Compte-Rendu TP Communication Numérique

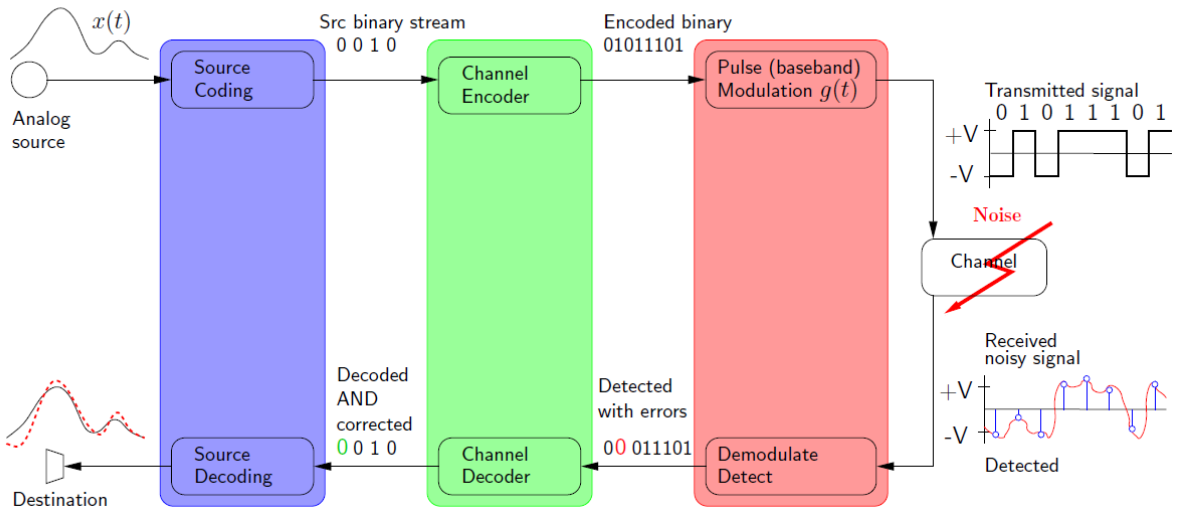

Schéma 1: Chaîne de communication

Préambule

L'ensemble du code rendu lors du TP est documenté et versionné sur la plateforme GitHub à l'adresse https://github.com/AxFrancois/Numeric-Comm. Par ailleurs, lors de notre finalisation des codes nous avons constaté un nombre important de personnes ayant visités notre répertoire (plus de 10 !), il est donc probable que quelques malins nous aient plagiés.

Pour des raisons de performances et pour limiter le temps de simulation, nous utilisons le librairie python « os » afin de réaliser de la programmation concurrente. Cette librairie ne fonctionnant que sur les systèmes Unix., notre code ne sera pas exécutable sur une machine Windows ou Mac OS. Pour donner une ordre d'idée, le temps de simulation sur une machine virtuelle équipée de 6 cœurs et fréquences à 2,60 GHz est d'environ 30 min.

Introduction

Le but de ces 3 TPs est de réaliser la chaîne de communication présentée ci-dessus. Le premier TP consistait à faire de la compression de source. Cette compression de source est représentée par la couleur bleue sur le schéma ci-dessus. Le second TP était lui axé sur le codage canal représenté par la couleur verte sur le schéma. Enfin le dernier TP a permis de mettre en place la modulation et la transmission de données dans un canal gaussien. Cette modulation est représentée par la couleur rouge sur notre schéma. Chaque TP était la suite de celui fait précédemment et permettait de rajouter un morceau à la chaîne de communication jusqu'à en avoir une complète.

TP Codage source

Avec ce premier TP, nous allons étudier la compression de la source par l'utilisation d'un code de Huffman à partir d'un message texte. Nous allons également simuler la transmission de mots-codes à travers un canal à erreurs. Dans un premier temps, après l'installation des différentes bibliothèques nécessaires, nous devons tracer l'histogramme de l'alphabet avec la probabilité d'apparition de chaque caractère dans le texte qui nous a été fourni. Nous obtenons la figure suivante :

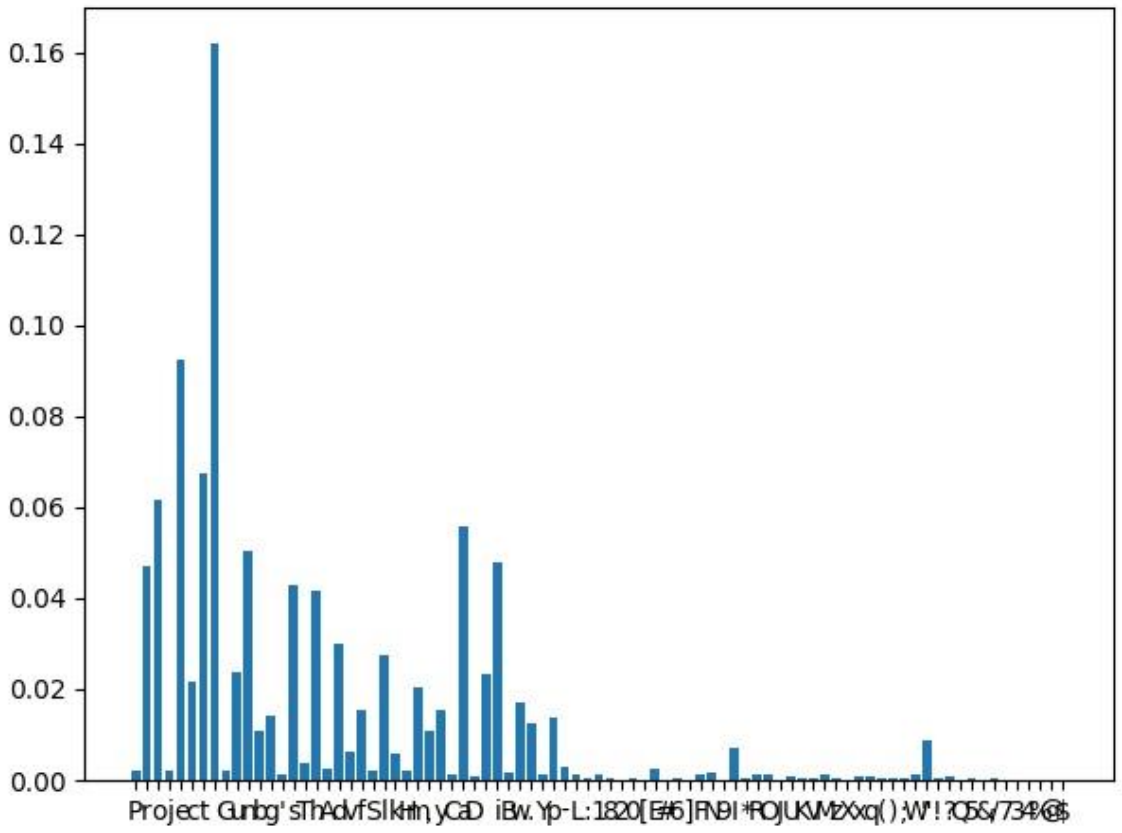

Figure 1: Probabilité d'apparition de chaque caractère

Une fois cet histogramme tracé, nous devons calculer l'entropie de notre alphabet. A l'aide de la formule du cours $H(X) = -\sum_i p_i \, log_2 p_i$, nous obtenons 4,614 bits comme valeur d'entropie. Cela signifie qu'en moyenne chaque symbole sera transmis sur 4,614 bits.

Ensuite, nous devons appliquer un algorithme de codage de source Huffman. Le codage de Huffman permet de construire un code qui est efficace, instantané et uniquement décodable. Ce code Huffman permet de minimiser la taille de représentation de l'information. A l'aide de cet algorithme, nous avons pu dresser la table du codec Huffman et ainsi voir le code correspondant à chaque caractère présent dans notre texte. On peut d'ailleurs afficher comment chaque caractère sera encodé, par exemple :

- Le caractère ' ' (espace) est le plus fréquent dans notre texte d'après la figure 1. Il est donc normal que celui-ci soit encodé sur le moins de bit possible (en l'occurrence 3 bits) afin d'optimiser la quantité d'information à transmettre.
- En comparaison, la caractère '@' est l'un des plus rare dans notre texte, et il est encodé sur de nombreux bits (en l'occurrence 16 bits) afin de permettre aux caractères plus fréquents d'être mieux encodé.

A la suite de cela, nous avons encodé puis décodé à l'aide du code Huffman puis nous avons réitéré cette opération en passant par le format binaire afin de préparer notre codage canal que nous effectuerons dans le TP suivant. Ensuite, nous avons converti notre vecteur binaire en une matrice de

k colonnes où k est le nombre correspondant à la taille des messages en entrée du Code détecteur et Correcteur des Erreurs. Pour la suite de notre TP, ce paramètre k aura une valeur de 4.

Nous avons également testé l'efficacité du code Huffman face au bruit. Le taux d'erreur sur un code Huffman bruité avec 1 erreur par bloc de 4 bits est de l'ordre de 22.6 % (entre 28 et 19 selon nos différents essais). Nous substituons 25% des bits par leur opposé, et nous obtenons à peu près le même taux d'erreur. Nous pouvons donc conclure que le codage Huffman ne permet pas de protéger un signal face au bruit lors d'une transmission, ce qui est le résultat attendu et la raison de l'utilisation du codage canal au TP suivant.

TP Codage canal

Le but de ce second TP est d'utiliser les différentes sorties du premier TP décodage-codage Huffman et d'appliquer un codage canal cyclique (aussi appelé code de Hamming) aux bits résultants de la compression source par Huffman afin d'obtenir les mots-codes. On décodera ensuite le code cyclique afin de reconstruire parfaitement la phrase initiale.

Pour réaliser cette opération nous avons besoin d'un polynôme générateur. Nous utiliserons un codage (7,4), et nous avons choisi d'utiliser le polynôme '1011', qui correspond à $x^3 + x^2 + 1$. Ce qu'il faut comprendre par cette phrase, c'est que l'on transforme notre message en bloc de 4 bits d'information auxquels s'ajoute 3 bits de vérification, soit une longueur totale de 7 bits transmis. Ce code présents un bon ratio de transmission, avec 4/7 de bit utile transmis.

Ce format d'encodage permet de corriger 1 erreur et d'en détecter 2. Nous pouvons obtenir ce résultat expérimentalement grâce à notre simulation. Une fois notre signal encodé avec notre polynôme générateur, nous ajoutons une ou plusieurs erreurs à chaque bloc de 7 bits et nous observons l'intégrité des données. On obtient les résultat suivant :

- Taux d'erreur sur un code cyclique bruité avec 1 erreur(s) par bloc de 7 bits 0.0 %.
- Taux d'erreur sur un code cyclique bruité avec 2 erreur(s) par bloc de 7 bits 19.0634 %.
- Taux d'erreur sur un code cyclique bruité avec 3 erreur(s) par bloc de 7 bits 25.7754 %.

Conformément à la propriété du code (7,4), lorsqu'une seule erreur se présente, elle est corrigée sans problème. Cependant, dès que plusieurs erreurs apparaissent, l'algorithme ne permet plus de les corriger.

TP Modulation et transmission dans un canal gaussien

Lors de ce dernier TP, à l'aide des deux derniers TP effectués, nous allons moduler les bits de source codés et non-codés. Nous allons appliquer deux modulations différentes : la 4-QAM et la 16-QAM. Puis nous simulerons la transmission des symboles modulés à travers le canal gaussien AWGN (Add White Gaussian Noise). La modulation permet de transformer et d'adapter le flux binaire (récupéré dans le dernier TP) en fréquence aux canaux de notre système de transmission.

Pour notre chaîne de communication, nous avons choisi d'utiliser une modulation 16-QAM, avec un signal qui sera bruité avec un rapport signal à bruit de 8 dB (décibels). Nous pouvons alors tracer sur un graphique la constellation de notre modulation ainsi que la position des points qui ont été reçu. On observe que ces points ont été décalés à cause du bruit. Le but de la démodulation est donc de

remédier à ces petits écarts en phase et en amplitude pour permettre aux points de revenir à leurs valeurs d'origine. Lorsque le rapport signal à bruit est trop faible, ces points sont donc trop écartés de leur position d'origine et le signal est modifié lors de la transmission car la démodulation ne réattribue pas correctement les points. Voici donc le graphique de la constellation et des points transmis pour notre texte.

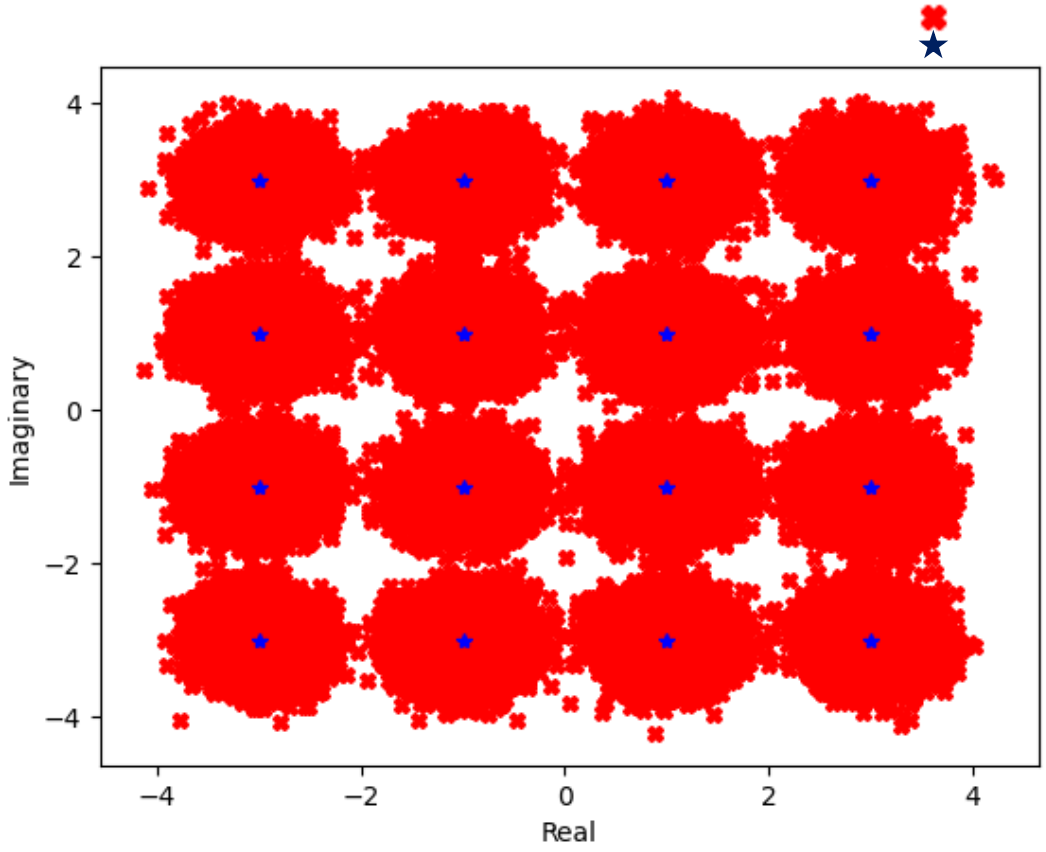

Figure 2: Constellation de la modulation

Nous pouvons également étudier les performances des différents modes de modulation QAM (4-QAM, 16-QAM, 64-QAM et 256-QAM) et leurs sensibilités par rapport au bruit. Nous avons donc effectué, grâce à plusieurs boucles, plusieurs essais avec différents types de modulation et différents ratios signal à bruit. Nous obtenons alors la courbe suivante (voir figure 3). Comme nous pouvons le constater avec la courbe rouge et verte, les résultats obtenus expérimentalement sont très proches des résultats théoriques ce qui valide bien le bon fonctionnement de notre système.

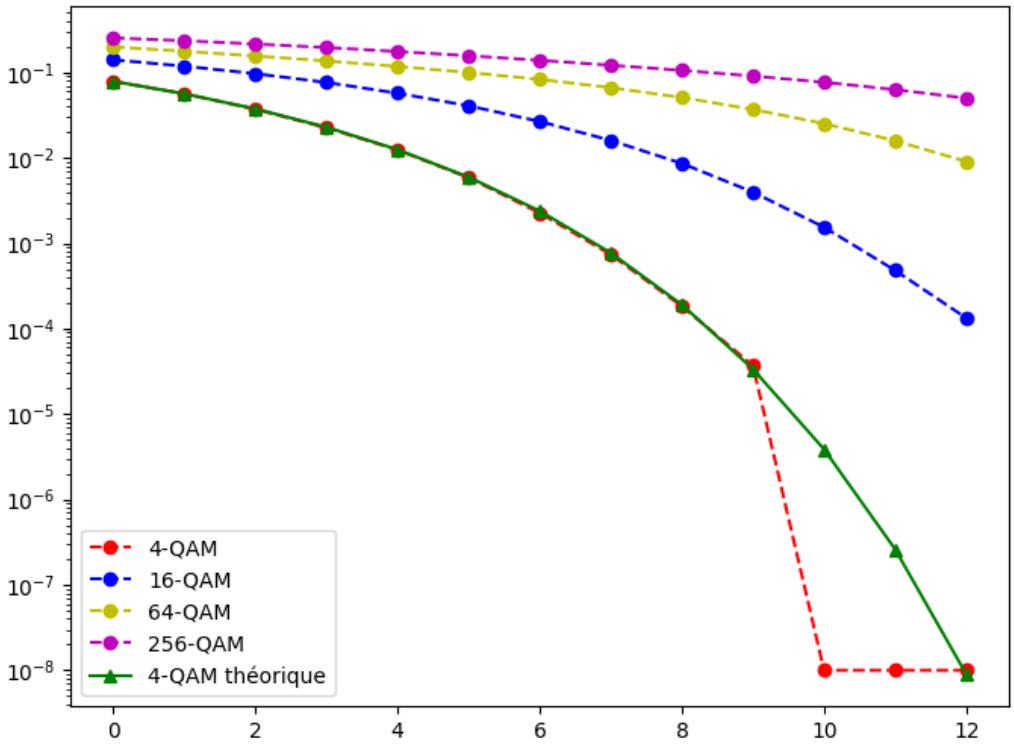

Figure 3 : Modélisation modulation M-QAM (M = 4, 16, 64, 254)

Nous remarquons, cependant, que la courbe en rouge effectue un petit saut brusque lors d'un rapport signal à bruit de 11 et 12 décibels. Cela est dû au fait que malgré la taille du texte transmis, le système n'a pas commis d'erreurs et le signal fut intégralement restauré (nous avons implémenté une erreur limite de 10^{-8} afin de rendre la courbe traçable sur l'échelle semi-logarithmique). Il faudrait, pour avoir des résultats expérimentaux corrects, transmettre un texte beaucoup plus grand de l'ordre de plusieurs millions de bits ($100 * \frac{1}{10^{-8}} = 10\ milliards\ de\ bits$!).

Une fois la figure 3 tracée, nous avons affiché le taux d'erreur suite à la modulation QAM pour un texte codé (par codage cyclique) et un texte non-codé. Nous obtenons alors la figure 4. Nous pouvons constater que l'écart entre les courbes avec le codage cyclique et les courbes sans le codage cyclique pour le même M est proportionnel. Nous pouvons donc en déduire que le codage cyclique permet de diminuer fortement la puissance nécessaire pour la transmission à un taux d'erreur fixé.

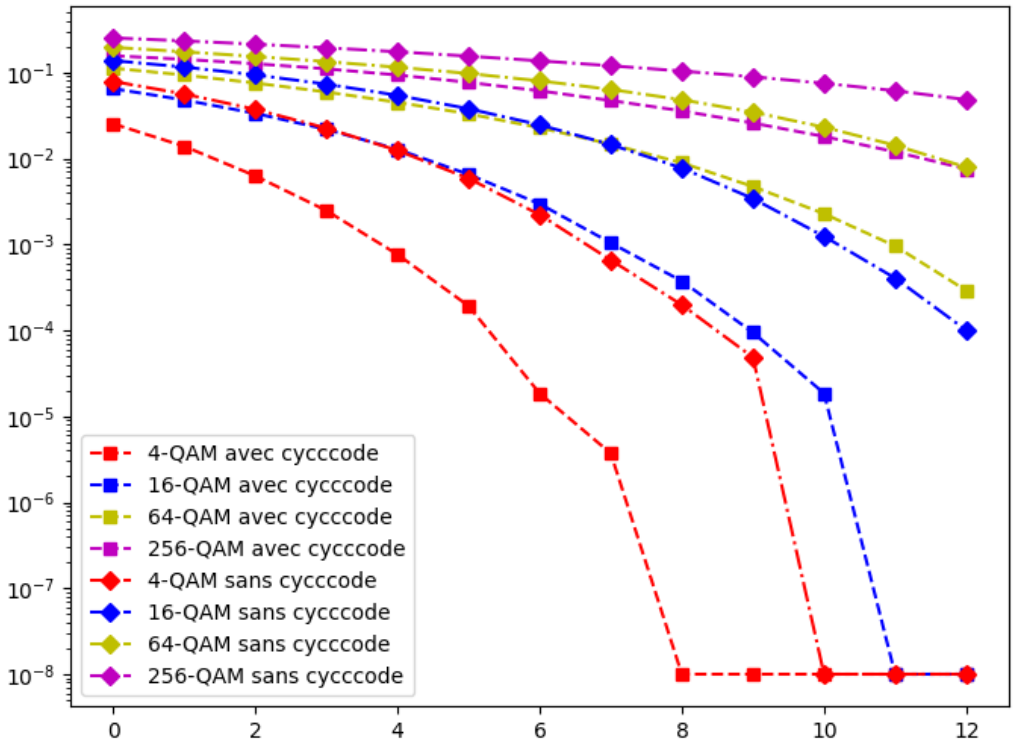

Figure 4 : Modélisation modulation M-QAM avec ou sans code cyclique (M = 4, 16, 64, 254)

Conclusion

Lors de ces trois TPs, nous avons pu mettre en pratique les différentes notions apprises en cours. Nous avons pu apprendre le fonctionnement d'une chaîne de communication entre une source et sa destination avec les différentes erreurs qui peuvent apparaître et comment elles peuvent être corrigées. Nous avons pu constater également qu'il faut faire des compromis entre la capacité d'envoi du signal et sa résilience face aux erreurs.