TP n3 : Modulation numérique et transmission dans un canal gaussien

Nikolai Lebedev[1,2]
Contacts : lebedev@cpe.fr
[1]CPE Lyon, BP 2077, F-69616, France
[2]University of Lyon, INRIA, INSA-Lyon, CITI lab, F-69621, Villeurbanne, France

I. Objectifs de ce TP

- **Reprendre les TPs n1, n2 du codage-décodage source Huffman et codage canal cyclique**
- **Moduler les bits de source non-codés et codés.**
- **Appliquer la modulation 4-QAM et 16-QAM**
- **Simuler la transmission des symboles modulés à travers le canal gaussien AWGN**
- **Démoduler, détecter et décoder les signaux transmis.**
- **Mettre en évidence les erreurs et comparer la probabilité d'erreur sur canal AWGN sans et avec le codage canal pour les deux modulation 4-QAM et 16-QAM**

II. Consignes et outils

1) Python, PyCharm/venv, libraries numpy, duhaffman, scikit-dsp-comm, scikit-commpy :
   `>>> pip install dahuffman scikit-dsp-comm scikit-commpy`
2) (En option : Matlab® / Comm Toolbox), sources complémentaires théorie et exemples :
   `https://fr.mathworks.com/help/comm/ug/error-detection-and-correction.html`
3) Lecture et étude d'un notebook Matlab "`16_QAM_Matlab.mlx`" fourni
4) `https://fr.mathworks.com/help/comm/ref/qammod.html`
   `https://fr.mathworks.com/help/comm/ref/qamdemod.html`

III. Introduction

Dans tout système de communication via n'importe quel canal média,—cuivre, radio, optique, la largeur de la bande des fréquences utilisable est limitée, et son contenu est situé dans une zone de spectre bien définie (basses fréquences $0 - 2 \cdot 10^6$ Hz $= 2$ MHz en ADSL, microondes vers $\approx 10^9$ Hz $= 1$ GHz en cellulaire 4G, ou dans le spectre visible vers $\approx 500 \cdot 10^{12}$ Hz $= 500$ THz en Ethernet sur fibre optique.

Chacun de ces systèmes peut être utilisé pour transmettre le même **message ou signal d'information utilisateur** dont le contenu spectral est différent du **système ou signal porteur** utilisé pour le véhiculer, transmettre. Par exemple, le spectre du signal de la voix est contenu dans $[300-3400]$ Hz, et il pourrait être transmis soit via le câble (projet modulation 3ETI-SSL), soit via ADSL, soit via un téléphone portable, soit par un système VoIP via Ethernet sur fibre optique.

En communications numériques, le signal d'information est préalablement numérisé par un système de codage de source qui produit le flux binaire $\{0, 1\}$. On a donc besoin de le transformer et adapter aux canaux en fréquence du système de transmission. Cette transformation s'appelle la **modulation**, dont le rôle est de :

1) Convertir ou *mapper* un bloc de k bits du message d'information source $\rightarrow$ en symbol transmis $m_i, i = 1, \ldots, M = 2^k$, binaire ($M = 2$) ou M-aire.
2) Créer un signal physique en bande de base ($\approx 0 - 2$ MHz), c'est à dire une impulsion électrique de durée et de contenu spectral défini, en utilisant un filtre de mise en forme (pulse shaping) $g(t)$ en créant une forme d'onde $g_i(t), i = 1, ..., M$ pour chaque i-ème symbole transmis des M possibles.

A ce stade, on pourra communiquer en bande de base, par exemple dans un système téléphonique ; ce type de communication s'appelle les « codes en ligne » (NRZ, RZ, Manchester, HDB3).

3) Translater ou transposer le contenu spectral du signal d'information dans la bande su système de transmission, lorsque celui-ci n'est pas en bande de base, mais dans une autre bande de fréquences, par exemple, de téléphonie cellulaire. Le signal résultant est $s_i(t)$, résultat de la translation et du centrage du spectre défini par $g_i(t)$ autour de la fréquence porteuse $f_p \gg \text{Bande}\{g_i(t)\}$.

On dit que le **signal d'information module la porteuse f_p**, en changeant une ou plusieurs de ses caractéristiques—amplitude, fréquence ou phase. La porteuse, elle, ne contient aucune information !

Lorsque le message d'information est composé d'un ensemble fini, discret, de symboles M-aires, le nombre d'états que peut prendre le paramètre modifié de la porteuse, par exemple, son amplitude, est égal à M. On parle alors des **modulations numériques**.

Le récepteur de son côté va récupérer le message d'information à partir du signal $s_i(t)$ transmis à travers le canal bruité par un procédé de **démodulation**.

L'étude de la modulation par simulation se fait à l'aide des modules Python cités dans la section II.

IV. Simulation de la modulation : mapping

Matlab et Python permettent de simuler les différentes modulations à la fois en bande de base (coût de calcul moindre) et en fréquence porteuse. Pour les modulations numériques qui nous intéressent, seule la simulation en bande de base est à réaliser.

Message d'information en bande de base définit la forme de changement imprimé sur les propriétés de la porteuse, il est aussi appelé l'enveloppe complexe en bande de base.

A. Représentation des symboles de modulation

1) Mapping: La première fonction du modulateur est donc le *mapping* : la collecte de k bits en provenance de la source numérisée (ex : codec voix), ou du codeur canal, et sa mise en correspondance avec un des symboles $m_1, m_2, ..., m_M$ des $M = 2^k$ symboles possibles.

Remarque. *Certaines fonctions de modulation en Matlab peuvent accepter à l'entrée non pas les blocs de k bits, mais leurs valeurs décimales, numérotant ainsi les symboles. Par exemple, en 16-QAM, au lieu de donner $[1110]_b$ (MSB à gauche), on peut donner 14_d. A la réception, de la même manière, le démodulateur peut avoir en sa sortie soit les bits, soit l'entier, le numéro du symbole.*

2) Ordre des symboles dans la constellation: Rappelons, que chaque signal-symbole m_i peut être représenté comme un vecteur dans un espace de dimension K, cet espace généré par les fonctions de base $\{f_\ell(t)\}, \ell = 1, ..., K$. Un tel vecteur a pour ses éléments les coordonnées, ou projections du signal sur chaque fonction de base, c'est à dire :

$$m_i = [m_i(1), ..., m_i(K)]$$

Pour les modulations de type M-QAM, la dimension de l'espace des signaux est $K = 2$, on peut donc visualiser les vecteurs m_i des symboles sur un plan,—c'est la **constellation** de la modulation. Chaque symbole est numéroté, par exemple, dans l'ordre de colonnes, de 0 à $M - 1$, c'est à dire, l'ordre dit naturel.

Remarque. *La manière de mapper, les k bits sur un symbole n'est pas unique.*

Considérons 2 principales techniques de mapping.

Ordre naturel *(symbol encoding)*, qui peut être choisi par une option `'binary'` dans certaines fonctions de Toolbox Matlab. Ex : Le 14-ème symbole a pour étiquette $[1110]_b = 14_d$.

Codage de Gray, qui n'est qu'un autre étiquetage binaire de chaque symbole, différent de leur numéro décimal ; c'est donc juste une permutation. Ce mode peut être choisi par une option `'Gray'` dans

certaines fonctions de Toolbox Matlab. L'astuce de cette technique est d'attribuer aux symboles voisins dans la constellation, plus proches au sens de la distance euclidienne, les codes binaires qui ne diffèrent que d'un seul bit ; les codes des symboles en diagonal, eux, plus éloignés, peuvent différer de deux bits. Cela part d'un constat que lors de la réception depuis un canal bruité, l'erreur de la démodulation la plus probable serait de se tromper dans la détection en faveur du symbole voisin, le plus proche, $\hat{m}_j$ par rapport au symbole original transmis m_i, qui est ensuite converti en k bits. Il faut donc qu'un tel événement affecte le moins de bits possible. Une telle technique permet de réduire la probabilité d'erreur binaire (PEB).

Ex : le 14_d-ème symbole est codé sur $[1011]_b = 11_d$.

Le code Gray 1D pour un code binaire $c_b = [c(k)...c(1)]$ est obtenu via l'opération suivante [1] :

$$c_g = c_b \oplus \left\lfloor \frac{c_b}{2} \right\rfloor$$

Ex : M-PAM, Pulse Amplitude Modulation. 8-PAM code Gray 1-dimensionnel. Le code Matlab :

```
M = 8; i = [0:M-1]'; % vecteur-colonne [0 1 2 ... 7]
i_gray = bitxor(i,floor(i/2));
i_b = de2bi(i_gray), log2(M),'left-msb');
```

ce qui donne le résultat suivant :

N^o de symbole, i	0	1	2	3	4	5	6	7
Binary, c_b	000	001	010	011	100	101	110	111
N^o de symbole en Gray, i_{gray}	0	1	3	2	6	7	5	4
Binary Gray, c_g	**000**	**001**	**011**	**010**	**110**	**111**	**101**	**100**

Ex : Modulation 16-QAM, le codage de Gray 2-dimensionnel est utilisé, l'expression théorique de cette permutation n'est pas triviale ! Le résultat, présenté dans la table et sur la Fig. 1 est typiquement tabulé [2] dans les outils de simulation ou mis directement dans les circuits logiques.

N^o symbole	0	1	2	3	4	5	6	7
Binary	0000	0001	0010	0011	0100	0101	0110	0111
Gray (i_{gray})	**0000(0)**	**0001(1)**	**0011(3)**	**0010(2)**	**0100(4)**	**0101(5)**	**0111(7)**	**0110 (6)**
N^o symbole	8	9	10	11	12	13	14	15
Binary	1000	1001	1010	1011	1100	1101	1110	1111
Gray (i_{gray})	**1100(12)**	**1101(13)**	**1111(15)**	**1110(14)**	**1000(8)**	**1001(9)**	**1011(11)**	**1010 (10)**

Remarque. *Notons que les symboles sont toujours numérotés dans l'ordre* $0, ..., M-1$*, colonne par colonne.*

En résumé, la correspondance bits–symboles fonctionne comme suit (voir Fig.1) :

1) Transmission
   - Prendre un bloc de k bits depuis la source. Ex : $c_b = [1011]_b = 11_d$.
   - Appliquer le codage de Gray, en binaire ou en décimal $11_d \rightarrow 14_d$, typiquement par la lecture dans une table, qui est soit à programmer, soit à utiliser implicitement via une fonction qui fait la modulation QAM, avec un argument 'gray'.

1. Un pseudo-code pour la conversion d'un vecteur binaire → Gray serait : x `XOR` $(x >> 1)$.
(src : `http://www.dspguru.com/comp.dsp/tricks/alg/grayconv.htm`)
2. Donc basé sur *LUT, LookUp Table*

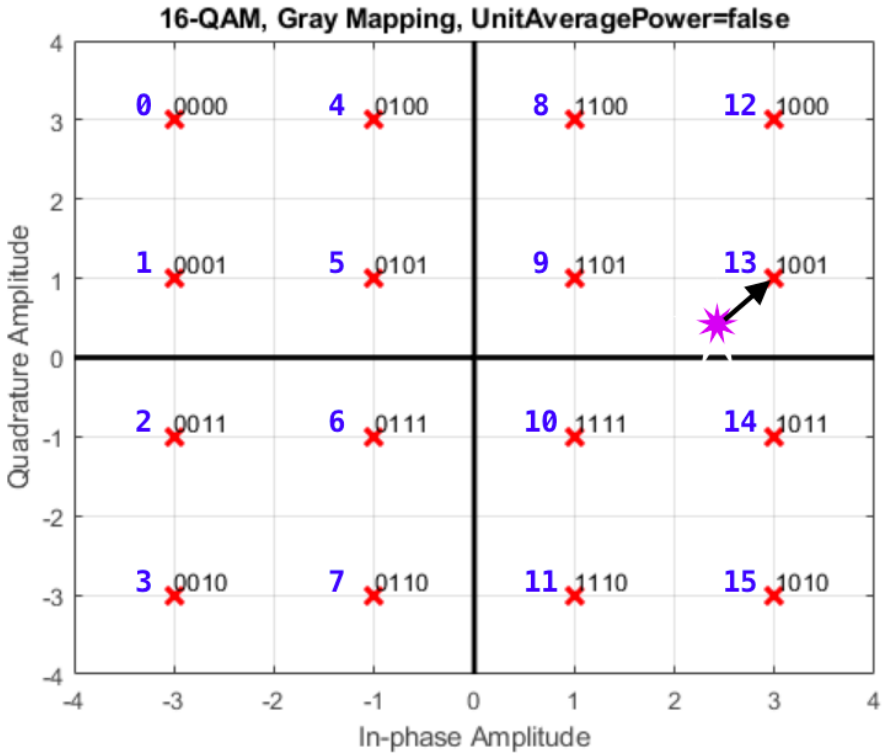

FIGURE 1: *Codage de Gray pour 16-QAM.*

— Transmettre le symbole N° 14 selon l'ordre naturel, avec les amplitudes en phase (I) et en quadrature (Q) qui lui sont propres, $(I, Q) = (3, -1)$.

2) Canal
— Le canal introduit du bruit, et le signal reçu ne correspond à aucun des symboles (Fig. **??** : ★).

3) Réception
— L'événement d'erreur le plus probable est celui où l'échantillon bruité reçu est le plus proche au sens de la distance d'un des symboles voisins du symbole transmis, en faveur duquel le démodulateur fait la décision. Supposon, dans notre cas, que ce sera le symbole N^{o} 13 selon l'ordre naturel qui a $(I, Q) = (3, 1)$.
— Le décodage de Gray convertit le numéro de symbole $13_d \rightarrow 9_d$ et donc en son étiquette binaire de Gray de k bits $[10 \overset{*}{0} 1]$. Une erreur sur un seul bit se produit (marqué par $(*)$).
— En sortie du démodulateur, ces k bits, dont certains erronés, sont présentés au bloc de traitement suivant, par exemple, reconstruction de source avec distortion, ou décodage code canal.

Remarque. *En comparaison, si l'on utilisait l'ordre naturel et transmettait le symbole* $c_b = 11_d = [1011]_b$, *la décision en faveur du symbole* $7_d = [\overset{**}{01}\ 11]_b$ *produirait 2 erreurs binaires.*

Remarque. *En faisant attention, on peut observer qu'une modulation* M*-QAM rectangulaire, ou* $M = 2^k$, *pour* k *pair, peut être vue en termes algorithmiques comme combinaison de modulations d'amplitude,* M'*-PAM, où* $M' = \sqrt{M}$ *sur chacune des deux dimensions. Ainsi, les* $(k/2)$ *bits de poids fort (MSB) marquent la colonne, ici en Gray* $[00\ 01\ 11\ 10]_b = [0\ 1\ 3\ 2]_d$, *et les* $(k/2)$ *bits de poids faible (LSB), de la même manière, marquent les lignes.*

Remarque. *Dans l'implémentation réelle d'un modem, ou en simulation, on peut se passer de l'ordre naturel des symboles, en associant directement les étiquettes aux amplitudes sur (I,Q), pour chacun des deux modes. Déjà illustré sur la Fig. 1, c'est par exemple ce qui est fait dans « Matlab/Simulink graphical*

models » , on parle de la « constellation encodée en Gray » .

V. ETAPES DE RÉALISATION

1) Prenez en main la boite à outils `scikit-dsp-comm` (module 'digitalcom') et `scikit-commpy` (module 'modulation'). Il peuvent être chargés à l'aide de :

```
import sk_dsp_comm.digitalcom as digcom
import commpy.modulation as mod
```

Manipulation	Faites tourner et analysez le Notebook Matlab fourni en exemple, et en particulier, le dernier programme de simulation de probabilité d'erreur en fonction de E_b/N_0. Analysez les résultats.

2) Les données qui seront modulées et transmis sont :
   - les bits source en sortie du codage Huffman
   - les bits encodés en sortie du codage canal cyclique (option : convolutif).

   Préparez les variables pour $M = 4, 16$ et $k\_mod = \log_2(M)$ entier, bits/symbole. En fonction de k_mod, ajuster la taille des données à moduler avec les bits de bourrage afin que sa longueur soit multiple de k_mod pour pouvoir mapper les bits sur les symboles.
3) Modulez les deux séquences de bits avec les fonctions des modules fournis.
4) Dans un premier temps, fixer une valeur de $E_b/N_0 = 8$ dB pour laquelle le schéma complet sera réalisé, et la probabilité d'erreur calculée. Plus loin, cette démarche sera effectuée pour l'intervalle de valeurs $E_b/N_0 = [0..12]$ dB afin de tracer les courbes de performances.
5) Rajouter le bruit gaussien AWGN aux symboles modulés. Pensez à convertir Eb/N_0 en linéaire, valeur à utiliser dans les fonctions, puis transformer en RSB (SNR) , Rapport Signal-sur-Bruit **par symbole !**
6) Tracez sur la même figure les symboles de la constellation de référence et les échantillons reçus bruités, afin de mettre en évidence l'effet du bruit. Essayez la même chose pour les valeurs Eb/N_0 différentes. Concluez.
7) Démodulez (les symboles dûs aux bits non-codés et codés).
8) Enlevez le bourrage et décoder les bits reçu avec le décodeur cycliques, puis reconstruisez la source avec un décodeur de Huffman, pour les données codées. Même opération pour les données non codées par code canal.
9) Calculez le nombre d'erreurs résiduelles pour chaque cas (en comparant aux bits d'information connus, transmis).
10) Englobez ce calcul dans la boucle sur quelques valeurs de E_b/N_0 par exemple, dans l'intervalle $[0...12]$ dB.
11) Calculez et tracez la probabilité d'erreur binaire pour au moins deux modulations QAM (ex 4-QAM et 16-QAM) pour le cas codé et non-codé. Produisez via votre programme le graphique avec les 4 courbes superposées. Vous inclurez ces courbes dans le rapport.
12) Recherchez les expressions théoriques qui donnent $P_e = fct(E_b/N_0)$ pour ces modulations et codage. Rajoutez les courbes correspondantes à votre graphique précédent afin de valider votre simulation.

Remarque. *Typiquement, les systèmes de communication sont simulés à l'aide d'une méthode dite Monte Carlo. Celle-ci consiste à générer les données aléatoires à transmettre et à mesurer. En toute rigueur, on devrait alors parler de la probabilité d'erreur* **moyenne** $\overline{BER} = fct(E_b/N_0)$ *, qui tend vers la vraie valeur de P_b lorsque la taille de l'échantillon de données, concrètement, du nombre de bits transmis,*

tend vers l'infini. Or, la simulation des valeurs de BER très faibles peut nécessiter la génération d'un nombre d'échantillons de plus en plus grand, afin de pouvoir constater une erreur ; les temps de calcul deviennent alors très importants, allant parfois jusqu'à plusieurs heures, voire jours.

A titre d'exemple, afin d'atteindre $BER = 10^{-6}$, à savoir, de n'avoir en moyenne qu'un bit erroné sur un million de bits transmis, et pour que ce résultat soit valide du point de vue statistique (intervalle de confiance de 95%), il faut pouvoir constater une 100-aine d'erreurs et donc de transmettre ≈ 100 millions de bits.

En règle générale, le nombre de N bits qui doivent être observés dans la simulation, serait proportionnel à K/P_b, où P_b est la BER visée, et K est une constante, typiquement dans l'intervalle 10–100, qui définit la fiabilité de l'estimation. Pour reprendre l'exemple plus haut, pour avoir un intervalle de confiance de 95% dans l'estimation de BER, on devrait fixer $K = 100$. Donc, si la probabilité d'erreur à atteindre est 10^{-6}, cela peut donc nécessiter de transmette 100 millions de bits !

C'est la raison pour laquelle, afin que les simulations ne durent que quelques minutes, on se limite à de valeurs de E_b/N_0 assez faibles pour avoir un grand nombre d'erreurs et donc P_b très petites.

Aussi, le nombre de bits représentant le livre encodé en binaire est typiquement plus petit que ce nombre de bit requis pour la simulation. Il convient alors artificiellement, et sans perdre de généralité, de concaténer un certain nombre de fois le livre en binaire pour les produire.

VI. (Option) Filtrage de mise en forme

Dans un système réel de transmission, la forme de l'impulsion pour chaque symbole n'est pas un signal porte rectangulaire. D'une part, un tel signal n'est pas réalisable, et de l'autre, il occuperait une bande infinie, car sa TF est un sinus cardinal.

Or, pour une transmission au rythme de symboles D_s, une forme d'impulsion réalisable physiquement serait plus longue que la durée d'un symbole, $T_{pulse} > T_s$.

Afin d'éviter l'Interférence Entre les Symboles (IES) successifs et donc leur superposition, qui rendrait difficile la détection à la réception, la forme d'impulsion spéciale est créée par filtrage. Typiquement, un filtre dit de « cosinus sur-élevé » est mis en œuvre, mais il n'est pas le seul possible.

Dans notre scénario, on va plutôt prendre un autre,—filtre de Butterworth d'ordre élevé. Celui-ci approxime bien un filtre anti-IES, sa réponse impulsionnelle est très proche de $sin(x)/x$ et de plus, sa réalisation en Matlab est facilitée par les fonctions intégrées, comme `'butter()'`.

L'implémentation d'un tel système nécessite un sur-échantillonnage de facteur $N_{samp} = T_{symb}/T_{ech}$, typiquement de 4 à 16.