet impact.

Trois grandes catégories de prétraitement ont été implémentées dans le cadre ce de projet et sont détaillées aux sections suivantes, soit le calcul de l'autocovariance à la section 9.1, celui de l'autocovariance résolue en phase à la section 9.2 et l'extraction et l'élimination de la partie déterministe du signal à la section 9.3. L'obtention d'histogrammes 1D et 2D a aussi été implémentée lors de l'exploration du montage et l'ajustement des paramètres de polarisation de la jonction. Cependant, puisque les histogrammes ne sont pas utilisés pour le traitement des données présentées ici, on présente plutôt ce travail à l'annexe B.1.

9.1 Autocovariance

9.1.1 Description générale

Les définitions mathématiques de la corrélation et des fonctions qui en dérivent sont présentées en détail aux sections 7.1.1 à 7.1.3 ; le reste du chapitre 7 décrivant les autres quantités d'intérêts pouvant en être déduites, notamment la densité spectrale de puissance. Cependant, ces formules théoriques — bien que fondamentales — ne sont pas bien adaptées au traitement à la volée de signaux numérisés. On présente ici l'approche et les algorithmes adoptés pour obtenir une mesure expérimentale aussi fiable que possible de l'autocovariance des signaux discrets mesurés expérimentalement.

On choisit l'autocovariance plutôt que l'autocorrélation ou la corrélation générale entre deux signaux puisqu'on s'intéresse aux corrélations au sein d'un seul signal, entre autres dans le but d'obtenir des densités spectrales, et qu'on étudie les fluctuations qui devraient normalement avoir une moyenne nulle. Cette approche à aussi l'avantage de donner le même résultat si les

données sont dans des contenants de type non signés [1] ou signés [2]. En pratique, l'utilisation de l'autocovariance plutôt que de l'autocorrélation pour l'obtention de densités spectrales devrait avoir un impact minime, soit un décalage par une constante à fréquence nulle ; fréquence qui se trouve d'ailleurs en dehors de la bande passante expérimentale. Pour ces raisons, et par abus de langage, l'implémentation utilise souvent le terme *autocorrélation* pour faire référence à l'*autocovariance* — cet abus de langage est d'ailleurs presque une tradition dans le domaine du traitement de signal [69, §1.1].

On décrit aussi les optimisations utilisées pour rendre le code aussi efficace que possible sans compromettre la précision des résultats — ce qui est primordial pour traiter les données à la volée sans ajouter de délais déraisonnables — en plus de présenter une comparaison de la performance des différents algorithmes implémentés.

9.1.2 Description des algorithmes

Soit, comme à la section B.1, un vecteur de données de taille N, représenté par l'élément de vecteur x_i et contenant les données expérimentales à traiter. Chaque valeur de l'index $i \in \mathbb{N}$ correspond à un échantillon mesuré, il agit donc comme le degré de liberté temporel au même titre que t dans le cas théorique continu. Ainsi, ils seront reliés par $t = i/f_e$, où f_e est la fréquence d'échantillonnage de la carte d'acquisition.

On cherche ici à calculer l'autocovariance telle que modélisée à l'équation (7.43), c'est-à-dire qu'on ne garde que le décalage temporel — représenté par le décalage d'index k — comme degré de liberté temporel ; on prétend que le signal est stationnaire au sens large. Cette modélisation correspond exactement à l'autocovariance dans le cas des signaux stationnaires au sens large, mais permet tout de même de définir la densité spectrale pour tout signal, sans égard à la stationnarité [3]. On profite aussi de la parité de l'autocovariance $\hat{S}(-\tau) = \hat{S}(\tau)$,

1. Sans valeurs négatives ; nécessairement de moyenne non nulle pour un signal non nul.
2. Pouvant tout de même être de moyenne non nulle à cause des imperfections expérimentales, bien que l'on mesure des fluctuations.
3. Voir le chapitre 7, spécifiquement la section 7.2.3, pour plus de détails.

voir (7.46), pour ne calculer que les délais positifs et ainsi dinimuer le temps de calcul.

En pratique, il n'est pas pertinent de mesurer toutes les valeurs possibles de k entre 0 et $N-1$; on se limite à un k maximal correspondant à la résolution voulu dans le domaine temporel [54, 55]. Typiquement, les valeurs de k plus grandes que $1/f_{\min}$, soit l'échelle de temps associée à la fréquence la plus faible contenue dans la bande passante de la mesure, sont ignorées. Elles apportent en fait très peu d'information pertinente et peuvent même venir brouiller les résultats en incluant des réflexions parasites du signal mesuré. Comme il est trivial d'éliminer des valeurs de k après la mesure, on préconise d'en mesurer un peu plus que le strict minimum, mais on se limitera tout de même à un nombre de valeurs de k très petit comparativement à N.

Une différence majeure entre le cas continu et celui discret est l'effet de bord à la fin du vecteur de données. En effet, les traces expérimentales sont de longueur N finie [4], ce qui fait que — pour un délai k donné — il y aura k échantillons manquants en fin de trace lors du calcul des termes $x_i x_{i+k}$; spécifiquement les termes $x_{N-j}x_{N-j+k}$ avec $1 \leq j \leq k$. La description qui suit tient compte de cette particularité.

Soit donc la version *autocovariance discrète* de (7.43), l'élément de vecteur

$$a_k = \frac{1}{N-k} \sum_{i=0}^{N-k-1} (x_i - \mu_{0,k})(x_{i+k} - \mu_{k,0}), \tag{9.1}$$

avec

$$\mu_{n,m} = \frac{1}{N-n-m} \sum_{i=n}^{N-m-1} x_i \tag{9.2}$$

$$\text{pour} \qquad n, m \in \mathbb{N} \qquad \text{et} \qquad 0 \leq m+n < N, \tag{9.3}$$

la moyenne partielle effectuée sur $N-n-m$ des N échantillons de la mesure. On utilise cette variété de la moyenne de manière à ce que chacun des termes soustraits dans l'une ou l'autre des parenthèses de (9.1) fasse intervernir le même

4. Par manque de temps… et de mémoire…

sous-ensemble d'échantillons. Autrement dit, puisque x_i dans (9.1) prendra les valeurs d'index 0 à $N-k-1$ lors de la somme, on lui soustrait $\mu_{0,k}$ qui est la moyenne sur ceux-ci [5]. Cela est plus visible si on explicite les $\mu_{n,m}$ utilisés dans (9.1), soit

$$\mu_{0,k} = \frac{1}{N-k} \sum_{i=0}^{N-k-1} x_i \tag{9.4}$$

et

$$\mu_{k,0} = \frac{1}{N-k} \sum_{i=k}^{N-1} x_i = \frac{1}{N-k} \sum_{i=0}^{N-k-1} x_{i+k}. \tag{9.5}$$

Cette approche permet d'éviter la seconde des *curieuses propriétés* décrites dans [101] pour l'estimation de l'autocovariance d'un processus statistique dont la moyenne est inconnue — propriété apparente lorsque $\mu_{0,k} < \mu_{0,0} < \mu_{k,0}$. En particulier, il serait alors possible que l'autocovariance estimée pour une fonction croissante monotone soit négative, contrairement à ce qui est attendu sachant que a_k correspond, à une normalisation près, au coefficient de corrélation linéaire r de Pearson [6] entre x_i et x_{i+k} [102]. Notons aussi que la somme de (9.1) est tronquée pour éviter les effets de bords discutés ci-haut.

Cependant, la motivation principale pour l'utilisation de cette définition de l'autocovariance est la possibilité de la réexprimer de manière élégante sous une forme similaire à la variance $\langle\langle X^2 \rangle\rangle = \langle X^2 \rangle - \langle X \rangle^2$, soit

$$a_k = \sum_{i=0}^{N-k-1} \frac{x_i x_{i+k}}{N-k} - \mu_{0,k}\,\mu_{k,0}. \tag{9.6}$$

Ce résultat se décompose alors très bien en termes appropriés au traitement numérique, c'est-à-dire

$$\boxed{a_k = \frac{r_k}{N-k} - \left(\frac{M-\beta_k}{N-k}\right) \cdot \left(\frac{M-\gamma_k}{N-k}\right)}, \tag{9.7}$$

5. De même pour x_{i+k}, balayant les valeurs d'index k à $N-1$, et de moyenne $\mu_{k,0}$.
6. Qui vaut ± 1 pour des données parfaitement linéaires, le signe étant celui de la pente.

avec

$$r_k = \sum_{i=0}^{N-k-1} x_i x_{i+k} \qquad \text{et} \qquad M = \sum_{i=0}^{N-1} x_i \,, \tag{9.8}$$

ainsi qu'avec

$$\beta_k = \sum_{i=N-k}^{N-1} x_i \qquad \text{et} \qquad \gamma_k = \sum_{i=0}^{k-1} x_i \,. \tag{9.9}$$

Ici, r_k et M sont essentiellement des versions non-normalisées de l'autocorrélation et de la moyenne, alors que β_k et γ_k sont des corrections de bords. Notons que cette formulation motive aussi le choix de $N - k$ pour la normalisation, bien que l'utilisation de N soit souvent préconisée par les statisticiens [103, §6.2].

Cette formulation est avantageuse pour le calcul numérique à plusieurs égards. Premièrement, puisque les x_i sont représentés en mémoire par des entiers, les quantités définies en (9.8) et (9.9) seront toujours elles-mêmes entières, ce qui permet d'accumuler les sommes sans craindre d'engendrer des erreurs d'arrondi numérique [104 ; 105, §4]. De plus, ces quantités sont aussi très facilement parallélisables, les sommes pouvant être scindées en plusieurs sous-sommes calculées parallèlement pour être recombinées à la fin du calcul. En plus, il est possible de calculer ces valeurs en traversant une seule fois les données de manière linéaire, et ce même pour plusieurs valeurs de k à la fois. Minimiser de la sorte la *dispersion* des accès à la mémoire est une bonne pratique pour optimiser la performance de calcul. De plus, il est possible d'accélérer grandement le calcul des r_k en tirant profit du théorème de Wiener–Khintchine, qui stipule que l'autocorrélation est la transformée de Fourier inverse de la densité spectrale de puissance — voir la section 7.2.3 pour plus de détails — pour réexprimer

$$r_k = F^{-1}\left[\left|F\left[x_i\right]_j\right|^2\right]_k \,, \tag{9.10}$$

avec F et F^{-1} les versions discrètes de la transformée de Fourier $\mathcal{F}$ et de sont opération inverse $\mathcal{F}^{-1}$; des opérations numériques si efficaces qu'elles sont

appelées *transformées de Fourier rapides* [7] [106–108]. Notons que la transformée de Fourier rapide, contrairement à sa version continue, donne naturellement lieu à des convolutions circulaires plutôt qu'à celles linéaires désirées ici. Pour obtenir des convolutions linéaires, et donc le bon résultat, il faut doubler la taille des données en leur ajoutant des valeurs nulles [8] avant de les utiliser dans (9.10) [109, §8.7.2]. Concrètement, il suffit de redéfinir x_i tel que $0 \leq i \leq M-1$ avec $x_{i \geq N} = 0$ et $M \geq 2N$ lors de l'implémentation de l'algorithme.

Bien que cette formulation soit adaptée au calcul numérique, une attention particulière doit être portée aux divisions de (9.7) ainsi qu'à la soustraction entre ses deux termes principaux ; des opérations qui doivent être effectuées par arithmétique en virgule flottante. En effet, r_k, M et N ont tous le *potentiel* d'être *énormes*, et les opérations à virgule flottante impliquant des nombres d'ordres de grandeur très différents — ou de très grands nombres tout court — ont le potentiel de causer une perte de précision importante [105, §2 et §4]. Lors de l'accumulation des r_k et de M, il faut aussi tenir en compte la possibilité d'un dépassement d'entier.

Les deux sous-sections qui suivent présentent l'interface du code et les détails plus techniques liés à l'optimisation de la performance et de la précision du calcul de (9.7).

9.1.3 Code et interface

Le code utilisé pour calculer l'autocovariance du signal est disponible à l'annexe B.2.2. Il est rédigé majoritairement en `C++`, avec une interface intermédiaire `Python` rédigée elle aussi en `C++` via la librairie `pybind11` [9] [110]. Plusieurs types de données d'entrée sont supportés via l'utilisation de *templates*, spécifiquement `uint8`, `int8`, `uint16` et `int16`. La performance du code provient principalement de la parallélisation réalisée grâce aux commandes de préprocesseur de l'interface de programmation `OpenMP` [98].

7. De l'anglais *Fast Fourier Transform* — plus connues sous le sigle `FFT`.
8. Opération typiquement appelée *zero-padding*.
9. Qui requiert l'utilisation de `C++11` ou plus récent ; on utilise spécifiquement `C++14`.

Deux algorithmes d'autocovariance sont implémentés [10], soit un algorithme implémentant directement l'équation (9.7) telle quelle et un algorithme optimisé calculant r_k à l'aide de transformées de Fourier rapides selon (9.10) — spécifiquement selon la version modifiée (9.17) discutée ci-bas. L'algorithme direct simple est plus rapide à bas nombre d'autocorrélations, alors que l'algorithme par transformées de Fourier — utilisant la librairie `FFTW3` [111, 112] — est beaucoup plus efficace à grand nombre d'autocorrélations. Notons que `FFTW3` fourni des fonctions profitant de la symétrie hermitienne [76, §4.1.2.1] de la densité spectrale d'un signal réel, voir (7.45), pour diviser par deux le nombre de coefficients de Fourier à calculer.

On porte une attention particulière pour éviter les erreurs numériques, si bien que les deux algorithmes donnent strictement le même résultat numérique dans tous les tests effectués. La librairie de nombre à virgule flottante à précision arbitraire `MPFR` [113, 114] est utilisée — à travers l'interface `MPFR C++` [115] — lors des opérations pouvant engendrer des erreurs d'arrondi, afin d'assurer la précision des résultats.

Une seconde interface en `Python` — voir `acorrs_otf.py` à l'annexe B.2.2 — facilite l'instantification de la bonne classe selon le type des données à traiter et le nombre d'autocorrélations désirées, elle est écrite directement en `Python` à fin de simplicité. C'est la fabrique [11] `ACorrUpTo` de cette interface qui est utilisée à travers `PyHegel` lors de l'acquisition de données.

Il est possible de calculer l'autocovariance sur une quantité virtuellement illimitée de données en fournissant itérativement plusieurs séries de données à un objet créé via `ACorrUpTo`. L'objet accumule alors à l'interne les quantités de (9.8) et (9.9) pour l'ensemble des données et la moyenne d'ensemble des a_k est accessible via son attribut `res`. Les séries de données sont considérées indépendantes et aucune corrélation n'est calculée entre la fin d'une série et le début de la série suivante. L'extrait de code qui suit en illustre le principe.

```
1 import numpy as np
2 from acorrs_otf import ACorrUpTo
3 # Instantification préalable. On peut aussi instantifier avec des données en
4 # second argument, e.g. "a = ACorrUpTo(100,x)", *x* sera alors accumulé.
```

10. En plus d'un algorithme pour l'autocovariance résolue en phase — voir la section 9.2.
11. Traduction libre de l'anglais *factory* ; fonction pour abstraire l'instantification de classes.

```
a = ACorrUpTo(100,'int16') # On veut les 100 première autocovariances
# Simulons 10 mesures 16bit signées
for n in range(10):
    # *x* simule une trace temporelle expérimentale
    x = np.random.randint(-2**15,2**15, size=2**20, dtype=np.int16)
    a(x)    # On accumule dans l'objet *a*
# a.res contient la moyenne d'ensemble de l'autocovariance des dix *x*
print a.res # Affiche les résultats; calcul final fait à l'accès de *a.res*
```

9.1.4 Considérations numériques et optimisations

Dépassement d'entiers

Pour éviter les dépassements d'entier lors de l'accumulation — de r_k, M_k, β_k, γ_k, et de N — on se concentre sur r_k. En effet, celui-ci devrait croître plus rapidement que les autres, étant constitué de la somme de produits $x_i\, x_{i+k}$. La méthode `compute_chunk_size()` du fichier `acorrs.tpp` permet de calculer le nombre d'accumulations critique nécessaire avant qu'un dépassement d'entier ait lieu dans le pire des cas, soit pour une série consitutée uniquement du nombre de plus grande valeur absolue possible selon le type de données d'entrée. Notons qu'il faut aussi tenir compte du type de donnée dans lequel l'accumulation est faite ; bien qu'on accumule toujours dans des types 64 bit, on accumule dans des types de même signe que les données pour des raisons de performance.

Par exemple, pour des données signées 16 bit, des `int16` $\in [-2^{15}, 2^{15}-1]$, on accumule les r_k dans des variables signées 64 bit, c'est-à-dire des `int64` $\in [-2^{63}, 2^{63}-1]$. La valeur de plus grande amplitude de $x_i\, x_{i+k}$ est alors $(-2^{15})^2 = 2^{30}$, ce qui entre $(2^{63}-1)/\!/2^{30} = 2^{33}-1 = 8\,589\,934\,591$ fois dans l'accumulateur avant un dépassement d'entier, avec $/\!/$ qui dénote la division entière.

Lorsque le nombre d'échantillons critique est atteint pendant une accumulation, les valeurs courantes des accumulateurs sont ajoutées à des accumulateurs de grande précision et les accumulateurs entiers sont remis à zéro. Les accumulateurs de grande précision sont des nombres à virgule flottante à précision arbitraire fournis par la librairie `MPFR C++` [115]. Ils ne sont pas utilisés directement comme accumulateurs puisqu'ils sont moins performants que les entiers

64 bit natifs. Sauf indication contraire, on choisit d'utiliser une précision de 48 décimales — correspondant à 160 bit — ce qui permet de représenter exactement les entiers positifs et négatifs d'amplitude allant jusqu'à $2^{160} \sim 10^{48}$. À 32 GSa/s et sans temps mort, cette limite est impossible à atteindre avant *quelques dizaines de milliards de fois l'âge de l'univers*. On choisit cette précision puisqu'elle permet d'effectuer le calcul final de l'équation (9.7) en minimisant les erreurs numériques lors d'opérations comprenant des très grands nombres ou des nombres d'ordres de grandeur très différents, et ce, sans affecter les performances de manière notable.

Transformées de Fourier de grande taille

Bien que la formulation (9.10) de r_k via transformées de Fourier offre la possibilité de performances enviables, elle présente aussi quelques défis.

D'abord, pour qu'elle soit exacte, il faut calculer $F[x_i]_j = \sum_{i=0}^{N-1} x_i \, \mathrm{e}^{\mathrm{i}2\pi ji/N}$, ce qui requiert l'ensemble des x_i, soit potentiellement plusieurs milliards d'échantillons. Pour des raisons techniques — présumément les détails de la cache du processeur et de l'accès à la mémoire — l'utilisation d'une transformée de Fourier d'une telle taille est inconvéniente et risque d'augmenter le temps de calcul comparativement à la méthode *naïve* par produits directs. Cependant, puisqu'on s'intéresse à un nombre de décalages k petit comparativement à N, on peut simplement *scinder* le signal d'entré en B blocs [12] de taille [13] $\beta > k$ appropriée à l'utilisation des transformées de Fourier.

Soit l'élément de vecteur $x_{i,b} = x_{b\beta+i}$, le i^{e} échantillon du b^{e} bloc avec $0 \leq i < \beta$, la contribution de ce bloc à r_k sera simplement

$$r'_{k,b} = F^{-1}\left[\left|F\left[x_{i,b}\right]_{j,b}\right|^2\right]_{k,b} \tag{9.11}$$

et on peut assembler les contributions de chaque bloc

$$r'_k = \sum_{b=0}^{B-1} r'_{k,b} \tag{9.12}$$

12. Si des données se retrouvent dans un bloc partiel en fin de trace, on les traite via (9.8).
13. On rappelle que les transformées de Fourier seront de taille 2β à cause du *zero-padding*.

$$= \sum_{b=0}^{B-1} F^{-1}\left[\left|F\left[x_{i,b}\right]_{j,b}\right|^2\right]_{k,b} \tag{9.13}$$

$$= F^{-1}\left[\sum_{b=0}^{B-1}\left|F\left[x_{i,b}\right]_{j,b}\right|^2\right]_k, \tag{9.14}$$

où on a utilisé la linéarité de la transformée de Fourier pour sauver $B-1$ transformées inverses.

Or, bien que $r'_k \approx r_k$ lorsque $\beta \gg k$, cette approche n'est pas exacte ; toutes les corrélations entre blocs adjacents sont perdues. Comme $k < \beta$, il ne devrait pas être trop délétère de simplement calculer les corrélations interblocs manquantes $\Delta r_{k,b}$ par produits directs, ce qui donne

$$\Delta r_{k,b} = \sum_{j=0}^{k-1} x_{\beta-1-j,b}\, x_{k-1-j,b+1} \tag{9.15}$$

$$\Delta r_k = \sum_{b=0}^{B-2} \Delta r_{k,b}, \tag{9.16}$$

puisqu'il y a une frontière de moins que le nombre de blocs. On trouve alors le $r_k = r'_k + \Delta r_k$ exact sous la forme

$$r_k = F^{-1}\left[\sum_{b=0}^{B-1}\left|F\left[x_{i,b}\right]_{j,b}\right|^2\right]_k + \sum_{b=0}^{B-2}\sum_{j=0}^{k-1} x_{\beta-1-j,b}\, x_{k-1-j,b+1}. \tag{9.17}$$

C'est ce qui est implémenté dans `acorrsFFT.tpp`.

Transformées de Fourier en `double` et erreurs d'arrondi

Cependant, même si cette formulation est exacte en théorie, elle est tout de même sensible aux erreurs d'arrondi numériques. En effet, les transformées de Fourier sont effectuées à l'aide de nombres à virgule flottante, spécifiquement des `double` de 53 bit de précision. Tel que discuté ci-haut, ces nombres peuvent donc représenter exactement les entiers d'amplitude jusqu'à 2^{53}. On calcule donc

le nombre d'accumulations possibles dans l'espace de Fourier avant que $|r'k| > 2^{53}$ à l'aide de la fonction `compute_accumul_max` [14], dans l'optique d'effectuer la transformée inverse et l'accumulation dans les nombres à virgule flottante haute précision avant ce nombre d'accumulations critique. Il faut d'ailleurs tenir compte du fait que la transformée inverse fournie par `FFTW` n'est pas normalisée, son résultat est donc multiplié par l'entier 2β. Pour une robustesse supplémentaire, et comme $2\beta x_i x_{i+k} \in \mathbb{Z}$ devrait toujours être respecté, on arrondit le résultat des r'_k vers l'entier le plus près avant l'accumulation finale à grande précision. Enfin, on multiplie ce nombre critique par un facteur de sécurité de 3/4 comme précaution supplémentaire.

Malgré toutes les précautions prises, cette heuristique n'est pas *mathématiquement garantie* à l'épreuve des erreurs numériques. Cela dit, aucune erreur numérique n'a été observée [15] lors des tests de performance de la figure 9.1 entre le a_k calculé via le r_k de (9.8) et celui de (9.17).

9.1.5 Performance

Il n'est pas simple de prévoir la performance du calcul de r_k via (9.17) comparativement à (9.8). En effet, l'accélération due aux *FFT* vient de pair avec une correction interbloc relativement complexe, demandant $\sim k^2$ opérations, mais qui doit être calculée de moins en moins souvent plus la taille β des blocs est grande. La performance des fonctions fournies par `FFTW3` dépend cependant beaucoup de la cache disponible sur le processeur et des blocs trop gros sont désavantageux. De plus, l'accumulation dans l'espace de Fourier doit être interrompue fréquemment pour ne pas engendrer trop d'erreurs d'arrondi.

Ainsi, plutôt que d'essayer de prédire la performance des algorithmes, on préfère effectuer des tests représentatifs de la situation expérimentale. La figure 9.1 présente les résultats d'une série de tests pendant lesquels l'autocovariance à été calculée par produits directs ou via transformées de Fourier — avec différentes

14. Autrement dit, le $b_{\max}$ dans $\sum_{b=0}^{b_{\max}} \left|F\left[x_{i,b}\right]_{j,b}\right|^2$ avant que sa transformée inverse puisse engendrer des erreurs d'arrondi notables.
15. Différence strictement nulle ; représentation binaire identique.

valeurs de β — pour une dizaine d'acquisitions de données expérimentales. On y voit que prendre des blocs d'une taille correspondant à un multiple entier de k est un bon indicateur de la performance pour l'algorithme par transformée de Fourier ; conséquence probable de l'équilibre entre l'utilisation de blocs de taille modeste [16] et la minimisation du nombre de corrections interblocs.

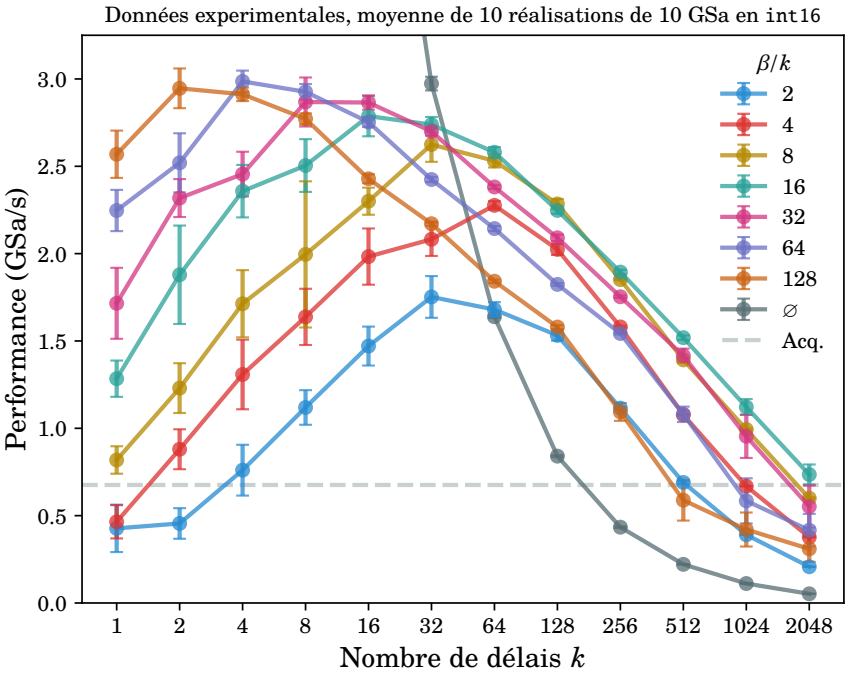


FIGURE 9.1 – Performance du calcul de l'autocovariance en fonction du nombre de délais k calculés. β est la taille des blocs de données utilisées pour les transformées de Fourier. Dans la légende, le symbole ∅ indique le calcul par produits directs et Acq. correspond au temps plancher d'acquisition et de transfert des données. Les barres d'erreur correspondent à l'écart-type des réalisations.

Ces résultats démontrent que, pour $k \lesssim 32$, l'algorithme par produits directs est plus performant que celui par transformées de Fourier, permettant d'ailleurs de calculer la variance ($k = 1$) à $\approx 15\,\mathrm{GSa/s}$. Cependant, à partir de $k \gtrsim 32$ l'utilisation des transformées de Fourier devient un avantage notable. Il est aussi clair que la taille de blocs $\beta = 16k$ est généralement la taille [17] optimale

16. Empiriquement, sur le système utilisé pour l'acquisition des données, les transformées de Fourier les plus efficaces (sans correction interbloc) sont celles effectuées sur 256 échantillons.
17. Correspondant à des transformées de Fourier de taille $2\beta = 32k$ à cause du *zero-padding*.

pour $k \gtrsim 32$.

Notons que dans la majorité des situations testées le calcul des autocorrélations voulues est plus rapide que l'acquisition et le transfert des données, ce qui permet de minimiser les temps morts en traitant les données d'un bloc pendant l'acquisition du bloc subséquent. Cependant, utiliser le bon algorithme, et les paramètres optimaux pour celui-ci, permet de libérer du temps pour effectuer d'autres opérations sur les données, comme établir leur histogramme — voir la section B.1 — ou bien en extraire et en éliminer la partie déterministe — voir la section 9.3.

Notons que, pour chaque série de données, tous les calculs effectués lors de ces tests ont donné exactement les mêmes résultats. Les stratégies utilisées pour minimiser les erreurs d'arrondi semblent donc adéquates.

9.2 Autocorrélations résolues en phases

9.2.1 Description générale

La carte d'acquisition ultrarapide présentée à la section 8.3 permet d'utiliser une horloge de référence externe afin de synchroniser l'acquisition avec d'autres appareils. En particulier, il est ainsi possible de synchroniser la mesure avec la photoexcitation de la jonction tunnel. Dans ce cas, si le rapport $\lambda = f_e/f_0$ entre la fréquence d'échantillonnage f_e et la fréquence de photoexcitation f_0 est un nombre entier, λ correspondra *exactement* au nombre d'échantillons associés à une période de f_0 ; c'est ni plus ni moins que la période en termes de nombre d'échantillons. Ainsi, chaque échantillon peut être associé à une phase précise de la photoexcitation selon son index, de manière similaire à ce qui est décrit à la section 7.6.1. Il est alors possible d'implémenter l'analogue du calcul des autocorrélations résolues en phase de la section 7.6.3.

Notons que la synchronisation entre la source et la carte d'acquisition est primordiale pour effectuer des mesures résolues en phase ; simplement ajuster f_e et f_0 n'est pas suffisant. En effet, si les appareils ne partagent pas une référence

d'horloge, leurs références respectives peuvent subir des dérives ou présenter un désaccord en fréquence. Puisque les mesures sont moyennées sur de longues périodes, une telle désynchronisation — même minime — implique que λ n'est plus entier et les mesures faites en présumant que les phases sont résolues se voient plutôt moyennées sur toute la période.

Les sous-sections qui suivent présentent l'algorithme et le code utilisés pour calculer les autocorrélations résolues en phase du signal.

9.2.2 Description des algorithmes

Soit un élément de vecteur x_i avec $0 \leq i < N$ représentant une mesure de N échantillons acquis à la fréquence d'échantillonnage f_e, elle-même synchronisée en phase avec une source à la fréquence f_0 tel que $f_e/f_0 = \lambda$ et commensurable avec celle-ci tel que $\lambda \in \mathbb{N}$. La source de référence permet donc de résoudre un nombre λ de phases ϕ séparées de $2\pi/\lambda$ radians. On défini l'index de phase $\varphi = \phi\lambda/2\pi$, tel que $\varphi \in \mathbb{N}$ et $0 \leq \varphi < \lambda$, qui représente les λ phases résolues et les relies à la phase réelle ϕ en radian. Ainsi, l'ensemble des échantillons respectant $x_{\varphi+j\lambda}$ avec $j \in \mathbb{N}$ tel que $\varphi + j\lambda < N$ seront associés au même index de phase φ et donc à la même phase ϕ.

Concrètement, si $f_e = 32\,\mathrm{GHz}$ et $f_0 = 4\,\mathrm{GHz}$, on a alors une référence de phase d'une période de $\lambda = 8$ échantillons et on distingue $\lambda = 8$ phases $\phi = 0, \pi/4, \cdots, 7\pi/4$ correspondant aux index de phase $\varphi = 0, 1, \cdots, 7$.

On cherche ici à calculer la version discrète de l'autocovariance résolue en phase de la trace temporelle x_i, tel que discuté à la section 7.6.3. On calcule l'autocovariance plutôt que l'autocorrélation pour les mêmes raisons qu'à la section 9.1. En pratique, on cherche simplement à généraliser le a_k de l'équation (9.7) pour obtenir un nombre λ de $a_{\varphi,k}$ adaptés à la situation résolue en phase. On note que, grâce à la propriété de symétrie (7.159), on peut se concentrer sur les valeurs de k positives et reconstruire les valeurs négatives par la suite.

Remarquons que, pour chaque index de phase φ, on peut conceptuellement séparer les données en blocs de période λ pour obtenir le nombre α_φ d'échantillons

associés à φ, soit

$$\alpha_\varphi = \underbrace{N /\!\!/ \lambda}_{\text{Blocs complets}} + \underbrace{[\![\varphi < N\%\lambda]\!]}_{\text{Bloc partiel potentiel}}, \tag{9.18}$$

avec $/\!\!/$ dénotant la division entière, $\%$ représentant l'opération *modulo* [18] et où le crochet double $[\![\cdot]\!]$ représente le crochet d'Iverson [116, §1.4 ; 117, §1] qui indique une comparaison logique évaluée à 1 si elle est respectée et à 0 sinon. Similairement, la phase de référence φ et le décalage en k viennent limiter le nombre de blocs participant au calcul de l'autocovariance résolue en phase $a_{\varphi,k}$; on obtient le nombre $n_{\varphi,k}$ de blocs participant au calcul de $a_{\varphi,k}$ via

$$n_{\varphi,k} = \underbrace{\alpha_{(\varphi+k)\%\lambda}}_{\text{Blocs phase décalée}} - \underbrace{(\varphi+k) /\!\!/ \lambda}_{\text{Blocs sautés au début}}. \tag{9.19}$$

Remarquons que $n_{\varphi,k} = n_{\varphi+k,0}$, par symétrie des effets de bords. On peut dès lors définir la version discrète de l'autocovariance résolue en phase [19] par

$$a_{\varphi,k} = \frac{1}{n_{\varphi,k}} \sum_{j=0}^{n_{\varphi,k}-1} \left(x_{\varphi+j\lambda} - \mu_{\varphi,k,0}\right)\left(x_{\varphi+j\lambda+k} - \mu_{\varphi,k,k}\right) \tag{9.20}$$

avec

$$\mu_{\varphi,k,b} = \frac{1}{n_{\varphi,k}} \sum_{j=0}^{n_{\varphi,k}-1} x_{\varphi+j\lambda+b} \tag{9.21}$$

$$\text{pour} \quad b \in \mathbb{N} \quad \text{et} \quad 0 \leq b \leq k, \tag{9.22}$$

la moyenne partielle effectuée sur les b^{e} échantillons des $n_{\varphi,k}$ blocs considérés. À l'instar de la démarche de la section 9.1.2, on peut réexprimer cette équation

18. La définition du modulo de [95, §1.2.4] diffère de celle utilisée en C++, mais les deux concordent pour des opérandes positifs.
19. Essentiellement la version discrète et centrée en zéro du $S_\phi(\tau)$ de (7.109).

sous la forme

$$a_{\varphi,k} = \sum_{j=0}^{n_{\varphi,k}-1} \frac{x_{\varphi+j\lambda} x_{\varphi+j\lambda+k}}{n_{\varphi,k}} - \mu_{\varphi,k,0}\mu_{\varphi,k,k}\,, \tag{9.23}$$

ou encore sous une forme plus propice au calcul numérique par accumulateurs entiers, soit

$$\boxed{a_{\varphi,k} = \frac{r_{\varphi,k}}{n_{\varphi,k}} - \left(\frac{M_\varphi - \beta_{\varphi,k}}{n_{\varphi,k}}\right)\left(\frac{M_{(\varphi+k)\%\lambda} - \gamma_{\varphi,k}}{n_{\varphi,k}}\right)}\,, \tag{9.24}$$

avec les accumulateurs de *type* autocorrélation et moyenne non normalisés

$$r_{\varphi,k} = \sum_{j=0}^{n_{\varphi,k}-1} x_{\varphi+j\lambda}\, x_{\varphi+j\lambda+k} \qquad \text{et} \qquad M_\psi = \sum_{j=0}^{\alpha_\psi-1} x_{\psi+j\lambda} \tag{9.25}$$

ainsi qu'avec les corrections de bords

$$\beta_{\varphi,k} = \sum_{j=n_{\varphi,k}}^{\alpha_\varphi-1} x_{\varphi+j\lambda} \qquad \text{et} \qquad \gamma_{\varphi,k} = \sum_{j=0}^{(\varphi+k)//\lambda-1} x_{(\varphi+k)\%\lambda+j\lambda}. \tag{9.26}$$

On utilise l'indice ψ pour souligner le fait que deux indices de M_ψ, soit $\psi = \varphi$ et $\psi = (\varphi + k)\,\%\lambda$, contribuent à $a_{\varphi,k}$.

On retrouve la situation non-résolue en phase en posant $\lambda = 1$, ce qui implique que seul l'index de phase $\varphi = 0$ est accessible ; les équations (9.18) et (9.19) impliquent alors $\alpha_0 = N$ et $n_{0,k} = N - k$. En substituant ces résultats dans (9.25), (9.26) et (9.24) — et en comparant à (9.8), (9.9) et (9.7) — on trouve

$$r_{0,k} = r_k\,, \qquad M_0 = M, \qquad \beta_{0,k} = \beta_k \qquad \text{et} \qquad \gamma_{0,k} = \gamma_k\,, \tag{9.27}$$

si bien que

$$a_{0,k} = a_k \qquad \text{pour} \qquad \lambda = 1. \tag{9.28}$$

L'approche résolue en phase est donc cohérente avec l'autocovariance régulière

présentée à la section 9.1.

Il est aussi possible d'obtenir l'autocovariance non-résolue en phase dans le cas $\lambda \neq 1$, avec très peu d'impact sur le temps de calcul, en adaptant légèrement l'approche résolue en phase. En effet, on remarque que, par construction, on a

$$N = \sum_{\varphi=0}^{\lambda-1} \alpha_\varphi \qquad \text{et} \qquad N - k = \sum_{\varphi=0}^{\lambda-1} n_{\varphi,k}\,, \tag{9.29}$$

de telle sorte que

$$r_k = \sum_{\varphi=0}^{\lambda-1} r_{\varphi,k}\,, \qquad\qquad M = \sum_{\varphi=0}^{\lambda-1} M_\varphi\,, \tag{9.30}$$

$$\beta_k = \sum_{\varphi=0}^{\lambda-1} \beta_{\varphi,k} \qquad \text{et} \qquad \gamma_k = \sum_{\varphi=0}^{\lambda-1} \gamma_{\varphi,k}. \tag{9.31}$$

Cependant, il est important de remarquer que, dû aux effets de bords ainsi qu'aux produits et divisions de (9.24), on a

$$a_k \neq \frac{1}{\lambda} \sum_{\varphi=0}^{\lambda-1} a_{\varphi,k}\,, \tag{9.32}$$

bien que cette expression tende vers l'égalité dans la limite $N \to \infty$; quand les effets de bords sont négligeables.

C'est donc dire que les accumulateurs de (9.25) et (9.26) — de pair avec les relations (9.29), (9.30) et (9.31) — permettent de calculer le a_k non résolu en phase selon l'équation (9.7). Tout comme le calcul de $a_{\varphi,k}$, le calcul de a_k se fait à la toute fin de l'acquisition de donnée. Du point de vue de la performance, il est donc à toutes fins pratiques *gratuit* de calculer a_k en plus de $a_{\varphi,k}$ lors de mesures résolues en phase.

9.2.3 Code et interface

Le code utilisé pour calculer l'autocovariance résolue en phase du signal est incorporé à celui des autocovariances régulières et est disponible à l'annexe B.2.2. La structure de base de la classe résolue en phase est héritée de la classe d'autocorrélation régulière par produit direct. Elle est donc très similaire à cette dernière du point de vue des considérations numériques des accumulateurs, de l'utilisation des nombres à virgule flottante à précision arbitraire, aux types de données supportés et à l'utilisation de `OpenMP` — voir la section 9.1.3 pour plus de détails sur la partie du code commune aux deux approches. Les fichiers spécifiques à la situation résolue en phase sont identifiés par le mot `Phi` dans leurs noms.

Les classes résolues en phases sont accessibles à travers la fabrique `ACorrUpTo` de l'interface `Python` `acorrs_otf.py`. Il suffit de passer l'argument `phi=`λ à `ACorrUpTo` pour calculer les λ autocovariances résolues en phases ; si `phi` est nul ou `False`, l'autocorrélation régulière sera utilisée. Une fois l'accumulation terminée, les autocorrélations résolues en phases sont accessibles à travers l'attribut `res` de la classe, alors que l'autocorrélation régulière est accessible par l'attribut `res0`.

Notons que l'accumulation résolue en phase est légèrement plus lente que celle du cas régulier par produit direct, et que l'optimisation par transformée de Fourier [20] ne s'y applique pas ; il est donc inutile de l'utiliser dans une situation sans référence de phase.

L'extrait de code qui suit est une modification de celui de la section 9.1.3 pour la situation résolue en phase.

```
1 import numpy as np
2 from acorrs_otf import ACorrUpTo
3 # Instantification préalable. On peut aussi instantifier avec des données en
4 # second argument, e.g. "a = ACorrUpTo(100,x)", *x* sera alors accumulé.
5 # Considérons acq. à 32GSa/s et une référence de phase à 4GHz, donc 8 phases.
6 a = ACorrUpTo(100, 'int16', phi=8) # 100 première autocovariances, pour 8 phases.
7 # Simulons 10 mesures 16bit signées
```

20. La transformée de Fourier 2D pourrait être utile à l'optimisation ; on ne s'y est pas attardé.

```
for n in range(10):
    # *x* simule une trace temporelle expérimentale
    x = np.random.randint(-2**15,2**15, size=2**20, dtype=np.int16)
    a(x)    # On accumule dans l'objet *a*
# a.res et a.res0 contiennent les moyennes d'ensemble des autocovariances
# résolues en phase et régulière des dix *x*.
print a.res, a.res0  # Affiche à la suite les résultats résolus en phase ou non.
```

9.3 Composante déterministe du signal

9.3.1 Description générale

La section 8.5 décrit la partie déterministe du signal qui est présente au sein du signal lors de mesures synchrones avec une excitation, ce qui correspond à la situation résolue en phase de la section 9.2. Conceptuellement, cette contribution fait en sorte que tous les échantillons séparés d'une période d'excitation voient la même amplitude du signal résiduel de photoexcitation.

Cette partie déterministe peut donc être extraite des mesures en faisant la moyenne sur les périodes — la moyenne du i^{e} point de chaque période pour toute la période — de manière à ce que les fluctuations s'estompent et que la seule contribution non-nulle soit la partie déterministe. Il est alors possible de soustraire la partie déterministe du signal pour ne garder que les fluctuations, une opération similaire à la compensation analogique, et même d'en extraire la phase de l'excitation résiduelle par lissage ou transformée de Fourier. Cette dernière approche est d'ailleurs utilisée à la section 8.5.2 dans le cadre de la stabilisation de la phase pendant de longues mesures.

Le signal d'excitation résiduel — typiquement un artefact néfaste — s'avère ici surprennamment utile, permettant de connaître la phase de la mesure comparativement à l'excitation [21] et de faire des mesures en bande large sans que l'excitation vienne polluer les résultats.

21. À une phase globale près, bien sûr.

9.3.2 Description des algorithmes

Considérons une mesure expérimentale représentée par un élément de vecteur x_i tel que décrit à la section 9.2.2. Connaissant le nombre α_φ d'échantillons associés à l'index de phase φ, voir l'équation (9.18), on extrait la partie déterministe du signal associé à cette même phase via

$$\bar{x}_\varphi = \frac{1}{\alpha_\varphi} \sum_{j=0}^{\alpha_\varphi - 1} x_{\varphi + j\lambda}. \tag{9.33}$$

Une fois $\bar{x}_\varphi$ extrait pour tous les index de phase $0 \leq \varphi < \lambda$, on peut simplement enlever la partie déterministe du signal via

$$\check{x}_i = x_i - \bar{x}_{i\%\lambda} \tag{9.34}$$

pour obtenir $\check{x}_i$, le signal avec la partie déterministe filtrée. Il est ensuite possible de traiter le signal $\check{x}_i$ comme n'importe quel autre signal, sans la contribution de la partie déterministe du signal.

9.3.3 Code et interface

Le code utilisé pour extraire et retirer la partie déterministe du signal est disponible à la section B.2.3. Il est constitué de fonctions `C++` *templatées* et d'une interface `Python` elle aussi rédigée en `C++` via la librairie `pybind11` [110]. Une seconde interface en python ne sert qu'à faciliter le chargement des bonnes librairies selon le système d'exploitation utilisé.

Trois fonctions sont accessibles par l'interface python, soit `getdet`, `deldet` et `remdet`. La fonction `getdet` prend en entrée des données et une période et retourne la partie déterministe en sortie ; c'est l'implémentation de (9.33). Quant à elle, `deldet` prends en entrée des données ainsi qu'une partie déterministe et en fait la soustraction en mémoire tel que décrit par (9.34) ; les données d'entrée sont donc modifiées. Enfin, `remdet` combine ces deux fonctions en prenant en entrée des données et la période attendue pour ensuite à la fois retourner la partie

déterministe en sortie et soustraire celle-ci des données d'entrée en mémoire.

Notons que bien que le code supporte les types de données non-signés via l'utilisation de templates, le résultat de `deldet` sur ce type de donnée peut être surprenant. En effet, comme le résultat de la soustraction est aussi non-signé, les points qui seraient autrement négatifs se retrouvent plutôt vers la limite supérieure supportée par le type.

9.3.4 Exemples sur données expérimentales

Un exemple d'extraction de la partie déterministe, de la soustraction [22] de celle-ci directement sur la trace temporelle $i(t)$, ainsi que des effets de cette opération sur les autocorrélations $S_\mathrm{m}(\tau)$ et densités spectrales $\tilde{S}_\mathrm{m}(f)$ est présenté à la figure 9.2.

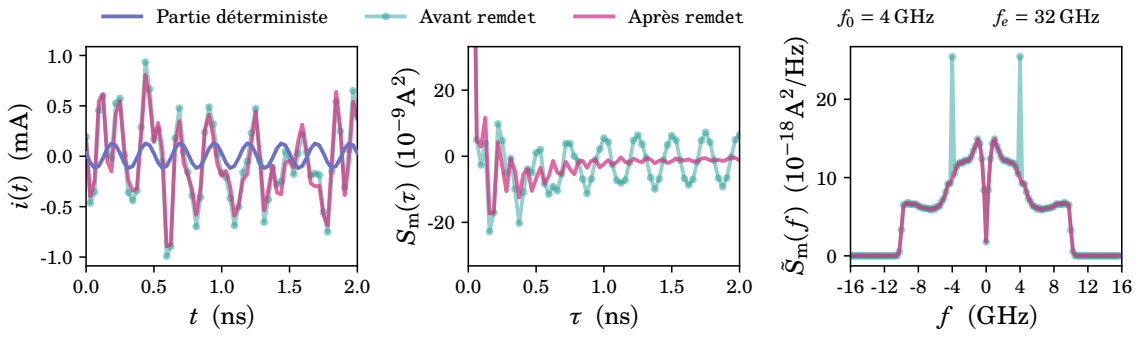


FIGURE 9.2 – Exemple de `remdet`. Extraction et compensation de la partie déterministe sur des données expérimentales.

La sous-figure de gauche montre une partie de la trace temporelle brute, soit les premières 2 ns sur une trace totale de ≈ 33.25 ms — correspondant aux 64 premiers points d'une acquisition de $2^{30} \sim 10^9$ points. La partie déterministe qui y est montrée a été extraite selon (9.33) en moyennant par bloc toutes les données ; les $f_e/f_0 = 8$ points la constituant sont répétés par souci de clarté. On remarque que la partie déterministe représente une partie minime des données

22. Opération qu'on appelle `remdet`.

brutes, la trace n'étant pratiquement pas affectée par sa soustraction. La phase extraite de la partie déterministe selon (9.34) est de $\approx 79.2°$.

Les sous-figures du centre et de droite montrent respectivement l'effet de la soustraction de la partie déterministe sur l'autocorrélation et la densité spectrale qui en est déduite. L'effet est frappant sur l'autocorrélation, où une partie oscillante importante causée par la présence de l'excitation résiduelle est complètement éliminée. Sans grande surprise, cette oscillation dans l'autocorrélation se traduit par un pic important dans la densité spectrale, qui se voit lui aussi éliminé de manière convaincante après `remdet`.

On note que l'opération de `remdet` ne correspond pas à un filtre numérique typique, auquel cas la densité spectrale présenterait un creux à f_0 après la compensation. Elle ressemble plutôt à une version digitale de la compensation analogique. Elle isole et soustrait de la trace temporelle que les composantes du signal qui sont à f_0 *et* en phase avec l'excitation.

Chapitre 10

Résultats et Analyse : non résolus en phase

10.1 Modélisation de la mesure et calibration

Lors de mesures expérimentales, on ne vient pas directement sonder l'échantillon de manière idéalisée. Il faut en effet tenir compte des appareils de mesures et des imperfections du montage expérimental, qui viennent affecter la mesure — voir la section 8.1 pour les détails du montage.

En particulier, la chaîne d'amplification utilisée pour obtenir un signal d'amplitude assez grande pour être mesurée vient non seulement ajouter un préfacteur multiplicatif au signal, ce qui est l'effet désiré, mais elle ajoute aussi une certaine quantité de bruit à la mesure. De manière générale, pour les mesures préservant la phase, ce bruit supplémentaire lié à l'amplification ne peut pas être nul ; sa limite quantique est équivalente à la contribution du vide [7]. Cependant, seuls quelques modèles d'amplificateurs bien particuliers — à la fine pointe de la recherche [26, 28, 31, 32, 118, 119] — ont le potentiel d'approcher cette limite. La grande majorité des amplificateurs commerciaux ajoutent plutôt une quantité de bruit substantielle.

10.1.1 Modélisation de la mesure

Toute chaîne d'amplification peut être modélisée par un amplificateur idéal et une source de bruit supplémentaire à son entrée [33, §3 ; 44, §2.3.3]. On modélise donc la mesure à la fréquence f selon le modèle de la figure 10.1, qui consiste simplement à ajouter le bruit d'entrée $\tilde{S}_A(f)$ de l'amplificateur au signal de bruit intrinsèque — qu'on dénote $\tilde{S}(f)$, un spectre de puissance arbitraire — pour en amplifier la somme à l'aide d'un amplificateur idéalisé de gain $\tilde{G}_A(f)$ en puissance. Mathématiquement, pour un bruit intrinsèque $\tilde{S}(f)$ quelconque, on mesure $\tilde{S}_\mathrm{m}(f)$, soit le bruit modifié par le montage, tel que

$$\boxed{\tilde{S}_\mathrm{m}(f) = \tilde{G}_A(f)\big(\tilde{S}(f) + \tilde{S}_A(f)\big)}. \tag{10.1}$$

Notons que les pertes — qu'elles proviennent du désaccord d'impédence à l'entrée de l'échantillon, d'autres réflexions ou de l'atténuation des câbles — sont automatiquement considérées ici ; elles viennent simplement diminuer le gain effectif du modèle. L'expression équivalente dans le domaine temporel est alors obtenue par transformée de Fourier, soit

$$\begin{aligned} S_\mathrm{m}(\tau) &= \mathcal{F}^{-1}\left[\tilde{S}_\mathrm{m}(f)\right](\tau) && (10.2)\\ &= G_A(\tau) * \big(S(\tau) + S_A(\tau)\big). && (10.3) \end{aligned}$$

Donc, le corrélateur d'intérêt — qui est déjà petit comparativement au bruit d'amplification — se voit convolué par la fonction de réponse temporelle de la chaîne d'amplification, ce qui le déforme d'autant plus que le gain est grand, le rendant méconnaissable de prime abord. C'est donc par souci de commodité qu'on préfère travailler dans le domaine fréquentiel, bien que la calibration puisse — conceptuellement — tout aussi bien être effectuée dans le domaine temporel.

En pratique, c'est la version discrète du $S_\mathrm{m}(\tau)$ de l'équation (10.3) qui est mesurée selon l'algorithme décrit à la section 9.1.2 et le $\tilde{S}_\mathrm{m}(f)$ associé en est déduit par transformée de Fourier discrète. Chaque mesure d'autocovariance large bande est donc équivalente à plusieurs mesures en bande étroite ; une pour

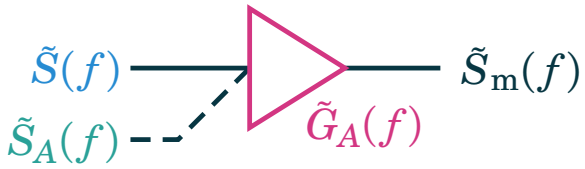


FIGURE 10.1 – Modèle de la mesure. $\tilde{S}(f)$ est le bruit intrinsèque qu'on cherche à mesurer. $\tilde{S}_A(f)$ et $\tilde{G}_A(f)$ sont, respectivement, le bruit ajouté par l'amplificateur ainsi que son gain. $\tilde{S}_\mathrm{m}(f)$ est le bruit qui est mesuré expérimentalement — voir l'équation (10.1).

chacune des fréquences résolues par la mesure en bande large.

Cette modélisation de la mesure expérimentale a un rôle double. Premièrement, dans les bonnes conditions expérimentales, elle permet de calibrer $\tilde{G}_A(f)$ et $\tilde{S}_A(f)$. Ensuite, une fois ces quantités calibrées, elle permet d'obtenir le bruit intrinsèque $\tilde{S}(f)$ à partir de mesures de $\tilde{S}_\mathrm{m}(f)$ simplement en inversant (10.1) pour obtenir

$$\tilde{S}(f) = \frac{\tilde{S}_\mathrm{m}(f)}{\tilde{G}_A(f)} - \tilde{S}_A(f). \tag{10.4}$$

10.1.2 Calibration via limite à grand V_{dc}

Le principe général de la calibration est somme toute assez simple ; il ne suffit que de trouver un régime où $\tilde{S}(f)$ est connu et de l'utiliser pour déduire le gain et le bruit d'amplification à partir de (10.1). On présente ici la méthode de calibration utilisée pour la majorité des résultats. Il s'agit essentiellement de la technique des références [52, §3.5 ; 53], adaptée aux mesures large bande et, le cas échéant, à la photoexcitation.

La forme attendue pour le bruit photoassisté d'une jonction tunnel est celle de l'équation (7.87), qu'on peut exprimer en fonction des variables expérimentales comme

$$\hat{\tilde{S}}^{\mathrm{pa}}(f, V_\mathrm{dc}, f_0, V_\mathrm{ac}) = \sum_{n=-\infty}^{\infty} J_n^2\left(\frac{eV_\mathrm{ac}}{hf_0}\right) \tilde{S}^{\mathrm{dc}}\Big(2\pi\left(f + nf_0\right), eV_\mathrm{dc}/\hbar\Big), \tag{10.5}$$

avec $\tilde{S}^{\mathrm{dc}}(\omega, \nu)$, une fonction de $\tilde{S}_{\mathrm{eq}}$ donnée par l'équation (7.86). La relation (7.53) implique alors que la limite à grande tension continue de (10.5) est

$$\hat{\tilde{S}}^{\mathrm{pa}}(f, V_{\mathrm{dc}}, f_0, V_{\mathrm{ac}}) = \frac{e\,|V_{\mathrm{dc}}|}{R} \qquad \text{pour} \qquad e\,|V_{\mathrm{dc}}| \gg \{|hf|, |hf_0|, |eV_{\mathrm{ac}}|\}\,; \quad (10.6)$$

une simple fonction valeur absolue. Notons que la valeur de $|f|$ pertinente aux comparaisons d'échelles d'énergie est la plus grande fréquence de la bande passante analogique, soit 10 GHz ici. Dans ce régime, la modélisation de la mesure de (10.1) donne donc

$$\tilde{S}_{\mathrm{m}}(f, V_{\mathrm{dc}}) = \tilde{G}_A(f)\left(\frac{e\,|V_{\mathrm{dc}}|}{R} + \tilde{S}_A(f)\right). \quad (10.7)$$

À f donnée, cela correspond simplement à une droite de pente $\tilde{G}_A(f)\,e/R$ et d'ordonnée $\tilde{G}_A(f)\,\tilde{S}_A(f)$. Il suffit ainsi de prendre des mesures à différentes valeurs de V_{dc} dans ce régime et d'effectuer un lissage linéaire pour extraire $\tilde{G}_A(f)$ et $\tilde{S}_A(f)$. Répéter cette opération pour chacune des fréquences discrètes résolues expérimentalement permet en pratique de calibrer le système pour l'ensemble des données et donc d'extraire le spectre de bruit intrinsèque recherché à partir des mesures brutes.

10.1.3 Exemple expérimental concret

Pour illustré le principe de la calibration, prenons des mesures de bruit aux bornes d'une jonction tunnel polarisée par une tension continue V_{dc} sans photoexcitation. Les données ont été acquises à la fréquence d'échantillonnage $f_e = 32$ GHz sur une bande passante de 10 GHz via le script de la section C.2.1. Elles consistent en des autocorrélations simples pour 26 valeurs de $V_{\mathrm{dc}} \geq 0$, incluant 4 valeurs à grande tension $eV_{\mathrm{dc}}/h \gg 10$ GHz pour la calibration. Pour chaque valeur de tension, plus de 22 téraoctets de données ont été acquises à la carte d'acquisition et traitées à la volée par le code présenté à la section 9.1 pour un total d'environ 594 téraoctets traités.

Sous biais en tension continue seulement, le signal est stationnaire au sens large et on peut poser $\hat{S}_{\mathrm{m}}^{\mathrm{dc}} \equiv S_{\mathrm{m}}^{\mathrm{dc}}$ en tant que corrélateur mesuré à la carte

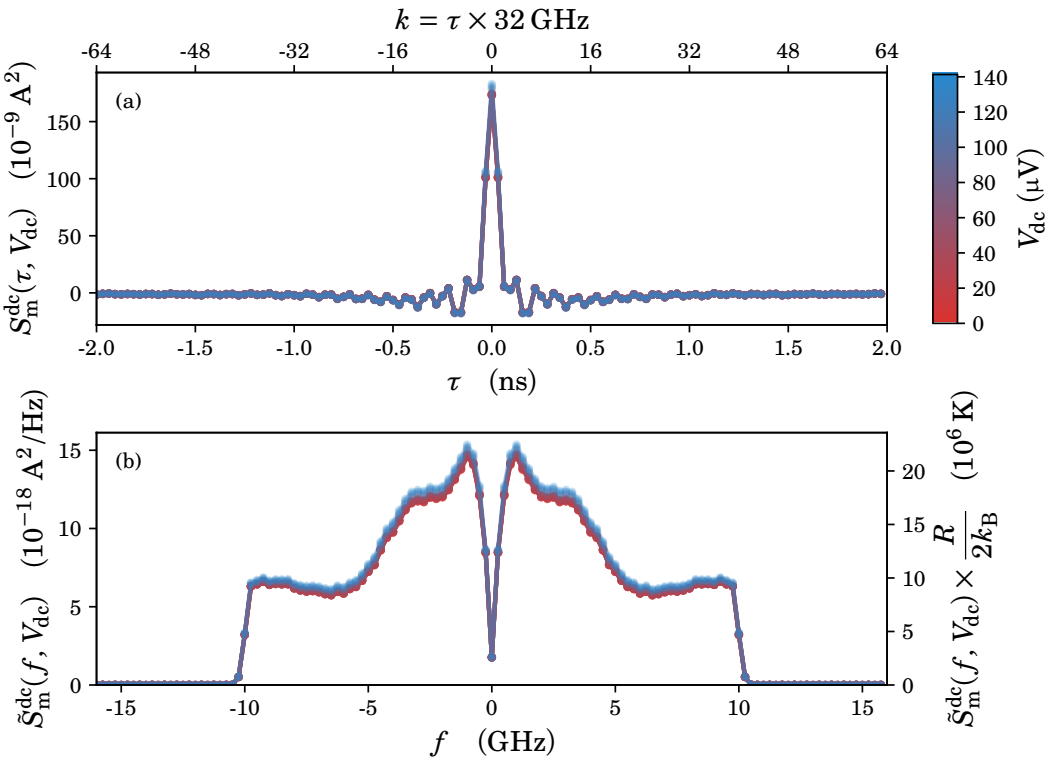


FIGURE 10.2 – Données brutes sous polarisation DC. (a) Autocovariances mesurées. (b) Densités spectrales de bruit associées. L'axe en haut de (a) présente les données selon l'index de décalage discret k plutôt qu'en fonction du décalage temporel τ.

d'acquisition. L'équation 10.1 devient alors

$$\tilde{S}_{\mathrm{m}}^{\mathrm{dc}}(f, V_{\mathrm{dc}}) = \tilde{G}_A(f)\Big(\tilde{S}^{\mathrm{dc}}(f, V_{\mathrm{dc}}) + \tilde{S}_A(f)\Big), \tag{10.8}$$

soit

$$S_{\mathrm{m}}^{\mathrm{dc}}(\tau, V_{\mathrm{dc}}) = G_A(\tau) * \Big(S^{\mathrm{dc}}(\tau, V_{\mathrm{dc}}) + S_A(f)\Big) \tag{10.9}$$

dans le domaine temporel. On s'intéresse, au bout du compte, à la plage de tension V_{dc} d'échelle d'énergie inférieure à la bande passante analogique de 10 GHz de la mesure, soit $V_{\mathrm{dc}} \lesssim 41.36\,\mu\mathrm{V}$. Cependant, en prévision de la calibration, quatre

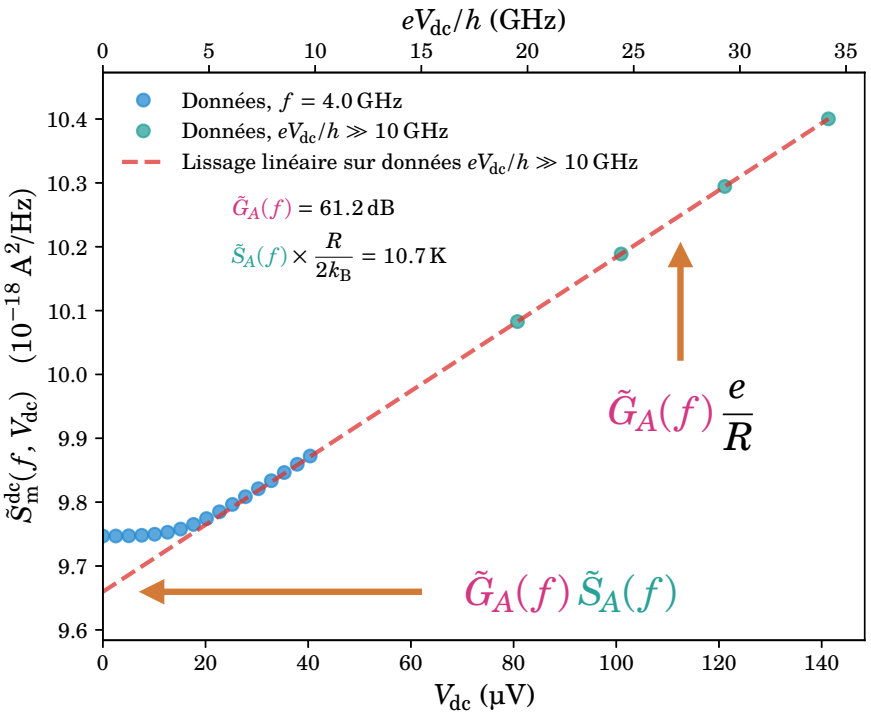


FIGURE 10.3 – Exemple de calibration à grand V_{dc} pour une tranche des données à $f = 4.0\,\mathrm{GHz}$.

mesures ont aussi été effectuées dans le régime $eV_{\mathrm{dc}}/h \gg 10\,\mathrm{GHz}$. [1]

La figure 10.2 présente, à la sous-figure (a), les autocovariances brutes telles que mesurées expérimentalement pour toutes les valeurs de V_{dc} appliquées. Les courbes sont pratiquement indistinguables les unes des autres, étant donné que le terme la convolution avec $G_A(\tau)$ de (10.9) domine comparativement à la contribution de $S^{\mathrm{dc}}(\tau, V_{\mathrm{dc}})$; on voit essentiellement la contribution de l'amplificateur. On obtient la sous-figure (b) en prenant la transformée de Fourier des courbes de la sous-figure (a), de manière à obtenir les densités spectrales associées et de se débarrasser de la convolution. La différence entre les différentes tensions de polarisation est plus évidente, pour une valeur de f donnée, dans le domaine fréquentiel. Cela dit, la forme des spectres est toujours dominée par la contribution du montage expérimental. En particulier, on voit bien l'effet de la bande

1. Notons que le V_{dc} est corrigé pour la résistance du T de polarisation tel que décrit à la section 10.1.4.

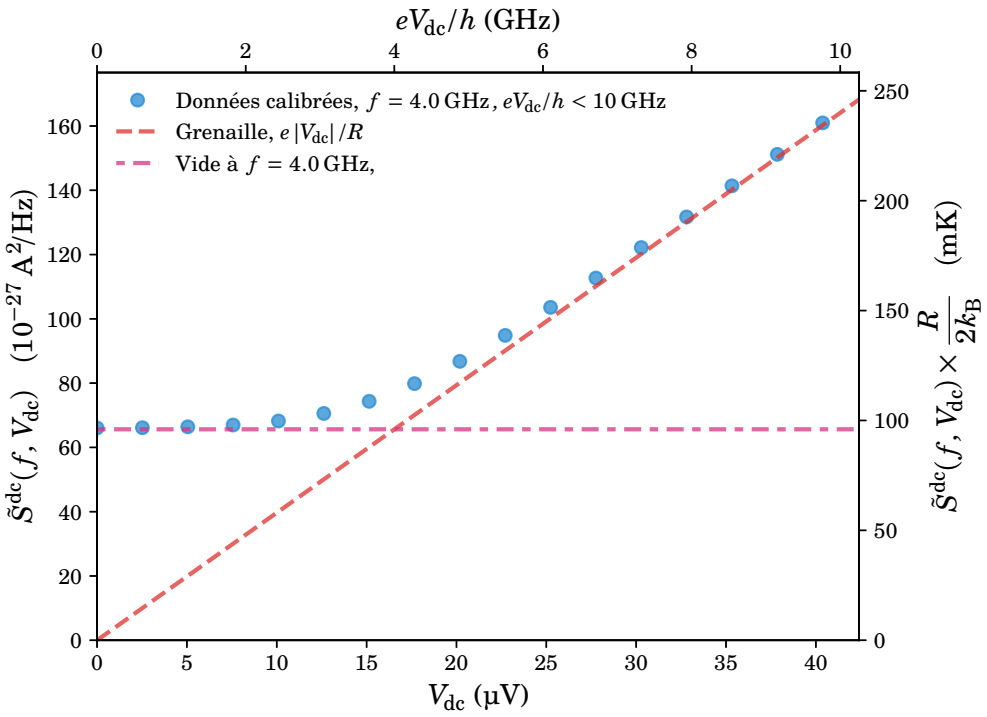


FIGURE 10.4 – Données de la figure 10.3 après calibration pour $eV_{dc} < hf$.

passante analogique de la carte d'acquisition qui filtre le signal à $f > 10$ GHz, bien que l'on résolve jusqu'à 16 GHz d'après le critère de Nyquist [54]. Le creux observé pour $|f| \lesssim 250$ MHz, soit les trois points centraux, est quant à lui attribuable à la limite inférieure de la bande passante de l'amplificateur cryogénique LNF-LNC1_12A. Malgré tout, il est clair que la calibration est primordiale pour obtenir des résultats quantitatifs convaincants, spécialement si on s'intéresse à des résultats à plusieurs fréquences.

Le principe de la calibration est illustré à la figure 10.3 pour la tranche des données correspondant à $f = 4.0$ GHz. On y voit en bleu les données correspondant aux valeurs de V_{dc} inférieures à 10 GHz, soient les données d'intérêts. Les données à grand V_{dc}, destinées à la calibration, sont plutôt présentées en vert. Un lissage linéaire de la forme (10.7) est effectué sur celles-ci et est prolongé sur toute la plage de données pour mettre en évidence son ordonnée à l'origine. La valeur de gain ainsi obtenue est d'environ 61.2 dB ce qui est raisonnable considérant qu'on s'attend à avoir autour de 70 dB de gain provenant des ampli-

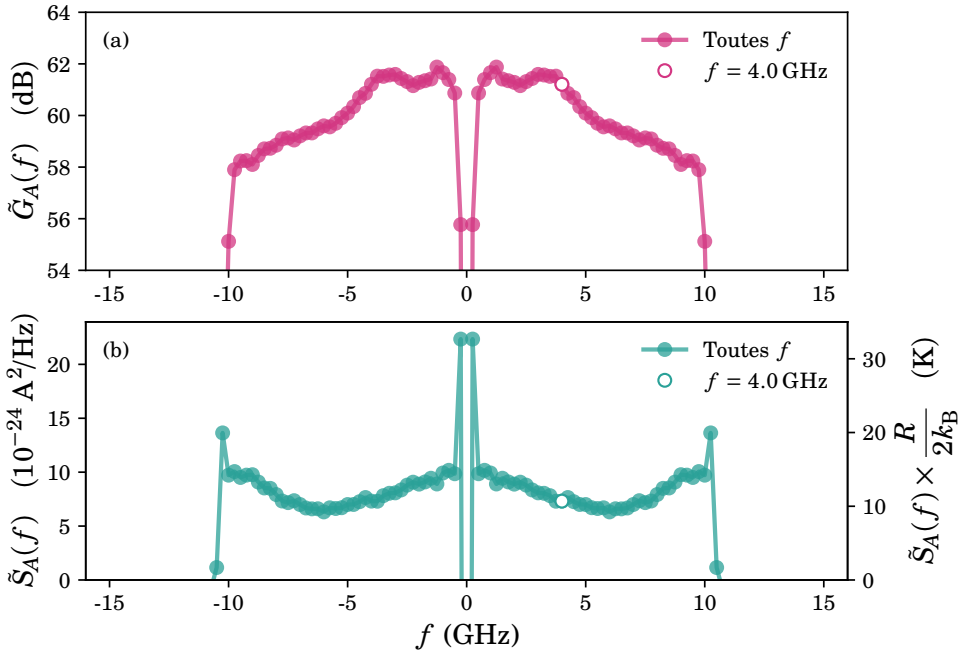


FIGURE 10.5 – Calibration de $\tilde{G}_A(f)$ et $\tilde{S}_A(f)$ pour toute la bande de mesure. Valeurs extraites des données de la figure 10.2 à grand V_{dc}. Les valeurs obtenues de la figure 10.3 sont identifiées par les marqueurs de centre blanc.

ficateurs. La différence est attribuable, entre autres, à l'atténuation des câbles et composantes, aux disparités d'impédance, aux réflexions dans le montage, à la variation du gain des amplificateurs sur leur bande passante ainsi qu'à la variabilité de leur ajustement — principalement dans le cas de l'amplificateur cryogénique qui est ajusté manuellement. La valeur de bruit d'amplification d'environ 10.7 K est aussi similaire à la valeur attendue d'après les spécifications de l'amplificateur cryogénique [2], celui ayant la contribution dominante [33, §3].

Ces paramètres permettent d'extraire $\tilde{S}^{\mathrm{dc}}(f, V_{\mathrm{dc}})$ à partir de la mesure de $\tilde{S}^{\mathrm{dc}}_{\mathrm{m}}(f, V_{\mathrm{dc}})$. La figure 10.4 en présente le résultat pour les données à $f = 4\,\mathrm{GHz}$ de la figure 10.3. On y distingue les deux régimes attendus, le bruit du vide et le bruit de grenaille, avec la largeur de la transition entre ceux-ci correspondant à la température électronique T_{e}. Comme attendu, la transition entre les régimes

2. Notons qu'on mesure typiquement le double de la valeur nominale des spécifications pour le bruit des amplificateurs, à cause des différentes conventions utilisées — voir la discussion sous l'équation (7.16).

est centrée en $eV_{\mathrm{dc}}/h = 4\,\mathrm{GHz}$, point auquel les deux contributions seraient équivalentes à $T_{\mathrm{e}} = 0$. Ces résultats concordent donc bien aux attentes théoriques, se qui confirme la probité de la méthode de calibration.

En répétant cette procédure pour toutes les fréquences f résolues expérimentalement, on obtient l'ensemble des valeurs de $\tilde{G}_A(f)$ et $\tilde{S}_A(f)$ pertinentes aux mesures. C'est ce qui est présenté à la figure 10.5, respectivement aux sous-figures (a) et (b). Tel que discuté pour les valeurs extraites de la figure 10.3, les valeurs de $\tilde{G}_A(f)$ et $\tilde{S}_A(f)$ obtenues sur toute la bande de mesure sont raisonnables. De plus, on remarque que le produit des sous-figures correspond essentiellement à la forme de la densité spectrale brute de la figure 10.2, tel qu'attendu des équations (10.1) et (10.8) considérant que le bruit d'amplification domine sur le bruit de la jonction.

Sans surprise, on constate aussi que la calibration est moins convaincante en dehors de la bande passante de mesure. Effectivement, le gain diminue drastiquement à $|f| \leq 250\,\mathrm{MHz}$ et $|f| > 10\,\mathrm{GHz}$, traduisant la faible quantité de signal parvenant à la carte d'acquisition à ces fréquences. Dans le cas de la température de bruit, on observe des pics près des limites de la bande passante suivis de diminutions rapides et importantes en sortie de celle-ci. Les pics s'expliquent probablement par la diminution du ratio signal sur bruit de la chaîne d'amplification aux bords de la bande, bien que le gain y subsiste suffisamment pour permettre la mesure. Selon la même logique, la diminution hors bande traduit le fait que pratiquement aucun signal n'y subsiste, pas même le bruit parasite. La calibration permet donc d'extraire la contribution de la jonction sur toute la bande passante de mesure, à l'instar de ce qui est fait à la figure 10.4. Ces résultats sont d'ailer ceux présentés à la figure 10.6 de la section 10.2.

En somme, l'approche de calibration est probante à l'intérieur de la bande passante expérimentale. Il importe cependant de ne pas présumer de sa validité en dehors de cette dernière. Le traitement des données doit impérativement en tenir compte pour ne pas introduire d'artefacts [3] dans les résultats, particulièrement lors de l'utilisation de transformées de Fourier pour changer de représentation.

3. Via le signal hors-bande insignifiant ou la coupure drastique du signal à la limite de la bande passante, par exemple.

10.1.4 Note sur la valeur de R

Comme montré au montage de la figure 8.1, il est possible de mesurer la résistance de la jonction tunnel en y imposant un courant connu et en mesurant la tension à ses bornes ; c'est la méthode commune de la courbe I-V basée sur la loi d'Ohm. Si l'injection du courant et la mesure de tension s'effectuaient par des paires de contacts distincts, on ferait une *mesure à quatre pointes* [120]. Cependant, le terminal de mise à la terre est partagé de telle manière à ce qu'on soit sensible à la résistance de celui-ci, dans une configuration qu'on pourrait qualifier de *mesure à trois pointes*. Dans cette configuration, on mesure une résistance d'environ 42.61 Ω à l'aide d'une courbe I-V standard.

Par contre, il est possible d'obtenir la vraie résistance de la jonction en utilisant les résultats de $\tilde{S}^{\mathrm{dc}}$ en fonction de f après calibration. En effet, à grande fréquence toutes les courbes devraient tendre vers une droite correspondant aux fluctuations du vide, tel que montré à la figure 10.6. Comme l'axe vertical de droite — en Kelvin — de cette figure dépend de R, il est alors possible d'ajuster la résistance pour que la prévision théorique de $\frac{h|f|}{2k_{\mathrm{B}}}$ en Kelvin concorde avec les résultats. C'est en quelque sorte une calibration de R à travers les fluctuations du vide. En utilisant ce principe, on détermine que la jonction a une résistance d'environ 40.38 Ω, correspondant à une résistance de contact d'environ 2.23 Ω pour la mise à la terre, une valeur raisonnable.

Notons que cet ajustement affecte la conversion $\mathrm{A}^2/\mathrm{Hz} \leftrightarrow \mathrm{K}$ ainsi que la relation entre la tension V'_{dc} appliquée par la source de tension hors cryostat et la tension V_{dc} en découlant aux bornes de la jonction. Ainsi, à travers V_{dc}, l'ajustement de R affecte légèrement la calibration utilisée pour ce même ajustement. On tient compte de cet effet lors de l'ajustement initial de telle sorte que les calibrations subséquentes soient en accord avec la limite des fluctuations du vide. La valeur de résistance R ainsi obtenue est celle utilisée lors de l'analyse ; elle a été déterminée préalablement au traitement des données du chapitre 10.

10.2 Résultats sous polarisation DC

Une fois la calibration faite, on applique celle-ci sur les les données de la section 10.1.3 à $eV_{\mathrm{dc}}/h < 10\,\mathrm{GHz}$ pour obtenir le bruit intrinsèque. C'est ce qu'on présente à la figure 10.6, soit $\tilde{S}^{\mathrm{dc}}(f)$ en fonction de la fréquence pour les différents V_{dc} d'intérêt. Les résultats sont très convaincants, avec toutes les courbes très lisses et les plateaux centraux s'élargissant avec V_{dc} comme attendu. La calibration, qui est faite à chaque f indépendamment, semble donc retirer l'effet du montage efficacement et de manière fiable.

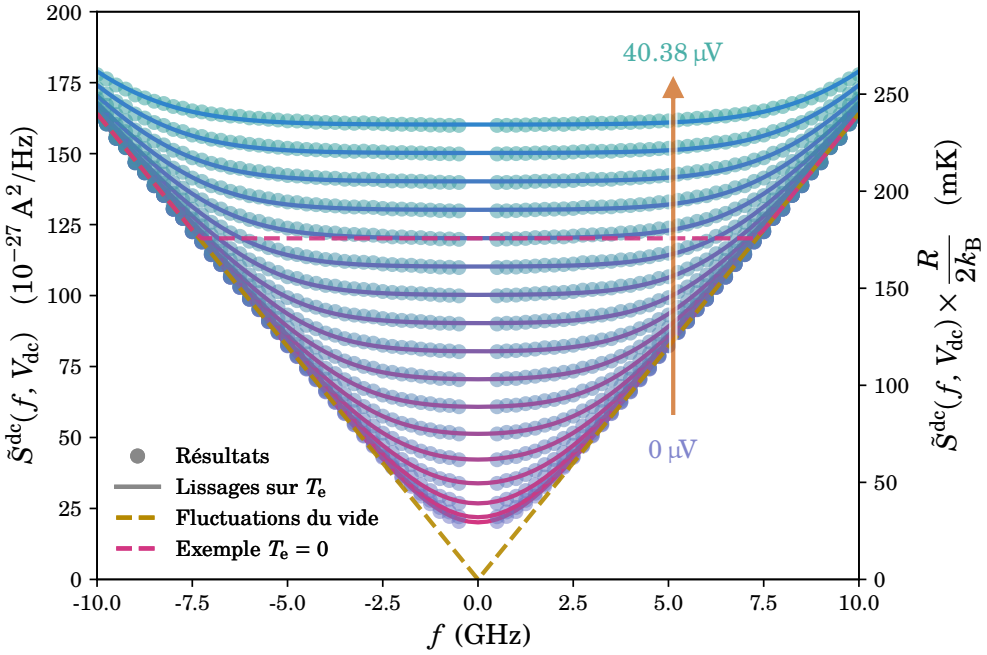


FIGURE 10.6 – Résultats de $\tilde{S}^{\mathrm{dc}}(f, V_{\mathrm{dc}})$ après calibration pour toute la bande de mesure. Les lignes correspondent à des régressions linéaires avec T_{e} comme seul paramètre libre. Les données en dehors de la bande passante à $f \approx 0$ ont été omises.

10.2.1 Effet de chauffage

En explorant les données, on a cependant remarqué que la température électronique semblait augmenter légèrement avec V_{dc}. Pour confirmer cela, on

applique un lissage de la forme de (7.86) sur les résultats, avec T_e comme seul paramètre libre. Les résultats de ces régressions sont visibles à la figure 10.6 sous forme de traits pleins. Les valeurs de T_e obtenues sont quant à elles présentées à la figure 10.7 en fonction de la puissance de polarisation. On y voit bien que la température électronique croît selon la puissance de polarisation, avec deux régimes distincts.

À basse polarisation, la pente de l'augmentation de T_e est assez raide et elle s'adoucit jusqu'à atteindre un régime linéaire à partir d'environ $V_{dc}^2/R = 7.5\,\text{pW}$. La résolution en V_{dc} ne permet pas de vérifier si l'augmentation initiale est linéaire ; on pourrait donc observer deux régimes linéaires avec une transition lisse entre les deux, ou un régime en $T_e \propto \left(V_{dc}^2/R\right)^n$ avec $0 < n < 1$ suivi de $T_e \propto V_{dc}^2/R$. Bien qu'il serait intéressant d'étudier ces régimes du point de vue du transport électronique [66, 121], on se contente ici d'un survol phénoménologique permettant de tenir compte de l'effet de chauffage lors de l'analyse des résultats, sans en expliquer les mécanismes fondamentaux.

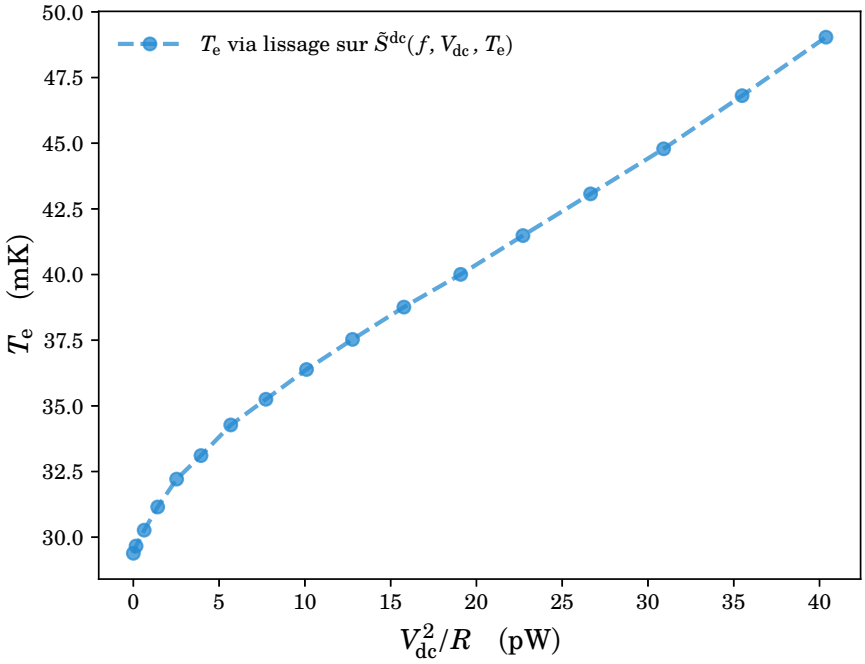


FIGURE 10.7 – Température électronique obtenue par lissage linéaire en fonction de la puissance de polarisation dc.

Cet effet de chauffage est assez inattendu, considérant que la plupart des mesures de bruit effectuées sur des jonctions tunnel présupposent — explicitement [59, 60, 63, 122] ou implicitement [61] — que T_e est indépendant de V_{dc}. On propose deux explications à l'observation du chauffage au sein de nos mesures.

Premièrement, les études citées ci-haut s'intéressent à des mesures de bruit en bande étroite. Ainsi, l'effet de la température y apparaît principalement à la transition entre le bruit du vide et le bruit de grenaille, à $V_{dc} \approx hf/e$, sur un éventail de V_{dc} de l'ordre de $\sim k_B T_e/e$. Il est tout à fait possible que la variation de la température causée par V_{dc} sur cette plage [4] soit négligeable ou à tout le moins suffisamment faible pour que la température moyenne sur la transition concorde avec les données de manière convaincante.

Deuxièmement, la jonction tunnel que l'on étudie ici est atypique. En effet, tel que discuté à la section 8.2, la jonction tunnel étudiée ici est en réalité une jonction Josephson au milieu d'un circuit supraconducteur au sein de laquelle on élimine la supraconductivité à l'aide d'un aimant ; elle est conçue pour agir en tant qu'amplificateur large bande [32]. Elle est donc beaucoup plus petite que les jonctions utilisées typiquement pour les mesures de bruit, en plus de se retrouver parmi un circuit conçu pour être supraconducteur et utilisé dans l'état normal. Les traces métalliques longues, fines et minces [5] qui forment ce circuit peuvent plus difficilement dissiper leur température dans les phonons — peu de matériel et surface de contact minime avec le substrat — ou dans des contacts macroscopiques — qui sont éloignés de la jonction. Le courant appliqué à travers la jonction vient donc probablement chauffer non seulement celle-ci, mais aussi le circuit dans lequel elle réside, ce qui la chauffe à son tour d'autant plus aisément qu'elle est petite. Bref, on suppose normalement que les contacts de la jonction tunnel sont des bains thermiques, ce qui n'est pas nécessairement le cas pour notre échantillon.

Dans tous les cas, la régression en fonction de f que permet les mesures en bande large a l'avantage d'être effectuée à polarisation constante, contrairement à l'approche en bande étroite. Ainsi, comme les conditions expérimentales de la jonction sont exactement les mêmes pour toutes les données fournies au lissage,

4. Typiquement de quelques μV.
5. Et donc aussi plus résistives que les *gros* contacts typiquement utilisés.

les mesures en bande large permettent d'obtenir une valeur plus sûre de la température électronique. Ce fait est d'ailleurs mis à profit aux chapitres 11 et 12 pour calibrer l'amplitude de la photoexcitation et correctement comparer les résultats à la théorie. Il serait aussi intéressant d'étudier une jonction tunnel plus *typique* avec cette approche afin de vérifier si l'effet de chauffage est réellement négligeable lors de mesures en bande étroite [6].

10.2.2 Bruit en excès DC

Les résultats de la figure 10.6 permettent aisément d'obtenir le bruit en excès DC. Il suffit de soustraire la courbe à $V_{\mathrm{dc}} = 0$ de chaque courbe à $V_{\mathrm{dc}} \neq 0$ afin d'obtenir les spectres différentiels $\Delta\tilde{S}^{\mathrm{dc}}(f, V_{\mathrm{dc}})$; on obtient ensuite $\Delta S^{\mathrm{dc}}(\tau, V_{\mathrm{dc}})$ par transformée de Fourier.

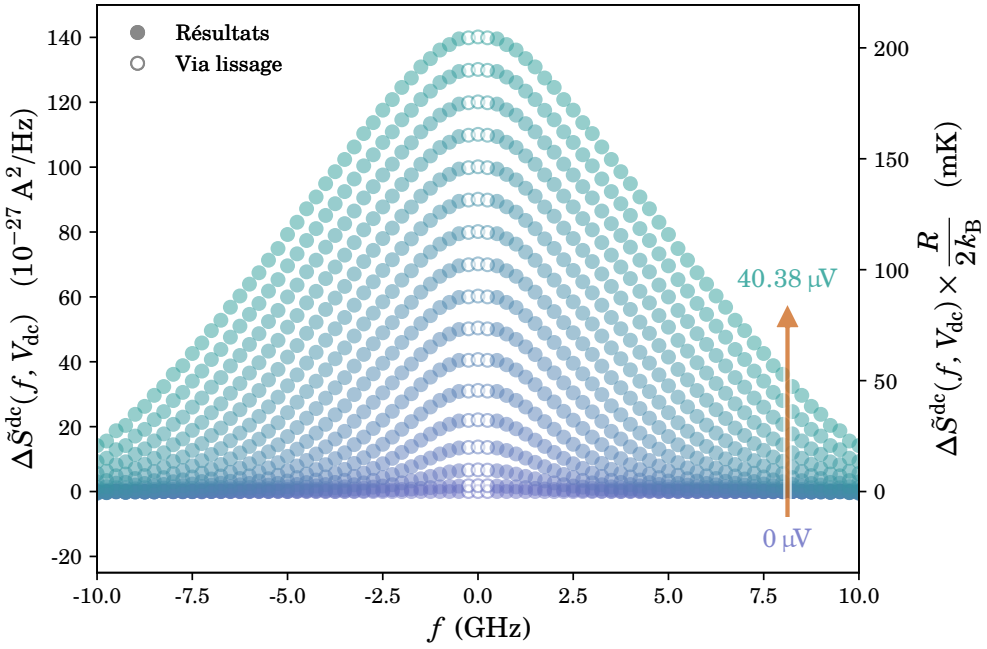


FIGURE 10.8 – Spectre de bruit en excès sous polarisation dc. Les points à $f \approx 0$ sont reconstruits à partir des lissages de la figure 11.7.

Cependant, puisque les fréquences $f \lesssim 250\,\mathrm{MHz}$ sont en dehors de la bande

6. Il pourrait d'ailleurs expliquer certains détails des résultats de [53], voir la section 10.2.3.

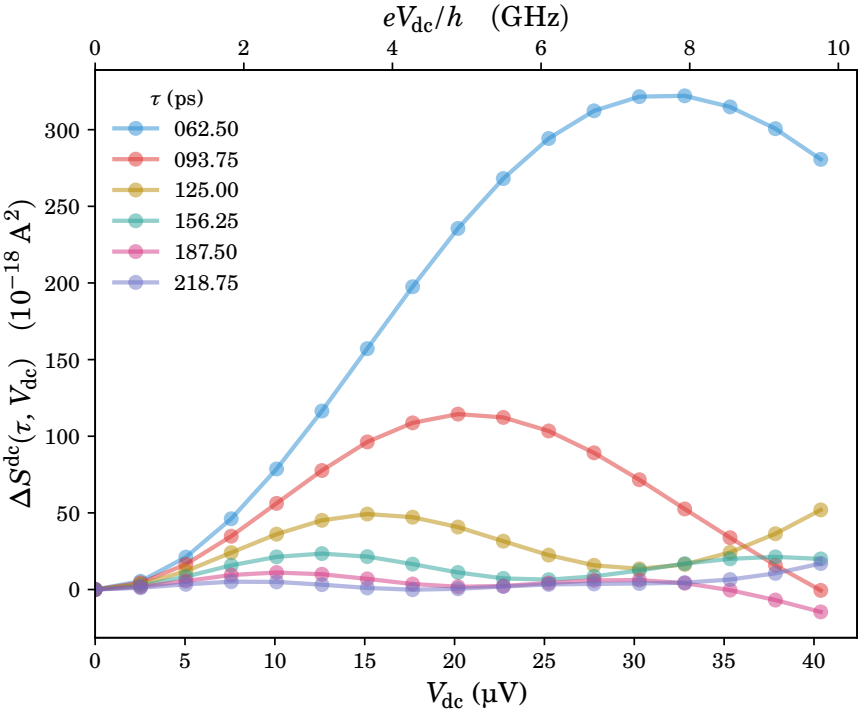


FIGURE 10.9 – Corrélateur courant-courant en excès sous polarisation dc.

passante — et puisque les basses fréquences ont un effet drastique [7] sur les résultats dans le domaine temporel — il importe de reconstruire les composantes basse fréquence avant d'effectuer la transformée de Fourier. On utilise donc les régressions à V_{dc} constant pour reconstruire ces données.

On présente les résultats de spectres différentiels $\Delta\tilde{S}_\phi(f, V_{dc})$ à la figure 10.8. On remarque que, bien que tous les résultats diminuent pour les grandes fréquences, la valeur de $\Delta\tilde{S}_\phi(f, V_{dc})$ n'atteint pas tout à fait 0 à $f = 10\,\text{GHz}$ pour les plus grandes polarisations V_{dc}. Ces courbes subiront donc un effet de fenêtrage dans le domaine temporel après transformée de Fourier, tel que discuté à la section 10.1.3.

Les résultats de $\Delta S^{dc}(\tau, V_{dc})$ en fonction [8] de V_{dc} pour quelques valeurs de τ

7. Du bruit hautes fréquences masquant moins les résultats que des oscillations globales lentes.

8. Les résultats à V_{dc} fixe en fonction de τ sont dominés par le terme $S_{eq}(\tau)$ qui décroît rapidement et sont donc visuellement moins intéressants.

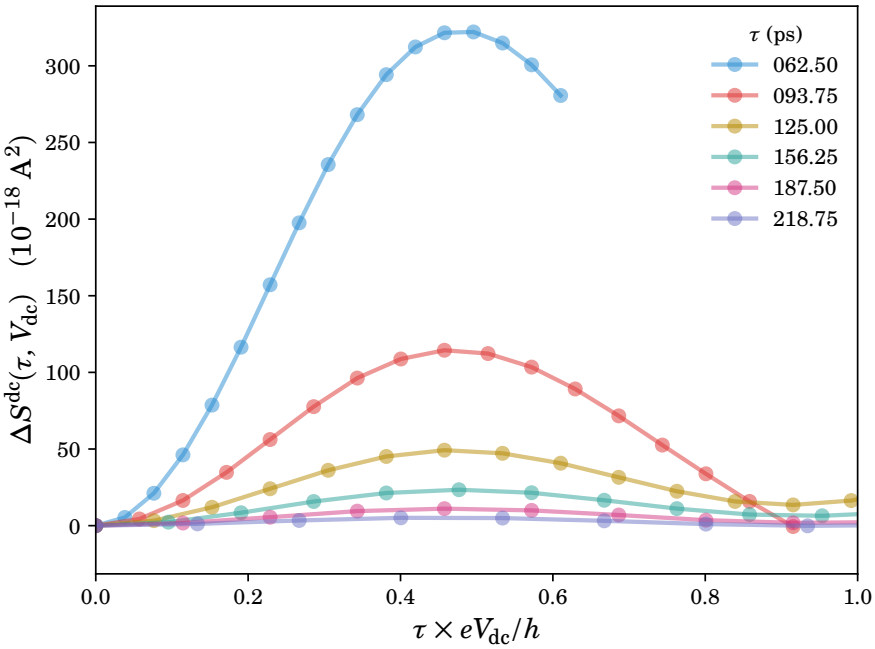


FIGURE 10.10 – Corrélateur courant-courant en excès sous polarisation dc mis à l'échelle.

sont montrés à la figure 10.9. Les oscillations en fonction de V_{dc} — prévues par la forme $\Delta S^{\mathrm{dc}}(\tau, V_{\mathrm{dc}}) = -2\,S_{\mathrm{eq}}(\tau)\sin^2\left(\frac{eV_{\mathrm{dc}}\tau}{2\hbar}\right)$ de l'équation (7.94) — sont évidentes. Effectivement, on observe bien une forme $\sin^2$ en V_{dc}, avec l'amplitude des oscillations qui décroît rapidement avec τ croissant ; en accord avec l'enveloppe $-2S_{\mathrm{eq}}(\tau)$ de l'équation (7.57) qui décroît en $\sim 1/\tau^2$ au premier ordre.

Pour mettre en évidence la périodicité des oscillations, on exprime l'abscisse selon la période attendue [9], soit $\tau \times eV_{\mathrm{dc}}/h$. Le résultat de cette opération correspond à la figure 10.10. Comme prévu, ainsi remises à l'échelle, les oscillations — qu'on peut qualifier d'oscillations quantiques étant donné que leur période dépend de la constante de Plank h [122] — oscillent toutes selon la même période. On remarque cependant que toutes les courbes ne se rejoignent pas parfaitement près de $\tau \times eV_{\mathrm{dc}}/h = 1$. On attribue cela à l'effet de fenêtrage apparaissant à grand V_{dc}, tel que discuté ci-haut, ainsi qu'à l'effet de chauffage. En effet, puisque T_{e} dépend de manière non négligeable de V_{dc} dans notre cas,

9. On rappelle que $\sin^2\theta$ a une période deux fois plus courte que $\sin\theta$.

les $\Delta S^{\mathrm{dc}}(f, V_{\mathrm{dc}})$ capturent non seulement l'effet de la polarisation, mais aussi celui de la différence de température entre les courbes. De plus, bien que les résultats de la figure 10.6 concordent très bien avec la théorie, de légères erreurs expérimentales peuvent fausser la reconstruction des données à basses fréquences.

Sommes toutes, les résultats démontrent bien la présence d'oscillations quantiques au sein de la partie des fluctuations engendrée par la polarisation en tension continue. À τ donné, l'application de V_{dc} vient donc moduler les corrélations à l'intérieur du bruit de grenaille de manière importante.

10.2.3 Oscillations de Pauli–Heisenberg

Il est aussi relativement simple de mettre en lumière les oscillations de Pauli–Heisenberg [53] à partir des résultats de la figure 10.6. L'approche est très similaire à celle de la section 10.2.2, à la différence qu'on soustrait une courbe à $T_{\mathrm{e}} = 0$ des résultats à $T_{\mathrm{e}} \neq 0$ en gardant V_{dc} constant, de manière à isoler la partie des fluctuations qui provient de l'agitation thermique.

Bien sûr, il est impossible d'obtenir des résultats à $T_{\mathrm{e}} = 0$; on soustrait donc la courbe théorique de $\tilde{S}^{\mathrm{dc}}(f, V_{\mathrm{dc}}, T_{\mathrm{e}} = 0)$ des résultats, comme ce qui est fait dans [53]. Cette courbe correspond simplement au bruit de grenaille $e|V_{\mathrm{dc}}|/R$ si $V_{\mathrm{dc}} > hf/e$ et au bruit du vide $h|f|/R$ autrement ; un exemple est présenté à la figure 10.6 en magenta. Comme dans le cas du bruit en excès DC, on reconstruit les données aux basses fréquences à l'aide des lissages de la figure 10.6.

Le résultat de cette soustraction est disponible à la figure 10.11. Pour chaque V_{dc}, on y voit clairement des pics à $f = \pm eV_{\mathrm{dc}}/h$, ceux-ci fusionnant à bas V_{dc} pour ne former qu'un pic d'amplitude double à $V_{\mathrm{dc}} = 0$. C'est le comportement attendu [10], puisque la contribution de la température se fait justement ressentir à $hf \approx eV_{\mathrm{dc}}$. On obtient alors les oscillations de Pauli–Heisenberg en prenant la transformée de Fourier de la figure 10.11, ce qu'on présente à la figure 10.12.

10. Autant théoriquement que *géométriquement* en regardant la soustraction effectuée à la figure 10.6.

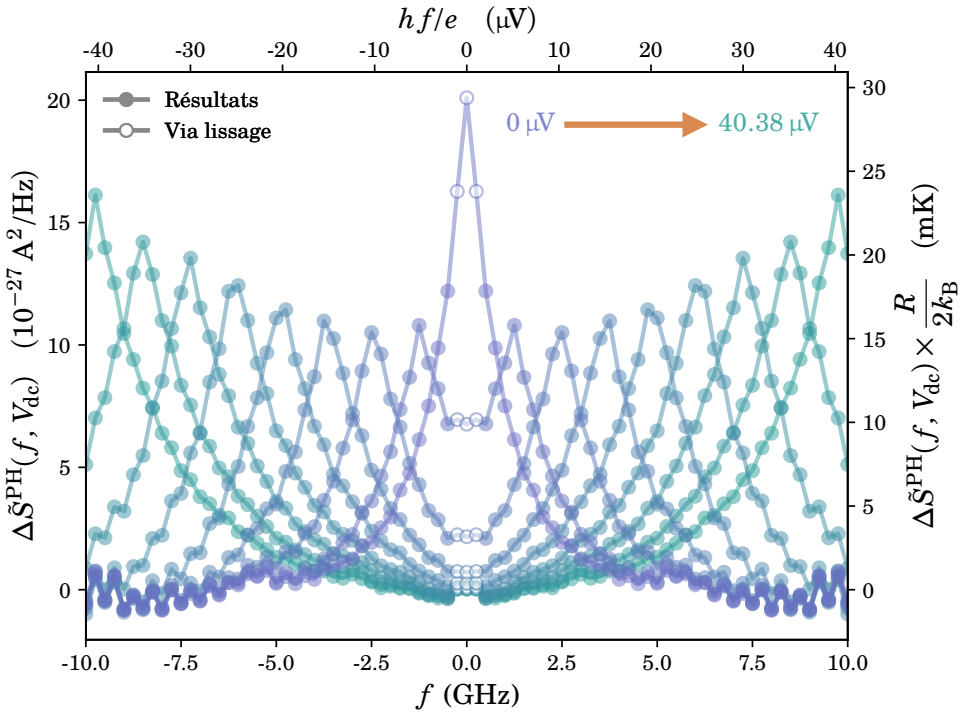


FIGURE 10.11 – Spectre de bruit en excès thermique.

Si on se concentre d'abord sur les données expérimentales de cette figure, on observe bien les oscillations attendues avec une enveloppe à $V_{\mathrm{dc}} = 0$ qui décroît avec τ. Cependant, on remarque que les données à $V_{\mathrm{dc}} \neq 0$ dépassent cette enveloppe à certaines valeurs de τ, en particulier à $\tau = 0$. On explique aisément cela par l'effet de chauffage dépendant de V_{dc} discuté à la section 10.2.1. En effet, comme discuté à la section 7.5.3, l'effet d'une température plus élevée sur les oscillations de Pauli–Heisenberg est d'augmenter leurs amplitudes à $\tau = 0$ et de faire décroître l'enveloppe plus rapidement. C'est exactement ce qu'on observe et ce qui permet aux courbes à plus grand V_{dc} de dépasser l'enveloppe à $V_{\mathrm{dc}} = 0$ pour les τ suffisamment petits. Les courbes théoriques tracées à la figure 10.12 utilisent le $T_{\mathrm{e}}(V_{\mathrm{dc}})$ de la figure 10.7 pour tenir compte de cet effet ; elles concordent de manière probante avec les résultats. Notons d'ailleurs que cet effet est aussi visible à la figure 10.11 par l'augmentation de l'amplitude des pics avec V_{dc}.

On remarque aussi que ce dépassement de l'enveloppe à $V_{\mathrm{dc}} = 0$ est observé

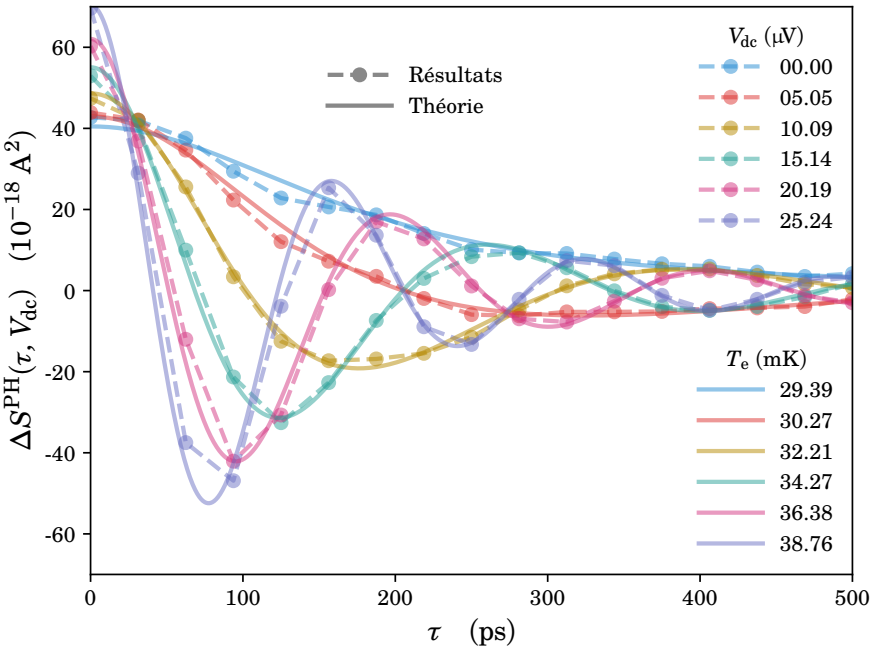


FIGURE 10.12 – Oscillations de Pauli–Heisenberg.

dans la référence [53, Fig. 4] bien qu'ils supposent la température constante en fonction de V_{dc}. Il est donc possible que l'effet de chauffage qu'on observe avec notre échantillon ne soit pas uniquement dû à ses *particularités*, mais soit aussi observable avec les jonctions plus communes. Cependant, c'est un effet qui reste probablement assez fin pour les jonctions tunnel typiques. S'il est visible au sein des oscillations de Pauli–Heisenberg, c'est probablement qu'elles traduisent spécifiquement les fluctuations liées à la température.

Similairement au bruit en excès DC, on peut remettre à l'échelle l'abscisse de la figure 10.12 pour mettre en évidence le temps caractéristique $\tau_e = h/(eV_{dc})$ auquel les électrons traversent la barrière par effet tunnel — voir la figure 10.13. Ce temps caractéristique n'est donc pas seulement lié à la différence entre le cas $V_{dc} \neq 0$ et $V_{dc} = 0$ dans le bruit en excès, mais représente l'effet de V_{dc} sur les évènements tunnels en général ; il s'applique aux fluctuations de source thermique à V_{dc} constant.

La figure 10.14 présente les mêmes résultats que la figure 10.12, mais pour plus de valeurs de valeurs de V_{dc} et en trois dimensions. Il est plus aisé d'y voir la

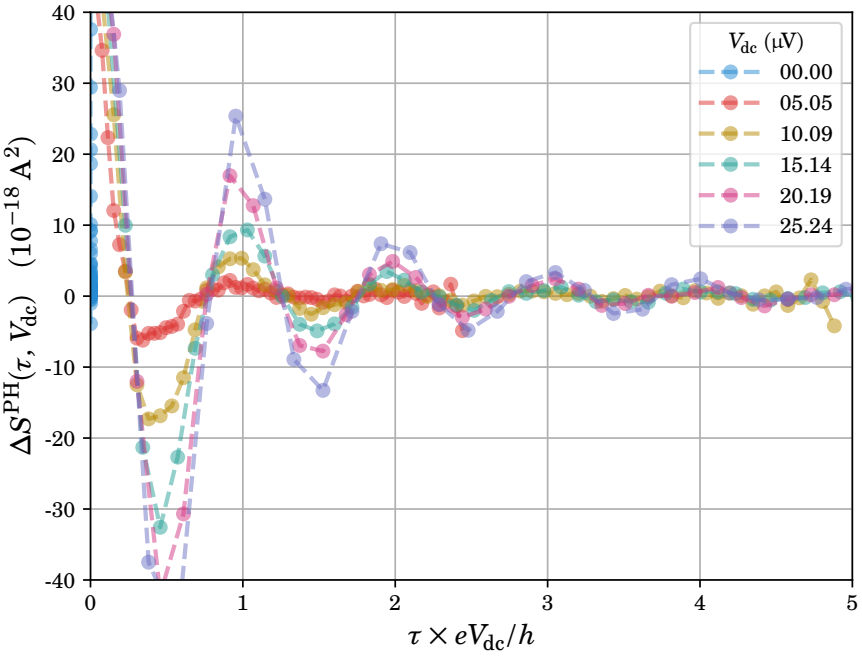


FIGURE 10.13 – Oscillations de Pauli–Heisenberg remise à l'échelle.

tendance des oscillations avec V_{dc} croissant. On y remarque aussi que la période de ces oscillations tend à devenir indépendante de V_{dc} à grande polarisation ; c'est simplement l'effet de fenêtrage causé par la coupure abrupte à $f = 10\,\mathrm{GHz}$ des spectres $\Delta\tilde{S}^{PH}(f)$ de la figure 10.11 qui leur donne l'allure d'un sinus cardinal.

Les mesures en bande large permettent donc de mesurer les oscillations de Pauli–Heisenberg aisément. Comme elles permettent aussi de connaître T_e pour chaque V_{dc}, elles expliquent quantitativement très bien les résultats même en présence d'effets de chauffage.

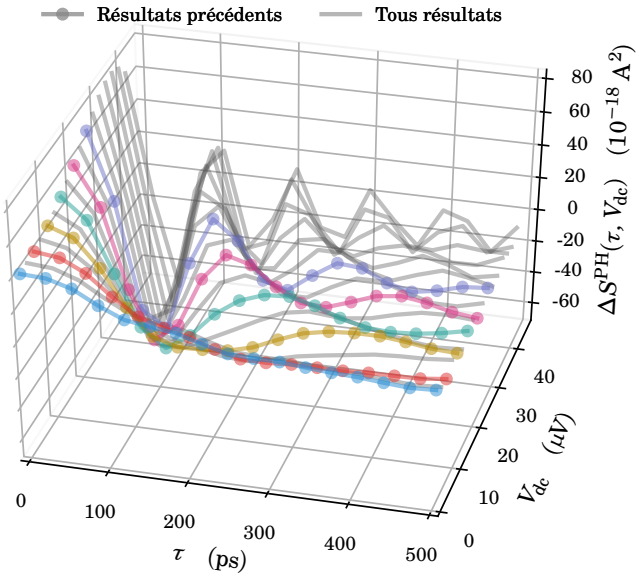


FIGURE 10.14 – Oscillations de Pauli–Heisenberg en 3D.

Chapitre 11

Calibration résolue en phase

11.1 Mesure résolue en phase et calibration

11.1.1 Modélisation de la mesure résolue en phase

Les mesures résolues en phase sont particulières en ce qui a trait à la modélisation de la mesure. Il ne suffit pas de modéliser le gain en puissance de l'amplificateur par un simple facteur multiplicatif réel pour chaque fréquence. En effet, l'équation (7.131) démontre que la mesure de $\tilde{S}_\phi(f)$ implique non seulement le signal à la fréquence $f = \omega/(2\pi)$, mais aussi ses composantes aux fréquences $-f + nf_0 \ \forall\, n \in \mathbb{Z}$. La modélisation présentée à la figure 10.1 est donc inadéquate dans le cas de mesures résolues en phase.

Plutôt que de considérer le gain en puissance de l'amplificateur, introduisons son gain en courant $\tilde{g}(f)$ qui s'applique directement sur le signal plutôt que sur la densité spectrale. Notons que puisqu'on mesure expérimentalement toujours un signal réel [1], on a nécessairement que $g(t) \in \mathbb{R}$ de sorte que $\tilde{g}^*(f) = \tilde{g}(-f)$. Ainsi, $\tilde{g}(f)$ peut être $\in \mathbb{C}$, mais doit être hermitien [76, §4.1.2.1]; sa partie complexe pouvant être assimilée à un délai temporel ou décalage en phase

1. Une série de valeurs de courants — ou de tensions associées à celui-ci — au cours du temps.

introduit à la fréquence f par l'amplificateur. De plus, comme l'amplificateur est stable au cours du temps et des mesures, on a l'identité $\langle \tilde{g}(f) \rangle = \tilde{g}(f)$.

On traite donc le gain de l'amplificateur en partant du signal $\tilde{\imath}_T(f)$ de l'équation (7.19) et en modélisant le signal mesuré post-amplification par

$$\tilde{\imath}_{T,\mathrm{m}}(f) = \tilde{g}(f)\,\tilde{\imath}_T(f)\,. \tag{11.1}$$

Si on regarde un seul des corrélateurs courant–courant dans le domaine fréquentiel de l'équation (7.131), soit $\langle \tilde{\imath}_T(-f + nf_0)\,\tilde{\imath}_T(f) \rangle$, et qu'on veut mesurer l'effet de l'amplification sur celui-ci, il suffit d'observer la conséquence de la substitution $\tilde{\imath}_M \longrightarrow \tilde{\imath}_{T,\mathrm{m}}$, soit

$$\left\langle \tilde{\imath}_{T,\mathrm{m}}(-f + nf_0)\,\tilde{\imath}_{T,\mathrm{m}}(f) \right\rangle = \tilde{\gamma}_n(f) \left\langle \tilde{\imath}_T(-f + nf_0)\,\tilde{\imath}_T(f) \right\rangle , \tag{11.2}$$

avec, par définition,

$$\boxed{\tilde{\gamma}_n(f) = \tilde{g}(-f + nf_0)\,\tilde{g}(f)}\,. \tag{11.3}$$

Ce facteur $\tilde{\gamma}_n(f)$ agit donc comme un gain complexe en puissance [2], de manière analogue au $\tilde{G}_A(f)$ du traitement non résolu en phase. En général, l'effet de la substitution $\tilde{\imath}_M \longrightarrow \tilde{\imath}_{T,\mathrm{m}}$ sur les $\tilde{\beta}_n(f)$ définis à l'équation (7.130) est donc

$$\tilde{\beta}_n(f) \;\longrightarrow\; \tilde{\gamma}_n(f)\,\tilde{\beta}_n(f)\,. \tag{11.4}$$

Ce dernier résultat, de pair avec la modélisation de la mesure non résolue en phase de l'équation (10.1) et l'identité (7.137), impose

$$\tilde{\gamma}_0(f) = \tilde{G}_A(f)\,. \tag{11.5}$$

On modélise alors la mesure en considérant $\tilde{S}_{\phi,\mathrm{m}}(f)$, le signal résolu en phase mesuré à la carte d'acquisition. Ce dernier est simplement le $\tilde{S}_\phi(f)$ de (7.131) soumis à la substitution $\tilde{\imath}_M \longrightarrow \tilde{\imath}_{T,\mathrm{m}}$ pour modéliser le gain de l'amplificateur et auquel on ajoute aussi un terme de de bruit d'amplification pour chaque

2. Avec le cas particulier $\tilde{\gamma}_0(f) = |\tilde{g}(f)|^2 \in \mathbb{R}$ puisque $\tilde{g}(f)$ est hermitien.

valeur de n, soit

$$\tilde{S}_{\phi,\mathrm{m}}(f) = \sum_{n=-\infty}^{\infty} \tilde{\gamma}_n(f)\left(\tilde{\beta}_n(f) + \tilde{\sigma}_n(f)\right) \mathrm{e}^{\mathrm{i}n\phi}, \tag{11.6}$$

où les termes $\tilde{\sigma}_n(f)$ représentent la contribution de bruit parasite du montage. Notons qu'à l'instar des $\tilde{\gamma}_n(f)$, on a que $\tilde{\sigma}_0 = \tilde{S}_A(f)$. Pour des amplificateurs idéaux, on s'attendrait normalement à ce que $\tilde{\sigma}_{n\neq 0} = 0$, puisque les propriétés des amplificateurs devraient être insensibles au signal d'entrée, le bruit qu'ils ajoutent ne devrait donc pas être modulé selon l'excitation. En pratique, un début de saturation ou de légères non-linéarités à n'importe quelle étape de l'acquisition peuvent causer ce comportement[3]. On s'attend donc à des valeurs de $\tilde{\sigma}_n(f)$ très faibles pour $|n| > 0$. Notons qu'on considère en général que $\tilde{\sigma}_n(f) \in \mathbb{C}$ bien qu'il soit tentant de le présumer réel, étant donné que la chaîne d'amplification n'est pas synchronisée avec la mesure ; cette présomption n'est cependant pas nécessaire au reste du traitement.

On profite aussi de la résolution en phase de la mesure, qui force $\tilde{S}_{\phi,\mathrm{m}}(f)$ à être périodique en ϕ, de sorte à pouvoir l'exprimer par la série de Fourier

$$\tilde{S}_{\phi,\mathrm{m}}(f) = \sum_{n=-\infty}^{\infty} \tilde{\beta}_{n,\mathrm{m}}(f)\, \mathrm{e}^{\mathrm{i}n\phi}, \tag{11.7}$$

où les $\tilde{\beta}_{n,\mathrm{m}}(f)$ sont les coefficients de Fourier du signal mesuré expérimentalement. On peut alors simplement égaler (11.6) et (11.7) pour trouver

$$\boxed{\tilde{\beta}_{n,\mathrm{m}}(f) = \tilde{\gamma}_n(f)\left(\tilde{\beta}_n(f) + \tilde{\sigma}_n(f)\right)}. \tag{11.8}$$

Le cas $n = 0$ correspond ainsi à la modélisation insensible à la phase de la section 10.1, ce qui est attendu puisque $\tilde{\beta}_0(f) = \hat{\tilde{S}}(f)$, $\tilde{\gamma}_0(f) = \tilde{G}_A(f)$ et $\tilde{\sigma}_0(f) = \tilde{S}_A(f)$, alors que les autres valeurs de n incarnent les subtilités supplémentaires liées à la résolution en phase. Partant des $\tilde{\beta}_{n,\mathrm{m}}(f)$ mesurés, il suffit donc de connaître les $\tilde{\gamma}_n(f)$ et $\tilde{\sigma}_n(f)$ pour obtenir les $\tilde{\beta}_n(f)$ du signal et ainsi reconstruire le signal

3. On peut par exemple imaginer qu'un amplificateur conçu pour fonctionner en large bande exhibe une légère non-linéarité lorsque le signal incident est concentré autour d'une seule fréquence.

d'intérêt. La modélisation de la mesure résolue en phase est présentée la figure 11.1.

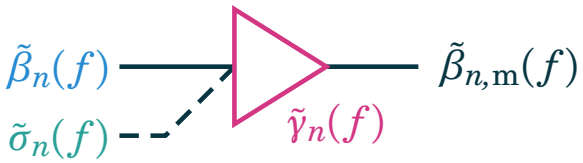


FIGURE 11.1 – Modèle de la mesure résolue en phase.

Bien qu'il puisse sembler difficile de mesurer tous les $\tilde{\gamma}_n(f)$ nécessaires à la calibration, la structure même de ceux-ci simplifie grandement le processus. En effet, il est possible d'exprimer $\tilde{\gamma}_n(f)$ pour un n donnée en fonction des deux $\tilde{\gamma}_n(f)$ de valeurs de n inférieures ou supérieures à la sienne. Par exemple si on connaît $\tilde{\gamma}_0(f) = \tilde{g}(-f)\,\tilde{g}(f)$ et $\tilde{\gamma}_1(f) = \tilde{g}(-f+f_0)\,\tilde{g}(f)$ et qu'on chercher à calculer $\tilde{\gamma}_2(f) = \tilde{g}(-f+2f_0)\,\tilde{g}(f)$, on peut simplement multiplier ce dernier par un facteur 1 judicieux pour le réexprimer

$$\tilde{\gamma}_2(f) = \tilde{g}(-f+2f_0)\,\tilde{g}(f) \tag{11.9}$$

$$= \frac{\overbrace{\tilde{g}(-f+2f_0)\,\tilde{g}(f-f_0)}^{\tilde{\gamma}_1(f-f_0)}\ \overbrace{\tilde{g}(-f+f_0)\,\tilde{g}(f)}^{\tilde{\gamma}_1(f)}}{\underbrace{\tilde{g}(f-f_0)\,\tilde{g}(-f+f_0)}_{\tilde{\gamma}_0(f-f_0)}} \tag{11.10}$$

$$= \frac{\tilde{\gamma}_1(f-f_0)\,\tilde{\gamma}_1(f)}{\tilde{\gamma}_0(f-f_0)}. \tag{11.11}$$

Cette même approche permet de trouver la forme générale [4]

$$\boxed{\tilde{\gamma}_n(f) = \frac{\tilde{\gamma}_{n\pm1}(f\pm f_0)\,\tilde{\gamma}_{n\pm1}(f)}{\tilde{\gamma}_{n\pm2}(f\pm f_0)}}. \tag{11.12}$$

Il est donc seulement nécessaire de connaître deux $\tilde{\gamma}_n(f)$ de n adjacents afin de pouvoir tous les calculer.

4. On utilise la convention selon laquelle le choix du signe est le même pour tous les $\pm$. L'équation (11.12) correspond donc à deux équations distinctes, et non pas trente-deux.

11.1.2 Calibration via limite à grand V_{dc}

Similairement à la calibration sans résolution en phase, c'est le régime à grande polarisation en tension continue qui permet d'extraire les $\tilde{\gamma}_n(f)$ requis pour la calibration. On rappelle le résultat de $\tilde{S}_{\phi,\mathrm{m}}$ à grand V_{dc} positif[5] de l'équation (7.168), soit $\tilde{S}_\phi^{\mathrm{pa}}(\hbar\nu \gg |\mathcal{E}|) = \frac{e}{R}\big(V_{\mathrm{dc}} + V_{\mathrm{ac}}\cos\phi\big)$ qu'on peut exprimer sous différentes formes, soient

$$\tilde{S}_\phi^{\mathrm{pa}}(eV_{\mathrm{dc}} \gg |\mathcal{E}|) = \frac{e}{R}\big(V_{\mathrm{dc}} + V_{\mathrm{ac}}\cos\phi\big) \tag{11.13}$$

$$= \frac{eV_{\mathrm{ac}}}{2R}\,\mathrm{e}^{-\mathrm{i}\phi} + \frac{e}{R}V_{\mathrm{dc}} + \frac{eV_{\mathrm{ac}}}{2R}\,\mathrm{e}^{\mathrm{i}\phi} \tag{11.14}$$

$$= \tilde{\beta}_{-1}(f)\,\mathrm{e}^{-\mathrm{i}\phi} + \tilde{\beta}_0(f) + \tilde{\beta}_1(f)\,\mathrm{e}^{\mathrm{i}\phi}, \tag{11.15}$$

pour remarquer que, dans ce régime,

$$\tilde{\beta}_0(f) = \frac{e}{R}V_{\mathrm{dc}} \tag{11.16}$$

$$\tilde{\beta}_{\pm1}(f) = \frac{e}{R}\frac{V_{\mathrm{ac}}}{2} \tag{11.17}$$

$$\tilde{\beta}_{|n|\geq 2}(f) = 0. \tag{11.18}$$

En combinant ces équations avec (11.8), on trouve

$$\tilde{\beta}_{0,\mathrm{m}}(f) = \tilde{G}_A(f)\left(\frac{e}{R}V_{\mathrm{dc}} + \tilde{S}_A(f)\right) \tag{11.19}$$

$$\tilde{\beta}_{\pm1,\mathrm{m}}(f) = \tilde{\gamma}_{\pm1}(f)\left(\frac{e}{R}\frac{V_{\mathrm{ac}}}{2} + \tilde{\sigma}_{\pm1}(f)\right) \tag{11.20}$$

$$\tilde{\beta}_{|n|\geq 2,\mathrm{m}}(f) = \tilde{\gamma}_n(f)\,\tilde{\sigma}_n(f). \tag{11.21}$$

Ainsi, à grand $V_{\mathrm{dc}} > 0$, on peut utiliser le $\tilde{\beta}_{0,\mathrm{m}}(f)$ mesuré pour obtenir $\tilde{\gamma}_0(f) = \tilde{G}_A(f)$ et $\tilde{S}_A(f)$ comme dans la situation habituelle. On peut aussi se mettre dans la situation $V_{\mathrm{ac}} = 0$ et extraire $\tilde{\gamma}_{\pm1}(f)\,\tilde{\sigma}_{\pm1}(f)$ de (11.20), ce qui permet ensuite d'effectuer une régression linéaire en V_{ac} de la forme

$$\left|\tilde{\beta}_{\pm1,\mathrm{m}}(f) - \tilde{\gamma}_{\pm1}(f)\,\tilde{\sigma}_{\pm1}(f)\right| = |\tilde{\gamma}_{\pm1}(f)|\,\frac{e}{R}\frac{V_{\mathrm{ac}}}{2} \tag{11.22}$$

5. On se concentre ici sur le cas $V_{\mathrm{dc}} > 0$, mais la situation $V_{\mathrm{dc}} < 0$ est aussi valide.

et d'en extraire $|\tilde{\gamma}_{\pm 1}(f)|$ de la pente. On utilise alors le fait que $\frac{eV_{\text{ac}}}{2R} \in \mathbb{R}$, si bien que

$$\text{Arg}\left[\tilde{\beta}_{\pm 1,\text{m}}(f, V_{\text{ac}}) - \tilde{\gamma}_{\pm 1}(f)\,\tilde{\sigma}_{\pm 1}(f)\right] = \text{Arg}\left[\tilde{\gamma}_{\pm 1}(f)\right], \tag{11.23}$$

pour obtenir la partie imaginaire des $\tilde{\gamma}_{\pm 1}(f)$. De là, tous les $\tilde{\gamma}_n(f)$ peuvent être calculés via (11.12). Finalement, l'équation (11.21) donne directement les termes constants des $\tilde{\beta}_{|n|\geq 2,\text{m}}(f)$. On a donc finalement accès à tous les $\tilde{\beta}_n(f)$ — et du même coup à la densité spectrale intrinsèque $\tilde{S}_\phi(f)$ résolue en phase — en inversant (11.8), c'est-à-dire

$$\tilde{\beta}_n(f) = \frac{\tilde{\beta}_{n,\text{m}}(f) - \tilde{\gamma}_n(f)\,\tilde{\sigma}_n(f)}{\tilde{\gamma}_n(f)}. \tag{11.24}$$

Il y a cependant un bémol sur ce dernier point. Bien qu'on puisse extraire les $\tilde{\gamma}_n(f)$ et $\tilde{\sigma}_n(f)$ via la procédure étalée ci-haut, la bande passante de l'amplificateur vient poser certaines limitations sur les fréquences pour lesquelles la reconstruction de $\tilde{S}_\phi(f)$ sera valide. En effet, les gains complexes en courant $\tilde{g}(f)$ approchent 0 en dehors de la bande passante de l'amplificateur et le mélange des $\tilde{g}(f)$ associés à des fréquences différentes dans les $\tilde{\gamma}_n(f)$ — mis en évidence par (11.3) — force les $\tilde{\gamma}_n(f)$ à être nuls pour certaines fréquences. Par exemple, $\tilde{\gamma}_1(f_0) = g(0)g(f) = 0$ parce que $f = 0$ est en dehors de la bande passante. Similairement, si on note f_{crit} la fréquence limite de la bande passante tel que $\tilde{g}(|f| \geq f_{\text{crit}}) = 0$, on aura $\tilde{\gamma}_{-1}(f_{\text{crit}} - f_0) = \tilde{g}(-f_{\text{crit}})\,\tilde{g}(f_{\text{crit}} - f_0) = 0$.

Généralement, la procédure décrite ici permet de calibrer l'effet de l'amplificateur sur la mesure, mais des effets incontournables liés à la bande passante de la mesure empêchent d'obtenir les $\tilde{\gamma}_n(f)$ pour certaines fréquences. Les multiples entiers de la fréquence d'excitation f_0 ainsi que les fréquences à moins de f_0 de la limite supérieure de la bande passante ne seront pas correctement reconstruites — puisque la mesure n'y est pas sensible.

Cette limitation n'est que l'effet de la bande passante finie de la mesure. Ce qui est inattendu est que cette dernière soit ressentie même pour des fréquences au sein de la bande passante. Concrètement, pour les conditions expérimentales étudiées ici, soit $f_0 = 4\,\text{GHz}$ et une bande passante de $\approx 250\,\text{MHz}$—$10\,\text{GHz}$, on

devrait obtenir une mesure calibrée fiable entre $\approx 250\,\mathrm{MHz}$ et $6\,\mathrm{GHz}$, hormis une plague d'environ $500\,\mathrm{MHz}$ centrée autour de $4\,\mathrm{GHz}$.

11.2 Calibration des données résolues en phase

On présente ici les étapes et résultats du processus de calibration pour les résultats résolus en phase du chapitre 12. Comme conditions expérimentales, on a la fréquence de photoexcitation $f_0 = 4\,\mathrm{GHz}$, la fréquence d'échantillonnage $f_e = 32\,\mathrm{GHz}$ et une bande passante analogique d'environ $10\,\mathrm{GHz}$. Les données ont été acquises avec le script de la section C.2.2 et consistent en des autocorrélations résolues en phase ($f_e/f_0 = 8$ phases) pour trois valeurs de photoexcitation, incluant un cas sans photoexcitation, spécifiquement $-\infty$, -20 et $-10\,\mathrm{dBm}$. Pour chacune de celles-ci, on balaye la polarisation en tension continue sur 19 valeurs ; respectivement 9 valeurs positives et négatives de manière symétrique, ainsi que la polarisation nulle. Parmi ces 19 tensions, 3 polarisations négatives et trois positives sont prises à $e\,|V_{\mathrm{dc}}|\,/h \gg 10\,\mathrm{GHz}$; les 13 autres valeurs correspondant à $e\,|V_{\mathrm{dc}}|\,/h < 10\,\mathrm{GHz}$. Afin de minimiser les imperfections potentiellement engendrées par la présence de plusieurs convertisseurs analogique–numériques au sein de la carte d'acquisition, on fait les mesures à 8 phases de référence séparées de $45°$; celles-ci sont recalées et combinées avant l'analyse, ce qui devrait moyenner l'effet des convertisseurs. On utilise aussi la propriété (7.160), qui nous permet de moyenner $\tilde{S}^{\mathrm{pa}}_{\phi,\mathrm{m}}(f, -V_{\mathrm{dc}})$ et $\tilde{S}^{\mathrm{pa}}_{\phi+\pi,\mathrm{m}}(f, V_{\mathrm{dc}})$ pour effectivement doubler la quantité de données sur lesquelles les résultats sont moyennés et éliminer des imperfections expérimentales potentielles. Les données brutes d'autocorrélation correspondent au traitement d'environ 323 téraoctets de données acquises à la carte d'acquisition.

Un exemple représentatif des données brutes d'autocorrélation résolues en phase $S_{\phi,\mathrm{m}}(\tau)$ est montré à la figure 11.2 pour des V_{dc} et V_{ac} non nuls. Les données à $\tau < 0$ on été reconstruites grâce à la propriété (7.159). En haut, on y voit les autocorrélations résolues en phase obtenues pour $\phi = 0°$ et $90°$. Contrairement au cas sans résolution en phase — et bien que ça ne soit pas évident visuellement — $S_{\phi,\mathrm{m}}(\tau)$ n'est pas symétrique. La sous-figure du bas

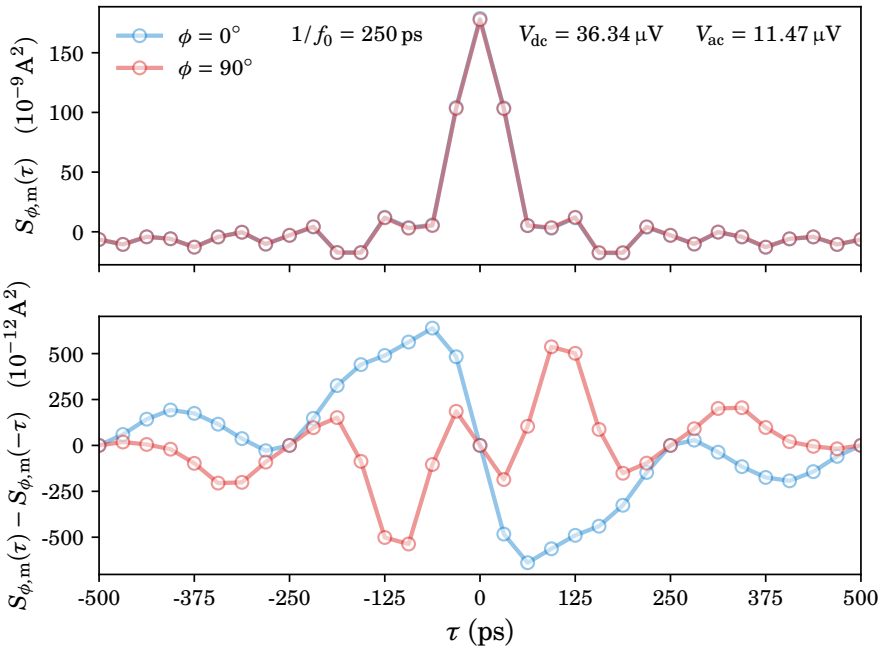


FIGURE 11.2 – Données brutes d'autocorrélations résolues en phase. La sous-figure du bas est la partie antisymétrique de celle du haut. On mesure $S_\phi(\tau)$ jusqu'à $\tau \approx 2\mathrm{ns}$; on ne montre que les temps courts par souci de clarté.

met ce résultat en évidence en présentant la partie antisymétrique de $S_{\phi,\mathrm{m}}(\tau)$. Celle-ci est quelques ordres de grandeur plus faible que la partie symétrique des autocorrélations, probablement puisque cette dernière est dominée par la contribution du bruit d'entrée de la chaîne d'amplification, qui ne devrait pas être sensible à la phase.

Aux fins d'analyse et de traitement des données, les spectres résolus en phase brutes $\tilde{S}_{\phi,\mathrm{m}}(f)$ s'obtiennent alors simplement par transformée de Fourier en τ, et les $\tilde{\beta}_{n,\mathrm{m}}(f)$ par série de Fourier subséquente en ϕ — voir les équations (7.113) et (7.115).

11.2.1 Calibration de V_{ac}

Lorsqu'on applique la photoexcitation via la source RF — voir le montage de la figure 8.1 pour les détails — on connaît très bien le V'_{ac} à la sortie de la

source. Il est cependant plus difficile de connaître l'amplitude $V_{\text{ac}} = \alpha V'_{\text{ac}}$ du signal effectivement appliqué aux bornes de la jonction tunnel, puisque le facteur α n'est pas connu à l'avance — ce dernier représentant à la fois l'atténuation appliquée volontairement sur les lignes d'excitation pour protéger l'échantillon du bruit thermique et la diminution de l'amplitude du signal due aux désaccords d'impédances et imperfections expérimentales. Or, avant de pouvoir obtenir une valeur probante de $\tilde{\gamma}_{\pm 1}(f)$ par régression linéaire en V_{ac} de la forme (11.22) tel que discuté précédemment, il faut d'abord connaître le véritable V_{ac} appliqué à l'échantillon.

Pour calibrer la tension V_{ac}, c'est-à-dire trouver α, on se concentre sur la partie non-résolue en phase de la mesure. On commence donc par effectuer la calibration régulière sur $\tilde{\beta}_{0,\text{m}}$ pour les données à $V_{\text{ac}} = V'_{\text{ac}} = 0$, tel que décrit à la section 10.1, de manière à obtenir $\tilde{G}_A(f) = \tilde{\gamma}_0(f)$ et $\tilde{S}_A(f) = \tilde{\sigma}_0(f)$. On en extrait ensuite les valeurs de $\tilde{\beta}_0 = \tilde{S}_{\langle\phi\rangle}$ intrinsèques[6] pour l'ensemble des combinaisons de V_{dc} et V'_{ac} appliquées à la jonction. Dans l'optique d'extraire α, on s'intéresse alors à la partie des données avec $V_{\text{ac}} \neq 0$.

L'approche évidente pour extraire α de mesures de $\tilde{S}_{\langle\phi\rangle}$ en bande étroite à f fixe et $V_{\text{ac}} \neq 0$ à partir des données serait alors d'effectuer une régression de la forme[7] théorique de $\hat{\tilde{S}}^{\text{pa}}(f, V_{\text{dc}}, \alpha V'_{\text{ac}}, T_{\text{e}}, f_0)$ en fonction de V_{dc} sur les données, avec α et T_{e} comme seuls paramètres libres. Cette approche n'est malheureusement pas appropriée dans notre situation. D'abord, comme discuté à la section 10.2 — spécifiquement à la figure 10.7 — le chauffage non-négligeable induit par V_{dc} rend les lissages en fonction de V_{dc} impropres à la situation. L'idéal en bande large est plutôt de faire les lissages en fonction de f à V_{dc} et V_{ac} constants. Ensuite, les mesures photoexcitées en bande étroite sont typiquement effectuées à des ratio f_0/f de 1 ou 2, ce qui optimise la visibilité des répliques modulées par les fonctions de Bessel. Le corollaire est donc que leur visibilité sera variable en fonction de f pour les mesures en bande large ; motivant d'autant plus le choix de lissage en fonction de f plutôt que de V_{dc}. Cependant, si on regarde $\hat{\tilde{S}}^{\text{pa}}(f)$ dans le domaine fréquentiel, on remarque que les répliques

6. Il est intéressant de noter que le $\tilde{S}_{\langle\phi\rangle}$ ainsi obtenu est insensible à la partie déterministe du signal — même dans le cas des autocorrélations résolues en phase calculées avant le `remdet` — grâce aux soustractions de moyennes partielles dans la définition du $a_{\varphi,k}$ à l'équation (9.20).

7. Soit l'équation (7.87) ayant toutes ses dépendances explicitées.

sont rapidement masquées par une température électronique $T_e > 0$. L'effet de V_{ac} est alors similaire à celui de la température et les algorithmes de lissage par moindres carrés peinent à obtenir des résultats fiables et cohérents d'une combinaison de V'_{ac} et V_{dc} à l'autre.

En effet, bien que l'on obtienne des résultats de lissage qui *semblent* adéquats pour chaque courbe à V_{dc} et V'_{ac} fixes, les valeurs de T_e et α ainsi obtenues varient de manière erratique en fonction des paramètres de polarisation. Par exemple, pour un V'_{ac} donné, les différentes valeurs de V_{dc} donnent des estimations différentes de α, et donc de V_{ac}. En plus, les températures électroniques T_e ainsi estimées pour des V_{dc} faibles sont parfois plus grandes que celles associées aux plus grandes valeurs de V_{dc} ; un comportement contraire à celui attendu d'après les résultats de la figure 10.7. Il semble que l'algorithme de régression arrive à trouver un compromis entre α et T_e pour chaque combinaison de V_{dc} et V'_{ac}, mais que celui-ci ne soit pas le même pour les différentes conditions ; l'imprévisible choix de compromis de l'algorithme cause alors les résultats aberrants décrits ci-haut.

Pour pallier à cette problématique, il faut ajouter des contraintes sur la régression de manière à ce que le compromis atteint par l'algorithme de lissage soit commun à toutes les courbes. L'idée étant essentiellement d'ajouter des contraintes valides à l'optimisation, de manière à restreindre les résultats possibles aux résultats qui ont un sens physique. Ainsi, plutôt que d'effectuer 26 lissages indépendants, chacun avec son propre T_e et α, on effectue un lissage commun sur les 26 courbes à la fois en partageant une seule valeur de α parmi celles-ci, tout en gardant T_e indépendant pour chacune. On passe ainsi de 52 paramètres libres totaux à seulement 27, et le compromis entre α et T_e est optimisé sur l'ensemble des résultats plutôt que pour chaque courbe indépendamment. Puisqu'un équilibre entre α et T_e qui serait jugé optimal pour un choix de V_{dc} et de V'_{ac} donné peut être très défavorable pour une autre combinaison, cette approche devrait permettre d'obtenir un lissage crédible et des paramètres optimisés valides. Concrètement, le α partagé force les différentes valeurs de V_{ac} à être les mêmes pour une puissance de sortie donnée et à respecter le ratio de puissance imposé par celles-ci, soit un ratio de $\sqrt{10}$ en amplitude de tension pour les signaux de sortie de -20 et $-10\,\mathrm{dBm}$.

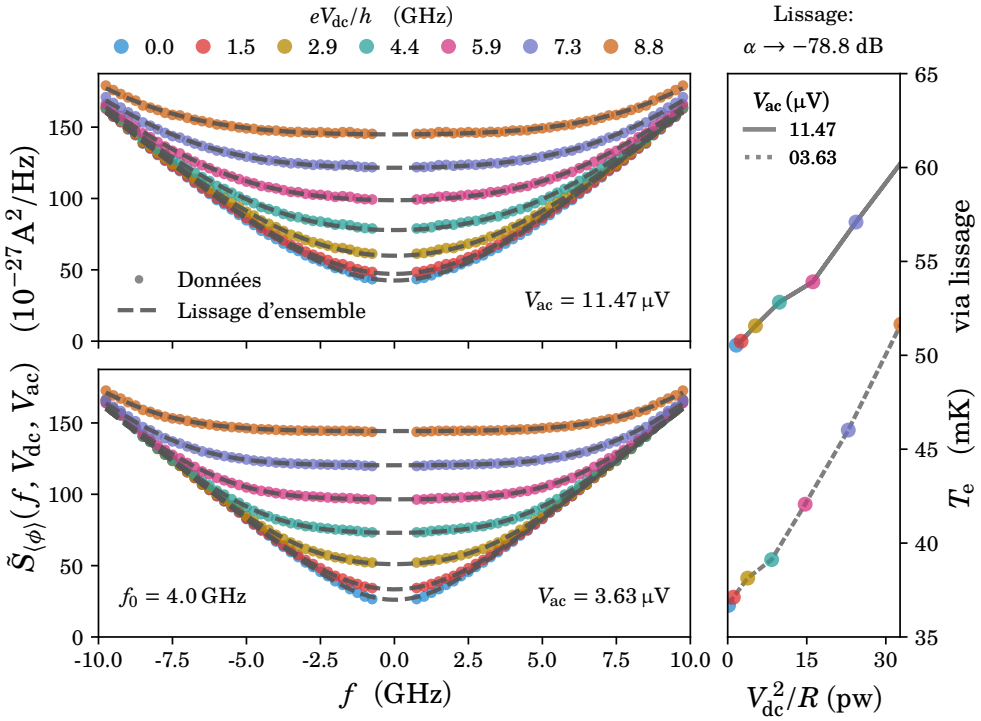


FIGURE 11.3 – Lissage d'ensemble des données à $V'_{ac} \neq 0$ et paramètres obtenus. Les couleurs sont cohérentes entre les trois sous-figures et correspondent à V_{dc}. Les valeurs de V_{ac} présentées sont déduites des V'_{ac} empirique et du résultat de lissage $\alpha \approx 114.7 \times 10^{-6}$. Seules les données utilisées pour la régression sont montrées.

La figure 11.3 présente le résumé de ce lissage commun et des paramètres qui en sont extraits [8]. L'accord entre les courbes de $\tilde{S}_{\langle\phi\rangle}(f, V_{dc}, V_{ac})$ expérimentales et le résultat du lissage est frappant. On obtient $\alpha \approx 114.7 \times 10^{-6}$, correspondant à -78.8 dB, une valeur crédible considérant l'atténuation appliquée sur les lignes d'excitation du cryostat pour protéger ses étages froids du bruit thermique des étages plus chauds. On a donc $V_{ac} = 3.63\,\mu V$ et $V_{ac} = 11.47\,\mu V$ pour les courbes $\tilde{S}_{\langle\phi\rangle}(f, V_{dc}, V_{ac})$ du haut et du bas, respectivement. Du côté des valeurs de T_e, on trouve, pour chacune des valeurs de photoexcitation, une tendance linéaire avec la puissance de biais en tension continue, similaire à celle observée à la figure 10.7 pour le même régime $T_e > 35$ mK. De plus, la courbe à plus grand V_{ac} a une température électronique systématiquement plus chaude que celle à

8. Le code personnalisé utilisé pour ce lissage particulier est disponible à l'annexe D.3.

plus petite excitation, ce qui est attendu.

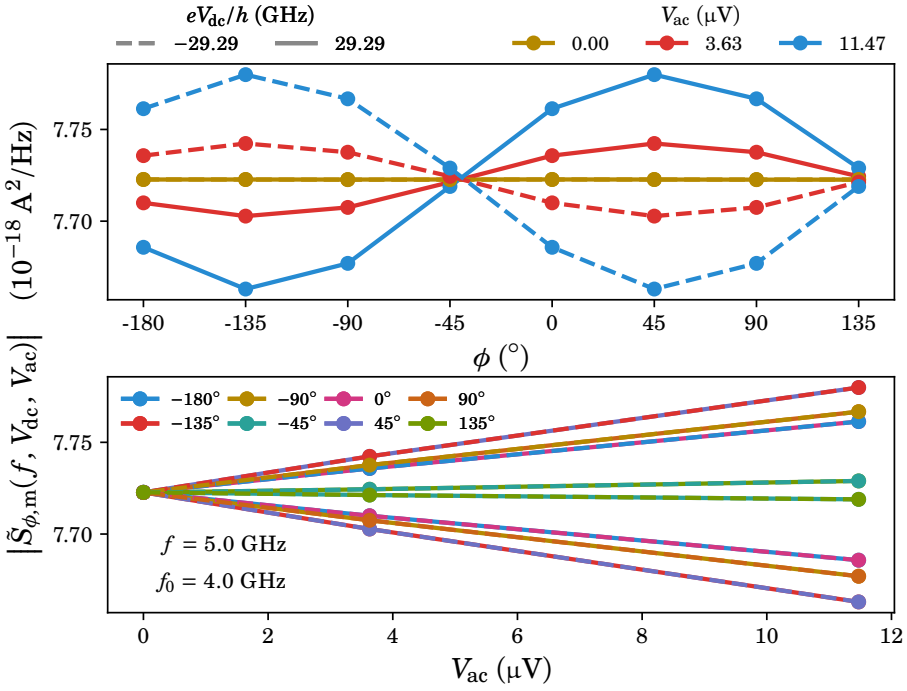


FIGURE 11.4 – $\tilde{S}_{\phi,\mathrm{m}}$ à haute polarisation V_{dc}, utilisée pour la calibration à $f = 5\,\mathrm{GHz}$.

11.2.2 Extraction des $\tilde{\gamma}_n(f)$

Connaissant maintenant les valeurs réelles de V_{ac} appliquées à l'échantillon, on poursuit la calibration en s'intéressant à $\tilde{S}_{\phi,\mathrm{m}}$ dans le régime $eV_{\mathrm{dc}}/h \gg 10\,\mathrm{GHz}$. La figure 11.4 montre un exemple de données brutes utilisées pour la calibration résolue en phase pour $f = 5\,\mathrm{GHz}$. On ne montre que la norme des données puisque, à fréquence fixe, l'argument complexe provient uniquement des $\tilde{\gamma}_n(f)$ et est donc constant. Sur l'axe du haut, on observe bien la forme sinusoïdale attendue de (7.168) en fonction de ϕ, avec une seule phase initiale ϕ_0 indépendante[9] de V_{ac}, et les courbes avec des V_{dc} de signes opposés sont décalées de 180° comme prévu par (7.160). L'axe du bas montre, quant à lui,

9. À priori variable selon f, à travers les $\tilde{\gamma}_n(f)$.

que l'amplitude des sinus est bel et bien proportionnelle à V_{ac}, confirmant que le régime de calibration est atteint.

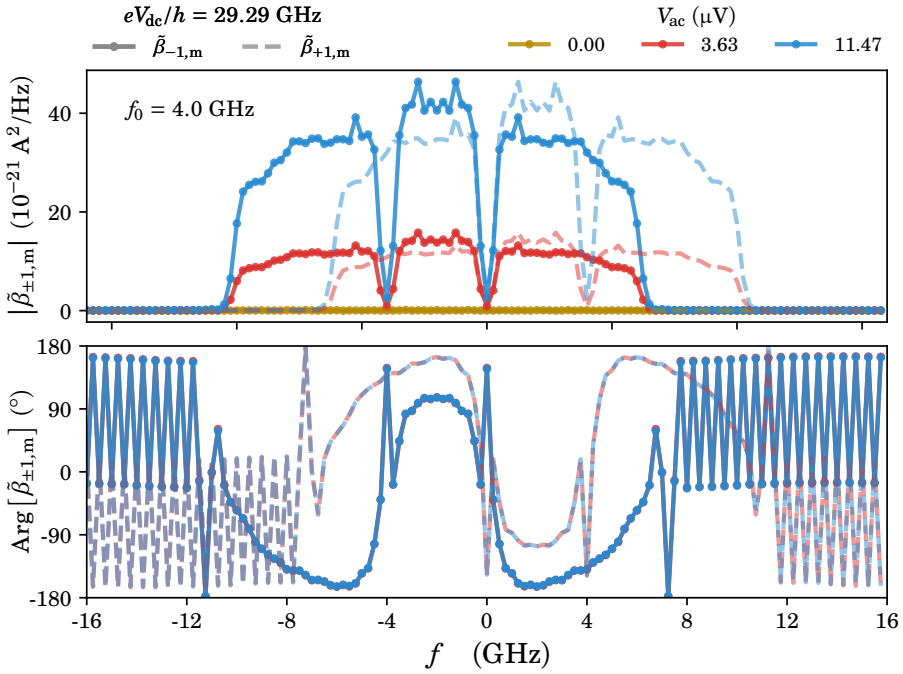


FIGURE 11.5 – Calibration résolue en phase, $\tilde{\beta}_{\pm1,\text{m}}(f)$. Cas $V_{\text{dc}} < 0$ montré ; changer le signe de V_{dc} ne fait que décaler l'argument/ de 180°.

On se concentre alors sur les coefficients de Fourier $\tilde{\beta}_{\pm1,\text{m}}$ — essentiellement l'amplitude des oscillations de la figure 11.4 — qu'on présente à la figure 11.5, avec l'amplitude et l'argument complexe sur les axes du haut et du bas, respectivement. On y remarque que $\tilde{\beta}^*_{1,\text{m}}(f) \approx \tilde{\beta}_{-1,\text{m}}(-f)$, en accord avec l'hermiticité des $\tilde{g}(f)$ et les équations [10] (11.3) et (11.20). Notablement, l'argument complexe est invariant selon V_{ac} comme prévu [10] ; on utilise donc ceux présentés ici pour reconstruire les $\tilde{\gamma}_{\pm1}$ complexes.

La figure 11.6 résume l'extraction des $\tilde{\gamma}_n(f)$ à l'aide de lissages de la forme (11.20) sur les $\tilde{\beta}_{\pm1,\text{m}}(f)$ pour chaque f résolue expérimentalement, suivi de la reconstruction des $\tilde{\gamma}_n(f)$ par (11.12). L'axe du haut présente le même $|\tilde{\beta}_{-1,\text{m}}|$ qu'à la figure 11.5, cette fois en fonction de V_{ac} et pour quatre valeurs représentatives de f, en plus des droites lissées sur les données à chaque fréquence. L'axe

10. Considérant que $|\tilde{\sigma}_{n\neq0}(f)| \approx 0$ même advenant le cas où $\tilde{\sigma}_{n\neq0}(f) \notin \mathbb{R}$.

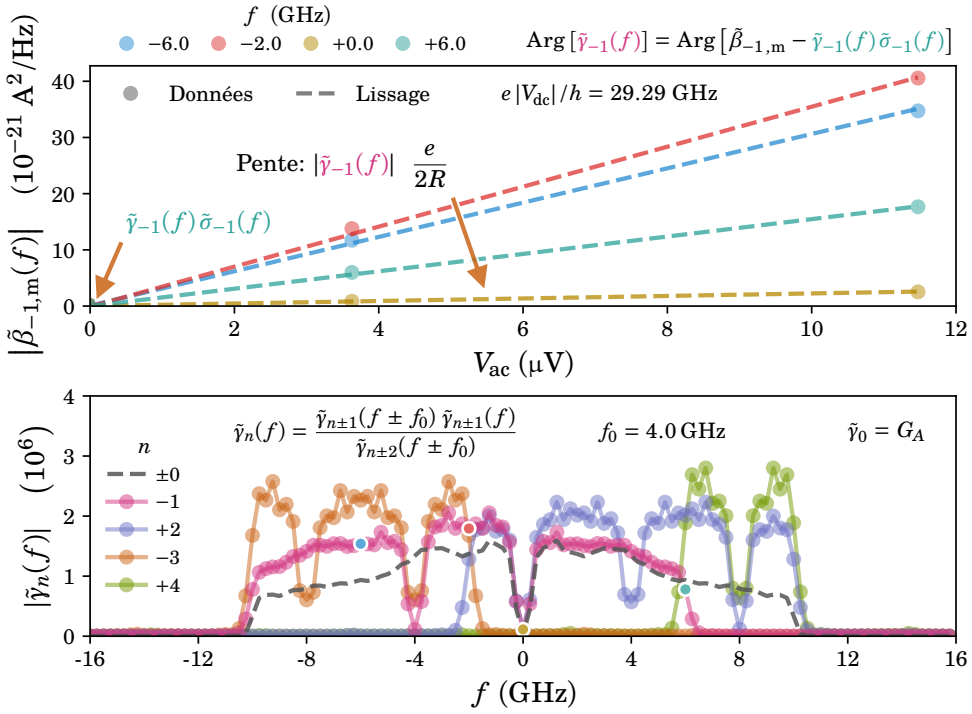


FIGURE 11.6 – Extraction des $|\tilde{\gamma}_n(f)|$. L'axe du haut correspond à l'extraction de $\tilde{\gamma}_{-1}$ via $\tilde{\beta}_{-1,\mathrm{m}}$ pour quatre valeurs de f. L'axe du bas présente les $|\tilde{\gamma}_n|$ ainsi obtenus pour certaines valeurs de n ; les résultats des n omis par souci de visibilité s'obtiennent par $|\tilde{\gamma}_{-n}(f)| = |\tilde{\gamma}_n(-f)|$. Les points de couleur entourés de blanc sur la courbe de $|\tilde{\gamma}_{-1}|$ correspondent aux résultats des courbes de même couleur sur l'axe du haut.

du bas contient la norme des $\tilde{\gamma}_n(f)$ pour certaines valeurs de n. Les valeurs manquantes, qui s'obtiennent trivialement de $|\tilde{\gamma}_{-n}(f)| = |\tilde{\gamma}_n(-f)|$, sont omises par souci de visibilité. On rapelle que l'argument complexe de $\tilde{\gamma}_{\pm1}(f)$ est simplement le $\mathrm{Arg}\left[\tilde{\beta}_{\pm1,\mathrm{m}}(f)\right]$ de la figure 11.5, si bien que — connaissant $|\tilde{\gamma}_{\pm1}(f)|$ et $\mathrm{Arg}\left[\tilde{\gamma}_{\pm1}(f)\right]$ — on peut reconstruire tous les $\tilde{\gamma}_n(f)$. En pratique, on extrait les $\tilde{\gamma}_n(f)$ avec $n > 1$ à partir de $\tilde{\gamma}_1(f)$ et ceux pour $n < -1$ via $\tilde{\gamma}_{-1}(f)$ à l'aide de la relation (11.12) et du $\tilde{\gamma}_0(f)$ obtenu au préalable. Sur l'axe du bas, l'effet du caractère passe-haut de la bande passante sur les $\tilde{\gamma}_n(f)$ est évident aux f multiples de 4 GHz, avec des creux à ces fréquences sur toutes les courbes, comme discuté à la section 11.1.2. On remarque aussi que l'étalement en fréquence des $\tilde{\gamma}_n(f)$ diminue plus $|n|$ est grand ; conséquence du décalage causé par n dans le

terme $\tilde{g}\,(-f + nf_0)$ de (11.3).

Cet effet fait en sorte que $|\tilde{\gamma}_{\pm 4}\,(f)|$ n'a pas de valeur non nulle pour $|f| < 6\,\text{GHz}$, ce qui peut sembler problématique étant donné que la bande de validité de la calibration discutée à la section 11.1.2 est de $250\,\text{MHz} \leq |f| \leq 6\,\text{GHz}$. Cependant, $|n| = 4$ correspond à une *périodicité de phase* ou *harmonique* de 16 GHz, bien au-delà des limites de la bande passante analogique de 10 GHz ; on ne devrait donc pas pouvoir extraire d'information probante de $|\tilde{\gamma}_{\pm 4}\,(f)|$ même s'il était dans la bande de validité. De plus, puisqu'on résout $f_e/f_0 = 8$ phases distinctes expérimentalement, on obtient aussi 8 valeurs de $n \in \mathbb{Z}$ pour les coefficients de Fourier $\tilde{\beta}_n\,(f)$ et $\tilde{\beta}_{n,\text{m}}\,(f)$ ainsi que pour les $\tilde{\gamma}_n\,(f)$; spécifiquement $-4 \leq n \leq 3$. Ainsi, on a $\tilde{\beta}_n\,(f)$ et $\tilde{\beta}_{n,\text{m}}\,(f)$ pour $n = -4$ sans pour autant les avoir pour $n = 4$. En outre, puisque l'étalement de $|\tilde{\gamma}_{-4}\,(f)|$ en fréquence n'est pas centré en $f = 0$, utiliser ce dernier dans la calibration engendrerait une asymétrie dans les résultats de $\tilde{S}_\phi\,(f)$ à $|f| > 6\,\text{GHz}$. Pour ces raisons, on choisit de ne pas utiliser $\tilde{\gamma}_{-4}\,(f)$ lors de la calibration et de plutôt forcer $\tilde{\beta}_{-4,\text{m}}\,(f) = 0$. En somme, les $\tilde{\gamma}_{\pm 4}\,(f)$ se retrouvent en dehors de l'éventail de fréquences valides pour la calibration, $\tilde{\beta}_{4,\text{m}}\,(f)$ n'est pas accessible par l'expansion en série de Fourier et la bande passante de la mesure nous empêche d'avoir un signal significatif pour $\tilde{\beta}_{-4,\text{m}}\,(f)$; il est donc jugé approprié de simplement annuler ce dernier lors de la calibration.

Selon un raisonnement similaire, on pourrait aussi éliminer les $\tilde{\beta}_{\pm 3}$, puisqu'ils correspondent à des *périodicités de phase*, ou *harmoniques*, de $\pm 12\,\text{GHz}$. Or, comme on a expérimentalement accès aux deux signes de $\tilde{\gamma}_{\pm 3}\,(f)$ — et que leur étalement en f couvre une large part de la zone de validité de la calibration — on préfère simplement appliquer la calibration telle quelle dans ce cas [11]. La limite de la bande passante s'exprime alors d'elle-même dans les résultats. En bout de compte, les n pertinents sont ceux de -2 à 2, correspondant à des $|f| < 10\,\text{GHz}$; valeurs pour lesquelles on obtient bien la zone de calibration valide entre $-6\,\text{GHz}$ et $6\,\text{GHz}$ avec des creux autour de $\pm 4\,\text{GHz}$ et $0\,\text{Hz}$, tel que prédit à la section 11.1.2. Les $|n| \geq 3$ sont en quelque sorte analogues aux résultats à $|f| > 10\,\text{GHz}$, qui sont résolus par la mesure ultrarapide et traités

11. On a vérifié que poser $\tilde{\gamma}_3\,(f) = 0$ lors du traitement des données n'a pas d'effet notable sur les résultats.

par la calibration, bien que le signal à ces fréquences soit au final inaccessible pour des raisons de bande passante.

Avec $\tilde{S}_A(f)$ et tout les $\tilde{\gamma}_n(f)$ pertinents en main, on peut maintenant passer des coefficients de Fourier brutes $\tilde{\beta}_{n,\mathrm{m}}(f)$ à ceux intrinsèques $\tilde{\beta}_n(f)$ à l'aide de (11.8) pour finalement obtenir les $\tilde{S}_\phi(f)$ recherchés à l'aide de (11.7). Les spectres de bruit résolus en phase de la jonction tunnel sont donc expérimentalement accessibles après cette calibration.

11.2.3 Validation de la calibration

Avant de poursuivre l'analyse des résultats, il importe de s'assurer que le processus de calibration ait été fructueux — celui-ci étant au coeur de la fiabilité des résultats. Afin de confirmer cela, on s'intéresse à deux régimes pour lesquels $\tilde{S}_\phi(f, V_{\mathrm{dc}}, V_{\mathrm{ac}})$ a des signatures franches et distinctes — soit le régime photoexcité à grande polarisation en tension continue, ainsi que celui photoexcité sans polarisation continue. La figure 11.7 présente des exemples de résultats dans ces régimes — respectivement aux ensembles de courbes du haut et du bas — en fonction de ϕ et pour $f = 5\,\mathrm{GHz}$.

D'abord, dans le régime à $eV_{\mathrm{dc}}/h \gg 10\,\mathrm{Hz}$, on s'attend à avoir exactement $\tilde{S}_\phi(eV_{\mathrm{dc}} \gg 10\,\mathrm{GHz}) = \frac{e}{R}\big(V_{\mathrm{dc}} + V_{\mathrm{ac}}\cos\phi\big)$ après la calibration ; cela correspond à l'équation (7.168) sur laquelle la calibration s'appuie. C'est effectivement ce qui est observé dans le groupe de courbes du haut de la figure 11.7, correspondant à $V_{\mathrm{dc}} \approx 121\,\mu\mathrm{V} \leftrightarrow 29.29\,\mathrm{GHz}$. Les courbes obtenues sont sinusoïdales, d'amplitudes linéaires selon V_{ac} et centrées autour d'une valeur constante correspondant à $eV_{\mathrm{dc}}/R \approx 481 \times 10^{-27}\,\mathrm{A}^2/\mathrm{Hz}$. Notablement, la phase initiale des oscillations a été ramenée à $\phi_0 = 0$ par la calibration. Ainsi, peu importe le décalage de phase observé au haut de la figure 11.4, celui-ci est corrigé par la calibration de manière à bien respecter le cosinus de (7.168).

À $V_{\mathrm{dc}} = 0$, (7.168) est bien-sûr invalide, mais on s'attend tout de même à ce que $\tilde{S}_\phi(V(t))$ s'approche de (7.173) — essentiellement $|V_{\mathrm{ac}}\cos(\phi)|\,e/R$ — plus V_{ac} est grand. On devrait donc observer un comportement similaire à $|\cos(\phi)|$ pour les valeurs de V_{ac} les plus grandes. C'est bel et bien ce qu'on voit au

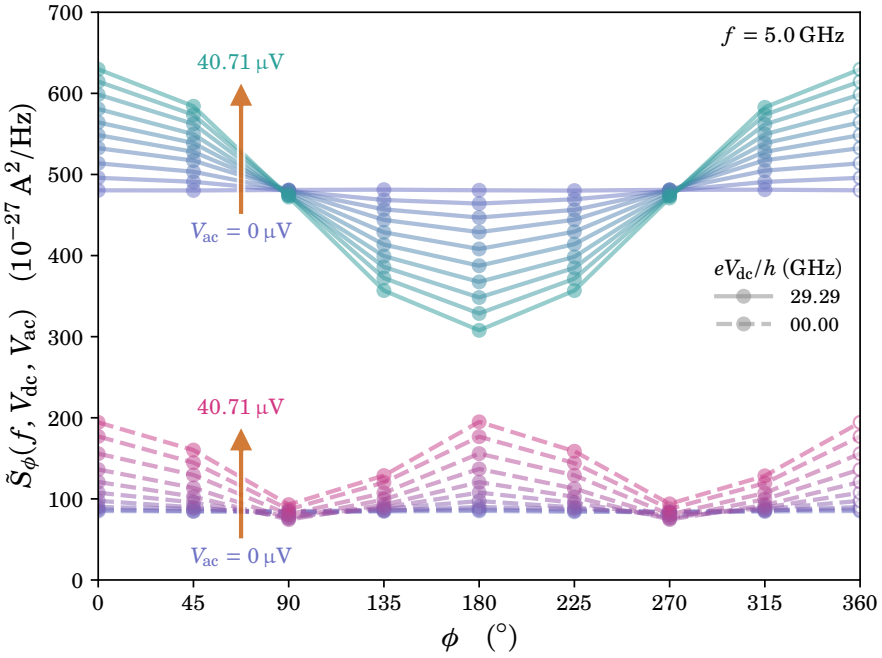


FIGURE 11.7 – Validation de la calibration. Le choix $f = 5\,\text{GHz}$ est arbitraire ; les résultats sont similaires pour toutes les fréquences pour lesquelles la calibration résolue en phase est valide.

groupe de courbes du bas de la figure 11.7. La courbe à $V_{ac} = 0$ est bien près de $hf/R \approx 82 \times 10^{-27}\,\text{A}^2/\text{Hz}$ tel qu'attendu et les courbes présentent une période à moitié moindre de celle du groupe de courbes du haut, caractère s'affirmant d'autant plus que V_{ac} est grand.

En somme, les résultats calibrés concordent bien avec les attentes théoriques à la fois dans le régime $eV_{dc}/h \gg 5\,\text{GHz}$ utilisé pour la calibration [12] et dans le régime opposé à $V_{dc} = 0$. Donc, la partie résolue en phase de la calibration est manifestement valide. Les résultats post-calibration sont donc fiables et on peut les analyser en toute confiance.

12. Ce qui n'est pas surprenant en soi, mais certainement rassurant. Notons que ça ne concorde pas simplement par construction ; les résultats pourraient être complètement inadéquats si les données brutes ne respectaient pas les modèles utilisés, pour extraire les $\tilde{\gamma}_n(f)$ par exemple.

Chapitre 12

Résultats résolus en phase

12.1 Spectres résolus en phase $\tilde{S}_\phi$

Une fois la calibration présentée au chapitre 11 faite et validée, on obtient des résultats pour $\tilde{S}_\phi(f)$ pour toutes les fréquences contenues dans la zone de validité de la calibration — voir la section 11.1.2 — et pour toutes les conditions expérimentales étudiées. Pour alléger la discussion, on présente $\tilde{S}_\phi(f)$ seulement pour quelques valeurs de V_{dc} et V_{ac} représentatives ; des figures supplémentaires dans d'autres conditions expérimentales sont disponibles à l'annexe A.1.

La majorité des résultats présentés ici, soit ceux à $V_{\text{ac}} = 3.63\,\mu\text{V}$ et $V_{\text{ac}} = 11.47\,\mu\text{V}$, proviennent du même ensemble de données que celui utilisé à la section 11.2. Les résultats comprenant d'autres valeurs de V_{ac} ont quant à eux [1] été obtenus via le script disponible à l'annexe C.2.3 ; un balayage en V_{ac} à $V_{\text{dc}} = 0$.

12.1.1 Résultats directs de $\tilde{S}_\phi$

On présente à la figure 12.1 les résultats obtenus pour $\tilde{S}_\phi(f)$ en fonction de f pour des polarisations V_{ac} et V_{dc} non nullles. Ils sont en très bon accord

1. C'est à dire aux figures 12.6 et 12.9.

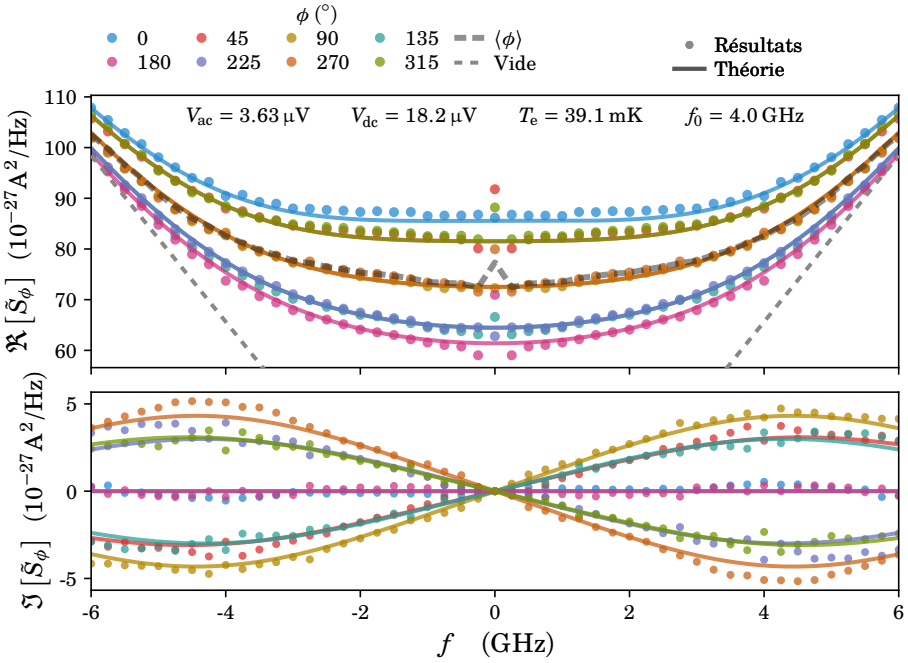


FIGURE 12.1 – Résultats expérimentaux de $\tilde{S}_\phi$ en fonction de f et théorie associée. La température électronique utilisée pour les courbes théoriques est déduite de la figure 11.3.

avec la théorie, considérant l'effet de chauffage discuté à la section 10.2.1. Pour la partie réelle, on observe selon ϕ une augmentation ou une diminution par rapport au bruit photoexcité standard. La situation est similaire du côté de la partie imaginaire, à la différence que celle-ci est nulle dans le cas non résolu en phase. On confirme donc expérimentalement l'équation (7.185), c'est-à-dire que $\tilde{S}_{\langle\phi\rangle}(f) = \tilde{S}(f)$. La partie imaginaire de $\tilde{S}_\phi(f)$ est toujours nulle à $f = 0$, situation pour laquelle le concept même de phase perd son sens. Cette partie imaginaire est difficile à interpréter directement, sinon qu'elle est la conséquence de l'asymétrie en τ de $S_\phi(\tau)$, mais ses différentes contributions entrent en jeu dans les analyses des sections ci-bas. Notons aussi que toutes les courbes tendent à se rejoindre à grande fréquence, ce qui est d'autant plus évident à plus basse polarisation — voir la figure A.4 en annexe.

La figure 12.2, quant à elle, montre les résultats de $\tilde{S}_\phi(f)$ à $f = 5\,GHz$ en fonction de V_{dc}. Similairement à la figure précédente, on remarque une

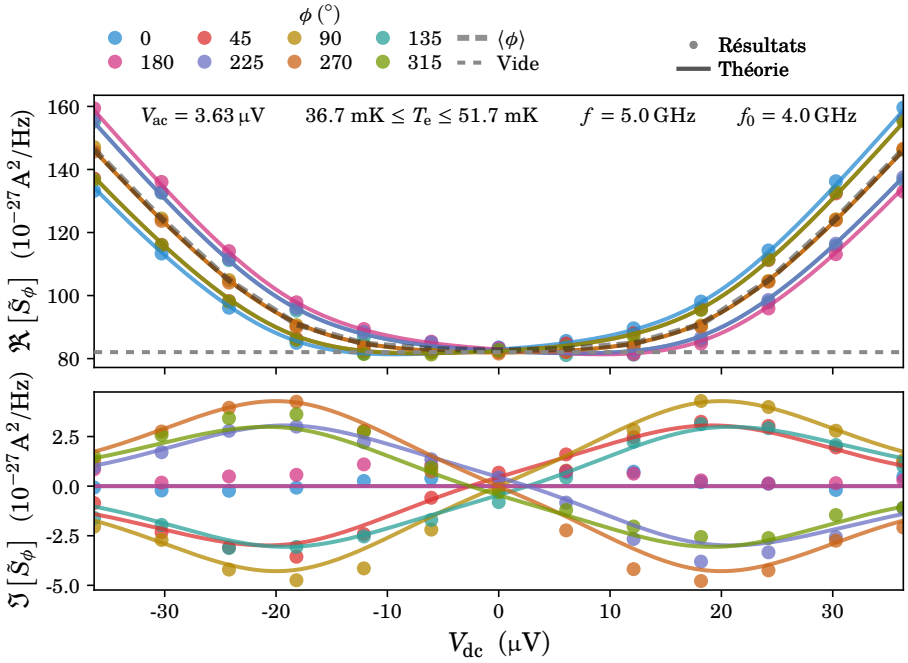


FIGURE 12.2 – Résultats expérimentaux de $\tilde{S}_\phi$ en fonction de V_{dc} et théorie associée. Les courbes théoriques tiennent compte de l'effet de chauffage de la figure 11.3.

augmentation ou diminution du niveau de bruit selon ϕ tout en respectant (7.185). Cependant, à la manière des résultats de [61], les courbes à ϕ donné changent le signe de leur contribution en traversant $V_{dc} = 0$. En fait, on remarque que $\tilde{S}_\phi(-V_{dc}) = \tilde{S}_{\phi+\pi}(V_{dc})$, en accord avec la prédiction théorique de l'équation 7.160. On remarque aussi que $\Im\left[\tilde{S}_\phi(f)\right]$ n'est pas en général nul à $V_{dc} = 0$, l'asymétrie en temps est donc bel et bien attribuable à la photoexcitation.

L'accord avec la prévision théorique est encore une fois remarquable. Cet accord est d'autant plus visuel à la figure 12.3, où on dresse la cartographie des parties imaginaire et réelle de $\tilde{S}_\phi(f)$ en fonction de V_{dc} et ϕ, en parallèle avec des calculs théoriques. En particulier, on voit bien la tendance oblongue et en alternance de direction des zones où la partie imaginaire est non nulle, comme prédit par la théorie [2]. La périodicité en ϕ des résultats y est aussi mise

2. Bien que la faible résolution expérimentale en phase rende ce résultat moins frappant en scrutant la figure de près, la similitude est convaincante lorsqu'observée à bonne distance en plissant les yeux.

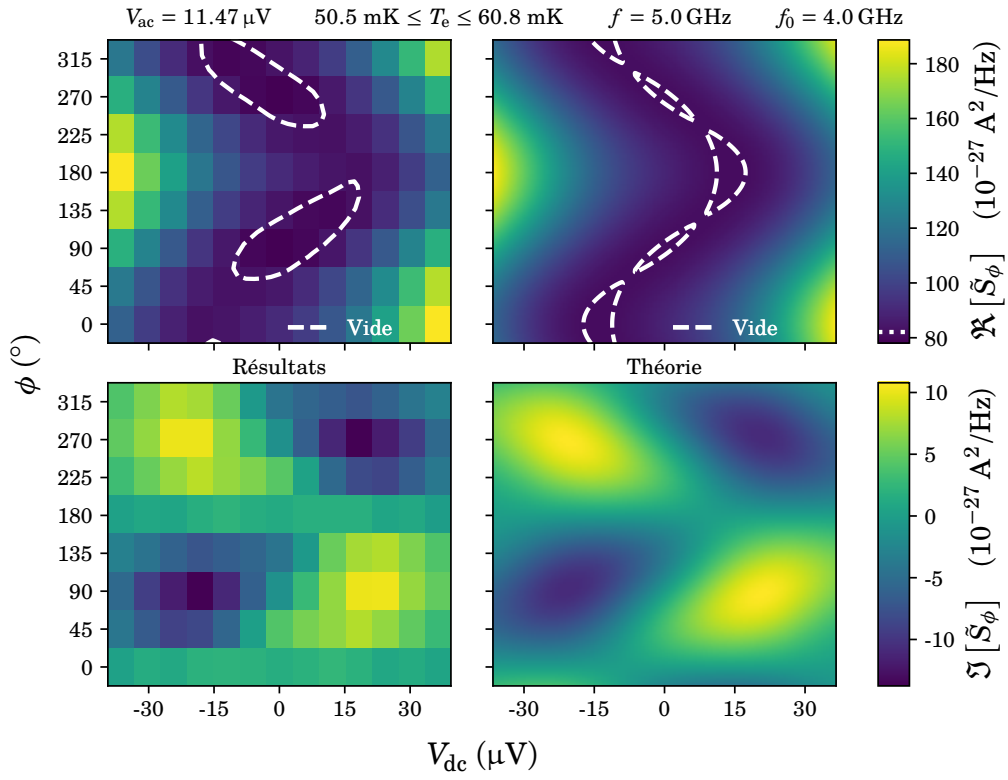


FIGURE 12.3 – Cartographie de $S_\phi(f)$ vis-à-vis ϕ et V_{dc}. Résultats expérimentaux et comparaison à la théorie.

en valeur.

Aux figures 12.2 et 12.2, on voit aussi qu'à certains ϕ donnés, le niveau de $\Re\left[\tilde{S}_\phi(f)\right]$ peut descendre sous celui du vide. Cependant, puisqu'on regarde ici des densités spectrales complexes, il n'est pas clair qu'on ait affaire à de la compression d'état comme observé, entre autres, aux références [61, 65]. Cela dit, les valeurs de ϕ pour lesquelles la partie réelle est au niveau le plus bas à la figure 12.2, soit 0° et 180 ° correspondent aussi à celles ou la partie imaginaire est ≈ 0 expérimentalement et nulle en théorie. Comme $f_0 = 4\,GHz$ n'est pas un multiple entier du $f = 5\,GHz$ de la figure 12.2, il pourrait donc s'agir d'une signature de compression d'état à deux modes comme observé aux références [65, 123]. Alors que, à ces références, une attention particulière doit être portée à la comparaison des niveaux de bruits des différents modes, il est probable que cela ne soit pas nécessaire dans le cas de l'approche par $\tilde{S}_\phi(f)$. En effet, nos résultats

sont obtenus sur une bande qui contient les deux modes, contrairement aux résultats cités ci-haut qui sont obtenus sur deux bandes étroites indépendantes ; les corrélations directes entre les modes sont donc perdues dans cette situation, alors qu'elles sont préservées en bande large. Une analyse théorique plus poussée serait cependant nécessaire pour confirmer si ce comportement de $\tilde{S}_\phi(f)$ est bel et bien associé à de la compression d'état à deux modes.

Bien que cette avenue soit intéressante en soi, on se concentre dans la suite de l'analyse aux autres aspects de $\tilde{S}_\phi(f)$ qui sont directement reliés à la résolution en phase. La section 12.4.1 traite tout de même plus en détail les contributions associées à la compression d'état à un mode.

12.1.2 Bruit en excès de phase

Comme discuté à la section 7.5.1, les densités spectrales qui, comme $\tilde{S}_\phi(f)$, se comportent en $\sim h|f|/e$ à grande fréquence ne se prêtent pas bien au passage dans le domaine temporel à cause du filtrage inhérent à la bande passante finie. Par contre, à l'aide de soustractions judicieuses, on peut obtenir des quantités qui sont nulles en fin de bande passante et ne sont donc pas affectées par cet effet. Le cas de $\tilde{S}_\phi(f)$ est particulièrement intéressant sur ce point, puisqu'il permet de définir une telle quantité avec tous les paramètres expérimentaux fixes.

En effet, on définit le bruit en excès de phase comme la contribution à $\tilde{S}_\phi(f)$ purement due à la phase ϕ comparativement au bruit photoexcité régulier, c'est à dire $\tilde{S}_\phi(f) - \tilde{S}_{\langle\phi\rangle}(f)$. Cette quantité est donc insensible aux effets de chauffage qui teintent les résultats du chapitre 10. Il ne suffit alors que de légèrement filtrer le résultat de la soustraction près de 6 GHz pour que la transition vers 0 soit bien lisse[3] avant de prendre la transformée de Fourier inverse. On obtient ainsi $S_\phi(\tau) - S_{\langle\phi\rangle}(\tau)$ dans le domaine temporel ; le corrélateur temporel représentant les corrélations supplémentaires observées à la phase ϕ grâce à la résolution en phase et qui seraient autrement moyennées à 0 par la moyenne sur ϕ.

3. On rappelle que 6 GHz est la limite supérieure de la zone de validité de la calibration résolue en phase.

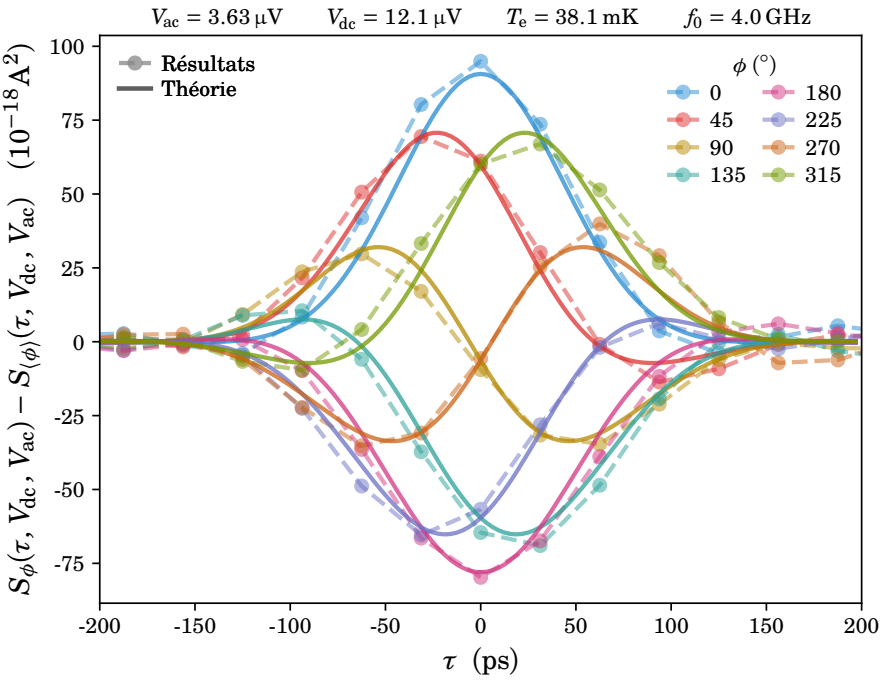


FIGURE 12.4 – Bruit en excès de phase à $V_{dc} \neq 0$. Résultats obtenus des données de la figure A.4.

On présente de tels résultats à la figure 12.4. On choisi les polarisations $V_{dc} = 12.1\,\mu V$ et $V_{ac} = 3.63\,\mu V$ de manière à avoir une signature claire pour chaque valeur de ϕ et que les différentes courbes se rejoignent tout de même le plus possible à grande fréquence. Les $\tilde{S}_\phi(f)$ utilisés pour générer cette courbe sont disponibles en annexe à la figure A.4. L'accord entre les résultats et les prévisions théoriques est toujours probant. Bien sûr, par construction, la moyenne de ces courbes sera nulle, mais il est intéressant de remarquer que celles-ci ne s'annulent pas exactement deux à deux. C'est particulièrement évident pour les phases 0° et 180° qui sont de signes opposés, mais n'ont pas la même amplitude ; les autres phases se doivent donc de compenser en moyenne pour ce déséquilibre. Cet effet est probablement une conséquence de $V_{dc} \neq 0$, spécifiquement du fait que la variation de la polarisation instantanée à ϕ comparativement à la polarisation moyenne n'est pas proportionnellement équivalent à toutes les phases de références. Concrètement, si on à $V_{ac} = 0.2V_{dc}$, la polarisation moyenne sera $1 - 1/1.2 = 16.\bar{6}\%$ plus faible que la polarisation instantanée subie par la

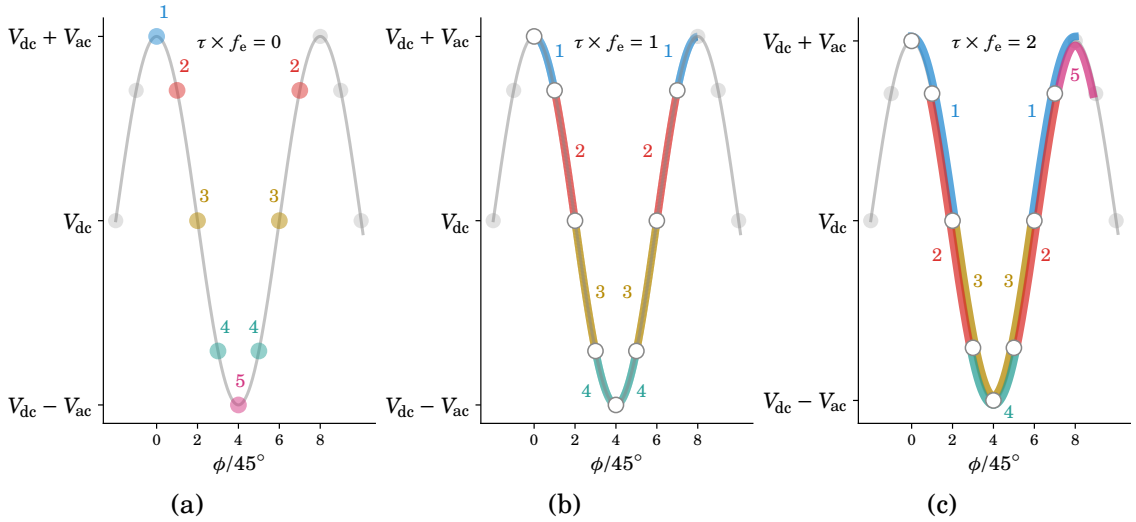


FIGURE 12.5 – Points équivalents de l'excitation pour trois délais τ à $V_{dc} \neq 0$. Explique le nombre de croisements à ces τ à la figure 12.4. Il y a huit phases résolues sur une période, représentées par les points blancs ou colorés ; les points grisés correspondent aux périodes adjacentes.

jonction à la phase 0°, alors qu'elle sera $1 - 1/0.8 = 25\%$ plus élevée que celle à 180 ° ; en général, on a $(V_{dc} + V_{ac})/V_{dc} \neq V_{dc}/(V_{dc} - V_{ac})$.

Ces résultats comportent plusieurs symétries intéressantes. En particulier, toutes les paires de courbes à ϕ_1 et ϕ_2 telles que $(\phi_1 + \phi_2) = 360°$ — ce qui est équivalent à $\phi_1 = -\phi_2$ considérant que les phases sont équivalentes modulo 360° — sont la réflexion l'une de l'autre par rapport à $\tau = 0$. Cette propriété est simplement la conséquence de l'excitation et du choix de référence de phase. En effet, puisque l'excitation $V(t)$ est paire, il va de soi que les points à $\pm t$ verront des tensions instantannées identiques et, comme $\phi = f_0 t$, ces temps sont associés aux phases $\pm\phi$. Le même argument étant valide pour $t + \tau$ implique directement la symétrie $S_{-\phi}(-\tau) = S_\phi(\tau)$ observée.

Une autre caractéristique intéressante de ces résultats est la présence de points de croisement des courbes qui concordent tous avec la discrétisation temporelle de la mesure. Le résultat précédent $S_{-\phi}(-\tau) = S_\phi(\tau)$ et l'équation (7.159) permettent d'expliquer cette observation. De paire, ces équations impliquent $S_\phi(\tau) = S_{(-\phi - 2\pi f_0 \tau)\%2\pi}(\tau)$, ce qui explique les croisements — par

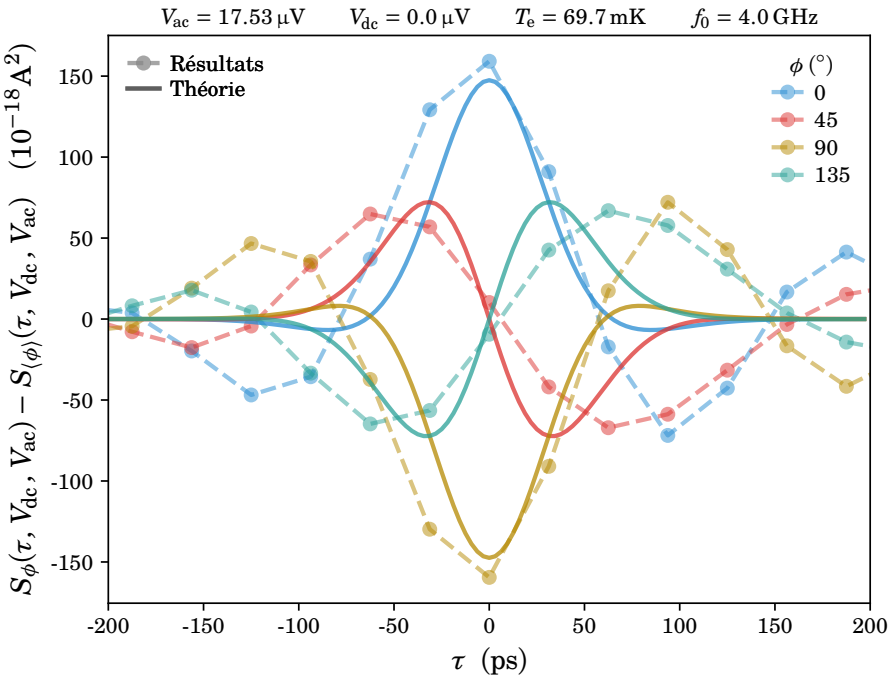


FIGURE 12.6 – Bruit en excès de phase à $V_{dc} = 0$. Les quatre phases omises, par souci le clarté, sont identiques à celles montrées.

exemple $S_{0°}(31.25\,\text{ps}) = S_{315°}(31.25\,\text{ps})$. Chaque décalage expérimental de $\Delta\tau$ à ϕ donné est donc associé à une autre phase qui est elle aussi résolue expérimentalement. Les points expérimentaux où il n'y a pas de croisement sont en fait l'exception et correspondent à la situation à ϕ et τ fixes pour lesquels [4] $(-\phi - 2\pi f_0\tau)\,\%2\pi = \phi$.

Le schéma de la figure 12.5 et l'équation (7.65) expliquent quant à eux le nombre de valeurs distinctes possibles à chaque τ. Effectivement, cette dernière équation stipule

$$S_\phi(\tau) = S_{eq}(\tau)\cos\left(\frac{e}{\hbar}\int_{\phi/\Omega}^{\phi/\Omega+\tau} V(t')\,dt'\right), \tag{12.1}$$

ce qui veut dire qu'à τ fixe, les valeurs de $S_\phi(\tau)$ et $S_\phi(\tau) - S_{\langle\phi\rangle}(\tau)$ sont essentiellement déterminées par l'intégrale du signal d'excitation entre le point de

4. Comme on le voit pour $S_{45°}(-62.5\,\text{ps}) = S_{(-45°-(-90°))\%2\pi}(-62.5\,\text{ps}) = S_{45°}(-62.5\,\text{ps})$.

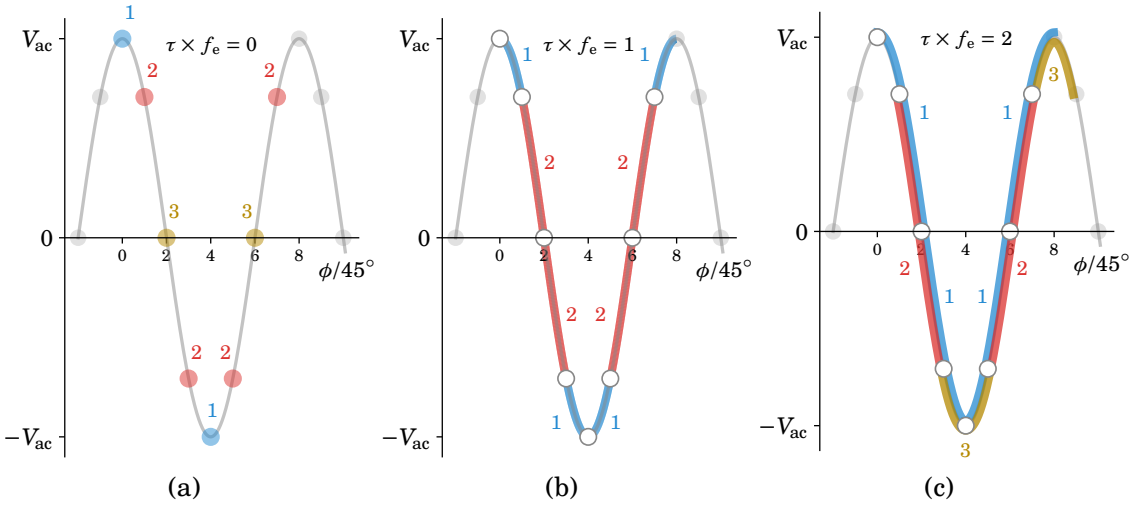


FIGURE 12.7 – Points équivalents de l'excitation pour trois délais τ à $V_{dc} = 0$. Explique le nombre de croisements à ces τ à la figure 12.6. Il y a huit phases résolues sur une période, représentées par les points blancs ou colorés ; les points grisés correspondent aux périodes adjacentes.

référence de la phase et celui décalé de τ. On détermine ainsi le nombre de valeurs distinctes possibles pour $S_\phi(\tau) - S_{\langle\phi\rangle}(\tau)$ en regardant les points associés à ϕ et τ sur une période de l'excitation.

C'est ce qu'on montre à la figure 12.5 pour les trois premières valeurs de τ dans la situation $V_{dc} \neq 0$ associée à la figure 12.4. Pour le cas $\tau = 0$ de la sous-figure (a), on a que $t = t + \tau$ si bien que (12.1) n'est sensible qu'à la tension instantanée de l'excitation. Comme l'excitation est centrée à $V_{dc} \neq 0$ — et comme la phase de référence est ajustée pour que la phase à $\phi = 0°$ corresponde bien au haut de l'excitation et que les $f_e/f_0 = 8$ phases résolues soient toutes distantes de 45° — il y a 5 valeurs de polarisation distinctes, identifiées par des cercles de couleurs différentes et énumérées sur la figure. C'est donc pourquoi il n'y a que 5 valeurs de $S_\phi(\tau)$ permises à $\tau = 0$ parmi les huit phases mesurées à la figure 12.4. La situation est similaire à la sous-figure (b), à la différence qu'on intègre cette fois (12.1) entre des échantillons adjacents. Le caractère cosinusoïdal de l'excitation fait en sorte que ce résultat ne dépend que des amplitudes de $V(t)$ aux bornes de l'intégration. On a donc 4 valeurs distinctes possibles à $\tau = 1$, énumérées et identifiées par les lignes de couleurs distinctes

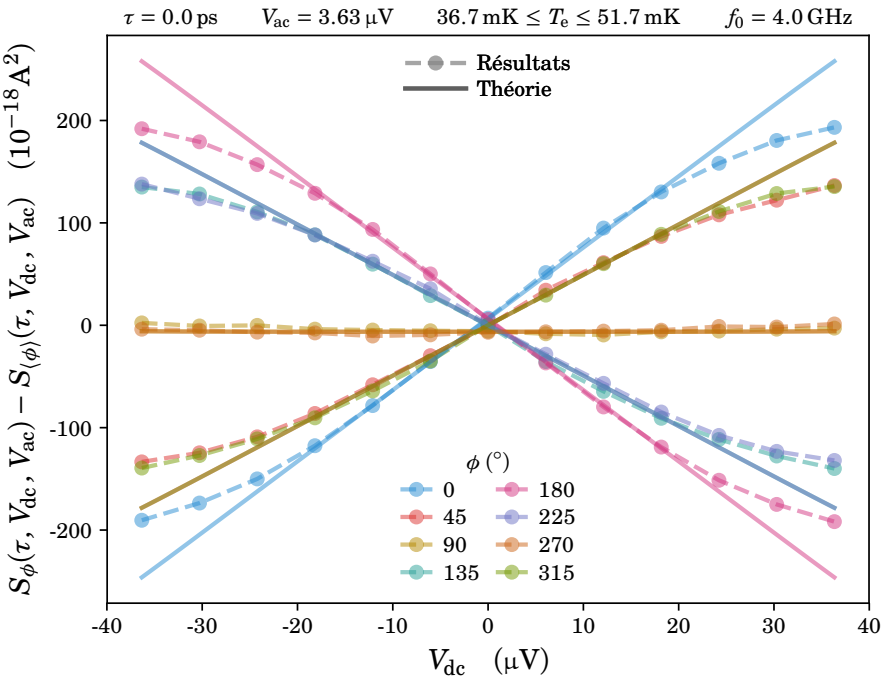


FIGURE 12.8 – Bruit en excès à $\tau = 0$ en fonction de V_{dc} à V_{ac} constant.

à la sous-figure (b). L'histoire se répète à la sous-figure (c), avec cette fois des points d'intérêts éloignés de deux acquisitions. On trouve 5 situations distinctes dans cette situation, ce qui est aussi vérifié à la figure 12.4.

Des résultats similaires à $V_{dc} = 0$, visibles à la figure 12.6, viennent conforter ces résultats. Dans cette situation, seules quatre phases sont pertinentes étant donnée $V_{dc} = 0$ et la propriété $\tilde{S}_\phi(f, -V_{dc}) = \tilde{S}_{\phi+\pi}(f, V_{dc})$ de l'équation (7.160). Les résultats concordent moins bien avec la théorie que ceux présentés ci-haut, puisque le V_{ac} nécessaire pour obtenir une amplitude satisfaisante est élevé est qu'on voit donc plus d'effet de fenêtrage. La forme et la tendance des courbes restent bonnes, mais il manque des contributions à hautes fréquences qui permettraient des variations plus rapides au cours du temps et rapprocheraient les résultats des prévisions théoriques. Cette fois-ci, comme attendu après la discussion ci-haut, les courbes sont symétriques autour de $V_{dc} = 0$ et s'annulent deux à deux, signe que la jonction tunnel n'est effectivement sensible qu'à l'amplitude de la polarisation et non à son signe. Sachant cela, l'approche schématique présentée à la figure 12.7 explique toujours le nombre de valeurs distinctes de

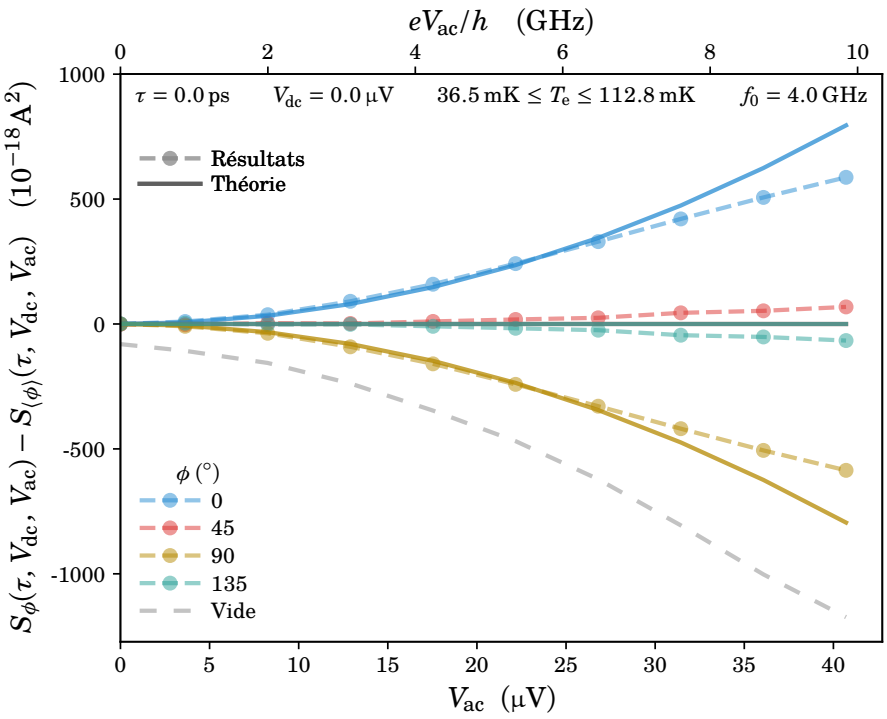


FIGURE 12.9 – Bruit en excès à $\tau = 0$ en fonction de V_{ac} à $V_{dc} = 0$.

$S_\phi(\tau)$ permises à chaque τ en théorie. En effet, comme les situations à $\pm V_{ac}$ sont équivalentes, les points à la même amplitude d'excitation, mais de signe opposés sont maintenant considérés équivalents, ce qui explique bien les structures observées.

On peut aussi observer la tendance du bruit en excès de phase en fonction de V_{dc} et V_{ac} en regardant comment se comporte l'amplitude du pic central à $\tau = 0$ en fonction de ces variables. La figure 12.8 montre cela pour $V_{ac} = 3.63\,\mu V$ et en balayant V_{dc}, alors que la figure 12.9 le présente à $V_{dc} = 0$ pour un éventail de valeurs de V_{ac}. Dans les deux cas, les résultats suivent fidèlement la théorie pour les basses polarisations, mais s'en écartent aux plus hautes valeurs. Cela est encore une fois attribuable au fenêtrage au bord de la zone de validité de la calibration qui est plus marqué plus V_{dc} et V_{ac} sont grand.

En sommes, les résultats de bruit en excès de phase dans le domaine temporel des figures 12.4 et 12.6 sont donc une signature claire de la validité de l'équation

(7.65). C'est-à-dire que les corrélations $S_\phi(\tau)$ et par la même occasion les densités spectrales $\tilde{S}_\phi(f)$ ne dépendent pas de tout l'historique de l'excitation, mais bien uniquement de l'intégrale de $V(t)$ entre $t = \phi/(2\pi f_0)$ et $t + \tau$.

Il est toutefois assez difficile d'interpréter plus en profondeur les résultats de $S_\phi(\tau)$ et $\tilde{S}_\phi(f)$ directement. C'est pourquoi on se concentre dans les sections qui suivent à décortiquer les différentes contributions et décompositions possibles des spectres résolus en phases. Entre autres, on en extrait une quantité substantielle d'information sur les corrélations courant–courant au sein du signal ainsi que sur la réponse de la jonction à la polarisation en tension alternative.

12.2 Distribution de Wigner du signal

Mathématiquement, la raison pour laquelle les spectres résolus en phase $\tilde{S}_\phi(f)$ ont une partie imaginaire est l'asymétrie par rapport à $\tau = 0$ de l'autocovariance $S_\phi(\tau)$, qui provient elle même de la partie impaire de l'autocovariance brute $S_{\phi,m}(\tau)$ mesurée expérimentalement — voir la figure 11.2. D'un autre côté, les sections 12.4.2 et 12.6.1 ci-après montrent qu'on peut aussi attribuer cette partie imaginaire à l'asymétrie des $\tilde{\beta}_n(f)$. La partie impaire en τ de $S_\phi(\tau)$ et l'asymétrie de $\tilde{\beta}_n(f)$ par rapport à $f = 0$ sont donc essentiellement la même chose.

Malgré qu'ils soient asymétriques selon le signe de f, les $\tilde{\beta}_n(f)$ sont symétriques autour de $nf_0/2$, tel que montré à la figure 12.12. Il est donc assez tentant de simplement décaler chaque $\tilde{\beta}_n(f)$ de $nf_0/2$ pour les recentrer en $f = 0$ et ainsi obtenir une nouvelle quantité — qu'on appelle, à tout hasard, $\tilde{W}(\phi, f)$ — similaire au spectre résolu en phase $\tilde{S}_\phi(f)$, mais purement réelle. Cette opération étant réversible [5] toute l'information de $\tilde{S}_\phi(f)$ serait préservée. On aurait alors une quantité purement réelle et symétrique contenant toute l'information sur la mesure, ce qui semble assez pratique.

5. Les hautes fréquences $|f| > 10\,\text{GHz}$ peuvent être considérées sans signal. Comme la fréquence maximale résolue est 16 GHz, on ne perd donc pas d'information par décalage de moins de 6 GHz.

Or, il se trouve que ce $\tilde{W}(\phi, f)$ n'est autre que la distribution de Wigner[6] du signal [124 ; 125, §8]. En effet, $\tilde{W}(\phi, f)$ a une expression en termes de corrélateurs courant–courant similaire à celle (7.113) de $\tilde{S}_\phi(\omega)$, soit

$$\tilde{W}(\phi, \omega) = \int \langle i(\phi/\Omega + \tau/2)\, i(\phi/\Omega - \tau/2)\rangle\, \mathrm{e}^{-\mathrm{i}\omega\tau}\, \mathrm{d}\tau, \tag{12.2}$$

avec $\omega = 2\pi f$ et $\phi = \Omega t$. On remarque aussi que

$$S_{\phi-\Omega\tau/2}(\tau) = \langle i(\phi/\Omega + \tau/2)\, i(\phi/\Omega - \tau/2)\rangle\,, \tag{12.3}$$

via (7.111). On a aussi, via (7.114),

$$S_\phi(\tau) = \mathcal{F}^{-1}\left[\tilde{S}_\phi(\omega)\right](\tau) \tag{12.4}$$

$$= \sum_{n=-\infty}^{\infty} \mathcal{F}^{-1}\left[\tilde{\beta}_n(\omega)\right](\tau)\, \mathrm{e}^{\mathrm{i}n\phi} \tag{12.5}$$

$$= \sum_{n=-\infty}^{\infty} \beta_n(\tau)\, \mathrm{e}^{\mathrm{i}n\phi}, \tag{12.6}$$

si bien que

$$S_{\phi-\Omega\tau/2}(\tau) = \sum_{n=-\infty}^{\infty} \mathrm{e}^{\mathrm{i}n\phi}\beta_n(\tau)\, \mathrm{e}^{-\mathrm{i}n\Omega\tau/2}. \tag{12.7}$$

Ainsi,

$$\tilde{W}(\phi, \omega) = \int \left(\sum_{n=-\infty}^{\infty} \mathrm{e}^{\mathrm{i}n\phi}\beta_n(\tau)\, \mathrm{e}^{-\mathrm{i}n\Omega\tau/2}\right) \mathrm{e}^{-\mathrm{i}\omega\tau}\, \mathrm{d}\tau \tag{12.8}$$

$$= \sum_{n=-\infty}^{\infty} \mathrm{e}^{\mathrm{i}n\phi} \int \beta_n(\tau)\, \mathrm{e}^{-\mathrm{i}(\omega+n\Omega/2)\tau}\, \mathrm{d}\tau \tag{12.9}$$

$$= \sum_{n=-\infty}^{\infty} \tilde{\beta}_n(\omega + n\Omega/2)\, \mathrm{e}^{\mathrm{i}n\phi} \tag{12.10}$$

6. En adaptant la définition au cas périodique et avec la notation en termes de ϕ plutôt que t.

et donc, en termes de fréquences plutôt que de pulsations, on trouve finalement

$$\boxed{\tilde{W}(\phi, f) = \sum_{n=-\infty}^{\infty} \tilde{\beta}_n(f + nf_0/2)\, \mathrm{e}^{\mathrm{i}n\phi}}\,. \tag{12.11}$$

Bien entendu, comme le terme $n = 0$ n'est pas affecté et que la seule dépendance en ϕ est dans l'exponentielle, on a que

$$\int_{-\pi}^{\pi} \tilde{W}(\phi, f)\, \frac{\mathrm{d}\phi}{2\pi} = \tilde{S}_{\langle\phi\rangle}(f) = \tilde{\beta}_0(f)\,, \tag{12.12}$$

le bruit photoexcité non-résolu en phase habituel.

De plus, on peut assigner une fonction $\tilde{M} = \tilde{\beta}_n(f + nf_0/2)$ aux coefficients de Fourier de $\tilde{W}(\phi, f)$ tel que

$$\begin{aligned} M(n, \tau) &= \mathcal{F}^{-1}\left[\tilde{M}\right](\tau) && (12.13)\\ &= \mathcal{F}^{-1}\left[\tilde{\beta}_n(f + nf_0/2)\right](\tau) && (12.14)\\ &= \mathrm{e}^{-\mathrm{i}\pi\tau n f_0}\mathcal{F}^{-1}\left[\tilde{\beta}_n(f)\right](\tau) && (12.15) \end{aligned}$$

si bien que

$$\boxed{M(n, \tau) = \mathrm{e}^{-\mathrm{i}\pi\tau n f_0}\beta_n(\tau)}\,, \tag{12.16}$$

ce qui n'est autre que la fonction caractéristique de la distribution de Wigner. C'est essentiellement sa représentation duale de Fourier à la fois en ϕ et en f. Le facteur de phase vient simplement faire tourner $\beta_n(\tau)$ dans le plan complexe pour éliminer sa partie imaginaire ; la distribution de Wigner et sa fonction caractéristique se devant d'être toutes deux réelles et symétriques. On note que $M(n, \tau)$ correspond à l'enveloppe du $\beta_n(\tau)$ auquel il est associé. En effet, en multipliant (12.16) par son complexe conjugé, on trouve $M^2(n, \tau) = |\beta_n(\tau)|^2$, ce qui implique

$$M(n, \tau) = \pm|\beta_n(\tau)|\,. \tag{12.17}$$

Il peut donc être positif ou négatif — et potentiellement changer de signe en variant n ou τ — mais suivra toujours l'amplitude de $\beta_n(\tau) \in \mathbb{C}$.

La figure 12.10 présente la distribution de Wigner $\tilde{W}(\phi, f)$ obtenue expérimentalement pour les mêmes valeurs de polarisation qu'à la figure 12.1. On y

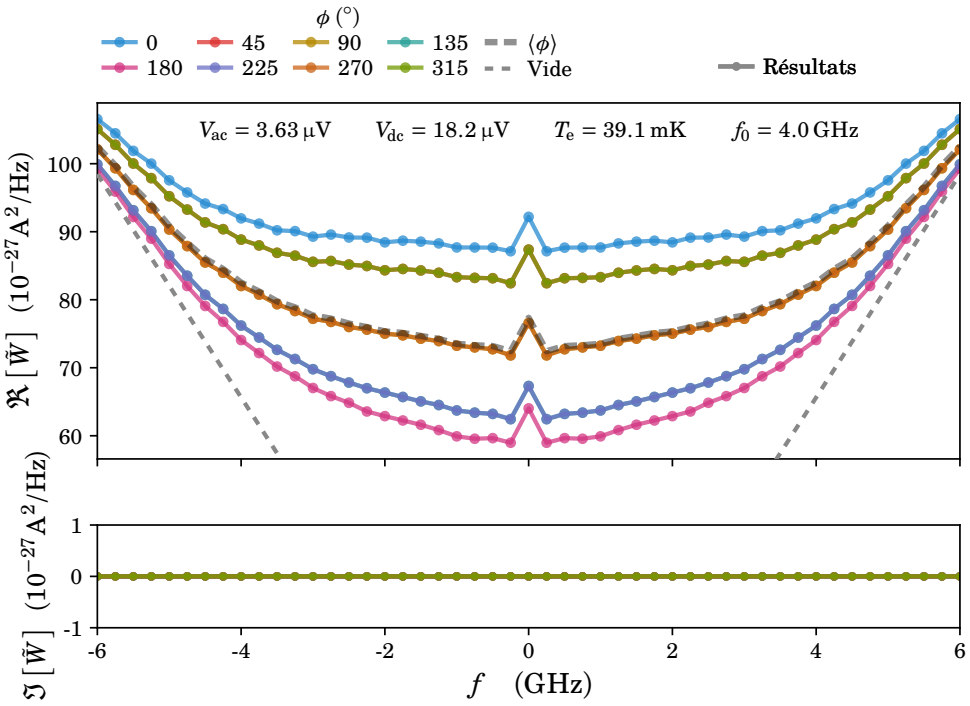


FIGURE 12.10 – Résultats expérimentaux de $\tilde{W}(\phi, f)$ en fonction de f. Résultats obtenus à partir des mêmes données que la figure 12.1.

voit clairement que la partie imaginaire est bien nulle et que toute l'information de $\tilde{S}_\phi(f)$ est maintenant contenue dans sa partie réelle. Les résultats expérimentaux de la fonction caractéristique $M(n, \tau)$ sont quant à eux abordés à la section 12.6.

La distribution de Wigner est très intéressante, puisque c'est une distribution de quasiprobabilité permettant d'extraire virtuellement toute l'information contenue dans un signal. Mesurer $W(\phi, f)$ correspond à faire la tomographie du signal, ce dernier pouvant même en être extrait à une phase globale près [125, §8.8]. Il est donc remarquable que les spectres résolus en phase — dont on

a posé la définition principalement pour des considérations pragmatiques reliées à la mesure et au traitement numérique des signaux — soient à toutes fins pratiques équivalents à la distribution de Wigner du signal. Conceptuellement, on a donc les équivalences

$$\tilde{S}_\phi(f) \Leftrightarrow \tilde{W}(\phi, f) \qquad \tilde{\beta}_n(f) \Leftrightarrow \tilde{M} \tag{12.18}$$

$$S_\phi(\tau) \Leftrightarrow W(\phi, \tau) \qquad \beta_n(\tau) \Leftrightarrow M(n, \tau) \tag{12.19}$$

entre l'approche des spectres résolus en phase et celle de Wigner. Chacune de ces quantités permettant d'obtenir toutes les autres.

Il ne faut toutefois pas confondre la distribution de Wigner du signal avec celle de l'état quantique associé à celui-ci [125, 13.13], qui correspond à la tomographie de l'état quantique [126, 127]. Ces deux *familles* de distributions de Wigner sont probablement reliées [128, 129], mais une étude théorique plus poussée serait requise pour bien cerner l'information qu'elles contiennent l'une sur l'autre.

Remarquons que le décalage en fréquence des composantes spectrales et le lien avec le caractère *réel* ou *complexe* de leurs transformées de Fourier rappelle — sans y correspondre exactement — le concept de *signal analytique* [130 ; 76, §4.1].

12.3 Remarque sur les $\tilde{\beta}_n$

Bien qu'on les ait introduits pour leur utilité — voire leur nécessité — à la calibration des données résolues en phase, l'intérêt des coefficients de Fourier $\tilde{\beta}_n(f)$ ne s'arrête pas là. En effet, par leur nature, et via les définitions (7.114) et (7.130), les $\tilde{\beta}_{\pm n}(f)$ correspondent à la partie du bruit oscillant à $\pm n f_0$ suite à l'excitation à f_0. C'est ainsi la partie de la réponse se retrouvant à la n^{e} harmonique, dépouillée de toute autre périodicité. Le résultat $\tilde{\beta}_0(f) = \tilde{S}_{\langle\phi\rangle}(f) = \hat{\tilde{S}}(f)$ de l'équation (7.137) est alors naturel ; c'est l'effet de la photoexcitation sans égard à la périodicité de la réponse, sans la modulation temporelle du bruit qu'elle engendre — à l'harmonique zéro pour ainsi dire. De leur côté, les $\tilde{\beta}_{\pm 1}(f)$

capturent la réponse oscillante à la première harmonique, et de même pour $\tilde{\beta}_{\pm 2}$ et éventuellement toutes les harmoniques résolues par la mesure.

Avant d'analyser les $\tilde{\beta}_n(f)$ plus en profondeur, soulignons une de leur propriété a priori inattendue : tous les $\tilde{\beta}_n(f)$ obtenus expérimentalement sont réels, bien que le traitement des données ne l'impose pas explicitement. En effet, numériquement les parties imaginaires de chaque $\tilde{\beta}_n(f)$ sont toutes négligeables comparativement aux parties réelles, pour autant que ces dernières ne soient pas ≈ 0. Les parties imaginaires sont donc attribuables à de faibles erreurs numériques ou expérimentales. De prime abord, ce résultat est surprenant. Or, dans le cas du bruit photoexcité, c'est en fait une conséquence naturelle de (7.157), soit $\tilde{S}_{\phi}^{\mathrm{pa}\,*}(f) = \tilde{S}_{-\phi}^{\mathrm{pa}}(f)$, l'équivalent *phase–harmonique* de l'hermiticité *temps–fréquence* du bruit photoexcité, ainsi que de la défissions (7.115) des $\tilde{\beta}_n(f)$ [76, §4.1.2.1]. Donc, les variables n et ϕ étant conjugées et $\tilde{\beta}_n(f)$ étant la série de Fourier de $\tilde{S}_{\phi}^{\mathrm{pa}}(f)$, on trouve nécessairement que $\tilde{\beta}_n(f) \in \mathbb{R}$. On remarque alors que, via (7.130), on a $\tilde{\beta}_n^*(f) = \tilde{\beta}_{-n}(-f) = \tilde{\beta}_n(f)$ et donc $\tilde{\beta}_{-n}(f) = \tilde{\beta}_n(-f)$ en corollaire.

Ces propriétés ne sont cependant pas fondamentales ; elles reposent sur le choix de la phase de référence à l'équation (7.68) et de la parité du cosinus. Par exemple, on peut changer la référence de phase en appliquant la substitution $\phi \to \phi + \Delta\phi$ sur l'indice de $\tilde{S}_{\phi}^{\mathrm{pa}}$, c'est-à-dire en posant $\tilde{S}_{\phi}^{\prime\mathrm{pa}} \equiv \tilde{S}_{\phi+\Delta\phi}^{\mathrm{pa}}$. Prendre la phase opposée donne alors $\tilde{S}_{-\phi}^{\prime\mathrm{pa}}(f) = \tilde{S}_{-\phi+\Delta\phi}^{\mathrm{pa}}(f) \neq \tilde{S}_{-\phi-\Delta\phi}^{\mathrm{pa}}(f)$. Cette dernière inégalité forcerait alors des $\tilde{\beta}_n'(f)$ définis via $\tilde{S}_{\phi}^{\prime\mathrm{pa}}(f)$ à être complexes.

Le choix $\Delta\phi = 0$ est cependant très utile pour faciliter le traitement des données, l'analyse des résultats et la représentation graphique de ceux-ci. De plus, comme le traitement des données en tant que tel permettrait d'obtenir des $\tilde{\beta}_n(f)$ complexes, le fait que les résultats soient réels est un indice fort qu'on mesure bel et bien du bruit photoexcité résolu en phase et que la calibration est adéquate — les $\tilde{\beta}_{n,\mathrm{m}}(f)$ mesurés au départ étant certainement complexes. À tout le moins, le signal mesuré a la bonne symétrie.

12.4 Corrélations modulées et compression d'état

12.4.1 Compression d'état à un mode

Comme les $\tilde{\beta}_n(f)$ représentent en quelque sorte la réponse du bruit à l'excitation, on s'attend à ce qu'ils soient reliés aux propriétés qui émergent du bruit photoexcité. Une de ces propriétés parmi les plus intéressantes est l'observation, dans les bonnes conditions, d'un niveau de bruit inférieur au bruit du vide pour une quadrature du signal [61], avec un niveau de bruit augmenté pour la quadrature conjugée de manière à globalement respecter le principe d'incertitude d'Heisenberg [91]. C'est ce qu'on appelle la compression d'état, ou plus communément le *squeezing* [39 ; 40, §3.7]. Ce phénomène est particulièrement utile pour augmenter la précision de certaines mesures [34, 131, 132] et est au coeur du fonctionnement de plusieurs concepts d'amplificateurs opérants à la limite quantique, comme l'amplificateur paramétrique Josephson [11, 30, 33, 133] et plusieurs autres [27, 28, 31, 32, 134].

Pour observer du *squeezing* au sein du bruit photoexcité — à l'instar des références [61, 123] — il faut donc tirer profit de la modulation du bruit engendrée par la photoexcitation, d'où le lien naturel avec les $\tilde{\beta}_n(f)$. En effet, $\tilde{\beta}_n(f)$ est essentiellement équivalent au $\chi = \langle X_1 X_2 \rangle$ de [123, Fig. 7.] — la contribution responsable du *squeezing* — avec $f_1 = -f + nf_0$ et $f_2 = f$. On remarque que les fréquences en jeu dans $\tilde{\beta}_n(f)$ respecteront toujours $f_1 + f_2 = nf_0$. Cette relation évoque immédiatement le mélange à $n+2$ ondes [42], où n photons à f_0 sont convertis de manière cohérente en une paire de photons à f_1 et f_2, tout en respectant la conservation de l'énergie. En optique non linéaire, cette génération de paires de photons corrélés [7] est d'ailleurs associée au *squeezing* [42, §10.6.3 ; 41, §9.4] ; ce sont en quelque sorte deux manifestations du même phénomène [8].

7. Aussi appellée conversion paramétrique descendante spontanée ou *spontaneous parametric down–conversion* (SPDC).

8. Le lien entre les fluctuations électroniques dans le régime de *squeezing* à un mode et l'émission de paires de photons micro-ondes corrélées est d'ailleurs le sujet de mes travaux de maîtrise [44 ; 46].

Les mesures de la référence [61] sont effectuées en bande étroite autour d'une seule fréquence $f_1 = f_2 = f$. Pour y observer du *squeezing*, il faut donc que les deux photons de la paire émise soient dégénérés en fréquence, c'est-à-dire que $2f = nf_0$; c'est ce qu'on appelle le *squeezing* à un mode. Les fréquences auxquelles le *squeezing* peut être observé dépendent alors du choix de n, soit $f = f_0/2$ pour $n = 1$ et $f = f_0$ à $n = 2$; respectivement associées à du mélange à trois et quatre ondes [9]. On devrait donc pouvoir reproduire ces résultats en traçant $\tilde{\beta}_0(f) \pm \tilde{\beta}_1(f)$ à $f = 2\,\mathrm{GHz}$ et $f = 4\,\mathrm{GHz}$ étant donnée la fréquence de photoexcitation de $f_0 = 4\,\mathrm{GHz}$. Cependant, comme 4 GHz est exclu de la zone de validité de la calibration résolue en phase — voir la section 11.1.2 — seule $f = 2\,\mathrm{GHz}$ devrait donner un résultat valide.

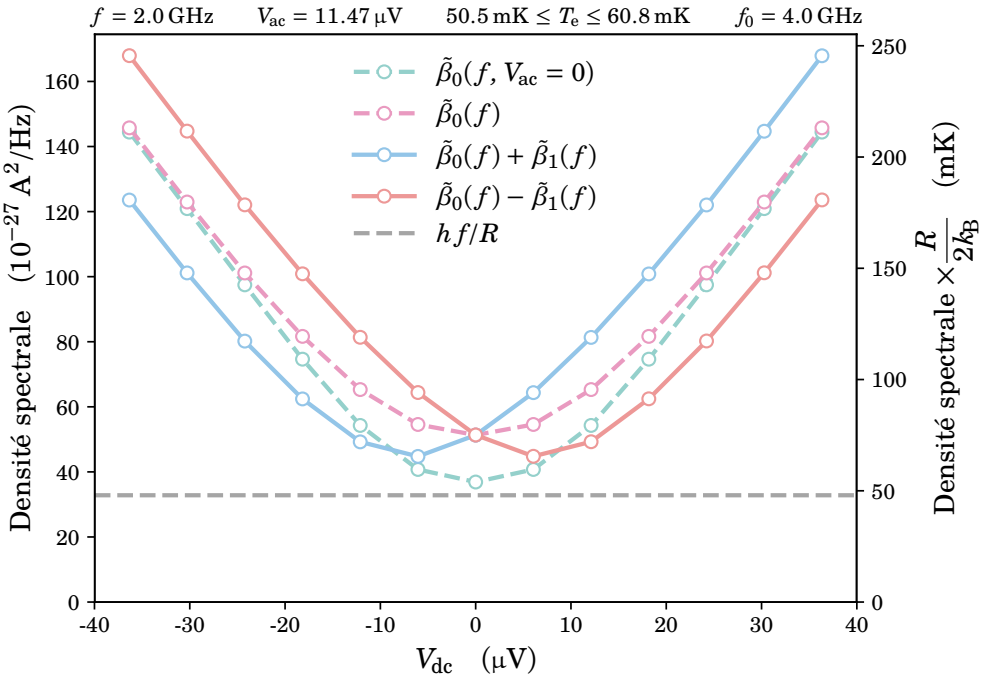


FIGURE 12.11 – Modulation du bruit par la photoexcitation dans le régime permettant le *squeezing* à un mode. Les conditions expérimentales ne permettent pas d'observer un niveau de bruit inférieur à celui du vide, mais on observe tout de même le transfert d'amplitude des fluctuations entre les quadratures.

On présente donc les résultats de *squeezing* à un mode obtenus à $f = 2\,\mathrm{GHz}$

9. Notons qu'une coquille s'est immiscée dans la référence [61] ; les mélanges à trois et quatre ondes y sont inversés.

à la figure 12.11 en traçant $\tilde{\beta}_0(f) \pm \tilde{\beta}_1(f)$. On y observe comme attendu que la moyenne des quadratures correspond à $\tilde{\beta}_0(f)$ et que celles-ci voient leur niveau de bruit alternativement augmenté ou diminué selon le signe de V_{dc}. Cependant, le niveau de bruit ne descend jamais sous le niveau du vide ; on n'observe pas de *squeezing*. Il semble que l'effet combiné de la fréquence de mesure assez faible et de la température électronique relativement élevée [10] soit plus grand que la diminution maximale du bruit engendrée par les paires.

En effet, la situation idéale pour observer le *squeezing* à $f = f_0/2$ est $hf \gg k_{\mathrm{B}} T_{\mathrm{e}}$ de manière à ce que le bruit photoexcité présente un plateau assez large centré en $V_{\mathrm{dc}} = 0$ et s'étendant jusqu'à $|V_{\mathrm{dc}}| \approx h|f|/e$. Conceptuellement, l'idée est de repousser l'effet de la température aux plus hautes tensions de manière à avoir le bruit minimal possible sur le plateau. La modulation du bruit engendrée par la photoexcitation et détectée de manière synchrone peut alors plus aisément amener le niveau de bruit sous celui du vide. En plus, la référence [63, Fig. 3.] montre que cette modulation elle-même est d'autant plus efficace que T_{e} est faible. Les résultats de la Fig. 2 de [61] sont bel et bien dans ce régime, comme y en atteste la courbe formée de cercles rouges, alors que les résultats (pointillé magenta) en sont éloignés ; les ratios $hf/k_{\mathrm{B}} T_{\mathrm{e}}$ étant respectivement de $\frac{h}{k_{\mathrm{B}}} \frac{7.2\,\mathrm{GHz}}{28\,\mathrm{mK}} \approx 12.3$ et $\frac{h}{k_{\mathrm{B}}} \frac{2\,\mathrm{GHz}}{50.5\,\mathrm{mK}} \approx 1.9$. En fait, on voit bien que même la courbe sans photoexcitation (pointillé cyan) n'atteint pas le niveau du vide. On observe donc bien le même comportement qu'à la référence [61], mais les conditions expérimentales ne permettent pas d'observer de *squeezing*. On observerait probablement du *squeezing* à un mode en utilisant une photoexcitation à $f_0 = 8\,\mathrm{GHz}$, ce qui nous limiterait cependant à seulement 4 phases résolues et restreindrait grandement l'éventail de fréquences sur lequel la calibration serait valide.

12.4.2 Modulation des corrélations en bande large

Contrairement aux mesures en bande étroite, les mesures en bande large ne nous limitent pas aux fréquences $f_0/2$ et f_0 associées au cas dégénéré ; on obtient des résultats de $\tilde{\beta}_n(f)$ sur toute la zone de validité de la calibration.

10. C'est-à-dire $\gtrsim 50\,\mathrm{mK}$ comparativement au $\approx 28\,\mathrm{mK}$ de [61].

On présente donc, à la figure 12.12, les résultats de $\tilde{\beta}_1(f)$ et $\tilde{\beta}_2(f)$ en fonction de f pour $V_{\text{ac}} = 11.47\,\mu\text{V}$ et pour l'éventail de V_{dc} exploré. Les résultats pour $n < 0$ vérifient expérimentalement $\tilde{\beta}_{-n}(f) = \tilde{\beta}_n(f)$, ils sont simplement l'image miroir par rapport à $V_{\text{dc}} = 0$ des résultats présentés ici, et sont omis par souci de clarté et de concision.

Tout d'abord, remarquons que les deux $\tilde{\beta}_n(f)$ sont symétriques autour de $f = nf_0/2$, ce qui est mis en lumière par les axes du haut. Cette symétrie est attendue de la forme de l'équation (7.130) ; en posant $\tilde{\beta}'_n(\Delta f) \equiv \tilde{\beta}_n(\Delta f + nf_0/2)$, on obtient $\tilde{\beta}'_n(\Delta f) = \lim_{T\to\infty} \frac{1}{T} \langle \tilde{\imath}_T(\Delta f + nf_0/2)\, \tilde{\imath}_T(-\Delta f + nf_0/2) \rangle$, ce qui est bien symétrique autour de $\Delta f = 0$. Cette symétrie se comprend aussi aisément par l'émission de paires séparées de Δf. Le cas $\Delta f = 0$ correspond bien sûr au régime dégénéré discuté ci-haut, alors qu'un $\Delta f \neq 0$ correspond à l'émission de paires à $f \pm \Delta f$; équivalent à $f \pm -\Delta f$.

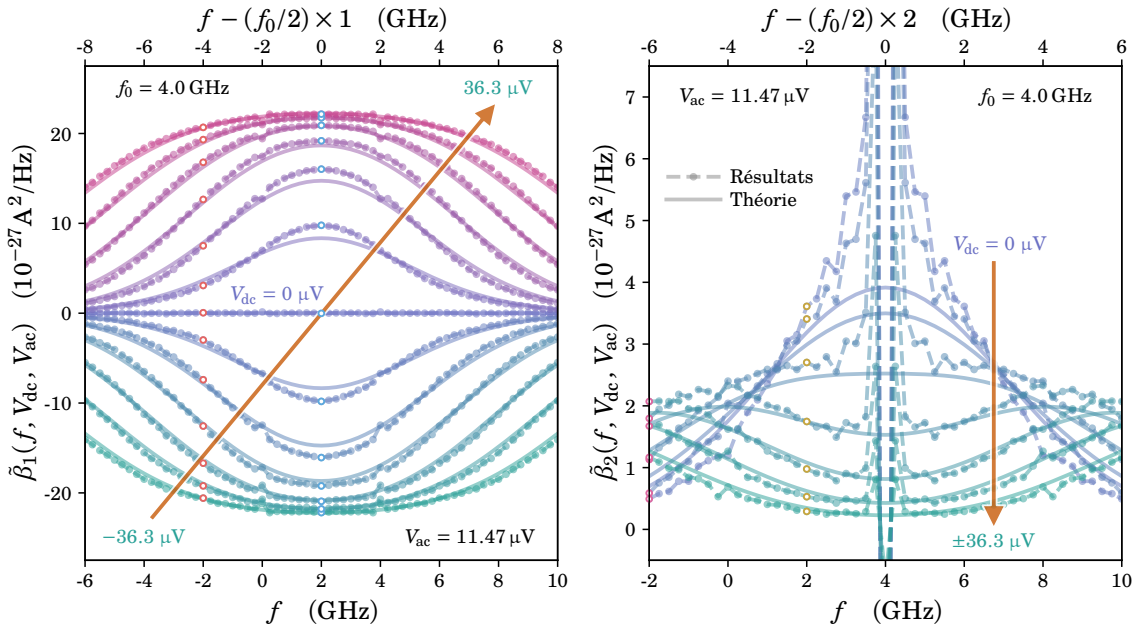


FIGURE 12.12 – Coefficients de Fourier $\tilde{\beta}_n(f)$ pour $n = 1, 2$. Les axes du haut sont décalés en fréquence pour mettre en évidence la symétrie autour de $(f_0/2) \times n$. Les $\tilde{\beta}_n(f)$ pour $n = -2, -1$ vérifient $\tilde{\beta}_{-n}(f) = \tilde{\beta}_n(-f)$ et ne sont pas affichés par souci de légibilité. Considérant cela, les points colorés au centre blanc correspondent à ceux de même couleur à la figure 12.13. Les courbes théoriques tiennent compte de l'effet de chauffage observé à la figure 11.3.

À la sous-figure de gauche, on observe bien une structure symétrique centrée en $f = f_0/2$ et d'amplitude grandissante avec V_{dc}. On remarque que $\tilde{\beta}_1(f)$ est nul à $V_{\mathrm{dc}} = 0$, ce qui est en accord avec le fait qu'il n'y a pas de *squeezing* possible dans ce cas à $n = 1$. Le fait que $\tilde{\beta}_1(f)$ soit antisymétrique avec V_{dc} concorde aussi avec les résultats de la figure 12.11, où le niveau de bruit de chaque quadrature se voit augmenté ou diminué selon le signe de V_{dc}.

La situation est similaire à la sous-figure de droite, correspondant à $\tilde{\beta}_2(f)$. Bien que l'on n'ait pas de résultat valide à $f = 4\,\mathrm{GHz}$, la tendance aux fréquences adjacentes indique qu'il s'agirait bien de la fréquence optimale pour $n = 2$. De plus, en contraste avec la situation $n = 1$, les corrélations additionnelles d'amplitudes maximales sont bien associées à $V_{\mathrm{dc}} = 0$ et tendent vers 0 pour les grandes tensions, en accord avec [61, Fig. 3].

Il est intéressant de remarquer que les $\tilde{\beta}_1(f)$ non nuls s'étendent au-delà de $|f| > f_0 = 4\,\mathrm{GHz}$ de manière à ce que les corrélateurs courant–courant associés puissent faire intervenir des paires de photons ayant des fréquences de signes opposés — par exemple $\tilde{\beta}_1(5\,\mathrm{GHz}) = \lim_{T\to\infty} \frac{1}{T} \langle \tilde{\imath}_T(-1\,\mathrm{GHz})\, \tilde{\imath}_T(5\,\mathrm{GHz}) \rangle$. Bien que cette dernière expression respecte la conservation de l'énergie, on a bien $f_1 + f_2 = f_0$ avec $f_1 = -1\,\mathrm{GHz}$ et $f_2 = 5\,\mathrm{GHz}$, la présence d'une fréquence négative est a priori surprenante. Il est cependant possible de simplement remanier l'expression de mélange à trois ondes tel que $f_2 = f_0 - f_1 = f_0 + |f_1|$ pour qu'une interprétation s'impose d'elle-même : un photon de bruit à 1 GHz se combine avec un photon d'excitation à 4 GHz pour générer le photon à 5 GHz ; c'est de la conversion paramétrique montante spontanée, ou *spontaneous parametric up-conversion* (SPUC) [135]. Notons que même si le photon de fréquence négative pourrait provenir du vide, aucun travail net n'en sera extrait en autant qu'un autre processus associé à un corrélateur $\tilde{\beta}_n(f \neq f_2)$ redonne l'énergie empruntée au vide, c'est-à-dire si seules des corrélations sont échangées globalement ; cette situation étant similaire à celle de [136, 137].

L'accord entre les résultats (points reliés par un pointillé) et la théorie (lignes continues) est convaincant, particulièrement aux plus grandes amplitudes de polarisation V_{dc}. Il y a cependant un léger désaccord autour de $f = nf_0/2$, la fréquence centrale, pour les valeurs de V_{dc} plus faibles. Plusieurs facteurs

peuvent expliquer ce léger désaccord, le plus probable étant des imperfections lors de la calibration. En particulier, si les V_{dc} appliqués sur la jonction pour la calibration sont trop grands, le bruit de grenaille émis par la jonction pourrait ne plus être linéaire [11] en V_{dc} ou V_{ac}. Aussi, tel que vu dans [44, Figure 3.13], ce même type de non-linéarité au sein du bruit de grenaille peut provenir de la saturation d'un amplificateur. Une autre cause d'erreur potentielle est la pureté de l'excitation à f_0 ; si celle-ci contient des harmoniques supérieures — 8 GHz en l'occurence — la calibration résolue en phase pourrait en être affectée. Les erreurs semblent affecter $\tilde{\beta}_2(f)$ plus que $\tilde{\beta}_1(f)$, ce qui est normal étant donné son amplitude moindre et le fait que sa calibration — essentielement l'extraction de $\tilde{\gamma}_2(f)$ — est déduite de deux autres calibrations ayant leur propres causes d'erreur, soit celle du $\tilde{\gamma}_1(f)$ utilisé pour calibrer $\tilde{\beta}_1(f)$ et celle de $\tilde{G}_A(f)$ dans la situation non résolue en phase.

Sommes toutes, les mesures de $\tilde{S}_\phi(f)$ en bande large nous donnent accès d'un seul coup à une myriade de corrélateurs courant–courant dans le domaine fréquentiel, les $\tilde{\beta}_n(f)$, sur une large bande de fréquence et l'accord avec la théorie est convaincant.

12.5 Susceptibilité de bruit

Comme le laissent entrevoir les sections précédentes, nos résultats permettent aussi de mettre en valeur le lien entre le *squeezing* et les corrélateurs courant–courant associés à la susceptibilité de bruit [63, 138]. En effet, l'expression (7.130) des coefficients de Fourier $\tilde{\beta}_n(f)$ en tant que corrélateurs courant–courant dans le domaine fréquentiel, c'est-à-dire $\tilde{\beta}_n(f) = \lim_{T\to\infty} \frac{1}{T} \langle \tilde{\imath}_T(-f + nf_0)\, \tilde{\imath}_T(f) \rangle$, est très similaire aux corrélateurs introduits dans [138] et [63]. En particulier, en adaptant la notation à celle utilisée ici, on

11. Par exemple, si la caractéristique I–V de la jonction devenait non-linéaire — en forme de « S » plutôt qu'une ligne — on surestimerait le V pour un I donné et, de même, le bruit de grenaille attendu.

peut exprimer le corrélateur de dynamique du bruit $X_2^{(p)}(f)$ de cette première référence comme

$$X_2^{(n)}(f) = \lim_{T\to\infty} \frac{1}{T} \frac{\langle \tilde{\imath}_T(f)\,\tilde{\imath}_T(nf_0 - f)\rangle + \langle \tilde{\imath}_T(-f)\,\tilde{\imath}_T(-nf_0 + f)\rangle}{2} \tag{12.20}$$

$$= \frac{\tilde{\beta}_n(f) + \tilde{\beta}_n^*(f)}{2}, \tag{12.21}$$

ce qui donne

$$\boxed{X_2^{(n)} = \tilde{\beta}_n(f)}, \tag{12.22}$$

sachant que $\tilde{\beta}_n(f) \in \mathbb{R}$ pour les raisons discutées ci-haut. D'ailleurs, en substituant la forme (7.156) de $\tilde{S}_\phi^{\text{pa}}(\omega)$ dans la définition (7.115) de $\tilde{\beta}_n(\omega)$ et en s'inspirant de la démarche de la section 7.6.4, on trouve

$$\tilde{\beta}_p(\omega) = \int_{-\pi}^{\pi} \tilde{S}_\phi^{\text{pa}}(\omega)\, \mathrm{e}^{-\mathrm{i}p\phi} \frac{\mathrm{d}\phi}{2\pi} \tag{12.23}$$

$$= \frac{1}{2}\Bigg\{ \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} J_n(z)\, J_m(z)\, \delta_{m,n-p}\, \tilde{S}_{\text{eq}}(-\omega + n\Omega + \nu) + \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} J_n(z)\, J_m(z)\, \delta_{m,n+p}\, \tilde{S}_{\text{eq}}(+\omega + n\Omega + \nu) \Bigg\}, \tag{12.24}$$

et donc

$$\boxed{\tilde{\beta}_p(\omega) = \sum_{n=-\infty}^{\infty} J_n(z)\, J_{n+p}(z)\, \frac{\tilde{S}_{\text{eq}}(\omega + n\Omega + \nu) + (-1)^p \tilde{S}_{\text{eq}}(-\omega - n\Omega + \nu)}{2}}, \tag{12.25}$$

ce qui est le même résultat que pour le $X_2^{(p)}(\omega)$ de [138, éq. (46)] [12]. Ainsi, les $\tilde{\beta}_n(f)$ représentent la dynamique du bruit et sont donc reliés à la susceptibilité de bruit, à la différence que cette dernière est définie dans la limite $V_{\text{ac}} \to 0$ et est remise à l'échelle par V_{ac}.

12. C'est aussi équivalent aux X de [61] et au χ_m de [123].

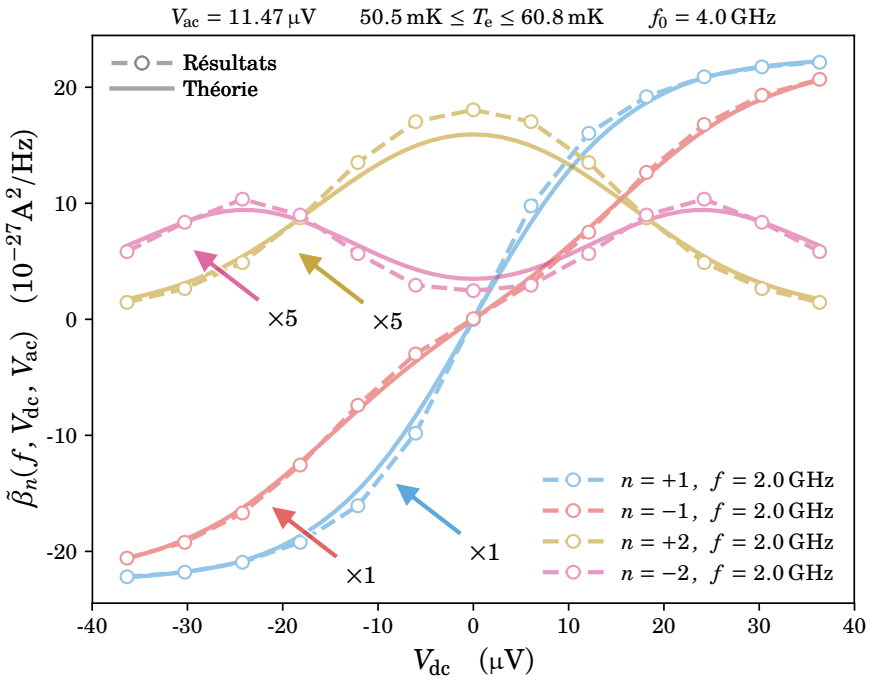


FIGURE 12.13 – Corrélateurs courant–courant $\tilde{\beta}_n(f)$ versus V_{dc}. Les courbes à $n = 2$ sont grossies d'un facteur 5 à fin de visibilité. Les données de cette figure sont mis en évidence à la figure 12.12 via les mêmes marqueurs que la présente, sachant que $\tilde{\beta}_{-n}(f) = \tilde{\beta}_n(-f)$.

On montre à la figure 12.13, des résultats représentatifs de $\tilde{\beta}_n(f)$ pour $n = \pm 1$ et $n = \pm 2$. Ce sont essentiellement des coupes transverses de la figure 12.12 ; les données de la figure 12.13 sont d'ailleurs mises en évidence sur la figure 12.12 par des symboles communs aux figures. De fait, l'accord avec la théorie est probant.

La courbe bleue, à $n = 1$ et $f = f_0/2 = 2\,GHz$, est très similaire aux résultats de [63, Fig.3], mais à $f = f_0/2$ plutôt que $f \approx f_0$. On observe bien une tendance strictement monotone impaire en V_{dc}, s'approchant d'une constante à grand $|V_{dc}|$. L'arrondi causé par la température électronique T_e et la dépendance en V_{dc} de celle-ci rendent difficile de confirmer que la transition entre le régime central quasi-linéaire et les régimes constants périphériques s'effectue à $V_{dc} \approx V_{ac}$, comme observé par [63], mais les résultats semblent tout de même bien concorder. Ce sont ces corrélations, non nulles seulement en présence de la photoexcitation,

qui sont responsables de la modulation du bruit à la figure 12.11 et, dans les bonnes conditions, du *squeezing* à un mode observé à $f = f_0/2$. Notre approche nous permet aussi aisément de regarder le cas $n = -1$ à la même fréquence $f = 2\,\mathrm{GHz}$ — voir la courbe rouge — ce qui ne semble pas avoir été étudié auparavant.

Pour $n = 2$, on choisi de regarder la fréquence $f = 2\,\mathrm{GHz}$, bien que $4\,\mathrm{GHz}$ correspondrait au régime de *squeezing* à un mode, pour éviter la zone de non-validité centrée en $f_0 = 4\,\mathrm{GHz}$ de la calibratin résolue en phase — voir la section 10.1.3 — et de manière à avoir accès à des mesures de $\tilde{\beta}_2(f)$ valides pour $\pm f$, ce qui sera pertinent à la section 12.6. Comme attendu, on voit qu'à cette harmonique les corrélations excédentaires crées par l'excitation sont maximales à $V_{\mathrm{dc}} = 0$. La courbe magenta, dans le cas $n = -2$, est quant à elle piquée à $V_{\mathrm{dc}} \neq 0$. Cela correspond bien à la séparation et au décalage en fréquence du pic à $f_0 = 4\,\mathrm{GHz}$ observé avec $|V_{\mathrm{dc}}|$ grandissant à la figure 12.12.

La forme générale de ces courbes s'explique bien par les propriétés de la jonction tunnel qui ont déjà été introduites pour la calibration résolue en phase. En effet, l'effet de V_{dc} sur la jonction tunnel peut être interprété comme l'imposition d'un temps caractéristique $\tau_e \sim h/|eV_{\mathrm{dc}}|$ auquel les électrons tentent de traverser la jonction par effet tunnel [53] ; V_{ac} venant moduler celui-ci régulièrement au cours du temps tel que $\tau_e = h/|eV_{\mathrm{dc}} + eV_{\mathrm{ac}}\cos(2\pi f_0 t)|$. Ainsi, à $eV_{\mathrm{dc}} \gg hf$, le régime de calibration, τ_e est très court et le bruit suit fidèlement l'excitation avec une réponse en phase purement à la première harmonique en $\phi = 2\pi f_0 t \,\%\, 2\pi$ — voir la figure 11.7. Cela explique que les $\tilde{\beta}_{\pm 1}$ tendent vers des plateaux à grand V_{dc} alors que les $\tilde{\beta}_{\pm 2} \to 0$ traduisent l'absence de réponse à l'harmonique supérieure. Les plateaux sont aussi de signe opposé à $\pm V_{\mathrm{dc}}$ simplement via (7.160). Bref, si τ_e est déjà court comparativement à $1/f$, le bruit observé suit à toutes fins pratiques l'excitation, si bien que raccourcir τ_e — augmenter V_{dc} — n'affecte pas sa réponse davantage. Près de $V_{\mathrm{dc}} = 0$, ce temps caractéristique diverge [13], mais il reste modulé périodiquement entre $h/|eV_{\mathrm{ac}}|$ et $\sim \infty$ par la photoexcitation de manière à généré une réponse périodique à l'excitation, même si celle-ci ne sera pas aussi fidèle que dans le régime de calibration. De plus, comme la jonction elle-même est insensible au signe de la polarisation, on s'attend à une

13. En pratique, le temps caractéristique sera alors limité thermiquement par $h/(k_{\mathrm{B}} T_{\mathrm{e}})$.

réponse similaire à $|\cos\phi|$ pour $V_{\mathrm{dc}} = 0$ et $V_{\mathrm{ac}} \neq 0$, encore une fois tel qu'observé à la figure 11.7. La périodicité de $|\cos\phi|$ étant la même que $\cos^2\phi \sim \cos(2\phi)$, la réponse des fluctuations se retrouve principalement à la seconde harmonique et la réponse à l'harmonique fondamentale est inexistante.

À l'instar de la susceptibilité de bruit, ces courbes de $\tilde{\beta}_n(f, V_{\mathrm{dc}})$ nous informent donc sur la modulation des corrélations qui sont induites au sein du bruit lorsque la jonction tunnel est soumise à une photoexcitation. Elles sont essentiellement la partie des fluctuations responsable du *squeezing*, complètement dénuées de la contribution stationnaire. L'approche large bande nous donne de plus la flexibilité supplémentaire et intéressante d'observer ces corrélations en dehors de la situation à un mode $f = nf_0$. En particulier, avoir accès à $f < 0$ pour $n > 0$ ou vice-versa nous permet d'introduire le concept de *réponse harmonique* traité à la section 12.6.

12.6 Réponse harmonique

Comme discuté aux sections précédentes, les $\tilde{\beta}_n(f)$ contiennent beaucoup d'information sur la réponse des fluctuations suite à une excitation périodique. Cependant, il reste difficile d'interpréter les $\tilde{\beta}_n(f)$ ayant un n et un f de signes opposés autrement qu'en termes de conversion de photons. On suggère ici une approche inspirée de la susceptibilité de bruit qui combine les $\tilde{\beta}_n(f)$ et $\tilde{\beta}_{-n}(f) = \tilde{\beta}_n(-f)$ et propose une interprétation intéressante, indépendante du signe de f et de n.

12.6.1 Réponse harmonique, en phase et en quadrature

En sciant la somme de l'équation (7.114) en trois, un terme à $n = 0$ et deux sous-sommes selon le signe de n, et en recombinant ces dernières terme à terme

selon $|n|$, on peut exprimer les spectres résolus en phase comme

$$\tilde{S}_\phi(f) = \tilde{\beta}_0(f) + \sum_{n=1}^{\infty} \left(\tilde{X}^{(n)}(f) \cos(n\phi) + \mathrm{i}\tilde{Y}^{(n)}(f) \sin(n\phi) \right), \tag{12.26}$$

avec

$$\tilde{X}^{(n)}(f) = \tilde{\beta}_n(f) + \tilde{\beta}_{-n}(f) = \tilde{\beta}_n(f) + \tilde{\beta}_n(-f) \tag{12.27}$$

$$\tilde{Y}^{(n)}(f) = \tilde{\beta}_n(f) - \tilde{\beta}_{-n}(f) = \tilde{\beta}_n(f) - \tilde{\beta}_n(-f), \tag{12.28}$$

où on a utilisé la propriété $\tilde{\beta}_{-n}(f) = \tilde{\beta}_n(-f)$ découlant de $\tilde{\beta}_n(f) \in \mathbb{R}$. On remarque que $\tilde{S}^*_\phi(f) = \tilde{S}_{-\phi}(f)$ est trivial sous cette forme[14], tout comme $\tilde{S}_{\langle\phi\rangle}(f) = \tilde{\beta}_0(f)$. De plus, il est évident que $\tilde{X}^{(n)}(f) = \tilde{X}^{(n)}(-f) = \tilde{X}^{(-n)}(f)$ et $\tilde{Y}^{(n)}(f) = -\tilde{Y}^{(n)}(-f) = -\tilde{Y}^{(-n)}(f)$, toute l'information se retrouve donc à $f > 0$ et $n > 0$.

Évidemment, comme les équations (12.27) et (12.28) sont réversibles, les $\tilde{X}^{(n)}(f)$ et $\tilde{Y}^{(n)}(f)$ contiennent la même information que les $\tilde{\beta}_n(f)$; ce n'est qu'une réexpression pratique. On note aussi que la réponse $\tilde{X}^{(n)}(f)$ associée à la partie réelle de $\tilde{S}_\phi(f)$ correspond au X' de [63] et au $X_1^{(n)}$ de [138, éq. (37)], mais le $\tilde{Y}^{(n)}(f)$ associé à la partie imaginaire de $\tilde{S}_\phi$ semble inédit.

On appelle $\tilde{X}^{(n)}(f)$ et $\tilde{Y}^{(n)}(f)$ la réponse harmonique de $\tilde{S}_\phi(f)$ à l'excitation $V(t) = V_{\mathrm{dc}} + V_{\mathrm{ac}} \cos\big(\phi(t)\big)$, avec $\phi(t) = 2\pi f_0 t$. Il faut cependant travailler un peu plus pour bien interpréter $\tilde{X}^{(n)}(f)$ et $\tilde{Y}^{(n)}(f)$. Prenons d'abord la transformée de Fourier de (12.26), ce qui donne

$$S_\phi(\tau) = \beta_0(\tau) + \sum_{n=1}^{\infty} \left(X^{(n)}(\tau) \cos(n\phi) + \mathrm{i}Y^{(n)}(\tau) \sin(n\phi) \right). \tag{12.29}$$

On profite alors du résultat (12.16) — voir la section 12.2 pour les détails —

14. Rappelons au passage que cette propriété — et donc aussi la décomposition discutée ici — dépend du choix de référence de phase et serait certainement plus compliquée pour un $\Delta\phi$ arbitraire. Par souci de simplicité, on choisit $\Delta\phi = 0$ lors de la calibration sans perte de généralité.

pour obtenir

$$\beta_n(\tau) = \mathrm{e}^{\mathrm{i}\pi\tau n f_0} M(n,\tau)\,, \tag{12.30}$$

avec $M(n,\tau) \in \mathbb{R}$ et $M(n,\tau) = M(-n,\tau) = M(n,-\tau)$.

On trouve donc,

$$\begin{aligned} X^{(n)}(\tau) &= \beta_n(\tau) + \beta_{-n}(\tau) && (12.31)\\ &= \left(\mathrm{e}^{\mathrm{i}\pi\tau n f_0} + \mathrm{e}^{-\mathrm{i}\pi\tau n f_0}\right) M(n,\tau) && (12.32)\\ &= 2\,M(n,\tau)\cos(\pi n f_0 \tau) && (12.33) \end{aligned}$$

et, de même,

$$\begin{aligned} Y^{(n)}(\tau) &= \beta_n(\tau) - \beta_{-n}(\tau) && (12.34)\\ &= \left(\mathrm{e}^{\mathrm{i}\pi\tau n f_0} - \mathrm{e}^{-\mathrm{i}\pi\tau n f_0}\right) M(n,\tau) && (12.35)\\ &= \mathrm{i}2\,M(n,\tau)\sin(\pi n f_0 \tau)\,. && (12.36) \end{aligned}$$

C'est donc dire que $X^{(n)}(\tau) = 2\Re\left[\beta_n(\tau)\right]$ et $Y^{(n)}(\tau) = \mathrm{i}2\Im\left[\beta_n(\tau)\right]$.

De (12.33), (12.36) et (12.29), on a alors

$$S_\phi(\tau) = \beta_0(\tau) + \sum_{n=1}^{\infty} 2\,M(n,\tau)\Big(\cos(\pi n f_0\tau)\cos(n\phi) - \sin(\pi n f_0\tau)\sin(n\phi)\Big), \tag{12.37}$$

ce qui équivaut finalement à

$$\boxed{S_\phi(\tau) = \beta_0(\tau) + 2\sum_{n=1}^{\infty} M(n,\tau)\cos(n\phi + \pi n f_0\tau)}\,. \tag{12.38}$$

De (12.29), (12.33) et (12.36), il est clair que $S_\phi(\tau)$ est réel, ce qui est nécessaire pour qu'il corresponde bien à un corrélateur courant–courant. De plus, d'après la forme de (12.29) et en notant que $Y^{(n)}(\tau) \in \mathrm{i}\mathbb{R}$, on peut interpréter

les $X^{(n)}(\tau)$ et $Y^{(n)}(\tau)$ comme respectivement la réponse temporelle en phase et en quadrature avec l'excitation au temps τ et à l'harmonique n. L'utilisation de corrélateurs à deux temps [15] complique l'interprétation, mais on voit quand même qu'à τ donné, les $X^{(n)}(\tau)$ et $Y^{(n)}(\tau)$ représentent les contributions de $S_\phi(\tau)$ qui sont paires ou impaires en ϕ, c'est-à-dire en phase ou en quadrature avec l'excitation.

Les $X^{(n)}(\tau)$ et $Y^{(n)}(\tau)$ correspondent donc à la réponse harmonique en phase et en quadrature avec l'excitation dans le domaine temporel. Leurs représentations duales de Fourier $\tilde{X}^{(n)}(f)$ et $\tilde{Y}^{(n)}(f)$ sont respectivement ce qu'on appellera la réponse harmonique en phase et en quadrature à la fréquence f.

12.6.2 Résultats expérimentaux

La figure 12.14 présente les $\tilde{\beta}_n(f)$ obtenus expérimentalement pour les deux premières harmoniques, les réponses harmoniques en phase et en quadrature associées ainsi que les $\tilde{M}$ qui y correspondent. On présente les résultats pour les plus petites valeurs de V_{ac} et V_{dc} étudiées qui génèrent des résultats non triviaux, de manière à ce que les spectres soient ≈ 0 en approchant la limite $|f| = 10\,\text{GHz}$ de la bande passante. On évite ainsi les effets de fenêtrage lors du passage subséquent au domaine temporel, comme discuté à la section 7.5.1 — voir les coupures abruptes en bords de bande sur certaines courbes de la figure 12.12.

Pour assurer un résultat probant dans le domaine temporel, il faut *assainir* les données, c'est à dire éliminer ou reconstruire les composantes des $\tilde{\beta}_n(f)$ en dehors de la zone de validité de la calibration — voir la section 11.1.2. Le résultat de cette opération est indiqué aux sous-figures du haut par des points au centre blanc. À $n = 1$, on élimine donc les valeurs à $f > 10\,\text{GHz}$ et $f < -6\,\text{GHz}$, alors qu'à $n = 2$ ce sont celles à $f > 10\,\text{GHz}$ et $f < -2\,\text{GHz}$ qui sont décimées [16]. Pour ce qui est de la reconstruction [17], les données à 0 et 4 GHz dans le cas $n = 1$ sont suffisament lisses pour qu'on puisse les laisser telles quelles. Dans le cas $n = 2$, par contre, on reconstruit les valeurs autour de 0, 4, et 8 GHz — trois points

15. La phase $\phi = t/(2\pi f_0)$ n'est fondamentalement qu'une remise à l'échelle pratique de t.
16. À $n < 0$, la situation est simplement opposée par rapport à $f = 0$.
17. Qui est nécessaire parce que la bande passante n'inclut pas 0 Hz — voir la section 11.1.2.

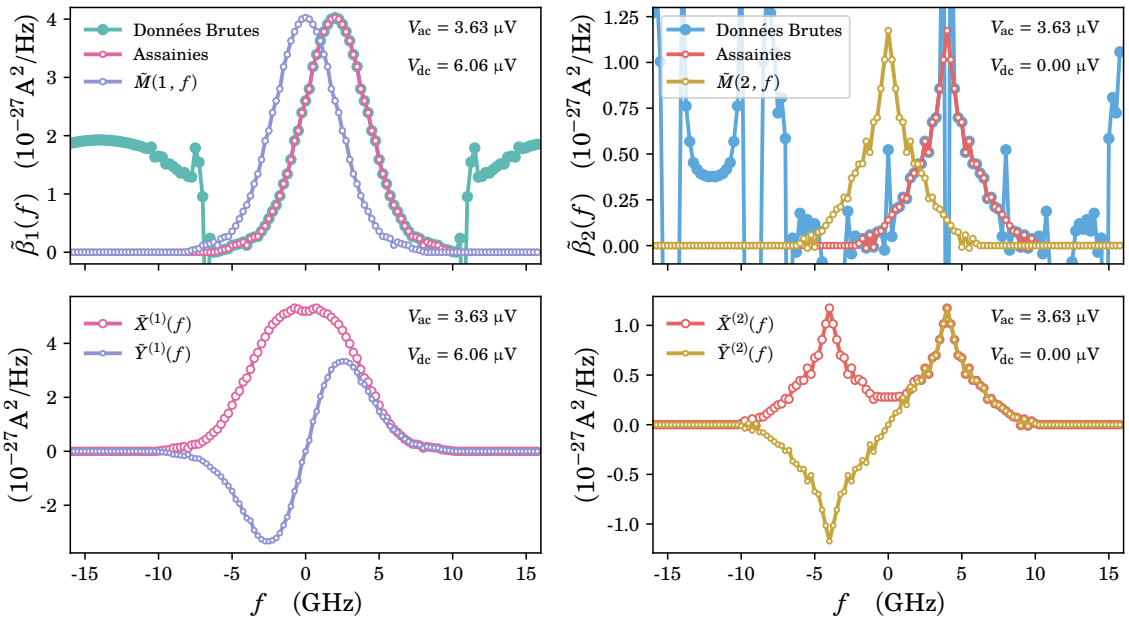


FIGURE 12.14 – Spectres de $\tilde{\beta}_n(f)$ et $\tilde{X}^{(n)}(f)$. Les résultats sont montrés bruts et après assainissement. On présente aussi les spectres décalés $\tilde{\beta}_n(f + nf_0/2)$. On rapelle que $\tilde{X}^{(n)}(f) = \tilde{\beta}_n(f) + \tilde{\beta}_n(-f)$, $\tilde{Y}^{(n)}(f) = \tilde{\beta}_n(f) - \tilde{\beta}_n(-f)$ et $\tilde{M} = \tilde{\beta}_n(f + nf_0/2)$.

pour chaque fréquence — en prolongeant linéairement la tendance des points adjacents. On élimine aussi les parties imaginaires parasites des $\tilde{\beta}_n$. L'intention ici est d'obtenir des $\tilde{\beta}_n(f)$ dans le domaine temporel qui soient le plus fidèles possible aux mesures tout en introduisant le moins d'artefacts possible lors du passage en τ par transformée de Fourier.

Aux sous-figures du bas de la figure 12.14, on voit la réponse harmonique déduite des $\tilde{\beta}_n(f)$ après l'assainissement des données discuté ci-haut. Les $\tilde{X}^{(n)}(f)$ sont bien paires en f alors que les $\tilde{Y}^{(n)}(f)$ sont impaires. On note aussi que $\tilde{Y}^{(n)}(0) = 0$ par construction ; le concept même de *réponse retardée* ou *hors phase* n'étant pas applicable dans ce cas. L'utilité de la procédure d'assainissement est évidente à grande fréquence, où les courbes tendent doucement et de manière lisse vers 0 aux limites de la bande passante, vers $|f| = 10\,\text{GHz}$.

On peut alors prendre les transformées de Fourier inverses des courbes de la

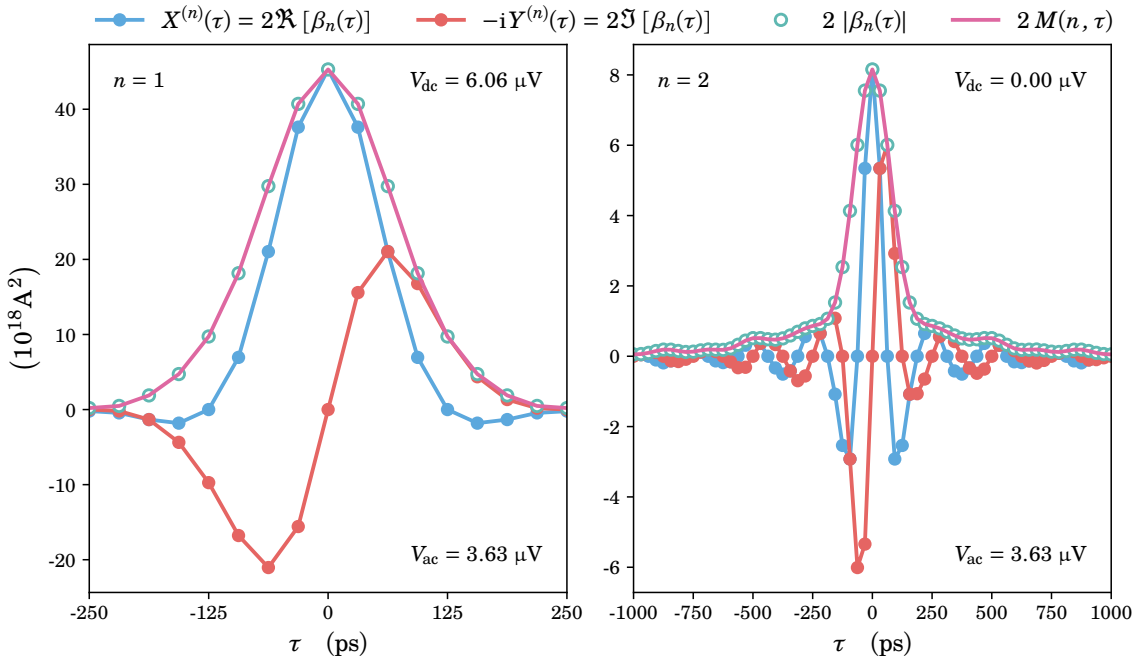


FIGURE 12.15 – Réponse harmonique dans le domaine temporel. On rappelle que $\Re\,[\beta_n(\tau)] = X^{(n)}(\tau)/2$ et $\Im\,[\beta_n(\tau)] = -\mathrm{i}Y^{(n)}(\tau)/2$.

figure 12.14 pour obtenir celles de la figure 12.15. On y omet les $\beta_n(\tau)$ puisque, dans cette représentation, leurs parties réelles et imaginaires sont sont simplement proportionnelles à $X^{(n)}(\tau)$ et $Y^{(n)}(\tau)$, respectivement. Le caractère pair et impair en τ de $X^{(n)}(\tau)$ et $-\mathrm{i}Y^{(n)}(\tau)$ est évident, confortant leur interprétation en tant que réponse en phase et en quadrature. La propriété (12.17) d'*enveloppage* de $M(n,\tau)$ est aussi frappante, celle-ci correspondant bien à $|\beta_n(\tau)|$. On note aussi que la *largeur* de l'enveloppe est plus grande [18] à $n = 2$ qu'à $n = 1$, et ce, qu'on s'intéresse aux pics centraux ou aux ailes. Cela n'est que la conséquence du principe d'incertitude [91] ; comme $\tilde{\beta}_2(f)$ est plus étroit que $\tilde{\beta}_1(f)$ dans le domaine fréquentiel, l'inverse se doit d'être vérifié dans le domaine temporel.

Ces résultats décrivent donc comment $S_\phi(\tau)$ répond à l'excitation aux deux premières harmoniques. Dans les conditions expérimentales présentées ici, on voit dans les deux cas qu'une partie de la corrélation est modulée en phase avec l'excitation — représentée par $X^{(n)}(\tau)$ — mais qu'une partie de la réponse est

18. Les abscisses ne couvrent pas la même période de τ.

en quadrature de phase avec celle-ci — capturée par $Y^{(n)}(\tau)$. La tendance des résultats est très similaire pour des valeurs de V_{ac} et V_{dc} similaires à celles sélectionnées. Cependant, en augmentant notablement ces tensions, les effets de fenêtrage dus aux bords de la bande passante se font sentir. Dans ce cas, $X^{(n)}(\tau)$ tend de plus en plus vers une fonction de type sinc alors que la réponse en quadrature disparaît, en accord avec les attentes pour le régime de calibration. Des versions des figures 12.14 et 12.15 dans ce régime sont disponibles à l'annexe A.2.

12.6.3 Familiarité avec la susceptibilité de bruit

La formulation (12.38) montre aussi que, en général, la réponse du bruit à l'excitation ne sera pas purement en phase avec celle-ci. En effet, pour que $S_\phi(\tau)$ soit purement en $\cos(n\phi)$ quelque soit τ, il faut que $M(n,\tau)$ agisse à la manière d'un delta de Dirac [19], soit

$$M(n,\tau) = M(n)\,\delta(\tau)\,. \tag{12.39}$$

En injectant cette forme dans (12.33) et (12.36), cela implique que $X^{(n)}(\tau)$ agit aussi comme un delta de Dirac et que $Y^{(n)}(\tau)$ est nul. Dans le domaine fréquentiel, cette situation correspond à $\tilde{X}^{(n)}(f)$ constant sur toute la bande de fréquence de la mesure — un buit *blanc* — alors que $\tilde{Y}^{(n)}(f)$ y est nul. Ces deux conditions sont simultanément respectées si les $\tilde{\beta}_n(f)$ sont blancs, comme dans le régime de calibration à grand V_{dc}. En effet, la figure 12.12 montre bien que $\tilde{\beta}_1(f)$ tend de plus en plus vers un bruit blanc avec V_{dc} croissant, alors que $\tilde{\beta}_2(f)$ s'estompe vers 0 ; des niveaux constants dans les deux cas.

Dans ce régime, (12.39) est respectée [20] et la réponse est bien purement en phase avec l'excitation, soit $S_\phi(\tau) = \beta_0(\tau) + 2M(1)\,\delta(\tau)\cos(\phi)$. On obtient alors $\tilde{S}_\phi(f) = eV_{\text{dc}}/R + 2M(1)\cos\phi$ en prenant la transformée de Fourier de $S_\phi(\tau)$ et en appliquant la limite grande tension (10.6), soit un résultat similaire à celui de l'équation (11.13) attendue pour le régime de calibration. Via (12.39),

19. Expérimentalement, on va mesurer $\delta(\tau)$ convolué avec la transformée de Fourier de la bande passante de la mesure, soit la transformée elle-même.
20. Avec $M(n) = 0\ \forall\ n \neq 1$.

(12.33) et (12.36), on trouve aussi $\tilde{X}^{(n)}(f) = 2M(1)$ et $\tilde{Y}^{(n)}(f) = 0$. Donc, dans le régime de calibration, on doit avoir $\tilde{X}^{(n)}(f) = eV_{\mathrm{ac}}/R$ de manière à ce que $\tilde{S}_\phi(f) = \frac{e}{R}(V_{\mathrm{dc}} + V_{\mathrm{ac}}\cos\phi)$.

En dehors du régime de calibration, on aura alors $\left|\tilde{X}^{(n)}(f)\right| < e\,|V_{\mathrm{ac}}|/R$ et $\tilde{Y}^{(n)}(f) \neq 0$ en général. En ce sens, $\tilde{X}^{(n)}(f)$ et $\tilde{Y}^{(n)}(f)$ agissent comme un type de susceptibilité de bruit à l'harmonique n, associés respectivement à la réponse instantanée et à celle retardée. Il est donc intéressant de regarder ces quantités en fonction de V_{dc} à la manière de ce qui est fait à la section 12.5.

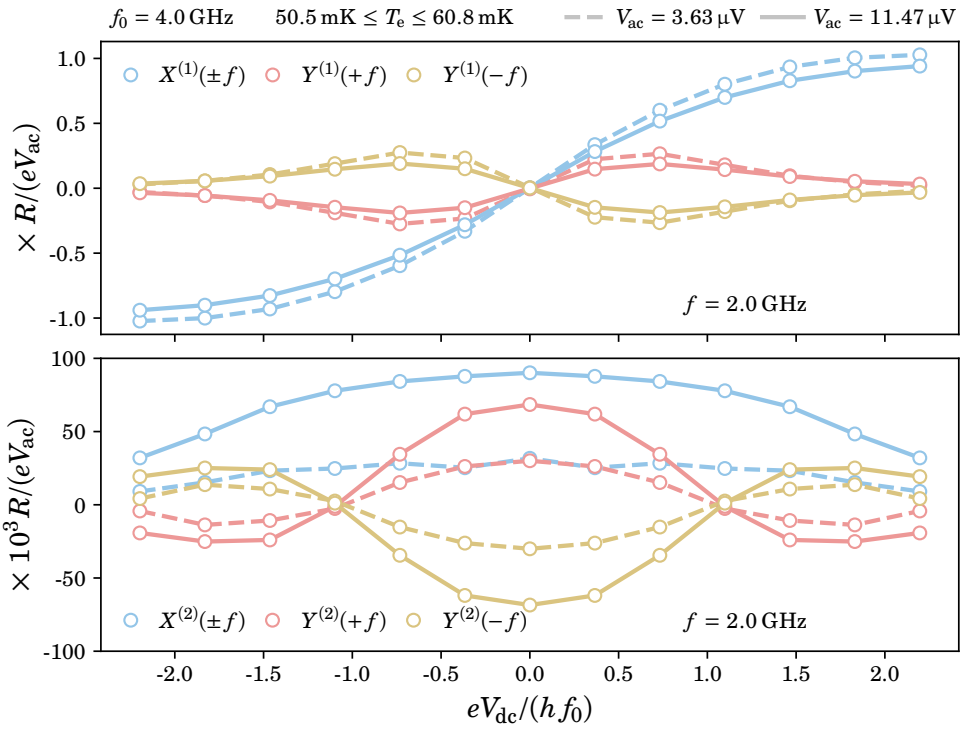


FIGURE 12.16 – Réponse harmonique sous forme similaire à la susceptibilité de bruit. Ces courbes sont obtenues en combinant celles de la figure 12.13.

En recombinant les résultats de $\tilde{\beta}_n(f)$ de la figure 12.13 de manière à obtenir des $\tilde{X}^{(n)}(f)$ et $\tilde{Y}^{(n)}(f)$ et en les normalisant par eV_{ac}/R, on obtient les résultats de la figure 12.16. La tendance vers le régime instantané à grand V_{dc} y est mise en valeur par la normalisation, avec $\tilde{X}^{(1)}(f) \times R/(eV_{\mathrm{ac}}) \to \pm 1$ et les autres réponses tendant vers 0. Les courbes rouges et jaunes montrent que $\tilde{Y}^{(n)}(f)$

est impaire en f et que, contrairement aux courbes de la figure 12.13, toute l'information est contenue dans les réponses à n positif.

Ces résultats s'interprètent essentiellement comme ceux de la figure 12.13 en termes de temps caractéristique τ_e entre événements tunnels électroniques ajusté par V_{dc} et modulé par V_{ac}, mais l'ambiguïté quant à l'interprétation des fréquences et harmoniques négatives est levée.

On remarque aussi que la réponse à l'harmonique fondamentale $n = 1$ est impaire en V_{dc} alors que celle à l'harmonique supérieure est paire. En effet, l'équation (7.160) montre que changer le signe de V_{dc} correspond au décalage $\phi \longrightarrow \phi + \pi$ et donc, puisque la n^{e} harmonique a une période n fois moindre que la fondamentale, prendre l'opposé de V_{dc} y aura comme effet $n\phi \longrightarrow n\phi + n\pi$. À $n = 1$ la réponse est donc bien décalée de π, ce qui correspond au changement de signe de la réponse harmonique, alors qu'à $n = 2$ le décalage de 2π laisse la réponse harmonique inchangée.

Il est aussi intéressant de regarder l'amplitude de la réponse en fonction des deux valeurs de V_{ac} étudiées. À $n = 1$, on voit que les réponses en phase et en quadrature sont plus grandes pour des valeurs de V_{ac} faible, alors que c'est l'inverse dans le cas $n = 2$.

C'est ce dernier cas qui a l'interprétation la plus éloquente. Comme discuté à la section 12.5, la réponse à $n = 2$ est maximale à $V_{\mathrm{dc}} = 0$ puisque la jonction est insensible au signe de la polarisation [21]. Dans cette situation, elle voit donc les deux *lobes* de l'excitation comme une augmentation de tension et sa réponse est au double de la fréquence de cette dernière ; purement à l'harmonique supérieure. Plus V_{ac} est grand, plus la jonction voit des temps caractéristiques [22] τ_e courts pendant la période d'excitation, sans pour autant s'approcher de la réponse instantanée. Cela explique donc l'augmentation de l'amplitude de la réponse à la fois en phase et en quadrature avec V_{ac} croissant ; le bruit répond à l'excitation sans la suivre fidèlement tout au long de la période.

La situation est un peu l'inverse dans le cas de la première harmonique où la réponse est plus grande à $V_{\mathrm{dc}} \neq 0$. Un grand V_{ac} implique alors de *visiter* au

21. En effet, $\tilde{S}^{\mathrm{dc}}(\pm V_{\mathrm{dc}}) = \tilde{S}^{\mathrm{dc}}(|V_{\mathrm{dc}}|)$ pour une jonction polarisée en tension continue.
22. L'interprétation en termes de temps caractéristique τ_e est étayée à la section 12.5.

cours d'une période d'excitation des temps caractéristiques notablement plus lents que celui associé à V_{dc}. De plus, comme τ_e est inversement proportionnel à $V(t)$, $V_{\mathrm{dc}} + V_{\mathrm{ac}}$ et $V_{\mathrm{dc}} - V_{\mathrm{ac}}$ n'ont pas des effets symétriques sur celui-ci et τ_e est globalement augmenté [23] ; la réponse est moindre.

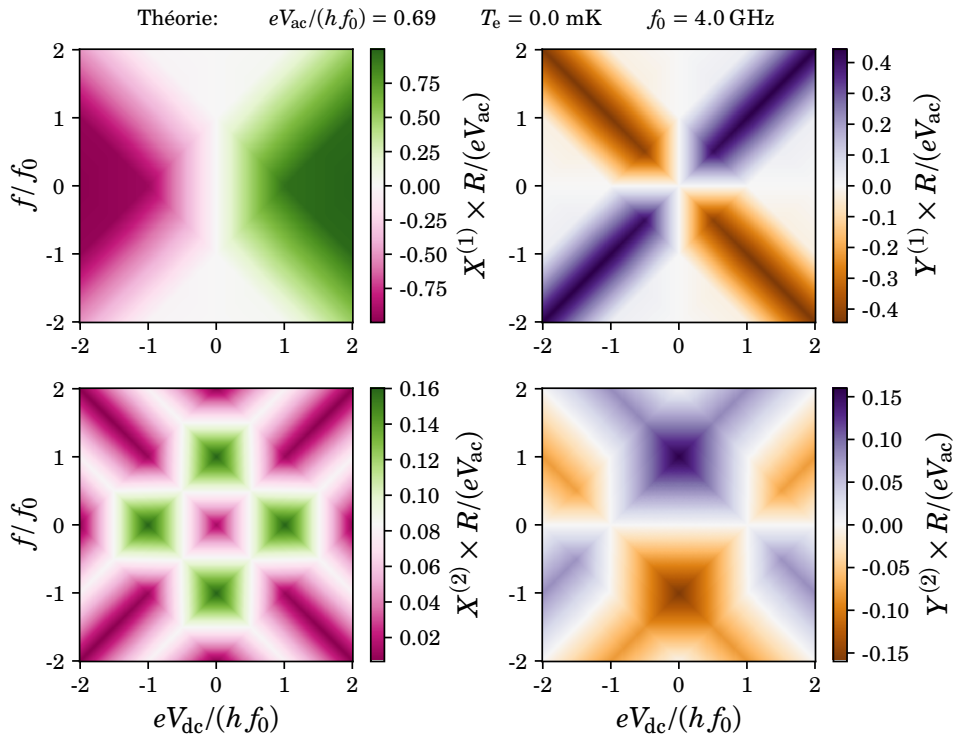


FIGURE 12.17 – Réponse harmonique théorique à température nulle. Formes théoriques pour $\tilde{X}^{(n)}$ et $\tilde{Y}^{(n)}$ à $T_{\mathrm{e}} = 0$ et au plus grand V_{ac} de la figure 12.16. Le f correspondant à cette figure est $f/f_0 = 1/2$. L'effet d'un $T_{\mathrm{e}} \neq 0$ est essentiellement d'ajouter un flou sur les prédictions.

On remarque de plus que la réponse en quadrature à la seconde harmonique a un zéro près de $eV_{\mathrm{dc}} = hf_0$, ce qui n'est certainement pas un hasard. En fait, les prévisions théoriques de la réponse harmonique en fonction de V_{dc} et f à V_{ac} suffisamment faible et pour $T_{\mathrm{e}} = 0$, disponibles à la figure 12.17, présentent des symétries remarquables selon V_{dc} et f.

En résumé, la réponse harmonique discutée ici permet de caractériser l'effet

23. La situation est similaire à $\frac{1}{1+x}$ avec x centré à 0 et symétrique, où on montre simplement par série de Taylor que $\frac{1}{1+\langle x\rangle} = 1$ alors que $\left\langle \frac{1}{1+x} \right\rangle = 1 + \langle x^2 \rangle + \langle x^4 \rangle + \ldots$, donc $\left\langle \frac{1}{1+x} \right\rangle > \frac{1}{1+\langle x\rangle}$.

que l'excitation à $\tilde{S}_\phi(f)$ via des termes en phase et en quadrature avec l'excitation, de manière à ce que toute l'information soit contenue aux fréquences et harmoniques positives. Elle offre une interprétation intéressante, similaire à la susceptibilité de bruit, qui explique les différentes contributions à la structure de $\tilde{S}_\phi(f)$.

Chapitre 13

Synthèse : mesures de bruit ultrarapides

En somme, on a étudié les fluctuations émises en bande large par une jonction tunnel dans différentes situations expérimentales à l'aide de mesures ultrarapides dans le domaine temporel. L'approche numérique utilisée est très versatile, permet d'étudier une grande variété de quantités à l'aide d'un seul montage expérimental et permet de tirer profit de la synchronisation entre l'excitation et la mesure pour ajuster la phase de détection et ainsi regarder les corrélations de manière synchrone avec l'excitation.

D'abord, cette approche nous a permis d'obtenir la densité spectrale de bruit résolue en fréquence sur toute la bande de détection. À chaque condition expérimentale donnée, les résultats de densité spectrale de bruit $\tilde{S}(f)$ dans le domaine fréquentiel sont ainsi obtenus d'un seul coup pour toutes les fréquences résolues par la mesure. Ces résultats de $\tilde{S}(f)$ mettent en lumière un effet de chauffage par V_{dc} et V_{ac} qui est normalement présumé minime, mais qui est ici suffisamment important pour qu'on ait à en tenir compte lors des comparaisons aux prédictions théoriques. Bien que notre approche y soit particulièrement sensible — toutes les données desquelles la température électronique est extraite correspondent aux mêmes V_{dc} et V_{ac} — il est possible que l'amplitude surprenante de cet effet soit due à l'échantillon particulier utilisé ici. Il serait donc pertinent de refaire ces mesures à l'aide d'une jonction tunnel plus traditionnelle pour

confirmer que cet effet de chauffage est réellement négligeable normalement, et s'assurer qu'il ne soit pas seulement masqué par l'étroitesse des bandes de mesures utilisées typiquement. Dans tous les cas, les mesures en bande large résolues en fréquence semblent particulièrement bien adaptées à l'extraction de la température T_e des électrons via le bruit de grenaille.

Ensuite, ces résultats nous ont permis d'obtenir le bruit en excès DC, qui présente des oscillations quantiques en fonction de V_dc et démontrent que la polarisation en tension continue — qui est constante — module au cours du temps, et de manière importante, les corrélations au sein des fluctuations. Du même ensemble de données, on reproduit aussi aisément les résultats de [52] quant au bruit thermique en excès — les oscillations de Pauli–Heisenberg — malgré la relative simplicité de notre montage expérimental et de la procédure d'acquisition. On explique même les effets de dépassement d'enveloppe thermique, qui n'étaient pas abordés par cette étude, à l'aide de l'effet de chauffage susmentionné [1].

De plus, en tirant profit de la synchronisation entre l'excitation et la mesure, on a introduit le concept de spectre $\tilde{S}_\phi(f)$ résolu en phase et en fréquence, une quantité qui contient la même information que la distribution de Wigner du signal étudié. Ces spectres, bien qu'*a priori* difficiles à interpréter directement, nous ont permis de corroborer expérimentalement les prédictions théoriques de [51] à travers le bruit en excès de phase. Les spectres résolus en phase se décomposent aussi par analyse de Fourier en une pléthore de corrélateurs courant–courant, que ce soit dans le domaine temporel ou fréquentiel. On a établi le lien entre ceux-ci et les corrélations responsables de la compression d'état au sein du bruit de grenaille [61], l'émission de paires de photons corrélée [44, 46] et la susceptibilité de bruit [138]. La décomposition des $\tilde{S}_\phi(f)$ sous forme de réponse harmonique nous donne quant à elle accès à des corrélateurs courant–courant directement dans le domaine temporel, en plus de séparer la réponse des fluctuations en une partie instantanée et en une partie retardée comparativement à l'excitation de manière analogue à une susceptibilité.

Ces résultats ouvrent aussi la voie à de nouvelles avenues d'études. Par

1. Ce qui motive d'autant plus la vérification expérimentale de cet effet sur des échantillons typiques.

exemple, la partie retardée de la réponse harmonique est conceptuellement similaire à une réponse dissipative. Il serait donc intéressant de l'étudier plus en détail pour vérifier si un résultat similaire au théorème fluctuation–dissipation [2] [139 ; 81, §15.7] viendrait relier les corrélateurs d'ordres supérieurs à celle-ci et, *in fine*, à la partie imaginaire de $\tilde{S}_\phi(f)$. De plus, le lien entre la distribution de Wigner du signal et celle de l'état quantique sous-jacent — la tomographie de l'état quantique — mériterait d'être élucidé pour bien cerner toute l'information contenue dans les spectres résolus en phase. Aussi, comme discuté à la partie I de cette thèse, l'approche par traitement numérique directement sur la trace expérimentale serait utile pour implémenter des schémas de détection originaux, comme celui utilisé à la référence [43] quant à la statistique de photons de l'amplificateur paramétrique Josephson. L'approche expérimentale adoptée ici serait aussi idéale pour étudier des phénomènes quantiques purement large bande s'exprimant plus naturellement selon une première quantification dans le temps plutôt qu'en fréquence, comme abordé théoriquement à la référence [140]. Finalement, la calibration résolue en phase introduite au chapitre 11 pourrait être cruciale à l'étude résolue en phase de différents systèmes, la jonction tunnel pouvant être utilisée comme échantillon de référence si nécessaire.

En conclusion, l'étude des fluctuations par mesures en bande large dans le domaine temporel offre une quantité considérable d'information. Que ce soit l'émission de paires de photons, la réponse des fluctuations à l'excitation, les corrélateurs courant–courant ou simplement la densité spectrale résolue en fréquence, cette approche semble adaptée à l'étude d'une panoplie de phénomènes aussi intéressants que disparates ; elle présente une profonde richesse. Avec l'avènement des technologies quantiques et les appareils de mesures qui s'améliorent sans cesse, les mesures large bande dans le domaine temporel auront définitivement l'occasion de faire valoir leur utilité pour le traitement des signaux quantiques dans un futur pas si lointain.

2. Un théorème *fluctuation-de-fluctuations—dissipation-de-fluctuations* pour ainsi dire.

Chapitre 14

Conclusion

En résumé, cette thèse présente deux études sur les fluctuations émises par des composantes mésoscopiques dans le régime quantique utilisant le traitement des signaux pour extirper l'information pertinente des mesures brutes ; nommément l'étude de la statistique de photons d'un amplificateur paramétrique Josephson et celle de la jonction tunnel dans le régime photoexcité, effectuée à l'aide de mesures dans le domaine temporel en bande large

En étudiant l'amplificateur paramétrique Josephson, on a confirmé que le signal à sa sortie correspondait bel et bien au vide comprimé, en plus de mettre en lumière l'importance de la procédure de détection sur les résultats. Du côté des mesures dans le domaine temporel, on a défini le concept de spectre résolu en phase, une quantité qui contient une grande quantité d'information sur les corrélations au sein du signal. De ces spectres, on a aussi dérivé la réponse harmonique des fluctuations à l'excitation, qui capture la réponse du bruit à l'excitation en incluant les effets de retards. Tous ces résultats mettent aussi en valeur la pertinence d'adapter la calibration au montage expérimental, à l'échantillon étudié et au type de mesures entreprises.

Sommes toutes, les travaux présentés ici démontrent bien la variété des applications et la flexibilité expérimentale offerte par le traitement des signaux quantiques. Ils ouvrent la voie à l'étude des signaux quantiques large bande et démontrent l'importance de la méthode de détection. Puisque le traitement numérique des données permet d'ajuster la détection sans changer le montage

expérimental, il serait prometteur de combiner les deux approches présentées ici pour étudier les amplificateurs bande large comme le TWPA [31], pour mesurer des spectres de compression d'état [141] ou encore pour étudier la compression d'état dans le domaine temporel.

Avec les technologies quantiques qui sont en plein essor, le traitement des signaux quantiques a le potentiel de devenir un outil incontournable du traitement de l'information ; ce n'est que la pointe de l'iceberg.

Annexe A

Matériel Supplémentaire

A.1 Spectres résolus en phase $\tilde{S}_\phi$

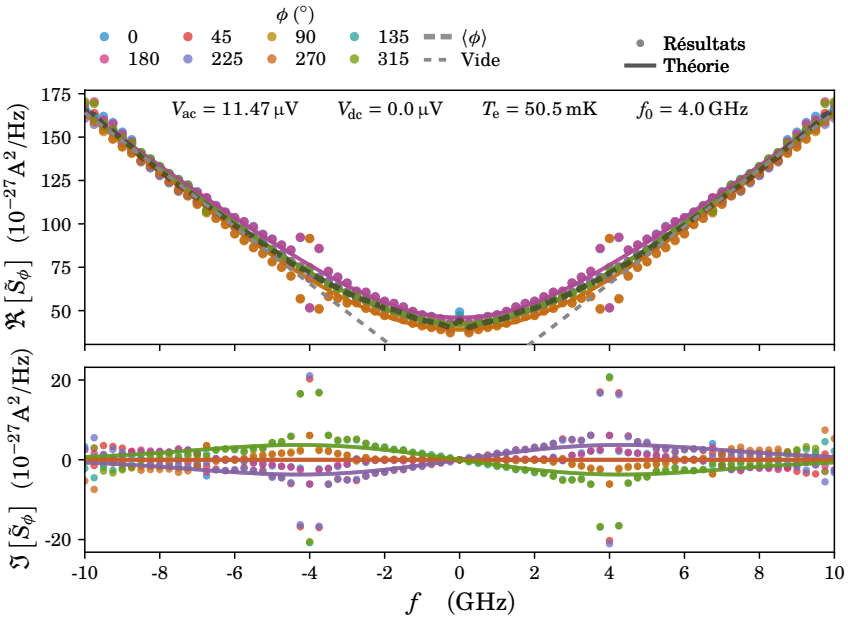


FIGURE A.1 – Résultats de $\tilde{S}_\phi$ en fonction de f à $V_{dc} = 0$, sur toute la bande passante.

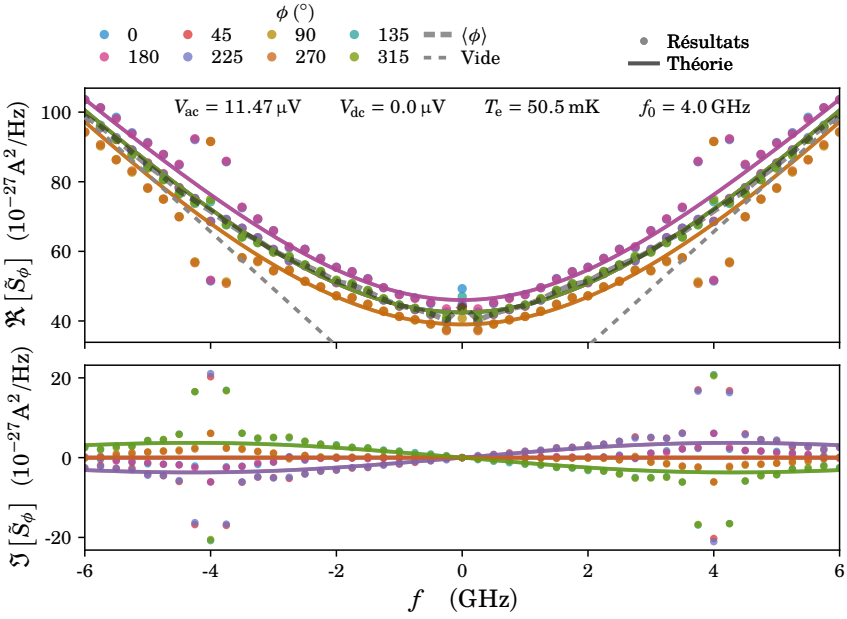


FIGURE A.2 – Résultats de $\tilde{S}_\phi$ en fonction de f à $V_{dc} = 0$, sur la zone de validité de la calibration.

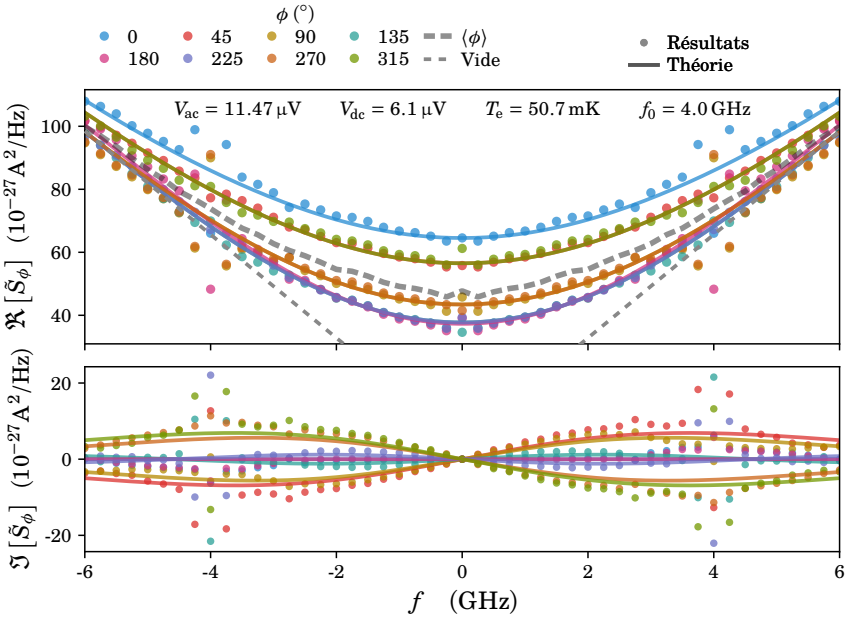


FIGURE A.3 – Résultats de $\tilde{S}_\phi$ en fonction de f

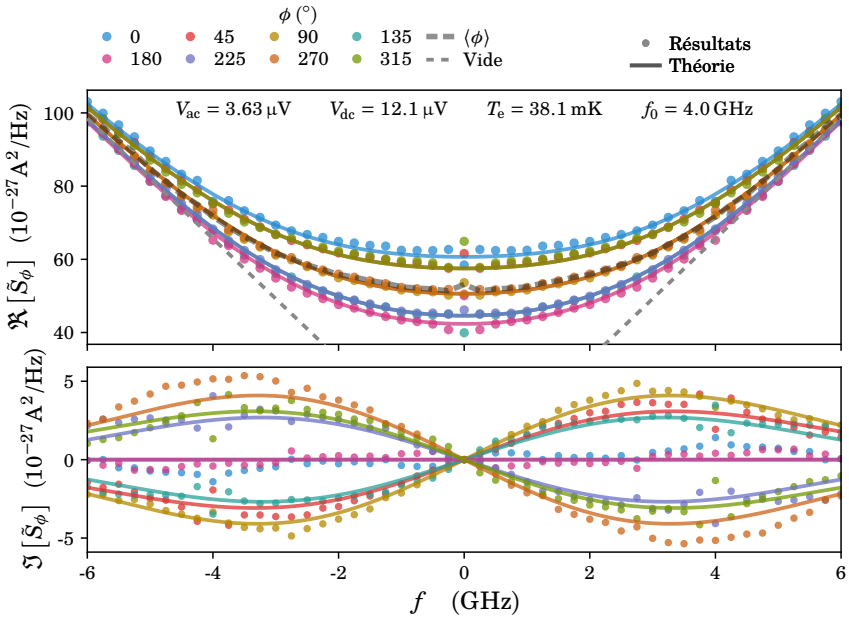


FIGURE A.4 – Résultats de $\tilde{S}_\phi$ en fonction de f et théorie associée. La température électronique utilisée pour les courbes théoriques est déduite de la figure 11.3.

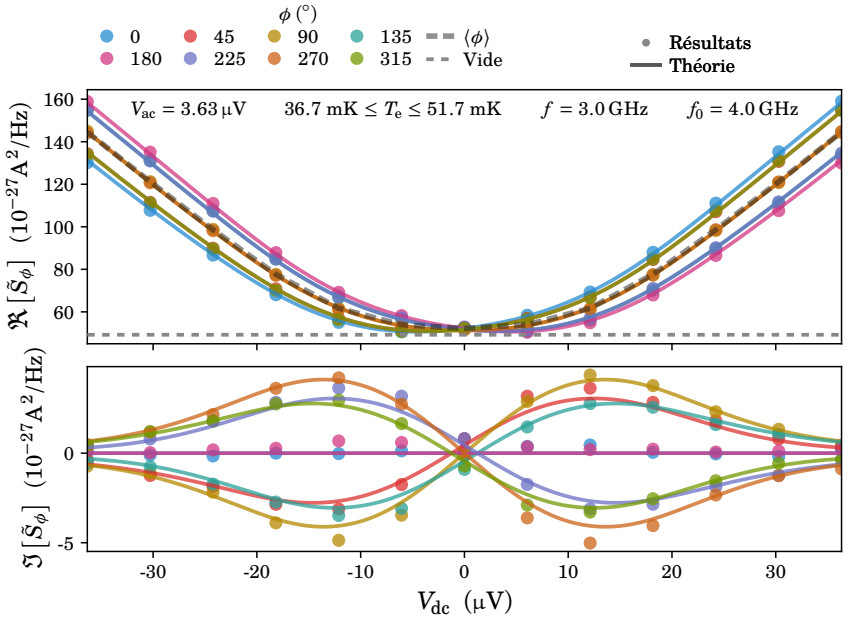


FIGURE A.5 – Résultats de $\tilde{S}_\phi$ en fonction de V_{dc} à $V_{ac} = 3.63\,\mu V$ et $f = 3\,GHz$.

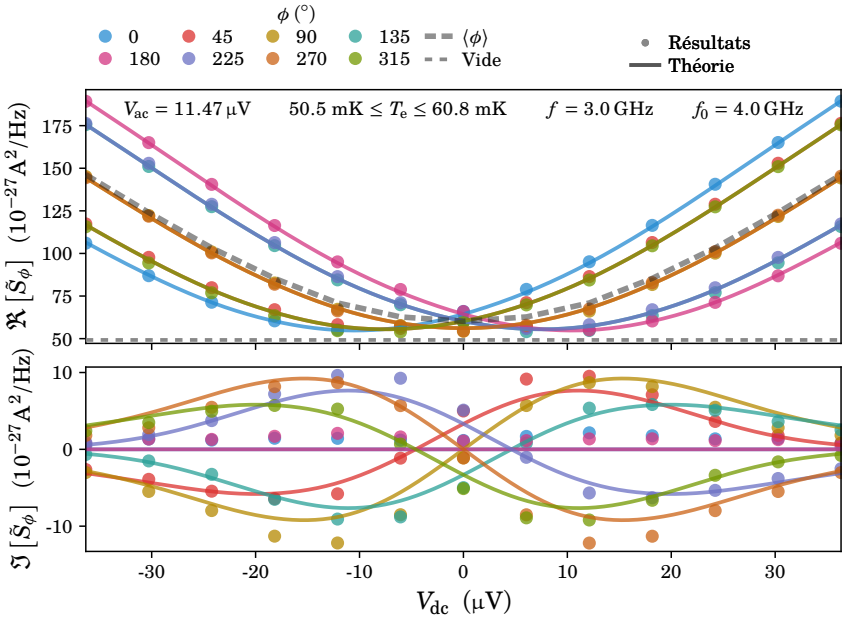


FIGURE A.6 – Résultats de $\tilde{S}_\phi$ en fonction de V_{dc}, à $V_{ac} = 11.47\,\mu V$ et $f = 3\,GHz$.

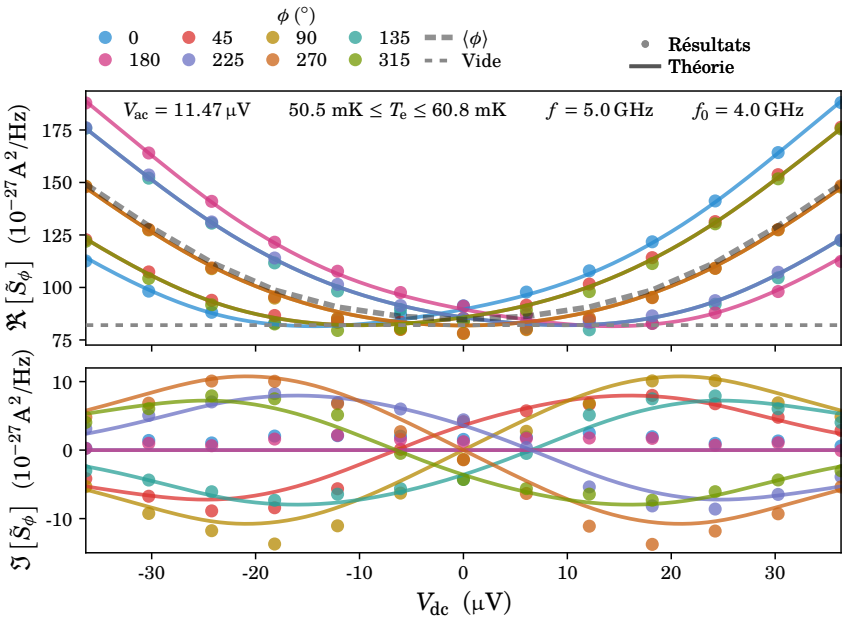


FIGURE A.7 – Résultats de $\tilde{S}_\phi$ en fonction de V_{dc}, à $V_{ac} = 11.47\,\mu V$ et $f = 5\,GHz$.

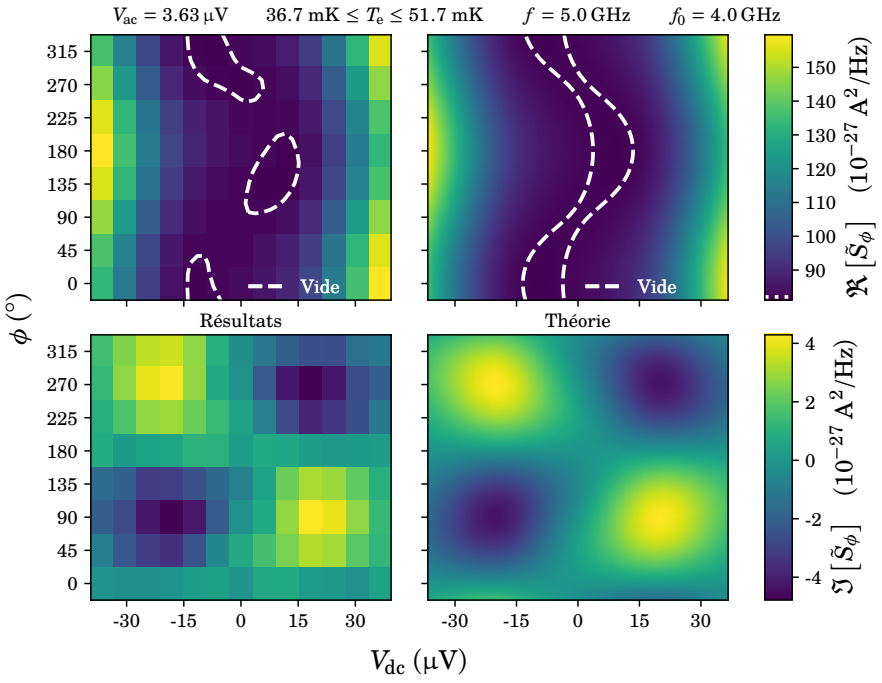
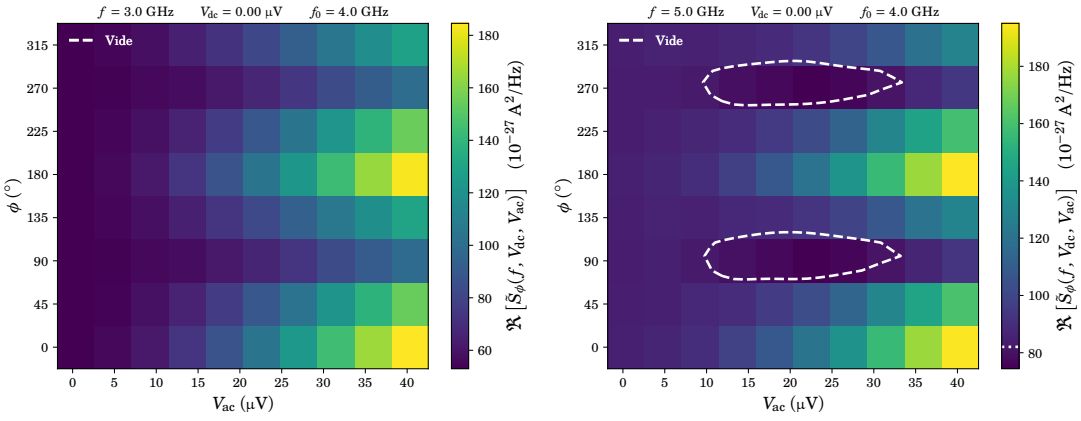


FIGURE A.8 – Cartographie de $\tilde{S}_\phi$ en fonction de V_{dc} et ϕ à $V_{ac} = 3.63\,\mu V$ et $f = 5\,GHz$.



(a) (b)

FIGURE A.9 – Cartographie de $\tilde{S}_\phi$ en fonction de V_{ac} et ϕ à $V_{dc} = 0$, pour $f = 3\,GHz$ et $f = 5\,GHz$.

A.2 Réponse Harmonique

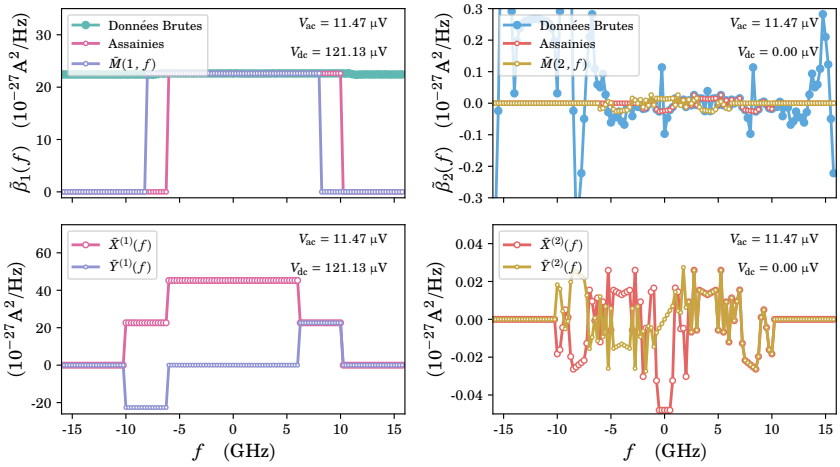


FIGURE A.10 – Spectres de $\tilde{\beta}_n(f)$ et $\tilde{X}^{(n)}(f)$. Régime de calibration.

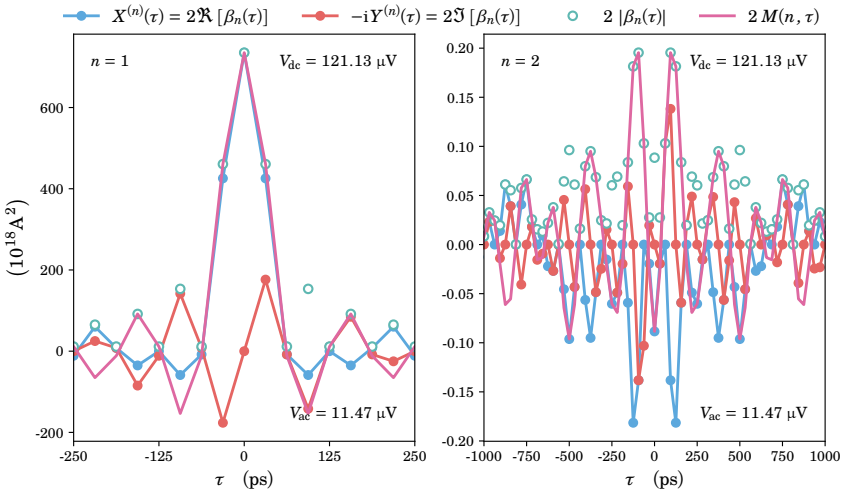


FIGURE A.11 – Réponse harmonique temporelle. Régime de calibration.

Annexe B

Traitement des données à la volée

B.1 Histogrammes : algorithme et interface

B.1.1 Description générale

Dresser l'histogramme d'un signal correspond, conceptuellement, à établir des éventails [1] de valeurs parmi toutes celles possible pour chaque échantillon et de compter, pour chaque éventail, le nombre de valeurs mesurées lui correspondant. Dans le cas qui nous intéresse ici, soit celui de mesures digitales discrètes représentées par des entiers, on peut simplement associer un éventail à chaque valeur possible lors de l'acquisition. Cela nécessite un entier en mémoire pour chaque valeur possible du signal, et cet entier doit être de taille assez grande pour éviter les dépassements d'entier, ou *integer overflow*.

Notons que si la résolution complète du type de données d'entrée n'est pas nécessaire, on peut simplement fusionner les éventails adjacents de manière récursive jusqu'à l'obtention de la résolution voulue. Cela correspond à ignorer les bits les moins significatifs des données d'entrée. On se limitera ici à cette situation d'éventails uniformes, bien qu'il soit possible en général de définir des éventails à taille variable.

Un histogramme construit à partir d'un grand nombre d'échantilllons va donc tendre, à la normalisation près, vers la densité de probabilité de celui-ci. Il permet ainsi le calcul de plusieurs propriétés statistiques du signal.

1. Traductuion libre de l'anglais *bin* typiquement utilisé dans le domaine.

B.1.2 Description des algorithmes

Soit l'élément de vecteur x_i représentant les données d'une acquisition de N points à B bits de résolution, où l'indice $i \in \mathbb{N}$ représente l'index de l'échantillon au sein de la série temporelle, de 0 à $N-1$. Soit aussi un élément de vecteur h_b, avec $0 \leq b < 2^B$, servant à représenter l'histogramme du signal x_i avec B bits de résolution. Notons que les données sont toujours interprétées comme non-signées tel que $0 \leq x_i < 2^B \; \forall i$; l'ordes des éventails est ajusté en fin de calcul le cas échéant. Partant initialement de $h_b = 0 \; \forall b$, on dresse l'histogramme des x_i en les traversant et en répétant l'incrémentation suivante pour chaque échantillon,

$$h_{x_i} \coloneqq h_{x_i} + 1 \qquad \forall i \,, \tag{B.1}$$

où $\coloneqq$ est l'opération d'assignation, et non l'égalité, si bien que $a \coloneqq a + b$ représente l'incrémentation de la variable a par une valeur b ; ce n'est pas une équation équivalente à $b = 0$. Dans le cas 2D, avec deux éléments de vecteurs x_i et y_i, h aura plutôt une taille de 2^{2B} et sera construit par l'incrémentation

$$h_{x_i,y_i} \coloneqq h_{x_i,y_i} + 1 \,, \tag{B.2}$$

où l'histogramme 2D est en pratique représenté en mémoire par un élément de vecteur 1D tel que

$$h_{i,j} \equiv h_{2^B i+j} = h_m. \tag{B.3}$$

Tel que discuté à la description générale ci-haut, il est possible de dresser un histogramme avec une résolution moindre que celle des données d'entrée en ignorant les bits les moins significatif. Pour éliminer β bits de résolution, il suffit en effet d'effectuer l'opération

$$x_i \coloneqq x_i >> \beta \,, \tag{B.4}$$

avec l'opération $>>$ représentant le décalage de bits vers la droite, avant d'appliquer les règles d'incrémentation ci-haut. Comme les routines travaillent avec des entiers non-signés à l'interne, cette opération est toujours valide et équivalente à une division entière par 2^β [142, §6.5.7]. Par exemple, partant de 2^B éventails de largeur 1 séparés de 1, soit $\{0 \leq x_i < 1\}$, $\{1 \leq x_i < 2\}$, $\cdots$, $\left\{2^B - 1 \leq x_i < 2^B\right\}$, on peut obtenir l'équivalent de $2^3 = 8$ fois moins d'éventails de largeur 8 séparés de 8 via un décalage vers la droite de $\beta = 3$, soit $\{0 \leq x_i < 8\}$, $\{8 \leq x_i < 16\}$, $\cdots$, $\left\{2^{B-3} - 2^3 \leq x_i < 2^{B-3}\right\}$. Avec une résolution initiale de 16 bit ou $2^{16} = 65\,536$ valeurs, on a donc une résolution finalement de 13 bit ou $2^{13} = 8\,192$ valeurs après le décalage de $\beta = 3$.

Un histogramme avec B bits de résolution et de dimension D correctement établi vérifie donc

$$N = \sum_{i=0}^{2^{DB}-1} h_i \,, \tag{B.5}$$

dans le cas contraire, toutes les données n'ont pas été traitées.

D'un histogramme 1D bâtit avec des données non signées, il est donc possible d'obtenir le k^{e} moment m'_k via

$$m'_k = \frac{1}{N} \sum_{i=0}^{2^B-1} h_i \, i^k \,, \tag{B.6}$$

où le lien entre h_i/N et la densité de probabilité est frappant. De même le k^{e} moment centré m_k se calcule via

$$m_k = \frac{1}{N} \sum_{i=0}^{2^B-1} h_i \left(i - m'_1\right)^k \,, \tag{B.7}$$

et le k^{e} cumulant c_k selon la formule de récursion [143–145]

$$c_k = m'_k - \sum_{i=1}^{k-1} \binom{k-1}{i} c_{k-i} \, m'_i \,, \tag{B.8}$$

qu'on peut écrire, en inversant l'orde de la somme via la substitution $i \longrightarrow k - i$,

$$c_k = m'_k - \sum_{i=1}^{k-1} \binom{k-1}{i-1} c_i \, m'_{k-i} \,, \tag{B.9}$$

avec le coefficient binomial $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

Dans le cas de données signées, le i de l'équation (B.6) ne balaie pas le bon éventail de valeurs [2], ce qui produit un résultat erroné sans pour autant affecter la validité des équations centrées (B.7) à (B.9) puisqu'elles sont insensibles à la valeur moyenne. Comme on s'intéresse en pratique aux moments centrés et aux cumulants — qui sont toujours *centrés* par construction — on n'implémente pas de fonction corrigée pour les moments non-centrés ; seuls les moments centrés et les cumulants devraient être calculés à l'aide des fonction fournies, par souci

2. Il devrait plutôt balayer l'éventail $-2^{B-1} \leq i < 2^{B-1}$.

d'exactitude.

Remarquons que le nombre d'opérations pour obtenir les moments ou cumulants dépend de la résolution B de l'histogramme, mais pas de la taille N des données traitées. De plus, l'opération ++ d'*incrémentation par 1* utilisée à l'interne pour construire l'histogramme est beaucoup plus rapides que les opérations arithmétiques typiques d'addition et de multiplications. Il est donc drastiquement plus rapide de construire un histogramme et d'en calculer les moments que de calculer les moments directement en traversant les données en mémoire.

B.1.3 Code et interface

Le code écrit pour dresser les histogrammes est disponible à l'annexe B.2.1. Il est rédigé en `C` pour être rapide et une interface en `Python`[3], via la librairie `ctypes`, permet de l'utiliser directement dans `PyHegel`. Le `C` est choisi plutôt que le `C++`, parce que les optimisations différentes faites pour chaque type de donnée supporté — soit `uint8`, `int8`, `uint16` et `int16` — rendent peu pertinente l'utilisation des *templates*. Par souci de performance, le code est parallélisé via l'interface de programmation `OpenMP` [98], principalement à travers des directives de préprocesseur. Le code est très efficace, étant plusieurs fois plus rapide que l'acquisition dans la majorité des cas, les histogrammes 2D à grande résolution[4] étant l'exception.

Le code en `C` fourni des fonctions pour dresser des histogrammes 1D et 2D pour des signaux d'entrées de 8 ou 16 bit, signés ou non. Ces fonctions sont accessibles via `Python` à travers les foncions `hist1dNbits` et `hist2dNbits` — voir le code en annexe pour les détails d'utilisation. Les données signées sont interprétées comme des données non-signées et une correction est appliquée à la toute fin pour reclasser l'histogramme en ordre croissant des valeurs représentées par son index. Ces fonctions peuvent aussi prendre en entrée un histogramme déjà entammé afin de continuer l'accumulation de données subséquentes dans le même objet en mémoire, par exemple dans le but d'effectuer une moyenne sur les réalisations. Le résultat est alors strictement équivalent à sommer les histogrammes effectués pour chaque réalisation. Dans le cas d'échantillons 16 bit, il est possible de limiter le nombre d'éventails — la résolution — de l'histogramme en ignorant les bits les moins significatifs de chaque échantillon.

Pour éviter les dépassements d'entier, on utilise des entiers 64 bit comme accumulateurs, ce qui permet de traiter au moins $2^{64} - 1 \approx 18.4 \times 10^{18}$ échan-

3. Voir le fichier `histograms_otf.py` à l'annexe B.2.1.
4. Typiquement plus de 11 bit ; variable selon la situation.

tillons — dans le pire des cas — avant qu'un dépassement d'entier soit possible. Cela correspond à ≈ 36.9 exaoctets [5] de données 16 bit ou ≈ 576.5 millions de secondes d'acquisition en continue à 32 GSa/s, soit environ 18 années ; ce qu'on juge *suffisant*.

Des fonctions permettant de calculer les moments et cumulants à partir d'histogrammes sont aussi fournies en `C` avec interface `Python` ainsi qu'en `Python` pur. Les routines en `C` visent principalement à effectuer les calculs au moment de l'acquisition lorsqu'on ne veut pas enregistrer les histogrammes directement, lors de tests par exemple, ou lorsque l'on désire avoir une trace des moments en temps réel. Notons que, pour des raisons techniques, les fonctions fournies pour calculer les moments non centrés seront erronées si l'histogramme a été bâti avec des données signées, mais les moment centrés et les cumulants seront justes. Le calcul des cumulants n'est cependant pas optimisé, la formule récursive (B.9) présentée ci-haut n'étant pas très efficace, il est donc préférable de mesurer les moments centrés lorsque la rapidité des calculs est critique et d'en déduire les cumulants plus tard lors de l'analyse des données. L'approche la plus sûre est d'enregistrer les histogrammes et d'en faire le taitement statistique plus tard.

5. Un exaoctet correspond à un milllion de teraoctets ; un milliard de milliard d'octets. Selon *Wolfram Alpha* [146], c'est approximativement la quantité d'information contenue dans l'ensemble des paroles prononcées pendant toute l'Histoire de l'humanité.

B.2 Codes et implémentations

B.2.1 Histogrammes

histograms.c

```c
#if defined(__CYGWIN__) || defined(__MINGW64__)
    // see number from: sdkddkver.h
    // https://docs.microsoft.com/fr-fr/windows/desktop/WinProg/using-the-windows-headers
    #define _WIN32_WINNT 0x0602 // Windows 8
    #include <Processtopologyapi.h>
    #include <processthreadsapi.h>
#endif

#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

#include <omp.h>
#include "mpfr.h"

#define __STDC_FORMAT_MACROS
#include <inttypes.h>
#include <math.h>


// Windows doesn't really like systems with over 64 logical cores.
// This function assign the thread it's called from to a core, bypassing the
// default assignation. It alternates between CPU Groups to assign a thread to
// each physical core first; then it can make use of HTT.
//
// This could be much more sophisticated, but it works well for dual identical
// cpu systems with HTT on and over 64 logical cores.
void manage_thread_affinity()
{
    #ifdef _WIN32_WINNT
        int nbgroups = GetActiveProcessorGroupCount();
        int *threads_per_groups = (int *) malloc(nbgroups*sizeof(int));
        for (int i=0; i<nbgroups; i++)
        {
            threads_per_groups[i] = GetActiveProcessorCount(i);
        }

        // Fetching thread number and assigning it to cores
        int tid = omp_get_thread_num(); // Internal omp thread number (0 -- OMP_NUM_THREADS)
        HANDLE thandle = GetCurrentThread();
        _Bool result;

        int set_group = tid%nbgroups; // We change group for each thread
        int nbthreads = threads_per_groups[set_group]; // Nb of threads in group for affinity mask.
        GROUP_AFFINITY group = {((uint64_t)1<<nbthreads)-1, set_group}; // nbcores amount of 1 in binary

        result = SetThreadGroupAffinity(thandle, &group, NULL); // Actually setting the affinity
        if(!result) fprintf(stderr, "Failed setting output for tid=%i\n", tid);
        free(threads_per_groups);
    #else
        //We let openmp and the OS manage the threads themselves
    #endif
}

// To store an i-bits value in a j-bits integer, with j being a power of 2,
// you need at least j = 2**( log(i)/log(2) + (1-(log(i)/log(2)%1)/1)%1 )


// An histogram done on int casted as uint will be swapped
// This swaps it back, b is the bitlength of the histogram
void swap_histogram(uint64_t *hist, const int b)
{
    const int halfsize = 1<<(b-1);
    uint64_t *tmp = calloc(halfsize, sizeof(uint64_t));
    int i=0;
    for (; i<halfsize; i++){  // Paralelizing those small loops is detrimental
        tmp[i] = hist[i];
        hist[i] = hist[i+halfsize];
    }
    for (; i<2*halfsize; i++){
        hist[i] = tmp[i-halfsize];
    }
    free(tmp);
}


// Computes the histogram for 8-bit samples in uint8 containers
void histogram8_unsigned(uint8_t *data, uint64_t size, uint64_t *hist)
{
    uint64_t *data_64 = (uint64_t *) data;
    #pragma omp parallel
    {
        manage_thread_affinity(); // For 64+ logical cores on Windows
        uint64_t tmp=0;
        #pragma omp for reduction(+:hist[:1<<8])
        for (uint64_t i=0; i<size/8; i++){
            tmp = data_64[i];
            hist[tmp >>  0 & 0xFF]++;
            hist[tmp >>  8 & 0xFF]++;
            hist[tmp >> 16 & 0xFF]++;
            hist[tmp >> 24 & 0xFF]++;
            hist[tmp >> 32 & 0xFF]++;
            hist[tmp >> 40 & 0xFF]++;
            hist[tmp >> 48 & 0xFF]++;
            hist[tmp >> 56 & 0xFF]++;
        }
    }
    // The data that doesn't fit in 64bit chunks, openmp would be overkill here.
    for (uint64_t i=size-(size%8); i<size; i++){
        hist[ data[i] ]++;
    }
}


void histogram8_signed(int8_t *data, uint64_t size, uint64_t *hist)
{
    uint8_t *data_unsigned = (uint8_t *) data;
    histogram8_unsigned(data_unsigned, size, hist);
    swap_histogram(hist, 8); // b is always 8 here
}


// Computes the histogram for (8<b<=16)-bit samples in uint16 containers
void histogram16_unsigned(uint16_t *data, uint64_t size, uint64_t *hist, const int b)
{
    const int32_t tail = 16-b;
    uint64_t *data_64 = (uint64_t *) data;
    #pragma omp parallel
    {
        manage_thread_affinity(); // For 64+ logical cores on Windows
        uint64_t tmp=0;
        #pragma omp for reduction(+:hist[:1<<b])
        for (uint64_t i=0; i<size/4; i++){
            tmp = data_64[i]; // tail get rid of bits > b
            hist[ (tmp >>  0 & 0xFFFF) >> tail ]++;
            hist[ (tmp >> 16 & 0xFFFF) >> tail ]++;
            hist[ (tmp >> 32 & 0xFFFF) >> tail ]++;
            hist[ (tmp >> 48 & 0xFFFF) >> tail ]++;
        }
    }
    // The data that doesn't fit in 64bit chunks, openmp would be overkill here.
    for (uint64_t i=size-(size%4); i<size; i++){
        hist[ data[i] >> tail ]++;
    }
}


```

```
void histogram16_signed(int16_t *data, uint64_t size, uint64_t *hist, const int b)
{
    uint16_t *data_unsigned = (uint16_t *) data;
    histogram16_unsigned(data_unsigned, size, hist, b);
    swap_histogram(hist, b);
}


// #Python POC implementation of the 2d swap, simple but not optimal:
//
// def swap(x):
//     tmp = copy(x[:len(x)/2])
//     x[:len(x)/2] = copy(x[len(x)/2:])
//     x[len(x)/2:] = copy(tmp)
//
// def swap2d(x):
//     xx = x.flatten()
//     swap(xx)
//     l = int(sqrt(len(xx)))
//     #print len(xx), l
//     for i in range(l):
//         swap(xx[i*l:(i+1)*l])
//     return xx.reshape(x.shape)
//
// A 2d histogram done on int casted as uint will be swapped on its two axis
// This swaps it back, b is the bitlength of the histogram
void swap_histogram2d(uint64_t *hist, const int b)
{
    uint64_t rsize = 1<<b;     // Number AND Size of rows (because it's a square)
    swap_histogram(hist, 2*b); // Vertical swap
    #pragma omp parallel
    {
        manage_thread_affinity();
        #pragma omp for
        for (uint64_t i=0; i<rsize; i++){ // For each row
            swap_histogram(hist+(i*rsize), b); // Horizontal swap of each row
        }
    }
}


// Computes the 2d histogram for 8-bit samples in uint8 containers
//
// The 2d histogram is represented by a single dimension array, logically
// seperated in 256 blocks corresponding to the data1 stream, with in-block
// indices corresponding to the data2 stream.
// It appears as a 2d array in the python wrapper.
void histogram2d8_unsigned(uint8_t *data1, uint8_t *data2, uint64_t size, uint64_t *hist)
{
    uint64_t *data1_64 = (uint64_t *) data1;
    uint64_t *data2_64 = (uint64_t *) data2;
    #pragma omp parallel
    {
        manage_thread_affinity(); // For 64+ logical cores on Windows
        uint64_t tmp1=0;
        uint64_t tmp2=0;
        #pragma omp for reduction(+:hist[:1<<(8*2)])
        for (uint64_t i=0; i<size/8; i++){
            tmp1 = data1_64[i];
            tmp2 = data2_64[i];
            hist[ (tmp1 <<  8 & 0xFF00) + (tmp2 >>  0 & 0xFF) ]++;
            hist[ (tmp1 >>  0 & 0xFF00) + (tmp2 >>  8 & 0xFF) ]++;
            hist[ (tmp1 >>  8 & 0xFF00) + (tmp2 >> 16 & 0xFF) ]++;
            hist[ (tmp1 >> 16 & 0xFF00) + (tmp2 >> 24 & 0xFF) ]++;
            hist[ (tmp1 >> 24 & 0xFF00) + (tmp2 >> 32 & 0xFF) ]++;
            hist[ (tmp1 >> 32 & 0xFF00) + (tmp2 >> 40 & 0xFF) ]++;
            hist[ (tmp1 >> 40 & 0xFF00) + (tmp2 >> 48 & 0xFF) ]++;
            hist[ (tmp1 >> 48 & 0xFF00) + (tmp2 >> 56 & 0xFF) ]++;
        }
    }
    // The data that doesn't fit in 64bit chunks, openmp would be overkill here.
    for (uint64_t i=size-(size%8); i<size; i++){
        hist[ (data1[i]<<8) + data2[i] ]++;
    }
}


void histogram2d8_signed(int8_t *data1, int8_t *data2, uint64_t size, uint64_t *hist)
{
    uint8_t *data1_unsigned = (uint8_t *) data1;
    uint8_t *data2_unsigned = (uint8_t *) data2;
    histogram2d8_unsigned(data1_unsigned, data2_unsigned, size, hist);
    swap_histogram2d(hist, 8); // b is always 8 here
}


void reduce(uint64_t** arrs, uint64_t bins, uint64_t begin, uint64_t end)
{
    assert(begin < end);
    if (end - begin == 1) {
        return;
    }
    uint64_t pivot = (begin + end) / 2;
    /* Moving the termination condition here will avoid very short tasks,
     * but make the code less nice. */
    //#pragma omp task
    reduce(arrs, bins, begin, pivot);
    //#pragma omp task
    reduce(arrs, bins, pivot, end);
    //#pragma omp taskwait
    /* now begin and pivot contain the partial sums. */
    #pragma omp parallel
    {
        manage_thread_affinity();
        #pragma omp for
        for (uint64_t i = 0; i < bins; i++)
            arrs[begin][i] += arrs[pivot][i];
    }
}

// Computes the 2d histogram for (8<b<=16)-bit samples in uint16 containers
//
// The 2d histogram is represented by a single dimension array, logically
// seperated in 2**b blocks corresponding to the data1 stream, with in-block
// indices corresponding to the data2 stream.
// It appears as a 2d array in the python wrapper.
//
// atomic: 0 = no atomic; 1 = full atomic
// Best value depends on data.
//  - Full atomic is better for random data and large b
//  - No atomic is better for correlated data
//
// The performance bottleneck seems to be the reduction of huge arrays -> lots of
↪  additions.
// Using critical reduction in the non-atomic case shows CPU load decreasing greatly
↪  after a
// short while but a few cores still at 100% (probably reducing critically).
// The reduce function above reduces manually in non-critical mode to speed this up.
void histogram2d16_unsigned(uint16_t *data1, uint16_t *data2, uint64_t size, uint64_t
↪  *hist, const uint32_t b, const int atomic)
{
    // Precomputing the correct mask and shift values. Helps readability, doesn't  really
    ↪  help performance.
    const int32_t tail0 = 16-b;
    const int32_t tail1 = tail0+16;
    const int32_t tail2 = tail1+16;
    const int32_t tail3 = tail2+16;
    const int32_t mask = (1<<b)-1; // Right amount of 0b1

    uint64_t *data1_64 = (uint64_t *) data1;
    uint64_t *data2_64 = (uint64_t *) data2;
    if (atomic==1){
        #pragma omp parallel
        {
            manage_thread_affinity(); // For 64+ logical cores on Windows

            uint64_t tmp1=0;
            uint64_t tmp2=0;
            // Full atomic should be faster when there's low memory collision, e.g. random
            ↪  data or large *b*.
            // Strikingly, for a given set of data it's typically faster for large *b*.
            // No local histogram; no reduction!
            #pragma omp for //reduction(+:h[:1<<(b*2)])
            for (uint64_t i=0; i<size/4; i++){
                tmp1 = data1_64[i];
                tmp2 = data2_64[i];
```

```
                #pragma omp atomic update
                hist[ ((tmp1 >> tail0 & mask) << b) + (tmp2 >> tail0 & mask) ]++;
                #pragma omp atomic update
                hist[ ((tmp1 >> tail1 & mask) << b) + (tmp2 >> tail1 & mask) ]++;
                #pragma omp atomic update
                hist[ ((tmp1 >> tail2 & mask) << b) + (tmp2 >> tail2 & mask) ]++;
                #pragma omp atomic update
                hist[ ((tmp1 >> tail3 & mask) << b) + (tmp2 >> tail3 & mask) ]++;
            }
        }
    }
    // Using local histograms that have to be reduced afterwards
    // OpenMP allocates its reduction arrays on the stack -> stack overflow for huge arrays
    else{
        uint64_t **hs;
        int n;
        #pragma omp parallel
        {
            manage_thread_affinity(); // For 64+ logical cores on Windows
            n = omp_get_num_threads(); // Amount of threads

            #pragma omp single // Affects only next line
            hs = (uint64_t **) malloc(n * sizeof(uint64_t));
            uint64_t *h = (uint64_t *) calloc(1<<(b*2), sizeof(uint64_t)); // Filled with 0s.
            hs[omp_get_thread_num()] = h;

            uint64_t tmp1=0;
            uint64_t tmp2=0;
            #pragma omp for nowait schedule(static)
            for (uint64_t i=0; i<size/4; i++){
                tmp1 = data1_64[i];
                tmp2 = data2_64[i];
                h[ ((tmp1 >> tail0 & mask) << b) + (tmp2 >> tail0 & mask) ]++;
                h[ ((tmp1 >> tail1 & mask) << b) + (tmp2 >> tail1 & mask) ]++;
                h[ ((tmp1 >> tail2 & mask) << b) + (tmp2 >> tail2 & mask) ]++;
                h[ ((tmp1 >> tail3 & mask) << b) + (tmp2 >> tail3 & mask) ]++;
            }
        }
        // Critical reduction was very slow, this is faster.
        reduce(hs, 1<<(b*2), 0, n); // hs[0] is the reduced array afterwards
        #pragma omp parallel
        {
            manage_thread_affinity();
            // Returning the result to the output array
            #pragma omp for
            for (uint64_t i=0; i<1<<(b*2); i++){
                hist[i]+=hs[0][i];
            }
        }
        for (int i=0; i<n; i++){
            free(hs[i]);
        }
        free(hs);
    }

    // The data that doesn't fit in 64bit chunks, OpenMP would be overkill here.
    for (uint64_t i=size-(size%4); i<size; i++){
        hist[ ((data1[i]>>tail0)<<b) + (data2[i]>>tail0) ]++;
    }
}


void histogram2d16_signed(int16_t *data1, int16_t *data2, uint64_t size, uint64_t *hist, const uint32_t b, const int atomic)
{
    uint16_t *data1_unsigned = (uint16_t *) data1;
    uint16_t *data2_unsigned = (uint16_t *) data2;
    histogram2d16_unsigned(data1_unsigned, data2_unsigned, size, hist, b, atomic);
    swap_histogram2d(hist, b);
}

// Simple, but could be faster
int64_t nCk(int n, int k)
{
    if (k==0){
        return 1;
    }
    return (n*nCk(n-1, k-1))/k; // Product form, division always yields an integer
}

double moment(uint64_t *hist, const int b, const int k, const int centered)
{
    const int size = 1<<b;
    long double bshift=0;
    long double val = 0;
    uint64_t n=0;

    if (centered){
        bshift = moment(hist, b, 1, 0);
    }
    #pragma omp parallel
    {
        manage_thread_affinity(); // For 64+ logical cores on Windows
        if (centered){
            #pragma omp for reduction(+:val), reduction(+:n)
            for (int i=0; i<size; i++){
                val += (long double)hist[i] * powl((long double)i - (long double)bshift, k);
                n += hist[i];
            }
        }
        else{
            #pragma omp for reduction(+:val), reduction(+:n)
            for (int i=0; i<size; i++){
                val += (long double)hist[i] * powl((long double)i, k);
                n += hist[i];
            }
        }
    }
    return (double)(val/(long double)n);
}

double cumulant(uint64_t *hist, const int b, const int k){
    double ret = moment(hist, b, k, 0);
    for (int i=1; i<k; i++){
        ret -= (double)nCk(k-1, i-1)*cumulant(hist, b, i)*moment(hist, b, k-i, 0);
    }
    return ret;
}
```

histograms_otf.py

```
#!/bin/python
# -*- coding: utf-8 -*-

import ctypes
import sys, os
import platform
from numpy.ctypeslib import ndpointer
from numpy import zeros, fromstring, arange, log2
from numpy import int8, uint8, int16, uint16, double
from ctypes import c_uint8, c_int8, c_uint16, c_int16, c_double, c_int, c_uint64, c_uint
import operator as op
from functools import reduce


libpath = os.path.abspath(os.path.dirname(__file__))
if not libpath in os.environ['PATH']:
    os.environ['PATH'] = libpath+os.path.pathsep+os.environ['PATH']

plat_info = dict(plat=platform.system())
if plat_info['plat'] == 'Windows':
    plat_info['lib'] = os.path.join(libpath, 'histograms.dll')
    plat_info['com'] = 'make histograms.dll'
    # Adding cygwin libs path for windows
    libspath = 'C:\\cygwin64\\usr\\x86_64-w64-mingw32\\sys-root\\mingw\\bin'
    if libspath not in os.environ['PATH']:
        os.environ['PATH'] = libspath+os.path.pathsep+os.environ['PATH']
```

```
else:
    plat_info['lib'] = os.path.join(libpath, 'histograms.so')
    plat_info['com'] = 'make histograms.so'


if not os.path.isfile(plat_info['lib']):
    raise IOError("{lib} is missing. To compile on {plat}:\n{com}\n".format(**plat_info))

lib = ctypes.cdll[plat_info['lib']]

# OpenMP stuff
if plat_info["plat"] == "Windows":
    omp = ctypes.CDLL('libgomp-1')
else:
    try:
        omp = ctypes.CDLL("/usr/lib/gcc/x86_64-linux-gnu/7/libgomp.so")
    except:
        omp = ctypes.CDLL("libgomp.so")
set_num_threads = omp.omp_set_num_threads
set_num_threads.argtypes=(c_int,)
set_num_threads.restype=None
get_num_threads = omp.omp_get_max_threads
get_num_threads.restype=c_int
get_num_threads.argtypes=None


def hist1dNbits(x, n=8, ihist=None):
    """
    *ihist* is for using a previously filled histogram and keep filling it.
    """
    signed = True if (x.dtype in [int8, int16]) else False
    container8 = x.dtype in [int8, uint8]
    assert n in range(8,16+1), "Supported bit depths are from 8 to 16"

    k = 2**n
    #fct = lib['histogram{:d}'.format(n)]
    #if n==8:
    if container8:
        fct = lib['histogram8_signed' if signed else 'histogram8_unsigned']
        fct.argtypes = (
            ndpointer(
                dtype=c_int8 if signed else c_uint8,
                shape=(len(x),)
            ),
            c_uint64,
            ndpointer(dtype=c_uint64, shape=(k,))
        )
    else:
        fct = lib['histogram16_signed' if signed else 'histogram16_unsigned']
        fct.argtypes = (
            ndpointer(
                dtype=c_int16 if signed else c_uint16,
                shape=(len(x),)
            ),
            c_uint64,
            ndpointer(dtype=c_uint64, shape=(k,)),
            c_int
        )


    if ihist is None:
        hist = zeros(k, dtype=c_uint64)
    else:
        assert ihist.size == k, "*ihist* has wrong size"
        if signed:
            swap_histogram = lib['swap_histogram']
            swap_histogram.argtypes = (ndpointer(dtype=c_uint64, shape=(k,)), c_int)
            swap_histogram(ihist, 8 if container8 else n)
        hist = ihist

    fct(x, len(x), hist) if container8 else fct(x, len(x), hist, n)

    return hist


def hist2dNbits(x, y, n=8, force_n=False, atomic=False, ihist=None):
    """
    No atomic chosen heuristically as a sweet spot for:
        - somewhat correlated data
        - 10 bit histogram
    Full atomic performance is highly dependent on data and bit depth.

    It requires testing but it can drastically improve performance for large
    bitdepth histograms and/or uncorrelated data.
    """
    if not force_n: # To avoid filling the ram instantly
        assert 8<=n<=12, "8<=n<=12 is required, set kwarg *force_n* to True to override"
    assert len(x)==len(y), "len(x)==len(y) is required"
    assert x.dtype == y.dtype, "x and y should be of the same type"
    signed = True if (x.dtype in [int8, int16]) else False
    container8 = x.dtype in [int8, uint8]
    a = 1 if atomic else 0

    k = 2**n

    if container8:
        assert n==8, "Only 8bit histograms are supported for 8bit containers"
        fct = lib['histogram2d8_signed' if signed else 'histogram2d8_unsigned']
        fct.argtypes = (
            ndpointer(
                dtype=c_int8 if signed else c_uint8,
                shape=(len(x),)
            ),
            ndpointer(
                dtype=c_int8 if signed else c_uint8,
                shape=(len(y),)
            ),
            c_uint64,
            ndpointer(dtype=c_uint64, shape=(k,k))
        )
    else:
        fct = lib['histogram2d16_signed' if signed else 'histogram2d16_unsigned']
        fct.argtypes = (
            ndpointer(
                dtype=c_int16 if signed else c_uint16,
                shape=(len(x),)
            ),
            ndpointer(
                dtype=c_int16 if signed else c_uint16,
                shape=(len(y),)
            ),
            c_uint64,
            ndpointer(dtype=c_uint64, shape=(k,k)),
            c_uint64,
            c_uint
        )

    if ihist is None:
        hist = zeros((k,k), dtype=c_uint64)
    else:
        assert ihist.size == k**2, "*ihist* has wrong size"
        if signed:
            swap_histogram2d = lib['swap_histogram2d']
            swap_histogram2d.argtypes = (ndpointer(dtype=c_uint64, shape=(k,k)), c_int)
            swap_histogram2d(ihist, 8 if container8 else n)
        hist = ihist
    fct(x, y, len(x), hist) if container8 else fct(x, y, len(x), hist, n, a)

    return hist


# Extra stuff: Computing moments and cumulants on histograms

#Python implementation
# n choose r: n!/(r!(n-r)!)
def ncr(n, r):
    r = min(r, n-r)
    numer = reduce(op.mul, range(n, n-r, -1), 1)
    denom = reduce(op.mul, range(1, r+1), 1)
    return numer / denom

# kth moment of h, centered by default
def moment_py(h, k, centered=True):
    if centered:
        bshift = double(moment_py(h, 1, centered=False))
    else:
```

```
            bshift = 0
        b = double(arange(h.size))-bshift
        n = double(h.sum())
        return (h*b**k).sum()/n

    def cumulant_py(h, k, centered=True):
        hh = double(h)
        ret = moment_py(hh,k,False)
        ret -= sum([ncr(k-1,m-1)*cumulant_py(hh,m)*moment_py(hh,k-m,False) for m in
        ↪  range(1,k)])
        return ret

    # C implementation
    def moment(h, k, centered=True):
        b = int(log2(len(h))) # Assumes h is a b-bit histogram
        c = int(centered)
        fct = lib['moment']
        fct.argtypes = (
                ndpointer(
                    dtype=c_uint64,
                    shape=(len(h),)
                ),
                c_int,
                c_int,
                c_int
            )
        fct.restype = c_double
        return fct(h, b, k, c)

    def cumulant(h, k):
        b = int(log2(len(h))) # Assumes h is a b-bit histogram
        fct = lib['cumulant']
        fct.argtypes = (
                ndpointer(
                    dtype=c_uint64,
                    shape=(len(h),)
                ),
                c_int,
                c_int
            )
        fct.restype = c_double
        return fct(h, b, k)
```

makefile

```
# Toolchain, using mingw on windows
CC = $(OS:Windows_NT=x86_64-w64-mingw32-)gcc
PKG_CFG = $(OS:Windows_NT=x86_64-w64-mingw32-)pkg-config
RM = rm

# flags
CFLAGS = -Ofast -march=native -Wall
OMPFLAGS = -fopenmp -fopenmp-simd
SHRFLAGS = -fPIC -shared

# libraries
LDLIBS = -lmpfr

# filenames
SRC = histograms.c
SHREXT = $(if $(filter $(OS),Windows_NT),.dll,.so)
SHRTGT = $(SRC:.c=$(SHREXT))


all: $(SHRTGT) #$(TARGET) $(SHRTGT)

$(SHRTGT): $(SRC)
        $(CC) $(SRC) -o $(SHRTGT) $(SHRFLAGS) $(CFLAGS) $(OMPFLAGS) $(LDLIBS)

force:
        $(CC) $(SRC) -o $(SHRTGT) $(SHRFLAGS) $(CFLAGS) $(OMPFLAGS) $(LDLIBS)

clean:
        @[ -f $(SHRTGT) ] && $(RM) $(SHRTGT) || true

.PHONY: all clean force
```

B.2.2 Autocorrelations

src/common.hpp

```
#ifndef common_H
#define common_H

#if defined(__CYGWIN__) || defined(__MINGW64__)
    // see number from: sdkddkver.h
    // https://docs.microsoft.com/fr-fr/windows/desktop/WinProg/using-the-windows-headers
    #define _WIN32_WINNT 0x0602 // Windows 8
    #include <windows.h>
    #include <Processtopologyapi.h>
    #include <processthreadsapi.h>
#endif

#define NOOP (void)0

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <math.h>

#include <iostream>
#include <iomanip>

#include <omp.h>
#include <limits>


#ifdef _WIN32_WINNT
    #include "mpreal.h"
#else
    #include <mpreal.h>
#endif


#define __STDC_FORMAT_MACROS
#include <inttypes.h>


using namespace std;
using mpfr::mpreal;

// For setting desired mpreal precision beforehand
void set_mpreal_precision(int d);

void manage_thread_affinity();


#endif // common_H
```

src/common.cpp

```
#include "common.hpp"

```

```cpp
using namespace std;

// For setting desired mpreal precision beforehand
void set_mpreal_precision(int d){
    // Before any mreal are created
    const int digits = d; // Setting high precision
    mpreal::set_default_prec(mpfr::digits2bits(digits));
}

void manage_thread_affinity()
{
    #ifdef _WIN32_WINNT
        int nbgroups = GetActiveProcessorGroupCount();
        int *threads_per_groups = (int *) malloc(nbgroups*sizeof(int));
        for (int i=0; i<nbgroups; i++)
        {
            threads_per_groups[i] = GetActiveProcessorCount(i);
        }

        // Fetching thread number and assigning it to cores
        int tid = omp_get_thread_num(); // Internal omp thread number (0 -- OMP_NUM_THREADS)
        HANDLE thandle = GetCurrentThread();
        bool result;

        WORD set_group = tid%nbgroups; // We change group for each thread
        int nbthreads = threads_per_groups[set_group]; // Nb of threads in group for affinity mask.
        GROUP_AFFINITY group = {((uint64_t)1<<nbthreads)-1, set_group}; // nbcores amount of 1 in binary

        result = SetThreadGroupAffinity(thandle, &group, NULL); // Actually setting the affinity
        if(!result) fprintf(stderr, "Failed setting output for tid=%i\n", tid);
    #else
        //We let openmp and the OS manage the threads themselves
    #endif
}
```

src/acorrs.hpp

```cpp
#ifndef autoco_H
#define autoco_H

#include "common.hpp"

//TODO: Make (it?) a general correlation class (with aCorr as a special case?)
template<class T> class ACorrUpTo
{
public:
// Variables //

    // Casting over different type sign is very slow, we avoid it.
    // (T)-1>0 is true only if T is unsigned
    // Would work with int < 64bits but would overflow faster
    typedef typename conditional<((T)-1>0), uint64_t, int64_t>::type accumul_t;

    // Math stuff
    accumul_t m;
    accumul_t n;
    int k;

    accumul_t *rk;
    accumul_t *bk;
    accumul_t *gk;

    // Precision stuff
    mpreal m_mpfr;
    mpreal n_mpfr;
    mpreal k_mpfr;

    mpreal *rk_mpfr;
    mpreal *bk_mpfr;
    mpreal *gk_mpfr;

    // Autocorrelations results
    mpreal *aCorrs_mpfr;
    double *aCorrs;

    // Managerial stuff
    uint64_t chunk_processed;
    uint64_t chunk_size;
    uint64_t block_processed;

// Constructors //
    ACorrUpTo(int k);

// Methods //
    void accumulate(T *buffer, uint64_t size);
    inline void accumulate_chunk(T *buffer, uint64_t size);
    inline void accumulate_chunk_edge(T *buffer, uint64_t size);
    virtual void accumulate_m_rk(T *buffer, uint64_t size);
    inline void accumulate_m_rk_edge(T *buffer, uint64_t size);
    // gk is the beginning corrections
    // It should be computed ONCE on the very first chunk of data
    inline void accumulate_gk(T *buffer, uint64_t size);
    // bk is the end corrections
    // Re-compute for each new chunk to keep autocorr valid
    inline void accumulate_bk(T *buffer, uint64_t size);

    inline void update();
    inline void update_mpfr();
    inline void reset_accumulators();

    mpreal get_mean_mpfr();
    double get_mean();
    mpreal get_var_mpfr();
    double get_var();
    mpreal* get_aCorrs_mpfr();
    void compute_aCorrs();
    double* get_aCorrs();
    void get_aCorrs(double* res, int size);
    void get_rk(double* res, int size);

    uint64_t compute_chunk_size();

// Destructor //
   virtual ~ACorrUpTo();
};


#endif // autoco_H
```

src/acorrs.tpp

```cpp
#include "acorrs.hpp"

using namespace std;

template<class T>
inline ACorrUpTo<T>::ACorrUpTo(int k): m(0), n(0), k(k), m_mpfr(0), n_mpfr(0), k_mpfr(k)
{
    rk = new accumul_t [k](); // Parentheses initialize to zero
    gk = new accumul_t [k]();
    bk = new accumul_t [k]();
    rk_mpfr = new mpreal [k]();
    gk_mpfr = new mpreal [k]();
    bk_mpfr = new mpreal [k]();

    aCorrs = new double [k]();
    aCorrs_mpfr = new mpreal [k]();

    block_processed = 0;
```

```
    chunk_processed = 0;
    chunk_size = compute_chunk_size(); // Auto largest possible
}

// Methods //
template<class T>
inline void ACorrUpTo<T>::accumulate(T *buffer, uint64_t size){
    // On each call, a new block being processed.
    block_processed++;
    n += size;
    accumulate_gk(buffer, size); // Compute bk on very first data
    // Loop on whole chunks
    uint64_t i; // Will point to last partial chunk after the loop
    for (i=0; i<(size-k)/chunk_size; i++){
        accumulate_chunk(buffer+i*chunk_size, chunk_size);
        update(); // Update mpfr values and reset accumulators
    }
    // Last (potentially) partial chunk, keeps chunk_processed accurate
    if ((size-k)%chunk_size){
        accumulate_chunk(buffer+i*chunk_size, (size-k)%chunk_size);
        update(); // Update mpfr values and reset accumulators
    }
    // Right edge chunk, doesn't count in chunk_processed because it's small
    accumulate_chunk_edge(buffer, size);
    update(); // Update mpfr values and reset accumulators
    accumulate_bk(buffer, size); // Computing (replacing) bk on last data
}

template<class T>
inline void ACorrUpTo<T>::accumulate_chunk(T *buffer, uint64_t size){
    accumulate_m_rk(buffer, size); // Accumulating
    chunk_processed++;
}

template<class T>
inline void ACorrUpTo<T>::accumulate_chunk_edge(T *buffer, uint64_t size){
    accumulate_m_rk_edge(buffer, size); // Accumulating
    //chunk_processed++;
}
template<class T>
inline void ACorrUpTo<T>::accumulate_m_rk(T *buffer, uint64_t size){
    #pragma omp parallel
    {
        manage_thread_affinity();
        #pragma omp for simd reduction(+:m), reduction(+:rk[:k])
        for (uint64_t i=0; i<size; i++){
            m += (accumul_t)buffer[i];
            #pragma omp ordered simd
            for (int j=0; j<k; j++){
                rk[j] += (accumul_t)buffer[i]*(accumul_t)buffer[i+j];
            }
        }
    }
}

template<class T>
inline void ACorrUpTo<T>::accumulate_m_rk_edge(T *buffer, uint64_t size){
    for (uint64_t i=size-k; i<size; i++){
        m += (accumul_t)buffer[i];
        for (uint64_t j=0; j<size-i; j++){
            rk[j] += (accumul_t)buffer[i]*(accumul_t)buffer[i+j];
        }
    }
}

// gk is the beginning corrections
// It should be computed ONCE on the very first chunk of data
template<class T>
inline void ACorrUpTo<T>::accumulate_gk(T *buffer, uint64_t size){
    for (int i=0; i<k; i++){
        for (int j=0; j<i; j++){
            gk[i] += (accumul_t)buffer[j];
        }
        gk_mpfr[i] += gk[i]; // Updating precision value
        gk[i] = 0; // Reseting accumulator
    }
}

// bk is the end corrections
// Re-compute for each new chunk to keep autocorr valid
template<class T>
inline void ACorrUpTo<T>::accumulate_bk(T *buffer, uint64_t size){
    for (int i=0; i<k; i++){
        for (uint64_t j=size-i; j<size; j++){
            bk[i] += (accumul_t)buffer[j];
        }
        bk_mpfr[i] += bk[i]; // Updating precision value
        bk[i] = 0; // Reseting accumulator
    }
}

template<class T>
inline void ACorrUpTo<T>::update(){
    update_mpfr();
    reset_accumulators();
}

template<class T>
inline void ACorrUpTo<T>::update_mpfr(){
    m_mpfr += m;
    n_mpfr += n;
    for (int i=0; i<k; i++){
        rk_mpfr[i] += rk[i];
    } // bk_mpfr and gk_mpfr are updated at computation
}

template<class T>
inline void ACorrUpTo<T>::reset_accumulators(){
    m=0;
    n=0;
    for (int i=0; i<k; i++){
        rk[i] = 0;
    }
}

template<class T>
inline mpreal ACorrUpTo<T>::get_mean_mpfr(){
    update(); // Just to be sure
    mpreal r = m_mpfr/n_mpfr;
    return r;
}

template<class T>
inline double ACorrUpTo<T>::get_mean(){
    return (double)get_mean_mpfr();
}

template<class T>
inline mpreal ACorrUpTo<T>::get_var_mpfr(){
    update(); // Just to be sure
    mpreal v = (rk_mpfr[0]-pow(m_mpfr,2)/n_mpfr)/(n_mpfr);
    return v;
}

template<class T>
inline double ACorrUpTo<T>::get_var(){
    return (double)get_var_mpfr();
}

template<class T>
inline mpreal* ACorrUpTo<T>::get_aCorrs_mpfr(){
    // No corr across blocks: i -> i*block_processed
    mpreal n_k;
    for (int i=0; i<k; i++){
        n_k = n_mpfr - (mpreal)(i*block_processed);
        aCorrs_mpfr[i] = (rk_mpfr[i] - (m_mpfr-bk_mpfr[i])*(m_mpfr-gk_mpfr[i])/n_k)/n_k;
    }
    return aCorrs_mpfr; // Return pointer to array
}

template<class T>
inline void ACorrUpTo<T>::compute_aCorrs(){
    get_aCorrs_mpfr();
    for (int i=0; i<k; i++){
        aCorrs[i] = (double)aCorrs_mpfr[i]; // Small loss of precision here
    }
```

```
}

template<class T>
inline double* ACorrUpTo<T>::get_aCorrs(){
    compute_aCorrs();
    return aCorrs; // Return pointer to array
}

template<class T>
inline void ACorrUpTo<T>::get_aCorrs(double* res, int size){
    compute_aCorrs();
    for (int i=0; i<size; i++){
        res[i] = aCorrs[i];
    }
}

template<class T>
inline void ACorrUpTo<T>::get_rk(double* res, int size){
        if (size>k){
            size = k;
        }
        for (int i=0; i<size; i++){
            res[i] = (double)rk[i];
        }
    }


// Max chunk_size to avoid overflow: chunk_size*buff_max² == accumul_max
// Relevant quantities are max(accumul_t) and max(abs(min(buff)), max(buff))
// e.g. int16 buff spanning -2^15:2^15-1 == -32768:32767 in int64 accumulator:
//   - max positive buff squared value is (-2^15)^2 = 2^30 = 1073741824
//   - max negative buff squared value is -2^15*(2^15-1) = -2^30+2^15 = -1073709056
//   - accumulator max positive value is 2^63-1 -> (2^63-1)/2^30 = (2^33 - 1) + (1 - 2^-30)
//       - With result stated as a positive sum of the integer and fractional parts
//       - Casting back to int64 yields 2^33-1 = 8589934591
//   - accumulator max negative value is -2^63
//          -> -2^63/(-2^15*(2^15-1) = 2^33/(1-2^-15) = (2^33+2**18)/(1-2**-30)
//          -> 2^33 + 2^18 + epsilon (first order Taylor, positive epsilon tends to 0)
//       - Casting back to int64 yields 2^33 + 2^18 + 0 = 8590196736 > 8589934591
//   - The chunk_size is thus the smallest results: 8589934591
// e.g. uint16 spanning 0:2^16-1 in uint64 accumulator
//   - Similarly: 2^64-1/(2^16-1)^2 = 2^32 + 2^17 - 1 + 4 + epsilon -> 4295098371
template<class T>
inline uint64_t ACorrUpTo<T>::compute_chunk_size(){
    uint64_t buff_max = max(abs(numeric_limits<T>::min()), abs(numeric_limits<T>::max()));
    uint64_t accumul_max = numeric_limits<accumul_t>::max();
    uint64_t ret = accumul_max/(buff_max*buff_max);
    return ret; // Int division removes fractional possibility
}


// Destructor //
template<class T>
inline ACorrUpTo<T>::~ACorrUpTo(){
    delete[] rk;
    delete[] gk;
    delete[] bk;
    delete[] rk_mpfr;
    delete[] gk_mpfr;
    delete[] bk_mpfr;
}
```

src/acorrs.cpp

```
#include "acorrs.tpp"

// Instantiating classes for the types we want to support
template class ACorrUpTo<uint8_t>;
template class ACorrUpTo<int8_t>;
template class ACorrUpTo<uint16_t>;
template class ACorrUpTo<int16_t>;
```

src/acorrsFFT.hpp

```
#ifndef autocoFFT_H
#define autocoFFT_H

#include "common.hpp"
#include "acorrs.hpp"
#include "fftw3.h"

inline void halfcomplex_norm2(double *buff, int fftwlen);

template <class T> class ACorrUpToFFT: public ACorrUpTo<T>
{
public:
// Base class stuff //
    typedef typename ACorrUpTo<T>::accumul_t accumul_t;
    accumul_t *rk = ACorrUpTo<T>::rk;
    accumul_t &m = ACorrUpTo<T>::m; // Some voodoo here; needed for openmp reduction?
    int &k = ACorrUpTo<T>::k;
    mpreal *rk_mpfr = ACorrUpTo<T>::rk_mpfr;
    // FFT(W) specific stuff
    int len;
    int fftwlen;
    fftw_plan fwd_plan;
    fftw_plan rev_plan;
    double *in;
    double *out;
    int counter_max;
// Constructors //
    ACorrUpToFFT(int k, int len);

    void accumulate_m_rk(T*, uint64_t);
    int compute_accumul_max();
    int test;

    virtual ~ACorrUpToFFT();
};


#endif // autocoFFT_H
```

src/acorrsFFT.tpp

```
#include "acorrsFFT.hpp"

using namespace std;

inline void halfcomplex_norm2(double *buff, int fftwlen){
    // Multiplying with conjugate in-place
    buff[0] = buff[0]*buff[0]; // By symmetry, first one is purely real.
    int j=0;
    // buff*buff.conj() (result is real), buff in half-complex format
    for (j++; j<fftwlen/2; j++){
        // (a+ib)*(a-ib) = a²+b²
        buff[j] = buff[j]*buff[j] + buff[fftwlen-j]*buff[fftwlen-j]; // norm²
    }
    buff[j] = buff[j]*buff[j]; // fftwlen even implies n/2 is purely real too
    // buff*buff.conj() (imaj part of result)
    for (j++; j<fftwlen; j++){
        buff[j] = 0;   // Norm² of complex has no imaj part
    }
}

```

```cpp
template <class T>
inline ACorrUpToFFT<T>::ACorrUpToFFT(int k, int len): ACorrUpTo<T>(k), len(len)
{
    // FFT length
    fftwlen = 1<<(int)ceil(log2(2*len-1)); //TODO: Assert that k < len

    // Tries to load wisdom if it exists
    fftw_import_wisdom_from_filename("FFTW_Wisdom.dat");

    // Generating FFTW plans
    in = fftw_alloc_real(fftwlen); // Temp buffers for plan
    out = fftw_alloc_real(fftwlen);
    fwd_plan = fftw_plan_r2r_1d(fftwlen, in, out, FFTW_R2HC, FFTW_EXHAUSTIVE);
    rev_plan = fftw_plan_r2r_1d(fftwlen, in, out, FFTW_HC2R, FFTW_EXHAUSTIVE);

    // Max number of double accumulation before we get errors on correlations
    counter_max = compute_accumul_max();
}

/*
The number of time we accumulate in Fourier space is limited for accuracy.
1. A double can exactly represent successive integers -2^53:2^53
2. After the FFT roundtrip, we have an int*int times the int fftwlen, an integer
3. If the accumulator reaches beyond this, we create errors no matter what
4. Standard double accumulation errors also occurs
5. To mitigate those, we cap the number of accumulations to a reasonable value of 4096
6. Testing with 2 GiSa yields rk rel. errors for {int8, uint8, int16, uint16} smaller than:
     {0, 0, 5e-15, 9e-16} for worse case scenario of max amplitude values
     {0, 0,    ±0, 2e-16} for random uniforms
   This is better or similar to the error induced when casting the final result to double
   It is thus considered acceptable
7. Kahan summation doesn't seem to help much and slows things down
8. 3/4 factor added empirically to avoid errors on rk[0]
*/
template <class T>
inline int ACorrUpToFFT<T>::compute_accumul_max(){
    uint64_t buff_max = max(abs(numeric_limits<T>::min()), abs(numeric_limits<T>::max()));
    int ret = (int)min((((uint64_t)1)<<53)/((uint64_t)fftwlen*buff_max*buff_max),(uint64_t)4096)*3/4;
    return ret;
}

template <class T>
inline ACorrUpToFFT<T>::~ACorrUpToFFT(){
    // Saving wisdom for future use
    fftw_export_wisdom_to_filename("FFTW_Wisdom.dat");
    // Deleting plans
    fftw_destroy_plan(fwd_plan);
    fftw_destroy_plan(rev_plan);
    // Deleting temp buffers
    fftw_free(in);
    fftw_free(out);
}

template <class T>
inline void ACorrUpToFFT<T>::accumulate_m_rk(T *buffer, uint64_t size){
    uint64_t fftnum = size/len;
    #pragma omp parallel
    {
        manage_thread_affinity();
        double *ibuff = fftw_alloc_real(fftwlen);
        double *obuff = fftw_alloc_real(fftwlen);
        double *rk_fft_local = fftw_alloc_real(fftwlen);
        for (int i=0; i<fftwlen; i++){
            rk_fft_local[i] = 0;
        }

        // Each thread accumulates at most counter_max iterations to minimize errors
        int counter = 0;

        #pragma omp for reduction(+:m), reduction(+:rk[:k])
        for (uint64_t i=0; i<fftnum; i++){
            T *buff = buffer + i*len;
            // Filling buffers and accumulating m
            int j;
            for (j=0; j<len; j++){
                m += (accumul_t)buff[j];
                ibuff[j] = (double)buff[j];
            }
            for(; j<fftwlen; j++){
                ibuff[j] = 0; // Needs zeroing, used as scratch pad
            }

            fftw_execute_r2r(fwd_plan, ibuff, obuff); // Forward FFT
            halfcomplex_norm2(obuff, fftwlen); // obuff*obuff.conj() element-wise
            // Reverse FFT used to be here instead of outside this loop

            // Accumulating rk, correcting for the missing data between fft_chunks
            for (j=0; j<k; j++){
                rk_fft_local[j] += obuff[j];
                // Exact correction for edges
                for(int l = j; l<k-1; l++){
                    rk[l+1] += (accumul_t)buff[len-j-1]*(accumul_t)buff[len-j+l];
                }
            }
            // Filling rk_fft_local beyond k
            for (; j<fftwlen; j++){
                rk_fft_local[j] += obuff[j];
            }
            counter++;
            if (counter==counter_max){
                counter = 0;
                // Here's the optimization. Thanks to FFT's linearity!
                fftw_execute_r2r(rev_plan, rk_fft_local, obuff); // Reverse FFT
                // Manual reduction of ifft(rk_fft_local) to rk_mpfr
                #pragma omp critical
                for (int i=0; i<k; i++){
                    // rk_mpfr would be an integer if not for floating point errors
                    // outter round might be sufficient, rint_round rounds .5 away from 0
                    rk_mpfr[i] += mpfr::rint_round(mpfr::rint_round((mpreal)obuff[i])/(mpreal)fftwlen);
                }
                // Zeroing for next iteration
                for (int i=0; i<fftwlen; i++){
                    rk_fft_local[i] = 0;
                }
            }
        }
        if (counter != 0){
            // Here's the optimization. Thanks to FFT's linearity!
            fftw_execute_r2r(rev_plan, rk_fft_local, obuff); // Reverse FFT
            // Manual reduction of ifft(rk_fft_local) to rk_mpfr
            #pragma omp critical
            for (int i=0; i<k; i++){
                // rk_mpfr would be an integer if not for floating point errors
                // outter round might be sufficient, rint_round rounds .5 away from 0
                rk_mpfr[i] += mpfr::rint_round(mpfr::rint_round((mpreal)obuff[i])/(mpreal)fftwlen);
            }
        }
        // Freeing memory
        fftw_free(ibuff);
        fftw_free(obuff);
        fftw_free(rk_fft_local);
    }
    // Leftover data! Probably too small to benefit from parallelization.
    for (uint64_t i=size-size%len; i<size; i++){
        m += (accumul_t)buffer[i];
        for (int j=0; j<k; j++){
            rk[j] += (accumul_t)buffer[i]*(accumul_t)buffer[i+j];
        }
    }
}
```

src/acorrsFFT.cpp

```cpp
#include "acorrsFFT.tpp"

```

```
// Instantiating classes for the types we want to support
template class ACorrUpToFFT<uint8_t>;
template class ACorrUpToFFT<int8_t>;
template class ACorrUpToFFT<uint16_t>;
template class ACorrUpToFFT<int16_t>;
```

src/acorrsPhi.hpp

```
#ifndef autocoPhi_H
#define autocoPhi_H

#include "common.hpp"

//TODO: Make (it?) a general correlation class (with aCorr as a special case?)
template<class T> class ACorrUpToPhi
{
public:
// Variables //

    // Casting over different type sign is very slow, we avoid it.
    // (T)-1>0 is true only if T is unsigned
    // Would workfk with int < 64bits but would overflow faster
    typedef typename conditional<((T)-1>0), uint64_t, int64_t>::type accumul_t;

    // Math stuff
    accumul_t n;  // Actual length of data
    uint64_t *nfk;  // nfk for current block
    int k;
    int lambda;     // Period, phases will span [0, lambda-1]

    // f,k specific accumulators
    accumul_t *mf;  // One m per phase
    accumul_t *rfk; // Conceptually: rfk[i][j] == rfk[i*k+j]
    accumul_t *bfk; // Similarly
    accumul_t *gfk; // Similarly
    accumul_t *bk; // Similarly
    accumul_t *gk; // Similarly

    // Precision stuff
    mpreal k_mpfr;
    mpreal l_mpfr;

    mpreal *mf_mpfr;
    mpreal *Nfk_mpfr;
    mpreal *rfk_mpfr;
    mpreal *bfk_mpfr;
    mpreal *gfk_mpfr;
    mpreal *bk_mpfr;
    mpreal *gk_mpfr;

    // Autocorrelations results
    mpreal *aCorrs_mpfr;
    double *aCorrs;
    double *ak;

    // Managerial stuff
    uint64_t chunk_processed;
    uint64_t chunk_size;
    uint64_t block_processed;

// Constructors //
    ACorrUpToPhi(int k, int lambda);

// Methods //
    uint64_t get_nfk(uint64_t N, int lambda, int f, int k);
    void compute_current_nfk(uint64_t size);
    void accumulate_Nfk(uint64_t size);   // Denominator for f and k
    void accumulate(T *buffer, uint64_t size);
    inline void accumulate_chunk(T *buffer, uint64_t size);
    inline void accumulate_chunk_edge(T *buffer, uint64_t size);
    virtual void accumulate_mf_rfk(T *buffer, uint64_t nphi);
    inline void accumulate_mf_rfk_edge(T *buffer, uint64_t size);
    // gfk is the beginning corrections
    // It should be computed ONCE on the very first chunk of data
    inline void accumulate_gfk(T *buffer, uint64_t size);
    // bfk is the end corrections
    // Re-compute for each new chunk to keep autocorr valid
    inline void accumulate_bfk(T *buffer, uint64_t size);
    // gk is the beginning corrections without a phase reference
    // It should be computed ONCE on the very first chunk of data
    inline void accumulate_gk(T *buffer, uint64_t size);
    // bk is the end corrections without a phase reference
    // Re-compute for each new chunk to keep autocorr valid
    inline void accumulate_bk(T *buffer, uint64_t size);

    inline void update();
    inline void update_mpfr();
    inline void reset_accumulators();

    mpreal* get_aCorrs_mpfr();
    void get_aCorrs0();
    void compute_aCorrs();
    double* get_aCorrs();
    void get_aCorrs(double* res, int size);
    void get_rfk(double* res, int size);

    uint64_t compute_chunk_size();

// Destructor //
   virtual ~ACorrUpToPhi();
};


#endif // autocoPhi_H
```

src/acorrsPhi.tpp

```
#include "acorrsPhi.hpp"

using namespace std;

template<class T>
inline ACorrUpToPhi<T>::ACorrUpToPhi(int k, int lambda): n(0), k(k), lambda(lambda),
↪  k_mpfr(k), l_mpfr(lambda), chunk_processed(0), block_processed(0)
{
    mf = new accumul_t [lambda]();      // Parentheses initialize to zero
    nfk = new uint64_t [lambda*k]();    // There are lambda phases and k lags -> lambda*k
    ↪  values
    rfk = new accumul_t [lambda*k]();
    gfk = new accumul_t [lambda*k]();
    bfk = new accumul_t [lambda*k]();
    gk = new accumul_t [lambda*k]();
    bk = new accumul_t [lambda*k]();

    mf_mpfr = new mpreal [lambda]();
    Nfk_mpfr = new mpreal [lambda*k]();
    rfk_mpfr = new mpreal [lambda*k]();
    gfk_mpfr = new mpreal [lambda*k]();
    bfk_mpfr = new mpreal [lambda*k]();
    gk_mpfr = new mpreal [lambda*k]();
    bk_mpfr = new mpreal [lambda*k]();

    aCorrs = new double [lambda*k]();
    aCorrs_mpfr = new mpreal [lambda*k]();

    // Phaseless correlations, for testing/camparison
    ak = new double [k]();

    chunk_size = compute_chunk_size(); // Auto largest possible
}

// Methods //
template<class T>
```

```cpp
inline void ACorrUpToPhi<T>::accumulate(T *buffer, uint64_t size){
    // On each call, a new block being processed.
    block_processed++;
    n += size;
    accumulate_Nfk(size);
    accumulate_gfk(buffer, size); // Compute gfk on very first data

    // 1. Do min(nfk) in blocks for all f and k with j-loop as the outer one; CHUNK
    // nfk[lambda*k-1] is min(nfk); nfk[f*k+j] >= nfk[lambda*k-1]
    // Loop on whole chunks
    uint64_t i; // Will point to last partial chunk after the loop
    for (i=0; i<(nfk[lambda*k-1])/chunk_size; i++){
        accumulate_chunk(buffer+i*chunk_size, chunk_size);
        update(); // Update mpfr values and reset accumulators
    }
    // Last (potentially) partial chunk, keeps chunk_processed accurate
    if ((nfk[lambda*k-1])%chunk_size){
        accumulate_chunk(buffer+i*chunk_size, (nfk[lambda*k-1])%chunk_size);
        update(); // Update mpfr values and reset accumulatoris
    }
    // 2. Do the remainder from min(nfk) to nfk with j-loop as the inner one; EDGE
    // Right edge chunk, doesn't count in chunk_processed because it's small
    accumulate_chunk_edge(buffer, size);
    update(); // Update mpfr values and reset accumulators
    accumulate_bfk(buffer, size); // Computing (replacing) bfk on last data
}

template<class T>
inline void ACorrUpToPhi<T>::accumulate_chunk(T *buffer, uint64_t size){
    accumulate_mf_rfk(buffer, size); // Accumulating
    chunk_processed++;
}

template<class T>
inline void ACorrUpToPhi<T>::accumulate_chunk_edge(T *buffer, uint64_t size){
    accumulate_mf_rfk_edge(buffer, size); // Accumulating
    //chunk_processed++;
}

template<class T>
uint64_t ACorrUpToPhi<T>::get_nfk(uint64_t N, int lambda, int f, int k){
    // Whole block + Potential Partial Block - Avoid k out of buffer
    //return N/lambda + ((N%lambda) > 0) -
    ↪  ((-((int)(N%lambda))+lambda)%lambda+k+f)/lambda;
    return N/lambda + ((uint64_t)((f+k)%lambda)<(N%lambda)) - (f+k)/lambda; // Same as
    ↪  above line, but cleaner.
}

template<class T>
void ACorrUpToPhi<T>::compute_current_nfk(uint64_t size){
    for (int f=0; f<lambda; f++){
        for (int i=0; i<k; i++){
            nfk[f*k+i] = get_nfk(size, lambda, f, i);
        }
    }
}

template<class T>
void ACorrUpToPhi<T>::accumulate_Nfk(uint64_t size){
    compute_current_nfk(size);  // Set nfk to that of current block if not already done
    for (int i=0; i<k*lambda; i++){
        Nfk_mpfr[i] += (mpreal)nfk[i];   // accumulating
    }
}

template<class T>
inline void ACorrUpToPhi<T>::accumulate_mf_rfk(T *buffer, uint64_t size){
    #pragma omp parallel
    {
        manage_thread_affinity();
        #pragma omp for simd collapse(2) reduction(+:mf[:lambda]),
        ↪  reduction(+:rfk[:k*lambda])
        for (uint64_t j=0; j<size*lambda; j+=lambda){
            for (int f=0; f<lambda; f++){
                mf[f] += (accumul_t)buffer[f+j];
                #pragma omp ordered simd
                for (int i=0; i<k; i++){
                    rfk[f*k+i] += (accumul_t)buffer[f+j]*(accumul_t)buffer[f+j+i];
                }
            }
        }
        //// Test to minimise multiplications by lambda
        //#pragma omp for simd collapse(2) reduction(+:mf[:lambda]),
        ↪  reduction(+:rfk[:k*lambda])
        //for (uint64_t j=0; j<size*lambda; j+=lambda){
        //    for (int f=0; f<lambda; f++){
        //        mf[f] += (accumul_t)buffer[f+j];
        //        #pragma omp ordered simd
        //        for (int i=0; i<k; i++){
        //            rfk[f*k+i] += (accumul_t)buffer[f+j]*(accumul_t)buffer[f+j+i];
        //        }
        //    }
        //}
    }
}

template<class T>
inline void ACorrUpToPhi<T>::accumulate_mf_rfk_edge(T *buffer, uint64_t size){
    for (int i=0; i<k; i++){
        for (int f=0; f<lambda; f++){
            // Remainder of nfk, passed the common min(nfk)==nfk[lambda*k-1]
            for (uint64_t j=nfk[lambda*k-1]; j<nfk[f*k+i]; j++){
                // Only counting once. Could probably be optmiized to avoid if
                if (i==0){
                    mf[f] += (accumul_t)buffer[f+j*lambda];
                }
                rfk[f*k+i] +=
                ↪  (accumul_t)buffer[f+j*lambda]*(accumul_t)buffer[f+j*lambda+i];
            }
        }
    }
}

// gfk is the beginning corrections
// It should be computed ONCE per block on the very first chunk of data
template<class T>
inline void ACorrUpToPhi<T>::accumulate_gfk(T *buffer, uint64_t size){
    for (int f=0; f<lambda; f++){
        //uint64_t alphaf = size/lambda + (f<(int)(size%lambda));
        for (int i=0; i<k; i++){
            //uint64_t alphaf = size/lambda + (((f+i)%lambda)<(int)(size%lambda));
            //for (uint64_t j=0; j<alphaf-nfk[f*k+i]; j++){
            for (uint64_t j=0; j<(uint64_t)(i+f)/lambda; j++){
                gfk[f*k+i] += (accumul_t)buffer[j*lambda+((i+f)%lambda)];
            }
            gfk_mpfr[f*k+i] += gfk[f*k+i]; // Updating precision value
            gfk[f*k+i] = 0; // Reseting accumulator
        }
    }
    // Hijacking gfk to compute gk too
    accumulate_gk(buffer,size); // Not required but fast and helps testing
}

// bfk is the end corrections
// It should be computed ONCE per block on the very last chunk of data
template<class T>
inline void ACorrUpToPhi<T>::accumulate_bfk(T *buffer, uint64_t size){
    for (int f=0; f<lambda; f++){
        uint64_t alphaf = size/lambda + (f<(int)(size%lambda));
        for (int i=0; i<k; i++){
            for (uint64_t j=nfk[f*k+i]; j<alphaf; j++){
                bfk[f*k+i] += (accumul_t)buffer[f+j*lambda];
            }
            bfk_mpfr[f*k+i] += bfk[f*k+i]; // Updating precision value
            bfk[f*k+i] = 0; // Reseting accumulator
        }
    }
    // Hijacking bfk to compute bk too
    accumulate_bk(buffer,size); // Not required but fast and helps testing
}

// gk is the beginning corrections without a phase reference
// It should be computed ONCE per block on the very first chunk of data
template<class T>
inline void ACorrUpToPhi<T>::accumulate_gk(T *buffer, uint64_t size){
    for (int i=0; i<k; i++){
```

```
            for (int j=0; j<i; j++){
                gk[i] += (accumul_t)buffer[j];
            }
            gk_mpfr[i] += gk[i]; // Updating precision value
            gk[i] = 0; // Reseting accumulator
        }
    }

    // bk is the end corrections without a phase reference
    // It should be computed ONCE per block on the very last chunk of data
    template<class T>
    inline void ACorrUpToPhi<T>::accumulate_bk(T *buffer, uint64_t size){
        for (int i=0; i<k; i++){
            for (uint64_t j=size-i; j<size; j++){
                bk[i] += (accumul_t)buffer[j];
            }
            bk_mpfr[i] += bk[i]; // Updating precision value
            bk[i] = 0; // Reseting accumulator
        }
    }

    template<class T>
    inline void ACorrUpToPhi<T>::update(){
        update_mpfr();
        reset_accumulators();
    }

    template<class T>
    inline void ACorrUpToPhi<T>::update_mpfr(){
        for (int f=0; f<lambda; f++){
            mf_mpfr[f] += mf[f];
            for (int i=0; i<k; i++){
                rfk_mpfr[f*k+i] += rfk[f*k+i];
                // bfk/gfk accumulated in their own respective function
            }
        }
    }

    template<class T>
    inline void ACorrUpToPhi<T>::reset_accumulators(){
        for (int f=0; f<lambda; f++){
            mf[f] = 0;
            for (int i=0; i<k; i++){
                rfk[f*k+i] = 0;
                // bfk/gfk are reset in their own respective function
            }
        }
    }

    template<class T>
    inline mpreal* ACorrUpToPhi<T>::get_aCorrs_mpfr(){
        // No corr across blocks: i -> i*block_processed
        for (int i=0; i<k; i++){
            for (int f=0; f<lambda; f++){
                //aCorrs_mpfr[f*k+i] = (rfk_mpfr[f*k+i] - (mf_mpfr[f] - bfk_mpfr[f*k+i]) *
                ↪  (mf_mpfr[f] - gfk_mpfr[f*k+i])/Nfk_mpfr[f*k+i])/Nfk_mpfr[f*k+i];
                // Corrected expression follows
                aCorrs_mpfr[f*k+i] = (rfk_mpfr[f*k+i] - (mf_mpfr[f] - bfk_mpfr[f*k+i]) *
                ↪  (mf_mpfr[(f+i)%lambda] -
                ↪  gfk_mpfr[f*k+i])/Nfk_mpfr[f*k+i])/Nfk_mpfr[f*k+i];
            }
        }
        return aCorrs_mpfr; // Return pointer to array
    }

    // Should be exact!
    template<class T>
    inline void ACorrUpToPhi<T>::get_aCorrs0(){
        // Result that will be cast to double into ak
        mpreal* ak_mpfr = new mpreal [k]();

        // Accumulators that we sum over phases
        mpreal* rk = new mpreal [k]();
        for (int f=0; f<lambda; f++){
            for (int i=0; i<k; i++){
                rk[i] += rfk_mpfr[f*k+i];
            }
        }
        mpreal* nk = new mpreal [k]();
        for (int i=0; i<k; i++){
            nk[i] = (mpreal)n - (mpreal)(i*block_processed);
        }

        // We could ensure \sum_\phi gfk = gk and same for bk

        // M summed over phases
        mpreal m = 0;
        for (int f=0; f<lambda; f++){
            m += mf_mpfr[f];
        }

        // COMPUTE!
        for (int i=0; i<k; i++){
            ak_mpfr[i] = (rk[i] - (m-bk_mpfr[i])*(m-gk_mpfr[i])/nk[i])/nk[i];
        }

        // TO DOULBES!
        for (int i=0; i<k; i++){
            ak[i] = (double)ak_mpfr[i];
        }

        delete[] ak_mpfr;
        delete[] rk;
        delete[] nk;
    }

    template<class T>
    inline void ACorrUpToPhi<T>::compute_aCorrs(){
        get_aCorrs_mpfr();
        for (int l=0; l<k*lambda; l++){ // Single loop because there are no f-specific value
            aCorrs[l] = (double)aCorrs_mpfr[l]; // Small loss of precision here
        }
    }

    template<class T>
    inline double* ACorrUpToPhi<T>::get_aCorrs(){
        compute_aCorrs();
        return aCorrs; // Return pointer to array
    }

    template<class T>
    inline void ACorrUpToPhi<T>::get_aCorrs(double* res, int size){
        compute_aCorrs();
        // Size has to equal lambda*k
        for (int i=0; i<size; i++){
            res[i] = aCorrs[i];
        }
    }

    template<class T>
    inline void ACorrUpToPhi<T>::get_rfk(double* res, int size){
            if (size>k){
                size = k;
            }
            // Size has to equal lambda*k
            for (int i=0; i<size; i++){
                res[i] = (double)rfk[i];
            }
        }


    // Max chunk_size to avoid overflow: chunk_size*buff_max² == accumul_max
    // Relevant quantities are max(accumul_t) and max(abs(min(buff)), max(buff))
    // e.g. int16 buff spanning -2^15:2^15-1 == -32768:32767 in int64 accumulator:
    //   - max positive buff squared value is (-2^15)^2 = 2^30 = 1073741824
    //   - max negative buff squared value is -2^15*(2^15-1) = -2^30+2^15 = -1073709056
    //   - accumulator max positive value is 2^63-1 -> (2^63-1)/2^30 = (2^33 - 1) + (1 -
    ↪  2^-30)
    //       - With result stated as a positive sum of the integer and fractional parts
    //       - Casting back to int64 yields 2^33-1 = 8589934591
    //   - accumulator max negative value is -2^63
    //          -> -2^63/(-2^15*(2^15-1) = 2^33/(1-2^-15) = (2^33+2**18)/(1-2**-30)
    //          -> 2^33 + 2^18 + epsilon (first order Taylor, positive epsilon tends to 0)
    //       - Casting back to int64 yields 2^33 + 2^18 + 0 = 8590196736 > 8589934591
    //   - The chunk_size is thus the smallest results: 8589934591
    // e.g. uint16 spanning 0:2^16-1 in uint64 accumulator
```

```
//   - Similarly: 2^64-1/(2^16-1)^2 = 2^32 + 2^17 - 1 + 4 + epsilon -> 4295098371
template<class T>
inline uint64_t ACorrUpToPhi<T>::compute_chunk_size(){
    uint64_t buff_max = max(abs(numeric_limits<T>::min()),
    ↪  abs(numeric_limits<T>::max()));
    uint64_t accumul_max = numeric_limits<accumul_t>::max();
    uint64_t ret = accumul_max/(buff_max*buff_max);
    return ret; // Int division removes fractional possibility
}


// Destructor //
template<class T>
inline ACorrUpToPhi<T>::~ACorrUpToPhi(){
    delete[] mf;
    delete[] nfk;
    delete[] rfk;
    delete[] gfk;
    delete[] bfk;
    delete[] gk;
    delete[] bk;
    delete[] mf_mpfr;
    delete[] Nfk_mpfr;
    delete[] rfk_mpfr;
    delete[] gfk_mpfr;
    delete[] bfk_mpfr;
    delete[] gk_mpfr;
    delete[] bk_mpfr;

    delete[] aCorrs;
    delete[] aCorrs_mpfr;

    delete[] ak;
}
```

src/acorrsPhi.cpp

```
#include "acorrsPhi.tpp"

// Instantiating classes for the types we want to support
template class ACorrUpToPhi<uint8_t>;
template class ACorrUpToPhi<int8_t>;
template class ACorrUpToPhi<uint16_t>;
template class ACorrUpToPhi<int16_t>;
```

src/acorrs_wrapper.cpp

```
#include <pybind11/pybind11.h>
#include <pybind11/numpy.h>
#include <pybind11/embed.h>
#include <pybind11/stl.h>
#include <string.h>
#include <vector>
#include "acorrs.hpp"
#include "acorrsFFT.hpp"
#include "acorrsPhi.hpp"


namespace py = pybind11;

//TODO: Minimize redundant code by somehow integrating both declaration classes?

// Equivalent to "from decimal import Decimal"
py::object Decimal = py::module::import("decimal").attr("Decimal");

template<typename T>
void declare_class(py::module &m, std::string typestr) {
    using Class = ACorrUpTo<T>;
    std::string pyclass_name = std::string("ACorrUpTo_") + typestr;
    py::class_<Class>(m, pyclass_name.c_str(), py::buffer_protocol(), py::dynamic_attr())
        .def(py::init<int>())
        .def("accumulate", [](Class& self, py::array_t<T, py::array::c_style>& array) {
            auto buff = array.request();
            pybind11::gil_scoped_release release;
            self.accumulate((T*)buff.ptr, buff.size);
            }
        )
        .def("accumulate_m_rk", [](Class& self, py::array_t<T, py::array::c_style>&
        ↪  array) {
            auto buff = array.request();
            pybind11::gil_scoped_release release;
            self.accumulate_m_rk((T*)buff.ptr, buff.size);
            }
        )
        .def("compute_aCorrs", &Class::compute_aCorrs)
        .def("get_aCorrs", [](Class& self) {
            return py::array_t<double>(
                {self.k,},          // shape
                {sizeof(double),},  // C-style contiguous strides for double
                self.aCorrs,        // the data pointer
                NULL);              // numpy array references this parent
            }
        )
        .def("__call__", [](Class& self, py::array_t<T, py::array::c_style>& array) {
            auto buff = array.request();
            pybind11::gil_scoped_release release;
            self.accumulate((T*)buff.ptr, buff.size);
            self.compute_aCorrs();
            }
        )

        .def_property_readonly("res", [](Class& self){
            double *tmp;
            if (self.n){
                tmp = self.get_aCorrs();
            }
            else {
                tmp = self.aCorrs;
            }
            return py::array_t<double>(
                {self.k,},          // shape
                {sizeof(double),},  // C-style contiguous strides for double
                tmp,                // the data pointer
                NULL);              // numpy array references this parent
            }
        )
        .def_property_readonly("rk", [](Class& self){
            vector<py::object> values;
            for (int i=0; i<self.k;
            ↪  i++){values.push_back(Decimal(self.rk_mpfr[i].toString()));}
            return py::array(py::cast(values));
            }
        )
        .def_property_readonly("bk", [](Class& self){
            vector<py::object> values;
            for (int i=0; i<self.k;
            ↪  i++){values.push_back(Decimal(self.bk_mpfr[i].toString()));}
            return py::array(py::cast(values));
            }
        )
        .def_property_readonly("gk", [](Class& self){
            vector<py::object> values;
            for (int i=0; i<self.k;
            ↪  i++){values.push_back(Decimal(self.gk_mpfr[i].toString()));}
            return py::array(py::cast(values));
            }
        )
        .def_property_readonly("m", [](Class& self){
            return Decimal(self.m_mpfr.toString());
            }
        )
        .def_property_readonly("n", [](Class& self){
            return Decimal(self.n_mpfr.toString());
```

```
            }
        )
        .def_property_readonly("k", [](Class& self) {return self.k;})
        .def_property_readonly("chunk_processed", [](Class& self) {return
        ↪  self.chunk_processed;})
        .def_property_readonly("chunk_size", [](Class& self) {return self.chunk_size;})
        .def_property_readonly("block_processed", [](Class& self) {return
        ↪  self.block_processed;})
        ;
}


template<typename T>
void declare_fftclass(py::module &m, std::string typestr) {
    using Class = ACorrUpToFFT<T>;
    std::string pyclass_name = std::string("ACorrUpToFFT_") + typestr;
    py::class_<Class>(m, pyclass_name.c_str(), py::buffer_protocol(), py::dynamic_attr())
        .def(py::init<int,int>())
        .def("accumulate", [](Class& self, py::array_t<T, py::array::c_style>& array) {
            auto buff = array.request();
            pybind11::gil_scoped_release release;
            self.accumulate((T*)buff.ptr, buff.size);
            }
        )
        .def("accumulate_m_rk", [](Class& self, py::array_t<T, py::array::c_style>&
        ↪  array) {
            auto buff = array.request();
            pybind11::gil_scoped_release release;
            self.accumulate_m_rk((T*)buff.ptr, buff.size);
            }
        )
        .def("compute_aCorrs", &Class::compute_aCorrs)
        .def("get_aCorrs", [](Class& self) {
            return py::array_t<double>(
                {self.k,},          // shape
                {sizeof(double),},  // C-style contiguous strides for double
                self.aCorrs,        // the data pointer
                NULL);              // numpy array references this parent
            }
        )
        .def("__call__", [](Class& self, py::array_t<T, py::array::c_style>& array) {
            auto buff = array.request();
            pybind11::gil_scoped_release release;
            self.accumulate((T*)buff.ptr, buff.size);
            self.compute_aCorrs();
            }
        )

        .def_property_readonly("res", [](Class& self){
            double *tmp;
            if (self.n){
                tmp = self.get_aCorrs();
            }
            else {
                tmp = self.aCorrs;
            }
            return py::array_t<double>(
                {self.k,},          // shape
                {sizeof(double),},  // C-style contiguous strides for double
                tmp,                // the data pointer
                NULL);              // numpy array references this parent
            }
        )
        .def_property_readonly("rk", [](Class& self){
            vector<py::object> values;
            for (int i=0; i<self.k;
            ↪  i++){values.push_back(Decimal(self.rk_mpfr[i].toString()));}
            return py::array(py::cast(values));
            }
        )
        .def_property_readonly("bk", [](Class& self){
            vector<py::object> values;
            for (int i=0; i<self.k;
            ↪  i++){values.push_back(Decimal(self.bk_mpfr[i].toString()));}
            return py::array(py::cast(values));
            }
        )
        .def_property_readonly("gk", [](Class& self){
            vector<py::object> values;
            for (int i=0; i<self.k;
            ↪  i++){values.push_back(Decimal(self.gk_mpfr[i].toString()));}
            return py::array(py::cast(values));
            }
        )
        .def_property_readonly("m", [](Class& self){
            return Decimal(self.m_mpfr.toString());
            }
        )
        .def_property_readonly("n", [](Class& self){
            return Decimal(self.n_mpfr.toString());
            }
        )
        .def_property_readonly("k", [](Class& self) {return self.k;})
        // Using the type of n instead of m because of &m voodoo. Always the same.
        .def_property_readonly("len", [](Class& self) {return self.len;})
        .def_property_readonly("fftwlen", [](Class& self) {return self.fftwlen;})
        .def_property_readonly("chunk_processed", [](Class& self) {return
        ↪  self.chunk_processed;})
        .def_property_readonly("chunk_size", [](Class& self) {return self.chunk_size;})
        .def_property_readonly("block_processed", [](Class& self) {return
        ↪  self.block_processed;})
        .def_property_readonly("counter_max", [](Class& self) {return self.counter_max;})
        ;
}

template<typename T>
void declare_phiclass(py::module &m, std::string typestr) {
    using Class = ACorrUpToPhi<T>;
    std::string pyclass_name = std::string("ACorrUpToPhi_") + typestr;
    py::class_<Class>(m, pyclass_name.c_str(), py::buffer_protocol(), py::dynamic_attr())
        .def(py::init<int,int>())
        .def("get_nfk", [](Class& self, uint64_t a, int b, int c, int d){
            return self.get_nfk(a,b,c,d);
            }
        )
        .def("compute_current_nfk", [](Class& self, py::array_t<T, py::array::c_style>&
        ↪  array) {
            auto buff = array.request();
            self.compute_current_nfk(buff.size);
            auto res = py::array_t<uint64_t>(
                {self.lambda*self.k,},  // shape
                {sizeof(uint64_t),},      // C-style contiguous strides for double
                self.nfk,             // the data pointer
                NULL);                  // numpy array references this parent
            res.resize({self.lambda, self.k});
            return res;
            }
        )
        .def("accumulate", [](Class& self, py::array_t<T, py::array::c_style>& array) {
            auto buff = array.request();
            pybind11::gil_scoped_release release;
            self.accumulate((T*)buff.ptr, buff.size);
            }
        )
        .def("accumulate_mf_rfk", [](Class& self, py::array_t<T, py::array::c_style>&
        ↪  array) {
            auto buff = array.request();
            pybind11::gil_scoped_release release;
            self.accumulate_mf_rfk((T*)buff.ptr, buff.size);
            }
        )
        .def("compute_aCorrs", &Class::compute_aCorrs)
        .def("get_aCorrs", [](Class& self) {
            auto res = py::array_t<double>(
                {self.lambda*self.k,},  // shape
                {sizeof(double),},      // C-style contiguous strides for double
                self.aCorrs,            // the data pointer
                NULL);                  // numpy array references this parent
            res.resize({self.lambda, self.k});
            return res;
            }
        )
        .def("__call__", [](Class& self, py::array_t<T, py::array::c_style>& array) {
            auto buff = array.request();
            pybind11::gil_scoped_release release;
            self.accumulate((T*)buff.ptr, buff.size);
```

```
            self.compute_aCorrs();
            }
        )

        .def_property_readonly("res", [](Class& self){
            double *tmp;
            if (self.nfk[0]){
                tmp = self.get_aCorrs();
            }
            else {
                tmp = self.aCorrs;
            }
            auto res = py::array_t<double>(
                {self.lambda*self.k,},  // shape
                {sizeof(double),},      // C-style contiguous strides for double
                tmp,                    // the data pointer
                NULL);                  // numpy array references this parent
            res.resize({self.lambda, self.k});
            return res;
            }
        )
        .def_property_readonly("res0", [](Class& self){
            self.get_aCorrs0();
            return py::array_t<double>(
                {self.k,},  // shape
                {sizeof(double),},      // C-style contiguous strides for double
                self.ak,                   // the data pointer
                NULL);                  // numpy array references this parent
            }
        )
        .def_property_readonly("rfk", [](Class& self){
            vector<py::object> values;
            for (int i=0; i<self.lambda*self.k;
            ↪  i++){values.push_back(Decimal(self.rfk_mpfr[i].toString()));}
            auto res = py::array(py::cast(values)); // auto saving my life here
            res.resize({self.lambda, self.k});
            return res;
            }
        )
        .def_property_readonly("nfk", [](Class& self){
            vector<py::object> values;
            for (int i=0; i<self.lambda*self.k;
            ↪  i++){values.push_back(Decimal(self.Nfk_mpfr[i].toString()));}
            auto res = py::array(py::cast(values)); // auto saving my life here
            res.resize({self.lambda, self.k});
            return res;
            }
        )
        .def_property_readonly("bfk", [](Class& self){
            vector<py::object> values;
            for (int i=0; i<self.lambda*self.k;
            ↪  i++){values.push_back(Decimal(self.bfk_mpfr[i].toString()));}
            auto res = py::array(py::cast(values)); // auto saving my life here
            res.resize({self.lambda, self.k});
            return res;
            }
        )
        .def_property_readonly("gfk", [](Class& self){
            vector<py::object> values;
            for (int i=0; i<self.lambda*self.k;
            ↪  i++){values.push_back(Decimal(self.gfk_mpfr[i].toString()));}
            auto res = py::array(py::cast(values)); // auto saving my life here
            res.resize({self.lambda, self.k});
            return res;
            }
        )
        .def_property_readonly("bk", [](Class& self){
            vector<py::object> values;
            for (int i=0; i<self.k;
            ↪  i++){values.push_back(Decimal(self.bk_mpfr[i].toString()));}
            return py::array(py::cast(values));
            }
        )
        .def_property_readonly("gk", [](Class& self){
            vector<py::object> values;
            for (int i=0; i<self.k;
            ↪  i++){values.push_back(Decimal(self.gk_mpfr[i].toString()));}
            return py::array(py::cast(values));
            }
        )
        .def_property_readonly("mf", [](Class& self){
            vector<py::object> values;
            for (int i=0; i<self.lambda;
            ↪  i++){values.push_back(Decimal(self.mf_mpfr[i].toString()));}
            return py::array(py::cast(values));
            }
        )
        .def_property_readonly("k", [](Class& self) {return self.k;})
        .def_property_readonly("n", [](Class& self) {return self.n;})
        .def_property_readonly("l", [](Class& self) {return self.lambda;})
        .def_property_readonly("chunk_processed", [](Class& self) {return
        ↪  self.chunk_processed;})
        .def_property_readonly("chunk_size", [](Class& self) {return self.chunk_size;})
        .def_property_readonly("block_processed", [](Class& self) {return
        ↪  self.block_processed;})
        ;
}


#define declare_class_for(U) declare_class<U##_t>(m, std::string(#U));
#define declare_fftclass_for(U) declare_fftclass<U##_t>(m, std::string(#U));
#define declare_phiclass_for(U) declare_phiclass<U##_t>(m, std::string(#U));

PYBIND11_MODULE(acorrs_wrapper, m) {
    m.doc() = "pybind11 wrapper for acorrs.h"; // optional module docstring
    m.attr("the_answer") = 42;
    m.def("set_mpreal_precision", &set_mpreal_precision);

    declare_class_for(uint8)
    declare_class_for(int8)
    declare_class_for(uint16)
    declare_class_for(int16)

    declare_fftclass_for(uint8)
    declare_fftclass_for(int8)
    declare_fftclass_for(uint16)
    declare_fftclass_for(int16)

    declare_phiclass_for(uint8)
    declare_phiclass_for(int8)
    declare_phiclass_for(uint16)
    declare_phiclass_for(int16)
}
```

src/acorrs_otf.py

```
#!/bin/python
# -*- coding: utf-8 -*-

import sys, os, platform, time
import numpy as np
import matplotlib.pyplot as plt
from numpy import uint8, int8, uint16, int16, double
from numpy import ndarray, ceil, log2, iinfo, zeros, allclose, arange, array
from numpy import floor, log10, savez_compressed, load
from decimal import Decimal

# Setting up the proper libraries and paths, mainly for Windows support
libpath = os.path.abspath(os.path.dirname(__file__))
plat_info = dict(plat=platform.system())
if plat_info['plat'] == 'Windows':
    plat_info['lib'] = os.path.join(libpath, 'acorrs_wrapper.pyd')
    plat_info['com'] = 'make acorrs_wrapper.pyd'
    # Adding cygwin libs path for windows
    libspath = 'C:\\cygwin64\\usr\\x86_64-w64-mingw32\\sys-root\\mingw\\bin'
    if libspath not in os.environ['PATH']:
        os.environ['PATH'] = libspath+os.path.pathsep+os.environ['PATH']
else:
    plat_info['lib'] = os.path.join(libpath, 'acorrs_wrapper.so')
    plat_info['com'] = 'make acorrs_wrapper.so'
```

```
if not os.path.isfile(plat_info['lib']):
    raise IOError("{lib} is missing. To compile on {plat}:\n{com}\n".format(**plat_info))

import acorrs_wrapper
from acorrs_wrapper import set_mpreal_precision

# Applies to instances created afterwards
set_mpreal_precision(48)

# For automatic fftchunk determination
def closest_power_of_two(x):
    a = 2**int(ceil(log2(x)))
    b = 2**int(floor(log2(x)))
    if abs(a-x)<abs(x-b):   # Biased towards smallest value if dead center
        return a
    else:
        return b

# Returns the proper class. Fancy name: factory. Ghetto name: wrapper wrapper.
def ACorrUpTo(k, data, phi=False, fft=None, fftchunk='auto', k_fft=32, k_fft_factor=16):
    if type(data) is ndarray:
        dtype = data.dtype.name
    else:
        dtype = data

    if phi:
        fft = False

    if fft is None:
        if k>=k_fft:  # k_fft is empirical for each system
            fft = True
        else:
            fft = False

    if fftchunk == 'auto':
        # Since fftwlen is a power of 2, fftchunk is optimal if it is too
        fftchunk = closest_power_of_two(k_fft_factor*k)

    if fft and k>fftchunk:
        fftchunk = int(2**ceil(log2(k))) # Ceil to power of two

    classname = "ACorrUpTo{fft}_{dtype}".format(dtype=dtype, fft="FFT" if fft else "Phi"
    ↪  if phi else "")

    if fft:
        retClass = getattr(acorrs_wrapper, classname)(k, fftchunk)
    elif phi:
        retClass = getattr(acorrs_wrapper, classname)(k, phi)
    else:
        retClass = getattr(acorrs_wrapper, classname)(k)

    if type(data) is ndarray:
        retClass(data)

    return retClass


# For testing

# Computes phase-resolved a.res using Decimal accumulators
# Casting result to double should be exactly a.res[k]
def check_ak(a,k):
    nk = a.n-k
    rk = a.rk[k]
    m = a.m
    bk = a.bk[k]
    gk = a.gk[k]
    return (rk-(m-bk)*(m-gk)/nk)/nk

# Computes phase-resolved a.res using Decimal accumulators
# Casting result to double should be exactly a.res[f,k]
def check_afk_phi(a,f,k):
    nfk = a.nfk[f,k]
    rfk = a.rfk[f,k]
    mf = a.mf[f]
    mfpk = a.mf[(f+k)%a.l]
    bfk = a.bfk[f,k]
    gfk = a.gfk[f,k]
    return (rfk-((mf-bfk)*(mfpk-gfk))/nfk)/nfk

# Computes phase-resolved a.res0 using Decimal accumulators
# Casting result to double should be exactly a.res0[k]
def check_ak_phi(a,k):
    nfk = a.nfk.sum(axis=0)[k]
    rfk = a.rfk.sum(axis=0)[k]
    mf = a.mf.sum()
    mfpk = a.mf.sum()
    bfk = a.bfk.sum(axis=0)[k]
    gfk = a.gfk.sum(axis=0)[k]
    return (rfk-((mf-bfk)*(mfpk-gfk))/nfk)/nfk

# Converts an ACorrUpTo object to a dict with the same information
def a_to_dict_phi(a):
    ks = 'bk block_processed chunk_processed chunk_size gk k m n res rk'.split(' ')
    return {k:getattr(a,k) for k in ks}

# Converts an ACorrUpToFFT object to a dict with the same information
def a_to_dict_fft(a):
    ks = 'bk block_processed chunk_processed chunk_size counter_max fftwlen gk k len m n
    ↪  res rk'.split(' ')
    return {k:getattr(a,k) for k in ks}

# Converts an ACorrUpToPhi object to a dict with the same information
def a_to_dict_phi(a):
    ks = 'bfk bk block_processed chunk_processed chunk_size gfk gk k l mf n nfk res res0
    ↪  rfk'.split(' ')
    return {k:getattr(a,k) for k in ks}
```

makefile

```
# Toolchain, using mingw on windows under cywgin
CXX = $(OS:Windows_NT=x86_64-w64-mingw32-)g++
CP = cp
RM = rm
PY = $(OS:Windows_NT=/c/Anaconda2/)python

# flags
CFLAGS = -Ofast -march=native -std=c++11 -MMD -MP -Wall $(OS:Windows_NT=-DMS_WIN64
↪  -D_hypot=hypot)
OMPFLAGS = -fopenmp -fopenmp-simd
SHRFLAGS = -fPIC -shared
FFTWFLAGS = -lfftw3 -lm

# includes
PYINCL = `$(PY) -m pybind11 --includes`
ifneq ($(OS),Windows_NT)
    PYINCL += -I /usr/include/python2.7/
endif

# libraries
LDLIBS = -lmpfr $(OS:Windows_NT=-L /c/Anaconda2/ -l python27) $(PYINCL)

# directories
OBJ_DIR = obj
SRC_DIR = src
BIN_DIR = bin

# filenames
BIN := acorrs_wrapper
PYBIN := acorrs_otf.py
EXT := $(if $(filter $(OS),Windows_NT),pyd,so)
SRCS := $(shell find $(SRC_DIR) -name *.cpp)
OBJS := $(SRCS:$(SRC_DIR)/%.cpp=$(OBJ_DIR)/%.o)
DEPS := $(OBJS:.o=.d)
TARGET := $(BIN_DIR)/$(BIN).$(EXT)
PYTARGET := $(BIN_DIR)/$(PYBIN)

```

```
all: $(TARGET) $(PYTARGET)

$(TARGET): $(OBJS)
	$(CXX) -o $(TARGET) $(OBJS) $(SHRFLAGS) $(CFLAGS) $(OMPFLAGS) $(FFTWFLAGS)
	↪  $(LDLIBS)

# compile source
$(OBJ_DIR)/%.o: $(SRC_DIR)/%.cpp
	$(CXX) $(SHRFLAGS) $(CFLAGS) $(OMPFLAGS) $(LDLIBS) -c $< -o $@

# bring python files along
$(BIN_DIR)/%.py: $(SRC_DIR)/%.py
	$(CP) $< $@

clean:
	$(RM) -f $(OBJS) $(DEPS)
	$(RM) -f $(SRC_DIR)/*.pyc $(BIN_DIR)/*.pyc

clean-all: clean
	[ -f $(TARGET) ] && $(RM) $(TARGET) || true
	[ -f $(PYTARGET) ] && $(RM) $(PYTARGET) || true

-include $(DEPS)

.PHONY: all clean clean-all
```

B.2.3 Remdet

src/common.hpp

```
#ifndef common_H
#define common_H

#if defined(__CYGWIN__) || defined(__MINGW64__)
    // see number from: sdkddkver.h
    // https://docs.microsoft.com/fr-fr/windows/desktop/WinProg/using-the-windows-headers
    #define _WIN32_WINNT 0x0602 // Windows 8
    #include <windows.h>
    #include <Processtopologyapi.h>
    #include <processthreadsapi.h>
#endif

#define NOOP (void)0

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <math.h>

#include <iostream>
#include <iomanip>

#include <omp.h>
#include <limits>


#ifdef _WIN32_WINNT
    #include "mpreal.h"
#else
    #include <mpreal.h>
#endif


#define __STDC_FORMAT_MACROS
#include <inttypes.h>

using namespace std;
using mpfr::mpreal;

// For setting desired mpreal precision beforehand
void set_mpreal_precision(int d);

void manage_thread_affinity();


#endif // common_H
```

src/common.cpp

```
#include "common.hpp"

using namespace std;

// For setting desired mpreal precision beforehand
void set_mpreal_precision(int d){
    // Before any mreal are created
    const int digits = d; // Setting high precision
    mpreal::set_default_prec(mpfr::digits2bits(digits));
}

void manage_thread_affinity()
{
    #ifdef _WIN32_WINNT
        int nbgroups = GetActiveProcessorGroupCount();
        int *threads_per_groups = (int *) malloc(nbgroups*sizeof(int));
        for (int i=0; i<nbgroups; i++)
        {
            threads_per_groups[i] = GetActiveProcessorCount(i);
        }

        // Fetching thread number and assigning it to cores
        int tid = omp_get_thread_num(); // Internal omp thread number (0 --
        ↪  OMP_NUM_THREADS)
        HANDLE thandle = GetCurrentThread();
        bool result;

        WORD set_group = tid%nbgroups; // We change group for each thread
        int nbthreads = threads_per_groups[set_group]; // Nb of threads in group for
        ↪  affinity mask.
        GROUP_AFFINITY group = {((uint64_t)1<<nbthreads)-1, set_group}; // nbcores amount
        ↪  of 1 in binary

        result = SetThreadGroupAffinity(thandle, &group, NULL); // Actually setting the
        ↪  affinity
        if(!result) fprintf(stderr, "Failed setting output for tid=%i\n", tid);
    #else
        //We let openmp and the OS manage the threads themselves
    #endif
}
```

src/remdet.hpp

```
#ifndef remdet_H
#define remdet_H

#include "common.hpp"

// Computes average signal on block of length *period* and removes it from
// buffer. Also returns the deterministic signal for a single period.
```

```

// All-in-one, initial detpart should be filled with 0x00
template<typename T> void remdet(T *buffer, uint64_t size, T *detpart, uint64_t period);
// Just get the det part, detpart should be zeroed
template<typename T> void getdet(T *buffer, uint64_t size, T *detpart, uint64_t period);
// Deletes the det part in-place inside buffer, detpart should be the real thing
template<typename T> void deldet(T *buffer, uint64_t size, T *detpart, uint64_t period);

// For T-specialized optimisations
template<typename T> inline void detsum_func(T *buff, auto *detsum, uint64_t period);
template<typename T> inline void detsub_func(T *buff, T *detpart, uint64_t period);

#endif // remdet_H
```

src/remdet.tpp

```

#include "remdet.hpp"

using namespace std;

// Computes average signal on block of length *period* and removes it from
// buffer. Also returns the deterministic signal for a single period.
template<typename T>
void remdet(T *buffer, uint64_t size, T *detpart, uint64_t period)
{
    getdet(buffer, size, detpart, period);  // detpart is now the det part
    deldet(buffer, size, detpart, period);  // detpart is removed from buffer
}

template<typename T>
void getdet(T *buffer, uint64_t size, T *detpart, uint64_t period)
{
    // detsum should have same sign as T for performance
    typedef typename conditional<((T)-1>0), uint64_t, int64_t>::type detsum_t;
    detsum_t *detsum = new detsum_t [period](); // Initialized to 0x00s
    uint64_t numblocks = size/period;
    // Whole blocks
    #pragma omp parallel
    {
        manage_thread_affinity();
        #pragma omp for simd reduction(+:detsum[:period])
        for (uint64_t i=0; i<numblocks; i++){
            T *buff = buffer + i*period;
            detsum_func(buff, detsum, period);
        }
        // Remainder
        T *buff = buffer + numblocks*period;
        #pragma omp for simd reduction(+:detsum[:period])
        for (uint64_t j=0; j<size%period; j++){
            detsum[j] += buff[j];
        }
        #pragma omp for simd reduction(+:detpart[:period])
        for (uint64_t j=0; j<period; j++){
            // size%period first values are averaged on numblocks+1 samples
            uint64_t N = numblocks + (j < size%period);
            detpart[j] = (T)(double)mpfr::rint_round((mpreal)detsum[j]/(mpreal)N);
            // The above should improve average accuracy via rounding in mpfr
            // Converting to double first to avoid using T-specific functions
            // Ensuing loss of precision should be less than discret«ization
        }
    }
    delete [] detsum;   // Clearing memory
}

template<typename T>
void deldet(T *buffer, uint64_t size, T *detpart, uint64_t period)
{
    // detpart is the actual deterministic part
    uint64_t numblocks = size/period; // Don't care about the remainder for now
    // Whole blocks
    #pragma omp parallel
    {
        manage_thread_affinity();
        #pragma omp for simd
        for (uint64_t i=0; i<numblocks; i++){
            T *buff = buffer + i*period;
            detsub_func(buff, detpart, period);
        }
        /*#pragma omp for simd // Was much slower, prob bc of the mod
        for (uint64_t i=0; i<size; i++){    // No need for remainder
            buffer[i] -= detpart[i%period];
        }*/
    }
    // Remainder
    T *buff = buffer + numblocks*period;
    for (uint64_t j=0; j<size%period; j++){
        buff[j] -= detpart[j];
    }
}

// To allow for T-specific optimisations

template<typename T>
inline void detsum_func(T *buff, auto *detsum, uint64_t period){
    for (uint64_t j=0; j<period; j++){
        detsum[j] += buff[j];
    }
}

template<typename T>
inline void detsub_func(T *buff, T *detpart, uint64_t period){
    for (uint64_t j=0; j<period; j++){
        buff[j] -= detpart[j];
    }
}


// SPECIALIZATIONS //
// This didn't actually help at all; same performance.
/*
template <>
inline void detsum_func(int16_t *buff, int64_t *detsum, uint64_t period){
    // detsum is int64_t if T is int16_t
    uint64_t *buff_64 = (uint64_t *) buff;
    uint64_t tmp;
    // Reading block of 64 bits
    #pragma omp simd
    for (uint64_t j=0; j<period/4; j++){
        tmp = buff_64[j];
        detsum[j*4+0] += (int16_t)(tmp       & 0xFFFF);
        detsum[j*4+1] += (int16_t)(tmp >> 16 & 0xFFFF);
        detsum[j*4+2] += (int16_t)(tmp >> 32 & 0xFFFF);
        detsum[j*4+3] += (int16_t)(tmp >> 48 & 0xFFFF);
    }
    // Remainder
    for (uint64_t j=period-period%4; j<period; j++){
        detsum[j] += buff[j];
    }
}
*/

// This also didn't speed things up; kept as an AVX example.
/*
#include <immintrin.h>
#include <smmintrin.h>
template <>
void deldet(int16_t *buffer, uint64_t size, int16_t *detpart, uint64_t period) {
    if (not period%16){ // If period is a multiple of 16
        #pragma omp parallel
        {
            manage_thread_affinity();
            #pragma omp for
            for (uint64_t i = 0; i < size; i+=16) {
                // load 256-bit chunks of each array
                __m256i first_values = _mm256_load_si256((__m256i*) &buffer[i]);
                __m256i second_values = _mm256_load_si256((__m256i*) &detpart[i%period]);

                // subs each pair of 16-bit integers in the 128-bit chunks
                first_values = _mm256_sub_epi16(first_values, second_values);

```

```
                // store 256-bit chunk to first array
                _mm256_store_si256((__m256i*) &buffer[i], first_values);
            }
        }
        // handle left-over
        for (uint64_t i = size-size%16; i < size; i++) {
            buffer[i] -= detpart[i%period];
        }
    }
    else{
        // detpart is the actual deterministic part
        uint64_t numblocks = size/period; // Don't care about the remainder for now
        // Whole blocks
        #pragma omp parallel
        {
        manage_thread_affinity();
            #pragma omp for simd
            for (uint64_t i=0; i<numblocks; i++){
                int16_t *buff = buffer + i*period;
                detsub_func(buff, detpart, period);
            }
        }
        // Remainder
        int16_t *buff = buffer + numblocks*period;
        for (uint64_t j=0; j<size%period; j++){
            buff[j] -= detpart[j];
        }
    }
}
*/
```

src/remdet.cpp

```
#include "remdet.tpp"

// Instantiating the types we want to support
#define declare_F_for_U(F,U) template void F<U##_t>(U##_t *buffer, uint64_t size, U##_t *detpart, uint64_t period);
#define declare_allF_for_U(U) \
    declare_F_for_U(getdet, U);\
    declare_F_for_U(deldet, U);\
    declare_F_for_U(remdet, U);

declare_allF_for_U(int8);
declare_allF_for_U(uint8);
declare_allF_for_U(int16);
declare_allF_for_U(uint16);
```

src/remdet_wrapper.cpp

```
#include <pybind11/pybind11.h>
#include <pybind11/numpy.h>
#include "remdet.hpp"


namespace py = pybind11;

template<typename T>
void declare_functions(py::module &m) {
    m.def("remdet", [](py::array_t<T, py::array::c_style>& array, uint64_t period) {
        auto buff = array.request();
        T *detpart = new T [period]();
        py::capsule free_when_done(detpart, [](void *f) {
            T *detpart = reinterpret_cast<T *>(f);
            delete[] detpart;
            });
        remdet((T*)buff.ptr, buff.size, detpart, period);
        return py::array_t<T>(
            {period,},      // shape
            {sizeof(T),},   // C-style contiguous strides for double
            detpart,        // the data pointer
            free_when_done);// numpy array references this parent
        }
    );
    m.def("getdet", [](py::array_t<T, py::array::c_style>& array, uint64_t period) {
        auto buff = array.request();

        T *detpart = new T [period]();
        py::capsule free_when_done(detpart, [](void *f) {
            T *detpart = reinterpret_cast<T *>(f);
            delete[] detpart;
            });
        getdet((T*)buff.ptr, buff.size, detpart, period);
        return py::array_t<T>(
            {period,},      // shape
            {sizeof(T),},   // C-style contiguous strides for double
            detpart,        // the data pointer
            free_when_done);// numpy array references this parent
        }
    );
    m.def("deldet", [](py::array_t<T, py::array::c_style>& array,
                       py::array_t<T, py::array::c_style>& detpart) {
        auto buff = array.request();
        auto dpart = detpart.request();
        deldet((T*)buff.ptr, buff.size, (T*)dpart.ptr, dpart.size);
        }
    );

}

PYBIND11_MODULE(remdet_wrapper, m) {
    m.doc() = "pybind11 wrapper for remdet"; // optional module docstring
    m.attr("the_answer") = 42;
    m.def("set_mpreal_precision", &set_mpreal_precision);
    m.def("set_num_threads", &omp_set_num_threads);

    set_mpreal_precision(100);  // Precision for all that's to follow

    declare_functions<int8_t>(m);
    declare_functions<uint8_t>(m);
    declare_functions<int16_t>(m);
    declare_functions<uint16_t>(m);
}
```

src/remdet.py

```
#!/bin/python
# -*- coding: utf-8 -*-

import os as _os
import platform as _platform

# Setting up the proper libraries and paths, mainly for Windows support
_libpath = _os.path.abspath(_os.path.dirname(__file__))
_plat_info = dict(plat=_platform.system())
if _plat_info['plat'] == 'Windows':
    _plat_info['lib'] = _os.path.join(_libpath, 'remdet_wrapper.pyd')
    _plat_info['com'] = 'make remdet_wrapper.pyd'
    # Adding cygwin libs path for windows
    _libspath = 'C:\\cygwin64\\usr\\x86_64-w64-mingw32\\sys-root\\mingw\\bin'
    if _libspath not in _os.environ['PATH']:
        _os.environ['PATH'] = _libspath+_os.path.pathsep+_os.environ['PATH']
else:
    _plat_info['lib'] = _os.path.join(_libpath, 'remdet_wrapper.so')
    _plat_info['com'] = 'make remdet_wrapper.so'

if not _os.path.isfile(_plat_info['lib']):
```

```
        raise IOError("{lib} is missing. To compile on
        ↪ {plat}:\n{com}\n".format(**_plat_info))

from remdet_wrapper import *
```

makefile

```
# Toolchain, using mingw on windows under cywgin
CXX = $(OS:Windows_NT=x86_64-w64-mingw32-)g++
CP = cp
RM = rm
PY = $(OS:Windows_NT=/c/Anaconda2/)python

# flags
CFLAGS = -Ofast -march=native -std=c++14 -MMD -MP -Wall $(OS:Windows_NT=-DMS_WIN64
↪ -D_hypot=hypot)
OMPFLAGS = -fopenmp -fopenmp-simd
SHRFLAGS = -fPIC -shared
FFTWFLAGS = -lfftw3 -lm

# includes
PYINCL = `$(PY) -m pybind11 --includes`
ifneq ($(OS),Windows_NT)
    PYINCL += -I /usr/include/python2.7/
endif

# libraries
LDLIBS = -lmpfr $(OS:Windows_NT=-L /c/Anaconda2/libs/ -l python27) $(PYINCL)

# directories
OBJ_DIR = obj
SRC_DIR = src
BIN_DIR = bin

# filenames
BIN := remdet_wrapper
PYBIN := remdet.py
EXT := $(if $(filter $(OS),Windows_NT),pyd,so)
SRCS := $(shell find $(SRC_DIR) -name *.cpp)
OBJS := $(SRCS:$(SRC_DIR)/%.cpp=$(OBJ_DIR)/%.o)
DEPS := $(OBJS:.o=.d)
TARGET := $(BIN_DIR)/$(BIN).$(EXT)
PYTARGET := $(BIN_DIR)/$(PYBIN)


all: $(TARGET) $(PYTARGET)

$(TARGET): $(OBJS)
        $(CXX) -o $(TARGET) $(OBJS) $(SHRFLAGS) $(CFLAGS) $(OMPFLAGS) $(LDLIBS)

# compile source
$(OBJ_DIR)/%.o: $(SRC_DIR)/%.cpp
        $(CXX) $(SHRFLAGS) $(CFLAGS) $(OMPFLAGS) $(LDLIBS) -c $< -o $@

# bring python files along
$(BIN_DIR)/%.py: $(SRC_DIR)/%.py
        $(CP) $< $@

clean:
        $(RM) -f $(OBJS) $(DEPS)
        $(RM) -f $(SRC_DIR)/*.pyc $(BIN_DIR)/*.pyc

clean-all: clean
        [ -f $(TARGET) ] && $(RM) $(TARGET) || true
        [ -f $(PYTARGET) ] && $(RM) $(PYTARGET) || true

-include $(DEPS)

.PHONY: all clean clean-all
```

Annexe C

Scripts expérimentaux

C.1 Stabilisation de la phase

C.1.1 Extraction de la phase

```
def get_or_getcache(dev, use_cache, *args, **kwargs):
    if use_cache == 'prev':
        return dev.alias._prev
    elif use_cache:
        try:
            return dev.getcache(*args, **kwargs)
        except:
            return dev.alias.getcache(*args, **kwargs)
    else:
        return get(dev, *args, **kwargs)

def coherent_spectrum(v, k, repeat_k=8, sr=32e9, polar=False, norm=False, nozero=False, *args, **kwargs):
    w = getdet(v,k)
    z = zeros((repeat_k,len(w)),dtype=w.dtype)
    z[:] = w
    z.shape = len(w)*repeat_k,
    x = fftshift(fft.fft(z))
    x = x if not norm else abs(x)
    f = fftshift(fftfreq(len(x),1./sr))
    if nozero:
        x[where(f==0)]=None
    if polar:
        return x
    else:
        return array([f,x])

def coherent_cw(v, k, f0, *args, **kwargs):
    kwargs.pop("polar",0)
    ff,cc = coherent_spectrum(v,k,polar=False,*args,**kwargs)
    idx = argmin(abs(ff-f0))
    f,c = ff[idx].real, cc[idx]
    if not isclose(f,f0):
        print("{:.6f} MHz selected instead of {:.6f} MHz".format(f/1.e6,f0/1.e6))
    return c

def gz_cohcw_wrap(f0=4e9, k=128, out_fmt="complex", use_cache=False, *args, **kwargs):
    v = get_or_getcache(gz, use_cache);
    w = coherent_cw(v, 2*k, f0, *args, **kwargs)
    if out_fmt.lower()=="complex":
        return w
    elif out_fmt.lower() in ["realimag","ri"]:
        return r_[w.real,w.imag]
    elif out_fmt.lower() in ["rtheta", "rt"]:
        return r_[abs(w),angle(w)]
    else:
        raise KeyError("Invalid output format. Valid values are 'complex', 'realimag', and 'rtheta'.")
```

```
47
48  def gz_theta_wrap(*args, **kwargs):
49      kwargs.pop('out_fmt', 0)
50      r,t = get(gz_cohcw, out_fmt='rt', *args, **kwargs)
51      return t
52
53  gz_cohcw = instruments.FunctionWrap(getfunc=gz_cohcw_wrap, basedev=gz.fetch, multi=('GZ_Coherent_CW',))
54  gz_cohcw_theta = instruments.FunctionWrap(getfunc=gz_theta_wrap, basedev=gz.fetch, multi=('GZ_Coherent_CW_Theta',))
```

C.1.2 Ajustement de la phase

```
1   def gz_phi_wrap(phi=-90, f0=fac, delta=delta_phi, k=phi_acorrs, mode='312.5',verbose=True, max_retry=10, *args, **kwargs):
2       if mode == "625ps":
3           epsilon = 1
4           dt = .5
5           tmax = 625  # Much more?
6       else:
7           epsilon = .5
8           dt = .25
9           tmax = 312.5
10      theta = get(gz_cohcw_theta, k=k, f0=f0, *args, **kwargs)*180./pi
11      if phi == None:
12          return theta
13      # Trying to keep deltas as small as possible considering mod 360
14      d = (theta-phi)+(arange(-2,3)*360)
15      d = d[argmin(abs(d))]
16      for i in range(max_retry):
17          if abs(d) > delta:
18              d0 = get_or_getcache(delay1, use_cache=False)
19              target = (d0+(d/360.)*(epsilon*1e12/f0))
20              target = dt*round(target/dt)
21              if target < 0:
22                  target += (epsilon*1e12/f0)
23
24              if target > tmax:
25                  target -= (epsilon*1e12/f0)
26
27              set(delay1, target)
28              snap([delay1, yo10, rs1_power_extended], 'delay1_change.log')
29              kwargs.pop('use_cache', False)
30              theta = get(gz_cohcw_theta, k=k, f0=f0, *args, **kwargs)*180./pi
31              if verbose:
32                  print "\nCorrected Phase on {} | delta = {} deg | retry {}".format(time.strftime("%m-%d %H:%M:%S"), d, i+1)
33
34              d = (theta-phi)+(arange(-2,3)*360)
35              d = d[argmin(abs(d))]
36
37      if abs(d) > delta: # In any cycle nearby
38          print "!WARNING!: PHASE ADJUSTMENT FAILED", ' | ', 'Theta = ', theta
39      return theta
40
41  gz_phi = instruments.FunctionWrap(getfunc=gz_phi_wrap, basedev=gz.fetch, multi=('GZ_PHASE',))
```

C.2 Scripts d'acquisition

Les scripts pyHegel utilisés pour l'acquisition des données sont listés ci-bas.

C.2.1 Mesure large bande avec biais DC seulement

On mesure les autocorrélations régulières avec la jonction biaisée à différentes valeurs de tension continue.

Batch_AutoCorrs.py

```
#!/bin/env/python
#! -*- coding: utf-8 -*-
import sys, os
sys.path.append('C:/Projets/TimeDomain_Guzik/Code/Histograms-OTF/')
sys.path.append('C:/Projets/TimeDomain_Guzik/Code/aCorrs-OTF//')
from acorrs_otf import ACorrUpTo
from histograms_otf import hist1dNbits, hist2dNbits, set_num_threads, cumulant, moment
from threading import Thread

# Mesures SANS filtre.
#
# On veut avoir des autocorrelations et spectres (via FFT) à divers bias sur la jonction.

###################
# NOTE IMPORTANTE #
###################


# Ici on a aucune atténuation devant l'ampli à chaud et 8 dB de gain au Guzik.
# FULLBAND

# EQUALIZER ON FOR THIS ONE
# NO HISTOGRAMS FOR AS MUCH DATA AS POSSIBLE IN ACORRS FOR THE TIME

test = False

make_dir("C://Projets/TimeDomain_Guzik/Data/%D/AutoCorrs_Prelim/" + ("test" if test else ""))


# Test de timing: 98s/2sweep = 49s/sweep (2.333s/meas).
# Aiming for Monday July 29th 13h00 ~145 hours: 145*3600/(49.) [h*(s/h)/(s/sweep)] = 10653.06 sweeps

# VARIABLES
n_measures = 2 if test else 10651
gz_samples = 1*2**30 if not test else 1*2**30    # 52.5 GiB is the max value
gz_gain = 8
gz_equalizer = 1
num_acorrs = 128 if not test else 128
fft_acorrs = True
fftchunk = 16*num_acorrs # Sweet-spot empirique pour num_acorrs grands, 8 est p-ê mieux pour petits
hist_bits = 16 if gz_equalizer else 10
snipsize = 1001
num_moment = 6



biases = r_[linspace(0,1,17),linspace(2,3.5,4)] # 26 points
updown = False

# Loading devices
try:
    if not isinstance(gz, instruments.guzik.guzik_adp7104):
        gz = instruments.guzik_adp7104()
except:
    gz = instruments.guzik_adp7104()
gz.config([1], n_S_ch=gz_samples, gain_dB=gz_gain, bits_16=True, equalizer_en=gz_equalizer)
load("yo10")
set(yo10,0)
set(yo10.output_en, True)
set(yo10.range,abs(biases).max())
load("dmm12")
set(dmm12.aperture, .5)

# Virtual Devices
def get_or_getcache(dev, use_cache, *args, **kwargs):
    if use_cache:
        try:
            return dev.getcache(*args, **kwargs)
        except:
            return dev.alias.getcache(*args, **kwargs)
    else:
        return get(dev, *args, **kwargs)

def gz_snip_wrap(*args, **kwargs):
    v = get_or_getcache(gz, kwargs.pop("use_cache", False))
    snip = v[:snipsize].copy()
    return snip
def gz_h_wrap(*args, **kwargs):
    ihist = kwargs.pop("ihist", None)
    v = get_or_getcache(gz, kwargs.pop("use_cache", False));
    h = hist1dNbits(v, hist_bits, ihist=ihist)
    return h
def gz_cumul_wrap(*args, **kwargs):
    h = gz_h_wrap(*args, **kwargs)
    return r_[[cumulant(h,n) for n in range(1, kwargs.pop("max", num_cumul+1))]]
def gz_ac_wrap(*args, **kwargs):
    v = get_or_getcache(gz, kwargs.pop("use_cache", False));
    a = ACorrUpTo(num_acorrs, v)
    return a.res

## The actual virtual instruments
gz_snip = instruments.FunctionWrap(getfunc=gz_snip_wrap, basedev=gz.fetch, multi=('Snippet',))
gz_h = instruments.FunctionWrap(getfunc=gz_h_wrap, basedev=gz.fetch, multi=('Histogram',))
gz_cumul = instruments.FunctionWrap(getfunc=gz_cumul_wrap, basedev=gz.fetch, multi=('Cumulants (1-6)',))
gz_ac = instruments.FunctionWrap(getfunc=gz_ac_wrap, basedev=gz.fetch, multi=('ACorrUpTo{}'.format(num_acorrs),))


# The EXPERIMENT

# Dummy sweep to save devices config
if not test:
    sweep(yo10, 0,0,1, out=[(gz_h, dict(bin='.npz')),dmm12], filename='Config_Dummy_Sweep_%D.txt', close_after=True)

# Saving biases to file
respath = get(sweep.path)
savez_compressed(os.path.join(respath, 'v_biases{}.npz'.format('_test' if test else '')), biases)


# Data structures that will be modified during measurements
acorr = [ACorrUpTo(num_acorrs, 'int16', fft=fft_acorrs, fftchunk=fftchunk) for i in biases] # ACorrUpTo classes
```

```
acorr_res = zeros((biases.size, num_acorrs), dtype=double) # Results of autocorrelations
moment_res = zeros((n_measures, biases.size, num_moment), dtype=double) # Results of autocorrelations
dmm_res = zeros((n_measures, biases.size), dtype=double) # Results of voltage measurements
h_tmp = zeros(2**hist_bits, dtype=uint64) # Histogram to build before computing moments (cumulants in analysis script)

# Thread stuff!
def acc_helper(i, x):
    acorr[i](x)
def hist_helper(x):
    h_tmp[:] = hist1dNbits(x, hist_bits)
def moment_helper(i, j):
    moment_res[i,j] = array([moment(h_tmp,l,False) for l in range(1, num_moment+1)])
def dmm_helper(i,j):
    dmm_res[i,j] = get(dmm12)

# The Measurement itself
for n in range(n_measures):
    print "Measuring iteration {:04d}/{:04d}".format(n+1, n_measures)
    m = 0
    # First voltage is done manually
    vb = biases[m]
    set(yo10.range, vb*1.01)
    set(yo10, vb)
    pause(0.75)
    t0 = Thread(target=dmm_helper, args=(n,m))
    t0.start()
    data = get(gz)
    t0.join()

    #get_helper(m%2) # Equivalent to c[0][:] = get(gz)[:]
    m += 1

    for vb in biases[1:]:
        it = time.time()
        set(yo10.range, vb*1.01)
        set(yo10, vb)

        t = Thread(target=acc_helper, args=(m-1,data))
        t.start()
        t.join()
        at = time.time()

        t5 = Thread(target=dmm_helper, args=(n,m))
        t5.start()
        data = get(gz)
        t5.join()

        ft = time.time()
        m += 1  # For next iteration
        print "  {:.1f} GiSa fetch:  {:.3f} sec  |  Set/Wait/Calc:  {:.3f} sec  |  TOTAL:  {:.3f} sec".format(gz_samples/2**30, ft-at, at-it, ft-it)

    # Computing the values on last data
    acc_helper(m-1, data)
    #hist_helper(data)
    #moment_helper(n, m-1)

    # Result array
    acorr_res[:] = array([a.res for a in acorr])[:]

    # Saving results, overwriting at each iteration in case there's an issue
    savez_compressed(os.path.join(respath, 'autocorrelations{}.npz'.format('_test' if test else '')), acorr_res)
    #savez_compressed(os.path.join(respath, 'moments{}.npz'.format('_test' if test else '')), moment_res)
    savez_compressed(os.path.join(respath, 'v_junction{}.npz'.format('_test' if test else '')), dmm_res)


# POST EXPERIMENT
set(yo10,0)
set_num_threads(36) # So the system stays responsive in interactive mode
```

C.2.2 Mesure photoexcitée résolue en phase

On fait une mesure synchrone avec la source servant à la photoexcitation sans V_{ac} et pour deux valeurs de $V_{ac} \neq 0$. On balaie aussi la polarisation en tension continue pour chacune de ces situations. On mesures les autocorrélations résolues en phase avant et après l'application de `remdet` sur les données brutes.

Sphi_Vdc_Vac_PhaseLocked_Avg_Phi_Ref.py

```
#!/bin/env/python
#! -*- coding: utf-8 -*-
import sys, os, ctypes
sys.path.append('C:/Projets/TimeDomain_Guzik/Code/Histograms-OTF/')
sys.path.append('C:/Projets/TimeDomain_Guzik/Code/aCorrs-OTF/bin/')
sys.path.append('C:/Projets/TimeDomain_Guzik/Code/remdet-OTF/bin/')
from acorrs_otf import ACorrUpTo
from histograms_otf import hist1dNbits, hist2dNbits, cumulant, moment
from remdet import getdet, deldet, remdet, set_num_threads
from threading import Thread
from itertools import product


# On veut comparer les résultats non calibrés avec ou sans Vac

test = False

make_dir("C://Projets/TimeDomain_Guzik/Data/%D/Guzik_Sphi_Vdc_Vac_PhaseLocked/" + ("test" if test else ""))


# Last sweep test de timing: ~637s / 1 iteration à 2/10 repeats et 2/8 phases -> 637*10/2*8/2 = 12740s
# Aiming for 115 hours from now:  115*3600/12740 = 32.5 sweeps
# Probably overestimating the time a little because of repeat speed gains.

########################
## DEVICES VARIABLES ##
########################
n_measures = 33 if not test else 1
n_repeat_each = 10 if not test else 2   # Speeds things up for timing test if need be
# gz
gz_samples = 1*2**30    # 52.5 GiB is the max value
gz_gain = 7
gz_equalizer = 1
gz_ref = 'ext'
bits_16 = True
# acorrs
num_acorrs = 65 # = 64+1 chosen not to slow down measurements too much
fft_acorrs = False
phi_acorrs = 8 # 32/4 = 8
fftchunk = 16*num_acorrs # Sweet-spot empirique pour num_acorrs=64, 16 est mieux pour grands délais
# moments
num_moments = 6
hist_bits = 16
# phases
```

```
phis = arange(-4,4, 1 if not test else 4)*45   # -180 à 135 par bonds de 45°, choisi pour fitter dans les délais accesibles
delta_phi = 0.5
fac = 4e9
delay_mode = '312.5ps' #'625ps'


#####################
## MANIP VARIABLES ##
#####################

# BIASES
def PdBm2Vac(PdBm):
    return 10**(PdBm/20.)*1./sqrt(10)

def Vac2PdBm(Vac):
    return 10*log10((Vac/sqrt(2))**2/50*1e3)

## Sweeps Vdc @ Vac=cte
vdc = r_[linspace(-3,-2,3),linspace(-.9,.9,13),linspace(2,3,3)]  # Plus grand pour etre plus sur que Vac a peu d'impact
# Previous values
#vac = r_[[-10,]*n_repeat_each] # -122 and less -> -inf
vac = r_[[-10,-20,-122]]

vdc_waittime = 1.0 # Seconds

########################
## SETTING UP DEVICES ##
########################

# Loading devices
try:
    if not isinstance(gz, instruments.guzik.guzik_adp7104):
        gz = instruments.guzik_adp7104()
except:
    gz = instruments.guzik_adp7104()
gz.config([1], n_S_ch=gz_samples, gain_dB=gz_gain, bits_16=bits_16, equalizer_en=gz_equalizer, ext_ref=gz_ref)
load("yo10")
set(yo10,0)
set(yo10.output_en, True)
set(yo10.range,10)
load("dmm12")
set(dmm12.aperture, .5)
set(dmm12.zero, 1)
load("att1")
set(att1, 101)
rs1 = instruments.rs_sgma('TCPIP::rssgs100a110885.mshome.net::inst0::INSTR')
set(rs1.output_en,False)
set(rs1.power_level_dbm, -20)
set(rs1.freq,fac)
set(rs1.ref_output_signal, 'ref')

def rs1_power_extended_set_wrap(p):
    if p < -121:
        set(rs1.output_en,0)
        return
    elif not get(rs1.output_en):
        set(rs1.output_en,1)
    if p < -20:
        set(att1, 101)
        set(rs1.power_level_dbm, -20+p%1)
        set(att1, -20-p + ceil(p%1))
    else:
        set(rs1.power_level_dbm, p)
        set(att1, 0)


def rs1_power_extended_get_wrap():
    if not get(rs1.output_en):
        return -inf
    tmp1 = get(rs1.power_level_dbm)
    tmp2 = get(att1)
    return tmp1-tmp2


rs1_power_extended = instruments.FunctionWrap(setfunc=rs1_power_extended_set_wrap,
                                              getfunc=rs1_power_extended_get_wrap,
                                              basedevs=[rs1,att1],
                                              multi=["rs1.output_level_dbm - att1"]
                                             )

# Setting output power to minimal value
set(rs1_power_extended, -122)


## Virtual GZ Devices
def get_or_getcache(dev, use_cache, *args, **kwargs):
    if use_cache == 'prev':
        return dev.alias._prev
    elif use_cache:
        try:
            return dev.getcache(*args, **kwargs)
        except:
            return dev.alias.getcache(*args, **kwargs)
    else:
        return get(dev, *args, **kwargs)

def gz_snip_wrap(snipsize=1000, use_cache=False, *args, **kwargs):
    v = get_or_getcache(gz, use_cache)
    snip = v[:snipsize].copy()
    return snip

def gz_h_wrap(hist_bits=16, use_cache=False, *args, **kwargs):
    v = get_or_getcache(gz, use_cache);
    h = hist1dNbits(v, hist_bits, *args, **kwargs)
    return h

def gz_moment_wrap(k=6, h=None, *args, **kwargs):
    if h is None:
        h = gz_h_wrap(*args, **kwargs)
    return r_[[moment(h,n) for n in range(1, k+1)]]

def gz_cumul_wrap(k=6, h=None, *args, **kwargs):
    if h is None:
        h = gz_h_wrap(*args, **kwargs)
    return r_[[cumulant(h,n) for n in range(1, k+1)]]

def gz_ac_wrap(k=64, use_cache=False, *args, **kwargs):
    v = get_or_getcache(gz, use_cache);
    a = ACorrUpTo(k, v, *args, **kwargs)
    return a.res

def spectrum_from_acorrs(a, rate=32e9, norm=True, *args, **kwargs):
    w = fftshift(fft.hfft(a))
    f = fftshift(fft.fftfreq(len(w), 1./rate))
    return array([f,w if not norm else abs(w)])

def gz_spec_wrap(k=128, use_cache=False, *args, **kwargs):
    v = get_or_getcache(gz, use_cache);
    a = ACorrUpTo(k, v)
    w = spectrum_from_acorrs(a.res, *args, **kwargs)
    return w

def coherent_spectrum(v, k, repeat_k=8, sr=32e9, polar=False, norm=False, nozero=False, *args, **kwargs):
    #if length is None:
    #    length = len(v)
    #if length%k:
    #    w = v[:length-(length%k)].view()
    #else:
    #    w = v[:length].view()
    #w.shape = (w.shape[0]/k,k)
    #z = w.mean(axis=0)
    w = getdet(v,k)
    z = zeros((repeat_k,len(w)),dtype=w.dtype)
    z[:] = w
    z.shape = len(w)*repeat_k,
    x = fftshift(fft.fft(z))
    x = x if not norm else abs(x)
    f = fftshift(fftfreq(len(x),1./sr))
    if nozero:
        x[where(f==0)]=None
    if polar:
        return x
```

```
        else:
            return array([f,x])


    def gz_cohpart_wrap(k=8, repeat_k=1, remove=False, use_cache=False, *args, **kwargs):
        fct = remdet if remove else getdet
        v = get_or_getcache(gz, use_cache);
        w = fct(v,k)
        z = zeros((repeat_k,len(w)),dtype=w.dtype)
        z[:] = w
        z.shape = len(w)*repeat_k,
        return z

    def gz_cohspec_wrap(k=8, repeat_k=8, use_cache=False, *args, **kwargs):
        v = get_or_getcache(gz, use_cache);
        w = coherent_spectrum(v, k, repeat_k=repeat_k, *args, **kwargs)
        return w

    def coherent_cw(v, k, f0, *args, **kwargs):
        kwargs.pop("polar",0)
        ff,cc = coherent_spectrum(v,k,polar=False,*args,**kwargs)
        idx = argmin(abs(ff-f0))
        f,c = ff[idx].real, cc[idx]
        if not isclose(f,f0):
            print("{:.6f} MHz selected instead of {:.6f} MHz".format(f/1.e6,f0/1.e6))
        return c

    def gz_cohcw_wrap(f0=4e9, k=128, out_fmt="complex", use_cache=False, *args, **kwargs):
        v = get_or_getcache(gz, use_cache);
        w = coherent_cw(v, 2*k, f0, *args, **kwargs)
        if out_fmt.lower()=="complex":
            return w
        elif out_fmt.lower() in ["realimag","ri"]:
            return r_[w.real,w.imag]
        elif out_fmt.lower() in ["rtheta", "rt"]:
            return r_[abs(w),angle(w)]
        else:
            raise KeyError("Invalid output format. Valid values are 'complex', 'realimag',
            ↪  and 'rtheta'.")

    def gz_r_wrap(*args, **kwargs):
        kwargs.pop('out_fmt', 0)
        r,t = get(gz_cohcw, out_fmt='rt', *args, **kwargs)
        return r

    def gz_theta_wrap(*args, **kwargs):
        kwargs.pop('out_fmt', 0)
        r,t = get(gz_cohcw, out_fmt='rt', *args, **kwargs)
        return t


    ## The actual virtual instruments
    gz_snip = instruments.FunctionWrap(getfunc=gz_snip_wrap, basedev=gz.fetch,
    ↪  multi=('GZ_Snippet',))
    gz_h = instruments.FunctionWrap(getfunc=gz_h_wrap, basedev=gz.fetch,
    ↪  multi=('GZ_Histogram',))
    gz_moment = instruments.FunctionWrap(getfunc=gz_moment_wrap, basedev=gz.fetch,
    ↪  multi=list('GZ_Moments',))
    gz_cumul = instruments.FunctionWrap(getfunc=gz_cumul_wrap, basedev=gz.fetch,
    ↪  multi=list('GZ_Cumulants',))
    gz_ac = instruments.FunctionWrap(getfunc=gz_ac_wrap, basedev=gz.fetch,
    ↪  multi=('GZ_ACorrs',))
    gz_spec = instruments.FunctionWrap(getfunc=gz_spec_wrap, basedev=gz.fetch,
    ↪  multi=('GZ_Spectrum',))
    gz_spec._format['xaxis']=True
    gz_cohpart = instruments.FunctionWrap(getfunc=gz_cohpart_wrap, basedev=gz.fetch,
    ↪  multi=('GZ_Coherent_Part',))
    gz_cohspec = instruments.FunctionWrap(getfunc=gz_cohspec_wrap, basedev=gz.fetch,
    ↪  multi=('GZ_Coherent_Spectrum',))
    gz_cohspec._format['xaxis']=True
    gz_cohcw = instruments.FunctionWrap(getfunc=gz_cohcw_wrap, basedev=gz.fetch,
    ↪  multi=('GZ_Coherent_CW',))
    gz_cohcw_r = instruments.FunctionWrap(getfunc=gz_r_wrap, basedev=gz.fetch,
    ↪  multi=('GZ_Coherent_CW_R',))
    gz_cohcw_theta = instruments.FunctionWrap(getfunc=gz_theta_wrap, basedev=gz.fetch,
    ↪  multi=('GZ_Coherent_CW_Theta',))

    def _gz_cummul_getformatfunc(**kwargs):
        hist_bits = kwargs.pop('hist_bits',6)
        fmt = gz_cumul._format
        fmt.update(multi=['C{k:d}'.format(k=i) for i in range(1,hist_bits+1)])
        return super(instruments.FunctionWrap, gz_cumul).getformat(**kwargs)

    gz_cumul._getformatfunc = _gz_cummul_getformatfunc

    def _gz_moment_getformatfunc(**kwargs):
        hist_bits = kwargs.pop('hist_bits',6)
        fmt = gz_moment._format
        fmt.update(multi=['M{k:d}'.format(k=i) for i in range(1,hist_bits+1)])
        return super(instruments.FunctionWrap, gz_moment).getformat(**kwargs)

    gz_moment._getformatfunc = _gz_moment_getformatfunc

    delay1 = instruments.colby_pdl_100a('GPIB0::2::INSTR')
    set(delay1.mode, delay_mode)
    set(delay1,312.5 if get(delay1.mode)=="625ps" else 156.25)
    def gz_phi_wrap(phi=-90, f0=fac, delta=delta_phi, k=phi_acorrs,
    ↪  mode=delay_mode,verbose=True, max_retry=10, *args, **kwargs):
        if mode == "625ps":
            epsilon = 1
            dt = .5
            tmax = 625  # Much more?
        else:
            epsilon = .5
            dt = .25
            tmax = 312.5
        theta = get(gz_cohcw_theta, k=k, f0=f0, *args, **kwargs)*180./pi
        if phi == None:
            return theta
        #d = theta-phi
        # Trying to keep deltas as small as possible considering mod 360
        d = (theta-phi)+(arange(-2,3)*360)
        d = d[argmin(abs(d))]
        #print theta, ' ', d
        for i in range(max_retry):
            if abs(d) > delta:
                d0 = get_or_getcache(delay1, use_cache=False)
                target = (d0+(d/360.)*(epsilon*1e12/f0))
                #target = target-target%.25
                target = dt*round(target/dt)
                #print "theta = ", theta, " | d = " , d, " | d0 = ", d0, " | target = ",
                ↪  target
                if target < 0:
                    target += (epsilon*1e12/f0)
                if target > tmax:
                    target -= (epsilon*1e12/f0)
                set(delay1, target)
                snap([delay1, yo10, rs1_power_extended], 'delay1_change.log')
                kwargs.pop('use_cache', False)
                #print d0, ' ' , target

                theta = get(gz_cohcw_theta, k=k, f0=f0, *args, **kwargs)*180./pi

                if verbose:
                    print "\nCorrected Phase on {} | delta = {} deg | retry
                    ↪  {}".format(time.strftime("%m-%d %H:%M:%S"), d, i+1)

                #d = theta-phi
                d = (theta-phi)+(arange(-2,3)*360)
                d = d[argmin(abs(d))]

        #if abs(theta-phi) > delta:
        #if min(abs((theta-phi)+(arange(-2,3)*360))) > delta: # In any cycle nearby
        if abs(d) > delta: # In any cycle nearby
            print "!WARNING!: PHASE ADJUSTMENT FAILED", ' | ', 'Theta = ', theta
        return theta #get_or_getcache(gz, True, *args, **kwargs)

    gz_phi = instruments.FunctionWrap(getfunc=gz_phi_wrap, basedev=gz.fetch,
    ↪  multi=('GZ_PHASE',))

    # To swap data with previous data
    get(gz) # Populates gz.fetch._cache)
    gz.fetch._prev = zeros(gz.fetch._cache.shape, dtype=int16 if bits_16 else uint8)
    gz.fetch._current = gz._gsa_data
    def swap_cache():
        gz.fetch._prev,gz.fetch._current = gz.fetch._current,gz.fetch._prev
```

```
    gz._gsa_data_res_arr[0].common.data.arr =
    ↪ gz.fetch._current.ctypes.data_as(ctypes.POINTER(ctypes.c_ubyte))
    gz._gsa_data = gz.fetch._current


######################
## Useful Functions ##
######################

def a_to_dict(a):
    ks = 'bfk bk block_processed chunk_processed chunk_size gfk gk k l mf n nfk res res0
    ↪ rfk'.split(' ')
    return {k:getattr(a,k) for k in ks}

def save_pyHegel(*args, **kwargs):
    l = list(args)
    l[0] = os.path.join(get(sweep.path),l[0])
    args = tuple(l)
    return np.save(*args, **kwargs)

def savez_pyHegel(*args, **kwargs):
    l = list(args)
    l[0] = os.path.join(get(sweep.path),l[0])
    args = tuple(l)
    return np.savez(*args, **kwargs)

def savez_compressed_pyHegel(*args, **kwargs):
    l = list(args)
    l[0] = os.path.join(get(sweep.path),l[0])
    args = tuple(l)
    return np.savez_compressed(*args, **kwargs)

######################
## Data structures ##
######################

savez_compressed_pyHegel( 'dc_biases{}.npz'.format('_test' if test else ''), vdc)
savez_compressed_pyHegel('ac_biases{}.npz'.format('_test' if test else ''), vac)
savez_compressed_pyHegel('phis{}.npz'.format('_test' if test else ''), phis)
savez_compressed_pyHegel('gz_config{}.npz'.format('_test' if test else ''), gz.config())

acorr_shape = (len(phis), len(vac), len(vdc))
acorr0_res_shape = (len(phis), len(vac), len(vdc), num_acorrs)
acorr_res_shape = (len(phis), len(vac), len(vdc), phi_acorrs, num_acorrs)
acorr = r_[ [ACorrUpTo(num_acorrs, 'int16', phi=phi_acorrs, fft=fft_acorrs,
↪ fftchunk=fftchunk) for i in range(prod(acorr_shape))] ]
acorr.shape = acorr_shape
acorr_remdet = r_[ [ACorrUpTo(num_acorrs, 'int16', phi=phi_acorrs, fft=fft_acorrs,
↪ fftchunk=fftchunk) for i in range(prod(acorr_shape))] ]
acorr_remdet.shape = acorr_shape
#acorr0_res = zeros(acorr0_res_shape, dtype=double)
#acorr_res = zeros(acorr_res_shape, dtype=double)

#moments_res_shape = (n_measures, len(vac), len(vdc), num_moments)
#moments_res = zeros(moments_res_shape, dtype=double)

dmm_res_shape = (n_measures, len(phis), len(vac), len(vdc), n_repeat_each)
dmm_res = zeros(dmm_res_shape, double)
phis_res = zeros(dmm_res_shape, dtype=double) # Interesting to check the variance of
↪ adjusted phi over experiement

detpart_res_shape = (n_measures, len(phis), len(vac), len(vdc), n_repeat_each,
↪ phi_acorrs)
detpart_res = zeros(detpart_res_shape, dtype=int16 if bits_16 else uint8) # Coherent
↪ Spectrum
## Delays obtained via
def get_initial_delays(phi,ac):
    set(rs1_power_extended,ac)
    if get(rs1_power_extended)<-121:
        phi = None
    t = get(gz_phi,phi=phi)
    print phi,t
    return get(delay1)


# Initial delays measured before, roughly linear
#last_delay = array([[181.75, 166.  , 150.5 , 135.  , 119.5 , 104.  ,  88.25,  72.75],[
↪ 55.  ,  39.25,  23.75,   8.  , 117.  , 101.5 ,  86.  ,  70.5 ],[ 70.5 ,  70.5 ,  70.5
↪ ,  70.5 ,  70.5 ,  70.5 ,  70.5 ,  70.5 ]]) # Initial value, then it'll be the last
↪ valid one.
last_delay = r_[[[get_initial_delays(i,j) for i in phis] for j in vac]]

######################
## Helper functions ##
######################
# n,p,a,d,r
# n=n_measures, p=phi, a=vac, d=vdc, r=repeats

def null():
    pass

# For manualy async of gz and dmm12
def dmm_helper(n,p,a,d,r):
    dmm_res[n,p,a,d,r] = get(dmm12)

def get_data(n,p,a,d,r):
    current_phi = phis[p]
    # For Vac=0
    if get(rs1_power_extended)<-121:
        current_phi = None
    thread_dmm = Thread(target=dmm_helper, args=(n,p,a,d,r))
    thread_dmm.start()
    # Next line will adjust phase before returning data
    phis_res[n,p,a,d,r] = get(gz_phi, phi=current_phi, f0=fac, k=phi_acorrs,
    ↪ delta=delta_phi, max_retry=5000, verbose=False if not test else True)
    thread_dmm.join()

    last_delay[a,p] = get_or_getcache(delay1, use_cache=True) # Caching for next loop

def process_previous_data(n,p,a,d,r):
    #initime = time.time()
    set_num_threads(65) # 72 is faster, but we leave threads for the acquisition
    # Extracting deterministic part and removing it from cached data used afterwards
    data = get_or_getcache(gz, use_cache='prev')
    detpart = getdet(data, phi_acorrs)
    detpart_res[n,p,a,d,r] = detpart
    # Computing aCorrs
    acorr[p,a,d](data)
    # Removing deterministic part
    deldet(data, detpart)
    acorr_remdet[p,a,d](data)
    # Computing histogram and saving its moments
    #moments_res[n,a,d] = get(gz_moment, use_cache='prev', k=num_moments,
    ↪ hist_bits=hist_bits)
    #fintime = time.time()
    #if test:
    #    print "Computation time: {:.8f}".format(fintime-initime)


def alist_to_dict(alist):
    ks = 'bfk bk block_processed chunk_processed chunk_size gfk gk k l mf n nfk res res0
    ↪ rfk'.split(' ')
    tmp = dict()
    for key in ks:
        if key in 'bfk gfk nfk res rfk'.split(' '):
            newshape = acorr_res_shape
        elif key in 'bk gk res0'.split(' '):
            newshape = acorr0_res_shape
        elif key in 'mf'.split(' '):
            newshape = acorr_shape + (phi_acorrs,)
        else:
            newshape = acorr_shape
        tmp[key] = r_[map(lambda x: getattr(x, key),
        ↪ alist.reshape(prod(alist.shape)))].reshape(newshape)

    return tmp


def save_data(baktime=0):
    # Saving results in case there's an issue, overwriting at each iteration
    acorr_dict = alist_to_dict(acorr)
    acorr_remdet_dict = alist_to_dict(acorr_remdet)
    savez_pyHegel('acorrs{}.npz'.format('_test' if test else ''), **acorr_dict)
    savez_pyHegel('acorrs_remdet{}.npz'.format('_test' if test else ''),
    ↪ **acorr_remdet_dict)
```

```
    savez_pyHegel('dmm_detpart_phis{}.npz'.format('_test' if test else ''),
    ↪  dmm_res=dmm_res, detpart_res=detpart_res, phis_res=phis_res)

    # Keeping a backup version every 6 hours in case there's a real bad issue
    currtime = time.time()
    if currtime - baktime > 6*3600:
        baktime = time.time()
        tstamp = time.strftime('%Y%m%d_%H%M%S')

        savez_pyHegel('acorrs{}_bak_{}.npz'.format('_test' if test else '', tstamp),
        ↪  **acorr_dict)
        savez_pyHegel('acorrs_remdet{}_bak_{}.npz'.format('_test' if test else '',
        ↪  tstamp), **acorr_remdet_dict)
        savez_pyHegel('dmm_detpart_phis{}_bak_{}.npz'.format('_test' if test else '',
        ↪  tstamp), dmm_res=dmm_res, detpart_res=detpart_res, phis_res=phis_res)

    return baktime

###################
## MEASUREMENTS! ##
###################

# Ensure it's ok
set(yo10.range,max(abs(vdc)))
set(yo10,0) # Just in case
set(rs1_power_extended,-122)

##############
## GO GO GO ##
##############


# The Measurement itself
n_width = "{:d}".format(int(log10(100))+1)
baktime = 0

# Initial setup
thread_processing = Thread(target=null) # First iteration there's nothing to process
thread_processing.start()
baktime = time.time() if test else 0 # On real run check if backup works at first
# n=n_measures, a=vac, d=vdc
for current_iteration, (n,p,a,d,r) in enumerate(product(range(n_measures),
↪  range(len(phis)), range(len(vac)), range(len(vdc)), range(n_repeat_each))): # was nlm
    it = time.time()
    # 1 - Let's set AC power and delay if it needs to change!
    if a+d+r==0:
        print ("\n\nMeasuring iteration {:0"+n_width+"d}/{:0"+n_width+"d} | Phase =
        ↪  {:d}").format(n+1, n_measures,p)
    if d+r==0:
        set(rs1_power_extended,vac[a])
        if not get(rs1_power_extended) < -121:
            if not get(delay1)==last_delay[a,p]:
                set(delay1, last_delay[a,p])
        #print(("  vac: {:02.2f} dBm").format(get(rs1_power_extended)))
        print(" ")

    if r==0: # if r==0, then we should set voltage and wait
        set(yo10,vdc[d])
        pause(vdc_waittime)  # Waiting until voltage is stable

    # 2 - Data Acquisition!
    iat = time.time()
    get_data(n,p,a,d,r) # Captures the GIL
    fat = time.time()
    # 3 - Ensuring data processing is done before swapping cache
    thread_processing.join()
    swap_cache()
    fpt = time.time()
    # 4 - Saving and Starting data processing
    # Pac loop is done processing after thread_processing on the next iteration is done.
    if current_iteration and a+d+r==0:    # Exception for first iteration
        baktime = save_data(baktime)
    thread_processing = Thread(target=process_previous_data, args=(n,p,a,d,r))
    thread_processing.start()
    ft = time.time()
    if r==0:
        print ""
    print ("{:0"+n_width+"d}/{:0"+n_width+"d}").format(n+1, n_measures) + u" phi: {: 4d}°
    ↪  ({: 4.2f}°) | vac: {:04.0F} dBm | vdc: {:05.2f} V | Acq: {:05.2F} sec | SET:
    ↪  {:03.2F} sec | CALC: {:.1F} | TOT: {:04.3F}
    ↪  sec".format(phis[p],phis_res[n,p,a,d,r],get(rs1_power_extended),get(yo10),
    ↪  fat-iat, iat-it, fpt-fat, ft-it)
# 5 - Final setup
thread_processing.join()
swap_cache()
baktime = save_data(baktime)
# 6 - Saving final results compressed
tstamp = time.strftime('%Y%m%d_%H%M%S') # Put there afterwards, script crashed here
↪  initially because tstamp wasn't defined. Re-executed afterwards.
savez_compressed_pyHegel('acorrs{}_final.npz'.format('_test' if test else '', tstamp),
↪  **alist_to_dict(acorr))
savez_compressed_pyHegel('acorrs_remdet{}_final.npz'.format('_test' if test else '',
↪  tstamp), **alist_to_dict(acorr_remdet))
savez_compressed_pyHegel('dmm_detpart_phis{}_final.npz'.format('_test' if test else '',
↪  tstamp), dmm_res=dmm_res, detpart_res=detpart_res, phis_res=phis_res)

# POST EXPERIMENT
set(yo10,0)
set(rs1_power_extended, -122)
set_num_threads(72) # So the system stays responsive in interactive mode
```

C.2.3 Mesure photoexcitée résolue en phase sans polarisation continue

On fait une mesure synchrone avec la source servant à la photoexcitation sans V_{ac} et pour plusieurs valeurs $V_{ac} \neq 0$ couvrant une grande plage. On ne mesures que les grandes valeurs de V_{dc} servant à la calibration ainsi que $V_{dc} = 0$. On mesures les autocorrélations résolues en phase avant et après l'application de `remdet` sur les données brutes.

`Sphi_Vdc_Vac_PhaseLocked_Avg_Phi_Ref.py`

```
#!/bin/env/python
#! -*- coding: utf-8 -*-
import sys, os, ctypes
sys.path.append('C:/Projets/TimeDomain_Guzik/Code/Histograms-OTF/')
sys.path.append('C:/Projets/TimeDomain_Guzik/Code/aCorrs-OTF/bin/')
sys.path.append('C:/Projets/TimeDomain_Guzik/Code/remdet-OTF/bin/')
from acorrs_otf import ACorrUpTo
from histograms_otf import hist1dNbits, hist2dNbits, cumulant, moment
from remdet import getdet, deldet, remdet, set_num_threads
from threading import Thread
from itertools import product


# On veut faire un sweep en Vac, et aussi pouvoir calibrer à Vdc=0 via grand Vdc.

test = False

make_dir("C://Projets/TimeDomain_Guzik/Data/%D/Guzik_Sphi_Vdc_Vac_PhaseLocked/" + ("test"
↪  if test else ""))

```

```

# Last run: 92 hours for 3*19 vac&vdc combinations @ 10 repeast and 33x measurements
# Aiming for 95 hours from now (friday morning before WeekEnd:  (95./92)*33*(3*19)/(10*7) = 27.7 sweeps
# To save some time, we repeat 20 times and do 13 loops

########################
## DEVICES VARIABLES ##
########################
n_measures = 13 if not test else 1
n_repeat_each = 20 if not test else 2   # Speeds things up for timing test if need be
# gz
gz_samples = 1*2**30    # 52.5 GiB is the max value
gz_gain = 7
gz_equalizer = 1
gz_ref = 'ext'
bits_16 = True
# acorrs
num_acorrs = 65 # = 64+1 chosen not to slow down measurements too much
fft_acorrs = False
phi_acorrs = 8 # 32/4 = 8
fftchunk = 16*num_acorrs # Sweet-spot empirique pour num_acorrs=64, 16 est mieux pour grands délais
# moments
num_moments = 6
hist_bits = 16
# phases
phis = arange(-4,4, 1 if not test else 4)*45   # -180 à 135 par bonds de 45°, choisi pour fitter dans les délais accesibles
delta_phi = 0.5
fac = 4e9
delay_mode = '312.5ps' #'625ps'


#####################
## MANIP VARIABLES ##
#####################

# BIASES
def PdBm2Vac(PdBm):
    return 10**(PdBm/20.)*1./sqrt(10)

def Vac2PdBm(Vac):
    return 10*log10((Vac/sqrt(2))**2/50*1e3)

def Vac_linspace_dBm(Pmin, Pmax, num):
    return Vac2PdBm(linspace(PdBm2Vac(Pmin), PdBm2Vac(Pmax), num))

## Sweeps Vdc @ Vac=cte
vdc = r_[linspace(-3,-2,3),0,linspace(2,3,3)]
# Previous values
vac = r_[-122, Vac2PdBm(linspace(PdBm2Vac(-20),PdBm2Vac(1),9))]

vdc_waittime = 1.5 # Seconds # Increased to be safe

########################
## SETTING UP DEVICES ##
########################

# Loading devices
try:
    if not isinstance(gz, instruments.guzik.guzik_adp7104):
        gz = instruments.guzik_adp7104()
except:
    gz = instruments.guzik_adp7104()
gz.config([1], n_S_ch=gz_samples, gain_dB=gz_gain, bits_16=bits_16, equalizer_en=gz_equalizer, ext_ref=gz_ref)
load("yo10")
set(yo10,0)
set(yo10.output_en, True)
set(yo10.range,10)
load("dmm12")
set(dmm12.aperture, .5)
set(dmm12.zero, 1)
load("att1")
set(att1, 101)
rs1 = instruments.rs_sgma('TCPIP::rssgs100a110885.mshome.net::inst0::INSTR')
set(rs1.output_en,False)
set(rs1.power_level_dbm, -20)
set(rs1.freq,fac)
set(rs1.ref_output_signal, 'ref')

def rs1_power_extended_set_wrap(p):
    if p < -121:
        set(rs1.output_en,0)
        return
    elif not get(rs1.output_en):
        set(rs1.output_en,1)
    if p < -20:
        set(att1, 101)
        set(rs1.power_level_dbm, -20+p%1)
        set(att1, -20-p + ceil(p%1))
    else:
        set(rs1.power_level_dbm, p)
        set(att1, 0)


def rs1_power_extended_get_wrap():
    if not get(rs1.output_en):
        return -inf
    tmp1 = get(rs1.power_level_dbm)
    tmp2 = get(att1)
    return tmp1-tmp2


rs1_power_extended = instruments.FunctionWrap(setfunc=rs1_power_extended_set_wrap,
                                              getfunc=rs1_power_extended_get_wrap,
                                              basedevs=[rs1,att1],
                                              multi=["rs1.output_level_dbm - att1"]
                                             )

# Setting output power to minimal value
set(rs1_power_extended, -122)


## Virtual GZ Devices
def get_or_getcache(dev, use_cache, *args, **kwargs):
    if use_cache == 'prev':
        return dev.alias._prev
    elif use_cache:
        try:
            return dev.getcache(*args, **kwargs)
        except:
            return dev.alias.getcache(*args, **kwargs)
    else:
        return get(dev, *args, **kwargs)

def gz_snip_wrap(snipsize=1000, use_cache=False, *args, **kwargs):
    v = get_or_getcache(gz, use_cache)
    snip = v[:snipsize].copy()
    return snip

def gz_h_wrap(hist_bits=16, use_cache=False, *args, **kwargs):
    v = get_or_getcache(gz, use_cache);
    h = hist1dNbits(v, hist_bits, *args, **kwargs)
    return h

def gz_moment_wrap(k=6, h=None, *args, **kwargs):
    if h is None:
        h = gz_h_wrap(*args, **kwargs)
    return r_[[moment(h,n) for n in range(1, k+1)]]

def gz_cumul_wrap(k=6, h=None, *args, **kwargs):
    if h is None:
        h = gz_h_wrap(*args, **kwargs)
    return r_[[cumulant(h,n) for n in range(1, k+1)]]

def gz_ac_wrap(k=64, use_cache=False, *args, **kwargs):
    v = get_or_getcache(gz, use_cache);
    a = ACorrUpTo(k, v, *args, **kwargs)
    return a.res

def spectrum_from_acorrs(a, rate=32e9, norm=True, *args, **kwargs):
    w = fftshift(fft.hfft(a))
    f = fftshift(fft.fftfreq(len(w), 1./rate))
    return array([f,w if not norm else abs(w)])
```

```
def gz_spec_wrap(k=128, use_cache=False, *args, **kwargs):
    v = get_or_getcache(gz, use_cache);
    a = ACorrUpTo(k, v)
    w = spectrum_from_acorrs(a.res, *args, **kwargs)
    return w

def coherent_spectrum(v, k, repeat_k=8, sr=32e9, polar=False, norm=False, nozero=False, *args, **kwargs):
    #if length is None:
    #    length = len(v)
    #if length%k:
    #    w = v[:length-(length%k)].view()
    #else:
    #    w = v[:length].view()
    #w.shape = (w.shape[0]/k,k)
    #z = w.mean(axis=0)
    w = getdet(v,k)
    z = zeros((repeat_k,len(w)),dtype=w.dtype)
    z[:] = w
    z.shape = len(w)*repeat_k,
    x = fftshift(fft.fft(z))
    x = x if not norm else abs(x)
    f = fftshift(fftfreq(len(x),1./sr))
    if nozero:
        x[where(f==0)]=None
    if polar:
        return x
    else:
        return array([f,x])


def gz_cohpart_wrap(k=8, repeat_k=1, remove=False, use_cache=False, *args, **kwargs):
    fct = remdet if remove else getdet
    v = get_or_getcache(gz, use_cache);
    w = fct(v,k)
    z = zeros((repeat_k,len(w)),dtype=w.dtype)
    z[:] = w
    z.shape = len(w)*repeat_k,
    return z

def gz_cohspec_wrap(k=8, repeat_k=8, use_cache=False, *args, **kwargs):
    v = get_or_getcache(gz, use_cache);
    w = coherent_spectrum(v, k, repeat_k=repeat_k, *args, **kwargs)
    return w

def coherent_cw(v, k, f0, *args, **kwargs):
    kwargs.pop("polar",0)
    ff,cc = coherent_spectrum(v,k,polar=False,*args,**kwargs)
    idx = argmin(abs(ff-f0))
    f,c = ff[idx].real, cc[idx]
    if not isclose(f,f0):
        print("{:.6f} MHz selected instead of {:.6f} MHz".format(f/1.e6,f0/1.e6))
    return c

def gz_cohcw_wrap(f0=4e9, k=128, out_fmt="complex", use_cache=False, *args, **kwargs):
    v = get_or_getcache(gz, use_cache);
    w = coherent_cw(v, 2*k, f0, *args, **kwargs)
    if out_fmt.lower()=="complex":
        return w
    elif out_fmt.lower() in ["realimag","ri"]:
        return r_[w.real,w.imag]
    elif out_fmt.lower() in ["rtheta", "rt"]:
        return r_[abs(w),angle(w)]
    else:
        raise KeyError("Invalid output format. Valid values are 'complex', 'realimag', and 'rtheta'.")

def gz_r_wrap(*args, **kwargs):
    kwargs.pop('out_fmt', 0)
    r,t = get(gz_cohcw, out_fmt='rt', *args, **kwargs)
    return r

def gz_theta_wrap(*args, **kwargs):
    kwargs.pop('out_fmt', 0)
    r,t = get(gz_cohcw, out_fmt='rt', *args, **kwargs)
    return t


## The actual virtual instruments
gz_snip = instruments.FunctionWrap(getfunc=gz_snip_wrap, basedev=gz.fetch, multi=('GZ_Snippet',))
gz_h = instruments.FunctionWrap(getfunc=gz_h_wrap, basedev=gz.fetch, multi=('GZ_Histogram',))
gz_moment = instruments.FunctionWrap(getfunc=gz_moment_wrap, basedev=gz.fetch, multi=list('GZ_Moments',))
gz_cumul = instruments.FunctionWrap(getfunc=gz_cumul_wrap, basedev=gz.fetch, multi=list('GZ_Cumulants',))
gz_ac = instruments.FunctionWrap(getfunc=gz_ac_wrap, basedev=gz.fetch, multi=('GZ_ACorrs',))
gz_spec = instruments.FunctionWrap(getfunc=gz_spec_wrap, basedev=gz.fetch, multi=('GZ_Spectrum',))
gz_spec._format['xaxis']=True
gz_cohpart = instruments.FunctionWrap(getfunc=gz_cohpart_wrap, basedev=gz.fetch, multi=('GZ_Coherent_Part',))
gz_cohspec = instruments.FunctionWrap(getfunc=gz_cohspec_wrap, basedev=gz.fetch, multi=('GZ_Coherent_Spectrum',))
gz_cohspec._format['xaxis']=True
gz_cohcw = instruments.FunctionWrap(getfunc=gz_cohcw_wrap, basedev=gz.fetch, multi=('GZ_Coherent_CW',))
gz_cohcw_r = instruments.FunctionWrap(getfunc=gz_r_wrap, basedev=gz.fetch, multi=('GZ_Coherent_CW_R',))
gz_cohcw_theta = instruments.FunctionWrap(getfunc=gz_theta_wrap, basedev=gz.fetch, multi=('GZ_Coherent_CW_Theta',))

def _gz_cummul_getformatfunc(**kwargs):
    hist_bits = kwargs.pop('hist_bits',6)
    fmt = gz_cumul._format
    fmt.update(multi=['C{k:d}'.format(k=i) for i in range(1,hist_bits+1)])
    return super(instruments.FunctionWrap, gz_cumul).getformat(**kwargs)

gz_cumul._getformatfunc = _gz_cummul_getformatfunc

def _gz_moment_getformatfunc(**kwargs):
    hist_bits = kwargs.pop('hist_bits',6)
    fmt = gz_moment._format
    fmt.update(multi=['M{k:d}'.format(k=i) for i in range(1,hist_bits+1)])
    return super(instruments.FunctionWrap, gz_moment).getformat(**kwargs)

gz_moment._getformatfunc = _gz_moment_getformatfunc

delay1 = instruments.colby_pdl_100a('GPIB0::2::INSTR')
set(delay1.mode, delay_mode)
set(delay1,312.5 if get(delay1.mode)=="625ps" else 156.25)
def gz_phi_wrap(phi=-90, f0=fac, delta=delta_phi, k=phi_acorrs, mode=delay_mode,verbose=True, max_retry=10, *args, **kwargs):
    if mode == "625ps":
        epsilon = 1
        dt = .5
        tmax = 625  # Much more?
    else:
        epsilon = .5
        dt = .25
        tmax = 312.5
    theta = get(gz_cohcw_theta, k=k, f0=f0, *args, **kwargs)*180./pi
    if phi == None:
        return theta
    #d = theta-phi
    # Trying to keep deltas as small as possible considering mod 360
    d = (theta-phi)+(arange(-2,3)*360)
    d = d[argmin(abs(d))]
    #print theta, '', d
    for i in range(max_retry):
        if abs(d) > delta:
            d0 = get_or_getcache(delay1, use_cache=False)
            target = (d0+(d/360.)*(epsilon*1e12/f0))
            #target = target-target%.25
            target = dt*round(target/dt)
            #print "theta = ", theta, " | d = " , d, " | d0 = ", d0, " | target = ", target
            if target < 0:
                target += (epsilon*1e12/f0)
            if target > tmax:
                target -= (epsilon*1e12/f0)
            set(delay1, target)
            snap([delay1, yo10, rs1_power_extended], 'delay1_change.log')
```

```
            kwargs.pop('use_cache', False)
            #print d0, ' ' , target

            theta = get(gz_cohcw_theta, k=k, f0=f0, *args, **kwargs)*180./pi

            if verbose:
                print "\nCorrected Phase on {} | delta = {} deg | retry
                ↪ {}".format(time.strftime("%m-%d %H:%M:%S"), d, i+1)

            #d = theta-phi
            d = (theta-phi)+(arange(-2,3)*360)
            d = d[argmin(abs(d))]

    #if abs(theta-phi) > delta:
    #if min(abs((theta-phi)+(arange(-2,3)*360))) > delta: # In any cycle nearby
    if abs(d) > delta: # In any cycle nearby
        print "!WARNING!: PHASE ADJUSTMENT FAILED", ' | ', 'Theta = ', theta
    return theta #get_or_getcache(gz, True, *args, **kwargs)

gz_phi = instruments.FunctionWrap(getfunc=gz_phi_wrap, basedev=gz.fetch,
↪ multi=('GZ_PHASE',))

# To swap data with previous data
get(gz) # Populates gz.fetch._cache)
gz.fetch._prev = zeros(gz.fetch._cache.shape, dtype=int16 if bits_16 else uint8)
gz.fetch._current = gz._gsa_data
def swap_cache():
    gz.fetch._prev,gz.fetch._current = gz.fetch._current,gz.fetch._prev
    gz._gsa_data_res_arr[0].common.data.arr =
    ↪ gz.fetch._current.ctypes.data_as(ctypes.POINTER(ctypes.c_ubyte))
    gz._gsa_data = gz.fetch._current


######################
## Useful Functions ##
######################

def a_to_dict(a):
    ks = 'bfk bk block_processed chunk_processed chunk_size gfk gk k l mf n nfk res res0
    ↪ rfk'.split(' ')
    return {k:getattr(a,k) for k in ks}

def save_pyHegel(*args, **kwargs):
    l = list(args)
    l[0] = os.path.join(get(sweep.path),l[0])
    args = tuple(l)
    return np.save(*args, **kwargs)

def savez_pyHegel(*args, **kwargs):
    l = list(args)
    l[0] = os.path.join(get(sweep.path),l[0])
    args = tuple(l)
    return np.savez(*args, **kwargs)

def savez_compressed_pyHegel(*args, **kwargs):
    l = list(args)
    l[0] = os.path.join(get(sweep.path),l[0])
    args = tuple(l)
    return np.savez_compressed(*args, **kwargs)

####################
## Data structures ##
####################

savez_compressed_pyHegel( 'dc_biases{}.npz'.format('_test' if test else ''), vdc)
savez_compressed_pyHegel('ac_biases{}.npz'.format('_test' if test else ''), vac)
savez_compressed_pyHegel('phis{}.npz'.format('_test' if test else ''), phis)
savez_compressed_pyHegel('gz_config{}.npz'.format('_test' if test else ''), gz.config())

acorr_shape = (len(phis), len(vac), len(vdc))
acorr0_res_shape = (len(phis), len(vac), len(vdc), num_acorrs)
acorr_res_shape = (len(phis), len(vac), len(vdc), phi_acorrs, num_acorrs)
acorr = r_[ [ACorrUpTo(num_acorrs, 'int16', phi=phi_acorrs, fft=fft_acorrs,
↪ fftchunk=fftchunk) for i in range(prod(acorr_shape))] ]
acorr.shape = acorr_shape
acorr_remdet = r_[ [ACorrUpTo(num_acorrs, 'int16', phi=phi_acorrs, fft=fft_acorrs,
↪ fftchunk=fftchunk) for i in range(prod(acorr_shape))] ]
acorr_remdet.shape = acorr_shape

#acorr0_res = zeros(acorr0_res_shape, dtype=double)
#acorr_res = zeros(acorr_res_shape, dtype=double)

#moments_res_shape = (n_measures, len(vac), len(vdc), num_moments)
#moments_res = zeros(moments_res_shape, dtype=double)

dmm_res_shape = (n_measures, len(phis), len(vac), len(vdc), n_repeat_each)
dmm_res = zeros(dmm_res_shape, double)
phis_res = zeros(dmm_res_shape, dtype=double) # Interesting to check the variance of
↪ adjusted phi over experiement

detpart_res_shape = (n_measures, len(phis), len(vac), len(vdc), n_repeat_each,
↪ phi_acorrs)
detpart_res = zeros(detpart_res_shape, dtype=int16 if bits_16 else uint8) # Coherent
↪ Spectrum
## Delays obtained via
def get_initial_delays(phi,ac):
    set(rs1_power_extended,ac)
    if get(rs1_power_extended)<-121:
        phi = None
    t = get(gz_phi,phi=phi)
    print phi,t,'\t\t',ac
    return get(delay1)


# Initial delays measured before, roughly linear
#last_delay = array([[181.75, 166.  , 150.5 , 135.  , 119.5 , 104.  ,  88.25,  72.75],[
↪ 55.  ,  39.25,  23.75,   8.  , 117.  , 101.5 ,  86.  ,  70.5 ],[ 70.5 ,  70.5 ,  70.5
↪ ,  70.5 ,  70.5 ,  70.5 ,  70.5 ,  70.5 ]]) # Initial value, then it'll be the last
↪ valid one.
last_delay = r_[[[get_initial_delays(i,j) for i in phis] for j in vac]]

######################
## Helper functions ##
######################
# n,p,a,d,r
# n=n_measures, p=phi, a=vac, d=vdc, r=repeats

def null():
    pass

# For manualy async of gz and dmm12
def dmm_helper(n,p,a,d,r):
    dmm_res[n,p,a,d,r] = get(dmm12)

def get_data(n,p,a,d,r):
    current_phi = phis[p]
    # For Vac=0
    if get(rs1_power_extended)<-121:
        current_phi = None
    thread_dmm = Thread(target=dmm_helper, args=(n,p,a,d,r))
    thread_dmm.start()
    # Next line will adjust phase before returning data
    phis_res[n,p,a,d,r] = get(gz_phi, phi=current_phi, f0=fac, k=phi_acorrs,
    ↪ delta=delta_phi, max_retry=5000, verbose=False if not test else True)
    thread_dmm.join()

    last_delay[a,p] = get_or_getcache(delay1, use_cache=True) # Caching for next loop

def process_previous_data(n,p,a,d,r):
    #initime = time.time()
    set_num_threads(65) # 72 is faster, but we leave threads for the acquisition
    # Extracting deterministic part and removing it from cached data used afterwards
    data = get_or_getcache(gz, use_cache='prev')
    detpart = getdet(data, phi_acorrs)
    detpart_res[n,p,a,d,r] = detpart
    # Computing aCorrs
    acorr[p,a,d](data)
    # Removing deterministic part
    deldet(data, detpart)
    acorr_remdet[p,a,d](data)
    # Computing histogram and saving its moments
    #moments_res[n,a,d] = get(gz_moment, use_cache='prev', k=num_moments,
    ↪ hist_bits=hist_bits)
    #fintime = time.time()
    #if test:
    #    print "Computation time: {:.8f}".format(fintime-initime)
```

```
def alist_to_dict(alist):
    ks = 'bfk bk block_processed chunk_processed chunk_size gfk gk k l mf n nfk res res0 rfk'.split(' ')
    tmp = dict()
    for key in ks:
        if key in 'bfk gfk nfk res rfk'.split(' '):
            newshape = acorr_res_shape
        elif key in 'bk gk res0'.split(' '):
            newshape = acorr0_res_shape
        elif key in 'mf'.split(' '):
            newshape = acorr_shape + (phi_acorrs,)
        else:
            newshape = acorr_shape
        tmp[key] = r_[map(lambda x: getattr(x, key), alist.reshape(prod(alist.shape)))].reshape(newshape)

    return tmp


def save_data(baktime=0):
    # Saving results in case there's an issue, overwriting at each iteration
    acorr_dict = alist_to_dict(acorr)
    acorr_remdet_dict = alist_to_dict(acorr_remdet)
    savez_pyHegel('acorrs{}.npz'.format('_test' if test else ''), **acorr_dict)
    savez_pyHegel('acorrs_remdet{}.npz'.format('_test' if test else ''), **acorr_remdet_dict)
    savez_pyHegel('dmm_detpart_phis{}.npz'.format('_test' if test else ''), dmm_res=dmm_res, detpart_res=detpart_res, phis_res=phis_res)

    # Keeping a backup version every 6 hours in case there's a real bad issue
    currtime = time.time()
    if currtime - baktime > 6*3600:
        baktime = time.time()
        tstamp = time.strftime('%Y%m%d_%H%M%S')

        savez_pyHegel('acorrs{}_bak_{}.npz'.format('_test' if test else '', tstamp), **acorr_dict)
        savez_pyHegel('acorrs_remdet{}_bak_{}.npz'.format('_test' if test else '', tstamp), **acorr_remdet_dict)
        savez_pyHegel('dmm_detpart_phis{}_bak_{}.npz'.format('_test' if test else '', tstamp), dmm_res=dmm_res, detpart_res=detpart_res, phis_res=phis_res)

    return baktime

###################
## MEASUREMENTS! ##
###################

# Ensure it's ok
set(yo10.range,max(abs(vdc)))
set(yo10,0) # Just in case
set(rs1_power_extended,-122)

###############
## GO GO GO ##
###############


# The Measurement itself
n_width = "{:d}".format(int(log10(100))+1)
baktime = 0

# Initial setup
thread_processing = Thread(target=null) # First iteration there's nothing to process
thread_processing.start()
baktime = time.time() if test else 0 # On real run check if backup works at first
# n=n_measures, a=vac, d=vdc
for current_iteration, (n,p,a,d,r) in enumerate(product(range(n_measures), range(len(phis)), range(len(vac)), range(len(vdc)), range(n_repeat_each))): # was nlm
    it = time.time()
    # 1 - Let's set AC power and delay if it needs to change!
    if a+d+r==0:
        print ("\n\nMeasuring iteration {:0"+n_width+"d}/{:0"+n_width+"d} | Phase = {:d}").format(n+1, n_measures,p)
    if d+r==0:
        set(rs1_power_extended,vac[a])
        if not get(rs1_power_extended) < -121:
            if not get(delay1)==last_delay[a,p]:
                set(delay1, last_delay[a,p])
        #print("  vac: {:02.2f} dBm").format(get(rs1_power_extended)))
        print(" ")

    if r==0: # if r==0, then we should set voltage and wait
        set(yo10,vdc[d])
        pause(vdc_waittime)  # Waiting until voltage is stable

    # 2 - Data Acquisition!
    iat = time.time()
    get_data(n,p,a,d,r) # Captures the GIL
    fat = time.time()
    # 3 - Ensuring data processing is done before swapping cache
    thread_processing.join()
    swap_cache()
    fpt = time.time()
    # 4 - Saving and Starting data processing
    # Pac loop is done processing after thread_processing on the next iteration is done.
    if current_iteration and a+d+r==0:    # Exception for first iteration
        baktime = save_data(baktime)
    thread_processing = Thread(target=process_previous_data, args=(n,p,a,d,r))
    thread_processing.start()
    ft = time.time()
    if r==0:
        print ""
    print ("{:0"+n_width+"d}/{:0"+n_width+"d}").format(n+1, n_measures) + u" phi: {: 4d}° ({: 4.2f}°) | vac: {:04.0F} dBm | vdc: {:05.2f} V | Acq: {:05.2F} sec | SET: {:03.2F} sec | CALC: {:.1F} | TOT: {:04.3F} sec".format(phis[p],phis_res[n,p,a,d,r],get(rs1_power_extended),get(yo10), fat-iat, iat-it, fpt-fat, ft-it)
# 5 - Final setup
thread_processing.join()
swap_cache()
baktime = save_data(baktime)
# 6 - Saving final results compressed
tstamp = time.strftime('%Y%m%d_%H%M%S') # Put there afterwards, script crashed here initially because tstamp wasn't defined. Re-executed afterwards.
savez_compressed_pyHegel('acorrs{}_final.npz'.format('_test' if test else '', tstamp), **alist_to_dict(acorr))
savez_compressed_pyHegel('acorrs_remdet{}_final.npz'.format('_test' if test else '', tstamp), **alist_to_dict(acorr_remdet))
savez_compressed_pyHegel('dmm_detpart_phis{}_final.npz'.format('_test' if test else '', tstamp), dmm_res=dmm_res, detpart_res=detpart_res, phis_res=phis_res)

# POST EXPERIMENT
set(yo10,0)
set(rs1_power_extended, -122)
set_num_threads(72) # So the system stays responsive in interactive mode
```

Annexe D

Autres Codes

D.1 Fonctions théoriques de densité spectrale et corrélation temporelle

Le code qui suit implémente des fonctions théoriques pour les densités spectrales et corrélateurs courant–courant résolus en phase ou non discuté dans cette thèse. On utilise les limites analitiques lorsque possible, pour éviter les divisions par zéro lorsque la limite existe par exemple. On profite de la vectorization pour garder le code simple, surtout en ce qui a trait aux différents régimes et limites, et flexible à l'utilisation. On utilise aussi la mémoization pour accélérer l'appel des fonctions avec un ensemble de paramètres ayant déjà été utilisés. Ces fonctions sont utilisées pour tracer les courbes théoriques sur la majorité des figures de cette thèse.

`Noise_Theory.py`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-


from pylab import *
from scipy.special import jv as besselJ
import scipy.constants as C


###############
# Description #
###############

"""
Theoretical autocovariance/noise fonctions.

- We use analitical limits when possible for special cases.
    - We should only get ±inf when it's the actual analitical result.
- We define everything in its proper representation; no FFTs are used.
```

```
    - Vectorization is used for convenience with special cases.
        - Code should still be performant enough for live plotting.
        - Memoization makes adjusting graphics much faster.

    Default units are A² for autocovariance and A²/Hz for spectral densities.
        - R can be set to 2*C.k to obtain Kelvin.
        - Or you can multiply the A²/Hz result by R/(2*C.k).
    """

    # R = 1 # Some theory paper use this
    # R = 2*C.k # Use this if you want noise expressed in Kelvin
    # Experimental resistance of my jonction
    R = 40.378788287194240069766237866133451461791992187 5 # Ohm

    ####################
    # General Functions #
    ####################

    try:
        import cPickle

        class MemoizeMutable:
            def __init__(self, fn, verbose=False):
                self.fn = fn
                self.memo = {}
                self.verbose=verbose
            def __call__(self, *args, **kwds):
                str = cPickle.dumps(args, 1)+cPickle.dumps(kwds, 1)
                if not self.memo.has_key(str):
                    self.memo[str] = self.fn(*args, **kwds)
                    if self.verbose:
                        print "MISS"
                else:
                    if self.verbose:
                        print "HIT"
                return self.memo[str]

        print "Will use memoization"
    except:
        MemoizeMutable = lambda x: x
        print "Will NOT use memoization"


    def coth(x):
        return 1./tanh(x)

    # Optimizing this function helps a lot in the end.
    def xcothx(x):
        x = r_[x]
        ret = empty(x.shape)
        mask = x==0
        xmask = x[~mask]
        ret[mask]=1
        ret[~mask]=xmask*coth(xmask)
        return ret

    # Equivalent and clearer but slower implementation of xcothx
    @vectorize
    def _xcothx(x):
        if x==0:
            return 1
        else:
            return x*coth(x)

    ###############
    # Time Domain #
    ###############

    def _tSeq(tau,Te,R=R):
        if Te==0:
            return -1./tau**2*C.hbar/(pi*R)
        else:
            return -pi*(C.k*Te)**2/(R*C.hbar)*1./(sinh(pi*C.k*Te*tau/C.hbar))**2

    def _tSdc(tau,nu,Te,R=R):
        return _tSeq(tau,Te,R)*cos(nu*tau)

    def _tDSdc(tau,nu,Te,R=R):
        return -2*_tSeq(tau,Te,R)*sin(nu*tau/2.)**2
        # Equivalent form for testing
        #return _tSdc(tau,nu,Te,R)-_tSdc(tau,0,Te,R)

    def _tSPH(tau,nu,Te,R=R):
        if tau==0:
            D = pi*(C.k*Te)**2/(3*C.hbar*R)
        else:
            D = _tSeq(tau,Te,R)-_tSeq(tau,0,R)
```

```
107         return D*cos(nu*tau)
108 
109 
110     ###################
111     # Frequency domain #
112     ###################
113 
114     def _Seq(omega,Te,R=R):
115         if Te==0:
116             return abs(C.hbar*omega)/R
117         else:
118             return 2*C.k*Te/R * xcothx(C.hbar*omega/(2*C.k*Te))
119 
120     def _Sdc(omega,nu,Te,R=R):
121         return (_Seq(nu-omega,Te,R)+_Seq(nu+omega,Te,R))/2.
122 
123     def _Spa(omega,nu,Te,nuac,Omega,R=R,nBessel=21):
124         if not nuac*Omega:
125             return _Sdc(omega,nu,Te,R)
126         z = nuac/Omega
127         nBessel -= nBessel%2-1    # Ensure it's odd
128         Ns = arange(-nBessel//2+1,nBessel//2+1)
129         freqs = omega+Ns*Omega
130 
131         Sdcs = _Sdc(freqs,nu,Te,R)
132         bessels = besselJ(Ns,z)**2.
133 
134         return dot(bessels, Sdcs)
135 
136     def _Sphi(phi,omega,nu,Te,nuac,Omega,R=R,nBessel=21):
137         if phi is None or not nuac*Omega:
138             return _Spa(omega,nu,Te,nuac,Omega,R=R,nBessel=nBessel)
139         z = nuac/Omega
140         nBessel -= nBessel%2-1    # Ensure it's odd
141         Ns = arange(-nBessel//2+1,nBessel//2+1)
142         freqs_m = -omega+Ns*Omega+nu
143         freqs_p = +omega+Ns*Omega+nu
144 
145         Sds_m = _Seq(freqs_m,Te,R)*exp(+1j*Ns*phi)
146         Sds_p = _Seq(freqs_p,Te,R)*exp(-1j*Ns*phi)
147 
148         bessels = besselJ(Ns,z)
149 
150         return 0.5*(exp(-1j*z*sin(phi))*dot(bessels,Sds_m) + exp(+1j*z*sin(phi))*dot(bessels,Sds_p))
151 
152     def _betap(p,omega,nu,Te,nuac,Omega,R=R,nBessel=21):
153         if p==0:
154             return _Spa(omega,nu,Te,nuac,Omega,R=R,nBessel=nBessel)
155         if not nuac*Omega:
156             return 0
157         z = nuac/Omega
158         nBessel -= nBessel%2-1    # Ensure it's odd
159         Ns = arange(-nBessel//2+1,nBessel//2+1)
160         freqs_m = -omega-Ns*Omega+nu
161         freqs_p = +omega+Ns*Omega+nu
162 
163         Sds_m = _Seq(freqs_m,Te,R)*(-1.)**p
164         Sds_p = _Seq(freqs_p,Te,R)
165 
166         bessels = besselJ(Ns,z)*besselJ(Ns+p,z)
167 
168         return dot(bessels, (Sds_m+Sds_p)/2.)
169 
170 
171     ###############
172     # Vectorizing #
173     ###############
174 
175     # Time Domain
176     tSeq = MemoizeMutable(vectorize(_tSeq))
177     tSdc = MemoizeMutable(vectorize(_tSdc))
178     tDSdc = MemoizeMutable(vectorize(_tDSdc))
179     tSPH = MemoizeMutable(vectorize(_tSPH))
180 
181     # Freq Domain
182     #Seq = MemoizeMutable(vectorize(_Seq))
183     Seq = MemoizeMutable(vectorize(_Seq))
184     Sdc = MemoizeMutable(vectorize(_Sdc))
185     Spa = MemoizeMutable(vectorize(_Spa))
186     Sphi = MemoizeMutable(vectorize(_Sphi))
187     betap = MemoizeMutable(vectorize(_betap))
188 
189     def Xp(p,omega,nu,Te,nuac,Omega,R=R,nBessel=21):
190         return betap(p,omega,nu,Te,nuac,Omega,R=R,nBessel=nBessel)+betap(-p,omega,nu,Te,nuac,Omega,R=R,nBessel=nBessel)
191 
192     def Yp(p,omega,nu,Te,nuac,Omega,R=R,nBessel=21):
193         return betap(p,omega,nu,Te,nuac,Omega,R=R,nBessel=nBessel)-betap(-p,omega,nu,Te,nuac,Omega,R=R,nBessel=nBessel)
```

```
194
195 def Sqz(p,omega,nu,Te,nuac,Omega, R=R,nBessel=21):
196     return sign(p)*betap(abs(p),omega,nu,Te,nuac,Omega,R=R,nBessel=21)+Spa(omega,nu,Te,nuac,Omega,R=R,nBessel=21)
```

D.2 Mathematica

D.2.1 Corrélateur courant-courant à l'équilibre

In[1]:=
```
Quit[]
```

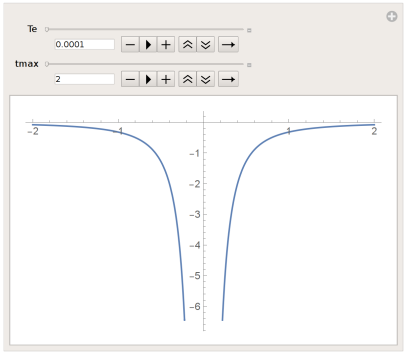
In[2]:=
```
f[ω_,Te_]:=2k Te/R  hbar ω/(2 k Te) Coth[hbar ω/(2 k Te)]
```

In[3]:=
```
g[t_,Te_]= InverseFourierTransform[f[ω,Te],ω,t,FourierParameters->{1,-1}]
```

Out[3]=
$$-\frac{k^2\ \pi\ Te^2\ Csch\left[\frac{k\ \pi\ t\ Te}{hbar}\right]^2}{hbar\ R}$$

In[4]:=
```
Manipulate[Plot[g[t,Te]/.{R→1,k→1,hbar→1},{t,-tmax,tmax}],{Te,0.0001,.1},{tmax,2,10}]
```

Out[4]=



In[5]:=
```
$Assumptions={hbar>0,α>0, k>0,Te>0,R>0,t∈Reals}
```

Out[5]=
```
{hbar>0,α>0,k>0,Te>0,R>0,t∈Reals}
```

In[6]:=
```
Limit[g[t,Te],Te→0]//FullSimplify
```

Out[6]=
$$-\frac{hbar}{\pi\ R\ t^2}$$

In[7]:=
```
Limit[g[t,Te],t→0]//FullSimplify
```

Out[7]=
```
-∞
```

In[8]:=
```
Limit[g[t,Te],t→∞]//FullSimplify
```

Out[8]=
```
0
```

In[9]:=
```
InverseFourierTransform[4k Te/R  Abs[hbar ω/(2 k Te)],ω,t,FourierParameters->{1,-1}]
```

Out[9]=
$$-\frac{hbar}{\pi\ R\ t^2}$$

In[10]:=
```
InverseFourierTransform[Abs[p],p,x,FourierParameters->{1,-1}]
```

Out[10]=
$$-\frac{1}{\pi\ x^2}$$

In[11]:=
```
Series[g[τ,Te],{τ,0,5}]
```

Out[11]= $$-\frac{\text{hbar}}{(\pi\ \text{R})\ \tau^2}+\frac{k^2\ \pi\ \text{Te}^2}{3\ \text{hbar}\ \text{R}}-\frac{(k^4\ \pi^3\ \text{Te}^4)\ \tau^2}{15\ (\text{hbar}^3\ \text{R})}+\frac{2\ k^6\ \pi^5\ \text{Te}^6\ \tau^4}{189\quad \text{hbar}^5\ \text{R}}+O[\tau]^6$$

In[12]:=
```
Series[g[τ,Te],{Te,0,5}]
```

Out[12]= $$-\frac{\text{hbar}}{\pi\ \text{R}\ \tau^2}+\frac{k^2\ \pi\ \text{Te}^2}{3\ \text{hbar}\ \text{R}}-\frac{(k^4\ \pi^3\ \tau^2)\ \text{Te}^4}{15\ (\text{hbar}^3\ \text{R})}+O[\text{Te}]^6$$

In[13]:=
```
h[τ_,Te_]=Limit[g[τ,Te]-g[τ,T0],{T0→0}];
```

In[14]:=
```
Series[h[τ,Te],{τ,0,2}]//FullSimplify
```

Out[14]= $$\{\frac{k^2\ \pi\ \text{Te}^2}{3\ \text{hbar}\ \text{R}}-\frac{(k^4\ \pi^3\ \text{Te}^4)\ \tau^2}{15\quad (\text{hbar}^3\ \text{R})}+O[\tau]^3\}$$

In[15]:=
```
Series[h[τ,Te],{Te,0,2}]//FullSimplify
```

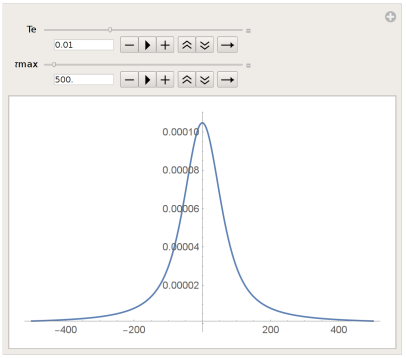
Out[15]= $$\{\frac{k^2\ \pi\ \text{Te}^2}{3\ \text{hbar}\ \text{R}}+O[\text{Te}]^3\}$$

In[16]:=
```
hh[τ_,Te_]=Piecewise[{
                {Series[h[τ,Te],{Te,0,2}]//Normal,Te==0},
                {Series[h[τ,Te],{τ,0,2}]//Normal,τ==0}
                    },
                 h[τ,Te]
                    ]
```

Out[16]=
$$\{\{\frac{k^2\ \pi\ \text{Te}^2}{3\ \text{hbar}\ \text{R}}\}\qquad \text{Te}==0$$
$$\{\frac{k^2\ \pi\ \text{Te}^2}{3\ \text{hbar}\ \text{R}}-\frac{k^4\ \pi^3\ \text{Te}^4\ \tau^2}{15\text{hbar}^3\ \text{R}}\}\qquad \tau==0$$
$$\{\frac{(-\text{hbar}^2-2\ k^2\ \pi^2\ \text{Te}^2\ \tau^2+\text{hbar}^2\ \text{Cosh}[\frac{2\ k\ \pi\quad \text{Te}\ \tau}{\text{hbar}}])\ \text{Csch}[\frac{k\ \pi\ \text{Te}\ \tau}{\text{hbar}}]^2}{2\ \text{hbar}\ \pi\ \text{R}\ \tau^2}\}\qquad \text{True}$$

In[17]:=
```
Manipulate[Plot[hh[τ,Te]/.{R→1,k→1,hbar→1},{τ,-τmax,τmax},
      PlotRange→All],{{Te,0.01},0,0.03,0.001},{{τmax,500},1,10000}]
```

Out[17]=



D.3 Régressions multiples avec paramètres partagés

Suit la classe Python personnalisée permettant d'effectuer un nombre arbitraire de lissages sur différents ensembles de données de manière commune. La version initiale du code, sans fonctionnalité de lissages communs, a été développée en collaboration avec Maxime Hardy.

Il est possible de partager certains paramètres parmi certains lissages via l'option de masques. Une version antérieure de ce même code est utilisée dans [44, 46] pour effectuer un lissage double sur deux séries de données distinctes, mais physiquement reliées, en partageant un paramètre parmi les régressions.

D.3.1 `nlfits.py`

```
#!/bin/python
# -*- coding: utf-8 -*-

# from pylab import * # Might be required by the loading script
from numpy import array, sqrt, diag, concatenate, mean, size, ndarray, iscomplex
from scipy.optimize import leastsq

class nlfit:
    '''
    Description to come
    '''
    def __init__(self, xs, ys, p0, fs, pmasks = None, fullo=1, xerrs=None, yerrs = None, verbose = True):

        xs = array(xs)
        ys = array(ys)

        if len(xs.shape)==1:
            xs, ys, fs = [xs], [ys], [fs]

        # Abscisse
        self.xs = array(xs)
        # xerrors not implemented yet (would require odr)
        if xerrs is None:
            self.xerrs = [[1.]*len(x) for x in self.xs]
        else:
            self.xerrs = array(xerrs)

        # Ordonnée
        self.ys = array(ys)
        if yerrs is None:
            self.yerrs = [[1.]*len(y) for y in self.ys]
        else:
            self.yerrs = array(yerrs)

        if pmasks is None:
            pmasks = [[1]*len(p0)]*len(xs)

        if any(len(i) != len(xs) for i in [ys, fs, pmasks, self.xerrs, self.yerrs]):
            raise(AssertionError,"List size don't match")

        self._p0 = p0
        self.para = p0
        self.fs = fs
        self.pmasks = pmasks
        #self.scales = [mean(abs(y)) for y in ys]
        self.scales = [y.max()-y.min() for y in ys]
        self.fullo = fullo
        self.verbose = verbose

```

```
50        def __call__(self, *args):
51            if len(args) == 1:
52                return self.__custom_call__(0,*args)
53            else:
54                return self.__custom_call__(*args)
55
56        def __custom_call__(self, fct_num, x):
57            return self.fs[fct_num](x,self._mask(self.para,self.pmasks[fct_num]))
58
59        def __getitem__(self,i):
60            return self.para[i]
61
62        def __len__(self):
63            return len(self.para)
64
65        def _mask(self, data, mask):
66            return [i for i,j in zip(data,mask) if j]
67
68        def _ps(self,p):
69            return [self._mask(p,mask) for mask in self.pmasks]
70
71        def _residuals(self, y, f, x, p, yerr, scale):
72            tmp = (y - f(x,p))/yerr/scale
73            return tmp
74
75        def _residuals_global(self, p):
76            errs = [self._residuals(y,f,x,mp,yerr,scale) for y,f,x,mp,yerr,scale in zip(self.ys, self.fs, self.xs, self._ps(p), self.yerrs,
                ↪  self.scales)]
77            return concatenate(errs)
78
79        def leastsq(self, **kwargs):
80            self.lsq = leastsq(self._residuals_global, self.para, full_output=self.fullo, **kwargs)
81            if self.lsq[1] is None:
82                if self.verbose: print('\n --- FIT DID NOT CONVERGE ---\n')
83                self.errs = None
84                self.err = None
85                self.chi2rs = None
86                return False
87            else:
88                self.para = self.lsq[0]
89                self.cv = self.lsq[1]
90                self.it = self.lsq[2]['nfev']
91                self.computevalues()
92                self.errs = array([self.sdcv*sqrt(chi2r) for chi2r in self.chi2rs])
93                self.err = self.errs[0]
94
95                if self.verbose:
96                    print(self)
97                return True
98
99
100       def computevalues(self):
101           self.sdcv = sqrt(diag(self.cv))
102           # Matrice de corrélation
103           self.corrM = self.cv/self.sdcv/self.sdcv[:,None]
104           self.chi2s = [sum(self._residuals(y,f,x,mp,yerr,scale)**2) \
105                         for y,f,x,mp,yerr,scale in zip(self.ys, self.fs, self.xs, self._ps(self.para),
106                                                        self.yerrs, self.scales)]
107           # Chi^2 réduit
108           self.chi2rs = [chi2/(len(y)-len(self.para)) for chi2,y in zip(self.chi2s, self.ys)]
109
110
111       def __str__(self):
112           s = '\n--- FIT ON FUNCTION{} {} ---'+\
113               '\n\nFit parameters are\n{}\nFit errors are\n{}\n\nFit covariance\n{}'+\
114               '\nFit correlation matrix\n{}\nReduced chi2s are {}\n\n'
115           fmt = ['S' if len(self.xs)>1 else '',', '.join([f.__name__ for f in self.fs]),
116                  self.para, self.errs, self.cv, self.corrM, self.chi2rs]
117           tmp = fmt[1].rfind(', ')
118           if not tmp == -1:
119               fmt[1] = fmt[1][:tmp] + ' and ' + fmt[1][tmp+2:]
120           return s.format(*fmt)
```

D.4 MPFR C++

Le fichier `mpreal.h` utilisé dans les codes ci-haut est présenté ici aux fins de référence et de complétude [115]. Par souci de concision, il est compressé par `zlib` et présenté en `base64`.

[illegible]

Pour obtenir le fichier original, exécuter le code suivant en `Python` :

```python
import zlib, base64, hashlib
s = "" # Bloc de texte brut ci-haut, *sans* retour à la ligne.
assert(hashlib.sha256(s).hexdigest() == '949c8f55d5e369d712d6c8f99df68e81e6440df20c7c1ea2d279ae428c860ea9') # Fails if *s* has errors.
with open("mreal.h",'w') as f:
    f.write(zlib.decompress(base64.b64decode(s)))
```

Bibliographie

[1] A. Einstein, B. Podolsky et N. Rosen. Can quantum-mechanical description of physical reality be considered complete ? *Phys. Rev.* **47**, 777–780 (1935). [cf. p. 1]

[2] J. S. Bell. On the Einstein Podolsky Rosen paradox. *Physics Physique Fizika* **1**, 195–200 (1964). [cf. p. 1]

[3] Nathaniel David Mermin. Is the Moon there when nobody looks ? reality and the quantum theory. *Physics Today* **38**(4), 38–47 (1985). [cf. p. 1]

[4] Alain Aspect, Philippe Grangier et Gérard Roger. Experimental tests of realistic local theories via bell's theorem. *Phys. Rev. Lett.* **47**, 460–463 (1981). [cf. p. 1]

[5] C. Abellán, A. Acín, A. Alarcón, O. Alibart, C. K. Andersen, F. Andreoli, A. Beckert, F. A. Beduini, A. Bendersky, M. Bentivegna, P. Bierhorst, D. Burchardt, A. Cabello, J. Cariñe, S. Carrasco, G. Carvacho, D. Cavalcanti, R. Chaves, J. Cortés-Vega, A. Cuevas, A. Delgado, H. de Riedmatten, C. Eichler, P. Farrera et al. Challenging local realism with human choices. *Nature* **557**(7704), 212–216 (2018). [cf. p. 1]

[6] Dominik Rauch, Johannes Handsteiner, Armin Hochrainer, Jason Gallicchio, Andrew S. Friedman, Calvin Leung, Bo Liu, Lukas Bulla, Sebastian Ecker, Fabian Steinlechner, Rupert Ursin, Beili Hu, David Leon, Chris Benn, Adriano Ghedina, Massimo Cecconi, Alan H. Guth, David I. Kaiser, Thomas Scheidl et Anton Zeilinger. Cosmic bell test using random measurement settings from high-redshift quasars. *Phys. Rev. Lett.* **121**, 080403 (2018). [cf. p. 1]

[7] Carlton M. Caves. Quantum limits on noise in linear amplifiers. *Phys. Rev. D* **26**, 1817–1839 (1982). [cf. p. 1, 5, 108]

[8] Jens Koch, Terri M. Yu, Jay Gambetta, A. A. Houck, D. I. Schuster, J. Majer, Alexandre Blais, M. H. Devoret, S. M. Girvin et R. J. Schoelkopf. Charge-insensitive qubit design derived from the Cooper pair box. *Phys. Rev. A* **76**, 042319 (2007). [cf. p. 1, 5]

[9] D. F. Santavicca, B. Reulet, B. S. Karasik, S. V. Pereverzev, D. Olaya, M. E. Gershenson, L. Frunzio et D. E. Prober. Characterization of terahertz single-photon-sensitive bolometric detectors using a pulsed microwave technique. *AIP Conference Proceedings* **1185**(1), 72–75 (2009). [cf. p. 1]

[10] John Clarke et Alex I. Braginski. *The SQUID Handbook Fundamentals and Technology of SQUIDs and SQUID Systems*, tome 1. Wiley-VCH, Weinheim, (2006). [cf. p. 1, 5]

[11] B. Yurke, L. R. Corruccini, P. G. Kaminsky, L. W. Rupp, A. D. Smith, A. H. Silver, R. W. Simon et E. A. Whittaker. Observation of parametric amplification and deamplification in a Josephson parametric amplifier. *Phys. Rev. A* **39**, 2519–2533 (1989). [cf. p. 1, 5, 163]

[12] Y. Nakamura, Yu. A. Pashkin et J. S. Tsai. Coherent control of macroscopic quantum states in a single-Cooper-pair box. *Nature* **398**(6730), 786–788 (1999). [cf. p. 1, 5]

[13] Vladimir E. Manucharyan, Jens Koch, Leonid I. Glazman et Michel H. Devoret. Fluxonium : Single Cooper-pair circuit free of charge offsets. *Science* **326**(5949), 113–116 (2009). [cf. p. 1, 5]

[14] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, C. Neill, P. O'Malley, P. Roushan, A. Vainsencher, J. Wenner, A. N. Korotkov, A. N. Cleland et John M. Martinis. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature* **508**(7497), 500–503 (2014). [cf. p. 1, 5]

[15] Rolf Landauer. The noise is the signal. *Nature* **392**(6677), 658–659 (1998). [cf. p. 1, 31]

[16] B. D. Josephson. Possible new effects in superconductive tunnelling. *Physics Letters* **1**(7), 251 – 253 (1962). [cf. p. 5, 75]

[17] P. W. Anderson et J. M. Rowell. Probable observation of the Josephson superconducting tunneling effect. *Phys. Rev. Lett.* **10**, 230–232 (1963). [cf. p. 5]

[18] B. D. Josephson. The discovery of tunnelling supercurrents. *Rev. Mod. Phys.* **46**, 251–254 (1974). [cf. p. 5]

[19] D. F. Santavicca, B. Reulet, B. S. Karasik, S. V. Pereverzev, D. Olaya, M. E. Gershenson, L. Frunzio et D. E. Prober. Energy resolution of terahertz single-photon-sensitive bolometric detectors. *Appl. Phys. Lett.* **96**(8), 083505 (2010). [cf. p. 5]

[20] Baptiste Royer, Arne L. Grimsmo, Alexandre Choquette-Poitevin et Alexandre Blais. Itinerant microwave photon detector. *Phys. Rev. Lett.* **120**, 203602 (2018). [cf. p. 5]

[21] Arne L. Grimsmo, Baptiste Royer, John Mark Kreikebaum, Yufeng Ye, Kevin O'Brien, Irfan Siddiqi et Alexandre Blais. Quantum metamaterial for nondestructive microwave photon counting. *Prépublication arXiv* (2020). [cf. p. 5]

[22] R. C. Jaklevic, John Lambe, A. H. Silver et J. E. Mercereau. Quantum interference effects in Josephson tunneling. *Phys. Rev. Lett.* **12**, 159–160 (1964). [cf. p. 5]

[23] Denis Vasyukov, Yonathan Anahory, Lior Embon, Dorri Halbertal, Jo Cuppens, Lior Neeman, Amit Finkler, Yehonathan Segev, Yuri Myasoedov, Michael L. Rappaport, Martin E. Huber et Eli Zeldov. A scanning superconducting quantum interference device with single electron spin sensitivity. *Nature Nanotechnology* **8**(9), 639–644 (2013). [cf. p. 5]

[24] K. M. Sliwa, M. Hatridge, A. Narla, S. Shankar, L. Frunzio, R. J. Schoelkopf et M. H. Devoret. Reconfigurable Josephson circulator/directional amplifier. *Phys. Rev. X* **5**, 041020 (2015). [cf. p. 5]

[25] Clemens Müller, Shengwei Guan, Nicolas Vogt, Jared H. Cole et Thomas M. Stace. Passive on-chip superconducting circulator using a ring of tunnel junctions. *Phys. Rev. Lett.* **120**, 213602 (2018). [cf. p. 5]

[26] M. A. Castellanos-Beltran, K. D. Irwin, L. R. Vale, G. C. Hilton et K. W. Lehnert. Bandwidth and dynamic range of a widely tunable Josephson parametric amplifier. *IEEE Transactions on Applied Superconductivity* **19**(3), 944–947 (2009). [cf. p. 5, 108]

[27] O. Yaakobi, L. Friedland, C. Macklin et I. Siddiqi. Parametric amplification in Josephson junction embedded transmission lines. *Phys. Rev. B* **87**, 144301 (2013). [cf. p. 5, 163]

[28] X. Zhou, V. Schmitt, P. Bertet, D. Vion, W. Wustmann, V. Shumeiko et D. Esteve. High-gain weakly nonlinear flux-modulated Josephson parametric amplifier using a squid array. *Phys. Rev. B* **89**, 214517 (2014). [cf. p. 5, 108, 163]

[29] J. Y. Mutus, T. C. White, E. Jeffrey, D. Sank, R. Barends, J. Bochmann, Yu Chen, Z. Chen, B. Chiaro, A. Dunsworth, J. Kelly, A. Megrant, C. Neill, P. J. J. O'Malley, P. Roushan, A. Vainsencher, J. Wenner, I. Siddiqi, R. Vijay, A. N. Cleland et John M. Martinis. Design and characterization of a lumped element single-ended superconducting microwave parametric amplifier with on-chip flux bias line. *Appl. Phys. Lett.* **103**(12), 122602 (2013). [cf. p. 5, 20]

[30] J. Y. Mutus, T. C. White, R. Barends, Yu Chen, Z. Chen, B. Chiaro, A. Dunsworth, E. Jeffrey, J. Kelly, A. Megrant, C. Neill, P. J. J. O'Malley, P. Roushan, D. Sank, A. Vainsencher, J. Wenner, K. M. Sundqvist, A. N. Cleland et John M. Martinis. Strong environmental coupling in a Josephson parametric amplifier. *Appl. Phys. Lett.* **104**(26) (2014). [cf. p. 5, 163]

[31] T. C. White, J. Y. Mutus, I.-C. Hoi, R. Barends, B. Campbell, Yu Chen, Z. Chen, B. Chiaro, A. Dunsworth, E. Jeffrey, J. Kelly, A. Megrant, C. Neill, P. J. J. O'Malley, P. Roushan, D. Sank, A. Vainsencher, J. Wenner, S. Chaudhuri, J. Gao et John M. Martinis. Traveling wave parametric amplifier with Josephson junctions using minimal resonator phase matching. *Appl. Phys. Lett.* **106**(24) (2015). [cf. p. 5, 108, 163, 187]

[32] Udson C. Mendes, Sébastien Jezouin, Philippe Joyez, Bertrand Reulet, Alexandre Blais, Fabien Portier, Christophe Mora et Carles Altimiras. Parametric amplification and squeezing with an ac- and dc-voltage biased superconducting junction. *Phys. Rev. Appl.* **11**, 034035 (2019). [cf. p. 5, 75, 76, 108, 120, 163]

[33] Daniel Huber Slichter. *Quantum Jumps and Measurement Backaction in a Superconducting Qubit*. Thèse de Doctorat, University of California, Berkeley, (2011). [cf. p. 5, 109, 115, 163]

[34] A. Bienfait, P. Campagne-Ibarcq, A. H. Kiilerich, X. Zhou, S. Probst, J. J. Pla, T. Schenkel, D. Vion, D. Esteve, J. J. L. Morton, K. Moelmer et P. Bertet. Magnetic resonance with squeezed microwaves. *Phys. Rev. X* **7**, 041011 (2017). [cf. p. 6, 163]

[35] A. D. Wilson-Gordon, V. Buek et P. L. Knight. Statistical and phase properties of displaced kerr states. *Phys. Rev. A* **44**, 7647–7656 (1991). [cf. p. 6, 24]

[36] Archana Kamal, Adam Marblestone et Michel Devoret. Signal-to-pump back action and self-oscillation in double-pump Josephson parametric amplifier. *Phys. Rev. B* **79**, 184301 (2009). [cf. p. 6, 24]

[37] Samuel Boutin. *Amplificateur paramétrique Josephson : Limite quantique, modélisation et caractérisation*. Mémoire de Maîtrise, Université de Sherbrooke, (2015). [cf. p. 6, 24]

[38] Samuel Boutin, David M. Toyli, Aditya V. Venkatramani, Andrew W. Eddins, Irfan Siddiqi et Alexandre Blais. Effect of Higher-Order Nonlinearities on Amplification and Squeezing in Josephson Parametric Amplifiers. *Phys. Rev. Appl.* **8**, 054030 (2017). [cf. p. 6, 24]

[39] G. Breitenbach, S. Schiller et J. Mlynek. Measurement of the quantum states of squeezed light. *Nature* **387**(6632), 471–475 (1997). [cf. p. 6, 163]

[40] S.M. Barnett et P.M. Radmore. *Methods in Theoretical Quantum Optics*. Oxford University Press, Oxford, (1997). [cf. p. 6, 163]

[41] Rodney Loudon. *The Quantum Theory of Light*. Clarendon Press, Oxford, (1973). [cf. p. 6, 163]

[42] Govind Agrawal. *Nonlinear fiber optics*. Quantum electronics–principles and applications. Elsevier / Academic Press, 4e édition, (2007). [cf. p. 6, 163]

[43] Ciprian Padurariu, Fabian Hassler et Yuli V. Nazarov. Statistics of radiation at josephson parametric resonance. *Phys. Rev. B* **86**, 054514 (2012). [cf. p. 6, 28, 185]

[44] Jean Olivier Simoneau. *Statistique de photons d'une jonction tunnel déduite de mesures de potentiel électrique à l'aide d'un amplificateur paramétrique Josephson*. Mémoire de Maîtrise, Université de Sherbrooke, (2015). [cf. p. 6, 8, 14, 20, 27, 109, 163, 168, 184, 237]

[45] Stéphane Virally, Jean Olivier Simoneau, Christian Lupien et Bertrand Reulet. Discrete photon statistics from continuous microwave measurements. *Phys. Rev. A* **93**, 043813 (2016). [cf. p. 6, 8, 14, 27]

[46] Jean Olivier Simoneau, Stéphane Virally, Christian Lupien et Bertrand Reulet. Photon-pair shot noise in electron shot noise. *Phys. Rev. B* **95**, 060301 (2017). [cf. p. 6, 8, 14, 27, 80, 163, 184, 237]

[47] Jean Olivier Simoneau, Stéphane Virally, Christian Lupien et Bertrand Reulet. Photocount statistics of the Josephson parametric amplifier : a question of detection. *Prépublication arXiv* (2020). [cf. p. 7, 8]

[48] Jean-Charles Forgues. *Étude du bruit de grenaille dans un conducteur simple : observation d'enchevêtrement, de compression d'état à deux modes et du quatrième cumulant des fluctuations statistiques dans le courant*

émis par une jonction tunnel. Thèse de Doctorat, Université de Sherbrooke, (2016). [cf. p. 18]

[49] E. Knill, R. Laflamme et G. J. Milburn. A scheme for efficient quantum computation with linear optics. *Nature* **409**(6816), 46–52 (2001). [cf. p. 28]

[50] R. J. Schoelkopf, P. J. Burke, A. A. Kozhevnikov, D. E. Prober et M. J. Rooks. Frequency dependence of shot noise in a diffusive mesoscopic conductor. *Phys. Rev. Lett.* **78**, 3370–3373 (1997). [cf. p. 31]

[51] B. L. Altshuler, L. S. Levitov et Yakovets A. Yu. Nonequilibrium noise in a mesoscopic conductor : A microscopic analysis. *JETP Letters* **59**(12), 857–863 (1994). [cf. p. 31, 32, 45, 47, 48, 184]

[52] Karl Thibault. *Corrélateur courant-courant dans le domaine temporel d'une jonction tunnel mesuré par spectroscopie micro-onde.* Mémoire de Maîtrise, Université de Sherbrooke, (2014). [cf. p. 31, 56, 57, 110, 184]

[53] Karl Thibault, Julien Gabelli, Christian Lupien et Bertrand Reulet. Pauli-heisenberg oscillations in electron quantum transport. *Phys. Rev. Lett.* **114**, 236604 (2015). [cf. p. 31, 32, 56, 110, 121, 124, 126, 171]

[54] H. Nyquist. Certain topics in telegraph transmission theory. *Transactions of the American Institute of Electrical Engineers* **47**(2), 617–644 (1928). [cf. p. 32, 45, 88, 114]

[55] C. E. Shannon. Communication in the presence of noise. *Proceedings of the IRE* **37**(1), 10–21 (1949). [cf. p. 32, 45, 88]

[56] Günther Krauss, Sebastian Lohss, Tobias Hanke, Alexander Sell, Stefan Eggert, Rupert Huber et Alfred Leitenstorfer. Synthesis of a single cycle of light with compact erbium-doped fibre technology. *Nature Photonics* **4**(1), 33–36 (2010). [cf. p. 32]

[57] A. Wirth, M. Th. Hassan, I. Grguraš, J. Gagnon, A. Moulet, T. T. Luu, S. Pabst, R. Santra, Z. A. Alahmed, A. M. Azzeer, V. S. Yakovlev, V. Pervak, F. Krausz et E. Goulielmakis. Synthesized light transients. *Science* **334**(6053), 195–200 (2011). [cf. p. 32]

[58] G. B. Lesovik et L. S. Levitov. Noise in an ac biased junction : Nonstationary Aharonov-Bohm effect. *Phys. Rev. Lett.* **72**, 538–541 (1994). [cf. p. 32, 53]

[59] Lafe Spietz, K. W. Lehnert, I. Siddiqi et R. J. Schoelkopf. Primary electronic thermometry using the shot noise of a tunnel junction. *Science* **300**(5627), 1929–1932 (2003). [cf. p. 32, 120]

[60] Lafe Spietz, R. J. Schoelkopf et Patrick Pari. Shot noise thermometry down to 10mK. *Appl. Phys. Lett.* **89**(18), 183123 (2006). [cf. p. 32, 120]

[61] Gabriel Gasse, Christian Lupien et Bertrand Reulet. Observation of squeezing in the electron quantum shot noise of a tunnel junction. *Phys. Rev. Lett.* **111**, 136601 (2013). [cf. p. 32, 53, 80, 81, 120, 148, 149, 163, 164, 165, 167, 169, 184]

[62] Gabriel Gasse. *Compression en phase et en quadrature dans le bruit de grenaille d'une jonction tunnel*. Mémoire de Maîtrise, Université de Sherbrooke, (2014). [cf. p. 32]

[63] J. Gabelli et B. Reulet. Dynamics of quantum noise in a tunnel junction under ac excitation. *Phys. Rev. Lett.* **100**, 026601 (2008). [cf. p. 32, 81, 120, 165, 168, 170, 173]

[64] Jean-Charles Forgues, Christian Lupien et Bertrand Reulet. Emission of microwave photon pairs by a tunnel junction . *Phys. Rev. Lett.* **113**, 043602 (2014). [cf. p. 32, 80]

[65] Jean-Charles Forgues, Christian Lupien et Bertrand Reulet. Experimental violation of bell-like inequalities by electronic shot noise. *Phys. Rev. Lett.* **114**, 130403 (2015). [cf. p. 32, 80, 149]

[66] Samuel Larocque, Edouard Pinsolle, Christian Lupien et Bertrand Reulet. Shot noise of a temperature-biased tunnel junction. *Phys. Rev. Lett.* **125**, 106801 (2020). [cf. p. 32, 119]

[67] Kun Il Park. *Fundamentals of Probability and Stochastic Processes with Applications to Communications*. Springer Publishing Company, Incorporated, 1re édition, (2017). [cf. p. 35, 36, 41, 60]

[68] John A. Gubner. *Probability and Random Processes for Electrical and Computer Engineers*. Cambridge University Press, USA, (2006). [cf. p. 35, 36, 37, 39, 40, 41, 60]

[69] Donald B. Percival et Andrew T. Walden. *Spectral Analysis for Physical Applications*. Cambridge University Press, (1993). [cf. p. 37, 87]

[70] Ya.M. Blanter et M. Büttiker. Shot noise in mesoscopic conductors. *Physics Reports* **336**(1), 1 – 166 (2000). [cf. p. 38, 45, 49]

[71] Michel Plancherel. Contribution à létude de la représentation d'une fonction arbitraire par des intégrales définies. *Rendiconti del Circolo Matematico di Palermo* **30**(1), 289–335 (1910). [cf. p. 40]

[72] Henri Lebesgue. *Leçons sur l'intégration et la recherche des fonctions primitives, professées au Collège de France*. Gauthier-Villars, Paris, (1904). [cf. p. 40]

[73] Robert G. Bartle. *The Elements of Integration and Lebesgue Measure*. John Wiley & Sons, (1995). [cf. p. 40, 42]

[74] Norbert Wiener. Generalized harmonic analysis. *Acta Math.* **55**, 117–258 (1930). [cf. p. 41]

[75] A. Khintchine. Korrelationstheorie der stationären stochastischen prozesse. *Mathematische Annalen* **109**(1), 604–615 (1934). [cf. p. 41]

[76] Boualem Picinbono. Chapitre 4 – Time-Frequency Signal and System Analysis. Dans *Time Frequency Analysis*, Boualem Boashash, 85 – 158. Elsevier Science, Oxford (2003). [cf. p. 44, 92, 129, 161, 162]

[77] J. B. Johnson. Thermal agitation of electricity in conductors. *Phys. Rev.* **32**, 97–109 (1928). [cf. p. 46]

[78] H. Nyquist. Thermal agitation of electric charge in conductors. *Phys. Rev.* **32**, 110–113 (1928). [cf. p. 46]

[79] B. M. Oliver. Thermal and quantum noise. *Proceedings of the IEEE* **53**(5), 436–454 (1965). [cf. p. 46]

[80] Wolfgang Pauli. Über den Zusammenhang des Abschlusses der Elektronengruppen im Atom mit der Komplexstruktur der Spektren. *Zeitschrift für Physik* **31**(1), 765–783 (1925). [cf. p. 47, 56]

[81] Frederick Reif. *Fundamentals of Statistical and Thermal Physics*. McGraw-Hill series in fundamentals of physics. McGraw Hill, New York, (1965). [cf. p. 48, 185]

[82] David W. Kammler. *A First Course in Fourier Analysis*. Cambridge University Press, 2e édition, (2008). [cf. p. 48]

[83] L. V. Keldysh. Diagram technique for nonequilibrium processes. *JETP* **20**(4), 1018–1026 (1965). [cf. p. 48]

[84] C. W. J. Beenakker et M. Büttiker. Suppression of shot noise in metallic diffusive conductors. *Phys. Rev. B* **46**, 1889–1892 (1992). [cf. p. 48]

[85] Annie A.M. Cuyt, Vigdis Petersen, Brigitte Verdonk, Haakon Waadeland et William B. Jones. *Handbook of Continued Fractions for Special Functions*. Springer, Pays-Bas, (2008). [cf. p. 50, 52, 67, 71]

[86] P. M. Woodward et I. L. Davies. Information theory and inverse probability in telecommunication. *Proceedings of the IEE - Part III: Radio and Communication Engineering* **99**(58), 37–44 (1952). [cf. p. 51, 54, 64]

[87] A. J. Dahm, A. Denenstein, D. N. Langenberg, W. H. Parker, D. Rogovin et D. J. Scalapino. Linewidth of the radiation emitted by a Josephson junction. *Phys. Rev. Lett.* **22**, 1416–1420 (1969). [cf. p. 53]

[88] R. J. Schoelkopf, A. A. Kozhevnikov, D. E. Prober et M. J. Rooks. Observation of "photon-assisted" shot noise in a phase-coherent conductor. *Phys. Rev. Lett.* **80**, 2437–2440 (1998). [cf. p. 53]

[89] A. A. Kozhevnikov, R. J. Schoelkopf et D. E. Prober. Observation of photon-assisted noise in a diffusive normal metal–superconductor junction. *Phys. Rev. Lett.* **84**, 3398–3401 (2000). [cf. p. 53]

[90] Werner Heisenberg. Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik. *Zeitschrift für Physik* **43**(3), 172–198 (1927). [cf. p. 56]

[91] H. P. Robertson. The uncertainty principle. *Phys. Rev.* **34**, 163–164 (1929). [cf. p. 56, 163, 177]

[92] Frédéric Bonnardot. *Comparaison entre les analyses angulaire et temporelle des signaux vibratoires de machines tournantes. Étude du concept de cyclostationnarité floue.* Thèse de Doctorat, Institut National Polytechnique de Grenoble, (2004). [cf. p. 58]

[93] William A. Gardner, Antonio Napolitano et Luigi Paura. Cyclostationarity : Half a century of research. *Signal Processing* **86**(4), 639 – 697 (2006). [cf. p. 58]

[94] Pierre Février, Christian Lupien et Bertrand Reulet. Fundamental and environmental contributions to the cyclostationary third moment of current fluctuations in a tunnel junction. *Phys. Rev. B* **101**, 245440 (2020). [cf. p. 58]

[95] Donald E. Knuth. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Addison-Wesley, Reading, Massachusetts, 3e édition, (1997). [cf. p. 65, 100]

[96] François Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault et Raymond Namyst. hwloc : a Generic Framework for Managing Hardware Affinities in HPC Applications. Dans *PDP 2010 - The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing* (IEEE Computer Society Press, Pisa, Italia, 2010). [cf. p. 78]

[97] Drew Batchelor, Michael Satran et Mike Jacobs. Processor groups. *Windows Dev Center* (2018). [cf. p. 78]

[98] Leonardo Dagum et Ramesh Menon. Openmp : an industry standard API for shared-memory programming. *Computational Science & Engineering, IEEE* **5**(1), 46–55 (1998). [cf. p. 78, 91, 198]

[99] Jean-Charles Forgues, Fatou Bintou Sane, Simon Blanchard, Lafe Spietz, Christian Lupien et Bertrand Reulet. Noise intensity-intensity correlations and the fourth cumulant of photo-assisted shot noise . *Sci. Rep.* **3**, 2869 (2013). [cf. p. 80]

[100] Gerald Goertzel. An algorithm for the evaluation of finite trigonometric series. *The American Mathematical Monthly* **65**(1), 34–35 (1958). [cf. p. 81]

[101] Donald B. Percival. Three curious properties of the sample variance and autocovariance for stationary processes with unknown mean. *The American Statistician* **47**(4), 274–276 (1993). [cf. p. 89]

[102] Tommy Wright. Lagrange's identity reveals correlation coefficient and straight-line connection. *The American Statistician* **46**(2), 106–107 (1992). [cf. p. 89]

[103] Wayne A. Fuller. *Introduction to statistical time series*. Wiley series in probability and statistics. Wiley, New York, (1976). [cf. p. 90]

[104] Nicholas J. Higham. The accuracy of floating point summation. *SIAM J. Sci. Comput* **14**, 783–799 (1993). [cf. p. 90]

[105] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, 2e édition, (2002). [cf. p. 90, 91]

[106] James W. Cooley et John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation* **19**, 297–301 (1965). [cf. p. 91]

[107] Johann Carl Friedrich Gauß. *Carl Friedrich Gauss, Werke*, tome 3, chapitre « Theoria interpolationis methodo nova tractata », 265–327. Königlichen Gesellschaft der Wissenschaften, Göttingen (1866). [cf. p. 91]

[108] Michael T. Heideman, Don H. Johnson et C. Sidney Burrus. Gauss and the history of the fast fourier transform. *Archive for History of Exact Sciences* **34**(3), 265–277 (1985). [cf. p. 91]

[109] Alan V. Oppenheim et Ronald W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall Press, New Jersey, 2e édition, (1999). [cf. p. 91]

[110] Wenzel Jakob, Jason Rhinelander et Dean Moldovan. *pybind11 – Seamless operability between C++11 and Python*. Librairie sous license de type BSD, (2015–2020). [cf. p. 91, 105]

[111] Matteo Frigo et Steven G. Johnson. *FFTW3 – Fastest Fourier Transform in the West*. Librairie sous license GNU General Public License, (2005–2018). [cf. p. 92]

[112] Matteo Frigo et Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE* **93**(2), 216–231 (2005). [cf. p. 92]

[113] Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, Philippe Théveny et Paul Zimmermann. *MPRF – A C library for Multiple-Precision Floating-point computations with correct Rounding*. Librairie sous license Lesser GNU General Public License, (2000–2019). [cf. p. 92]

[114] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier et Paul Zimmermann. MPFR : A Multiple-Precision Binary Floating-Point Library with Correct Rounding. *ACM Trans. Math. Softw.* **33**(2), 13–es (2007). [cf. p. 92]

[115] Pavel Holoborodko. *MPFR C++*. Librairie sous license GNU General Public License, (2008–2015). Disponible à l'annexe D.4. [cf. p. 92, 93, 239]

[116] Kenneth E. Iverson. *A Programming Language*. John Wiley & Sons, Inc., USA, (1962). [cf. p. 100]

[117] Donald E. Knuth. Two notes on notation. *The American Mathematical Monthly* **99**(5), 403–422 (1992). [cf. p. 100]

[118] Byeong Ho Eom, Peter K. Day, Henry G. LeDuc et Jonas Zmuidzinas. A wideband, low-noise superconducting amplifier with high dynamic range. *Nat Phys* **8**(8), 623–627 (2012). [cf. p. 108]

[119] D. Hover, S. Zhu, T. Thorbeck, G. J. Ribeill, D. Sank, J. Kelly, R. Barends, John M. Martinis et R. McDermott. High fidelity qubit readout with the superconducting low-inductance undulatory galvanometer microwave amplifier. *Appl. Phys. Lett.* **104**(15) (2014). [cf. p. 108]

[120] Michael B. Heaney. *Electrical Measurement, Signal Processing, and Displays*, chapitre 7 – Electrical Conductivity and Resistivity, 7.1–7.14. CRC Press, Boca Raton, 1re édition (2003). [cf. p. 117]

[121] Edouard Pinsolle, Alexandre Rousseau, Christian Lupien et Bertrand Reulet. Direct measurement of the electron energy relaxation dynamics in metallic wires. *Phys. Rev. Lett.* **116**, 236601 (2016). [cf. p. 119]

[122] Gabriel Gasse, Lafe Spietz, Christian Lupien et Bertrand Reulet. Observation of quantum oscillations in the photoassisted shot noise of a tunnel junction. *Phys. Rev. B* **88**, 241402 (2013). [cf. p. 120, 123]

[123] Jean-Charles Forgues, Gabriel Gasse, Christian Lupien et Bertrand Reulet. Non-classical radiation emission by a coherent conductor. *Comptes Rendus Physique* **17**(7), 718 – 728 (2016). Quantum microwaves / Micro-ondes quantiques. [cf. p. 149, 163, 169]

[124] E. Wigner. On the quantum correction for thermodynamic equilibrium. *Phys. Rev.* **40**, 749–759 (1932). [cf. p. 158]

[125] Leon Cohen. *Time-Frequency Analysis: Theory and Applications*. Prentice-Hall, Inc., USA, (1995). [cf. p. 158, 160, 161]

[126] P. Bertet, A. Auffeves, P. Maioli, S. Osnaghi, T. Meunier, M. Brune, J. M. Raimond et S. Haroche. Direct measurement of the wigner function of a one-photon fock state in a cavity. *Phys. Rev. Lett.* **89**, 200402 (2002). [cf. p. 161]

[127] Yoni Shalibo, Roy Resh, Ofer Fogel, David Shwa, Radoslaw Bialczak, John M. Martinis et Nadav Katz. Direct wigner tomography of a superconducting anharmonic oscillator. *Phys. Rev. Lett.* **110**, 100404 (2013). [cf. p. 161]

[128] Mihajlo Vanević, Julien Gabelli, Wolfgang Belzig et Bertrand Reulet. Electron and electron-hole quasiparticle states in a driven quantum contact. *Phys. Rev. B* **93**, 041416 (2016). [cf. p. 161]

[129] R. Bisognin, A. Marguerite, B. Roussel, M. Kumar, C. Cabart, C. Chapdelaine, A. Mohammad-Djafari, J.-M. Berroir, E. Bocquillon, B. Plaçais, A. Cavanna, U. Gennser, Y. Jin, P. Degiovanni et G. Fève. Quantum tomography of electrical currents. *Nat. Commun.* **10**(1), 3379 (2019). [cf. p. 161]

[130] Jean Ville. Theorie et Applications de la Notion de Signal Analytique. *Câbles et Transmissions* **2**(1), 61–74 (1948). [cf. p. 161]

[131] J. Aasi, J. Abadie, B. P. Abbott, R. Abbott, T. D. Abbott, M. R. Abernathy, C. Adams, T. Adams, P. Addesso, R. X. Adhikari, C. Affeldt, O. D. Aguiar, P. Ajith, B. Allen, E. Amador Ceron, D. Amariutei, S. B. Anderson, W. G. Anderson, K. Arai, M. C. Araya, C. Arceneaux, S. Ast, S. M. Aston, D. Atkinson et al. Enhanced sensitivity of the ligo gravitational wave detector by using squeezed states of light. *Nature Photonics* **7**(8), 613–619 (2013). [cf. p. 163]

[132] B. P. Abbott, R. Abbott, T. D. Abbott, M. R. Abernathy, F. Acernese, K. Ackley, C. Adams, T. Adams, P. Addesso, R. X. Adhikari, V. B. Adya, C. Affeldt, M. Agathos, K. Agatsuma, N. Aggarwal, O. D. Aguiar, L. Aiello, A. Ain, P. Ajith, B. Allen, A. Allocca, P. A. Altin, S. B. Anderson, W. G. Anderson et al. Observation of gravitational waves from a binary black hole merger. *Phys. Rev. Lett.* **116**, 061102 (2016). [cf. p. 163]

[133] B. Yurke, P. G. Kaminsky, R. E. Miller, E. A. Whittaker, A. D. Smith, A. H. Silver et R. W. Simon. Observation of 4.2-K equilibrium-noise squeezing

via a Josephson-parametric amplifier. *Phys. Rev. Lett.* **60**, 764–767 (1988). [cf. p. 163]

[134] N. E. Frattini, U. Vool, S. Shankar, A. Narla, K. M. Sliwa et M. H. Devoret. 3-wave mixing Josephson dipole element. *Appl. Phys. Lett.* **110**(22), 222603 (2017). [cf. p. 163]

[135] C. N. Lashmore-Davies. Parametric up-conversion of langmuir waves into transverse electromagnetic waves. *Phys. Rev. Lett.* **32**, 289–291 (1974). [cf. p. 167]

[136] Jinyu Sun, Shian Zhang, Tianqing Jia, Zugeng Wang et Zhenrong Sun. Femtosecond spontaneous parametric upconversion and downconversion in a quadratic nonlinear medium. *J. Opt. Soc. Am. B* **26**(3), 549–553 (2009). [cf. p. 167]

[137] Christina E. Vollmer, Christoph Baune, Aiko Samblowski, Tobias Eberle, Vitus Händchen, Jaromír Fiurášek et Roman Schnabel. Quantum up-conversion of squeezed vacuum states from 1550 to 532 nm. *Phys. Rev. Lett.* **112**, 073602 (2014). [cf. p. 167]

[138] Julien Gabelli et Bertrand Reulet. The noise susceptibility of a coherent conductor. Dans *Noise and Fluctuations in Circuits, Devices, and Materials*, Massimo Macucci, Lode K.J. Vandamme, Carmine Ciofi et Michael B. Weissman, tome 6600, 246 – 257. International Society for Optics and Photonics, SPIE, (2007). [cf. p. 168, 169, 173, 184]

[139] Herbert B. Callen et Theodore A. Welton. Irreversibility and generalized noise. *Phys. Rev.* **83**, 34–40 (1951). [cf. p. 185]

[140] Stéphane Virally et Bertrand Reulet. Unidimensional time-domain quantum optics. *Phys. Rev. A* **100**, 023833 (2019). [cf. p. 185]

[141] Arne L. Grimsmo et Alexandre Blais. Squeezing and quantum state engineering with Josephson travelling wave amplifiers. *npj Quantum Information* **3**(1), 20 (2017). [cf. p. 187]

[142] JTC 1/SC 22/WG 14. Information technology — Programming languages — C. Standard ISO/IEC 9899:2018, International Organization for Standardization, Genève, Suisse, (2018). [cf. p. 196]

[143] Peter J. Smith. A recursive formulation of the old problem of obtaining moments from cumulants and vice versa. *The American Statistician* **49**(2), 217–218 (1995). [cf. p. 197]

[144] Thorvald Nicolai Thiele. *Almindelig Iagttagelseslære : Sandsynlighedsregning og mindste Kvadraters Methode*. C. A. Reitzel, København, (1889).

Une traduction anglaise intitulée « *The general theory of observations : Calculus of probability and the method of least squares* » est disponible à la référence [145, §4]. [cf. p. 197]

[145] Steffen L. Lauritzen. *Thiele: Pioneer in Statistics*. Oxford University Press, Oxford, (2002). [cf. p. 197, 253]

[146] *Wolfram | Alpha*. Wolfram Alpha LLC, a subsidiary of Wolfram Research. Site Web, visité le 1er mai 2020. [cf. p. 199]