
Bureau d'étude

Logiciel de simulation d'un Lock-IN Amplifier avec évaluation des incertitudes utilisant un algorithme de Monte-Carlo



LE RUYET Guillaume Mise en place incertitude Monte Carlo



LORVELLEC Evan Génération signaux aléatoires & Lock-In



SIGUIER Adrien Chef de projet & Interface Graphique & Exportation en logiciel

Basé sur l'article : *Software simulation of a lock-in amplifier with application to the evaluation of uncertainties in real measuring systems.* [1]

Université Toulouse III - Paul Sabatier
M1 - Ingénierie du Diagnostic de l'Instrumentation et de la Mesure
Année universitaire 2023-2024

Table des matières

Table des figures

1	Introduction	1
2	Fondements théoriques et analyse en instrumentation	2
2.1	Le Lock-IN Amplifier	2
2.1.1	Le fonctionnement	2
2.1.2	Les applications	4
2.2	La Méthode Monte-Carlo	4
2.2.1	Le fonctionnement	5
2.2.2	Les applications	5
3	Application	7
3.1	Les entrées	7
3.2	Les sorties	8
3.3	Les paramètres d'influences	9
3.4	Les options ajoutées	10
4	Analyse	12
4.1	Quelques exemples	12
4.1.1	Instrument parfait, Signal parfait	12
4.1.2	Instrument parfait, Signal imparfait	13
4.2	Validation avec Kramers-Kronig	13
5	Compléments et perspectives	18
5.1	Concernant l'application	18
6	Conclusion	19
7	Bibliographie	20
A	Annexe	21
A.1	Les lois de Distributions	21
A.1.1	Distribution Gaussienne (Normale)	21
A.1.2	Distribution Uniforme	21
A.2	Modifications pour utilisateur avancé	22
A.2.1	Ajouter de nouveaux signaux	22
A.2.2	Ajouter de nouvelles distributions	23
A.2.3	L'exécutable	23
A.3	Document de spécification	23
A.4	Document de conception	26
A.5	Planning Initial	28
A.6	Planning réalisé	29
A.7	Compte-Rendu réunion hebdomadaire	30
A.8	Script Application	32
A.9	Script Python Kramers_Kronig	32

Table des figures

1	Face avant du Lock-In Amplifier	2
2	Vue synoptique d'un Lock-In Amplificateur	3
3	Face avant logiciel.	7
5	Distribution de Theta et R	9
6	Distribution de Theta et R non bimodale	9
8	Réponse indicielle d'un filtre du premier ordre (RC).	11
9	Simulation pour un Instrument parfait, signal parfait	12
10	Simulation pour un Instrument parfait, signal imparfait	13
11	Partie réelle et imaginaire de la fonction de transfert du circuit RC . .	15
12	Résultats du programme de simulation	15
13	Résultats de l'expérience réelle	16
14	Données traitées pour la partie imaginaire et réelle	16
15	Tracé Loi Gaussienne	21
16	Tracé Loi Uniforme	22

1 Introduction

Le Lock-In ou détecteur synchrone est un instrument de mesure permettant de mettre en oeuvre des méthodes du traitement du signal pouvant extraire l'amplitude de signaux basse fréquence sur des signaux pouvant être noyés dans du bruit par multiplication du signal avec un signal sinusoïdal de fréquence proche de celle de la fréquence moyenne à détecter.

La majorité des détections synchrones actuelles sont basées sur une haute performance de traitement numérique du signal. Les détecteurs synchrones numériques quant à eux, sont supérieurs à leurs homologues analogiques dans tous les critères de performance comme le bruit d'entrée ou la stabilité. En plus d'une meilleure performance, elles peuvent inclure plusieurs démodulateurs qui permettent d'analyser un signal avec différents paramètres de filtre (passe bande ou autre) ou plusieurs fréquences simultanément. De plus, les données expérimentales peuvent être analysées par des outils tels qu'un oscilloscope, analyseur de spectre FFT etc... Certains modèles de détecteurs synchrones numériques sont contrôlables par ordinateur et disposent d'une interface graphique dédiée et un choix d'interfaces de programmation varié.

La mesure des signaux basses fréquences étant de plus en plus répandue dans de nombreux domaines de la physique et de l'ingénierie, cela engendre une nécessité de contrôler la détection synchrone. Par exemple en physique des matériaux, on utilise de faibles signaux pour caractériser les propriétés de la matière en comparant les signaux de référence avec ceux appliqués aux matériaux.

Dans le cadre de ce bureau d'étude, nous allons réaliser une simulation de détecteur synchrone par la création d'un logiciel. En plus d'apporter une probabilité de cet instrument physique sur un ordinateur, cette application permettra de configurer une incertitude de Monte Carlo sur les paramètres d'entrées et d'évaluer l'incertitude sur les paramètres de sorties. Il sera ainsi possible d'en tirer des conclusions sur la fiabilité des tests réalisés.

Remarque : Le logiciel peut être détecté comme un virus sur l'ordinateur. En effet, celui-ci n'étant pas signé, il peut être détecté comme un virus car il interagit avec des dossiers afin de pouvoir charger la notice d'utilisation et un preset de données. Pouvoir signer ce logiciel serait une solution, mais étant coûteuse, nous avons jugé qu'il n'était pas nécessaire de la déployer.

2 Fondements théoriques et analyse en instrumentation

Cette section est dédiée à la description du sujet d'étude. Ainsi, une présentation du Lock-In Amplifier, son fonctionnement et ses domaines d'applications seront présentés. Ensuite, une explication du calcul de la propagation des incertitudes via la méthode de Monte-Carlo sera amené.

2.1 Le Lock-IN Amplifier

Le Lock-In Amplifier est un appareil de mesure utile dans le cadre de l'instrumentation et du traitement du signal. Cet instrument permet d'extraire un signal avec une onde porteuse au milieu d'un environnement très bruyant. Selon le type d'instrument, il peut détecter des signaux jusqu'à un million de fois plus faible que les composantes de bruits.

2.1.1 Le fonctionnement



FIGURE 1 – Face avant du Lock-In Amplifier

Le Lock-In Amplifier possède 2 entrées, utilisable pour le signal ainsi qu'une autre entrée "*Ref*" utilisée par le signal de référence sur lequel l'instrument doit se verrouiller pour la détection synchrone.

La sortie *X* correspond à la composante en phase du signal modulé après détection synchrone.

La sortie *Y* correspond à la composante en quadrature du signal modulé après détection synchrone.

C'est grâce à ces deux sorties que nous sommes en capacité de calculer l'amplitude et la phase de notre signal d'entrée modulé à la fréquence de référence grâce au procédé expliqué plus bas. Par deux boutons, nous avons accès aux options d'affichage, nous permettant ainsi de choisir d'afficher l'amplitude là où est affiché *X* et la phase là où est affiché *Y*.

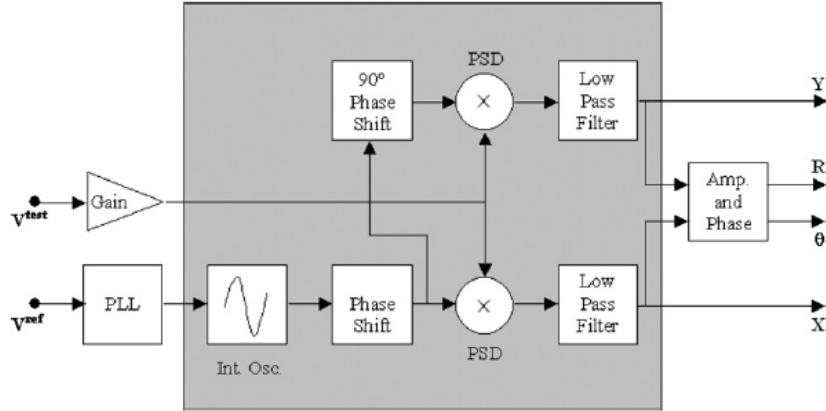


FIGURE 2 – Vue synoptique d'un Lock-In Amplificateur

Pour pouvoir effectuer une détection synchrone sur un signal bruité, le Lock-In Amplifier à besoin de se "verrouiller" sur une fréquence de référence.

L'instrument va moduler le signal d'entrée à la fréquence du signal de référence. Pour se faire, 2 signaux de référence sont produits, un signal déphasé de 90 degrés par rapport au premier et un autre inchangé, cette opération s'appelle la démodulation de quadrature et permet d'extraire à la fois l'amplitude et la phase des composantes qui nous intéressent du signal d'entrée bruité et ce, avec une grande précision.

Une fois ces 2 signaux de références produits ils sont multipliés, à l'aide d'un multiplicateur analogique, au signal d'entrée produisant une sortie à la fréquence de modulation et de ses harmoniques. Les deux signaux obtenus sont ensuite filtrés par un filtre passe bas permettant d'éliminer les composantes aux fréquences les plus élevées.

C'est alors que la détection synchrone se fait, le détecteur synchronisé au signal de référence extrait les composantes du signal à cette même fréquence ce qui nous permet d'obtenir la phase et l'amplitude du signal qui nous intéresse.

Mathématiquement, pour un signal sinusoïdal, voici les opérations clés du Lock-In Amplifier :

$$V_{sig} = \sqrt{2}V_{sig0}\sin(2\pi f_0 t + \theta_{sig}) \quad (1)$$

$$V_{ref} = \sqrt{2}V_{sig0}\sin(2\pi f_0 t + \theta_{deph}) \text{ et } V_{ref\perp} = \sqrt{2}V_{sig0}\sin(2\pi f_0 t + \theta_{deph} + \pi/2) \quad (2)$$

On a, après passage dans les multiplicateur analogique :

$$\otimes_A = V_{ref} * V_{sig} = V_{sig0}[\cos(\theta_{deph} - \theta_{sig}) - \cos(2w_0 + \theta_{deph} + \theta_{sig})] \quad (3)$$

$$\otimes_H = V_{ref\perp} * V_{sig} = V_{sig0}[-\sin(\theta_{deph} - \theta_{sig}) + \sin(2w_0 + \theta_{deph} + \theta_{sig})] \quad (4)$$

Nous pouvons donc en déduire les expressions de X et Y en sortie des filtres passe-bas :

$$X(t) = V_{sig0}[\cos(\theta_{deph} - \theta_{sig}) - |H(2w_0)| \cos(2w_0 + \theta_{deph} + \theta_{sig} + \arg[H(2w_0)])] \quad (5)$$

$$X(t) = V_{sig0}[-\sin(\theta_{deph} - \theta_{sig}) + |H(2w_0)| \sin(2w_0 + \theta_{deph} + \theta_{sig} + \arg[H(2w_0)])] \quad (6)$$

On peut donc avoir pour des fréquences basses devant la fréquence de coupure du filtre les relations suivantes ; pour θ déphasé nul :

$$X(t) = V_{sig0}\cos(\theta_{sig}) = X_0 \quad (7)$$

$$Y(t) = V_{sig0}\sin(\theta_{sig}) = Y_0 \quad (8)$$

Nous pouvons calculer l'amplitude et la phase des composantes du signal qui nous intéressent :

$$V_{sig0} = \sqrt{Y_0^2 + X_0^2} \quad \theta_{sig} = \arctan\left(\frac{Y_0}{X_0}\right) \quad (9)$$

La même méthodologie peut s'appliquer pour tout type de signaux, il suffit d'adapter l'expression des signaux théoriques dans le calcul des différentes composantes du Lock-In. Nous retrouvons bien alors en sortie des filtres au travers de X et Y des signaux déphasé si un déphasage est appliqué et de même fréquence que le signal de référence.

2.1.2 Les applications

Le Lock-In Amplifier est bénéfique dans une multitude de domaines divers et variés, notamment dans la spectroscopie optique où il permet de détecter les variations de signaux optiques faibles lorsque l'on a du bruit élevé pour de la spectroscopie de fluorescence ou d'absorption optique.

On l'utilise aussi en mécanique pour évaluer la contrainte et la déformation, celui-ci peut mesurer de maigres déformations dans les matériaux sous l'application de différentes charges.

Le Lock-In Amplifier permet de mesurer les vibrations et les signaux acoustiques à des fréquences précises dans des environnements bruyants et pour des signaux spécifiques.

Cet instrument retrouve son utilisation intéressante dans plusieurs domaines en plus de ceux cités plus haut : l'imagerie médicale, les systèmes de commutation et la télemétrie ou encore en électronique. C'est un outil intéressant permettant d'effectuer des mesures précises et fiables.

Une grande partie de ces applications peuvent se retrouver dans le site de Zurich Instrument[2] avec un spectre d'application plus important que celles citées ci-dessus.

2.2 La Méthode Monte-Carlo

La méthode Monte-Carlo est une technique probabiliste utilisée pour estimer des résultats numériques en utilisant des échantillons aléatoires. En effet, ici nous

préférions utiliser ce genre de méthode, les incertitudes étant nombreuses, il est plus simple de réaliser un algorithme tel que celui-ci plutôt que de réaliser des calculs via la propagation des incertitudes.

2.2.1 Le fonctionnement

Pour mettre en place la méthode il faut d'abord poser le contexte de ce que l'on souhaite estimer, cela signifie qu'il faut définir un système dont on veut estimer un résultat numérique. Dans notre cas on définit le système comme quelque chose d'assimilable au Lock-In, les grandeurs d'entrées sont alors un signal bruité dont chaque paramètre est indépendant, un signal de référence et les paramètres d'un filtre passe bas. Les grandeurs de sortie du système sont les valeurs de R et θ (2.1.1) qui sont les valeurs qu'on souhaite estimer, de même que leur incertitude sur chacune d'elles.

Les paramètres de chaque grandeurs d'entrées sont générés aléatoirement selon un type de distribution choisi et dont les valeurs sont tirées depuis un intervalle donné, fixé par l'incertitude de chaque paramètre (3.1).

Puis vient le moment de l'évaluation du système pour chaque ensemble de paramètres d'entrées générées, le système estime le résultat en fonction des ces grandeurs. Une fois que le système a été évalué pour un grand nombre d'ensembles de valeurs aléatoires, la moyenne et l'écart type des résultats obtenus sont calculés. Ces estimations sont utilisées comme approximation du résultat souhaité.

La précision de l'estimation dépend du nombre d'échantillons aléatoires utilisés. Plus le nombre d'échantillons est élevé, plus l'estimation est précise.

Pour confirmer la fiabilité de l'estimation, il est souvent nécessaire de répéter le processus avec différents ensembles de nombres aléatoires et/ou des paramètres différents. Cela permet de valider l'estimation et d'évaluer sa robustesse.

2.2.2 Les applications

Elle est largement utilisée dans divers domaines tels que la physique, les finances, l'ingénierie, les sciences de la vie et bien d'autres encore. Dans chacun de ces domaines, où la méthode est utilisée, il y a de nombreux systèmes complexes comportant de nombreuses variables interdépendantes dont l'estimation de grandeur de sortie est difficile.

La méthode de Monte Carlo permet de modéliser ces systèmes de manière probabiliste en prenant en compte l'incertitude et la variabilité des paramètres impliquées. En simulant différentes conditions possibles et en évaluant les résultats probables, cela impacte la prise de décision en matière de gestion des risques. Mais peut être aussi utilisée pour l'optimisation de systèmes et de processus complexes. En générant un grand nombre de solutions potentielles de manière aléatoire et en évaluant leurs performances, on peut ainsi déterminer des solutions qui se rapprochent de la

forme la plus optimisée de notre système.

Dans notre cas on utilise cette méthode pour simuler des processus aléatoires. Cette simulation permet d'estimer au mieux l'approximation de R et θ mais aussi de calculer leurs incertitudes, dont la valeur obtenue par une méthode analytique d'incertitude composée est plus délicate à déterminer, du fait de la nature de l'objet du Lock-In. En répétant un grand nombre de fois le calcul avec, pour chaque paramètre, un tirage aléatoire différent, on obtient l'optimum de la meilleure estimation possible.

3 Application

L'application que nous avons réalisé permet de simuler l'instrument de mesure. Cet exécutable permet ainsi de saisir des paramètres d'entrées et nous affiche des graphes ainsi que des résultats. L'interface réalisée ressemble à celle fournie dans l'article [1]. On note que cette interface a été réalisée avec le langage Python.

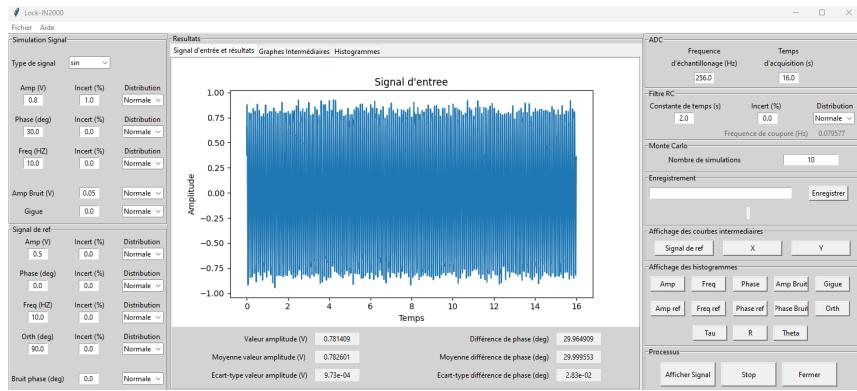


FIGURE 3 – Face avant logiciel.

On observe alors qu'il y a plusieurs paramètres à saisir pour faire fonctionner ce logiciel. En retour, plusieurs quantités sont renvoyées, on explorera en détail le fonctionnement de l'application ci-dessous.

3.1 Les entrées

Les différentes entrées que peut saisir l'utilisateur concernent le signal à évaluer, le signal de référence, ainsi que les paramètres de l'instrument. En réalité, les signaux analysés et l'appareil de mesure sont imparfaits, ainsi, ils présentent une incertitude. C'est pour cette raison qu'il est possible de saisir des "*erreurs*" associées aux différents paramètres. Dans cette interface, la saisie des incertitudes ce fait en pourcentage. Ce choix est délibéré : en effet, l'objectif est de pouvoir représenter au mieux un instrument réel, ainsi, dans cette optique, il a été nécessaire de donner une incertitude. Dans cet objectif, la réelle valeur de l'écart-type dépend principalement du paramètre "*moyen*". Le logiciel a été conçu de sorte que l'écart-type soit calculé comme suit :

$$\sigma = \text{Valeur_saisie} * \frac{\text{incertitude_pourcent}}{100} \quad (10)$$

Ce choix nous a semblé nécessaire à des fins pratiques d'utilisation. L'utilisateur est libre de choisir ses paramètres, ainsi, pour plusieurs paramètres de simulations, l'utilisateur est dans la possibilité de laisser sa mesure en pourcent. De plus, la plupart des incertitudes fournies dans une documentation sont exprimées en pourcentage, c'est aussi une des raisons de notre choix. Nous avons également implémenté deux lois de distributions, permettant de choisir entre une distribution Normale

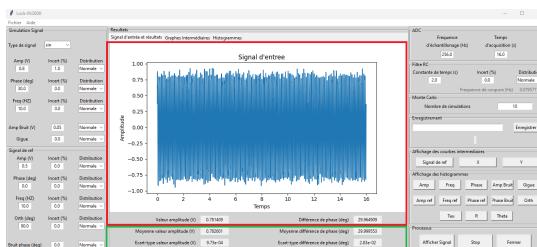
(A.1.1) ou Uniforme (A.1.2). Ces incertitudes sont produites notamment par le calibre de l'appareil ou encore l'affichage avec des digits.

3.2 Les sorties

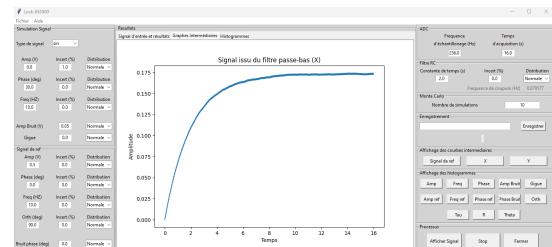
En sortie, plusieurs quantités sont observées. Premièrement, après avoir appuyé sur le bouton "Afficher Signal", le signal d'entrée (*signal étudié*) est affiché dans l'onglet "Signal d'entrée et résultats" pour une seule réalisation de Monte-Carlo. Dans ce même onglet, on retrouve le calcul de l'amplitude du signal (R) et de la différence de phase (θ) entre le signal de référence et le signal d'entrée, toujours pour cette simulation de Monte-Carlo (*cadre rouge*). On retrouve en dessous, dans le cadre vert, les mêmes calculs pour l'ensemble des itérations réalisées par le script. On y trouve également les écarts-types associés à ces calculs. Ce signal d'entrée est visible sur l'image (4a).

Ensuite, nous avons opté pour l'affichage du signal de référence, l'affichage du signal d'entrée multiplié à celui de signal de référence (*noté X*) ?? puis le même signal déphasé de 90° (*noté Y*) ?? dans le cadre "Graphes Intermédiaires" (4b).

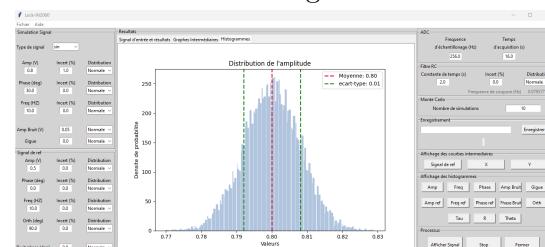
Enfin, nous affichons les histogrammes des distributions de chaque paramètres d'entrées et de sorties (R et θ) dans l'onglet "Histogrammes" (4c). Sur chaque histogramme, la valeur moyenne et l'écart-type des distributions sont affichés.



(a) Affichage cadre signal d'entrée et résultats



(b) Affichage du cadre d'affichage des signaux intermédiaires. (Exemple avec X)



(c) Affichage du cadre des histogrammes.

Remarque : Après modification des paramètres, si l'affichage des signaux intermédiaires ou des histogrammes est souhaité, il est indispensable de passer par la première étape (unique) "Afficher Signal" avant d'effectuer toute autre opération. Une fois cette dernière commande effectuée, il est possible d'actionner d'autres bou-

tons sans passer par la première étape.

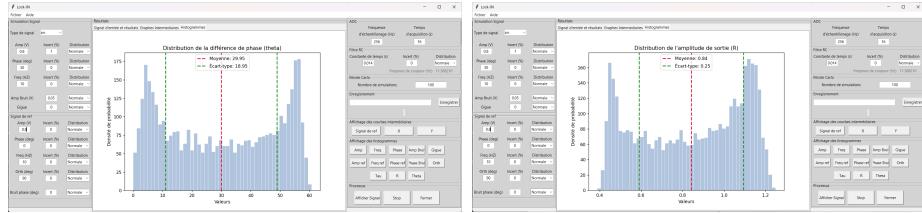


FIGURE 5 – Distribution de Theta et R

Nous nous attendions à avoir une distribution de R et θ de type gaussienne (*ou Normale*), selon le choix de distribution majoritaire dans les paramètres, autour de la valeur moyenne et d'écart type l'incertitude associé aux paramètres. Hors nous observons dans certains cas que les graphiques forment deux maxima distincts formant deux gaussiennes (5) et ceux-ci quelque que soit le type de distribution et la présence de bruit dans le signal. C'est une distribution bi-modale gaussienne (*/3*), répartie de manière symétrique par rapport à la valeur moyenne, cela montre que les calculs se font avec deux signaux distincts, que chacune des gaussiennes représentent la répartition d'un signal et que la moyenne donne résultat final attendu.

Dans d'autres cas (6), l'explication de la distribution polymodale reste complexe et ouverte à l'interprétation. Le Lock-In, sensible à de nombreux paramètres et caractérisé par sa non-linéarité, suggère une explication probablement complexe, étant donné l'entrecroisement des paramètres influents.

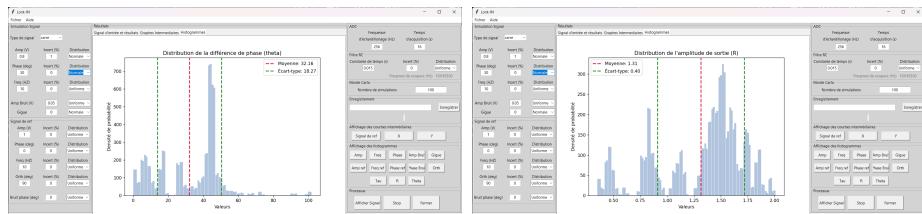


FIGURE 6 – Distribution de Theta et R non bimodale

Remarque : Énormément de valeurs sont représentées dans l'histogramme des distributions (quelque soit le nombre d'itérations de Monte-Carlo), en effet, nous avons préféré générer une valeur aléatoire pour chaque instant du signal. Cette représentation a été choisie pour simuler au mieux un instrument réel.

3.3 Les paramètres d'influences

Notre logiciel simule un Lock-In Amplifier numérique utilisant un filtre de type passe bas, ce qui signifie que nous devons avoir la possibilité, selon notre signal, de modifier sa fréquence de coupure et donc τ sa constante de temps. Nous devons aussi avoir la possibilité de modifier la fréquence d'échantillonnage.

Ces deux paramètres influent sur le résultat de R et Θ , le paramètre τ impacte sur le temps de montée de la réponse du filtre, car celui-ci n'est pas idéal, le régime permanent n'est pas atteint instantanément après ce dernier il faut alors attendre qu'il finisse par se stabiliser avant de commencer les calculs des grandeurs de sortie souhaités. Il permet aussi, selon sa taille, de réduire ou augmenter la bande passante, permettant d'affaiblir plus aisément le bruit de fond et de mieux suivre les changements rapides du signal en présence de bruit.

La fréquence d'échantillonnage, quant à elle, doit être assez grande pour capturer le plus de détail du signal d'entrée pour le moduler plus rapidement ; il est conseillé de faire attention au sur-échantillonnage qui ici pourrait entraîner une consommation de ressources plus importantes sans réel intérêt.

Un autre paramètre clé dans cette simulation se retrouve dans la gigue qui est un bruit supplémentaire entraînant une variation temporelle aléatoire causant des modifications de phase, fréquence et période. L'impact du bruit sur la mesure peut être augmenté par son amplitude ou sa gigue, une trop grande valeur de gigue aurait plusieurs effets dont la dégradation du signal, de sa qualité ou encore d'une perte de synchronisation.

Enfin, les incertitudes jouent aussi un rôle clé. Une incertitude sur la fréquence induit un décalage lors de la synchronisation entre les signaux d'entrée et de référence, ainsi toute la mesure est faussée. Enfin, une incertitude sur le degré d'orthogonalité joue aussi un rôle essentiel. Si cette orthogonalité n'est déviée que d'un demi degré, l'ensemble des mesures fournies par l'instrument sont erronées. En effet, ce déphasage étant important dans la reconstruction du signal, il est impératif que ce degré d'orthogonalité soit précis.

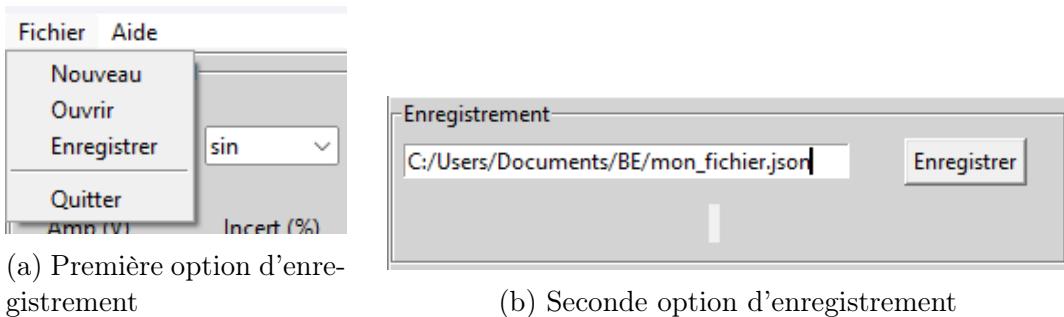
3.4 Les options ajoutées

Pour rendre l'application le plus ergonomique possible, nous avons ajouté quelques options. Premièrement, nous avons fait en sorte de déployer un jeu de données par défaut. Ce jeu de données est basé sur l'article fourni initialement. Il permet de simuler un instrument parfait avec un signal imparfait (*que nous verrons dans la prochaine partie*). Nous avons également défini d'autres jeux de données afin de simuler différents cas que nous étudierons dans le chapitre suivant.

D'autres fonctionnalités ont été ajoutées dans le menu (*7a*). En plus de pouvoir charger des jeux de données pré-faits, il est possible pour l'utilisateur de sauvegarder son propre jeu de données et le charger dans l'application ultérieurement. Cette option a été mise en place pour pouvoir étudier à nouveau un set de paramètres qui semblait présenter des caractéristiques intéressantes ou encore pouvoir étudier ce jeu sur une autre machine possédant le logiciel. Il est possible également de pouvoir "*reset*" (*effacer*) les paramètres afin de reprendre une interface vierge.

Concernant l'enregistrement des données, il est possible pour l'utilisateur de

sauvegarder les paramètres entrés en utilisant deux moyens : le premier consiste à naviguer dans le menu et sélectionner l'option "*Enregistrer*" (7a). La seconde option permet, via l'interface, d'enregistrer le jeu de données actuel dans un répertoire et nom de fichier précis (7b). Lorsque l'utilisateur oublie de saisir une donnée, un "*warning*" (*message d'alerte*) intervient sur l'écran pour le prévenir.



Afin de ne pas fournir des résultats erronés, nous avons mis en place un autre *Warning*, en effet, si l'utilisateur entre un temps d'acquisition inférieur ou égal à 2τ (τ étant la constante de temps du filtre *RC*), un autre message apparaîtra, demandant à l'utilisateur de modifier un des paramètres. En effet, le filtre *RC* à besoin de se "*charger*", ainsi, d'après notre configuration, il commencera à fournir des résultats fiable (à 86% de sa valeur finale).

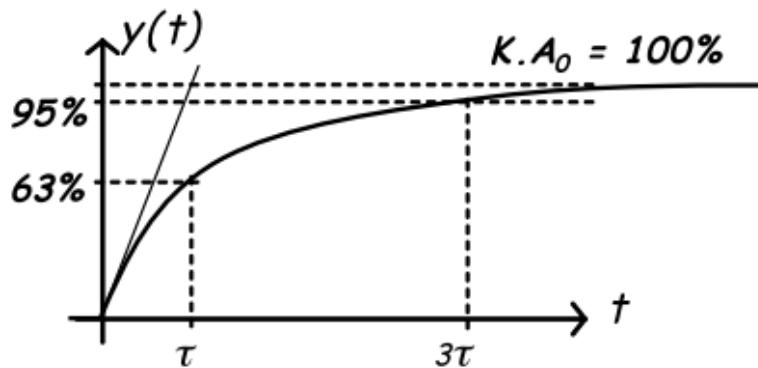


FIGURE 8 – Réponse indicielle d'un filtre du premier ordre (RC).

La figure représentée ci-dessus est une réponse indicielle d'un filtre du premier ordre, tel que le filtre passe-bas (*Circuit RC*). La valeur de cette courbe à 2τ est donc fixée à 86% de la valeur finale fournie par le filtre. Nous avons estimé qu'au bout de ce temps, le filtre donne une valeur à laquelle on peut se fier. A des fins de stabilité des résultats, nous aurions pu fournir une précision plus accrue en fixant le traitement des valeurs à 3τ soit 95% de la valeur finale.

4 Analyse

Avant d'aborder les potentielles évolutions de cette application, nous allons discuter de quelques exemples. Nous essaierons d'en simuler quelques-uns en comprenant des signaux parfaits ou non. Enfin, nous aborderons une validation de l'instrument mis sous forme numérique en explorant les relations de Kramers-Kronig.

4.1 Quelques exemples

Abordons d'abord quelques exemples de simulations que nous pouvons réaliser avec notre instrument numérique.

4.1.1 Instrument parfait, Signal parfait

Bien que les signaux parfaits ne sont pas représentatifs de ce que nous pouvons rencontrer dans la réalité, ils n'en restent pas moins un moyen simple d'étudier et de comprendre la théorie des signaux.

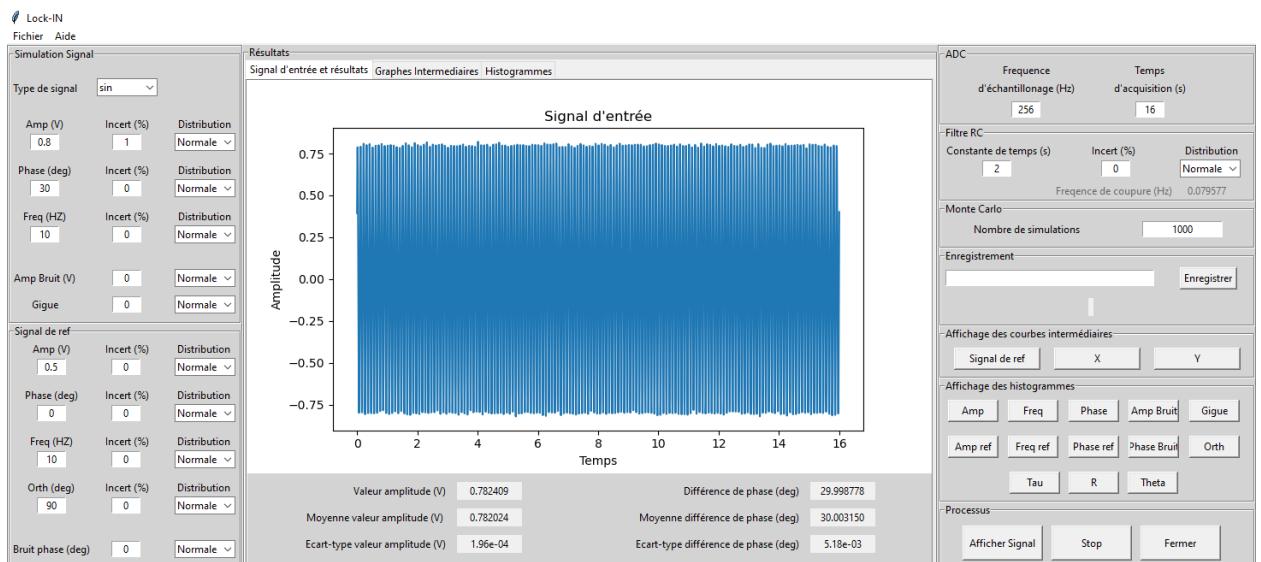


FIGURE 9 – Simulation pour un Instrument parfait, signal parfait

Il ne faut pas confondre bruit et incertitude. Un signal présentant un aspect aléatoire (*comme ici appliqué sur l'amplitude*), n'est pas un signal bruité, ainsi, cette incertitude n'interfère pas dans l'étude menée.

4.1.2 Instrument parfait, Signal imparfait

Un signal peut être considéré imparfait du moment qu'on le noie dans du bruit.

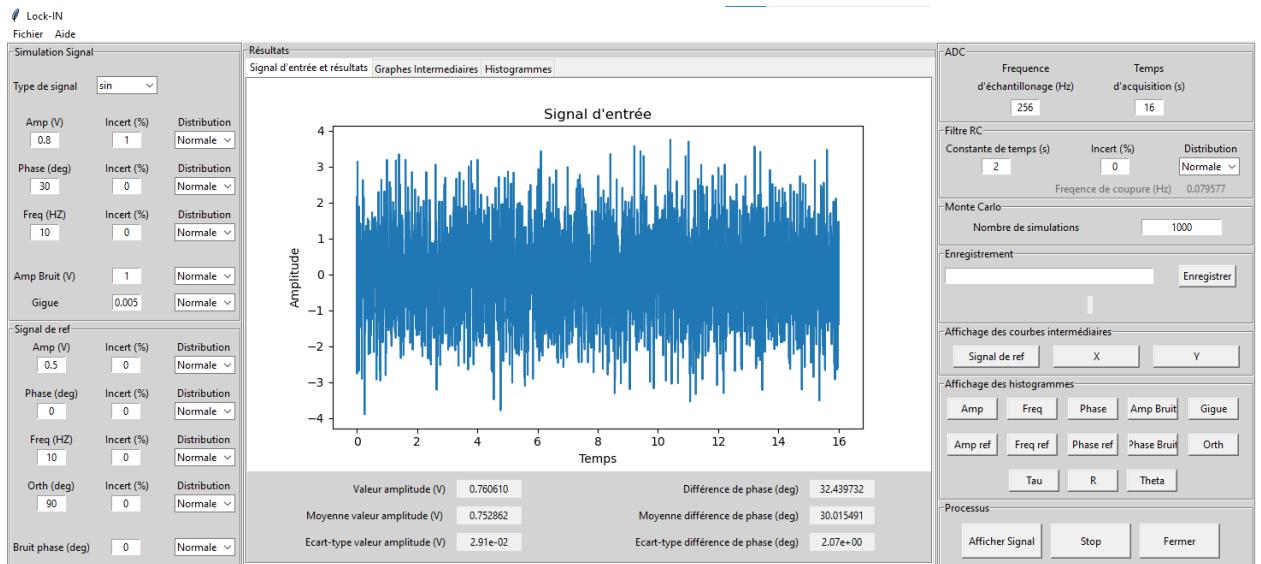


FIGURE 10 – Simulation pour un Instrument parfait, signal imparfait

Dans notre simulation la création du bruit se fait par la modification de "Amp bruit" autrement dit l'amplitude du bruit qui crée un bruit aléatoire d'une amplitude maximale à celle de la donnée choisie.

Un bruit supplémentaire peut être ajouté au travers de la Gigue qui est une variation temporelle aléatoire entraînant des modifications de phase, fréquence et période.

Nous pouvons constater que pour un signal imparfait la simulation génère un résultat moins fidèle que pour un signal parfait.

Aucune simulation de l'instrument imparfait n'a été faite, pour cela il aurait fallu pouvoir quantifier les données selon un certain nombre de bit, nous n'avons donc pas d'incertitude sur la quantification de ces données. De plus l'application de l'instrument imparfait de l'article sur lequel se base ce bureau d'étude prend racine dans une application à la mesure de radiation par une source infrarouge, application que nous n'avons pas effectuée ici.

4.2 Validation avec Kramers-Kronig

Les relations de Kramers-Kronig permettent de relier la partie réelle et imaginaire d'une fonction complexe qui, dans notre cas, est la fonction de transfert d'un circuit RC.

Ces relations ne sont applicables que sur un système qui est causal, invariant dans le temps, linéaire et stable.

On peut donc les écrire tels quels :

$$u(\omega) = \frac{1}{\pi} P \int_{-\infty}^{\infty} \frac{v(\omega')}{\omega - \omega'} d\omega' \quad (11)$$

$$v(\omega) = -\frac{1}{\pi} P \int_{-\infty}^{\infty} \frac{u(\omega')}{\omega - \omega'} d\omega' \quad (12)$$

Dans le cadre de notre simulation d'un Lock-In Amplifier, les relations de Kramers-Kronig nous seront utiles afin de démontrer la véracité de notre méthode de programmation du Lock-In.

En effet à l'aide d'un Lock-In Amplifier, d'un GBF et d'un circuit RC nous avons la possibilité de mettre ces relations en lumière. L'objectif ici, est de comparer les résultats de notre simulation à ceux obtenus par des mesures concrètes effectuées en 2023 au cours d'un projet de L3 physique visant à démontrer ces relations dans le circuit RC.

Avant de s'avancer aux résultats il reste tout de même intéressant de savoir à quoi s'attendre. Le circuit RC est un des circuits les plus basiques en électronique et l'intérêt de l'étudier ici est qu'il possède une fonction de transfert d'ordre 1, ce qui reste plus simple à démontrer mathématiquement qu'une fonction de transfert d'ordre supérieur.

Étudions alors son comportement théorique, nous pouvons alors donner la fonction de transfert suivante :

$$Z(w) = \frac{1}{1 + jwRC} = \begin{cases} ReZ(w) = \frac{1}{1 + (wRC)^2} \\ ImZ(w) = -\frac{jwRC}{1 + (wRC)^2} \end{cases} \quad (13)$$

Dès lors, nous pouvons tracer les graphes de la partie réelle et imaginaire :

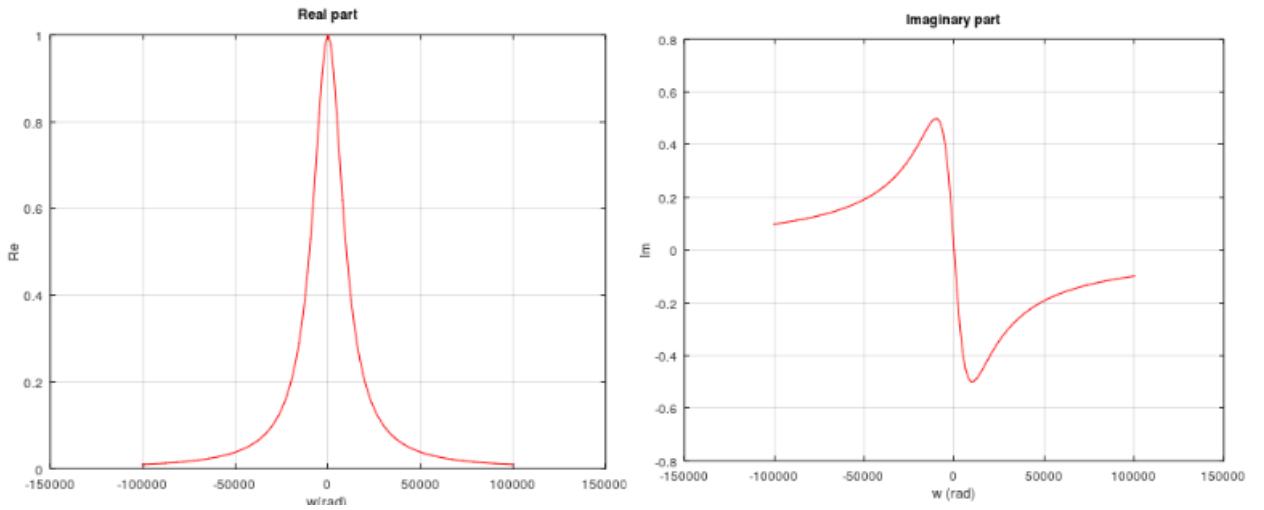


FIGURE 11 – Partie réelle et imaginaire de la fonction de transfert du circuit RC

Nous connaissons maintenant le résultat à retrouver à travers cette simulation. Une chose importante à savoir est que, théoriquement, en-dessous d'une fréquence de 0 hertz, la représentation négative de la partie imaginaire et réelle n'est physiquement pas vraie ce qui permet d'expliquer les résultats simulés suivants :

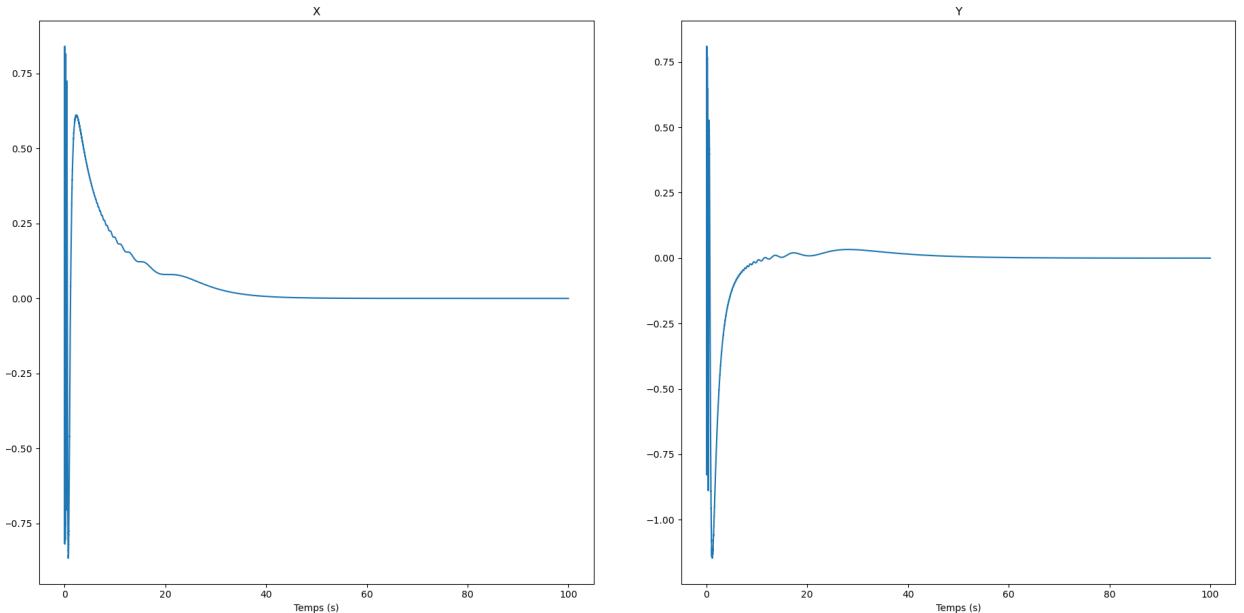


FIGURE 12 – Résultats du programme de simulation

Nous constatons bien une allure des courbes similaires aux courbes théoriques de la fonction de transfert du circuit RC.

Comparons ces résultats aux résultats obtenus par traitement numérique des mesures effectuées avec l'instrument réel :

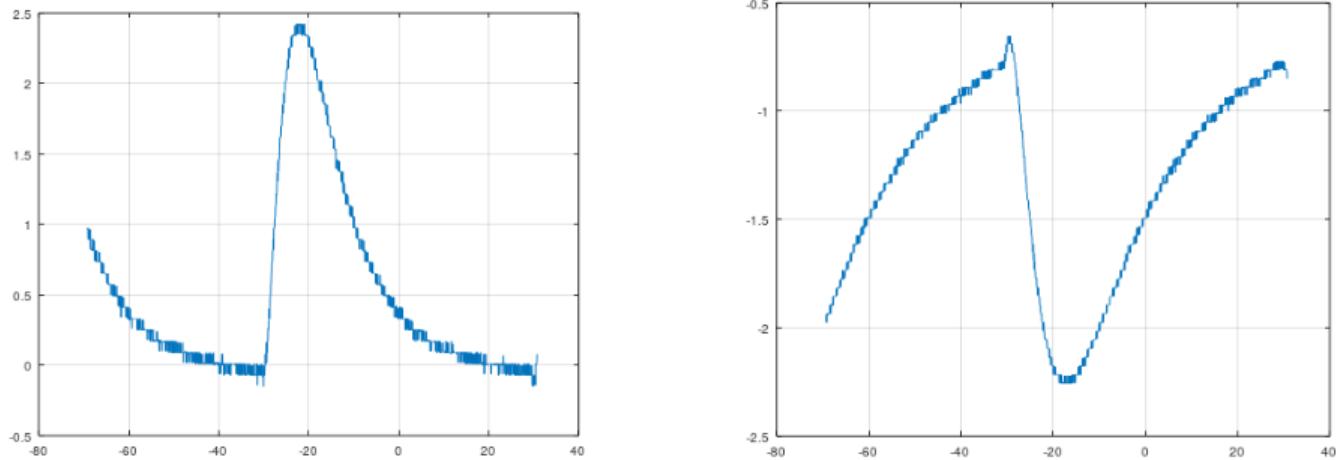


FIGURE 13 – Résultats de l’expérience réelle

Nous pouvons ici distinguer la même allure des courbes, que ce soit pour la partie réelle ou imaginaire.

Par un simple traitement numérique il est possible de reconstituer la fonction de transfert en suivant ces règles :

- * La partie réelle étant paire, sa contrepartie aux fréquences négatives est donc symétrique par rapport à l’axe des ordonnées.
- * La partie imaginaire étant impaire, sa contrepartie aux fréquences négatives est donc inversement symétrique par rapport à l’axe des ordonnées.

Cela nous permet donc d’obtenir les graphes suivants :

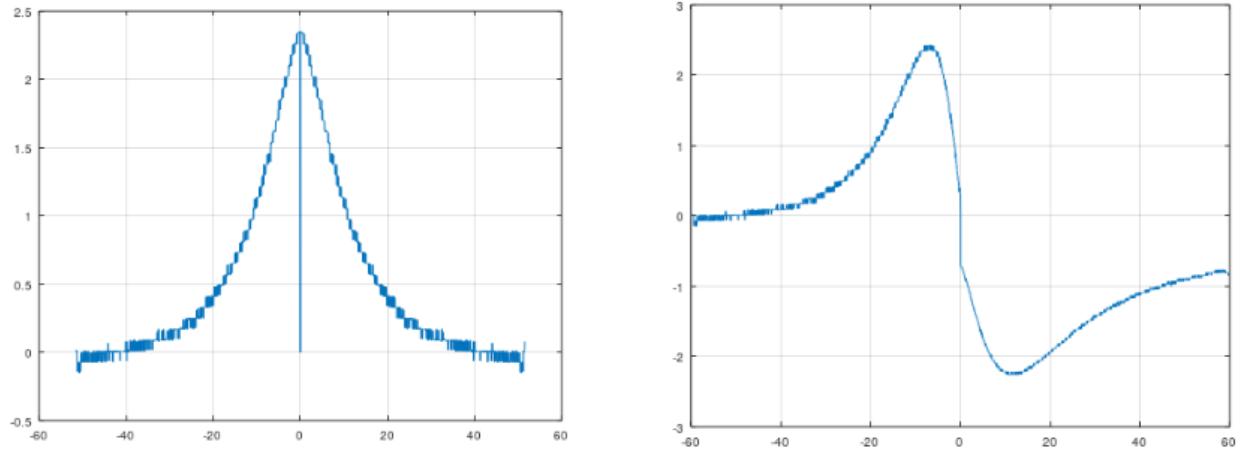


FIGURE 14 – Données traitées pour la partie imaginaire et réelle

Nous arrivons donc bien, en comparant avec notre méthode de programmation, aux résultats obtenus par ce projet de L3 à valider le fonctionnement de notre méthode de simulation du Lock-In Amplifier.

Au-delà de prouver la véracité de nos résultats, la possibilité d’utiliser les relations

de Kramers-Kronig au travers d'un outil de simulation permettrait, dans le cas de circuits électriques, de savoir rapidement retrouver les graphes de la fonction de transfert de n'importe quel système sans avoir à le réaliser.

Intégrer Kramers-Kronig à l'outil de simulation actuel serait donc un avantage certain pouvant prouver son utilité pour des cas concrets.

5 Compléments et perspectives

Sont consignés dans cette section les ajouts potentiels et les idées qui n'ont pas été traités en priorité durant ce projet et pourraient faire l'objet d'un travail ultérieur.

5.1 Concernant l'application

Bien que l'application soit tout à fait fonctionnelle, certaines mises à jours auraient pu (et pourraient) être réalisées pour la rendre plus complète et ergonomique.

- * Le première amélioration consiste à pouvoir exporter le signal généré dans un fichier au format *CSV*. En effet pouvoir faire ce type d'exportation pourrait permettre une analyse de ce signal via un algorithme (*sur Python par exemple*) ou encore de pouvoir programmer un *GBF* avec ce signal.
- * Une seconde amélioration possible serait la lecture de signaux réels. En effet, il est possible, via un oscilloscope, de pouvoir récupérer un signal réel sous le format *CSV*.
- * Comme mentionné dans la section (4.2), afin d'aider l'utilisateur à comprendre les relations de Kramers-Kronig et de valider la programmation du Lock-IN, il serait envisageable d'intégrer, à l'interface, un moyen de pouvoir utiliser cette validation.
- * Ensuite, nous aurions pu prévoir une exportation des figures. La possibilité de les enregistrer permettrait à l'utilisateur de les employer ultérieurement (*Compte-rendu, analyse expérimentale...*).
- * Enfin, nous aurions pu ajouter d'autres lois de distributions ainsi que d'autres types de signaux. Ajouter ces options permettrait à l'utilisateur d'observer l'influence de différentes lois de probabilités ainsi que l'effet des différents types de signaux.
- * L'ajout d'un mode sombre serait aussi envisageable, l'utilisateur pourra donc passer plus de temps à étudier des simulations sans affecter grandement sa fatigue oculaire.

6 Conclusion

Dans le cadre du projet de développement d'un lock-in sur Python, nous avons exploré les principes fondamentaux de cette technique de détection de signaux faibles dans un environnement bruyant. En utilisant les concepts de modulation, de filtrage et de détection synchrone, nous avons conçu et implémenté un système capable d'extraire avec précision des signaux utiles même dans des conditions de bruits élevés.

Notre travail a abouti à la création d'un programme Python fonctionnel et exécutable qui peut être utilisé pour effectuer des mesures de signaux bruités avec une précision relative. Nous avons utilisé des bibliothèques standard telles que *NumPy* pour les calculs numériques et *tKinter* pour la création de l'interface, ainsi que *Matplotlib* pour la visualisation des données.

En réalisant ce projet, nous avons acquis une compréhension approfondie des concepts théoriques derrière le Lock-In Amplifier et avons mis en pratique nos connaissances en les appliquant à un cas d'utilisation concret. Nous avons également développé nos compétences en programmation Python, en particulier dans le domaine du traitement de signal et de la manipulation de données.

En conclusion, ce projet nous a permis d'acquérir de nouvelles compétences en programmation et connaissances en traitement du signal, tout en nous fournissant une expérience pratique précieuse dans la conception et l'implémentation d'un systèmes de détection de signaux basés sur le Lock-In Amplifier.

7 Bibliographie

- [1] P. Clarkson, T. J. Esward, A. A. Smith P. M. Harris, and I. M. Smith. Software simulation of a lock-in amplifier with application to the evaluation of uncertainties in real measuring systems. *Measurement Science and Technology*, 21(4), 3 Mars 2010. URL <https://iopscience.iop.org/article/10.1088/0957-0233/21/4/045106>.
- [2] Zurich Instrument. Applications. 2024. URL <https://www.zhinst.com/europe/en>.
- [3] Benjamin ANDERSON. Statorial, 2024. URL <https://statorial.org/distribution-bimodale/>. Qu'est-ce qu'une distribution bimodale ?

A Annexe

A.1 Les lois de Distributions

A.1.1 Distribution Gaussienne (Normale)

Une distribution Gaussienne est une loi de probabilités continues. Elle dépend de deux paramètres : son espérance noté μ et son écart-type noté σ . La densité de probabilité d'une telle loi s'écrit de la façon suivante :

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \quad (14)$$

On obtient alors le tracé suivant :

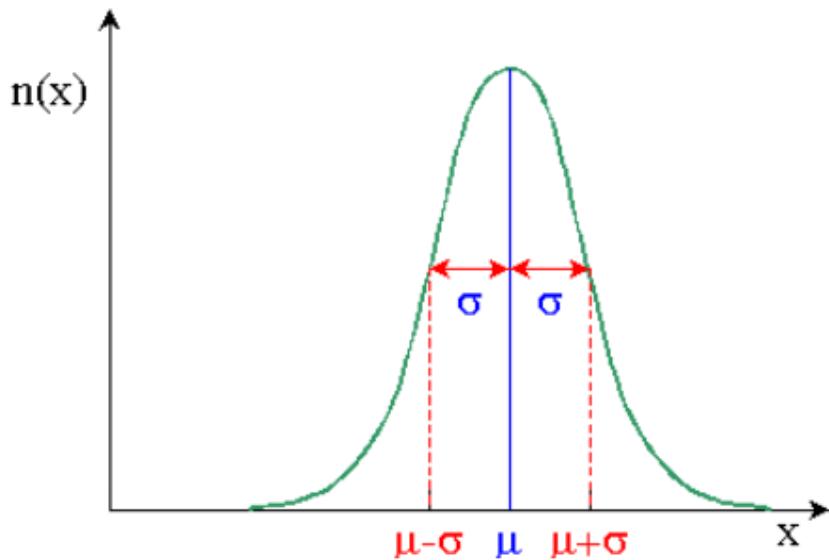


FIGURE 15 – Tracé Loi Gaussienne

Il vient alors une pensée, la majorité des valeurs tirées seront très proches de la moyenne. Plus la valeur se trouve loin de cette moyenne, et plus sa probabilité d'apparition est faible. Pour revenir à notre application, l'incertitude exprimée en pourcent correspond à l'écart-type.

A.1.2 Distribution Uniforme

Une distribution Uniforme peut également être une loi de probabilités continues. Cette loi prend deux paramètres : le plus petite valeur qu'elle peut avoir, noté a et la plus haute qu'elle peut fournir, noté b . Ainsi, elle peut s'écrire selon l'équation suivante :

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{pour } a \leq x \leq b \\ 0 & \text{pour } x \end{cases} \quad (15)$$

Ce qui nous donne la distribution suivante :

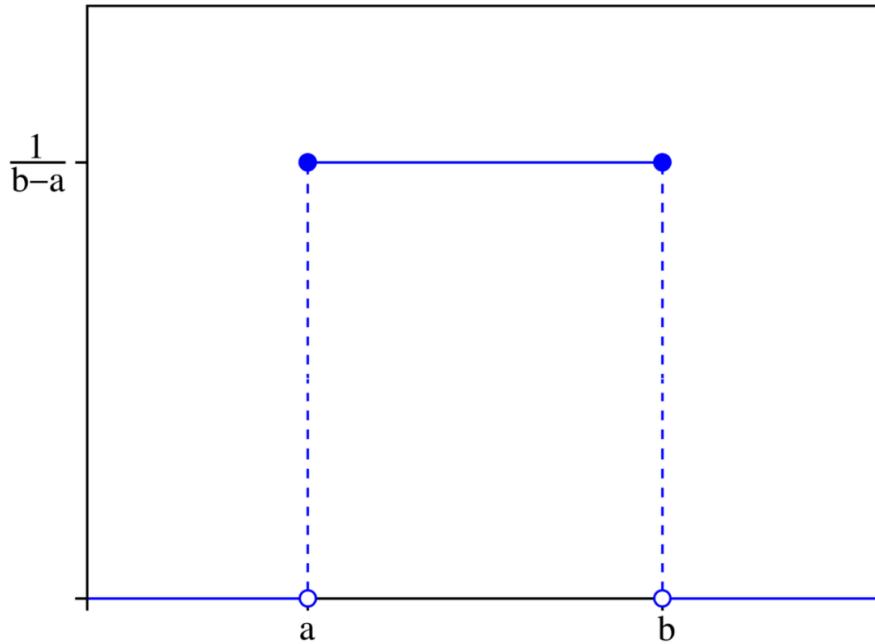


FIGURE 16 – Tracé Loi Uniforme

Ainsi, toutes les valeurs présentent dans l'intervalle $[a, b]$ ont la même chance d'apparaître, on appelle cette notion l'équiprobabilité. Les valeurs en dehors, elles, n'ont aucune chance d'apparaître.

A.2 Modifications pour utilisateur avancé

Il est possible pour l'utilisateur d'ajouter de nouveaux types de signaux ou encore de nouvelles lois de probabilités. Pour cela il est nécessaire de disposer du script de l'application, ces modifications seront à faire directement sur celui-ci. Après l'avoir modifié, l'utilisateur sera dans l'obligation de créer un nouvel exécutable pour que ces modifications soient effectives.

Les modifications à effectuer seront décrites ci-dessous :

A.2.1 Ajouter de nouveaux signaux

Afin d'ajouter de nouveaux signaux, l'utilisateur devra trouver la ligne de commande suivante :

```
1 signal_type = ["sin", "carre"]
```

Après l'avoir identifié (*ligne 774 dans notre script*) il suffit, à la suite de "carre" d'ajouter une nouvelle forme, comme l'exemple suivant :

```
1 signal_type = ["sin", "carre", "cos"]
```

Suite à ça, l'utilisateur sera dans l'obligation de chercher la fonction "*"forme_signal"*" et d'ajouter la condition de sélection du signal à la suite de ceux déjà présent (*et avant la fonction "return"*), comme suit :

```

1 if signal_shape == "cos":
2     signal_entree = amplitude_var * np.cos(2 * np.pi * frequence_var * (temps +
3     gigue_var) + np.radians(phase_var)) + amplitude_bruit_var
4     signal_ref = amplitude_ref_var * np.cos(2 * np.pi * frequence_ref_var *
5     temps + np.radians(phase_ref_var) + np.radians(phase_bruit_ref_var))
6     signal_ref_perp = amplitude_ref_var*np.cos(2 * np.pi * frequence_ref_var *
7     temps + np.radians(phase_ref_var) + np.radians(phase_bruit_ref_var) + np.
8     radians(ortho_var))

```

Suite à ces étapes, il faudra transformer le script en logiciel.

A.2.2 Ajouter de nouvelles distributions

Pour ajouter de nouvelles lois de distributions, il faut réaliser la même procédure. Néanmoins, ici il est important de d'ajouter la loi de distribution à chaque paramètres, ainsi il faut trouver dans le logiciel commençant par "*"distribution_*" et finissant par "*" = ["Normale", "Uniforme"]*" comme à la ligne 796 :

```
1 distribution_amp = ["Normale", "Uniforme"]
```

Il faudra ainsi ajouter notre nouvelle loi à chacune des lignes présentant ces caractéristiques. Suite à ça, il faudra ajouter aussi (*une seule fois*) la condition de sélection dans la fonction "*"generer_valeurs_aleatoires"*" en suivant l'exemple des autres lois dans cette fonction, il est facile d'ajouter des lois. Attention toutefois aux lois prenant d'autres paramètres d'entrées.

A.2.3 L'exécutable

Après avoir apporté l'une, l'autre ou les deux types de modification à mon script à travers un interpréte Python, si un exécutable est souhaité, il faut ouvrir un interpréte de commande (*Windows + R puis taper cmd dans la barre de recherche*) et exécuter la commande suivante :

```
1 pyinstaller --onefile --hidden-import=numpy --hidden-import=matplotlib --hidden-
import=tkinter --hidden-import=os --hidden-import=json --hidden-import=time --
hidden-import=sys --hidden-import=PyMuPDF --hidden-import=pillow Scriptv3.8.py
```

Cette commande va compiler les librairies requises et le Script pour former un exécutable Windows. Pour ce faire il est requis d'avoir les librairies nécessaires sur son ordinateur et d'ouvrir un environnement virtuel au moyen de la commande *activate* dans le dossier où les librairies sont présentes ; le script doit l'être aussi.

A.3 Document de spécification



Version 1.0, créé le 10/11/2023

Adrien Siguier, Evan Lorvellec, Guillaume Le Ruyet

Client : M.Cafarelli

DOCUMENT DE SPÉCIFICATION

Fonctionnalités du logiciel :

- Simuler et afficher les signaux d'entrée et de référence et les signaux en sortie de filtre X et Y.
- Simulateur du fonctionnement du Lock-In-Amplifier.
Afficher les résultats de R et Theta..
- Calculer les incertitudes sur la phase (Thêta) et sur l'amplitude (R) par la méthode de Monte Carlo.

Afficher les histogrammes suivants :

- Amplitude
- Fréquence
- Phase
- Amplitude du bruit
- Gigue
- Amplitude de référence
- Fréquence de référence
- Phase de référence
- Phase du bruit
- Orthogonalité
- Tau
- R
- Thêta.

Afficher les moyennes et écarts-types de l'ensemble des paramètres.

- Enregistrer et ouvrir des presets de données.
- Possibilité de vider l'interface.

Réglages ajustables par l'utilisateur :

- Signal d'entrée :
 - Amplitude.
 - Phase.
 - Fréquence.
 - Amplitude du bruit.
 - Gigue.
- Signal de référence :
 - Amplitude.
 - Phase.
 - Fréquence.
 - Orthogonalité.
- Convertisseur Analogique Numérique :
 - Fréquence d'échantillonnage.
 - Temps d'acquisition.
- Monte Carlo :
 - Nombre de simulations

Les messages d'alertes :

- Concernant l'enregistrement :

Si l'utilisateur oublie d'entrer un paramètre.

- Concernant la constante de temps :

Si l'utilisateur définit une constante de temps pour le filtre RC plus grande que la moitié du temps total d'acquisition.

A.4 Document de conception

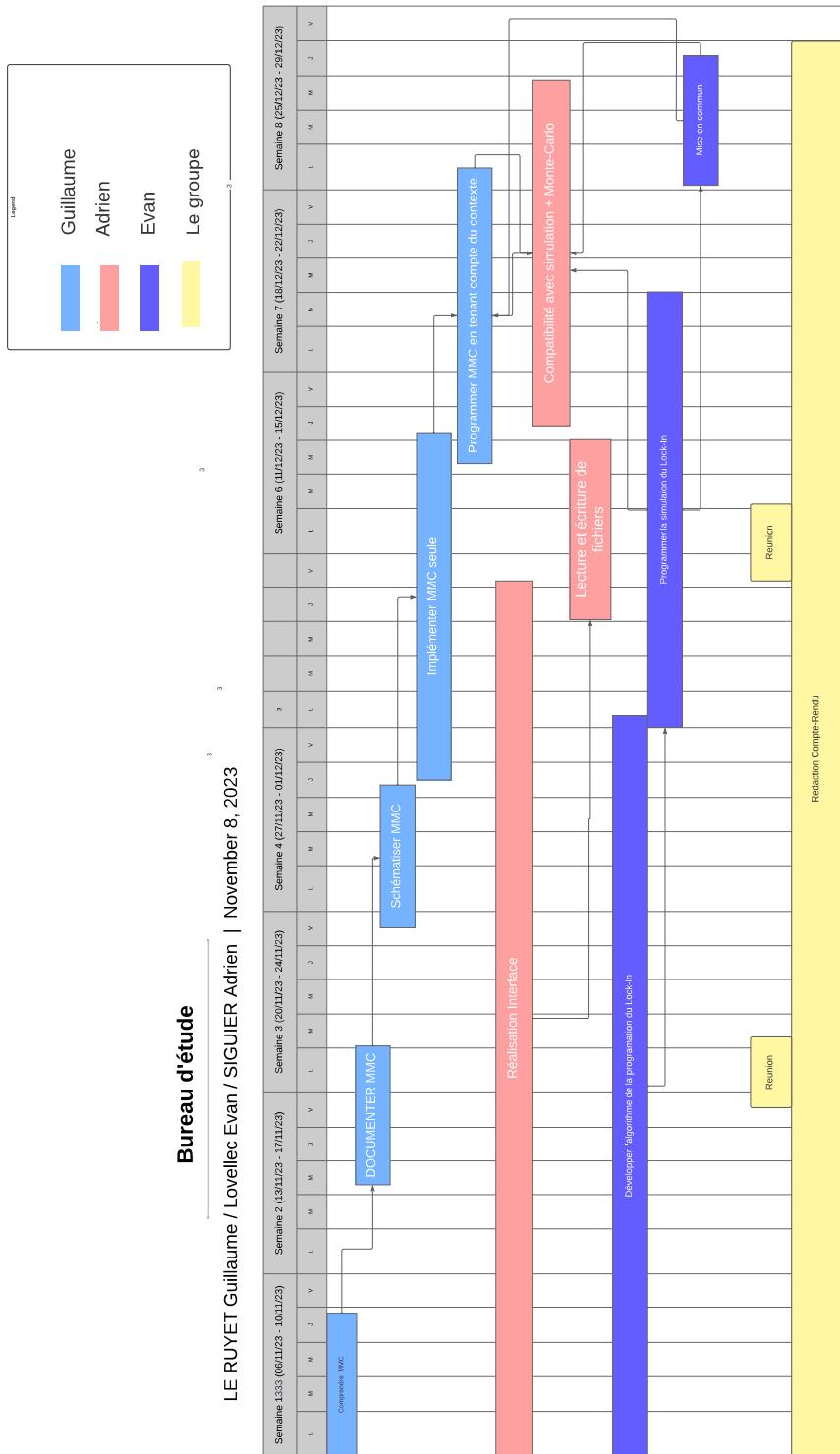
**BUREAU D'ETUDE
IDIM**



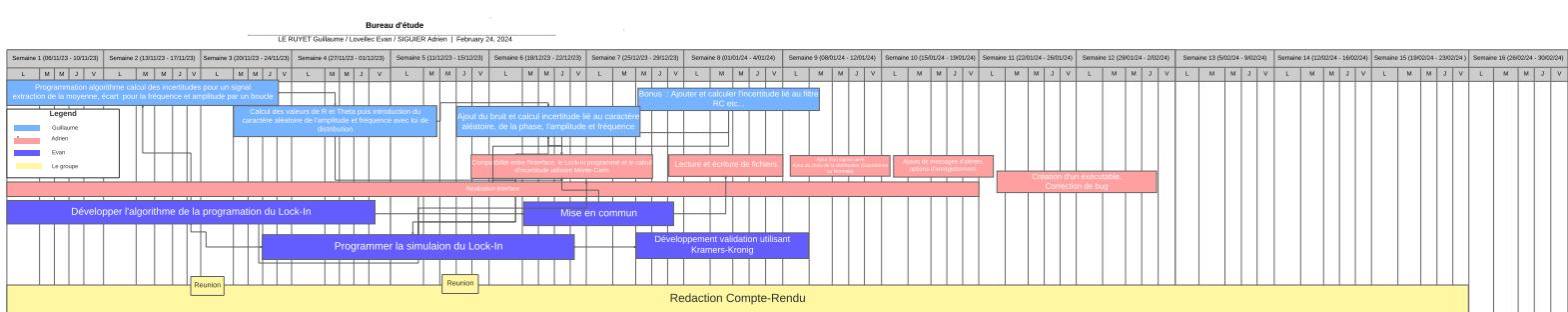
DOCUMENT DE CONCEPTION	Nom du produit	Simulation Python d'un lock-in avec calcul des incertitudes
	Début du projet	06/11/2023
	Fin du projet (prévue)	25 février 2024
Objectif		
Membres du groupe	Adrien SIGUIER (AS)	Simuler l'interface Homme-Machine Chef de projet
	Evan LORVELLEC (EL)	Simuler le fonctionnement du Lock-In
	Guillaume LE RUYET (GLR)	Programmer la méthode de Monté Carlo adapté au lock-in

	Sujet	État	Commentaire
1	* Lire et comprendre l'article fourni	Terminé	A reprendre pour valider le cahier des charges
2	* Familiarisation avec le Lock-In	Terminé	Travaux pratiques /manipulations
3	* Révisions des bases du traitement du signal	Terminé	Déphasage, mutiplication de signaux, RMS, TRMS, Puisance etc..
4	GLR Comprendre et savoir programmer la méthode de Monté Carlo (de l'article)	Terminé	Différencier moyenne temporelle pour chaque signaux créé et moyenne total. Comprendre et se rapprocher des valeurs
5	GLR Créer l'algorithme de la méthode	Terminé	Reprendre le travail de Evan et adapter avec une boucle for et/ou while pour la génération des M ~10^6 signaux attendu pour se rapprocher des valeurs significatives
6	EL Créer l'algorithme de programmation du fonctionnement du Lock-In	Terminé	Programmation du filtre grâce à la fonction de récurrence issu de l'équation différentielle du circuit
7	AS + GLR Évaluer l'incertitude des sources	Terminé	Il reste à déterminer ce qu'on souhaite avoir comme valeur d'entré avec une incrtitude
8	* Identifier les librairies et fonctions utile	Terminé	numpy, matplotlib, ...
9	AS Définir les variables/grandeurs d'entrées, de sortie	Terminé	Cas idéal seule l'amplitude, la fréquence varie sans incertitude
10	AS Associer chaque grandeur d'entré à une variable, définir les variables	Terminé	
11	EL Simuler un signal numérique	Terminé	
12	EL Simuler un bruit (au moins pour le test)	Terminé	
13	AS Se mettre d'accord sur le type de distribution	Terminé	Varie selon la grandeur d'entrée ou pourra être choisi par l'utilisateur à determiner dans le cas non idéal
14	GLR Trouver les fonctions associé à chaque distribution	Terminé	Fonction randn pour la loi normale ou fonction rand pour une distribution uniforme
15	AS Réaliser une interface Homme-Machine	Terminé	Saisir un certains nombres de valeurs d'entrées avec leur incertitudes
16	AS Rendre compatible l'interface avec la simulation Lock-IN et Monte-Carlo	Terminé	Valeurs d'entrées, graphiques, valeurs de sorties
17	AS Lecture / Ecriture de fichiers sur interface	Terminé	Lire des données présentes sur un fichier .txt / Sortir des données d'un signal générée.
18	* Compatibilité entre générations du signal et Monte-Carlo	Terminé	Création de M signaux par une boucle while et/ou for
19	AS Rendre compatible l'interface avec la simulation Lock-IN	Terminé	
20	AS Génération de différents signaux	Terminé	Possibilité pour l'utilisateur de sélectionner la forme du signal (sinus ou carré)
21	AS Génération de différentes distributions	Terminé	Possibilité pour l'utilisateur de sélectionner un type de distribution (Normale ou Uniforme)
22	AS Transformation en logiciel	Terminé	Rendre l'application accessible aux utilisateurs en tout genre sans connaissance en langage de programmation.

A.5 Planning Initial



A.6 Planning réalisé



A.7 Compte-Rendu réunion hebdomadaire

Lors de ce Bureau d'Etudes, une seule réunion a été organisée par le client. Cette réunion a été fixée au 4 décembre 2023, voici ce qui a été retenu lors de cette rencontre.

Bilan de réunion

Suite à la réunion ayant eu lieu le lundi 4 décembre, voici un rapport résumant cette réunion. Suite à notre présentation faisant figurer les différents avancements du bureau d'étude, voici un résumé des tâches dès lors accomplie ce jour même, mise en place d'une ébauche d'interface, création et génération de signaux ainsi que traitement de ces signaux via la simulation du LOCK-IN, calculs d'incertitudes. Les prochaines étapes seront de fixer un bug de compatibilité entre les différents appareils (problèmes d'affichages des signaux), de générer un signal à aspect aléatoire, et de fixer les erreurs de scripts associés à Monte Carlo. Ensuite l'intégration de Monte Carlo à l'interface sera effectué, et l'ajout de nouveaux paramètres tels que la génération des signaux de référence ainsi que la constante de temps du filtre RC sera également mise en place. Une étude plus approfondie sur les lois de distributions doit être réalisée.

SIGUIER Adrien
LE RUYET Guillaume
LORVELLEC Evan

A.8 Script Application

Le script de l'application étant d'une taille conséquente, il n'est pas judicieux de l'intégrer dans les annexes, celui-ci étant disponible en parallèle du compte-rendu. Il est important de noter que cette application a été développée avec le langage de développement Python et que pour être exécuté il est nécessaire de disposer de certaines librairies installées sur l'ordinateur (*liste ci-dessous*).

```
— numpy
— matplotlib
— tkinter
— os
— json
— time
— sys
— PyMuPDF
— pillow
```

Remarque : Ces librairies sont requises pour l'exécution du script en utilisant un interpréteur Python, il est à noté que pour l'utilisation de l'application (format .exe) il n'est pas nécessaire d'avoir ces librairies Python sur son ordinateur.

A.9 Script Python Kramers_Kronig

Le Script suivant permet de démontrer les équations de Kramers-Kronig. Pour tester ce script il suffit de le copier / coller dans un environnement Python, aucune librairie supplémentaire n'est demandée pour l'exécution de ce script. Ce script est également fourni avec le script de l'application et son exécutable.

```
1 """
2 Cree le 19/12/2023 par LE RUYET Guillaume, LORVELLEC Evan, SIGUIER Adrien
3
4 Version 3
5
6 But : 1. Modeliser la generation de signal sinusoïdale ou SWEEP.
7       2. Modeliser fonctionnement d'un Lock-In Amplifier.
8       3. Redemontrer Kramers-Kronig en utilisant la fonction SWEEP et un circuit RC
9       externe.
10 Entrées : Amplitude, fréquence et déphasage du signal. Constante de temps du Lock
11 -IN Amplifier.
12 Sorties : Graphes de R, theta, X et Y ainsi que leurs moyennes.
13 """
14 import numpy as np
15 import matplotlib.pyplot as plt
16
17 # Génération d'un signal SWEEP
18 def generate_sweep(amplitude, freq_deb, freq_fin, duree, echantillonnage, dephasage):
19     t      = np.arange(taux, duree, 1/echantillonnage)
20     freq   = np.linspace(freq_deb, freq_fin, len(t))
21     signal = amplitude * np.sin(2 * np.pi * freq * t + dephasage)
```

```

22     return signal
23
24 # Modelisation du circuit RC
25 def rc_circuit(v, t, R, C, Vin):
26     tau = R * C
27     dvdt = (Vin - v) / tau
28     return dvdt
29
30 # Fonction pour concevoir un filtre passe-bas RC
31 def rc(signal, fc, fs):
32     dt = 1 / fs
33     alpha = dt / (1/(2*np.pi*fc) + dt)
34     signal_filtre = np.zeros_like(signal)
35     for i in range(1, len(signal)):
36         signal_filtre[i] = alpha * signal[i] + (1 - alpha) * signal_filtre[i - 1]
37     return signal_filtre
38
39 # Parametres du signal d'entree
40 amplitude = float(input("Entrer l'amplitude du signal A0 : "))
41 taux = float(input("Entrer la constante de temps : "))
42 # frequence = float(input("Entrer la frequence du signal f0 : "))
43 phase = float(input("Entrer la phase du signal phi : "))
44
45 # Parametres du Lock-In Amplifier
46 R2 = 0.01
47 fc = 1/R2*taux
48 # frequence_ref = frequence
49 phase_ref = phase
50 R = 1.0 # Resistance en ohms
51 C = 1e-3 # Capacite en farads
52
53 # Variables pour le SWEEP
54 frequence_depart = 0 # Frequence de depart en Hz
55 frequence_fin = 16000 # Frequence de fin en Hz
56 duree_sweep = 60 # Duree de la sweep en secondes
57 echantillonnage = 1000 # Taux d'echantillonnage en Hz
58 t = np.arange(taux, duree_sweep, 1/echantillonnage) # Axe des temps
59 f = 1/t # Axe des frequences
60
61 # Creation du signal d'entree
62 signal_entree = generate_sweep(amplitude, frequence_depart, frequence_fin,
       duree_sweep, echantillonnage, 0)
63
64 # Simulation du circuit RC
65 signal_sortie = np.zeros_like(signal_entree)
66 amplitude_sortie_filtre = 0.0
67
68 for i in range(len(signal_entree) - 1):
69     dt = 1/echantillonnage
70     amplitude_entree = signal_entree[i]
71     amplitude_sortie_filtre = amplitude_sortie_filtre + rc_circuit(
72         amplitude_sortie_filtre, i*dt, R, C, amplitude_entree) * dt
73     signal_sortie[i + 1] = amplitude_sortie_filtre
74
75 # Creation du signal de reference pour le Lock-In Amplifier non dephase et dephase
76 signal_ref = generate_sweep(1, frequence_depart, frequence_fin, duree_sweep,
    echantillonnage, 0)
77 signal_ref_perp = generate_sweep(1, frequence_depart, frequence_fin, duree_sweep,
    echantillonnage, 1.571)
78
79 # Mixage du signal d'entree avec le signal de reference
80 signal_mixe = signal_ref * signal_sortie
81 signal_mixe_perp = signal_ref_perp * signal_sortie
82
83 # Filtre passe-bas RC pour extraire la composante a la frequence de reference
84 composante_verrouillee = rc(signal_mixe, fc, 10000)
85 composante_verrouillee2 = rc(signal_mixe_perp, fc, 10000)
86
87 # Creation de R et Theta

```

```

87 R      = 2*np.sqrt(composante_verrouillee**2 + composante_verrouillee2**2)
88 theta = np.arctan2(composante_verrouillee2, composante_verrouillee)
89
90 # Conversion des radians en degrés
91 theta_deg = np.degrees(theta)
92
93 # Affichage des signaux
94 plt.figure(figsize=(10, 6))
95
96 plt.figure(1)
97 plt.subplot(1, 2, 1)
98 plt.plot(f, signal_entree, label='Signal d\'entrée')
99 plt.title('Signal d\'entrée')
100 plt.xlabel('Temps (s)')
101 plt.legend()
102
103 plt.subplot(1, 2, 2)
104 plt.plot(f, signal_ref, label='Signal de référence')
105 plt.title('Signal de référence')
106 plt.xlabel('Temps (s)')
107 plt.legend()
108
109 plt.figure(2)
110 plt.subplot(1, 2, 1)
111 plt.plot(f, R, label='Composante verrouillée')
112 plt.title('R')
113 plt.xlabel('Temps (s)')
114 plt.legend()
115
116 plt.subplot(1, 2, 2)
117 plt.plot(f, theta, label='Composante verrouillée 2')
118 plt.title('Theta')
119 plt.xlabel('Temps (s)')
120 plt.legend()
121
122 plt.figure(3)
123 plt.subplot(1, 2, 1)
124 plt.plot(f, composante_verrouillee, label='Composante verrouillée')
125 plt.title('X')
126 plt.xlabel('Temps (s)')
127
128 plt.subplot(1, 2, 2)
129 plt.plot(f, composante_verrouillee2, label='Composante verrouillée 2')
130 plt.title('Y')
131 plt.xlabel('Temps (s)')
132
133 plt.tight_layout()
134 plt.show()
135
136 moyenne_R      = np.mean(R)
137 moyenne_theta = np.mean(theta)
138
139 print("Moyenne de R:", moyenne_R)
140 print("Moyenne de Theta:", moyenne_theta)

```