UMONS
Université de Mons

Faculté Polytechnique

Développement d'un outil pédagogique de simulation pour l'enseignement des systèmes de télécommunications numériques en bande de base exploitant le language de programmation Python

Rapport de projet
présenté dans le cadre du projet d'électricité de BAB3

Denis ALBRECQ
Tom FONTAINE

POLYTECH MONS

Sous la direction de :
Madame Véronique MOEYAERT (promotrice)
Monsieur Alexander GROS (co-promoteur)
Monsieur Thierry DUTOIT (rapporteur)

Année académique 2023-2024

pôle hainuyer

Abstract

This document presents a comprehensive study on the validation and testing of a simulation tool dedicated to the analysis of digital line coding in the telecommunications field. The primary objective is to create a tool for students in telecommunications courses that allows for the simulation of key concepts, thereby facilitating their understanding. It is necessary to evaluate the concordance between the results obtained through simulation and theoretical predictions for various types of coding, including NRZ, RZ, Manchester, and AMI. This comparison aims to confirm the reliability of the simulation tool while identifying any potential discrepancies relative to established theoretical models. The in-depth analysis focuses particularly on the power spectral density, intersymbol interference induced by the transmission channel, and the impact of noise on signal quality. It has been demonstrated that the tool can effectively simulate key baseband transmission concepts. Moreover, the study explores potential improvements to the simulation tool, including the integration of new types of coding such as M-level codes or duo-binary code, as well as the addition of advanced functionalities such as symbol regeneration. Collaboration with other fields, such as control and system optimization, is also considered to better manage the loop in the PLL clock recovery system. Finally, the integration of more advanced modulation techniques such as QAM modulation is suggested to broaden the range of simulation scenarios and deepen the study of the tool's performance. This work thus makes a significant contribution to the understanding of digital communication systems and the enhancement of simulation methods in this constantly evolving field.

Résumé

Ce document présente une étude approfondie sur la validation et le test d'un outil de simulation dédié à l'analyse du codage numérique dans le domaine des télécommunications. L'objectif principal est de créer un outil pour les étudiants en cours de télécommunications qui permet de simuler des concepts clés, facilitant ainsi leur compréhension. Il est nécessaire d'évaluer la concordance entre les résultats obtenus par simulation et les prédictions théoriques pour différents types de codage, y compris NRZ, RZ, Manchester et AMI. Cette comparaison vise à confirmer la fiabilité de l'outil de simulation tout en identifiant les éventuelles divergences par rapport aux modèles théoriques établis. L'analyse approfondie se concentre particulièrement sur la densité spectrale de puissance, l'interférence entre symboles induite par le canal de transmission et l'impact du bruit sur la qualité du signal. Il a été démontré que l'outil peut simuler efficacement les concepts clés de la transmission en bande de base. De plus, l'étude explore des améliorations potentielles de l'outil de simulation, y compris l'intégration de nouveaux types de codage tels que les codes à M niveaux ou le code duo-binaire, ainsi que l'ajout de fonctionnalités avancées comme la régénération de symboles. La collaboration avec d'autres domaines, tels que le contrôle et l'optimisation des systèmes, est également envisagée pour mieux gérer la boucle dans le système de récupération d'horloge par PLL. Enfin, l'intégration de techniques de modulation plus avancées telles que la modulation QAM est suggérée pour élargir la gamme des scénarios de simulation et approfondir l'étude des performances de l'outil. Ce travail apporte ainsi une contribution significative à la compréhension des systèmes de communication numérique et à l'amélioration des méthodes de simulation dans ce domaine en constante évolution.

Table des matières

Table des figures IV
Liste des tableaux VI
Liste des abréviations VIII

Introduction **1**

Cadre théorique **2**
0.1 Source 2
0.2 Codage de canal 2
0.3 Modélisation du canal et interférence entre symboles (Inter Symbol Interference) 5
0.4 Filtrage de Nyquist 6
0.5 Diagramme de l'oeil 7

1 Cahier des charges **8**

2 État de l'Art **10**

3 Conception et Architecture **12**
3.1 Choix des illustrations 12
3.2 Conception du signal 13
3.3 Représentation du signal 14
3.4 Conception du canal 14
3.5 Conception du filtre de Nyquist 16
3.6 Conception du diagramme de l'oeil 17
3.7 Choix des paramètres de simulation 17
3.8 Architecture de l'outil 18

4 Implémentation **20**
4.1 Structure du script 20
4.2 Créer une Séquence Binaire Pseudo-Aléatoire (SBPA) 20
4.2.1 Vérification de la Séquence Binaire Pseudo-Aléatoire (SBPA) . . . 21
4.3 Représentations spectrales du signal 22
4.4 Codage du signal 22
4.5 Appliquer le filtre de Nyquist 22
4.6 Conception d'outils dynamiques 22
4.6.1 Conception de l'animation 22
4.7 Problèmes rencontrés 23

5 Validation et Tests **24**

6 Feedbacks **28**

7 Perspectives **29**

Conclusion **30**

Annexes **33**
A.1 Figures 33
A.2 Tableaux 43
A.3 Documents 51
A.3.1 Google Forms 51
A.3.2 Notice d'utilisation 53
A.3.3 Commentaires sur le simulateur de transmissions en bande de base . 56
A.4 Script 59
A.4.1 Installation des librairies requises 59
A.4.2 Transmission d'un signal binaire pseudo aleatoire 59
A.4.3 Animation de l'oeil de Nyquist 78
A.4.4 Verification SBPA 80

Table des figures

0.1 Modèle d'une transmission numérique en bande de base [13] 2
0.2 Influence du canal sur un symbole [13] 5
0.3 Exemple d'interférence entre symboles (Inter Symbol Interference) [13] . . 6
0.4 Filtre de Nyquist pour différents α [13] 6
0.5 Exemple de diagramme de l'oeil avec le code Non Return to Zero polaire [13] 7

3.1 Fonction caractéristique du filtre passe-bas de Butterworth [3] 14
3.2 Pôles et zéros, gain et délai de groupe de l'approximation de Butterworth d'un filtre passe-bas normalisé [3] 15
3.3 Statistique de tension de bruit thermique [13] 16
3.4 Page d'accueil - Transmission of a pseudo-random binary sequence 19

4.1 Autocorrélation d'une Séquence Binaire Pseudo-Aléatoire (SBPA) [13] . . 21
4.2 Autocorrélation de la suite binaire avec un débit de 5 [bit/s] 21

5.1 Densité spectrale de puissance pour le code Non Return to Zero (NRZ) polaire en pratique 24
5.2 Densité sprectrale de puissance pour le code Non Return to Zero (NRZ) polaire en théorie [13] 24
5.3 Étalement d'un symbole par passage dans le canal 25
5.4 Diagramme de l'oeil pour le codeNon Return to Zero (NRZ) polaire avec interférence entre symboles (Inter Symbol Interference) (ISI) en pratique . 25
5.5 Diagramme de l'oeil pour Non Return to Zero (NRZ) polaire avec interférence entre symboles (Inter Symbol Interference) (ISI) en théorie [13] . . . 25
5.6 Filtre de Nyquist en pratique pour $\alpha = 0.2$ 26
5.7 Filtre de Nyquist en théorie [13] 26
5.8 Diagramme de l'oeil filtré en pratique pour le code Non Return to Zero (NRZ) polaire avec $\alpha = 1$ 26
5.9 Diagramme de l'oeil filtré en théorie pour le code Non Return to Zero (NRZ) polaire avec $\alpha = 1$ [13] 26
5.10 Signal bruité avec rapport signal sur bruit (SNR) de 20dB 26
5.11 Signal non bruité avec rapport signal sur bruit (SNR) de 40dB 26
5.12 Oeil bruité avec rapport signal sur bruit (SNR) de 20dB 27
5.13 Oeil non bruité avec rapport signal sur bruit (SNR) de 40dB 27

A.1 Séquence Binaire Pseudo-Aléatoire [13] 33
A.2 Modélisation du canal 33
A.3 1ère étape de la construction du diagramme de l'oeil 34
A.4 2ème étape de la construction du diagramme de l'oeil 34
A.5 3ème étape de la construction du diagramme de l'oeil 35

A.6 4ème étape de la construction du diagramme de l'oeil 35
A.7 Croquis de l'architecture de l'application de simulation 36
A.8 Fenêtre - Influence of the channel 37
A.9 Fenêtre - Nyquist filter 37
A.10 Fenêtre - Eye pattern of the output signal 38
A.11 Autocorrélation nulle d'une suite binaire avec un débit de 10 [bit/s] 38
A.12 Autocorrélation de la suite binaire avec un débit de 1 [bit/s] 39
A.13 Autocorrélation de la suite binaire avec un débit de 10 [bit/s] 39
A.14 Convolution du signal avec le filtre de Nyquist (Facteur de roll-off = 0.1, $\Delta = 0.3$ s) 40
A.15 Animation du diagramme de l'oeil non réussie 41
A.16 Page d'accueil avec des dimensions non-adéquates 41
A.17 Filtre de Nyquist en pratique pour $\alpha = 0$ 42
A.18 Filtre de Nyquist en pratique pour $\alpha = 0.4$ 42
A.19 Filtre de Nyquist en pratique pour $\alpha = 0.6$ 42
A.20 Filtre de Nyquist en pratique pour $\alpha = 0.8$ 42
A.21 Filtre de Nyquist pour $\alpha = 1$ 42
A.22 Section 3 du cours de Communications Analogiques et Numériques à l'intention des étudiants 56

Liste des tableaux

1 Panel du Non Return to Zero (NRZ) polaire 43
2 Panel du Non Return to Zero (NRZ) unipolaire 44
3 Panel du Non Return to Zero Inverted (NRZI) 45
4 Panel du Return to Zero (RZ) polaire 46
5 Panel du Return to Zero (RZ) unipolaire 47
6 Panel du Manchester 48
7 Panel du Manchester différentiel 49
8 Panel du Alternate Mark Inversion (AMI) 50

Liste des abréviations

AMI Alternate Mark Inversion

AWGN bruit blanc additif gaussien (Additif White Gaussian Noise)

BER taux d'erreur binaire

ISI interférence entre symboles (Inter Symbol Interference)

NRZI Non Return to Zero Inverted

NRZ Non Return to Zero

PLL boucle à verrouillage de phase (Phase-Locked Loop)

RZ Return to Zero

SBPA Séquence Binaire Pseudo-Aléatoire

SNR rapport signal sur bruit

Introduction

Les télécommunications numériques jouent un rôle crucial dans notre société moderne, facilitant la transmission efficace et fiable de l'information à travers de vastes réseaux de communication. Dans ce contexte, la conception et l'analyse des systèmes de communication numérique nécessitent une compréhension approfondie des principes fondamentaux du codage, de la modulation et de la transmission des signaux numériques. Les outils de simulation sont des éléments essentiels dans ce processus, permettant aux ingénieurs et chercheurs de modéliser et d'évaluer le comportement des systèmes de communication dans des conditions variées.

L'objectif de cette étude est de concevoir un outil de simulation dédié à la transmission en bande de base, en mettant l'accent sur la validation et les tests par rapport aux modèles théoriques établis. Notre attention se porte sur plusieurs aspects cruciaux de la communication numérique, notamment les caractéristiques de la densité spectrale de puissance , l'interférence entre symbole induite par le canal de transmission, ainsi que l'impact du bruit sur la qualité du signal.

Le premier volet de cette étude consiste à détailler comment nous avons procédé pour concevoir cet outil de simulation. Partant d'une proposition effectuée par le service de télécommunication, en collabaration étroite avec le corps enseignant et en concertation avec les étudiants des cours associés, nous avons choisi les illustrations les plus pertinentes à simuler. Ensuite, pour concevoir cet outil, il a été nécessaire de s'assurer que notre système correspondait bien à la théorie. Nous expliquons la manière dont nous avons procédé tant pour déterminer le champ d'action, réaliser la conception de l'outil et son implémentation.

La seconde partie compare les résultats obtenus par simulation avec les prédictions théoriques du codage Non-Return-to-Zero (NRZ) illusré en détail dans les cours de téléommunication. Cette comparaison vise à évaluer la fiabilité de l'outil de simulation dans la représentation fidèle des caractéristiques de transmission de ces différents codages, en mettant en lumière les similitudes et les divergences entre les résultats expérimentaux et les modèles théoriques. Nous avons également eu la possibilité de mettre à disposition cet outil pour les étudiants du cours de "Communication Analogique et Numérique".

Enfin, cette étude explore les perspectives d'amélioration de l'outil de simulation, en proposant des pistes pour étendre sa fonctionnalité et sa capacité à modéliser des scénarios de communication numérique plus complexes. Ces évolutions visent à offrir aux ingénieurs et chercheurs un outil plus complet et plus flexible pour l'analyse et la conception de systèmes de communication numérique dans un environnement en constante évolution.

Cadre théorique

Le sujet abordé dans ce projet se consacre à la simulation d'une transmission numérique en bande de base via le langage de programmation Python. Nous modélisons la transmission en bande de base grâce à un modèle vu en cours de "Télécommunications Numériques". Nous nous sommes consacrés à l'émission et à la transmission plutôt qu'à la réception, l'étude s'arrêtera donc après le bloc "Canal" de la figure 0.1 représentant ce modèle.

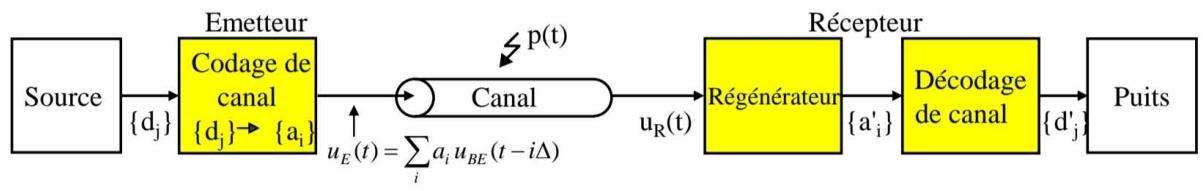


Figure 0.1 – *Modèle d'une transmission numérique en bande de base* [13]

0.1 Source

Pour garder le caractère aléatoire d'un signal lors d'une transmission, il est possible d'utiliser une Séquence Binaire Pseudo-Aléatoire plus communément notée SBPA. Une SBPA s'obtient grâce à une succession de registres à décalage (bistables). Un exemple se trouve à la figure A.1. Ceux-ci permettent de créer une boucle qui, judicieusement choisie, crée une SBPA. L'utilisation de cet outil permet de rester conforme aux hypothèses utilisées concernant la théorie des signaux aléatoires. En effet, cette notion est primordiale pour que les termes tels que la covariance, la corrélation ou encore la densité spectrale de puissance puissent avoir un sens.

0.2 Codage de canal

Pour rappel, le codage de canal transforme une suite binaire en une suite de symboles qui suit des "règles" prédéfinies. Tout au long de notre étude, la forme d'onde élémentaire $u_{BE}(t)$ choisie sera toujours une onde rectangulaire telle que :

$$u_{BE}(t) = A\ rect(\frac{t}{x})\ \text{où}\ x = \begin{cases} \Delta & \textit{Pour les codes NRZ, Manchester et AMI} \\ \frac{\Delta}{2} & \textit{Pour les codes RZ} \end{cases} \tag{1}$$

Voici la liste des différents codes que nous avons utilisés lors de ce projet (et vus au cours de "Télécommunications" de M. Wuilpart et V. Moeyaert) :

1. Non Return to Zero (NRZ) polaire :

$$a_i = \begin{cases} 1 & \text{si } d_i = 1 \\ -1 & \text{si } d_i = 0 \end{cases} \tag{2}$$

   Le codage NRZ polaire change la suite de bits en une suite de symboles valant -1 ou 1 tout en gardant $\Delta = T$ où Δ représente la durée d'un symbole et T la durée d'un bit. Pour ce code, la densité spectrale de puissance $\Phi_e(f)$ est maximum en $f = 0$ et s'annule en $f = 1/T$ ainsi qu'en ses multiples. La bande passante, définie comme l'intervalle entre $f = 0$ et la première annulation de la densité spectrale, est donc égale à $1/T$. Elle ne possède pas de raies car la moyenne des symboles $\mu_a = 0$ (Voir formule de Bennett [1]).

2. NRZ unipolaire :

$$a_i = \begin{cases} 1 & \text{si } d_i = 1 \\ 0 & \text{si } d_i = 0 \end{cases} \tag{3}$$

   Le codage NRZ unipolaire ressemble fortement au précédent. Quelques différences restent à noter comme le fait de n'avoir qu'une seule polarité ainsi que la présence d'une raie en $f = 0$. Ce code possède donc une composante continue puisque μ_a est non nulle et vaut exactement 1/4. Une partie de la puissance est donc gaspillée dans cette composante continue en comparaison avec le code NRZ polaire. Il est alors plus sensible au bruit puisque, à puissance d'émission constante, une plus petite fraction de puissance est consacrée à la transmission du signal.

3. Non Return to Zero Inverted (NRZI) :

$$a_i = \begin{cases} a_{i-1} & \text{si } d_i = 1 \\ -a_{i-1} & \text{si } d_i = 0 \end{cases} \tag{4}$$

   avec $a_i = \{-1; 1\}$

   Le NRZI possède des propriétés analogues au codage NRZ polaire tout en se débarrassant de la sensibilité à la polarité et en garantissant suffisamment de transitions afin d'éviter de trop longues suites de '1'.

4. Return to Zero (RZ) unipolaire :

$$a_i = \begin{cases} 1 & \text{si } d_i = 1 \\ 0 & \text{si } d_i = 0 \end{cases} \tag{5}$$

   mais u_E, le signal émis, est égal à +A ou 0 (A étant la tension du rectangle de la fonction élémentaire de base $u_{BE}(t)$) sur une demi-période Δ avant de revenir à 0.

1. Formule de Bennett :

$$\Phi_e = \frac{|U_{BE}(f)|^2}{\Delta}[C_a(0) + 2\sum_{k=1}^{\infty} C_a(k)\cos(2\pi f k\Delta)] + (\frac{\mu_a}{\Delta})^2 \sum_n |U_{BE}(\frac{n}{\Delta})|^2 \delta(f - \frac{n}{\Delta})$$

Pour une même valeur d'amplitude du signal émis, la puissance est réduite de moitié comparé au code NRZ unipolaire. Étant donné que la moyenne des symboles est non nulle ($\mu_a \neq 0$) et que nous avons choisi une onde élémentaire rectangulaire de durée $\Delta/2$, la densité spectrale de puissance possèdera une raie à tous les multiples impairs de $1/T$ ainsi qu'une composante continue. Cette fonction s'annule aussi en $2/T$ et tous ses multiples. La bande passante est donc double par rapport au code NRZ et vaut $2/T$.

5. RZ polaire :

$$a_i = \begin{cases} 1 & \text{si } d_i = 1 \\ -1 & \text{si } d_i = 0 \end{cases} \tag{6}$$

mais u_E, le signal émis, est égal à +A ou -A (A étant la tension du rectangle de la fonction élémentaire de base $u_{BE}(t)$) sur une demi-période Δ avant de revenir à 0.

Ce code combine deux aspects : la polarité et le fait de revenir à 0 à chaque symbole. Ses propriétes sont donc un mélange entre le code NRZ polaire et le code RZ unipolaire. La densité spectrale de puissance s'annule pour tous les multiples de $2/T$. Elle ne possède pas de raie puisque la moyenne des symboles μ_a est nulle. Pour une même valeur d'amplitude, la puissance totale est égale à la moitié de la puissance totale pour le code NRZ polaire. La bande passante est la même que pour le code RZ unipolaire.

6. Manchester :

$$\begin{cases} a_{2i} = 1,\ a_{2i+1} = -1 & \text{si } d_i = 1 \\ a_{2i} = -1,\ a_{2i+1} = 1 & \text{si } d_i = 0 \end{cases} \tag{7}$$

Pour une même valeur d'amplitude du signal, ce code possède une puissance totale égale à celle du code NRZ polaire mais répartie différemment sur le spectre des fréquences. En effet, la densité spectrale de puissance du code Manchester s'annule en $f = 0$ et en tous les multiples de $2/T$. Elle ne possède pas de raie. Comme pour le RZ, la bande passante est double par rapport au NRZ et vaut aussi $2/T$. Ce code est très utilisé en pratique car il est simple à mettre en oeuvre et garantit une transition à chaque intervalle binaire, ce qui facilite la récupération de l'horloge symbolique à la réception grâce à une boucle à verrouillage de phase.

7. Manchester différentiel : Ce code possède les mêmes propriétés que celui présenté précédemment mais il s'affranchit de la sensibilité à la polarité du canal. Si le bit reçu est '0', alors ça entraîne une transition au début du bit. Tandis qu'un bit reçu à '1' garde la valeur précédente en la laissant inchangée.

8. Alternate Mark Inversion (AMI) :

$$\begin{cases} a_i = 0 & \text{si } d_i = 0 \\ a_i = \begin{cases} 1 \\ -1 \end{cases} & \text{alternativement si } d_i = 1 \end{cases} \tag{8}$$

La densité spectrale de puissance s'annule en $f = 0$ et aux multiples de $1/T$. La bande passante est donc de $1/T$. Cependant, pour une même valeur d'amplitude, la puissance totale obtenue avec ce code est égale à la moitié de celle obtenue avec le NRZ polaire.

0.3 Modélisation du canal et interférence entre symboles (Inter Symbol Interference)

Lors d'une transmission, le canal engendre une atténuation du signal et le récepteur ajoute du bruit thermique et de Schottky [2] reprsenté âr une source de bruit à l'entrée $b(t)$ sur la figure A.2. Le canal sera quant à lui assimilé à un filtre passe-bas. Puisque nous travaillons en bande de base, le signal sera viable dans une gamme de fréquences allant de 0 Hz jusqu'à la fréquence de coupure du canal. Au-delà de cette fréquence, l'atténuation ne permettra pas de garder un signal avec un bon rapprot signal à bruit. Ceci a pour effet d'amener de l'interférence entre symboles (Inter Symbol Interference) (ISI) et d'être défavorable à la liaison si le bruit additif est trop important. Le canal provoque un étalement des signaux reçus sur plusieurs durées d'un symbole. Par exemple, un symbole à '1' qui avait à l'émission une durée Δ et une amplitude unitaire sera atténué par le canal et aura de l'influence sur les symboles suivants comme montré sur la figure 0.2.

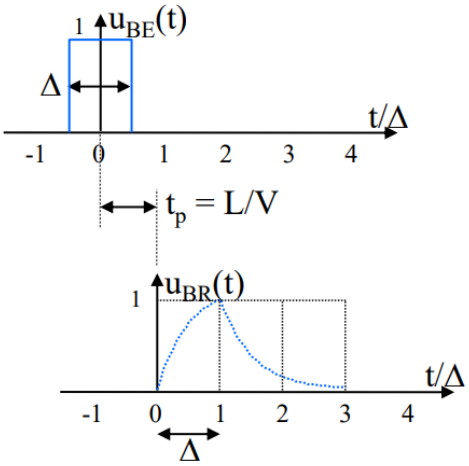


Figure 0.2 – *Influence du canal sur un symbole* [13]

À un instant donné t_i, le signal reçu ne comporte pas seulement l'impact du symbole a_i mais bien la superposition de plusieurs symboles atténués par le canal. Cela aura tendance à déformer le signal dont voici un exemple à la figure 0.3.

2. Bruit provenant de la fluctuation statistique du nombre d'électrons ou de trous participant à la création d'un courant lorsqu'ils traversent une barrière de potentiel. Ce bruit suit une loi de Poisson pouvant être assimilée à une loi normale puisque les porteurs de charge sont nombreux et indépendants. Il est considéré comme un bruit blanc additif gaussien (Additif White Gaussian Noise), dans une gamme de fréquences inférieures à 1 THz (ce qui est largement supérieur aux fréquences utilisées en pratique).

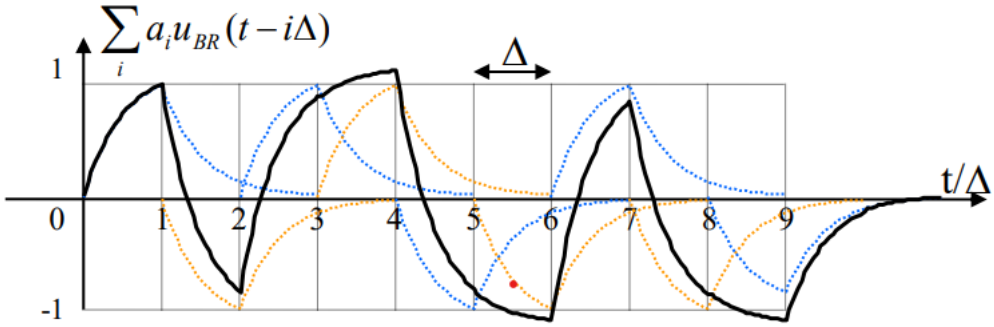


FIGURE 0.3 – *Exemple d'interférence entre symboles (Inter Symbol Interference)* [13]

Quant au bruit, l'hypothèse d'un bruit blanc additif gaussien (Additif White Gaussian Noise) sera considérée. Cette hypothèse implique que la puissance moyenne du bruit est répartie uniformément sur toutes les fréquences. Un tel bruit n'existe pas en pratique mais, sur une largeur de bande limitée, cette hypothèse est très raisonnable.

0.4 Filtrage de Nyquist

Le filtre de Nyquist est un filtre qui permet de respecter la condition de non-ISI. Cette condition est vérifiée lorsque la valeur du signal équivaut à la valeur du signal émis pour chaque instant d'échantillonnage. Les réponses temporelle et fréquentielle du filtre de Nyquist est donnée par l'expression suivante :

$$g(t) = \frac{1}{\Delta}\frac{\sin(\pi t/\Delta)}{\pi t/\Delta}\frac{\cos(\pi\alpha t/\Delta)}{1-(2\alpha t)^2} \tag{9}$$

$$G(f) = \begin{cases} 1 & \text{si } |f| < \frac{R}{2} - \Delta f \\ \frac{1}{2}(1 - \sin\left(\frac{\pi|f|}{\alpha R} - \frac{\pi}{2\alpha}\right)) & \text{si si } \frac{R}{2} - \Delta f < |f| < \frac{R}{2} + \Delta f \\ 0 & \text{si } \frac{R}{2} + \Delta f < |f| \end{cases} \tag{10}$$

Le facteur d'arrondi α (roll-off factor en anglais) est le paramètre important pour caractériser le filtre en cosinus surélevé. Ce facteur a une valeur comprise entre 0 et 1 inclus. Au plus α est proche de 0, au plus la bande passante du filtre diminue et au plus les oscillations entre les instants d'échantillonnage sont grandes. Voici une représentation du filtre de Nyquist pour différentes valeurs du facteur de roll-off.

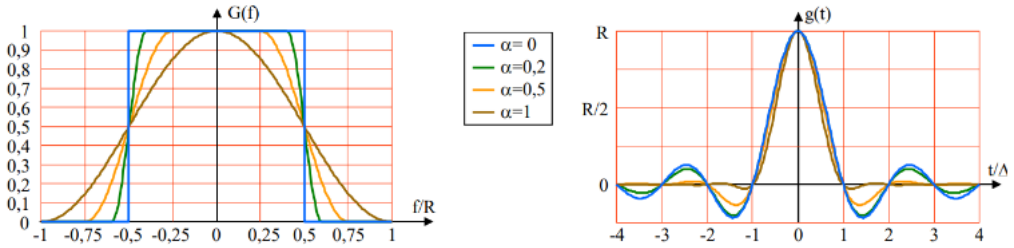


FIGURE 0.4 – *Filtre de Nyquist pour différents* α [13]

En pratique, un filtre de Nyquist avec $\alpha = 0$ est irréalisable car il est non causal. Le choix optimal du facteur d'arrondi sera donc aux alentours de 0,2 pour trouver un compromis entre la bande passante du filtre et les oscillations résiduelles entre les instants d'échantillonnage.

0.5 Diagramme de l'oeil

Le diagramme de l'oeil est un outil visuel qui permet de déterminer plusieurs caractéristiques de la transmission. Ce graphique se construit en superposant chaque trace temporelle du signal reçu après le premier filtre. C'est l'horloge symbolique récupérée au récepteur qui sert de base de temps pour l'élaboration du diagramme de l'oeil par un oscilloscope. Il suffit de représenter le signal temporel par pas de 2Δ et revenir en $t = 0$ pour les deux prochains symboles et ainsi de suite. Les premières étapes de la construction se trouvent aux figures A.3, A.4, A.5 et A.6.

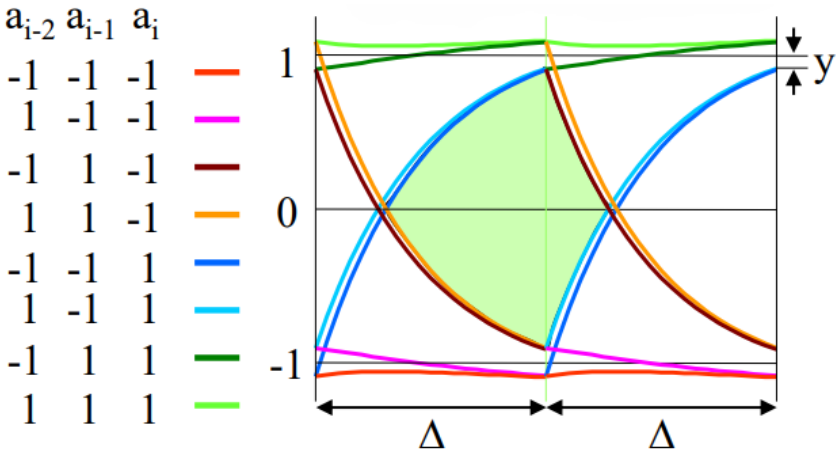


Figure 0.5 – *Exemple de diagramme de l'oeil avec le code Non Return to Zero polaire* [13]

L'ouverture verticale de l'oeil permet d'évaluer la marge de bruit que la transmission peut avoir pour garder de bonnes performances. L'ouverture horizontale montre l'impact qu'un décalage sur la récupération de l'horloge peut avoir sur la régénération des symboles. Le meilleur instant d'échantillonnage et le meilleur seuil de décision peuvent être ainsi évalués. La figure 0.5 est représentative d'une bonne transmission en NRZ polaire avec peu d'ISI.

Chapitre 1

Cahier des charges

Le projet proposé par le département de télécommunications se concentre sur le développement d'un outil pédagogique qui intègre des simulations interactives au sein du cours de "Principes des télécommunications numériques" [13]. L'objectif fondamental de cet outil est de remplacer les illustrations statiques utilisées traditionnellement dans ce cours par des simulations à aspect pédagogique. Ces dernières permettent aux utilisateurs de modifier les paramètres clés des systèmes de télécommunications numériques et d'observer directement les effets de ces modifications sur divers aspects du fonctionnement de ces systèmes. L'outil, une fois achevé, sera mis à la disposition des étudiants de ce cours ainsi que des étudiants de cours connexes, tels que "Communications Analogiques et Numériques", ce qui devrait entraîner une amélioration significative de la compréhension et de l'assimilation des concepts complexes inhérents à ce domaine.

L'outil doit répondre à des critères stricts d'utilisation pratique de manière à faciliter son adoption par les étudiants. À cet fin, un Google forms (Voir annexe A.3.1), a été réalisé dés le début du projet et a été envoyé aux étudiants de troisième bachelier et de première année de master électricité ayant déjà suivi ce cours. L'objectif est de récolter les éléments clés qui nécessitent une particulière attention lors du choix des illustrations à simuler. Ce questionnaire sert aussi à définir des critères d'utilisation sachant que le but est de garantir un outil pédagogique simple d'usage. Pour le peu de réponses reçues (une vingtaine d'étudiants ont reçu le questionnaire et sept réponses nous ont étées renvoyées), les points de matières qui ont été majoritairement mis en avant furent : la récupération de l'horloge symbolique [1] et l'influence de paramètres (rapport signal sur bruit, facteur de roll-off, débit binaire, ...) sur le diagramme de l'oeil. De plus, cela a permis d'identifier que le support le plus adéquat pour les étudiants est sous forme d'un Jupyter Notebook à compléter. Et qu'ils souhaiteraient également que l'outil soit utilisé en classe afin de faciliter la compréhension dès l'introduction des concepts.

Par ailleurs, l'outil proposé doit répondre à plusieurs exigences fondamentales. Tout d'abord, il doit être intuitif, offrant ainsi aux étudiants une interface utilisateur conviviale et exploitable facilement. Cela est essentiel pour permettre aux étudiants de se concentrer sur la compréhension des principes sous-jacents sans être distraits par des complexités techniques inutiles. Ensuite, il est crucial que l'outil présente les différentes illustrations de manière précise et structurée, afin de rendre l'expérience d'apprentissage aussi claire et

1. Les deux méthodes vues au cours sont la boucle à verrouillage de phase (Phase-Locked Loop) (PLL) et le filtrage sélectif d'une raie présente dans le spectre de la densité de puissance du signal.

pédagogique que possible. L'outil doit être modulable et facilement réexploitable pour un futur projet complémentaire. Par ce fait, le script devra être argumenté de commentaires permettant une compréhension aisée du programme. Toutefois, lors de l'implémentation, nous nous sommes vite rendus compte que la disposition sous forme d'un Jupyter Notebook ne permettrait pas une utilisation aisée de l'outil. C'est pourquoi nous nous sommes écartés de l'exigence des étudiants à ce sujet et avons plutôt essayé de créer une interface graphique modulable et permettant diverses simulations.

Enfin, le projet se déroule en différentes phases. La première étape consiste à identifier les illustrations du cours qui sont transformées en simulations interactives. Cela nécessite une collaboration étroite avec les enseignants responsables du cours, afin de garantir que les simulations choisies couvrent les aspects les plus importants et les plus pertinents des télécommunications numériques. Une fois ces illustrations sélectionnées, la phase de conception commence. Chaque module de simulation demande une attention méticuleuse aux détails, avec des tests rigoureux pour s'assurer de la précision et de la fiabilité des résultats obtenus. Les modules doivent être suffisamment flexibles pour permettre aux étudiants de les expérimenter avec différents paramètres, tout en étant robustes pour éviter des erreurs ou des résultats inattendus. Une fois ces différents modules de simulation créés et testés, ils doivent être regroupés au sein d'un seul outil cohérent. Cette unification garantit une expérience plus fluide et homogène. Pour compléter cet outil, un guide complet et détaillé doit être rédigé offrant des instructions claires sur la façon d'utiliser les différentes fonctionnalités.

Pour conclure, ce projet représente une opportunité de développer un outil pédagogique innovant qui non seulement modernise le cours de "Principes des télécommunications numériques", mais qui pourrait également avoir des applications plus larges au sein du département de télécommunications. L'acquisition et la mise en service de compétences techniques contribue directement à l'amélioration de l'expérience d'apprentissage pour les générations futures.

Chapitre 2

État de l'Art

Les télécomunications numériques constituent la colonne vertébrale de la communication moderne et son importance dans le contexte pédagogique réside dans la nécessité de former des ingénieurs, des techniciens ou bien des chercheurs capables de concevoir, de maintenir ou bien d'analyser ces systèmes complexes. Il est donc crucial d'apporter un outil illustratif à l'apprentissage des concepts fondamentaux comme la détection d'erreur, la modulation, le type de codage et l'influence du canal dans une transmission. L'objectif principal est de fournir des outils et des ressources pédagogiques plus accessibles et encourager également l'expérimentation pratique. Certains modèles ou outils ont déjà fait leurs preuves, voici une discussion qui laissera transparaître leurs avantages et inconvénients.

Le logiciel de simulation le plus enclin à répondre à ces objectifs est GNU Radio [1][10]. Il a été conseillé lors de la proposition de projet effectuée par le service de télécomunications. Son avantage réside dans son accès Open source et sa flexibilité sur le traitement de signaux personnalisés. Il peut également interagir avec des signaux réels, ce qui pourrait rentrer dans le cadre d'une utilisation en travaux pratiques. Mais il ne permet pas une interface graphique avancée comme Matlab et nécessite des connaissances poussées du logiciel pour pouvoir réaliser ses propres projets expérimentaux.

Un autre logiciel, qui permet aussi de réaliser un tel outil de simulation, est Matlab avec la plateforme Simulink [2] [12]. Il offre une interface conviviale avec de nombreux outils graphiques et de visualisation. Avec sa large bibliothèque de fonctions et d'extensions, cela le rend idéal pour des projets pédagogiques. Matlab bénéficie aussi d'une documentation détaillée, de nombreux tutoriels et d'aide en ligne [4, 7, 11]. Néanmoins, c'est un logiciel sous licence, ce qui est un obstacle pour certaines institutions éducatives. Il nécessite aussi un matériel relativement performant. De plus, Matlab a une prise en main compliquée et son interface peut être perçue comme complexe pour des utilisateurs débutants.

Plus puissant et précis, Cadence [3] est reconnu pour son efficacité dans la simulation de circuits et de systèmes complexes [1]. Il est largement utilisé dans l'industrie, ce qui le rend attrayant pour les étudiants cherchant à acquérir des compétences pratiques demandées par les employeurs. Cependant, Cadence n'est pas spécifiquement conçu pour les simulations de télécommunications numériques, ce qui peut le rendre moins convivial

1. Site officiel : `https://www.gnuradio.org/`
2. Site officiel : `https://nl.mathworks.com/products/simulink.html`
3. Site officiel : `https://www.cadence.com/en_US/home.html`

pour des utilisations pédagogiques. Son orientation vers la conception de circuits le rend également moins flexible pour des simulations didactiques qui nécessitent une approche plus visuelle et interactive.

Ces trois logiciels, bien qu'efficaces pour simuler et visualiser des aspects des télécommunications numériques, présentent des limitations lorsqu'il s'agit d'offrir une visualisation globale de l'influence de différents paramètres sur une transmission. Les simulations existantes se concentrent souvent sur des éléments spécifiques tels que l'influence du bruit sur le diagramme de l'œil de Nyquist, la régénération des symboles, ou encore l'effet de l'ISI sur le taux d'erreur binaire.

Sans être un logiciel, mais avec de nombreuses bibliothèques, il est envisageable de concevoir cet outil au moyen du language de programmation Python [9]. Son principal intèrêt réside dans le fait qu'il ne possède pas de bloc pré-fait et que tout est à concevoir. Ainsi, il est possible d'adapter les fonctions pour qu'elles s'interfacent toutes entre elles. De plus, il offre l'avantage de ne demander que peu de ressources supplémentaires dont les étudiants ne disposeraient pas encore. Il est possible de faire fonctionner le tout grâce à un logiciel d'exploitation comme Visual Studio ou Anaconda Jupyter Notebook, facile à installer et utilisable sur le long terme. Mais aussi de structurer le script en blocs individuels permettant de séparer les différentes simulations, cela permet un aspect pédagogique idéal [2]. Enfin, il est possible de trouver beaucoup d'aides en ligne et d'exemples de réalisations pratiques similaires à certaines fonctionnalités à développer.

En résumé, le projet pédagogique vise à combiner ces différentes simulations pour créer un outil intégré capable de regrouper diverses fonctionnalités en une seule interface. En prenant inspiration des outils existants, l'objectif est de fournir un environnement de simulation qui permet aux étudiants d'explorer les différents paramètres affectant les télécommunications numériques tout en offrant une interface utilisateur conviviale et intuitive.

Chapitre 3

Conception et Architecture

Voici le chapitre dédié à la conception et à l'architecture, où nous explorons le design de notre application de simulation. De la genèse des idées à la concrétisation des concepts de télécommunications à travers le langage de programmation python, chaque étape reflète notre engagement à créer un produit fonctionnel et intuitif.

3.1 Choix des illustrations

Tout d'abord, comme stipulé dans le cahier des charges, en concertation avec le corps enseignant, nous avons réalisé une liste de sujets potentiels à illustrer dynamiquement. Voici la liste des illustrations retenues par ordre de priorité en fonction de notre choix personnel mais aussi du Google Forms (Voir annexe A.3.1) reprenant l'avis des étudiants ayant suivi les cours de télécommunications numériques :

1. Récupération de l'horloge symbolique en fonction du type de code par filtrage ou par la boucle à verrouillage de phase (Phase-Locked Loop). (Chapitre 9 : Codage de canal en bande de base [13])
2. Impact du changement du filtre de Nyquist dans le domaine temporel et fréquentiel sur le signal et sur le diagramme de l'oeil, pour différentes valeurs du facteur de roll-off. (Chapitre 10 : Influence du canal de transmission en bande de base [13])
3. Transposition de fréquence, récepteur superhétérodyne et principe de démodulation FM. (Chapitre 4 : Changement de fréquence [13])
4. Impact des non-linéarités au niveau temporel. (Chapitre 5 : Non-linéarités [13])
5. Modulation AM, FM et PM avec pour paramètres : la fréquence de la porteuse, la profondeur de modulation, le taux de modulation, l'écart de fréquence et l'écart de phase. (Chapitre 4 : Modulation [13])
6. Principe de génération d'une SBPA et son spectre pour différentes valeurs de période. (Chapitre 6 : Signaux aléatoires [13])
7. Impact de la longueur du canal sur l'étalement temporel des symboles et de l'ISI. (Chapitre 8 : Schématiques des transmissions numériques [13])

Pour commencer le projet, nous voulions réaliser la première d'entre-elles, la récupération d'horloge symbolique, mais notre promotrice nous a désorienté de ce choix à raison. En effet, la PLL est un système dynamique avec des rétroactions. Son comportement peut être décrit par des équations différentielles et comprendre leur comportement est essentiel pour concevoir efficacement la boucle. Cela nécessite donc des connaissances transversales avec le cours de Commande des procédés [1]. De plus, lors de l'explication de ce sujet, seule une introduction du concept et des idées clés de ce système nous ont été évoquées, il est en fait plus complexe qu'à priori. Un tel outil de simulation nécessiterait une collaboration avec le service de SECO (Systems, Estimation, Control and Optimisation Unit) et une amélioration approfondie de nos connaisances sur la PLL. Il aurait été possible de réaliser cette étude mais ça sortait un peu de la thématique initiale et ça nous aurait pris l'entièreté du quadrimestre pour étudier la PLL. Nous avons donc préféré réaliser un outil de simulation qui, certes démontrerait un potentiel d'apprentissage sur les télécommunications, mais nécessiterait également d'étudier des éléments et concepts supplémentaires à propos de cette thématique.

Nous avons donc choisi d'apporter une représentation dynamique de l'influence du canal (bruit et ISI) et de l'impact du filtre de Nyquist. Nous simulons en fait la transmission en bande de base d'une suite binaire pseudo-aléatoire en laissant certains paramètres variables pour permettre d'en voir leurs influences. Ainsi, l'outil de simulation permettra de voir le signal source, puis le signal attribué de ses symboles, ensuite ce même signal en sortie du canal et enfin, en sortie du filtre de Nyquist. De plus, il permettra de montrer l'influence du canal sur un symbole, et donc de témoigner de l'importance de l'ISI, mais aussi celle du filtre de Nyquist. Pour terminer, il sera également possible d'afficher dynamiquement le diagramme de l'oeil des signaux en sortie du canal et du filtre de Nyquist.

3.2 Conception du signal

Pour commencer, il a été nécessaire de concevoir un signal de départ qui servirait de fil rouge dans les différentes simulations, témoignant des différents impacts subis lors de sa transmission. Cela fut vite réalisé. Grâce à python, il est très facile de créer un vecteur de taille réglable qui contient des valeurs aléatoires entre 0 et 1 représentant les bits. Toutefois, il faut s'assurer que cette suite comporte assez de transitions pour pouvoir conserver l'hypothèse d'une SBPA considérée dans les théories sous-jacentes. Cette preuve est réalisée plus loin dans ce rapport (Voir section 4.2.1). De plus, une fois le signal binaire créé il faut espacer ces points d'un certain écart : la période, et associer pour chacun d'eux une forme d'onde dépendant du type de code utilisé. Il faut aussi s'assurer d'avoir assez de points pour chaque symbole pour respecter le critère de Shannon [2].

1. Ce cours aborde des concepts de gestion de contrôleur PID.
2. La représentation discrète d'un signal exige des échantillons régulièrement espacés à une fréquence d'échantillonnage supérieure au double de la fréquence maximale présente dans ce signal.

3.3 Représentation du signal

Lors des séances de cours théoriques, il n'est parfois pas possible de montrer une évolution temporelle et spectrale, par manque de place sur les slides ou par manque d'un générateur pertinent d'illustrations. C'est une opportunité à saisir. Pour chaque moment clé de sa transmission les deux évolutions seront affichées permettant ainsi aux étudiants de comprendre les influences des paramètres de simulation dans leur globalité et fournir également des illustrations à certains concepts parfois complexes à visualiser mentalement.

3.4 Conception du canal

Pour simuler la transmission, nous sommes contraints de concevoir un canal fictif mais, comme expliqué au point 0.3 du cadre théorique, il peut être assimilé à un filtre passe-bas suivi d'un bruit blanc gaussien et additif. Pour réaliser ce filtre passe-bas, nous avons choisi un filtre de Butterworth. Il dispose d'une caractéristique de réponse en fréquence plate dans la bande passante utile, ce qui signifie qu'il introduit peu ou pas de distorsion de phase dans cette plage de fréquences (Voir figures 3.1 et 3.2). Cela peut être crucial pour maintenir l'intégrité du signal de communication à travers le canal.

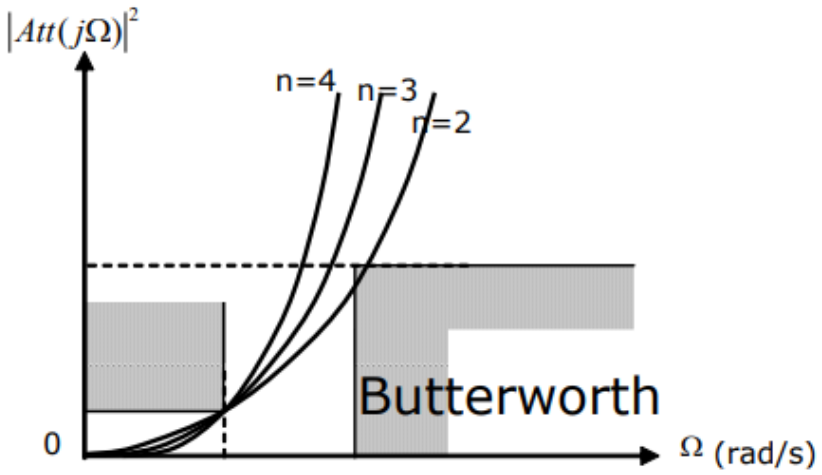


Figure 3.1 – *Fonction caractéristique du filtre passe-bas de Butterworth [3]*

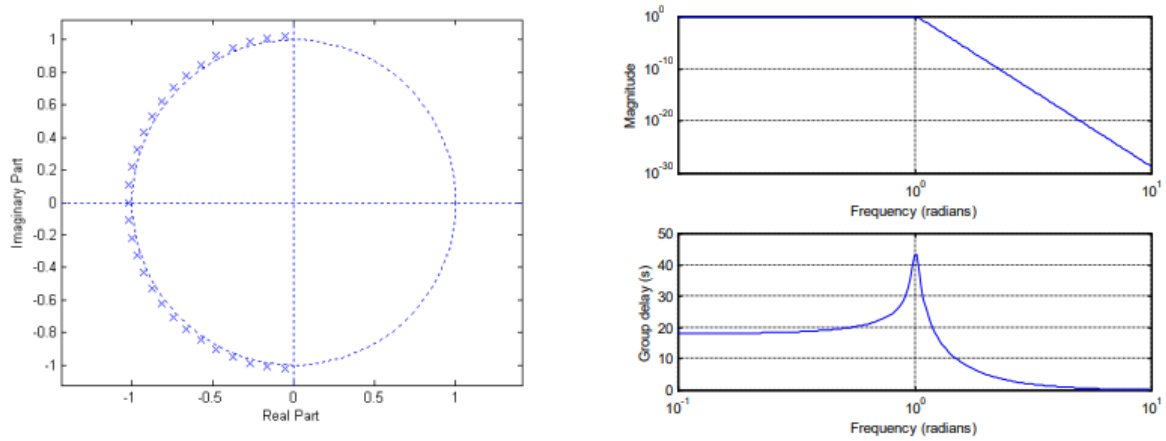


FIGURE 3.2 – *Pôles et zéros, gain et délai de groupe de l'approximation de Butterworth d'un filtre passe-bas normalisé [3]*

Les filtres de Butterworth sont relativement simples à concevoir et à mettre en œuvre, ce qui les rend attrayants pour la simulation de canaux de communication. Leurs caractéristiques de conception, telles que l'ordre du filtre et la fréquence de coupure, peuvent être facilement ajustées pour répondre aux besoins spécifiques de la simulation. Enfin, les filtres de Butterworth sont stables et linéaires, ce qui signifie qu'ils préservent la forme d'onde du signal d'entrée sans introduire de distorsions non linéaires. Cela permet de simuler plus fidèlement le comportement du canal dans des conditions réalistes.

Ensuite, il nous reste à concevoir un ajout de bruit, cela consiste en l'ajout d'une proportion de la puissance du signal émis, régie par le rapport signal sur bruit imposé et la racine carrée de la covariance du code [3]. Ainsi, pour un rapport signal sur bruit de 20 dB et une puissance de signal unitaire, la puissance de bruit sera 100 fois plus faible. Le signal sera affecté, pour chaque point qui le représente, d'un bruit variant de -3 à 3 fois la puissance du bruit (comme le témoigne la figure 3.3), pour une répartition gaussienne centrée en 0 et d'une variance égale à 1 fois la puissance du bruit (c'est le cas du code NRZ par exemple), ce qui est statistiquement valable si l'on considère du bruit thermique.

3. Pour rappel, puisque la moyenne (= l'espérance mathématique) du bruit AWGN est nulle, alors sa puissance (= valeur quadratique moyenne, c'est-à-dire l'espérance mathématique de la variable aléatoire au carré) est égale à sa variance. Et la variance, c'est le carré de l'écart type de la gaussienne.

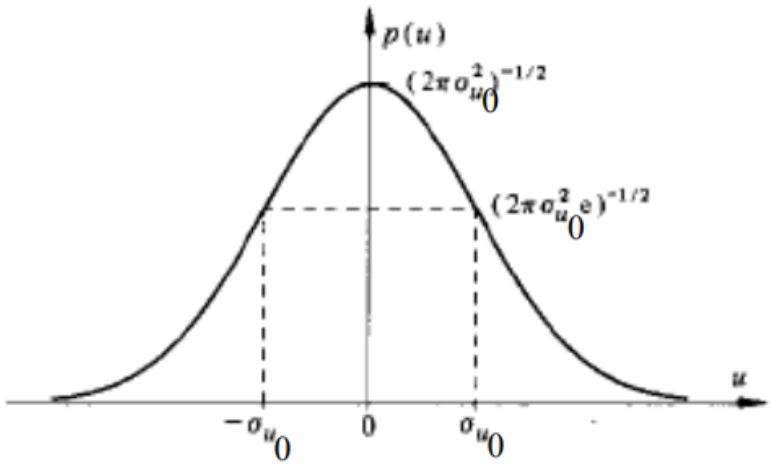


Figure 3.3 – *Statistique de tension de bruit thermique [13]*

Maintenant que le signal a été transmis, il ne nous reste plus qu'à le faire passer dans le filtre de Nyquist pour faciliter sa réception.

3.5 Conception du filtre de Nyquist

Le filtre de Nyquist est utilisé dans les systèmes de communication numérique pour régénérer les signaux transmis afin de minimiser l'ISI et ainsi améliorer la performance du système. Numériquement, la conception d'un filtre de Nyquist implique généralement la spécification de sa réponse fréquentielle et l'application de techniques de conception de filtres numériques pour atteindre cette réponse souhaitée. En pratique, il est utilisé une fois à l'entrée du canal et une seconde fois à la sortie mais pas dans son intégrité, nous plaçons en fait la racine carrée de la fonction de transfert du filtre pour qu'au final cela ait le même impact mais en prenant en compte les éventuels changements dans le canal. En effet dans la réalité, le canal est un élément physique qui est donc soumis à des modifications permanentes. Ainsi, le filtre permet également de jouer le rôle d'égaliseur. Dans notre cas, la nécessité d'utiliser un filtre en racine est moindre puisque le canal reste fixe et stable dans le temps, ainsi que ses caractéristiques de transmission qui restent constantes.

Pour le concevoir, nous avons utilisé des outils de traitement du signal inclus dans la bibliothèque *scipy*. Nous avons spécifié la forme de la réponse en fréquence souhaitée (équation 10), qui répond au critère de Nyquist :

$$U'_{BR}(f) + U'_{BR}(f - \frac{1}{\Delta}) = \Delta U_{BR} \quad si\ 0 \leq |f| \leq \frac{1}{\Delta} \tag{3.1}$$

Dès lors, si $U'_{BR}(f)$ est nulle pour $|f| \geq \frac{1}{\Delta}$ et présente, dans l'intervalle $[0, \frac{1}{\Delta}]$, une symétrie centrale autour de la fréquence $f = \frac{1}{2\Delta}$, alors l'ISI est nulle pour une rapidité de modulation $R = \frac{1}{\Delta}$. La figure 0.4 du cadre théorique donne à gauche la représentation spectrale pour différentes valeurs du facteur de roll-off respectant ce critère. Mais sans maîtriser l'outil parfaitement, le résultat n'était pas fort concluant. Nous avons alors eu l'idée d'utiliser l'outil de convolution plutôt que de jouer sur l'aspect fréquentiel du signal.

Nous avons donc créé la fonction en cosinus surélevé représentant le filtre de Nyquist provenant de l'équation (9). Il ne reste alors qu'à effectuer la convolution entre cette fonction et le signal. Mais sans oublier de maximiser l'amplitude du signal convolué à la fin de chaque symbole, sans prendre aucune précaution le maximum se tiendra au milieu de chaque symbole. Nous avons donc réaliser la convolution de manière plus manuelle sans passer par un outil tout fait (Ceci est détailé à la section 4.5).

3.6 Conception du diagramme de l'oeil

La conception d'un tel diagramme est assez simple lorsque l'on dispose déjà du signal. Il suffit de découper le signal en différentes courbes de "longueur temporelle" égale à 2Δ et de les superposer pour obtenir quelque chose de similaire à la figure 0.5 du cadre théorique [8]. Toutefois, il faut aussi pouvoir gérer les différentes couleurs des courbes et donner une couleur pour chaque triple de symboles. Ainsi, une sucession -1, -1, 1 aura une couleur différente de -1, 1, 1. Cela nous aurait demandé beaucoup de temps pour gérer un petit détail d'affichage nous ne l'avons donc pas fait. L'outil permet également de déterminer des caractéristiques de la transmission mais aussi de visualisuer le meilleur instant d'échantillonage pour la regénération des bits initiaux. Pour améliorer cette visibilité, nous avons ajouté un quadrillage aux multiples de $\frac{1}{\Delta}$. Néanmoins, souvent lors de l'apprentissage de ce diagramme, il est parfois difficile de comprendre comment il a été réalisé, nous avons donc pallié ce problème et créé une animation qui construit l'oeil pas à pas.

3.7 Choix des paramètres de simulation

Pour chaque transformation effectuée sur le signal de départ, nous laissons le choix à l'utilisateur de choisir certains paramètres de simulation dont en voici la liste :

1. Débit binaire
2. Choix du code
3. Ordre du filtre représentant le canal
4. Fréquence de coupure du filtre représentant le canal
5. Rapport signal sur bruit de la communication
6. Facteur de roll-off

Avec ces 6 paramètres, il est possible de voir beaucoup de phénomènes et de concepts issus du cours de télécomunications numériques. Visualiser, par exemple, l'impact de l'ISI, l'impact du bruit sur le diagramme de l'oeil ou l'impact d'une fréquence de coupure du filtre représentant le canal plus faible que le débit binaire est tout à fait réalisable. Pour faciliter l'utilisation de notre outil, nous avons pensé à une interface graphique séparant en différentes fenêtres les parties de la transmission du signal.

3.8 Architecture de l'outil

L'idée est de montrer le signal à divers moments de sa transmission dans une représentation temporelle et spectrale. Nous avons donc réalisé un croquis [4] qui intégrerait diverses fonctionnalités (Voir annexe A.7).

Finalement, après réalisation de l'outil, notre page d'accueil se compose donc de 7 graphes, le premier d'entre-eux représente l'évolution temporelle de la suite binaire [5]. Il vient ensuite la représentation du signal codé (le choix du code s'effectue par les boutons radio en haut à droite de la fenêtre) puis en sortie de canal et enfin en sortie du filtre de Nyquist. Entre chaque graphe, un bouton permet d'ouvrir une seconde fenêtre qui permet respectivement de visualiser la modification du canal et du filtre de Nyquist. Ces fenêtres supplémentaires témoignent de l'influence de l'étape concernée sur un seul symbole. Cependant, ce ne sont pas les seules fenêtres que l'on peut ouvrir, il est possible grâce au bouton 'eye' d'ouvrir également le diagramme de l'oeil du signal qui est à côté du bouton. Une représentation de la page d'accueil est illustrée à la figure 3.4, ainsi qu'en annexes A.8, A.9 et A.10 représentant les fenêtres du canal, du filtre de Nyquist et du diagramme de l'oeil.

Quant à la modification des paramètres, nous laissons l'utilisateur choisir sa valeur grâce à une glissière. Chaque glissière a un minimum et un maximum, ce qui limite les observations possibles. Néanmoins, seules les glissières de débit binaire, de fréquence de coupure et du rapport signal sur bruit, ont un impact non-négligeable sur la qualité d'une transmission. En effet, l'ordre du filtre est un paramètre plutôt fixe qui reste unitaire dans la majeure partie des cas. Des valeurs types d'utilisation pour voir tel ou tel phénomène sont explicitées dans le guide d'utilisation (Voir annexe A.3.2).

4. Remarque : Ce croquis tient compte de fonctionnalités qui n'ont pas été implémentées mais qui pourront l'être si ce projet est continué par de futurs étudiants.

5. Il n'y a pas de sens d'afficher son spectre car c'est une suite d'éléments discrets.

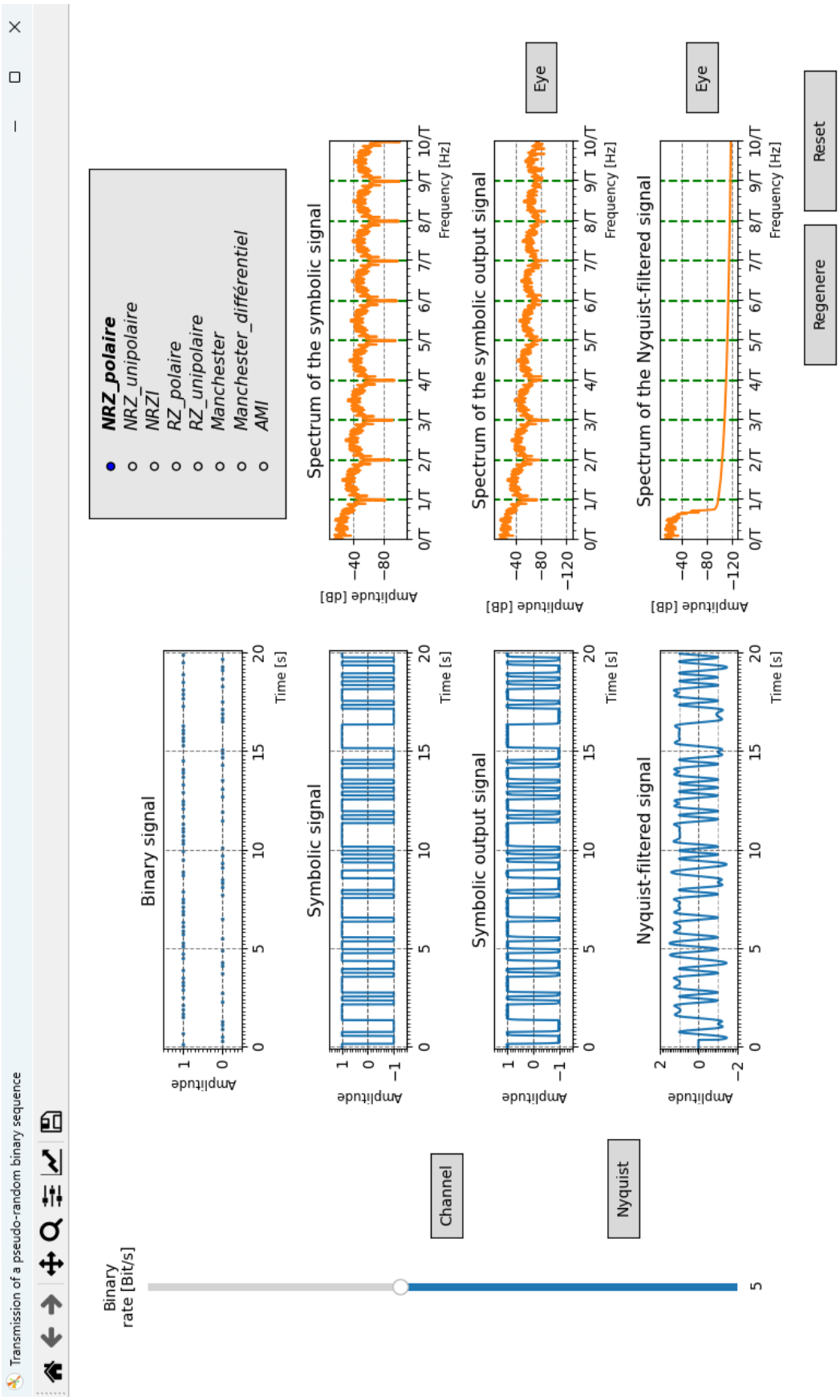


FIGURE 3.4 – *Page d'accueil - Transmission of a pseudo-random binary sequence*

Chapitre 4

Implémentation

Ce chapitre est dédié à l'explication des méthodes utilisées lors de l'implémentation de notre outil pédagogique de simulation. La manière dont le signal de départ a été créé et transformé s'y retrouve ainsi que l'application d'un filtre, la création d'outils de simulation et d'une animation. Pour concevoir le script, nous avons utilisé le logiciel Visual Studio Code avec l'extension Jupyter, ce qui nous permet de pouvoir structurer plus facilement le script en diffèrentes cellules exécutables séparément.

4.1 Structure du script

La première cellule sert à l'installation de toutes les librairies nécessaires. Nous avons eu besoin d'importer la librairie numpy pour effectuer des calculs, scipy pour traiter les signaux et concevoir le canal fictif, random pour générer de l'aléatoire, matplotlib pour afficher ou animer des graphes et PyQt5 pour paramétrer l'affichage des fenêtres. La seconde cellule reprend le script général et la troisième cellule le script de l'animation du diagramme de l'oeil.

La seconde cellule a été entièrement écrite en suivant toujours l'ordre des différentes étapes de la transmission du signal. À savoir, la création du signal, sa codification, sa transmission et son filtrage. La première partie se contente d'importer les outils nécessaires de chaque librairie et ensuite toutes les fonctions nécessaires au différentes étapes de transmission y sont détaillées. Après cela, nous effectuons toutes les étapes nécessaires à la transformation du signal de départ jusqu'à la sortie du filtre de Nyquist. Puis, les fonctions de chaque outil de simulation y sont détaillées.

Enfin, la troisième cellule vient récupérer le signal en sortie du canal et en sortie du filtre de Nyquist pour superposer sur un graphe les courbes d'une durée de 2Δ créant ainsi pas à pas le diagramme de l'oeil.

4.2 Créer une SBPA

Pour bien faire, il aurait été en meilleure adéquation avec le cours de créer le signal binaire à l'aide de registres à décalage (comme expliqué au point 0.1) et d'apporter ainsi une illustration supplémentaire. Toutefois, nous n'avons pas réalisé cela. En commençant le projet nous avons créé la suite binaire pseudo-aléatoire à l'aide de la fonction $random()$

pour être sûr que la séquence ne poserait pas de problème par la suite avec nos autres fonctions. Par après, nous avons toujours gardé cette source. Néanmoins, il serait désormais préférable de la remplacer par la source à registre de décalage mais nous avons préféré garder notre temps restant à implémenter d'autres concepts plus important du cours.

4.2.1 Vérification de la SBPA

Pour se convaincre que notre suite de départ est bien une SBPA, il faut calculer son autocorrélation grâce à la formule suivante :

$$R_{xx}(t - \Delta t) = \frac{1}{N\Delta t} \int_0^{N\Delta t} x(t)x(t - \Delta t)\,\mathrm{d}t \tag{4.1}$$

Pour un débit binaire de 5 [bit/s], voici le résultat ainsi que la représentation théorique pour une SBPA :

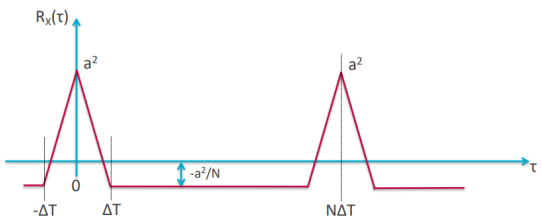


Figure 4.1 – *Autocorrélation d'une SBPA [13]*

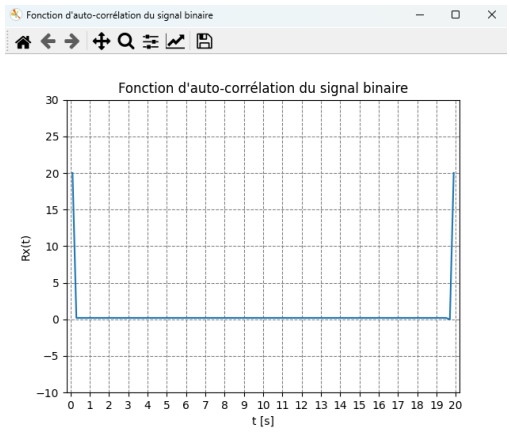


Figure 4.2 – *Autocorrélation de la suite binaire avec un débit de 5 [bit/s]*

Cela prouve que notre suite générée avec la fonction *random*() peut être assimilée à une SBPA, et respecte donc les théories basées sur les signaux aléatoires. Parfois, l'autocorrélation est strictement positive, cela s'explique par le fait qu'elle est basée sur une suite purement aléatoire. En effet, avec un registre à décalage la période N est en fait toujours de la forme $2^n - 1$ qui est donc un nombre impair. Par ce fait, il y a toujours un nombre impair de valeurs négatives dans la suite ce qui en résulte la valeur $-\frac{a^2}{N}$. De plus, dans certains cas, notre suite est composée d'autant de valeurs négatives que de valeurs positives. Il en résulte donc une autocorrélation purement nulle (Voir annexe A.11). En pratique, un générateur de SBPA est créé avec un N très grand ce qui a pour effet de transformer le graphe de l'autocorrélation en une fonction nulle avec une impulsion de Dirac en $\tau = 0$ et en $N\Delta T$. Dans notre cas, cela revient à augmenter le débit binaire. Un exemple avec un débit binaire de 1 et un autre avec un débit binaire de 10 [bit/s] sont illustrés aux annexes A.12 et A.13.

4.3 Représentations spectrales du signal

Pour permettre aux étudiants de comprendre ce qui se passe d'un point de vue fréquentiel. Nous avons, pour chaque moment clé de la transmision, appliqué la fonction *maplotlib.pyplot.magnitude_spectrum*() qui affiche le spectre en magnitude. Nous pouvons ainsi déterminer les modifications apportées par les différents paramètres sur la réponse fréquentielle du signal et aussi à différentes étapes de la transmission.

4.4 Codage du signal

Les fonctions de chaque type de code sont implémentées au tout début du script. La seule remarque à faire est que l'on a affecté une forme d'onde à la séquence binaire et ensuite nous avons codé chaque point du signal en changeant sa valeur par celle appropriée. Cette méthode donne les mêmes résultats mais est plus simple à implémenter.

4.5 Appliquer le filtre de Nyquist

Voici la dernière transformation à effectuer sur notre signal. Comme mentionné plus haut (Voir section 3.5), nous avons conçu une fonction de convolution adaptée à nos besoins. Cette fonction vient rechercher la valeur de chaque point de chaque symbole, la multiplie par la fonction temporelle en cosinus surélevé du filtre de Nyquist (Voir équation 9) et effectue une somme superposant ainsi toutes les courbes. Une illustration pour une suite de 8 symboles, respectivement 1, -1, 1, -1, -1, 1, 1 et -1 se retrouve en annexe A.14. Pour chaque valeur de symbole, la réponse temporelle du filtre est centrée sur le symbole. En sommant le tout, nous obtenons le signal filtré (représenté en pointillé noir sur l'illustration). Une façon de vérifier que le filtrage a bien été réalisé est de regarder les spectres. Sur la figure 3.4, nous pouvons voir qu'après filtrage de Nyquist (le dernier graphe en bas à droite) seul le premier lobe n'a pas été atténué, cela correspond donc bien à un tel filtrage.

4.6 Conception d'outils dynamiques

Grâce aux nombreuses librairies de matplotlib, il est simple de trouver de la documentation sur la création d'outils dynamiques [6], comme des glissières, des boutons ou des sélecteurs. Une fois les fonctions écrites cela permet pour chaque interruption (clic sur un bouton ou déplacement d'une glissière) l'exécution de cette routine qui applique de nouveau la transmission en prenant ces nouveaux changements en compte. Cela permet donc de répercuter toutes les modifications sur toutes les fenêtres.

4.6.1 Conception de l'animation

Pour réaliser l'animation du diagramme de l'oeil, nous avons utilisé l'outil *animation*() de matplotlib [5]. Nous l'avons conçue pour qu'elle trace chaque courbe représentant 2 symboles, l'une à la suite de l'autre illustrant ainsi la construction de ce diagramme. Mais cela a demandé certaines considérations supplémentaires à apporter à notre signal.

4.7 Problèmes rencontrés

Le problème le plus difficile et qui nous a demandé le plus de temps était lié à l'animation de la construction du diagramme de l'oeil. Notre signal n'ayant pas un nombre entier de points par seconde avait posé des soucis lors de son découpage en morceaux de 2 symboles. Chaque vecteur reprenant ces morceaux devait avoir la même taille. Nous avons donc rendu la fréquence d'échantillonage fonction du débit binaire. Nous l'avons posée 50 fois plus grande pour s'assurer de toujours être au dessus de la limite imposée par le théorème de Shannon même dans les pires des cas, à savoir pour les codes RZ et Manchester, qui divisent la durée des symboles de moitié. Toutefois, ce n'est pas la seule considération à prendre en compte, il faut également s'assurer que tous les points d'une courbe se retrouvent toujours strictement dans l'intervalle de temps $[0, 2\Delta]$ sinon l'animation tracera une ligne qui reliera les points en fin d'intervalle avec ceux du début et l'oeil sera alors illisible. Un contre-exemple est représenté en annexe A.15.

Un autre problème que l'on a du traiter pour que chaque étudiant puisse bénéficier d'une expérience optimale était le paramètrage des dimensions des différentes fenêtres. Désormais, nous récupérons la dimension de l'écran à l'aide de l'outil $screen.geometry()$ ce qui permet d'ajuster l'affichage et de ne plus avoir une fenêtre similaire à celle représentée à l'annexe A.16.

Chapitre 5

Validation et Tests

Ce chapitre montre la comparaison entre les résultats obtenus grâce à notre outil et les résultats théoriques. Une analyse détaillée des résultats concernant le code NRZ polaire est effectuée.

La densité spectrale de puissance $\Phi_e(f)$ présente les mêmes caractéristiques qu'en théorie. En effet, la figure de droite représente le cas théorique alors que celle de gauche est directement tirée de notre logiciel. Les annulations de la densité spectrale de puissance se retrouvent effectivement à tous les multiples de $1/T$ dans les deux cas.

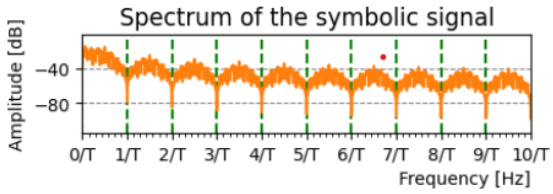


Figure 5.1 – *Densité spectrale de puissance pour le code NRZ polaire en pratique*

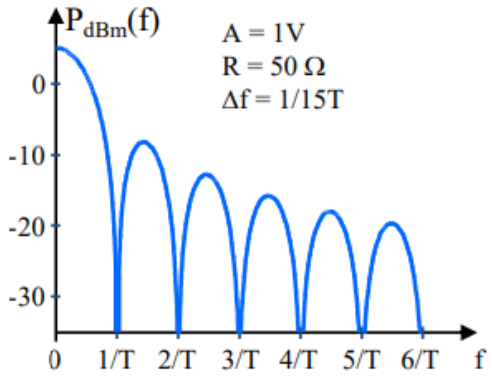


Figure 5.2 – *Densité sprectrale de puissance pour le code NRZ polaire en théorie [13]*

Le canal, assimilé à un filtre passe-bas, entraîne une atténuation du signal après une certaine fréquence de coupure. Cela procure un étalement des symboles et peut engendrer de l'interférence entre eux. La figure 5.3 démontre l'étalement d'un symbole suite au passage dans le canal. La durée initiale de celui-ci est de $1/T = 0.2s$ mais après le passage dans le canal le symbole dure plus longtemps et sa forme d'onde a aussi été atténuée. Ce symbole impacte donc le signal et les symboles qui le suivent.

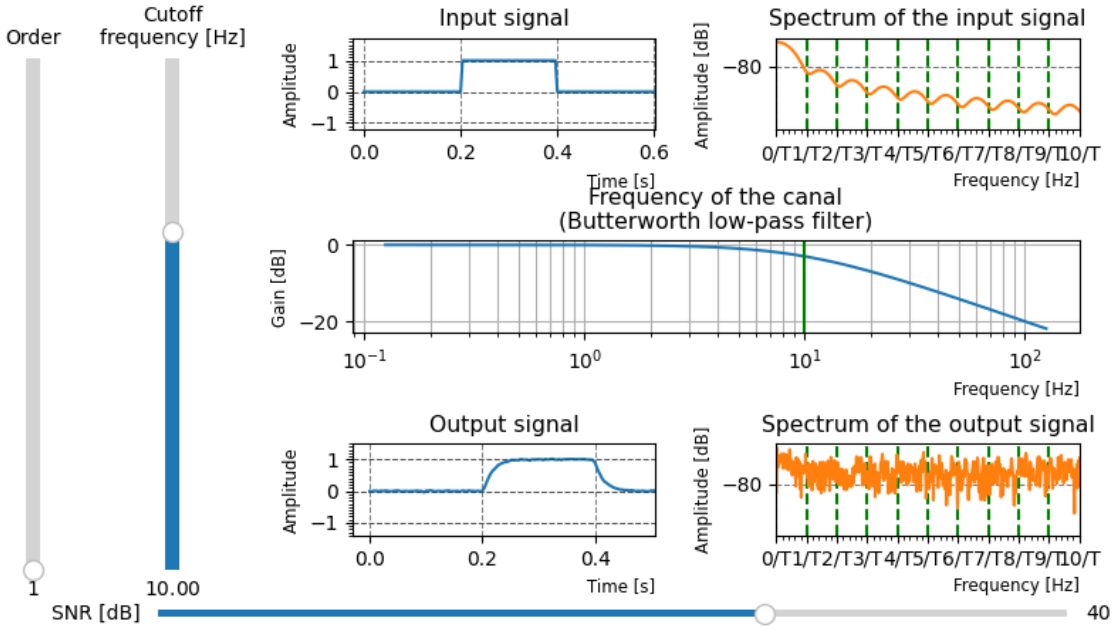


FIGURE 5.3 – *Étalement d'un symbole par passage dans le canal*

L'ISI provoqué par le canal amène à utiliser une représentation de notre transmission en bande de base via le diagramme de l'oeil. La figure 5.5 présente ce qui se retrouve en théorie pour un codage NRZ polaire alors que la figure 5.4 est ce que nous obtenons via notre programme.

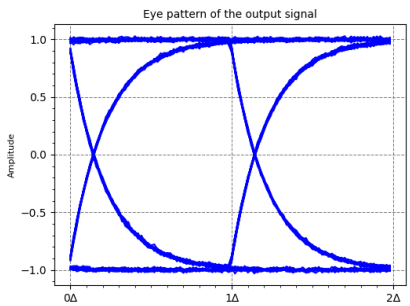


FIGURE 5.4 – *Diagramme de l'oeil pour le codeNRZ polaire avec ISI en pratique*

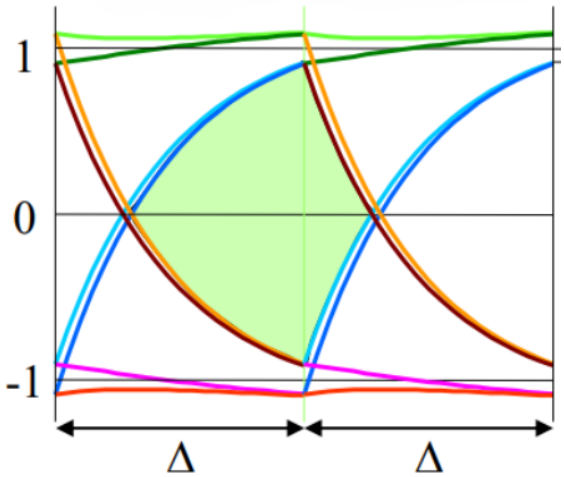


FIGURE 5.5 – *Diagramme de l'oeil pour NRZ polaire avec ISI en théorie [13]*

Le signal passe ensuite par le filtre de Nyquist pour le maximiser aux instants d'échantillonnage. La figure 5.6 représente le filtre de Nyquist que nous avons implémenté pour une valeur de $\alpha = 0.2$. En théorie, la figure 5.7 montre le filtre de Nyquist pour plusieurs valeurs de α : 0, 0.2, 0.5 et 1. La comparaison entre les deux images prouvent que le filtre est bien implémenté pour cette valeur de α. Les figures A.17, A.18, A.19, A.20 et A.21 se retrouvant en annexes illustrent les mêmes conclusions mais pour d'autres valeurs de α.

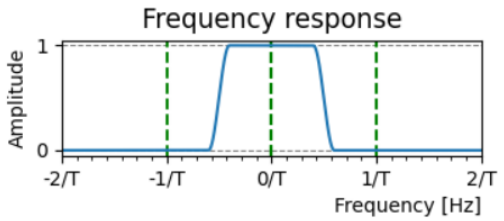


FIGURE 5.6 – *Filtre de Nyquist en pratique pour* $\alpha = 0.2$

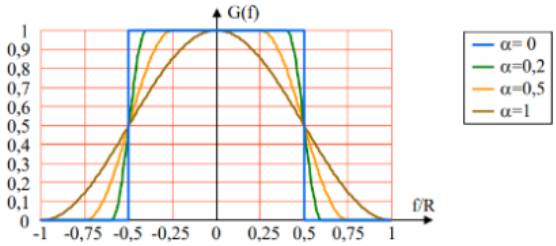


FIGURE 5.7 – *Filtre de Nyquist en théorie [13]*

Une deuxième construction du diagramme de l'oeil peut être réalisée. Cette fois-ci, le signal est passé dans le filtre en cosinus surélevé (dit 'filtre de Nyquist'). Pour une valeur de $\alpha = 1$, nous pouvons comparer ce qui est obtenu grâce à notre programme à la figure 5.8 et l'oeil construit via la théorie à la figure 5.9.

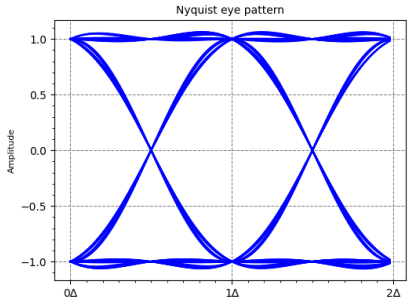


FIGURE 5.8 – *Diagramme de l'oeil filtré en pratique pour le code NRZ polaire avec* $\alpha = 1$

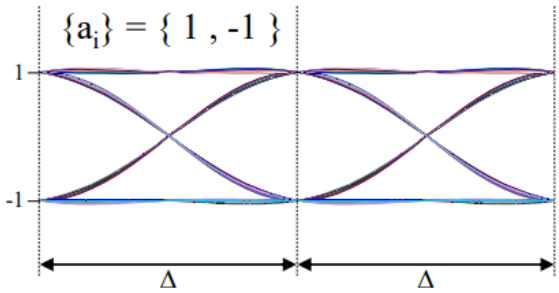


FIGURE 5.9 – *Diagramme de l'oeil filtré en théorie pour le code NRZ polaire avec* $\alpha = 1$ *[13]*

En rajoutant du bruit et en fixant le rapport signal sur bruit (SNR) à 20 dB pour que ça reste lisible, la figure 5.10 représente le signal bruité. L'autre signal à la figure 5.11 a son SNR fixé à 40 dB ce qui revient à ne contenir presque pas de bruit. Les deux figures montrent que notre programme reste en accord avec la théorie car le SNR diminue impliquant un bruit plus marqué. Sur les figures, ça revient à avoir un signal qui varie beaucoup plus autour du signal utile généré par l'émetteur.

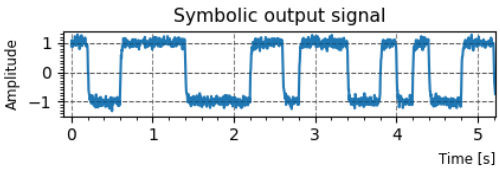


FIGURE 5.10 – *Signal bruité avec SNR de 20dB*

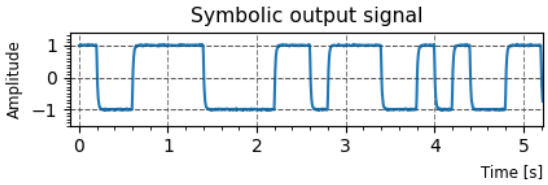


FIGURE 5.11 – *Signal non bruité avec SNR de 40dB*

Nous pouvons aussi montrer l'impact qu'a ce bruit sur le diagramme de l'oeil avant filtrage de Nyquist. C'est ce que les figures 5.12 et 5.13 montrent en comparant à nouveau un signal très faiblement bruité et un signal bruité avec le SNR fixé à 20 dB.

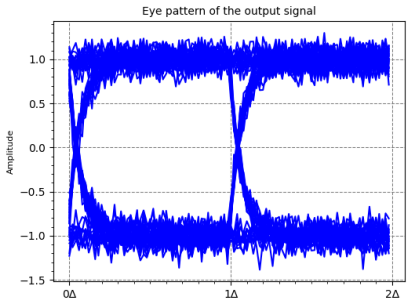


Figure 5.12 – *Oeil bruité avec SNR de 20dB*

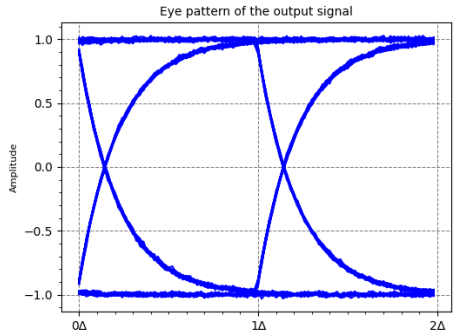


Figure 5.13 – *Oeil non bruité avec SNR de 40dB*

Cette analyse peut être recommencée pour les différents codages et pour différentes situations en incluant de l'interférence et/ou du bruit. Les tableaux 1, 2, 3, 4, 5, 6, 7 et 8 reprennent en compte les quatre possibilités pour chaque code, ainsi que les valeurs des paramètres choisis pour chaque situation indiquée sur les glissières des fenêtres de notre outil de simulation.

Chapitre 6

Feedbacks

Après cette validation de notre outil, nous l'avons mis à disposition des étudiants de troisième bachelier mais également de master qui ont suivi des cours de télécommunications. Nous leur avons laissé l'opportunité de nous transmettre leurs retours vis à vis de cet outil de simulation (Voir annexe A.3.3). Cela permettra d'améliorer les éventuels coquilles qui pourrait avoir dans les simulations, des enchaînements de modifications qui n'ont pas été prévus ou autres. Mais également d'apporter d'avantages d'aide qui seront ajoutées dans le guide d'utilisation. Toutefois, l'échéance de remise de ce rapport étant proche de la date de mise à disposition (une semaine avant), nous n'avons donc pas encore eu de réponses.

Chapitre 7

Perspectives

Dans le but d'améliorer notre application de simulation, plusieurs axes de développement peuvent être envisagés. Dans un premier temps, l'ajout de nouveaux codes tels que les codes à M niveaux ou bien le code duo binaire serait le bienvenu. Générer le signal de départ avec différentes types de sources pourrait constiter une piste prometteuse. Par exemple, l'intégration d'une structure de registres à décalage avec bouclage permettrait de créer une SBPA et d'illustrer un concept clé du cours. L'utilisateur pourrait alors choisir lui même la période de cette séquence et pourrait alors voir l'impact résultant. De plus, la mise en place de fonctionnalités de régénération de symboles pourrait être explorée afin d'analyser la qualité de la transmission et taux d'erreur binaire (BER). Mais cela demandera une collaboration avec le service SECO (Systems, Estimation, Control and Optimisation Unit) pour la gestion de la boucle dans le système de récupération d'horloge PLL. Une possibilité pourrait aussi d'établir l'hypothèse que la récupération d'horloge est connue puisque les signaux à l'émetteur et au récepteur sont connus dans nos programmes et ainsi étudier ensuite le BER par exemple. Ce sont deux manières distinctes de voir les choses. Mais les futurs étudiants pourraient développer par exemple un récepteur qui se placerait après notre transmission et de remplacer notre générateur de signaux par une image. Le but serait ainsi de voir l'impact des différents paramètres d'une transmission sur le BER et de voir la déformation des pixels sur l'image à la réception. En comparant l'image introduite à l'entrée et celle à la réception, les étudiants pourraient ainsi avoir une idée de la qualité de la transmission. Parallèlement, l'intégration de techniques de modulation, telles que la modulation QAM (Quadrature Amplitude Modulation), permettrait d'élargir la gamme des scénarios de simulation et d'approfondir l'étude des performances du système. En outre, il serait également pertinent d'incorporer des outils d'analyse pour visualiser et interpréter les caractéristiques des différents codes utilisés, permettant ainsi aux utilisateurs de comprendre les caractéristiques de chaque type de code mais aussi pour vérifier empiriquement que le bruit introduit par le canal correspond bien à un bruit AWGN. Ces développements offriraient ainsi aux futurs étudiants électriciens de 3ème bachelier la possibilité de poursuivre ce projet en enrichissant davantage l'application et en explorant de nouveaux aspects des télécommunications numériques.

Conclusion

Ce rapport a détaillé le développement et l'évaluation d'un outil de simulation pour l'étude des systèmes de communication numérique en se focalisant sur plusieurs aspects clés tels que la conception, l'architecture, l'implémentation, la validation et les perspectives d'amélioration. La démarche suivie tout au long de ce projet visait à créer un outil capable de modéliser avec précision la transmission en bande de base d'une suite binaire aléatoire dans des conditions variées.

Dans le chapitre de conception, nous avons défini une architecture modulaire pour notre outil, conçue pour séparer les différents composants essentiels tels que la génération de signal par un codage en ligne, la transmission à travers un canal et la détection du signal à la réception. Cette approche modulaire facilite non seulement la maintenance et l'extension de l'outil mais également la validation de chaque composant de manière indépendante. Pour implémenter cet outil, nous avons utilisé des techniques de programmation avancées pour assurer l'efficacité et la précision des simulations. Ensuite, nous avons procédé à une série de tests qui ont constitué une étape cruciale pour s'assurer que notre outil fonctionne conformément aux attentes théoriques. Nous avons comparé les résultats de simulation avec des modèles théoriques en analysant la densité spectrale de puissance, l'interférence intersymbole (ISI) et l'impact du bruit sur les signaux. Les résultats obtenus ont montré une concordance significative avec les théories établies prouvant ainsi la fiabilité et la précision de notre simulateur.

Ce qui fait la force de notre outil est qu'il répond aux besoins des étudiants et permet de visualiser de nombreux concepts du cours théorique. De par sa conception avec le language de programmation Python, il reste modulable facilement et exploitable dans un avenir proche par d'autres étudiants de BAB3 désireux de continuer cette inovation. L'outil peut encore être amélioré en générant différentes sources, d'autres codages et voir utiliser des compétences de contrôle de systèmes pour arriver à réaliser une simulation de la récupération d'horloge par boucle à verrouillage de phase (Phase-Locked Loop). En intégrant ces améliorations, notre simulateur deviendra un outil encore plus puissant et polyvalent pour l'étude et la conception de systèmes de communication numérique offrant aux futurs étudiants et professionnels un support précieux pour leurs travaux et innovations.

En conclusion, notre étude de validation et de tests a démontré que notre outil de simulation est capable de reproduire fidèlement les résultats théoriques pour une variété de codages numériques et de paramètres variables dynamiquement. En offrant une plateforme intuitive et modulaire, cet outil facilite la compréhension des concepts complexes des télécommunications et ouvre de nouvelles perspectives pour la recherche et l'innovation dans ce domaine.

Bibliographie

[1] H. Arja. Virtuosity : New eye diagram measurements. https://community.cadence.com/cadence_blogs_8/b/cic/posts/virtuosity-eye-diagram-measurements, 2018.

[2] Z. L. Arturo and A. C. Carlos. Digital signal processing course on jupyter–python notebook for electronics undergraduates. *Comp Applic In Engineering*, 28 :1045–1057, 2020. doi : 10.1002/cae.22277.

[3] T. Dutoit. Cours de "théorie des circuits", 2023. Université de Mons, Faculté polytechnique.

[4] E. Gopi. *Digital Signal Processing for Wireless Communication using Matlab*. Springer International Publishing, 2016.

[5] J. Hunter, D. Dale, E. Firing, M. Droettboom, and the Matplotlib development team. matplotlib.animation.animation. https://matplotlib.org/stable/api/_as_gen/matplotlib.animation.Animation.html, 2024.

[6] J. Hunter, D. Dale, E. Firing, M. Droettboom, and the Matplotlib development team. matplotlib.widgets. https://matplotlib.org/stable/api/widgets_api.html, 2024.

[7] J. W. Leis. *Communication sytems principles using Matlab*. John Wiley Sons, Inc., 2018.

[8] V. Mathuranathan. Construct eye diagram from stored signal samples. https://www.gaussianwaves.com/2018/11/construction-of-eye-diagrams-from-signals-represented-in-computer-memory/, 2018.

[9] V. Mathuranathan. *Digital Modulations using Python*. 2019.

[10] G. Matten, J.-M. Friedt, and E. Carry. Prototypage de traitements numériques de signaux au moyen d'un outil libre : Gnuradio. *J3eA*, 14 :1011, 2015. doi : 10.1051/j3ea/2015011.

[11] J. G. Proakis and M. Salehi. *Contemporary communication systems using Matlab*. PWS Publishing Company, 1998.

[12] E. Rius and C. Quendo. Simulation sous matlab/simulink d'une chaîne de transmission numérique en bande de base. application au codage canal. *J3eA*, 2 :14, 2003. doi : 10.1051/bib-j3ea:2003014.

[13] M. Wuilpart and V. Moeyaert. Cours de "principes des télécommnications numériques", 2023. Université de Mons, Faculté polytechnique.

Annexes

A.1 Figures

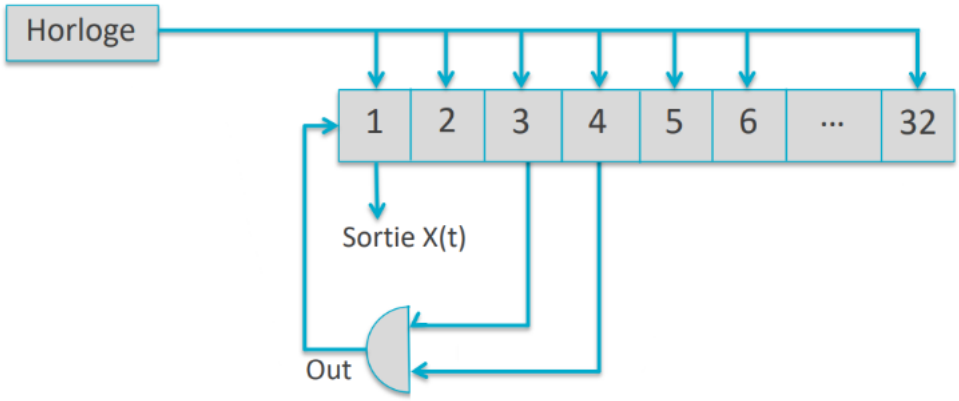


Figure A.1 – *Séquence Binaire Pseudo-Aléatoire* [13]

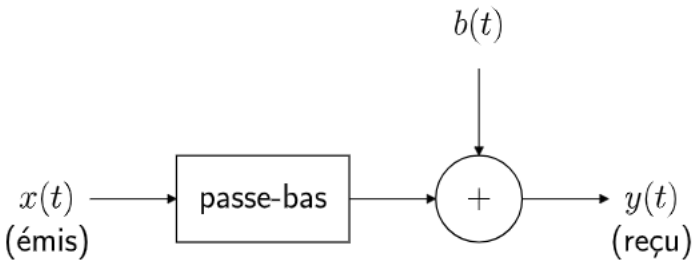


Figure A.2 – *Modélisation du canal*

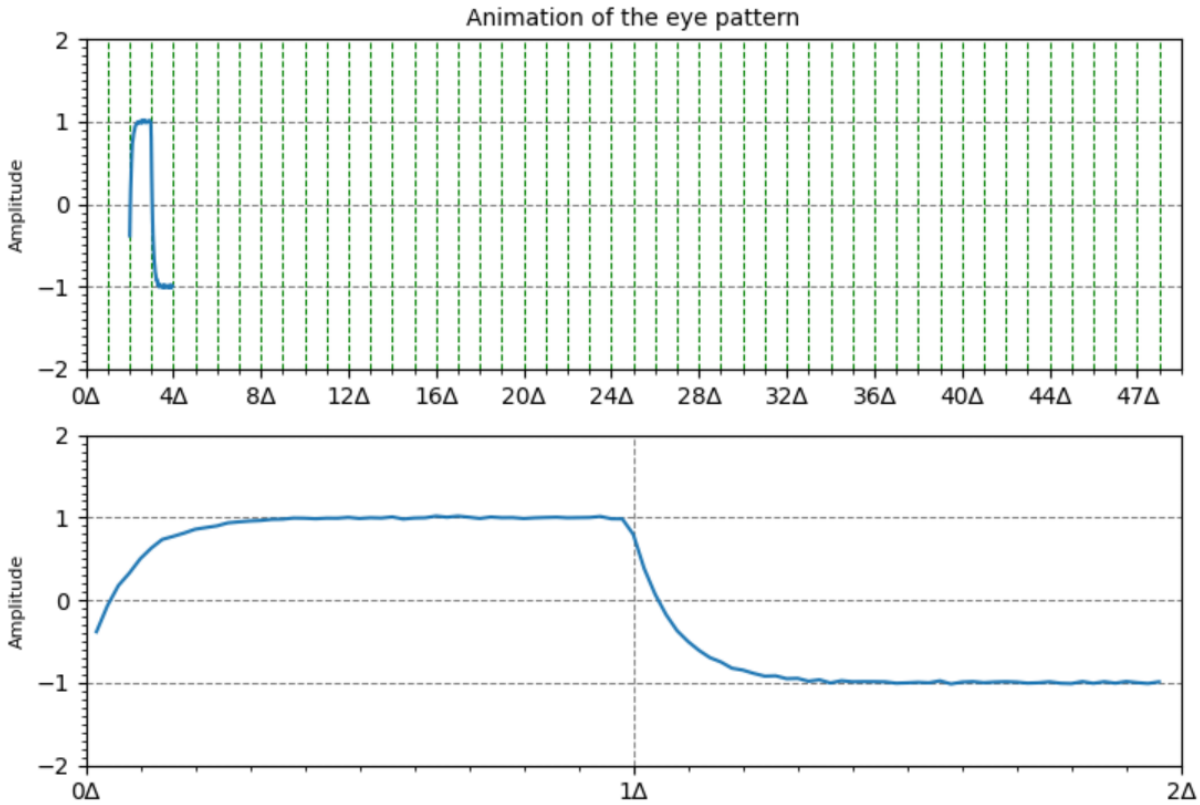


Figure A.3 – *1ère étape de la construction du diagramme de l'oeil*

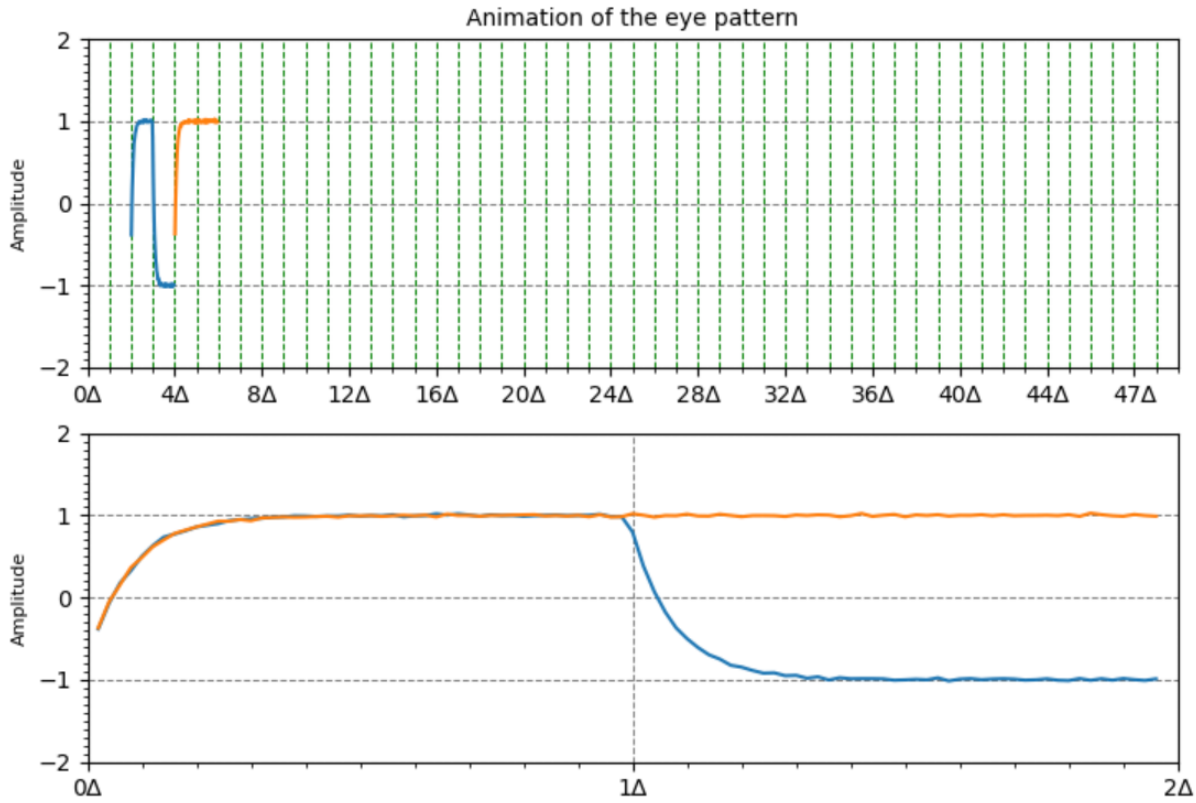


Figure A.4 – *2ème étape de la construction du diagramme de l'oeil*

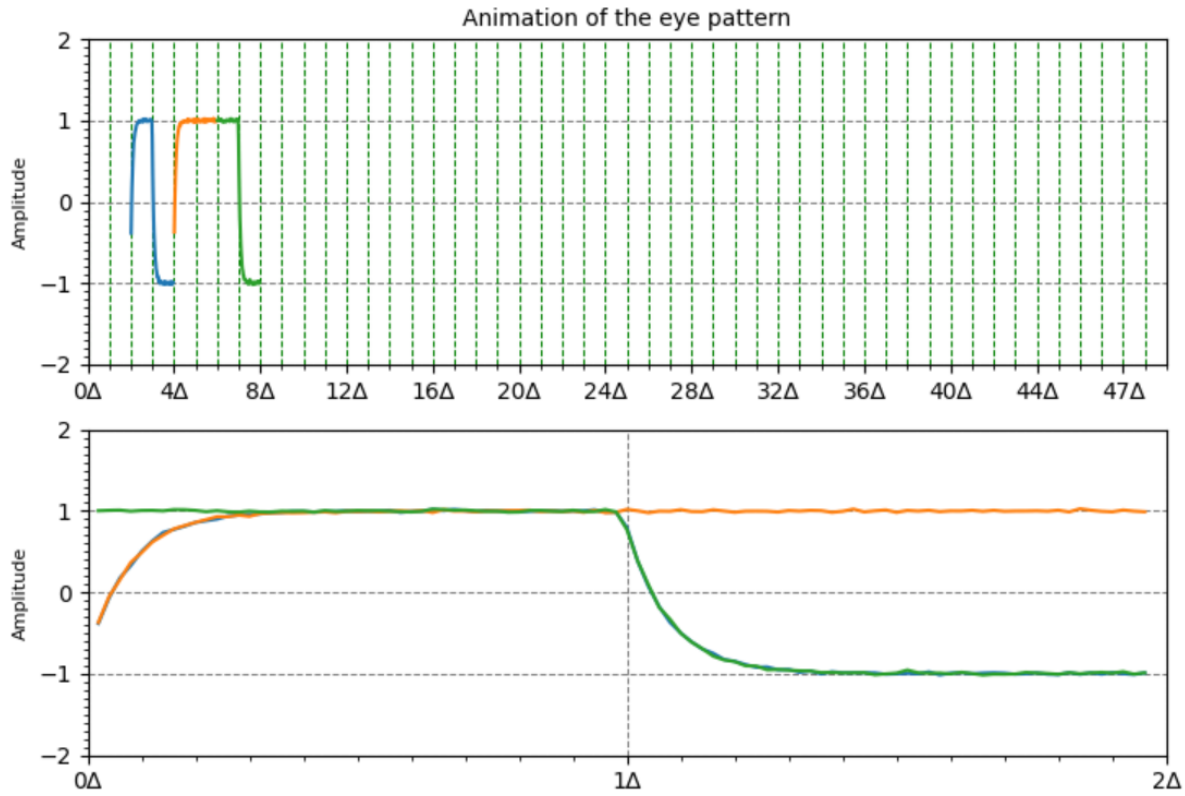


Figure A.5 – *3ème étape de la construction du diagramme de l'oeil*

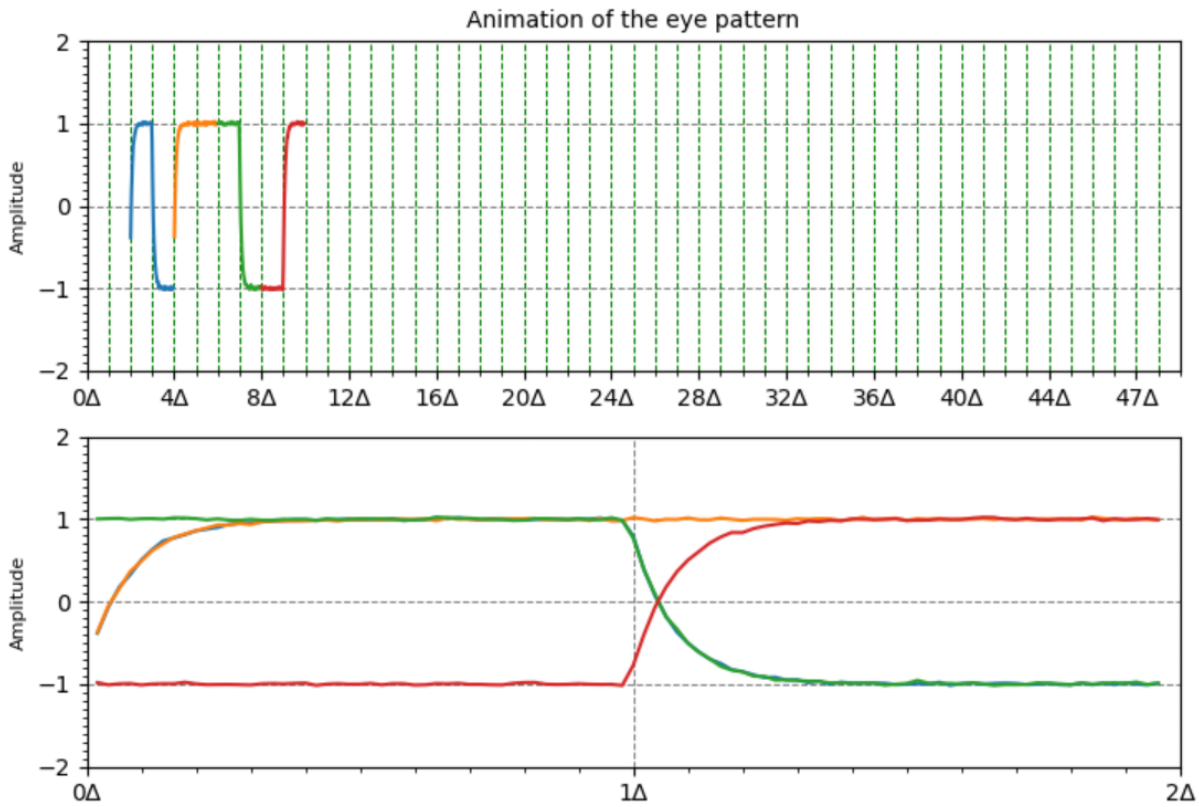


Figure A.6 – *4ème étape de la construction du diagramme de l'oeil*

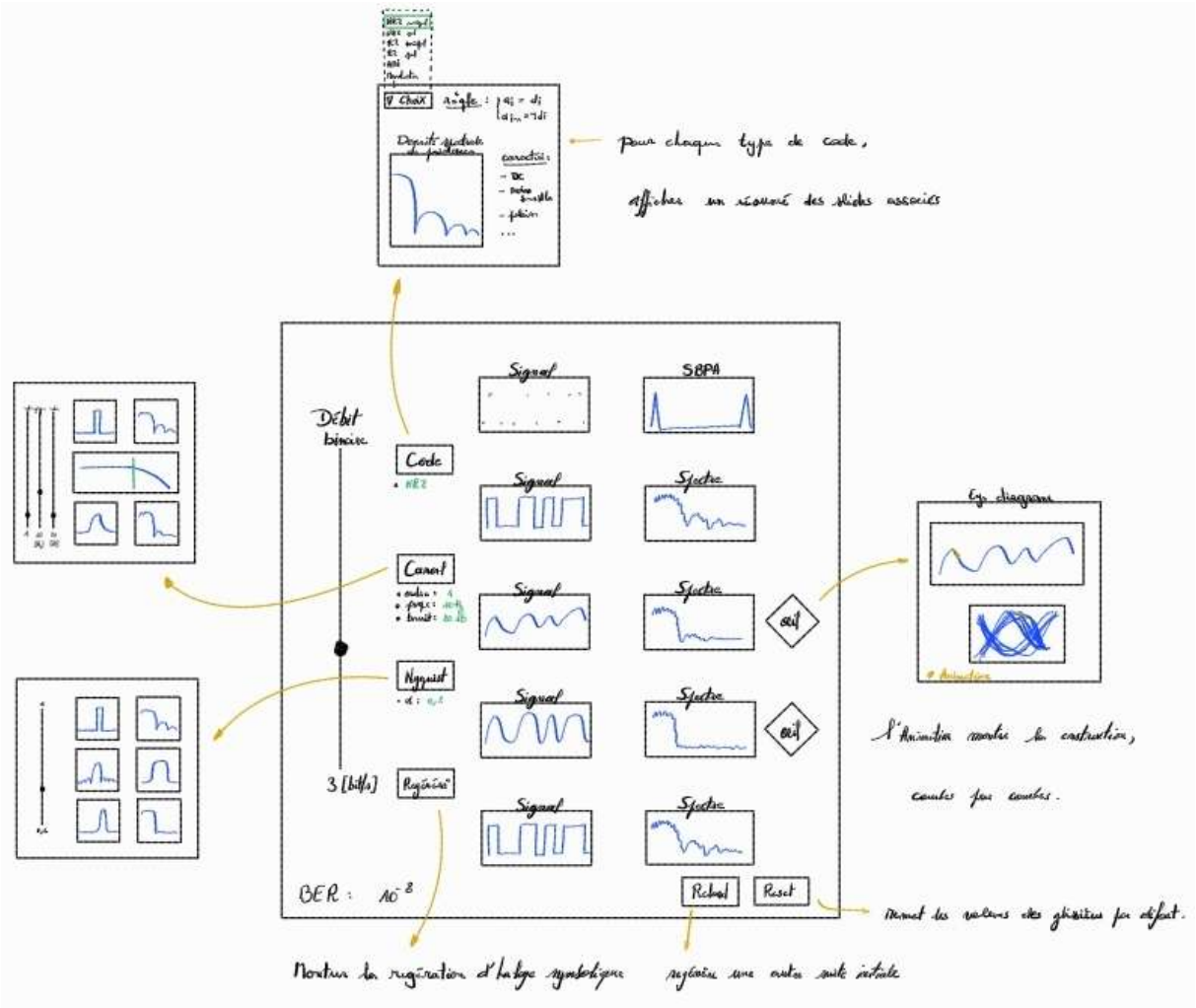


Figure A.7 – *Croquis de l'architecture de l'application de simulation*

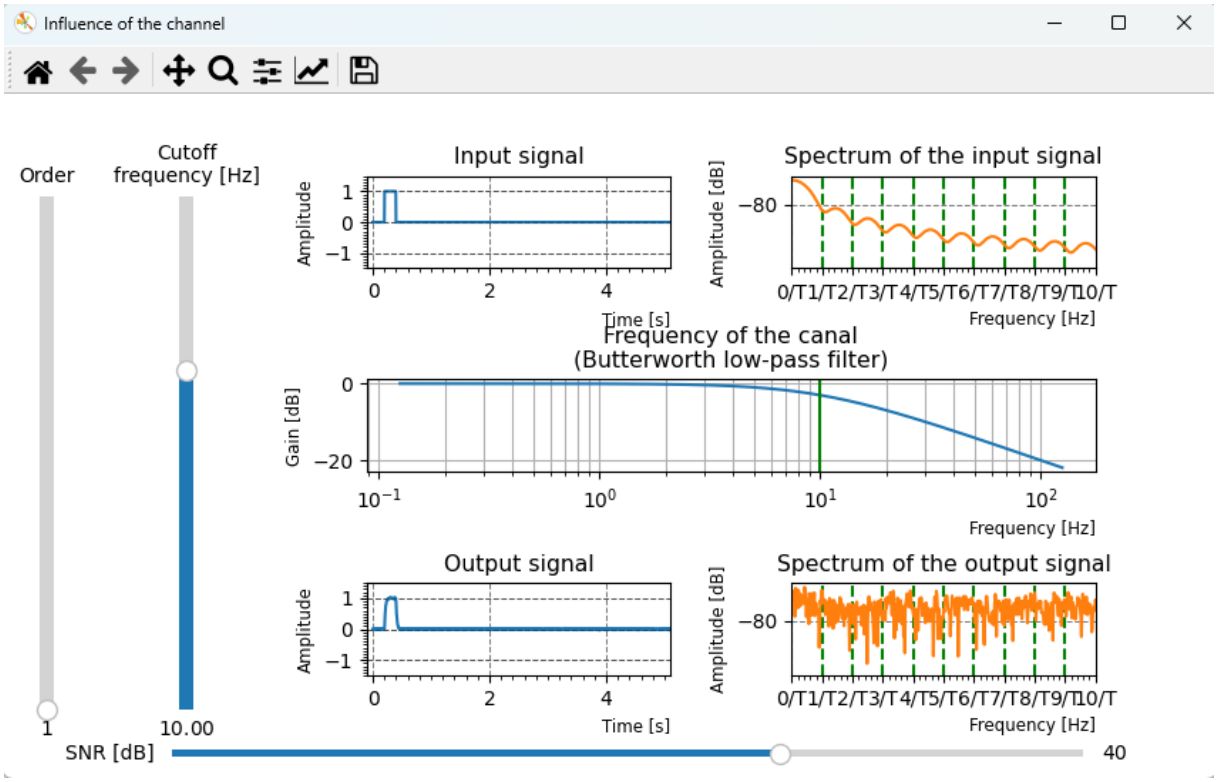


Figure A.8 – *Fenêtre - Influence of the channel*

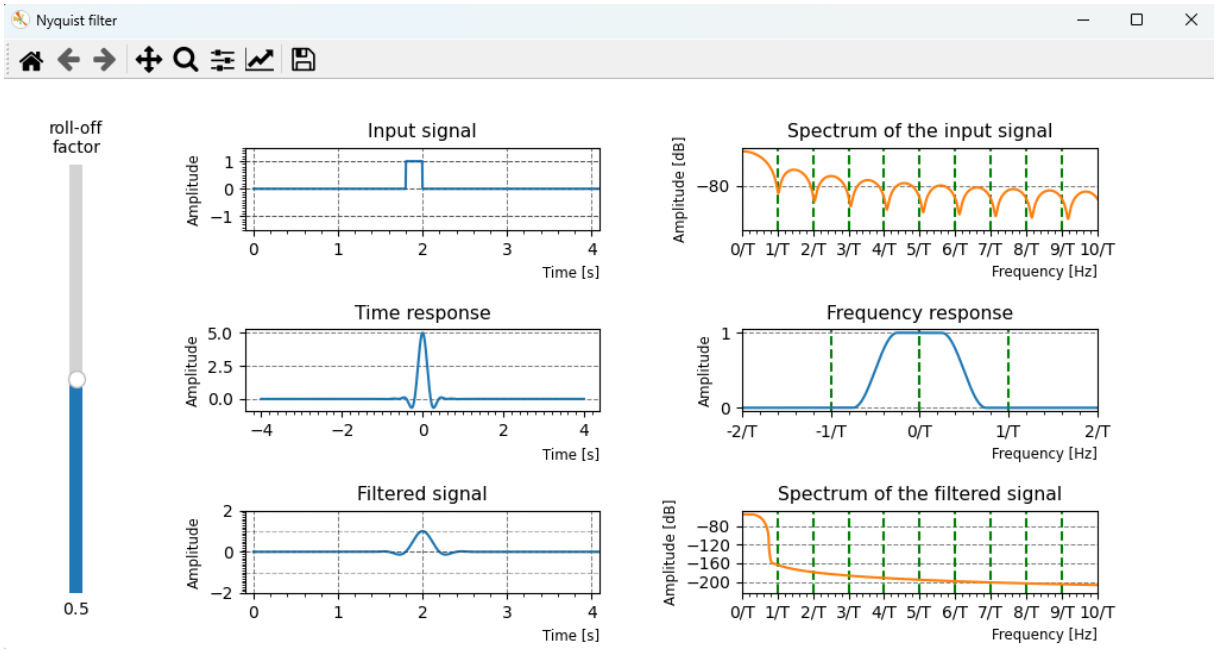


Figure A.9 – *Fenêtre - Nyquist filter*

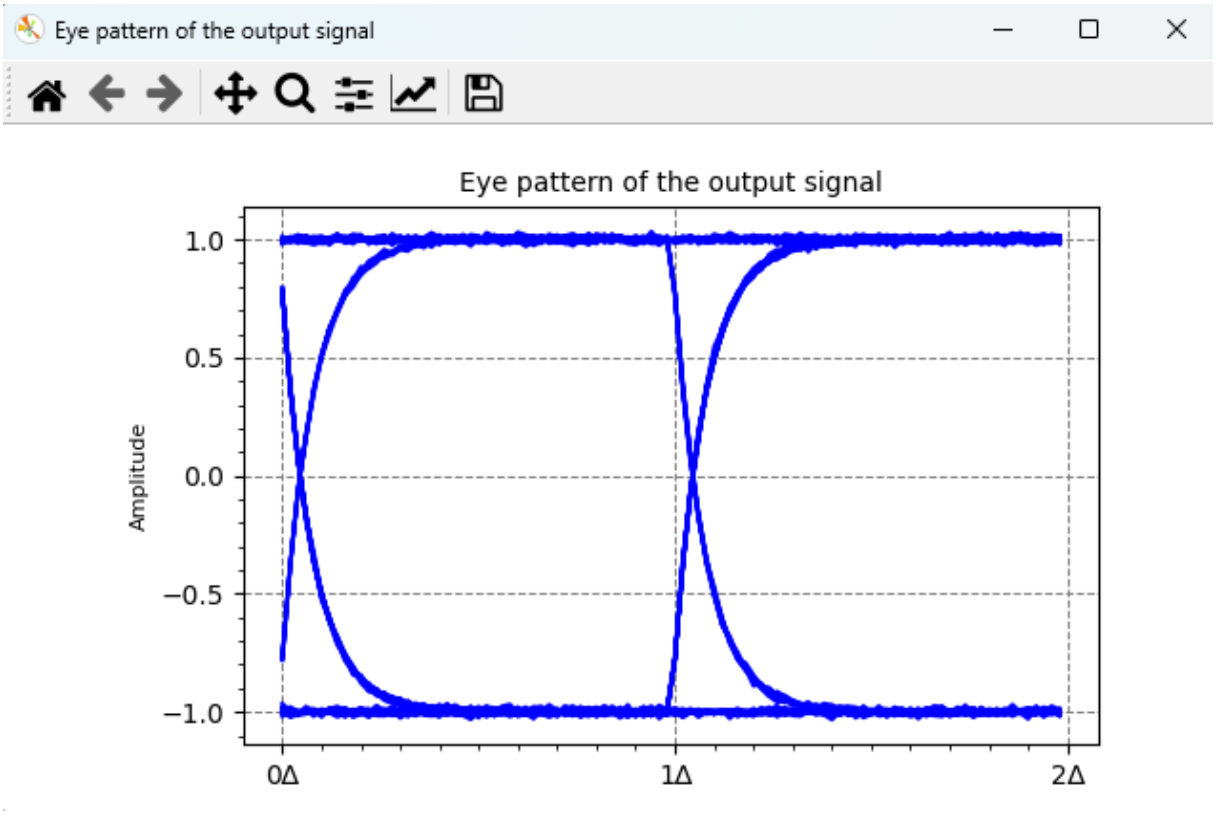


Figure A.10 – *Fenêtre - Eye pattern of the output signal*

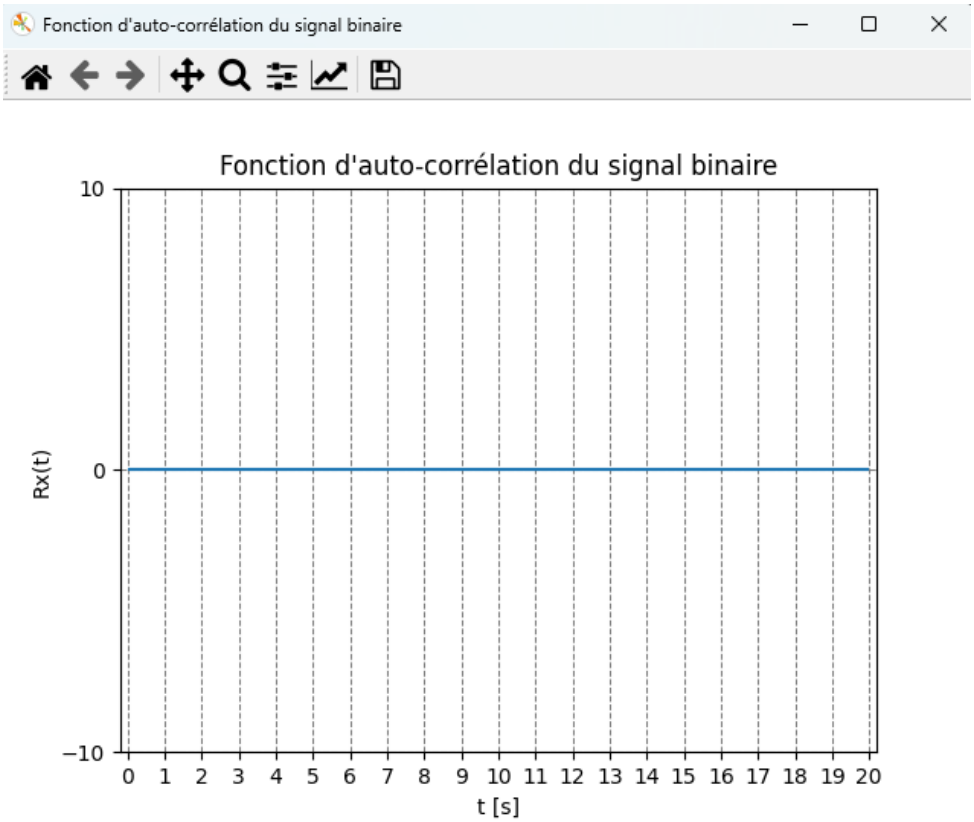


Figure A.11 – *Autocorrélation nulle d'une suite binaire avec un débit de 10 [bit/s]*

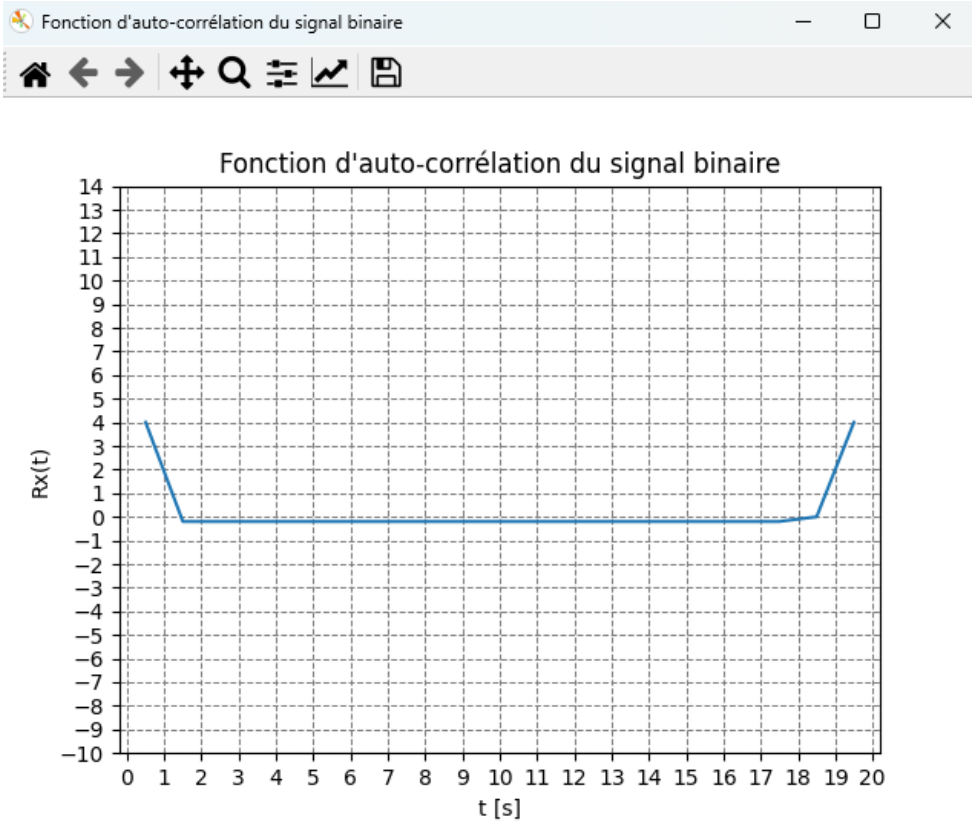


Figure A.12 – *Autocorrélation de la suite binaire avec un débit de 1 [bit/s]*

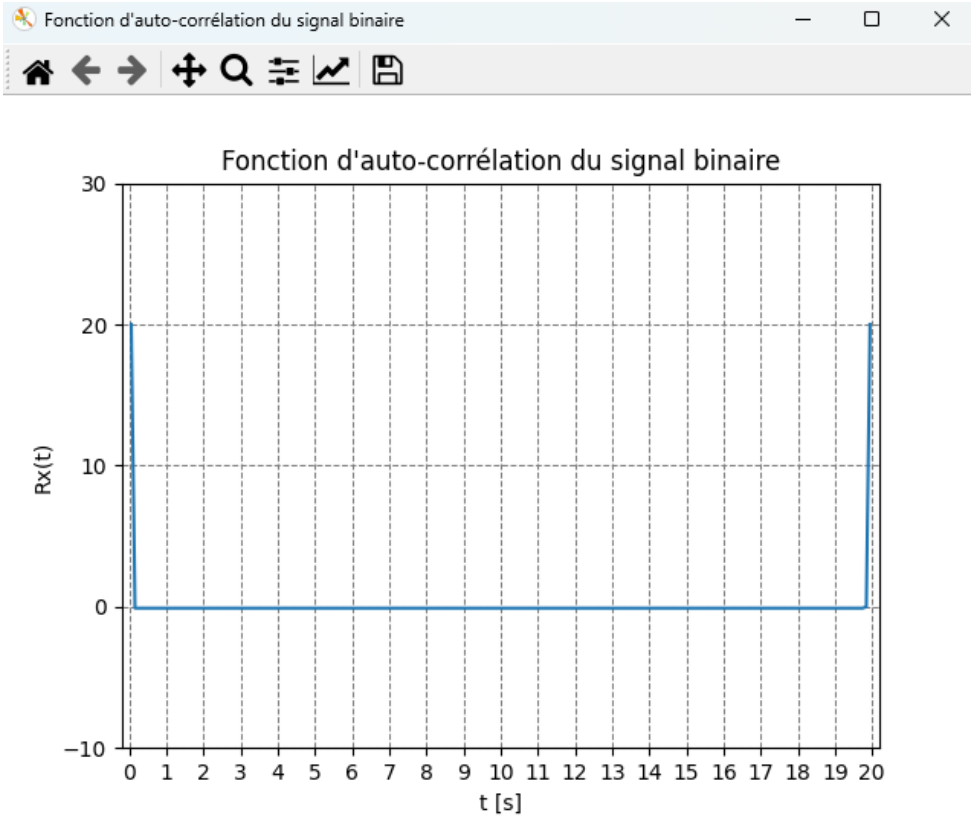


Figure A.13 – *Autocorrélation de la suite binaire avec un débit de 10 [bit/s]*

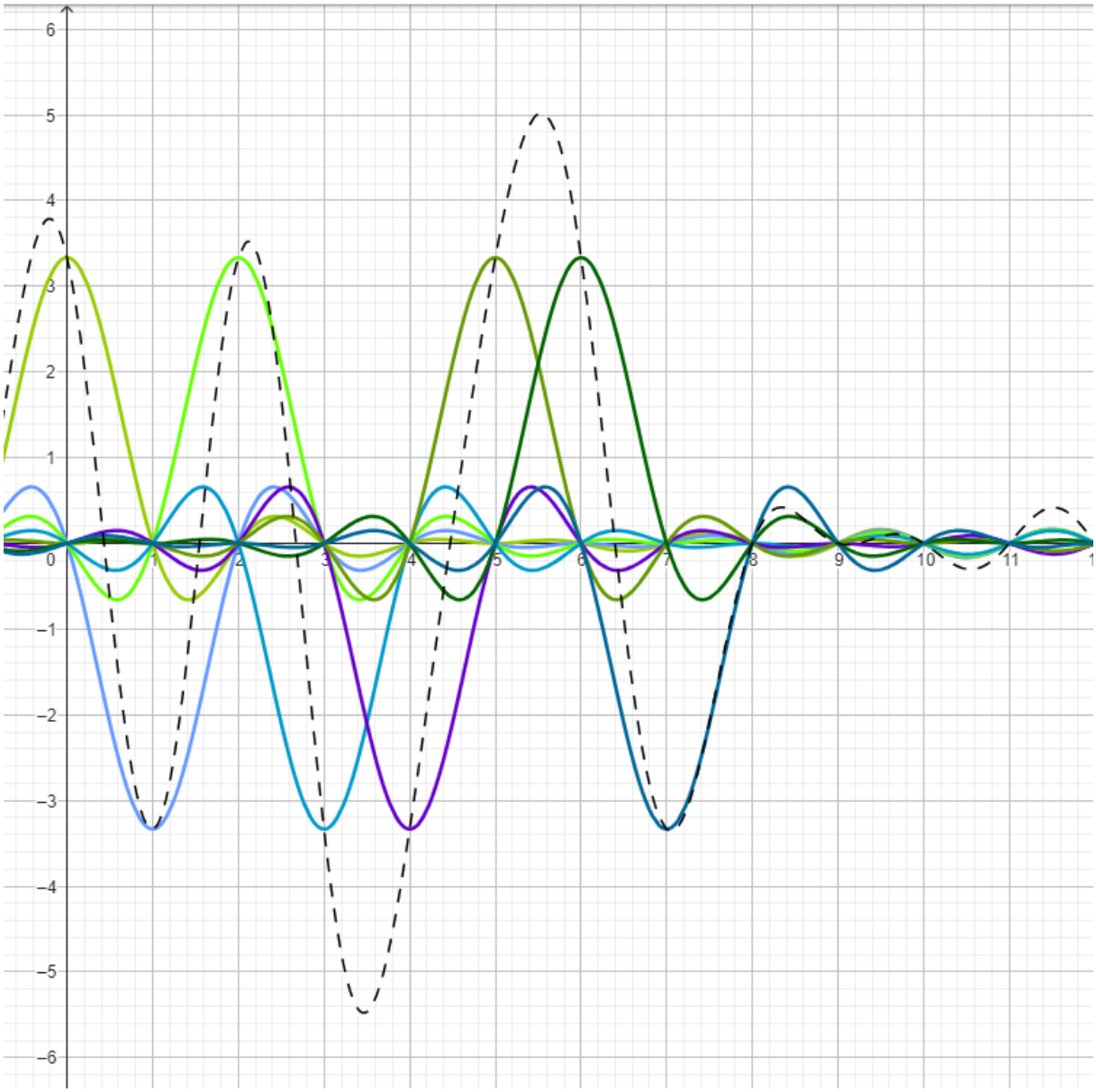


Figure A.14 – *Convolution du signal avec le filtre de Nyquist (Facteur de roll-off = 0.1, Δ = 0.3 s)*

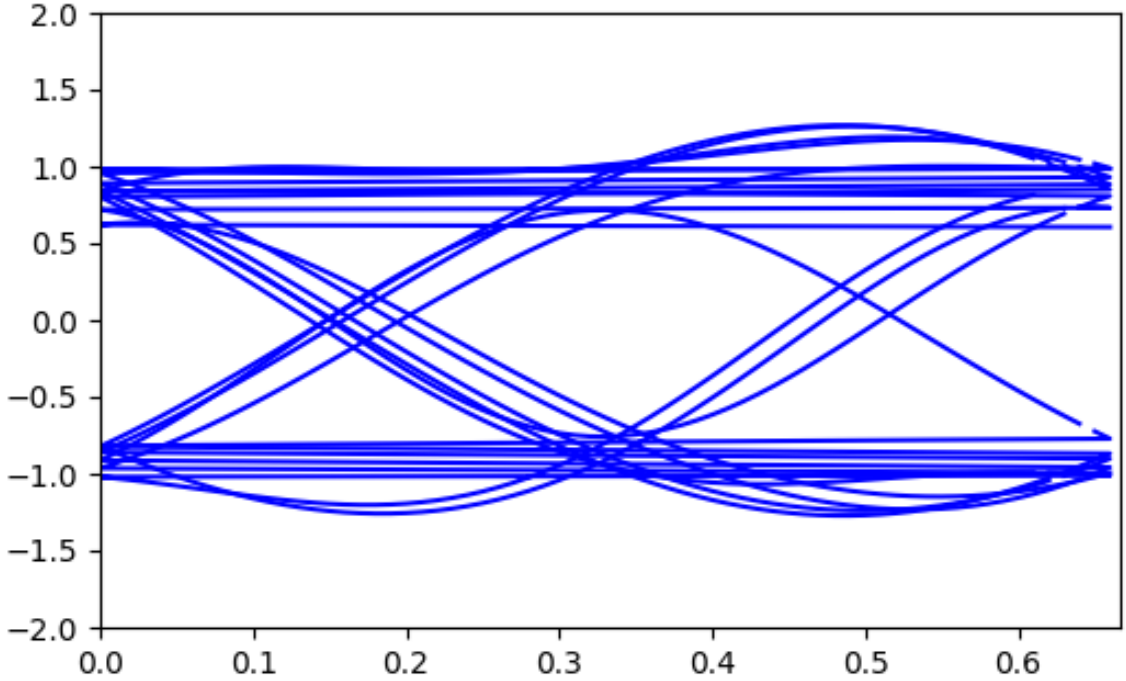


Figure A.15 – *Animation du diagramme de l'oeil non réussie*

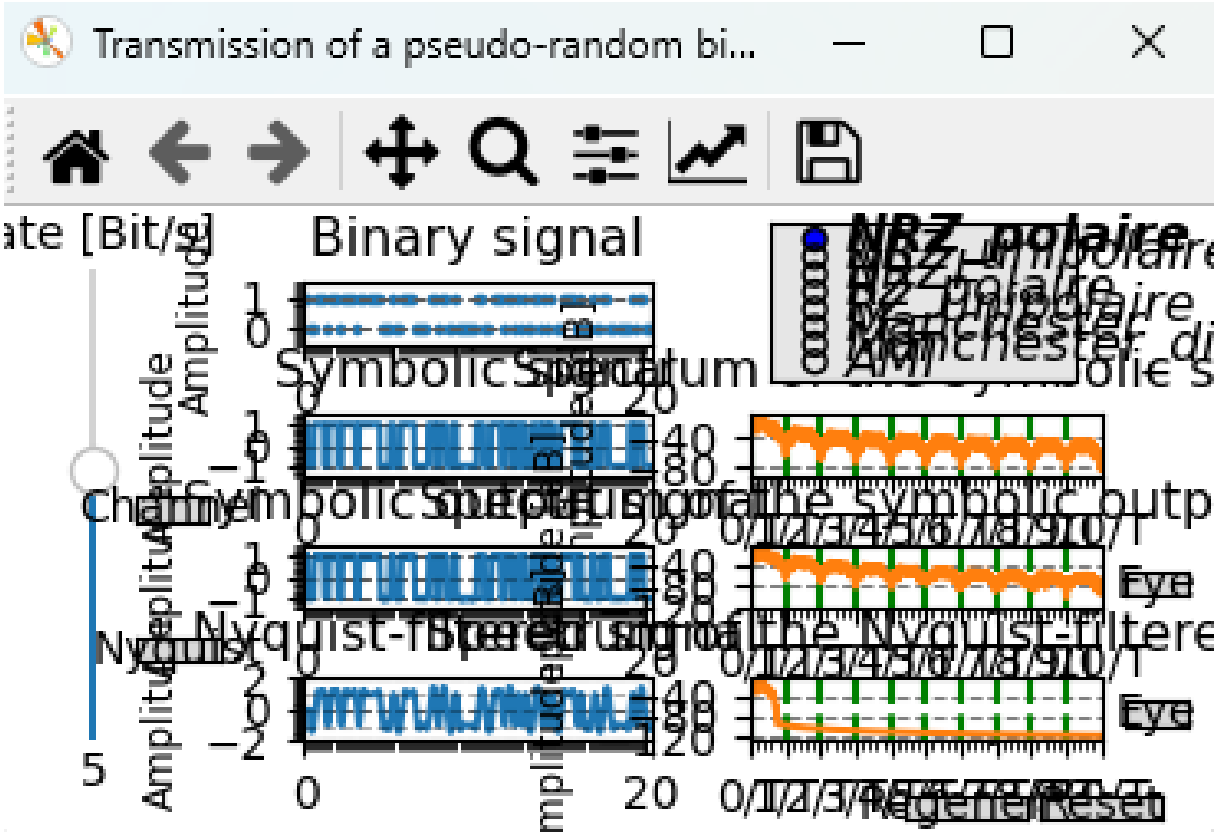


Figure A.16 – *Page d'accueil avec des dimensions non-adéquates*

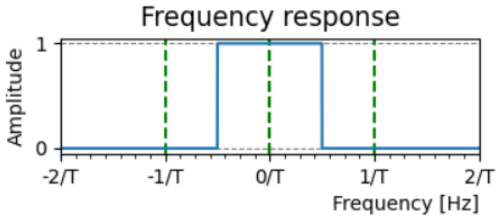


Figure A.17 – *Filtre de Nyquist en pratique pour* $\alpha = 0$

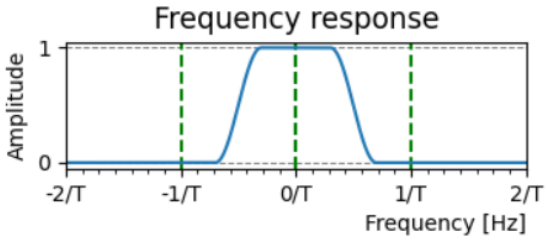


Figure A.18 – *Filtre de Nyquist en pratique pour* $\alpha = 0.4$

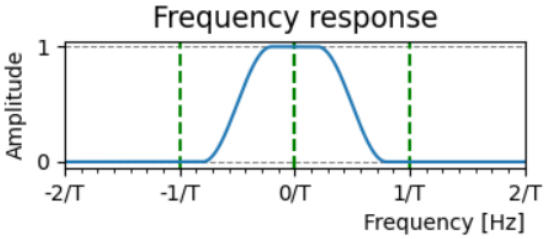


Figure A.19 – *Filtre de Nyquist en pratique pour* $\alpha = 0.6$

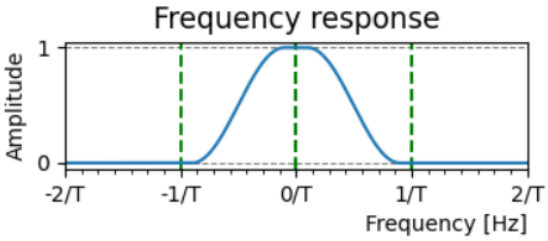


Figure A.20 – *Filtre de Nyquist en pratique pour* $\alpha = 0.8$

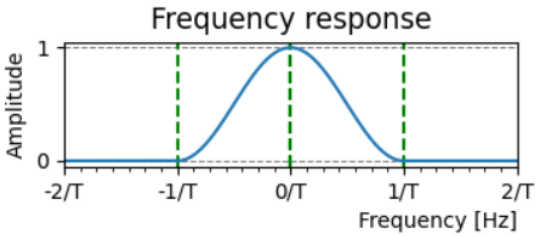


Figure A.21 – *Filtre de Nyquist pour* $\alpha = 1$

A.2 Tableaux

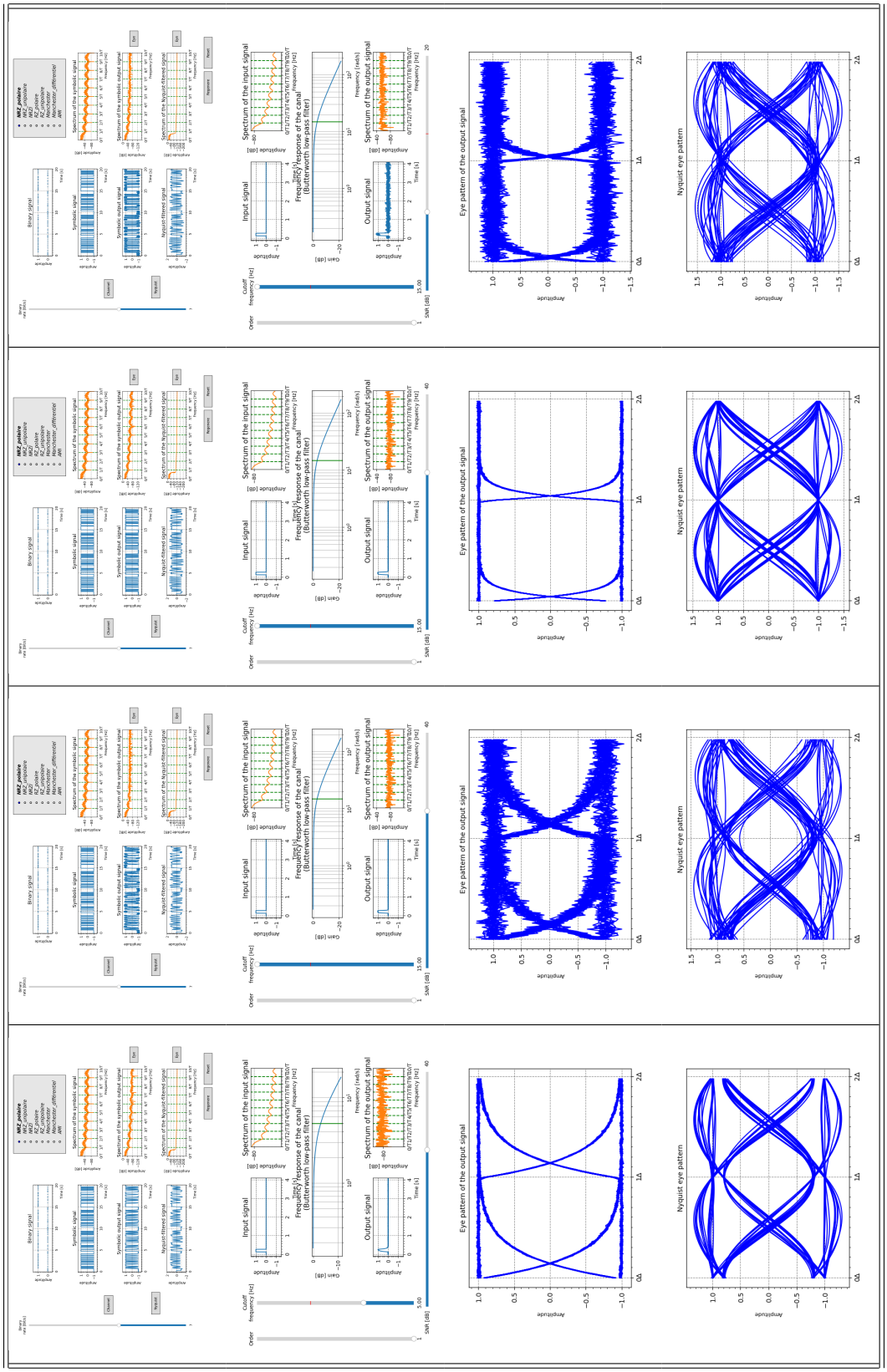


Table 1 – *Panel du NRZ polaire*

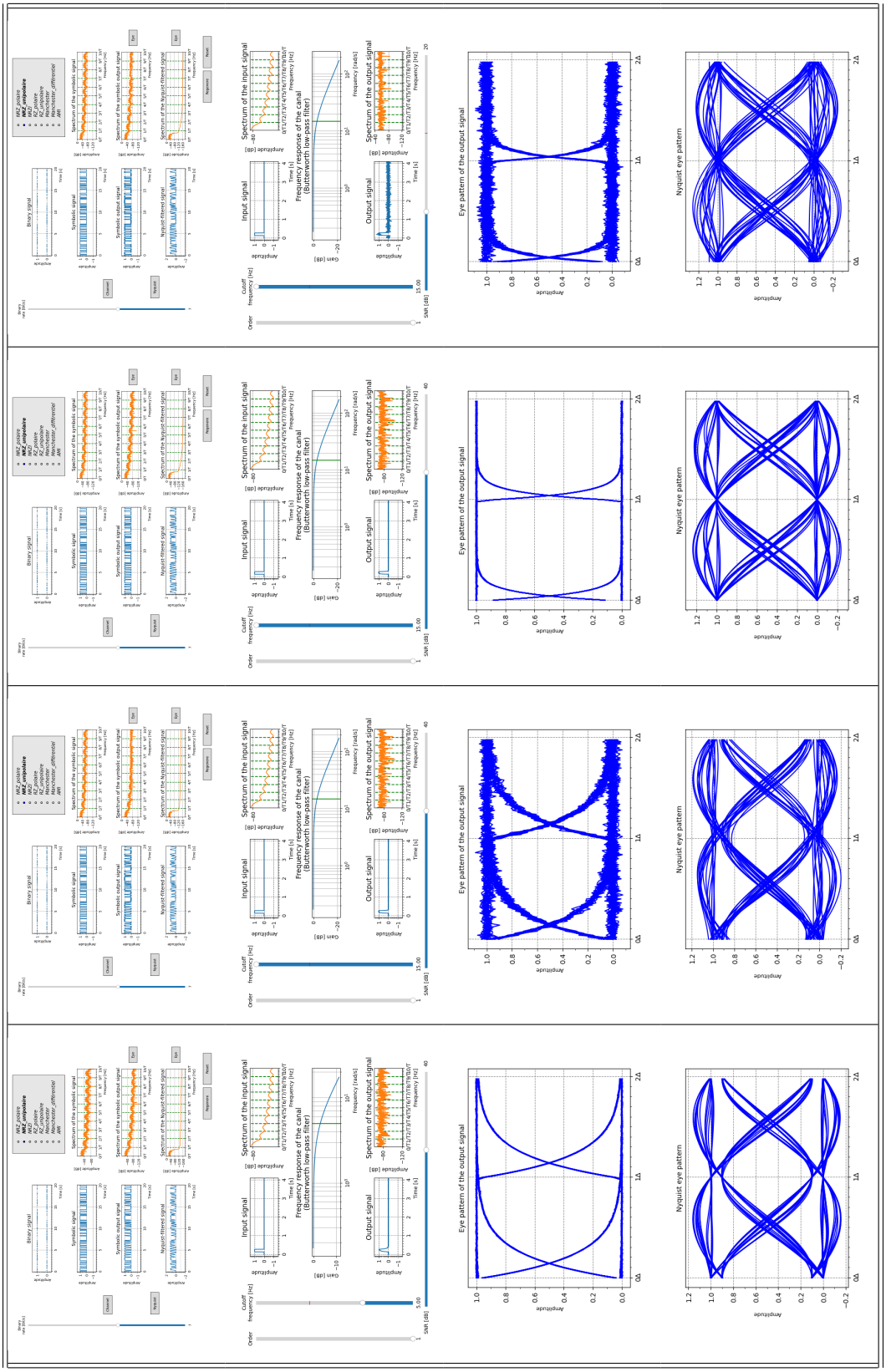


Table 2 – *Panel du NRZ unipolaire*

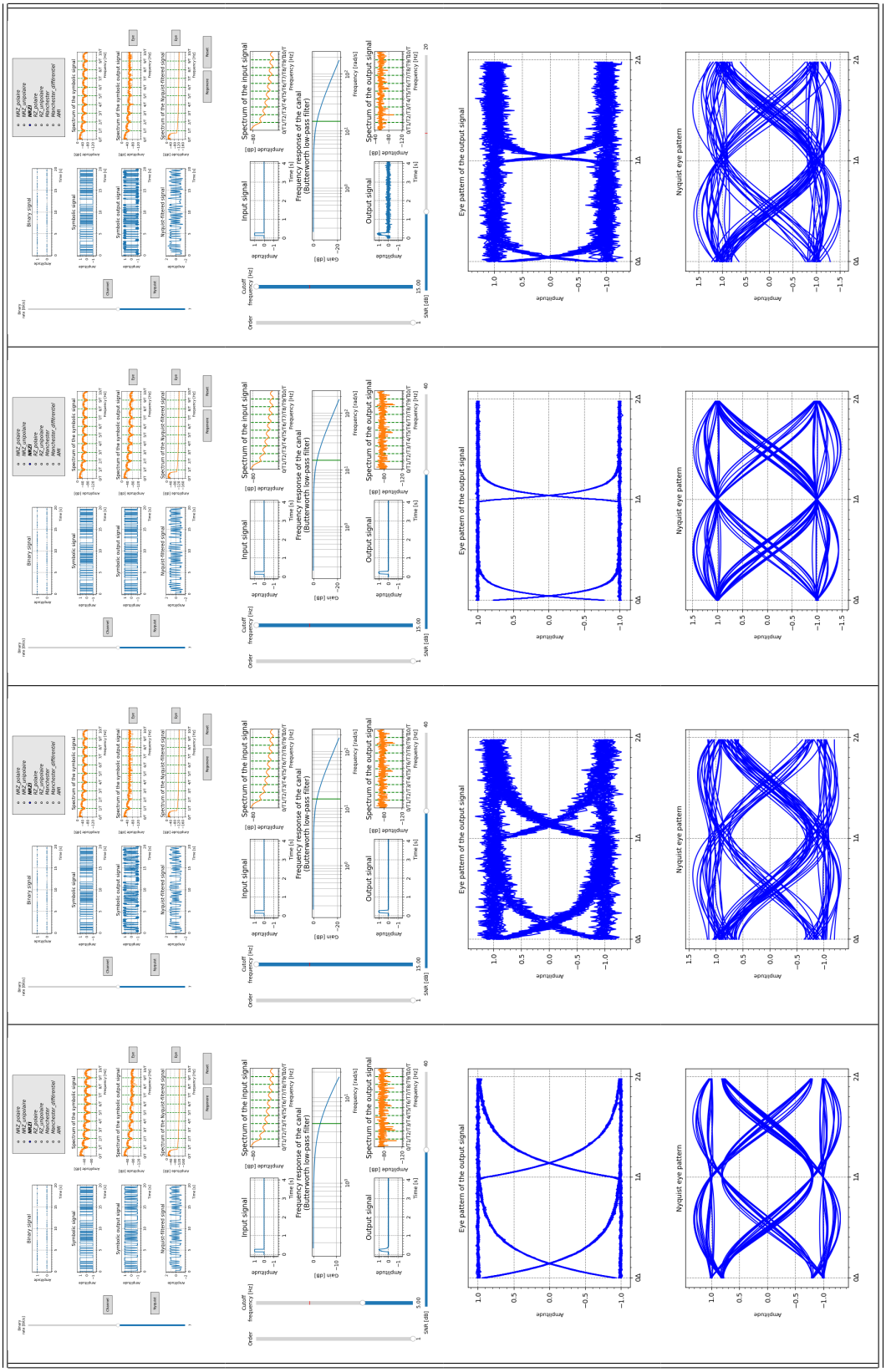


Table 3 – *Panel du NRZI*

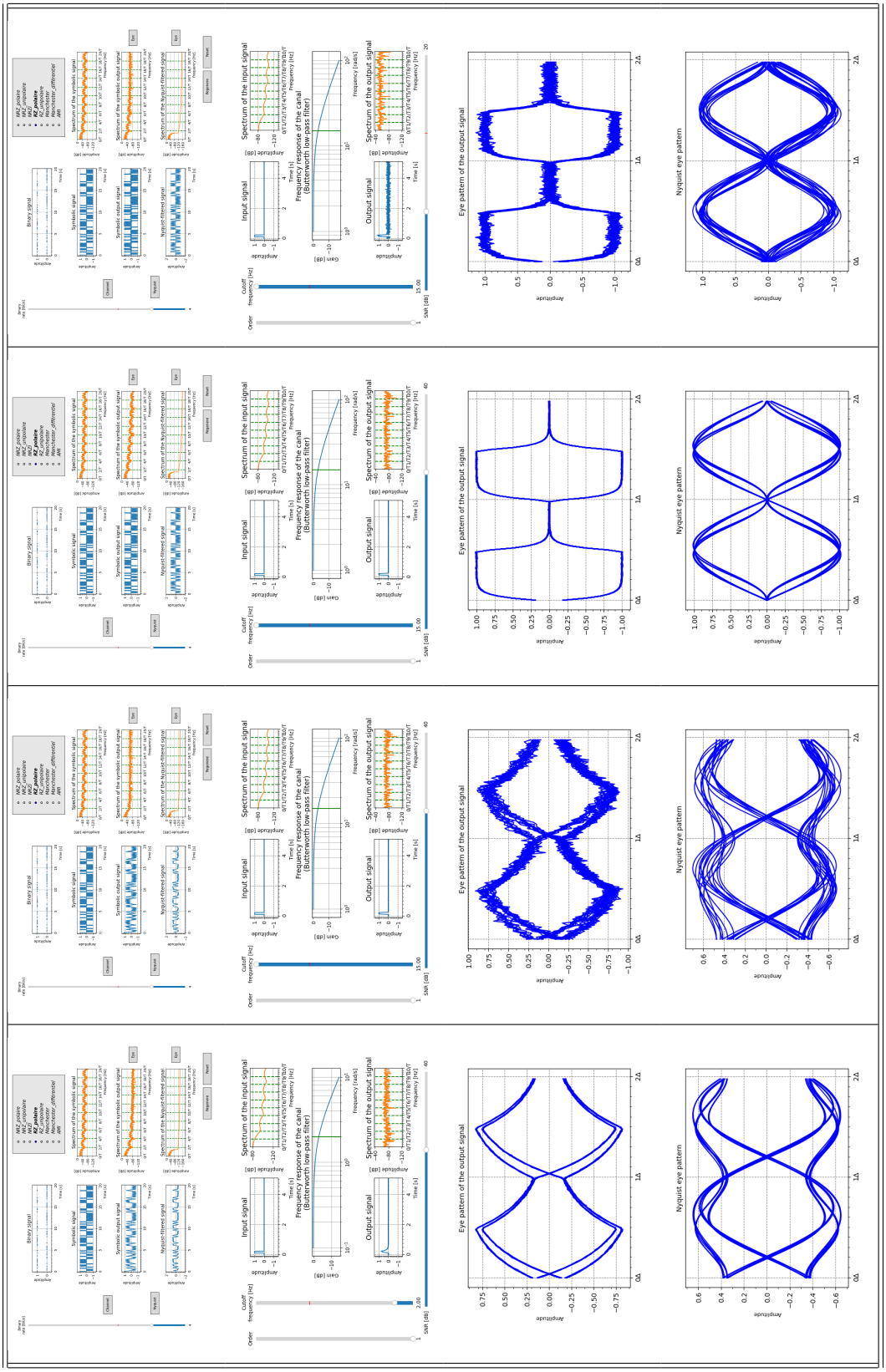


Table 4 – *Panel du RZ polaire*

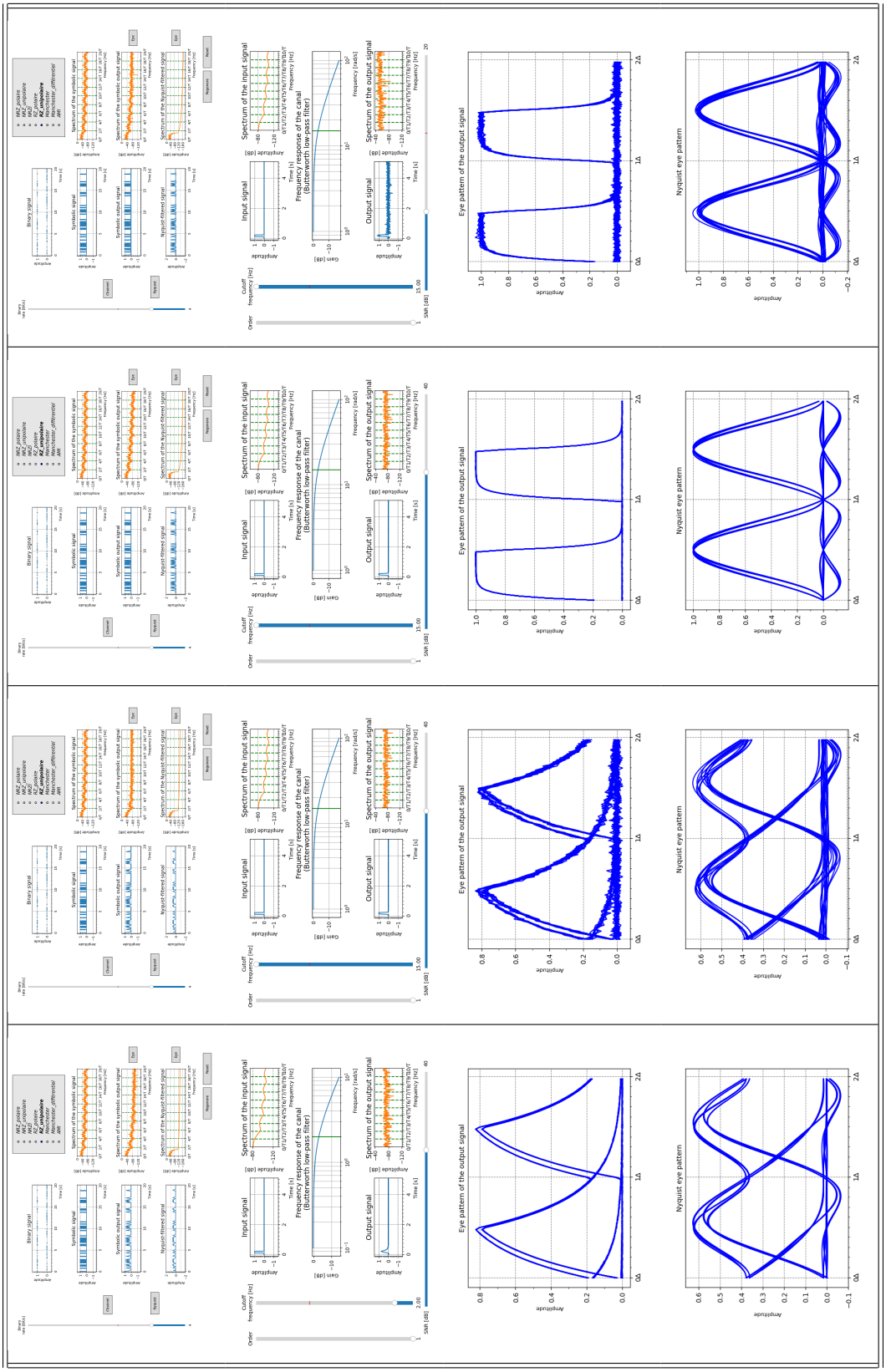


Table 5 – *Panel du RZ unipolaire*

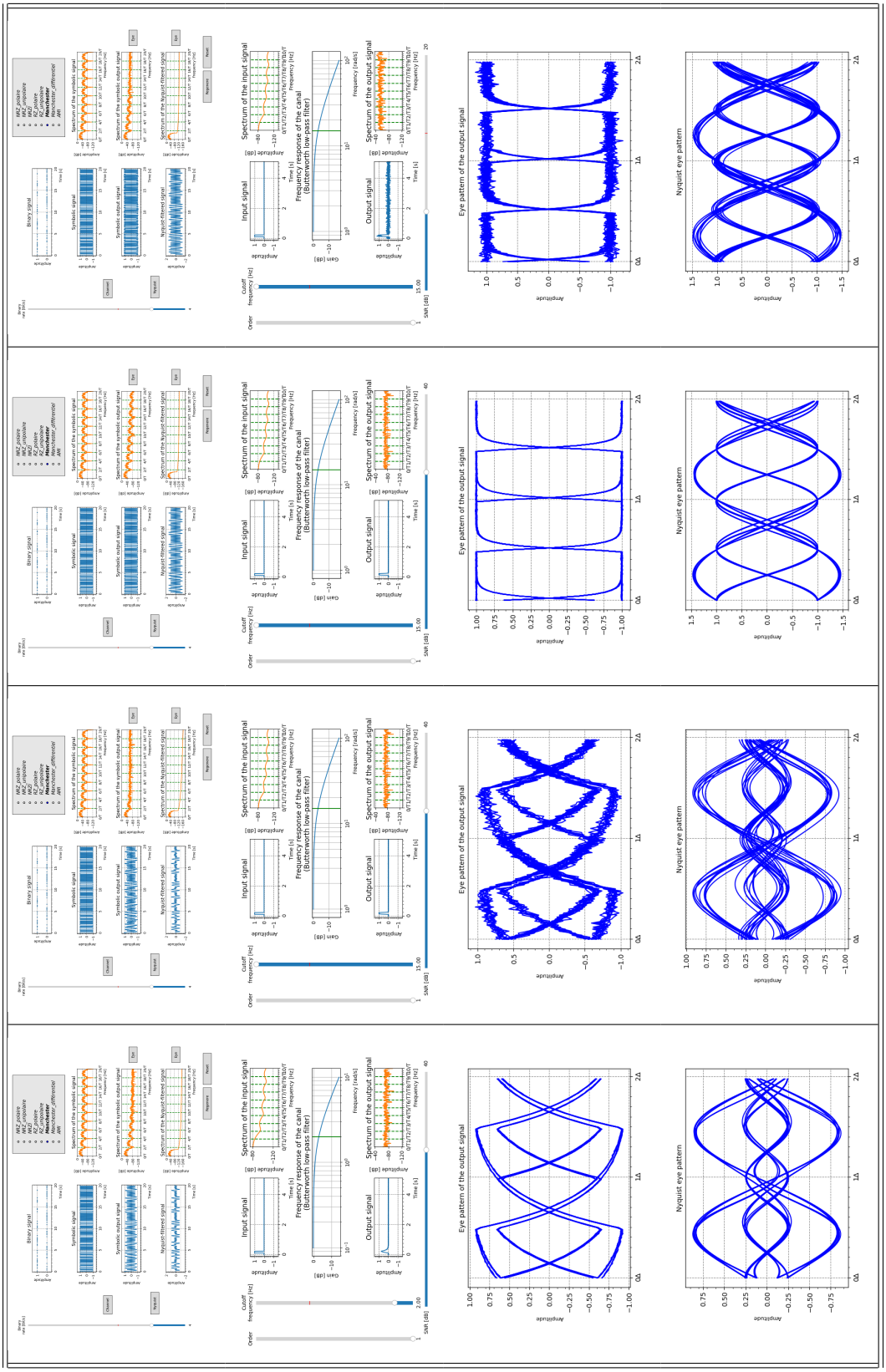


Table 6 – *Panel du Manchester*

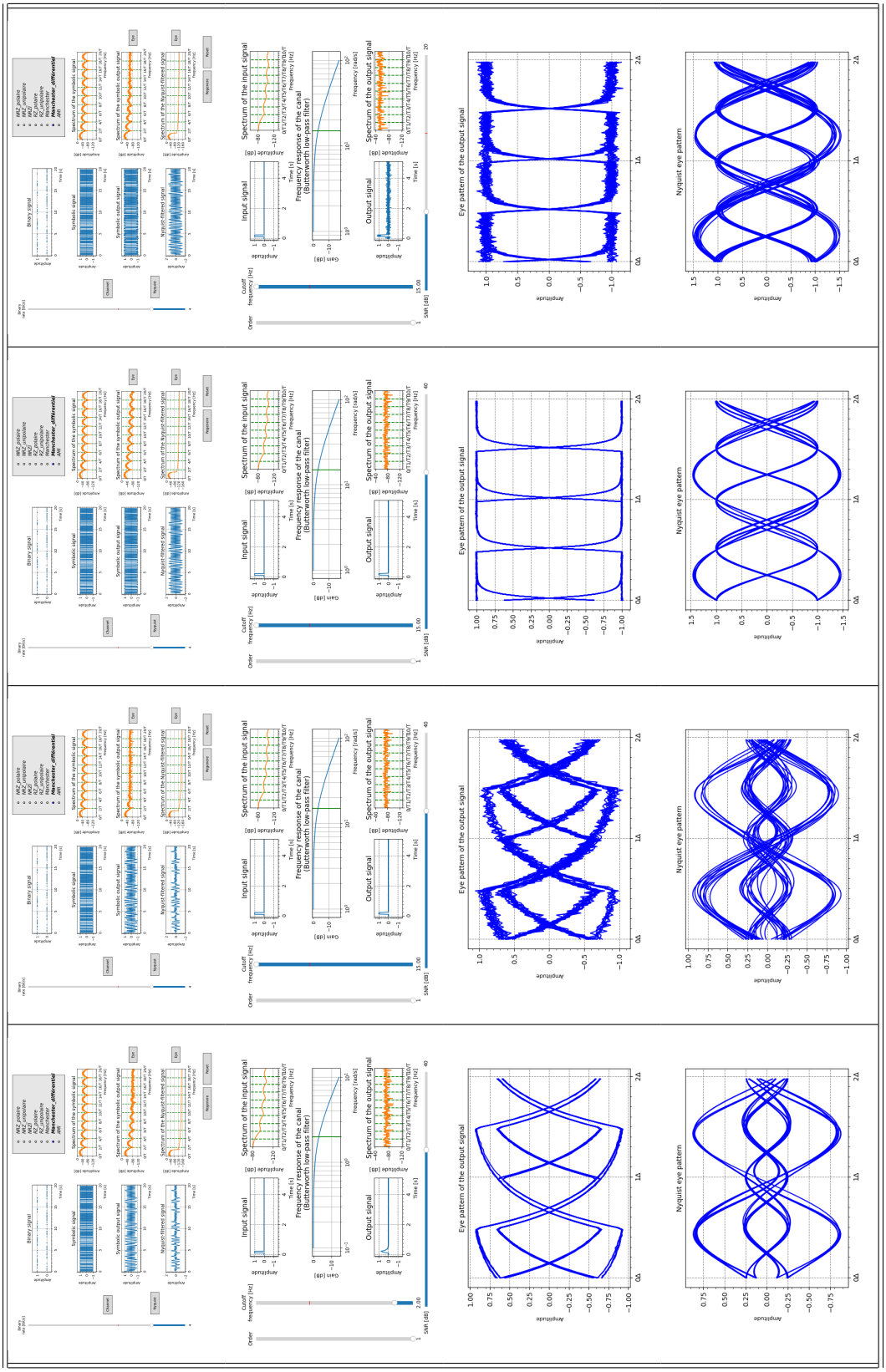


Table 7 – *Panel du Manchester différentiel*

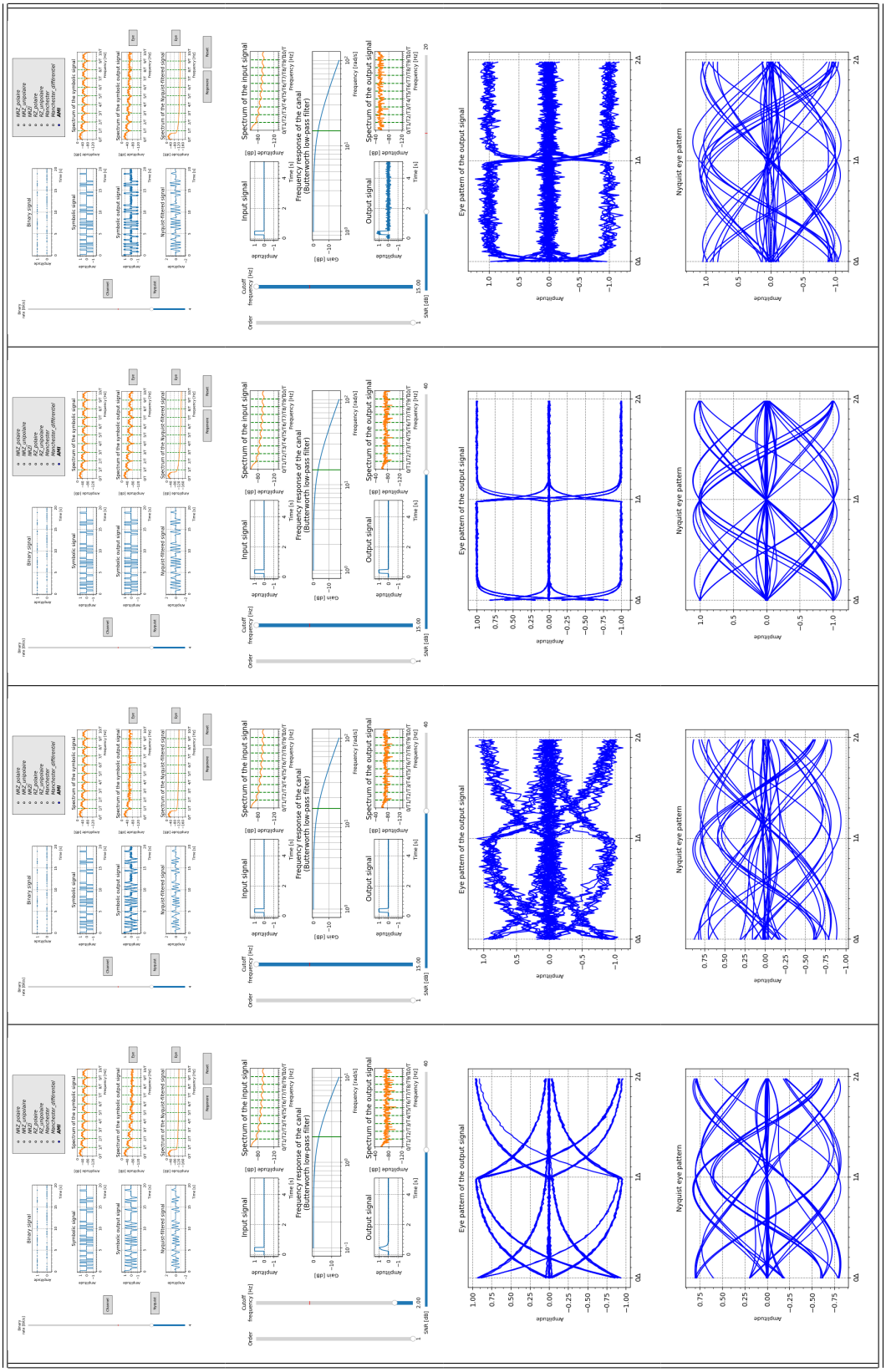


Table 8 – *Panel du AMI*

A.3 Documents

A.3.1 Google Forms

Projet BAB3 - Outil pédagogique pour le cours de télécommunications

Nous cherchons à savoir quels points du cours étaient les plus "problématiques". Pour ce faire, nous vous demandons de compléter ce formulaire pour nous donner des améliorations potentielles sur les éléments qui méritent un peu plus de développements/illustrations dynamiques.

* Indique une question obligatoire

Adresse e-mail *

Impossible de préremplir l'adresse e-mail

Donnez 3 points de matière au minimum où une explication par illustration/graphique dynamique vous aurait offert une meilleure compréhension du concept. Précisez bien quel genre d'illustrations en plus vous auriez aimé avoir. (Par exemple, évolution temporelle d'un signal ou animation dynamique d'un concept en temps réel) Vous pouvez mentionner le chapitre/slide concerné. Classez vos 3 propositions par ordre d'importance

Choix n°1 *

Votre réponse

Choix n°2 *

Votre réponse

Préremplissez les réponses, puis cliquez sur "Obtenir le lien"

Choix n°3 *

Votre réponse

Choix supplémentaire(s) facultatif(s)

Votre réponse

Comment voulez-vous que le support soit disponible? *

- ○ Sous forme de code Jupyter Notebook à compléter et faire varier les paramètres à votre guise
- ○ Sous forme d'une application .exe téléchargeable sur vos appareils
- ○ Uniquement un support utile pour le professeur pendant les cours
- ○ Autre :

Comment voudriez-vous que le support soit utilisé? *

- ☐ En cours par vous ou l'enseignant
- ☐ Chez vous

Remarque(s) supplémentaire(s) :

Votre réponse

Obtenir le lien

Page 1 sur 1

N'envoyez jamais de mots de passe via Google Forms.

Préremplissez les réponses, puis cliquez sur "Obtenir le lien"

A.3.2 Notice d'utilisation

UMONS Université de Mons

TeleSimU

POLYTECH MONS

Notice d'utilisation

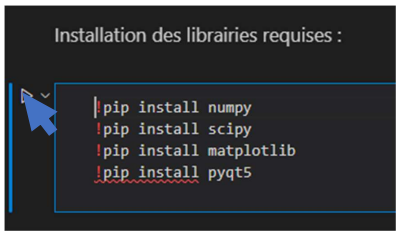
De l'application de simulation pour l'enseignement des télécommunications numériques

L'application permet de représenter l'influence de différents paramètres sur la transmission en bande de base d'une suite binaire pseudo aléatoire. Toutes les instructions nécessaires pour son bon fonctionnement sont présentées ci-après.

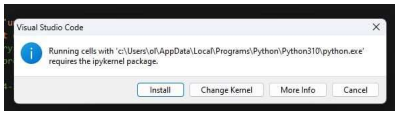
1. Instructions d'installations

1 – Ouvrez le fichier '**TeleSimU.ipynb**' avec un logiciel d'exécution comme Visual Studio Code ou Jupyter notebook d'Anaconda. A défaut, installez-le : https://code.visualstudio.com/Download ou https://www.anaconda.com/download

2 – Exécutez la première cellule, pour installer toutes les librairies nécessaires.

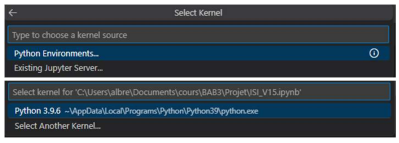


Si une fenêtre similaire apparaît, cliquez sur '**Install**'



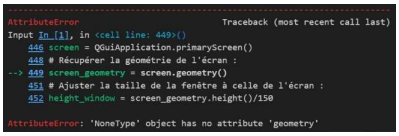
et l'extension Python Jupyter s'installera à Visual studio.

Lors du choix du noyau, choisissez '**Python Environments**' puis une version de Python supérieure à 3.9 .

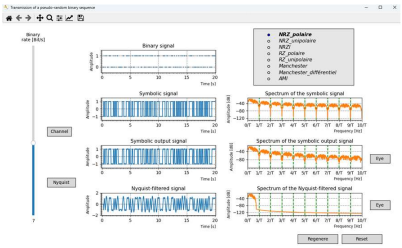


Si l'excécution n'aboutit à rien, installez ces librairies dans l'invité de commande.

3 – Faites de même pour la cellule de code suivante. Si vous obtenez cette erreur, réexécutez la cellule. La première exécution du script a permis d'obtenir les dimensions de votre écran.



4 – Si une fenêtre comme celle-ci s'ouvre, c'est que tout s'est bien déroulé. Sinon, voir Section 5.



2. Fonctionnalités générales

Cette fenêtre principale montre l'évolution du signal à 4 moments clés de sa transmission.

Le graphe en haut à gauche représente la suite de bits. Ensuite, par-dessous, se trouve le signal représentant l'attribution des symboles d'un code précis (ici : **NRZ_polaire**) pour chaque bit ainsi qu'à sa droite la transformée de Fourier de ce signal. Le troisième moment clé illustrant le signal en sortie du canal de transmission. Celui-ci est représenté juste en-dessous avec encore une fois une évolution temporelle à gauche et fréquentielle à droite. De même pour le dernier, à la sortie du filtre de Nyquist. Il y a également 6 boutons, une glissière, un cadre avec 8 sélections possibles ainsi qu'une ''toolbar''.

Projet de BAB3 – Innovations dans l'enseignement des télécommunications numériques

Tom Fontaine
Denis Albrecq
Mai 2024

Toolbar :

Si vous laissez votre souris immobile sur un icône, une explication de son utilité apparaîtra. Cette barre permet de modifier l'affichage des graphes mais en aucun cas les données. Elle est incluse lors de l'exécution de l'affichage d'un graphe sur une fenêtre extérieure à l'environnement du script.

Cadre de sélection :

Ce cadre représente tous les codes possibles pour affecter une forme de symbole au signal binaire. Pour changer de code, il suffit de cliquer sur le nom du code, il se mettra alors en gras pour indiquer qu'il a bien été sélectionné.

Glissière :

Elle permet de modifier la valeur du paramètre renseigner au-dessus (ou à gauche). En cliquant sur la glissière, cela modifie la valeur qui est alors affichée en dessous (ou à droite).

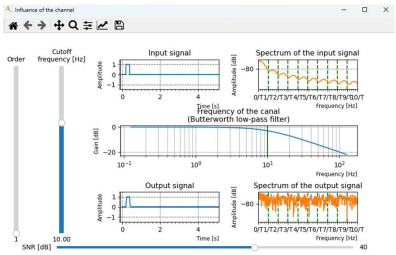
Bouton :

Lorsque l'on clique sur un bouton cela ouvre une seconde fenêtre avec d'autres graphes et d'autres éléments d'interactions. Les seules exceptions sont ; le bouton '**Regenere**' qui comme son nom l'indique regénère une autre suite pseudo aléatoire de bits et le bouton '**Reset**' qui permet de remettre toutes les glissières à leur valeur initiale.

3. Fonctionnalités avancées

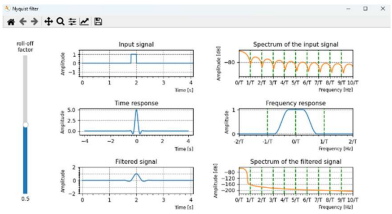
Il est possible d'ouvrir en tout 4 fenêtres supplémentaires interconnectées entre-elles. Lorsque vous changez le débit binaire ou bien le choix du code utilisé sur la fenêtre principale, la modification se répercute également sur toutes les autres fenêtres.

En cliquant sur '**Channel**', il est possible d'ouvrir une première fenêtre pour modifier les paramètres du canal de transmission. Tels que l'ordre, la fréquence de coupure et le rapport signal sur bruit.

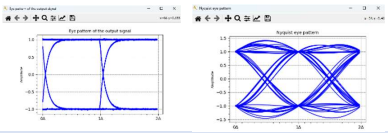


Les modifications engendrées seront automatiquement mises à jour pour toutes les fenêtres et cela vous permettra de voir l'impact de vos changements.

La seconde fenêtre est ouvrable par le bouton '**Nyquist**' et permet de modifier le facteur de roll-off du filtre de Nyquist.



Les deux derniers boutons '**Eye**' permettent de représenter les yeux des signaux à leur gauche.



Animation :

Il est possible de voir la construction du diagramme de l'œil pas à pas. Pour ce faire, rendez-vous à nouveau dans l'environnement du script et exécutez la dernière cellule de code.

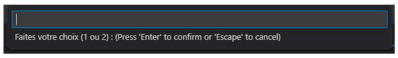
Animation de l'oeil de Nyquist :

```
from matplotlib import rc
%matplotlib qt

delta_test=1/symbolic_rate
delta=1/bit_rate_slider.val
ratio_bi_sy_rate = delta_test/delta
```


Tom Fontaine
Denis Albrecq
Mai 2024

Vous devriez obtenir un invité de saisie similaire :

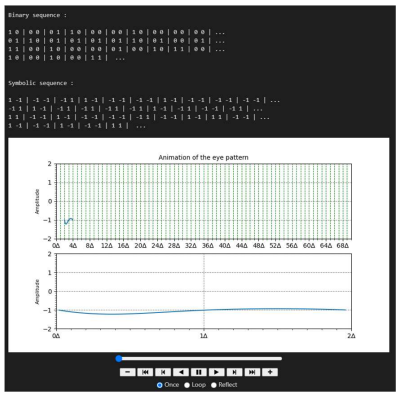



```
Signal à représenter :
1 : Symbolic output signal
2 : Nyquist-filtered signal
```

Il ne vous reste alors qu'à faire votre choix et l'animation se créera.

Le résultat s'affiche dans le script en-dessous de la cellule de code.

Un rappel des premiers bits est affiché ainsi que les symboles correspondants, ce qui vous permet de comprendre chaque courbe une à une.



Les boutons en dessous du graphe vous permettent de gérer l'animation.

- / + : Régler la vitesse,

|<< / >>| : Aller au tout début/fin,

|< / >| : Afficher la courbe précédente/suivante,

< / > : Démarrer l'animation à l'envers/à l'endroit,

|| : Faire pause.

La glissière permet de voyager à différents moments de l'animation. Et les sélecteurs permettent de choisir entre une animation unique, en boucle ou bien en réflexion.

4. Limitations

Chaque glissière a un minimum et un maximum, ce qui limite les observations possibles. Néanmoins, seules les glissières de débit binaire, de fréquence de coupure et du rapport signal sur bruit, ont un impact non-négligeable sur la qualité d'une transmission.

En effet, l'ordre du filtre est un paramètre plutôt fixe qui reste unitaire dans la majeure partie des cas. Quant au facteur de roll-off, il n'affecte que la réception du signal.

Le débit binaire doit être réglé pour assurer un certain nombre de transitions différentes de bits, il est donc conseillé de le régler entre 5 et 7 [bit/s] et entre 3 et 5 [bit/s] pour les 2 codes RZ et les 2 codes Manchester.

La fréquence de coupure du filtre passe-bas représentant le canal est dépendant de cette même valeur. Pour observer de l'interférence conséquente entre symboles, il faut placer la valeur en deçà de la moitié du débit binaire. Par ailleurs, pour que l'interférence disparaisse, il faut placer la valeur au-dessus de 2 fois la valeur du débit binaire (ou 3 fois pour les 4 codes particuliers mentionnés ci-avant).

Enfin, pour le rapport signal sur bruit, il faut le laisser dans la plage des 20 à 40 [dB], en-dessous le bruit sera fort impactant et au-dessus négligeable.

5. Support

Aide pour l'installation de Visual Studio :

https://code.visualstudio.com/docs#vscode

Aide pour l'installation de l'extension Jupyter :

https://code.visualstudio.com/docs/datascience/jupyter-notebooks#:~:text=You%20can%20create%20a%20Jupyter,picker%20in%20the%20top%20right.

6. Contact

Pour tout autre problème et/ou défaut veuillez nous contacter par mail :

Denis Albrecq : 221084@umons.ac.be

Tom Fontaine : 221537@umons.ac.be

Ou bien notre promotrice :

Véronique Moeyaert :

veronique.moeyaert@umons.ac.be

A.3.3 Commentaires sur le simulateur de transmissions en bande de base

Section 3

Vous trouverez dans cette partie un simulateur de transmissions en bande de base montrant les impacts des choix de paramètres que l'on peut faire : choix du code, densité spectrale de puissance, diagramme de l'oeil, impact de la bande passante du canal, interférences entre symboles, filtrage de Nyquist, ...

Il a été programmé en python par Denis Albrecq et Tom Fontaine dans le cadre d'un projet de 3ème bachelier ELEC.

Le programme est disponible ci-dessous, ainsi que le mode d'emploi pour l'installation et l'utilisation. Un fichier reprenant des ensembles de paramètres pour une bonne visualisation des notions est également disponible.

Votre feedback est important. N'hésitez pas à envoyer vos commentaires à veronique.moeyaert@umons.ac.be, tom.fontaine@student.umons.ac.be et denis.albrecq@student.umons.ac.be, ou de remplir le champ 'feedback' prévu à cet effet dans cette section du cours.

Code et mode d'emploi

Commentaires sur le simulateur de transmissions en bande de base

Figure A.22 – *Section 3 du cours de Communications Annalogiques et Numériques à l'intention des étudiants*

Mode: Le nom du participant sera enregistré et affiché avec ses réponses

1. Remarques sur l'installation : ouverture de fichier? installation des librairies? exécution du programme? ....

2. Remarques concernant l'interface : taille des fenêtres? Affichage des graphes? Utilisation intuitive des objects interactifs? .....

3. Remarques concernant le contenu de l'application par rapport au cours théorique?

4. Autres remarques

?

Remettre vos réponses Annuler

A.4 Script

A.4.1 Installation des librairies requises

```
!pip install numpy
!pip install scipy
!pip install matplotlib
!pip install pyqt5
```

A.4.2 Transmission d'un signal binaire pseudo aleatoire

```
# -*- coding: utf-8 -*-
#Project:
'''Developpement d'un outil pedagogique de simulation novateur
pour l'enseignement des systemes de telecommunications numeriques'''
#Created on February 2024
#@author: Denis Albrecq & Tom Fontaine
#Version: 1.0
#Last update: 30-04-24



# LIBRAIRIES REQUISES :

import numpy as np
{'''# Importe le module NumPy, utilise pour les operations mathematiques
    '''}

from scipy.signal import butter, lfilter, freqs, lfilter_zi
{'''Importe des fonctions specifiques du module SciPy pour le traitement
     des signaux :
- butter : Conception de filtres Butterworth
- lfilter : Filtrage d'un signal avec un filtre lineaire
- freqs : Calcule la reponse en frequence d'un filtre
- lfilter_zi : Obtient les conditions initiales pour un filtre
- convolve : Effectue une convolution entre deux signaux'''}

import random
{'''Importe le module random pour generer des nombres aleatoires'''}

import matplotlib.pyplot as plt
{'''Importe matplotlib.pyplot pour tracer des graphiques'''}

from matplotlib.widgets import Button, Slider, RadioButtons
{'''Importe des widgets interactifs pour matplotlib :
- Button : Bouton cliquable
- Slider : Curseur interactif
- RadioButtons : Boutons radio pour selectionner des options'''}

from matplotlib.ticker import MultipleLocator, FuncFormatter
{'''Importe des outils pour formater les axes dans les graphiques
    matplotlib :
- MultipleLocator : Definit des intervalles reguliers pour les
    graduations
- FuncFormatter : Permet des formats personnalises pour les etiquettes
    des axes'''}
```

```python

import matplotlib.animation as animation
{'''Importe le module d'animation de matplotlib pour creer des animations'''}

%matplotlib qt
{'''Configurations speciales pour matplotlib :
Indique e Jupyter d'utiliser l'interface graphique Qt pour afficher les graphiques'''}

from PyQt5.QtGui import QGuiApplication
{'''Importe QGuiApplication, qui fournit les fonctions necessaires pour une application graphique
basee sur Qt5, principalement pour la gestion des evenements de l'interface utilisateur,
des fen tres, et des aspects graphiques generaux'''}



# LES FONCTIONS :

# Fonction pour generer un signal binaire :
def generate_binary_signal(duration, sampling_rate, bit_rate):
    t = np.arange(0, duration, 1/sampling_rate)
    signal = np.zeros(len(t))
    # e intervalles definis par le ratio sampling_rate/bit_rate, genere un bit aleatoire (0 ou 1)
    # et maintient cette valeur pendant toute la duree du bit
    for i in range(len(t)):
        if i%(int(sampling_rate/bit_rate))==0:
            signal[i]=random.getrandbits(1)
        else :
            signal[i]=signal[i-1]
    return t, signal

# Fonction pour generer un signal binaire e 1! symbole :
def generate_rect_signal(duration, sampling_rate, delta, emplacement) :
    t = np.arange(0, duration, 1/sampling_rate)
    signal = np.zeros(len(t))
    # Si le temps actuel est dans la plage de l'impulsion, definissez le signal e 1
    for i in range(len(t)):
        if t[i]>emplacement*delta and t[i]<(emplacement+1)*delta :
            signal[i]=1
        else :
            signal[i]=0
    return t, signal

# Differents types de code :
def get_code(code_name):
    if code_name == 'NRZ_polaire' :
        return NRZ_polaire
    if code_name == 'NRZ_unipolaire' :
        return NRZ_unipolaire
    if code_name == 'NRZI' :
        return NRZI
    if code_name == 'RZ_polaire' :
        return RZ_polaire
```

```
    if code_name == 'RZ_unipolaire' :
        return RZ_unipolaire
    if code_name == 'Manchester' :
        return Manchester
    if code_name == 'Manchester_differentiel' :
        return Manchester_differentiel
    if code_name == 'AMI' :
        return AMI

# Les bits '1' sont mappes e +1 et les bits '0' e -1
def NRZ_polaire(binary_signal, bit_rate, sampling_rate):
    symbolic_rate = bit_rate
    P_signal = 1
    Covariance = 1
    symbolic_signal=np.zeros(len(binary_signal))
    for i in range(len(binary_signal)):
        symbolic_signal[i]=1 if (binary_signal[i]==1) else -1
    return symbolic_signal, symbolic_rate, P_signal, Covariance

# Les bits '1' sont mappes e +1 et les bits '0' restent e 0
def NRZ_unipolaire(binary_signal, bit_rate, sampling_rate):
    symbolic_rate = bit_rate
    P_signal = 0.5
    Covariance = 0.25
    symbolic_signal=binary_signal
    return symbolic_signal, symbolic_rate, P_signal, Covariance

# Un '0' indique un changement de niveau, tandis qu'un '1' maintient le
    niveau
def NRZI(binary_signal, bit_rate, sampling_rate):
    symbolic_rate = bit_rate
    P_signal = 1
    Covariance = 1
    symbolic_signal=np.zeros(len(binary_signal))
    symbolic_signal[0]=1 # Commence avec +1 (choix par defaut)
    for i in range(1, len(binary_signal)):
        if i%(int(sampling_rate/bit_rate))==0:
            if (binary_signal[i]==0):
                symbolic_signal[i]=-1*symbolic_signal[i-1]
            else :
                symbolic_signal[i]=symbolic_signal[i-1]
        else :
            symbolic_signal[i]=symbolic_signal[i-1]

    return symbolic_signal, symbolic_rate, P_signal, Covariance

# Les bits '1' mappes e +1, bits '0' mappes e -1 et retour e zero au
    milieu de chaque bit
def RZ_polaire(binary_signal, bit_rate, sampling_rate):
    symbolic_rate = bit_rate*2
    P_signal = 0.5
    Covariance = 1
    symbolic_signal=np.zeros(len(binary_signal))
    k=0
    for i in range(len(binary_signal)):
        if i%(int(sampling_rate/bit_rate))==0:
            symbolic_signal[i]=1 if (binary_signal[i]==1) else -1
            k+=1
```

```
        elif i%(int(sampling_rate/(2*bit_rate)))==0 :
            symbolic_signal[i]=0
        else :
            symbolic_signal[i]=symbolic_signal[i-1]
    return symbolic_signal, symbolic_rate, P_signal, Covariance

# Les bits '1' mappes e +1 et retour e zero au milieu
def RZ_unipolaire(binary_signal, bit_rate, sampling_rate):
    symbolic_rate = bit_rate*2
    P_signal = 0.25
    Covariance = 0.25
    symbolic_signal=np.zeros(len(binary_signal))
    k=0
    for i in range(len(binary_signal)):
        if i%(int(sampling_rate/bit_rate))==0:
            symbolic_signal[i]=binary_signal[i]
            k+=1
        elif i%(int(sampling_rate/(2*bit_rate)))==0 :
            symbolic_signal[i]=0
        else :
            symbolic_signal[i]=symbolic_signal[i-1]
    return symbolic_signal, symbolic_rate, P_signal, Covariance

# Chaque bit '1' commence par +1 et retourne e -1, et chaque '0' commence par -1 et retourne e +1
def Manchester(binary_signal, bit_rate, sampling_rate):
    symbolic_rate = bit_rate*2
    P_signal = 0.5
    Covariance = 1
    symbolic_signal=np.zeros(len(binary_signal))
    k=0
    for i in range(len(binary_signal)):
        if i%(int(sampling_rate/bit_rate))==0:
            if (binary_signal[i]==0) :
                symbolic_signal[i]=-1
            else :
                symbolic_signal[i]=1
            k+=1
        elif i%(int(sampling_rate/(2*bit_rate)))==0 :
            symbolic_signal[i]=-1*symbolic_signal[i-1]
        else :
            symbolic_signal[i]=symbolic_signal[i-1]
    return symbolic_signal, symbolic_rate, P_signal, Covariance

# Un '1' signifie maintien du niveau, un '0' signifie inversion du niveau
def Manchester_differentiel(binary_signal, bit_rate, sampling_rate):
    symbolic_rate = bit_rate*2
    P_signal = 0.5
    Covariance = 1
    symbolic_signal=np.zeros(len(binary_signal))
    k=0
    symbolic_signal[0]=1 # Commence avec +1 (choix par defaut)
    for i in range(1, len(binary_signal)):
        if i%(int(sampling_rate/bit_rate))==0:
            if (binary_signal[i]==0) :
                symbolic_signal[i]=-1*symbolic_signal[i-1]
            else :
```

```
                symbolic_signal[i]=symbolic_signal[i-1]
            k+=1
        elif i%(int(sampling_rate/(2*bit_rate)))==0 :
            symbolic_signal[i]=-1*symbolic_signal[i-1]
        else :
            symbolic_signal[i]=symbolic_signal[i-1]
    return symbolic_signal, symbolic_rate, P_signal, Covariance

# Un '1' change d'un niveau positif e un niveau negatif de fa on alternee, et un '0' reste e zero
def AMI(binary_signal, bit_rate, sampling_rate):
    symbolic_rate = bit_rate
    P_signal = 1
    Covariance = 0.5
    symbolic_signal=np.zeros(len(binary_signal))
    k=1 # Niveau initial +1 (choix par defaut)
    for i in range(len(binary_signal)):
        if i%(int(sampling_rate/bit_rate))==0:
            if (binary_signal[i]==1):
                symbolic_signal[i]=k
                if k==1:
                    k=-1
                else :
                    k=1
            else :
                symbolic_signal[i]=0
        else :
            symbolic_signal[i]=symbolic_signal[i-1]
    return symbolic_signal, symbolic_rate, P_signal, Covariance

# Fonction pour appliquer un filtre passe-bas :
def apply_lowpass_filter(signal, normal_cutoff, order):
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    zi = lfilter_zi(b, a)
    filtered_signal, _ = lfilter(b, a, signal, zi=zi*signal[0])
    #Pour la reponse indicielle du filtre
    b, a = butter(order, normal_cutoff, btype='low', analog=True)
    w, h = freqs(b, a)
    return filtered_signal, w, h

# Fonction pour generer le bruit AWGN (bruit additif blanc gaussien) pour chaque echantillon :
def get_random_value(sigma):
    mu=0 #car centre sur la valeur de l'amplitude du symbole
    value=random.gauss(mu, sigma)
    (mu, sigma)
    return value

# Fonction pour appliquer du bruit :
def apply_noise(signal, snr, P_signal, Covariance):
    signal_noise=np.zeros(len(signal))
    Resistance = 1 #[Ohm]
    P_noise=P_signal*(10**(-(snr+10*np.log10(Resistance))/20)) # conversion de puissance
    for i in range(len(signal)):
        signal_noise[i]=signal[i]+get_random_value(np.sqrt(Covariance))*P_noise
    return signal_noise
```

```python

# Fonction pour appliquer le filtre de nyquist :
def apply_nyquist_filter(signal, duration, sampling_rate, symbolic_rate, delta, rolloff_factor) :
    a = np.zeros(int(duration*symbolic_rate))  # a = suite de symboles
    for i in range(len(a)):
        a[i] = signal[int(((sampling_rate*delta)/2)+(sampling_rate*delta)*i)]

    # Fonction en cosinus sureleve
    def get_rc(T, rolloff_factor):
        def rc(t, beta):
            return np.sinc(t)*np.cos(np.pi*beta*t)/(1-(2*beta*t)**2)
        return lambda t: rc(t/T, rolloff_factor)

    # Fonction pour generer la transmission du signal comme suit : sum(a[k]*g(t-k*delta) )
    def get_signal(g, a):
        t_ = np.arange(-duration, duration, 1/sampling_rate)
        xt = sum(a[k]*g(t_-(k+1)*delta) for k in range(len(a)))
        signal = xt[int((duration + 2*delta)*sampling_rate):int(2*duration*sampling_rate)]
        begin = np.zeros(int(2*delta*sampling_rate))
        return np.concatenate((begin, signal))

    g = get_rc(delta, rolloff_factor)
    return  get_signal(g, a)
'''Remarque :
ici le code vient chercher une valeur au milieu de chaque symbole,
la multiplie avec g(=filtre rc) et somme le tout.

Ceci a ete fait pour simplifier la charge de calcul, il aurait fallu le faire pour chq point.
Ainsi on aurait pris en compte de maniere plus significative l'impact du bruit.

(Il est d'ailleurs tout e fait possible de le faire avec un np.convolve()
et ce qui se fait aussi rapidement en terme de charge de calcul.
Neanmoins, il faut arrive e decaler g pour que les symboles soient maximises au moment d'echantillonage.
Ced e la fin du symbole.)

En outre, nous avons remarque que le changement n'est pas fort significative,
l'approximation faite est donc tout e fait acceptable.
'''

# Fonction pour formatter les graduations des axes :
def formatter(symbolic_rate, char) :
    return FuncFormatter(lambda x, _: f'{int(x/symbolic_rate)}' + char)

# Fonction pour tracer le signal binaire :
def draw_binary_signal(t, signal, ax, title, bit_rate, duration, val):
    a = np.zeros(int(duration*bit_rate))  # a = suite de symboles
    t_ = np.zeros(len(a))
    for i in range(len(a)):
```

```
        a[i] = signal[int(((sampling_rate/bit_rate)/2)+(sampling_rate/bit_rate)*i)]
        t_[i] = t[int(((sampling_rate/bit_rate)/2)+(sampling_rate/bit_rate)*i)]
    line = ax.scatter(t_, a, s=20*dsize/bit_rate)
    ax.set_title(title, fontsize=20*dsize)
    ax.set_xlabel('Time [s]', fontsize=15*dsize, loc='right')
    ax.set_xlim(-0.1,duration+0.1)
    ax.set_ylabel('Amplitude', fontsize=15*dsize)
    ax.set_ylim(-val+1,val)
    ax.xaxis.set_minor_locator(MultipleLocator(1/bit_rate))
    ax.yaxis.set_minor_locator(MultipleLocator(0.1))
    ax.axhline(1, color='black', linestyle='--', alpha=0.3, linewidth=0.75)
    ax.axhline(0, color='black', linestyle='--', alpha=0.3, linewidth=0.75)
    ax.grid(axis = 'both', linestyle='--', color='grey', linewidth=0.75)
    return line

# Fonction pour tracer un signal temporel :
def draw_signal(t, signal, ax, title, symbolic_rate, duration, val):
    line, = ax.plot(t, signal)
    ax.set_title(title, fontsize=20*dsize)
    ax.set_xlabel('Time [s]', fontsize=15*dsize, loc='right')
    ax.set_xlim(-0.1,duration+0.1)
    ax.set_ylabel('Amplitude', fontsize=15*dsize)
    ax.set_ylim(-val,val)
    ax.xaxis.set_minor_locator(MultipleLocator(1/symbolic_rate))
    ax.yaxis.set_minor_locator(MultipleLocator(0.1))
    ax.axhline(1, color='black', linestyle='--', alpha=0.3, linewidth=0.75)
    ax.axhline(0, color='black', linestyle='--', alpha=0.3, linewidth=0.75)
    ax.axhline(-1, color='black', linestyle='--', alpha=0.3, linewidth=0.75)
    ax.grid(axis = 'both', linestyle='--', color='grey', linewidth=0.75)
    return line

# Fonction pour tracer le spectre d'un signal :
def draw_spectre(signal, ax, title, symbolic_rate, sampling_rate, bit_rate):
    ax.xaxis.set_major_formatter(formatter(bit_rate, '/T'))
    ax.xaxis.set_major_locator(MultipleLocator(symbolic_rate))
    ax.xaxis.set_minor_locator(MultipleLocator(1))
    ax.yaxis.set_major_locator(MultipleLocator(40))
    ax.set_xlim(0,10*symbolic_rate)
    for i in range(10):
        ax.axvline(i*symbolic_rate, color='g', linestyle = '--')
    line = ax.magnitude_spectrum(signal, Fs=sampling_rate, scale='dB', color='C1')[2]
    ax.grid(axis ='y', linestyle='--', color='grey', linewidth=0.75)
    ax.set_title(title, fontsize=20*dsize)
    ax.set_xlabel('Frequency [Hz]', fontsize=15*dsize, loc='right')
    ax.set_ylabel('Amplitude [dB]', fontsize=15*dsize)
    return line

# Fonction pour tracer l'oeil de nyquist :
def drawFullEyeDiagram(signal, ax, delta, sampling_rate, title):
```

```
    t_eye = np.arange(0,2*delta,1/sampling_rate)
    for i in range(int(duration/(2*delta))-2):
        x_eye = signal[(int(2*sampling_rate*delta*(i+2))):(int(2*sampling_rate*delta*(i+3)))]
        ax.plot(t_eye,x_eye, 'b-')
    ax.set_title(title, fontsize=10)
    ax.set_ylabel('Amplitude', fontsize=8)
    ax.xaxis.set_major_formatter(formatter(delta, chr(8710)))
    ax.xaxis.set_major_locator(MultipleLocator(delta))
    ax.xaxis.set_minor_locator(MultipleLocator(0.1*delta))
    ax.yaxis.set_minor_locator(MultipleLocator(0.1))
    ax.grid(axis = 'both', linestyle='--', color='grey', linewidth=0.75)

# Fonction pour afficher les graphes du filtre de nyquist :
def plot_nyquist(duration, symbolic_rate, sampling_rate, delta, rolloff_factor, ax2_5, ax2_6) :
    t_g = np.arange(-duration/symbolic_rate, duration/symbolic_rate, 1/sampling_rate)
    g=np.zeros(len(t_g))
    for i in range(len(t_g)) :
        g[i] = (((np.sinc(t_g[i]/delta)) * np.cos((np.pi*rolloff_factor*t_g[i])/delta)) / ( 1 - ((2*rolloff_factor*t_g[i])/delta)**2))/delta
    line2_5, = ax2_5.plot(t_g, g)
    ax2_5.set_title('Time response', fontsize=20*dsize)
    ax2_5.set_xlabel('Time [s]', fontsize=15*dsize, loc='right')
    ax2_5.set_ylabel('Amplitude', fontsize=15*dsize)
    ax2_5.xaxis.set_minor_locator(MultipleLocator(delta))
    ax2_5.yaxis.set_major_locator(MultipleLocator(0.5/delta))
    ax2_5.grid(axis = 'y', linestyle='--', color='grey', linewidth=0.75)

    delta_f=(rolloff_factor*symbolic_rate)/2
    f=np.arange(-2*symbolic_rate, 2*symbolic_rate, 1/sampling_rate)
    g_f=np.zeros(len(f))
    for i in range (int(sampling_rate*symbolic_rate*4)):
        if abs(f[i])<=(symbolic_rate/2)-delta_f :
            g_f[i]=1
        elif abs(f[i])>=(symbolic_rate/2)+delta_f :
            g_f[i]=0
        elif rolloff_factor!=0 :
            g_f[i]=0.5*(1-np.sin(((np.pi*abs(f[i]))/(rolloff_factor*symbolic_rate))-(np.pi/(2*rolloff_factor))))
    ax2_6.xaxis.set_major_formatter(formatter(symbolic_rate, '/T'))
    ax2_6.xaxis.set_major_locator(MultipleLocator(symbolic_rate))
    ax2_6.xaxis.set_minor_locator(MultipleLocator(1))
    ax2_6.yaxis.set_minor_locator(MultipleLocator(50))
    ax2_6.set_xlim(-2*symbolic_rate,2*symbolic_rate)
    for i in range(2):
        ax2_6.axvline(i*symbolic_rate, color='g', linestyle = '--')
        ax2_6.axvline(-i*symbolic_rate, color='g', linestyle = '--')
    ax2_6.grid(axis ='y', linestyle='--', color='grey', linewidth=0.75)
    ax2_6.set_title('Frequency response', fontsize=20*dsize)
    ax2_6.set_xlabel('Frequency [Hz]', fontsize=15*dsize, loc='right')
    ax2_6.set_ylabel('Amplitude', fontsize=15*dsize)
    ax2_6.plot(f, g_f)



# INITIALISATION :
```

```python

# Parametres du signal :
duration = 20.0  # duree en secondes
bit_rate = 5 # nombre de bits par seconde
sampling_rate = int(bit_rate*50)  # frequence d'echantillonage en [Hz]
symbolic_rate = bit_rate # nombre de symboles par seconde

# Creation du signal binaire :
t, binary_signal = generate_binary_signal(duration, sampling_rate, bit_rate)

# Codage du signal :
list_code = ('NRZ_polaire','NRZ_unipolaire','NRZI','RZ_polaire','RZ_unipolaire','Manchester','Manchester_differentiel','AMI')
code_name = list_code[0] # Choix par defaut du type de code
code = get_code(code_name)
symbolic_signal, symbolic_rate, P_signal, Covariance = code(binary_signal, bit_rate, sampling_rate)

# Parametres du canal :
snr = 40 # rapport signal sur bruit en [dB]
# Parametres filtre passe-bas representant le canal :
cutoff_frequency = 10 # frequence de coupure en [Hz] du filtre
normal_cutoff= (2*cutoff_frequency) / sampling_rate
order = 1 # ordre du filtre

# Faire passer le signal dans le canal :
symbolic_signal_filtered, w, h = apply_lowpass_filter(symbolic_signal, normal_cutoff, order)
symbolic_signal_out = apply_noise(symbolic_signal_filtered, snr, P_signal, Covariance)

# Appliquer le filtre de nyquist :
rolloff_factor = 0.5 # facteur de roll-off
delta = 1/symbolic_rate # duree en secondes d'un symbole
filtered_signal = apply_nyquist_filter(symbolic_signal_out, duration, sampling_rate, symbolic_rate, delta, rolloff_factor)



# FENETRE PRINCIPALE :

# Recuperer l'ecran principal :
screen = QGuiApplication.primaryScreen()

# Recuperer la geometrie de l'ecran :
screen_geometry = screen.geometry()

# Ajuster la taille de la fen tre e celle de l'ecran :
height_window = screen_geometry.height()/150
width_window = screen_geometry.width()/150
dsize=height_window/width_window

# Creer la fen tre :
homepage=plt.figure("Transmission of a pseudo-random binary sequence", figsize=(width_window,height_window))
axs_homepage = homepage.subplots(4,2)  # 4 lignes et 2 colonnes

```

```python
# Creer les graphes :

# Signal binaire
line0_1 = draw_binary_signal(t, binary_signal, axs_homepage[0,0], "Binary signal", bit_rate, duration, val=1.5)

plt.delaxes(axs_homepage[0,1])

# Signal symbolique
line0_3 = draw_signal(t, symbolic_signal, axs_homepage[1,0], "Symbolic signal", symbolic_rate, duration, val=1.5)

# Spectre du signal symbolique
line0_4 = draw_spectre(symbolic_signal, axs_homepage[1,1], "Spectrum of the symbolic signal", symbolic_rate, sampling_rate, bit_rate)

# Signal symbolique de sortie
line0_5 = draw_signal(t, symbolic_signal_out, axs_homepage[2,0], "Symbolic output signal", symbolic_rate, duration, val=1.5)

# Spectre du signal symbolique de sortie
line0_6 = draw_spectre(symbolic_signal_out, axs_homepage[2,1], "Spectrum of the symbolic output signal", symbolic_rate, sampling_rate, bit_rate)

# Signal filtre par nyquist
line0_7 = draw_signal(t, filtered_signal, axs_homepage[3,0], "Nyquist-filtered signal", symbolic_rate, duration, val=2)

# Spectre du signal filtre par nyquist
line0_8 = draw_spectre(filtered_signal, axs_homepage[3,1], "Spectrum of the Nyquist-filtered signal", symbolic_rate, sampling_rate, bit_rate)

# Ajuster les graphes dans la fen tre :
homepage.subplots_adjust(left=0.25, bottom=0.15,hspace=2*dsize, wspace=dsize/2)



# FENETRE DE L'OEUIL DU SIGNAL EN SORTIE DE CANAL :
oeil_outcanal_window = plt.figure("Eye pattern of the output signal", figsize=(width_window/2, height_window/2))
plt.close()
ax_oeil_outcanal_window = oeil_outcanal_window.add_subplot(111)
oeil_outcanal_window.subplots_adjust(left=0.2, bottom=0.1,hspace=1.5, wspace=0.5)



# FENETRE DE L'OEUIL DE NYQUIST :
oeil_window = plt.figure("Nyquist eye pattern", figsize=(width_window/2, height_window/2))
plt.close()
ax_oeil_window = oeil_window.add_subplot(111)
oeil_window.subplots_adjust(left=0.2, bottom=0.1,hspace=1.5, wspace=0.5)
```

```
# FENETRE DE L'INFLUENCE DU CANAL :
canal_window = plt.figure("Influence of the channel", figsize=(2*width_window/3, 2*height_window/3))
plt.close()

# Creer un signal avec 1! symbole :
rect_t, rect_signal=generate_rect_signal(duration/2, sampling_rate, 1/bit_rate, emplacement=1)

# Faire passer le signal dans le canal :
rect_signal_filtered, w, h = apply_lowpass_filter(rect_signal, normal_cutoff, order)
rect_signal_out = apply_noise(rect_signal_filtered, snr, P_signal=1, Covariance=1)

# Creer les graphes :

# Signal d'entree
ax1_1=canal_window.add_subplot(321)
line1_1 = draw_signal(rect_t, rect_signal, ax1_1, "Input signal", bit_rate, duration/(2+int(bit_rate/2)), val=1.5)

# Spectre du signal d'entree
ax1_2=canal_window.add_subplot(322)
line1_2 = draw_spectre(rect_signal, ax1_2, "Spectrum of the input signal", bit_rate, sampling_rate, bit_rate)

# Signal de sortie
ax1_3=canal_window.add_subplot(325)
line1_3 = draw_signal(rect_t, rect_signal_out, ax1_3, "Output signal", bit_rate, duration/(2+int(bit_rate/2)), val=1.5)

# Spectre du signal de sortie
ax1_4=canal_window.add_subplot(326)
line1_4 = draw_spectre(rect_signal_out, ax1_4, "Spectrum of the output signal", bit_rate, sampling_rate, bit_rate)

# Reponse en frequence du canal
ax1_5=canal_window.add_subplot(312)
line1_5 = ax1_5.semilogx((w*sampling_rate)/2, 20 * np.log10(abs(h)))
ax1_5.set_title('Frequency of the canal\n(Butterworth low-pass filter)', fontsize=20*dsize)
ax1_5.set_xlabel('Frequency [Hz]', fontsize=15*dsize, loc='right')
ax1_5.set_ylabel('Gain [dB]', fontsize=15*dsize)
ax1_5.grid(which='both', axis='both')
ax1_5.axvline(cutoff_frequency, color='green')

# Ajuster le graphe dans la fen tre :
canal_window.subplots_adjust(left=0.3, bottom=0.15,hspace=1.2, wspace=0.4)



# FENETRE DU FILTRE DE NYQUIST :
nyquist_window = plt.figure("Nyquist filter", figsize=(4*width_window/5, 2*height_window/3))
plt.close()

```

```
# Creer un signal avec 1! symbole :
rect_t_ny, rect_signal_ny=generate_rect_signal(duration, sampling_rate, 1/symbolic_rate, emplacement=int(symbolic_rate*duration/10)-1)
# Appliquer le filtre de nyquist
rect_signal_ny_filtered = apply_nyquist_filter(rect_signal_ny, duration, sampling_rate, symbolic_rate, delta, rolloff_factor)

# Creer les graphes :

# Signal d'entree
ax2_1=nyquist_window.add_subplot(321)
line2_1 = draw_signal(rect_t_ny, rect_signal_ny, ax2_1, "Input signal", bit_rate, duration/5, val=1.5)

# Spectre du signal d'entree
ax2_2=nyquist_window.add_subplot(322)
line2_2 = draw_spectre(rect_signal_ny, ax2_2, "Spectrum of the input signal", symbolic_rate, sampling_rate, bit_rate)

# Signal filtre
ax2_3=nyquist_window.add_subplot(325)
line2_3 = draw_signal(rect_t_ny, rect_signal_ny_filtered, ax2_3, "Filtered signal", bit_rate, duration/5, val=2)

# Spectre du signal filtre
ax2_4=nyquist_window.add_subplot(326)
line2_4 = draw_spectre(rect_signal_ny_filtered, ax2_4, "Spectrum of the filtered signal", symbolic_rate, sampling_rate, bit_rate)

# Reponse temporelle et frequentielle du filtre de nyquist
ax2_5=nyquist_window.add_subplot(323)
ax2_6=nyquist_window.add_subplot(324)
plot_nyquist(duration, bit_rate, sampling_rate, delta, rolloff_factor, ax2_5, ax2_6)

# Ajuster le graphe dans la fen tre :
nyquist_window.subplots_adjust(left=0.2, bottom=0.1,hspace=1.2, wspace=0.4)



# BOUTON DE RELOAD :

def regenere(event):
    global t, binary_signal, filtered_signal, rect_t, rect_signal, normal_cutoff, delta, sampling_rate, symbolic_signal, symbolic_signal_out, symbolic_rate
    sampling_rate = int(bit_rate_slider.val*50)
    normal_cutoff = (2*cutoff_frequency_slider.val) / sampling_rate

    # Creation d'une autre suite binaire :
    t, binary_signal = generate_binary_signal(duration, sampling_rate, bit_rate_slider.val)
    # Codage du signal :
    code = get_code(code_name)
    symbolic_signal, symbolic_rate, P_signal, Covariance = code(binary_signal, bit_rate_slider.val, sampling_rate)
    delta = 1/symbolic_rate
```

```
    # Faire passer le signal dans le canal :
    symbolic_signal_filtered, w, h = apply_lowpass_filter(symbolic_signal, normal_cutoff, order_slider.val)
    symbolic_signal_out = apply_noise(symbolic_signal_filtered, snr_slider.val, P_signal, Covariance)
    # Appliquer le filtre de nyquist :
    filtered_signal = apply_nyquist_filter(symbolic_signal_out, duration, sampling_rate, symbolic_rate, delta, rolloff_factor_slider.val)

    # Retracer les graphes :
    axs_homepage[0,0].cla()
    line0_1 = draw_binary_signal(t, binary_signal, axs_homepage[0,0], "Binary signal", bit_rate_slider.val, duration, val=1.5)
    line0_3.set_data(t, symbolic_signal)
    axs_homepage[1,1].cla()
    line0_4 = draw_spectre(symbolic_signal, axs_homepage[1,1], "Spectrum of the symbolic signal", symbolic_rate, sampling_rate, bit_rate_slider.val)
    line0_5.set_data(t, symbolic_signal_out)
    axs_homepage[2,1].cla()
    line0_6 = draw_spectre(symbolic_signal_out, axs_homepage[2,1], "Spectrum of the symbolic output signal", symbolic_rate, sampling_rate, bit_rate_slider.val)
    line0_7.set_data(t, filtered_signal)
    axs_homepage[3,1].cla()
    line0_8 = draw_spectre(filtered_signal, axs_homepage[3,1], "Spectrum of the Nyquist-filtered signal", symbolic_rate, sampling_rate, bit_rate_slider.val)

    # Mettre e jour la fen tre principale :
    homepage.canvas.draw_idle()

    # Mettre e jour la fen tre du signal en sortie de canal :
    ax_oeil_outcanal_window.cla()
    drawFullEyeDiagram(symbolic_signal_out, ax_oeil_outcanal_window, 1/bit_rate_slider.val, sampling_rate, "Eye pattern of the output signal")
    oeil_outcanal_window.canvas.draw_idle()

    # Mettre e jour la fen tre de l'oeil de nyquist :
    ax_oeil_window.cla()
    drawFullEyeDiagram(filtered_signal, ax_oeil_window, 1/bit_rate_slider.val, sampling_rate, "Nyquist eye pattern")
    oeil_window.canvas.draw_idle()

    # Mettre e jour le signal d'un seul symbole :
    rect_t, rect_signal = generate_rect_signal(duration/2, sampling_rate, delta, emplacement=1)
    rect_signal_filtered, w, h = apply_lowpass_filter(rect_signal, normal_cutoff, order_slider.val)
    rect_signal_out = apply_noise(rect_signal_filtered, snr_slider.val, P_signal=1, Covariance=1)

    # Retracer les graphes :
    ax1_1.cla()
    line1_1 = draw_signal(rect_t, rect_signal, ax1_1, "Input signal", bit_rate_slider.val, duration/(2+int(bit_rate_slider.val/2)), val=1.5)
```

```
    ax1_2.cla()
    line1_2 = draw_spectre(rect_signal, ax1_2, "Spectrum of the input signal", bit_rate_slider.val, sampling_rate, bit_rate_slider.val)
    ax1_3.cla()
    line1_3 = draw_signal(rect_t, rect_signal_out, ax1_3, "Output signal", bit_rate_slider.val, duration/(2+int(bit_rate_slider.val/2)), val=1.5)
    ax1_4.cla()
    line1_4 = draw_spectre(rect_signal_out, ax1_4, "Spectrum of the output signal", bit_rate_slider.val, sampling_rate, bit_rate_slider.val)
    ax1_5.cla()
    line1_5 = ax1_5.semilogx(w*sampling_rate/2, 20 * np.log10(abs(h)))
    ax1_5.set_title('Frequency response of the canal\n(Butterworth low-pass filter)', fontsize=20*dsize)
    ax1_5.set_xlabel('Frequency [rad/s]', fontsize=15*dsize, loc='right')
    ax1_5.set_ylabel('Gain [dB]', fontsize=15*dsize)
    ax1_5.grid(which='both', axis='both')
    ax1_5.axvline(cutoff_frequency_slider.val, color='green')

    # Mettre e jour la fen tre de l'influence du canal :
    canal_window.canvas.draw_idle()

    # Mettre e jour le signal d'un seul symbole pour le filtre de nyquist :
    rect_t_ny, rect_signal_ny = generate_rect_signal(duration, sampling_rate, 1/symbolic_rate, emplacement=int(symbolic_rate*duration/10)-1)
    rect_signal_ny_filtered = apply_nyquist_filter(rect_signal_ny, duration, sampling_rate, symbolic_rate, delta, rolloff_factor_slider.val)

    # Retracer les graphes :
    ax2_1.cla()
    line2_1 = draw_signal(rect_t_ny, rect_signal_ny, ax2_1, "Input signal", bit_rate_slider.val, duration/5, val=1.5)
    ax2_2.cla()
    line2_2 = draw_spectre(rect_signal_ny, ax2_2, "Spectrum of the intput signal", symbolic_rate, sampling_rate, bit_rate_slider.val)
    ax2_3.cla()
    line2_3 = draw_signal(rect_t_ny, rect_signal_ny_filtered, ax2_3, "Filtered signal", bit_rate_slider.val, duration/5, val=1.5)
    ax2_4.cla()
    line2_4 = draw_spectre(rect_signal_ny_filtered, ax2_4, "Spectrum of the filtered signal", symbolic_rate, sampling_rate, bit_rate_slider.val)
    ax2_5.cla()
    ax2_6.cla()
    plot_nyquist(duration, bit_rate_slider.val, sampling_rate, delta, rolloff_factor_slider.val, ax2_5, ax2_6)

    # Mettre e jour la fen tre du filtre de nyquist :
    nyquist_window.canvas.draw_idle()

# [Gauche, Bas, Largeur, Hauteur]
#donne en pourcentage par rapport aux dimension de la fen tre
reload_ax= homepage.add_axes([0.74, 0.025, 0.1, 0.04])
```

```
button_reload = Button(reload_ax, 'Regenere', hovercolor='0.975')
button_reload.on_clicked(regenere)



# BOUTON DE RESTE GLOBAL :

def reset_all(event):
    bit_rate_slider.reset()
    order_slider.reset()
    cutoff_frequency_slider.reset()
    snr_slider.reset()
    rolloff_factor_slider.reset()
reset_all_ax = homepage.add_axes([0.85,0.025,0.1,0.04])
button_reset_all = Button(reset_all_ax, 'Reset', hovercolor='0.975')
button_reset_all.on_clicked(reset_all)



# BOUTON DE MISE A JOUR :

def update(val):
    global filtered_signal, normal_cutoff, delta, sampling_rate, symbolic_signal_out, symbolic_rate, symbolic_signal
    sampling_rate = int(bit_rate_slider.val*50)
    normal_cutoff = (2*cutoff_frequency_slider.val) / sampling_rate

    # Codage du signal :
    code = get_code(code_name)
    symbolic_signal, symbolic_rate, P_signal, Covariance = code(binary_signal, bit_rate_slider.val, sampling_rate)
    delta = 1/symbolic_rate
    # Faire passer le signal dans le canal :
    symbolic_signal_filtered, w, h = apply_lowpass_filter(symbolic_signal, normal_cutoff, order_slider.val)
    symbolic_signal_out = apply_noise(symbolic_signal_filtered, snr_slider.val, P_signal, Covariance)
    # Appliquer le filtre de nyquist :
    filtered_signal = apply_nyquist_filter(symbolic_signal_out, duration, sampling_rate, symbolic_rate, delta, rolloff_factor_slider.val)

    # Retracer les graphes :
    axs_homepage[0,0].cla()
    line0_1 = draw_binary_signal(t, binary_signal, axs_homepage[0,0], "Binary signal", bit_rate_slider.val, duration, val=1.5)
    line0_3.set_data(t, symbolic_signal)
    axs_homepage[1,1].cla()
    line0_4 = draw_spectre(symbolic_signal, axs_homepage[1,1], "Spectrum of the symbolic signal", symbolic_rate, sampling_rate, bit_rate_slider.val)
    line0_5.set_data(t, symbolic_signal_out)
    axs_homepage[2,1].cla()
    line0_6 = draw_spectre(symbolic_signal_out, axs_homepage[2,1], "Spectrum of the symbolic output signal", symbolic_rate, sampling_rate, bit_rate_slider.val)
    line0_7.set_data(t, filtered_signal)
    axs_homepage[3,1].cla()
```

```
    line0_8 = draw_spectre(filtered_signal, axs_homepage[3,1], "Spectrum of the Nyquist-filtered signal", symbolic_rate, sampling_rate, bit_rate_slider.val)

    # Mettre e jour la fen tre principale :
    homepage.canvas.draw_idle()

    # Mettre e jour la fen tre du signal en sortie de canal :
    ax_oeil_outcanal_window.cla()
    drawFullEyeDiagram(symbolic_signal_out, ax_oeil_outcanal_window, 1/bit_rate_slider.val, sampling_rate, "Eye pattern of the output signal")
    oeil_outcanal_window.canvas.draw_idle()

    # Mettre e jour la fen tre de l'oeil de nyquist :
    ax_oeil_window.cla()
    drawFullEyeDiagram(filtered_signal, ax_oeil_window, 1/bit_rate_slider.val, sampling_rate, "Nyquist eye pattern")
    oeil_window.canvas.draw_idle()

    # Mettre e jour le signal d'un seul symbole :
    rect_t, rect_signal = generate_rect_signal(duration/2, sampling_rate, delta, emplacement=1)
    rect_signal_filtered, w, h = apply_lowpass_filter(rect_signal, normal_cutoff, order_slider.val)
    rect_signal_out = apply_noise(rect_signal_filtered, snr_slider.val, P_signal=1, Covariance=1)

    # Retracer les graphes :
    ax1_1.cla()
    line1_1 = draw_signal(rect_t, rect_signal, ax1_1, "Input signal", bit_rate_slider.val, duration/(2+int(bit_rate_slider.val/2)), val=1.5)
    ax1_2.cla()
    line1_2 = draw_spectre(rect_signal, ax1_2, "Spectrum of the input signal", bit_rate_slider.val, sampling_rate, bit_rate_slider.val)
    ax1_3.cla()
    line1_3 = draw_signal(rect_t, rect_signal_out, ax1_3, "Output signal", bit_rate_slider.val, duration/(2+int(bit_rate_slider.val/2)), val=1.5)
    ax1_4.cla()
    line1_4 = draw_spectre(rect_signal_out, ax1_4, "Spectrum of the output signal", bit_rate_slider.val, sampling_rate, bit_rate_slider.val)
    ax1_5.cla()
    line1_5 = ax1_5.semilogx(w*sampling_rate/2, 20 * np.log10(abs(h)))
    ax1_5.set_title('Frequency response of the canal\n(Butterworth low-pass filter)', fontsize=20*dsize)
    ax1_5.set_xlabel('Frequency [rad/s]', fontsize=15*dsize, loc='right')
    ax1_5.set_ylabel('Gain [dB]', fontsize=15*dsize)
    ax1_5.grid(which='both', axis='both')
    ax1_5.axvline(cutoff_frequency_slider.val, color='green')

    # Mettre e jour la fen tre de l'influence du canal :
    canal_window.canvas.draw_idle()

```

```
    # Mettre e jour le signal d'un seul symbole pour le filtre de nyquist :
    rect_t_ny, rect_signal_ny = generate_rect_signal(duration, sampling_rate, 1/symbolic_rate, emplacement=int(symbolic_rate*duration/10)-1)
    rect_signal_ny_filtered = apply_nyquist_filter(rect_signal_ny, duration, sampling_rate, symbolic_rate, delta, rolloff_factor_slider.val)

    # Retracer les graphes :
    ax2_1.cla()
    line2_1 = draw_signal(rect_t_ny, rect_signal_ny, ax2_1, "Input signal", bit_rate_slider.val, duration/5, val=1.5)
    ax2_2.cla()
    line2_2 = draw_spectre(rect_signal_ny, ax2_2, "Spectrum of the input signal", symbolic_rate, sampling_rate, bit_rate_slider.val)
    ax2_3.cla()
    line2_3 = draw_signal(rect_t_ny, rect_signal_ny_filtered, ax2_3, "Filtered signal", bit_rate_slider.val, duration/5, val=1.5)
    ax2_4.cla()
    line2_4 = draw_spectre(rect_signal_ny_filtered, ax2_4, "Spectrum of the filtered signal", symbolic_rate, sampling_rate, bit_rate_slider.val)
    ax2_5.cla()
    ax2_6.cla()
    plot_nyquist(duration, bit_rate_slider.val, sampling_rate, delta, rolloff_factor_slider.val, ax2_5, ax2_6)

    # Mettre e jour la fen tre du filtre de nyquist :
    nyquist_window.canvas.draw_idle()



# LES GLISSIERES :

# Glissiere de debit :
bit_rate_slider_ax = homepage.add_axes([0.075, 0.15, 0.01, 0.75])
bit_rate_slider = Slider(
    ax=bit_rate_slider_ax,
    label='Binary\nrate [Bit/s]',
    valmin=1,
    valmax=8,
    valinit=bit_rate,
    valstep=1,
    orientation="vertical"
)
bit_rate_slider.on_changed(regenere)

# Glissiere de l'ordre du filtre representant le canal :
order_ax = canal_window.add_axes([0.025, 0.1, 0.0225, 0.75])
order_slider = Slider(
    ax=order_ax,
    label="Order",
    valmin=order,
    valmax=5*order,
    valinit=order,
    valstep = 1,
    orientation="vertical"
```

```
)
order_slider.on_changed(update)

# Glissiere de la frequence de coupure du filtre representant le canal :
cutoff_frequency_ax = canal_window.add_axes([0.14, 0.1, 0.0225, 0.75])
cutoff_frequency_slider = Slider(
    ax=cutoff_frequency_ax,
    label='Cutoff\nfrequency [Hz]',
    valmin=0.25,
    valmax=15,
    valinit=cutoff_frequency,
    valstep=0.25,
    orientation="vertical"
)
cutoff_frequency_slider.on_changed(update)

# Glissiere du rapport signal sur bruit du canal :
snr_ax = canal_window.add_axes([0.14, 0.025, 0.75, 0.0225])
snr_slider = Slider(
    ax=snr_ax,
    label='SNR [dB]',
    valmin=0,
    valmax=60,
    valinit=snr,
    valstep=1,
    orientation="horizontal"
)
snr_slider.on_changed(update)

# Glissiere du facteur de roll-off :
rolloff_factor_ax = nyquist_window.add_axes([0.05, 0.1, 0.0225, 0.75])
rolloff_factor_slider = Slider(
    ax=rolloff_factor_ax,
    label='roll-off\nfactor',
    valmin=0,
    valmax=1,
    valinit=rolloff_factor,
    valstep=0.05,
    orientation="vertical"
)
rolloff_factor_slider.on_changed(update)



# LES BOUTONS :

# Bouton de l'oeil du signal de sortiee de canal :
def oeil_outcanal(event):
    ax_oeil_outcanal_window.cla()
    drawFullEyeDiagram(symbolic_signal_out, ax_oeil_outcanal_window, 1/
    bit_rate_slider.val, sampling_rate, "Eye pattern of the output signal
    ")
    oeil_outcanal_window.canvas.draw_idle()
    oeil_outcanal_window.show()

oeil_outcanal_ax = homepage.add_axes([0.92,0.38,0.05,0.04])
button_oeil_outcanal = Button(oeil_outcanal_ax, 'Eye', hovercolor='0.975
    ')
```

```
868 button_oeil_outcanal.on_clicked(oeil_outcanal)
869
870 # Bouton de l'oeil de nyquist :
871 def oeil_nyquist(event):
872     ax_oeil_window.cla()
873     drawFullEyeDiagram(filtered_signal, ax_oeil_window, 1/
    bit_rate_slider.val, sampling_rate, "Nyquist eye pattern")
874     oeil_window.canvas.draw_idle()
875     oeil_window.show()
876
877 oeil_ax = homepage.add_axes([0.92,0.175,0.05,0.04])
878 button_oeil = Button(oeil_ax, 'Eye', hovercolor='0.975')
879 button_oeil.on_clicked(oeil_nyquist)
880
881 # Bouton radio de codage :
882 def codage(label):
883     global code_name
884     code_name=label
885     button_codage.labels[0].set_weight('normal')
886     button_codage.labels[1].set_weight('normal')
887     button_codage.labels[2].set_weight('normal')
888     button_codage.labels[3].set_weight('normal')
889     button_codage.labels[4].set_weight('normal')
890     button_codage.labels[5].set_weight('normal')
891     button_codage.labels[6].set_weight('normal')
892     button_codage.labels[7].set_weight('normal')
893     ptr = 0
894     while list_code[ptr]!=code_name:
895         ptr = ptr + 1
896     button_codage.labels[ptr].set_weight('bold')
897     update(label)
898     homepage.canvas.draw()
899 codage_ax = homepage.add_axes([0.63,0.725,0.25,0.25])
900 codage_ax.set_facecolor('0.9')
901 button_codage = RadioButtons(
902     codage_ax,
903     list_code
904 )
905 button_codage.on_clicked(codage)
906 button_codage.labels[0].set_weight('bold')
907 for r in button_codage.labels:
908     r.set_fontsize(20*dsize)
909     r.set_fontstyle('italic')
910
911 # Bouton de canal :
912 def canal(event) :
913     canal_window.show()
914 canal_ax = homepage.add_axes([0.115,0.5,0.06,0.04])
915 button_canal = Button(canal_ax, 'Channel', hovercolor='0.975')
916 button_canal.on_clicked(canal)
917
918 # Bouton de nyquist :
919 def nyquist(event):
920     nyquist_window.show()
921 nyquist_ax = homepage.add_axes([0.115,0.275,0.07,0.04])
922 button_nyquist = Button(
923     nyquist_ax,
924     'Nyquist',
```

```
    hovercolor='0.975'
)
button_nyquist.on_clicked(nyquist)
```

A.4.3 Animation de l'oeil de Nyquist

```
from matplotlib import rc
%matplotlib qt

print("Signal e representer :\n1 : Symbolic output signal\n2 : Nyquist-filtered signal")
choice=0
while(choice!=1 and choice!=2):
    choice = int(input("Faites votre choix (1 ou 2) : "))
    if choice==1 :
        signal = symbolic_signal_out
        print("Signal = Symbolic output signal\n")
    elif choice==2 :
        signal = filtered_signal
        print("Signal = Nyquist-filtered signal\n")
    else :
        print("Erreur ! Veuillez-recommencer")


delta=1/bit_rate_slider.val
ratio_bi_sy_rate = bit_rate_slider.val/symbolic_rate

fig=plt.figure("Animation of the eye pattern", figsize=(width_window/1.5,height_window/1.5))
axs = fig.subplots(2,1) #2 lignes et une colonne
plt.close()

text = str()
compteur = 0
a = np.zeros(int(duration*bit_rate_slider.val/(2/ratio_bi_sy_rate)))  # a = suite de symboles
for i in range(len(a)):
        a[i] = binary_signal[int(((sampling_rate/bit_rate_slider.val)/2)+(sampling_rate/bit_rate_slider.val)*i)]
j=0
while j<len(a)-1 :
    if (compteur%10==0 and compteur!=0) :
            text+='...\n'
            compteur=0
    text += str(int(a[j])) + ' ' + str(int(a[j+1])) + ' | '
    j+=2
    compteur+=1

print("Binary sequence :\n")
print(text, '...')
print('\n')

text = str()
compteur = 0
a = np.zeros(int(duration*symbolic_rate/(2/ratio_bi_sy_rate)))  # a = suite de symboles
```

```
for i in range(len(a)):
        a[i] = symbolic_signal[int(((sampling_rate/symbolic_rate)/2)+(sampling_rate/symbolic_rate)*i)]
if (delta==1/symbolic_rate) :
    j=0
    while j<len(a)-1 :
        if (compteur%10==0 and compteur!=0) :
            text+='...\n'
            compteur = 0
        text += str(int(a[j])) + ' ' + str(int(a[j+1])) + ' | '
        j+=2
        compteur+=1
else :
    j=0
    while j<len(a)-3 :
        if (compteur%10==0 and compteur!=0) :
            text+='...\n'
            compteur = 0
        text += str(int(a[j])) + ' ' + str(int(a[j+1])) + ' ' + str(int(a[j+2])) + ' ' + str(int(a[j+3])) + ' | '
        j+=4
        compteur+=1

print("Symbolic sequence :\n")
print(text, '...')

t_eye1 = np.arange(0,2*delta,1/sampling_rate)
t_eye2 = np.arange(0,2*delta,1/sampling_rate)
axs[0].set_xlim(0,int(duration*ratio_bi_sy_rate/2))
axs[0].set_ylim(-2,2)
axs[0].set_title("Animation of the eye pattern", fontsize=10)
axs[0].set_ylabel('Amplitude', fontsize=8)
axs[0].xaxis.set_major_formatter(formatter(delta, chr(8710)))
axs[0].xaxis.set_major_locator(MultipleLocator(4*delta))
axs[0].xaxis.set_minor_locator(MultipleLocator(1*delta))
axs[0].yaxis.set_minor_locator(MultipleLocator(0.1))
axs[0].grid(axis = 'y', linestyle='--', color='grey', linewidth=0.75)
for i in range(int(len(a))):
    axs[0].axvline(delta*i*ratio_bi_sy_rate, color='g', linestyle = '--', linewidth=0.75)

axs[1].set_xlim(0,2*delta)
axs[1].set_ylim(-2,2)
axs[1].set_ylabel('Amplitude', fontsize=8)
axs[1].xaxis.set_major_formatter(formatter(delta, chr(8710)))
axs[1].xaxis.set_major_locator(MultipleLocator(delta))
axs[1].xaxis.set_minor_locator(MultipleLocator(0.1*delta))
axs[1].yaxis.set_minor_locator(MultipleLocator(0.1))
axs[1].grid(axis = 'both', linestyle='--', color='grey', linewidth=0.75)

lines = [[],[]]

for i in range(int(duration/(4*delta/ratio_bi_sy_rate))):
    t_eye2 = np.concatenate((t_eye2,t_eye1))

# Fonction d'animation :
def animate(frame):
    start_index = int((frame + 1) * 2 * delta * sampling_rate) + 1
```

```
    end_index = int((frame + 2) * 2 * delta * sampling_rate) - 1
    lines[0].append(axs[0].plot([], [])[0])  # Creer une nouvelle ligne et l'ajouter e la liste
    lines[1].append(axs[1].plot([], [])[0])
    lines[0][frame].set_data(t[start_index:end_index], signal[start_index:end_index])
    lines[1][frame].set_data(t_eye2[start_index:end_index], signal[start_index:end_index])
    return lines[0] + lines[1] # Retourner toutes les lignes

ani = animation.FuncAnimation(fig, animate, frames=int(duration/(4*delta/ratio_bi_sy_rate)), blit=True, interval=200, repeat=False)

rc('animation', html = 'jshtml')
ani
```

A.4.4 Verification SBPA

```
bit_rate = 1
t, signal = generate_binary_signal(duration, sampling_rate, bit_rate)
a = np.zeros(int(duration*bit_rate))  # a = suite de symboles
t_ = np.zeros(len(a))
for i in range(len(a)):
    a[i] = signal[int(((sampling_rate/bit_rate)/2)+(sampling_rate/bit_rate)*i)]
    t_[i] = t[int(((sampling_rate/bit_rate)/2)+(sampling_rate/bit_rate)*i)]
    if a[i]==0 : a[i]=-1
delta=1/bit_rate
print(a)
Rxx=0
for i in range(int(len(a)*delta)) :
    Rxx+=a[i]*a[int(i-delta)]
    print(Rxx)
Rxx=Rxx/(len(a)*delta)
print('\n')
print('Rxx = ' + str(Rxx))
R = np.zeros(int(duration*bit_rate))
R[0] = np.sqrt((Rxx*len(a))**2)
R[-1] = R[0]
R[1:-2] = Rxx
Fig = plt.figure("Fonction d'auto-correlation du signal binaire")
ax = Fig.add_subplot(111)
ax.set_title("Fonction d'auto-correlation du signal binaire")
ax.plot(t_, R)
ax.set_xlabel("t [s]")
ax.set_ylabel("Rx(t)")
ax.set_xlim(-0.01*len(a)*delta,1.01*len(a)*delta)
ax.set_ylim(-10,R[0]+10)
ax.xaxis.set_major_locator(MultipleLocator(bit_rate*delta))
ax.yaxis.set_major_locator(MultipleLocator(bit_rate))
ax.grid(axis = 'both', linestyle='--', color='grey', linewidth=0.75)
Fig.show()
```