Signal – TP4 – Thème4

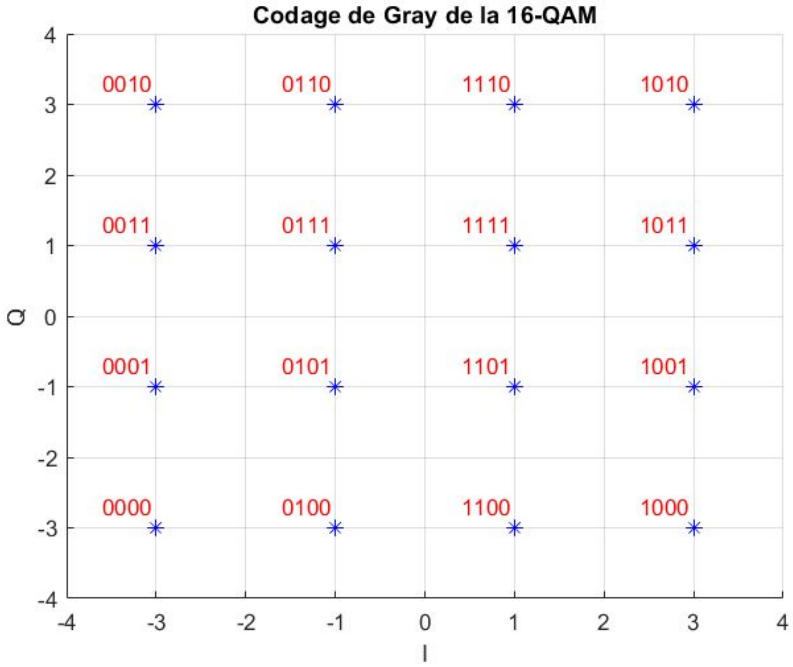
Maéva Bachelard - Margot Laleu

I - Constellation 16-QAM

Notre première objectif est de visualiser la constellation de la 16-QAM sur laquelle apparaît le code de Gray correspondant aux symboles a_n de la 16-QAM tels que $a_n = a_n^I + j a_n^Q$ avec $a_n^I, a_n^Q \in \{\pm 1, \pm 3\}$.

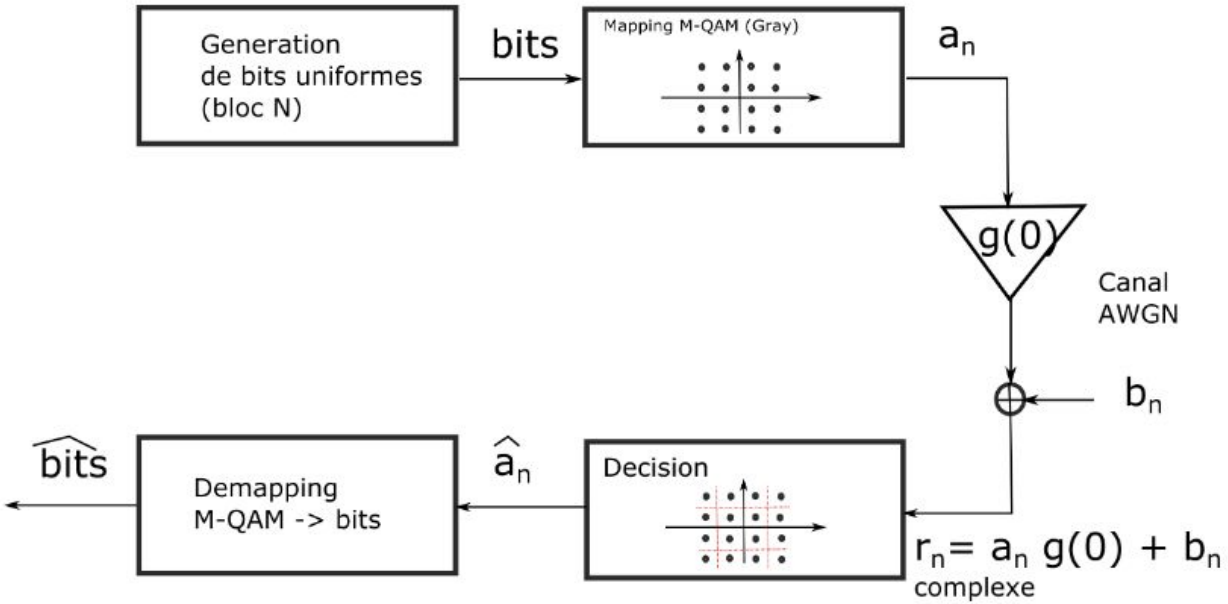
Chaque a_n^I, a_n^Q suivant donc le codage de Gray d'une 4-ASK.

Dans notre représentation ci-dessous, les 2 premiers bits de chaque symbole correspondent au code de Gray de l'axe des abscisses, et les 2 derniers bits de chaque symbole correspondent au code de Gray de l'axe des ordonnées.



II - Transmission complexe équivalente en bande de base de la 16-QAM

L'objectif de ce TP est de simuler la chaîne de communication 16-QAM ci-dessous, dont le bruit est Gaussien.



Pour cela, nous allons coder les fonctions :

- **genBin**, correspondant au bloc de génération aléatoire de notre séquence binaire à transmettre.
- **mappingGray**, correspondant à la conversion de paquets de 4 bits en symboles de la 16-QAM.
- **canalAWGN**, correspondant à l'addition du bruit provenant du canal de transmission.
- **Decision**, correspondant au bloc de décision.
- **demapGray**, permettant de retrouver la séquence binaire transmise.

1) Emetteur

genBin : génération d'une séquence aléatoire binaire

Cette fonction a pour but de générer une séquence aléatoire binaire que l'on transmettra via la chaîne de communication. Cette séquence suit donc une loi uniforme sur [| 0, 1|].

```
function [VectN] = genBin(N)

VectN=randi([0, 1],[1,N]);


end
```

randi([a, b] , [n, m]) : renvoie une matrice de dimension n*m comportant des entiers compris entre a et b selon une loi uniforme.

Nous avons testé notre fonction :
Elle renvoie bien des vecteurs de la taille indiquée en argument et contenant uniquement des zéros et des uns.

```
VectN = genBin(4)
VectN1 = genBin(8)
VectN2 = genBin(12)
```

```
VectN = 1×4
     0     0     0     1

VectN1 = 1×8
     1     0     1     0     0     0     1     1

VectN2 = 1×12
     0     0     0     1     1     1     0     1     1     0     0     0
```

mappingGray

L'objectif de cette fonction est de regrouper les bits à envoyer par paquet de 4. Pour cela on transforme 4 bits successifs de notre séquence à transmettre, SeqBin, en string (via int2str) et on les concatène ensuite grâce à strcat.

```
function[SeqComp] = mappingGray(SeqBin, mGray, mComplex)

N = length(SeqBin);
%N4 = N/4;

SeqDeci = zeros(1, round(N/4));


for i = 1 : 4 : N-3
    Temp = strcat(int2str(SeqBin(i)), int2str(SeqBin(i+1)), int2str(SeqBin(i+2)), int2str(SeqBin(i+3)));

    SeqDeci(round(i/4+1)) = bin2dec(Temp);

end

SeqComp = zeros(1, round(N/4));
for l = 1 : length(SeqDeci)                    % Il aurait été plus simple d'utiliser la fonction find
    for j = 1 : 4
        for k = 1:4
            if SeqDeci(l) == mGray(j,k)

              SeqComp(l)= mComplex(j,k);
            end
        end
     end
end


end
```

Chaque paquet de 4 bits constitue alors un des 16 symboles binaires (sur 4 bis) de la 16-QAM. On convertit donc chacun de ces symboles en la valeur décimale égale à leur symbole binaire, grâce à la fonction bin2dec et on les ajoute au fur et à mesure dans un nouveau vecteur, SeqDeci, de taille N/4, puisque chacune des valeures décimales qu'il contient correspond à 1 symbole de la 16-QAM, codant 4 bits de la séquence d'origine.

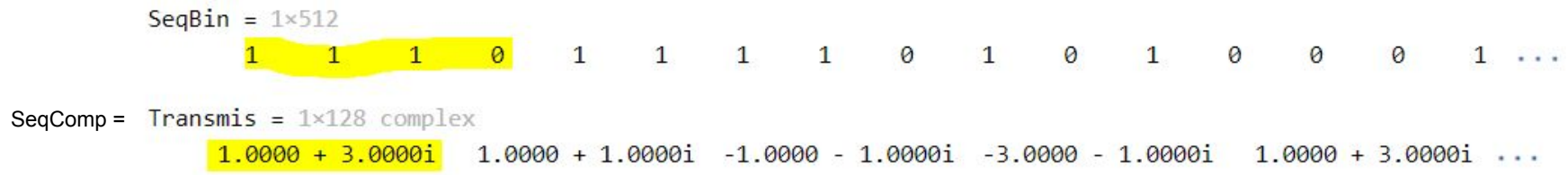
L'étape suivante consiste à identifier l'emplacement de cette valeur décimale dans la constellation 16-QAM, via notre matrice mGray (4x4) précédemment créée et qui contient les valeurs décimales du code de Gray (donc à un emplacement précis dans la matrice).
On identifie cet emplacement en comparant chacun des symboles de SeqDeci à chacun des symboles de mGray. Cela aurait été plus simple avec la fonction find.
Une fois cet emplacement identifié, on en récupère ses coordonnées dans la matrice mGray.
Il ne reste plus qu'à aller chercher la valeur des symboles complexes a_n de la 16-QAM à l'emplacement correspondant dans notre matrice mComplex (4x4) précédemment créée et stocker cette valeur dans un nouveau vecteur SeqComp (Séquence compressée) qui sera renvoyé en sortie de notre fonction.

L'information à transmettre via le canal est ainsi portée par la partie réelle et la partie imaginaire du signal (ou amplitude et phase).

find(Vect==x) : qui renvoie les indices de Vect où se trouve la valeur x.

strcat(x, y, z,...) : concatène horizontalement et dans l'ordre les strings x, y, z, … données en entrée.

Pour tester son bon fonctionnement, on regarde si les mots de 4 bits envoyés correspondent bien au symbole de notre constellation 16-QAM. Et on se rend compte que cela fonctionne :



2) Canal complexe AWGN
canalAWGN

L'objectif de cette fonction est de simuler le canal de transmission et plus particulièrement le bruit qu'il apporte. En effet, le filtre g(0) = 1 n'impacte pas notre signal, nous ne l'avons donc pas codé.
Le bruit apporté est Gaussien, de variance σ_b^2 et porte indépendamment sur les parties réelle et imaginaire.

```
function[SeqRecu] = canalAWGN(SeqComp, sigmab2)

    N = length(SeqComp);

    bruitR = genBruitN(N, sigmab2);
    bruitI = genBruitN(N, sigmab2);

    SeqRecuR = real(SeqComp)+bruitR;
    SeqRecuI = imag(SeqComp)+bruitI;


    SeqRecu = SeqRecuR + 1i.*SeqRecuI  ;


end
```

Pour cela, nous avons choisi de réutiliser la fonction genBruitN que nous avions codée lors d'un précédent TP.

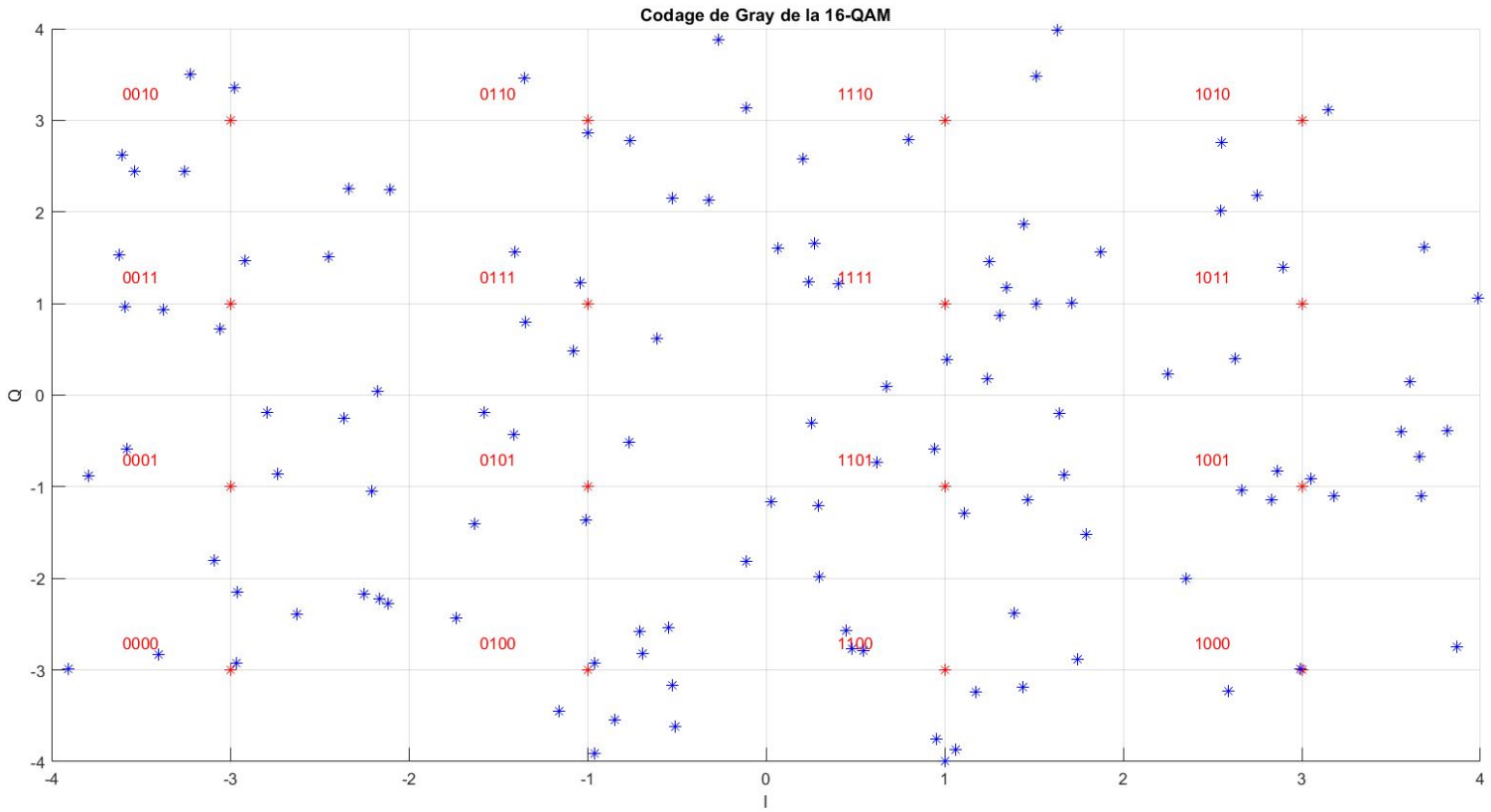
Le test ci-dessous montre bien que nos valeurs subissent bien des variations dues à l'ajout du bruit.

```
Transmis = 1×128 complex
    1.0000 + 3.0000i   1.0000 + 1.0000i  -1.0000 - 1.0000i  -3.0000 - 1.0000i

Recu = 1×128 complex
    1.3815 + 2.7202i   0.9006 + 0.8004i  -0.7590 - 1.4713i  -3.7694 - 1.3243i
```

Signal Reçu r_n superposé à la constellation 16-QAM

On teste nos formules précédentes pour N = 512 et $\sigma_b^2 = ½$, et on visualise nos symboles reçus sur la constellation.



On remarque que le bruit perturbe le signal transmis de manière importante, puisqu'aucun des signaux reçus ne correspond exactement à un des symboles de la 16-QAM.

3) Récepteur
Bloc de décision

A cause du bruit, et dans le cas réel de l'atténuation g(0) ≠1, les symboles reçus ne correspondent plus à des symboles existant de la 16-QAM, comme on a pu le voir sur la slide précédente. Il est donc nécessaire de trancher et de dire de quel symbole de la 16-QAM notre symbole reçu se rapproche le plus.

```
function[Estimation] = Decision(Recu, mComplex)

    N = length(Recu);
    Nmapping = numel(mComplex);

    Estimation = zeros(1, N);

    for i = 1 : N

        min = abs(Recu(i)-mComplex(1));
        indice = 1;

        for j = 2 : Nmapping

            cmp = abs(Recu(i)-mComplex(j));

            if cmp<min
                min = cmp;
                indice = j;
            end
        end
    Estimation(i) = mComplex(indice);
    end
end
```

Pour cela, pour chaque symbole reçu, on cherche la plus petite distance entre le symbole reçu et les symboles de la 16-QAM, grâce à la fonction abs.

length(Vect) : renvoie le nombre d'éléments dans un vecteur.

Attention! **length(Matrix)** renvoie le nombre le nombre de lignes de la matrice.

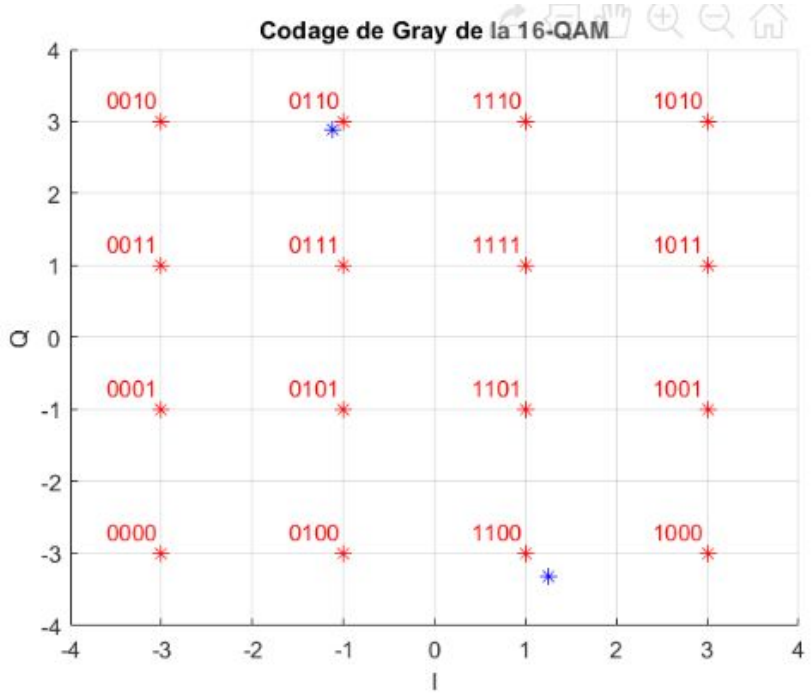
numel(Matrix): renvoie le nombre d'éléments dans une matrice.

abs(a) : si a est réel, renvoie la valeur absolue de a.
si a est complexe, renvoie son module.
Donc **abs(z-y)** avec z et y complexe, renvoie leur distance dans le plan complexe.

Pour tester cette fonction, on regarde quel est le signal reçu (Recu2 comportant seulement 2 points pour plus de lisibilité) et l'emplacement des points correspondant, en bleu, sur la constellation. Et on regarde si le résultat du bloc de décision est celui auquel on s'attendait. C'est bien le cas ici, puisque le bloc de décision tranche bien pour le symbole le plus proche des points bleus. Le résultat est dans SeqDecid.

```
Recu2 = 1×2 complex
     1.2571 - 3.3088i  -1.1167 + 2.8968i

SeqDecid = 1×128 complex
     1.0000 - 3.0000i  -1.0000 + 3.0000i
```



demapGray

Cette fonction consiste en l'exercice inverse de la fonction mappingGray. On a en entrée notre vecteur SeqDecid contenant les symboles complexes transmis par la chaîne de communication. Il faut donc les reconvertir en leur symbole binaire correspondant mais exprimé en décimale pour l'instant.

```
function[SeqBits] = demapGray(SeqDecid, mGray, mComplex)

    N = length(SeqDecid);
    SeqTemp = zeros(1, N);
    SeqBits = [];

    for k = 1 : N
        [i, j] = find(mComplex==SeqDecid(k));
        SeqTemp(k)= mGray(i, j);
    end

    for  h = 1 : length(SeqTemp)
        Seq = decimalToBinaryVector(SeqTemp(h), 4);
        SeqBits = cat(2,SeqBits,Seq);
    end

end
```

Pour cela, on recherche dans notre matrice mComplex le symbole de la constellation identique et on en récupère ses coordonnées dans mComplex grâce à la fonction find cette fois. Ensuite, on stocke dans une séquence temporaire de taille N, la valeur décimale correspondante se trouvant dans mGray aux mêmes coordonnées.
Il s'agit ensuite de convertir chaque valeur décimale en vecteur de 4 bits correspondant via la fonction decimalToBinaryVector. On concatène alors, au fur et à mesure, ces matrices de 4 bits obtenues avec la fonction cat, dans une nouvelle séquence SeqBits.

decimalToBinaryVector(x, N): convertit la valeur décimal x en le vecteur binaire correspondant de taille N. c'est à dire que si x est codé sur moins de N bits, des 0 seront ajoutés à gauche pour obtenir un vecteur de taille N.

Le test est concluant puisque nous obtenons bien les mots de 4 bits correspondant au symbole complexe associé :

```
SeqDecid = 1×128 complex
    1.0000 - 3.0000i  -1.0000 + 3.0000i ...

SeqBits = 1×431
     1     1     0     0     1     1     0     1 ...
```

4) Chaîne de transmission globale

On crée à présent une fonction qui reprend toutes les précédentes et ne renvoie que les vecteurs les plus importants, à savoir la séquence binaire envoyée, sa traduction en symbole de la 16-QAM et leurs équivalents à la réception, en sortie du canal.

Le test ci-dessous semble concluant.

```
function[bitsE, bitsS, anE, anS] = chaine16QAM(N, sigmab2)

    mComplex = const16QAM();
    mGray = Gray16QAM();

    bitsE = genBin(N);

    anE = mappingGray(bitsE, mGray, mComplex);

    anEBruit = canalAWGN(anE, sigmab2);

    anS = Decision(anEBruit, mComplex);

    bitsS = demapGray(anS, mGray, mComplex);

end
```

```
[bitsE, bitsS, anE, anS] = chaine16QAM(N, sigmab2)
```

```
bitsE = 1×512
     1     1     0     1     0     0     1     0 ...

bitsS = 1×512
     1     1     0     1     0     1     1     0 ...

anE = 1×128 complex
   1.0000 - 1.0000i  -3.0000 + 3.0000i ...

anS = 1×128 complex
   1.0000 - 1.0000i  -1.0000 + 3.0000i ...
```

Test de la fiabilité de notre chaîne

Pour savoir si notre chaîne est fiable, c'est-à-dire si elle transmet fidèlement le message binaire, nous allons calculer son taux d'erreur binaire.

Pour cela, on compte le nombre de bits faux en sortie par soustraction en valeur absolue des vecteurs en entrée et en sortie.

```
NBerreurBin = sum(abs(bitsE-bitsS))
TauxErreurBin = NBerreurBin/N
```

Et on obtient un taux d'erreur binaire d'environ 6.5 %
(Entre 6 et 7% sur différents tests).
Ce qui est paraît tout à fait acceptable.

```
NBerreurBin = 33
TauxErreurBin = 0.0645
```

Ainsi, à nombre de bits à transmettre égaux (N = 512), le canal 16-QAM semble un peu plus fiable que le canal BPSK (qui avait un taux d'erreur entre 7 et 8%).
Mais il faudrait tracer les courbes Taux d'erreur binaire en fonction du RSB normalisé en dB de ces 2 types de modulation, afin de vérifier.