

ABDILLAH Bouh  
Matricule : 1940646  
ELE 6701A : Projet

Partie 1 :

a) On envoie  $\vec{s} = \begin{bmatrix} s(0) \\ s(1) \\ \vdots \\ s(3) \end{bmatrix}$

avec  $s(i) = \pm 1$

On observe  $\vec{y} = \begin{bmatrix} y(0) \\ y(1) \\ \vdots \\ y(3) \end{bmatrix}$

$$\stackrel{\pm 1}{s(i)} \xrightarrow{n(i)} y(i)$$

$n(i) \sim N(0, \sigma_n^2)$

Puis nous utilisons un détecteur ML :

$$y(i) \rightarrow \boxed{\text{Détecteur ML}} \rightarrow \hat{s}(i)$$

Nous transmettons 10 symboles  $\pm 1$ ,

on a donc  $M = 2^{10}$  possibilités

$\Rightarrow M$  hypothèses  $H_i$  avec  $i = 1, \dots, M$

$k=10$  observations  $\vec{y} = \begin{bmatrix} y(0) \\ \vdots \\ y(9) \end{bmatrix}$

$$\vec{y} = \vec{s}_i + \vec{n}, \quad i = 0, \dots, M-1$$

$s_i$  est le signal associé à l'hypothèse  $H_i$

On a  $R_i$  régions de décisions ( $i = 0, \dots, M-1$ )

La caractéristique de décision devient de choisir

$$H_j \text{ si } : \|\vec{y} - \vec{s}_j\|^2 < \|y - s_l\|^2 \quad \forall l \neq j \quad (1)$$

Pour chaque paire d'hypothèse  $H_j$  et  $H_l$  avec  $j \neq l$  dans les 2<sup>10</sup> hypothèses possibles.

Trouvons la relation équivalente de (1).

$$\vec{y} = \vec{s}_i + \vec{n}$$

Pour chaque paire  $\{H_j; H_l\}$  on a :

$$s_i = x_i (\vec{s}_j - \vec{s}_l) + \frac{1}{2} (\vec{s}_j + \vec{s}_l)$$

$$x_i = \begin{cases} 1/2 & \text{si } H_j \\ -1/2 & \text{si } H_l \end{cases}$$

$$\Rightarrow \vec{y} = \vec{y} - \frac{1}{2} (\vec{s}_j + \vec{s}_l) = x_i (\vec{s}_j - \vec{s}_l) + \vec{n}$$

Soit  $\vec{v}$  le vecteur directeur tel que :

$$\vec{v} = \frac{\vec{s}_j - \vec{s}_l}{\|\vec{s}_j - \vec{s}_l\|}$$

Nous avons  $N = 10$  observations, on pose

$$\Omega = [\vec{v} \ \vec{\sigma}_1 \ \vec{\sigma}_2 \dots \vec{\sigma}_9]$$

Tel que  $\vec{\sigma}_i^* \vec{v} = 0$  et  $\vec{\sigma}_i^* \vec{\sigma}_j = \delta_{ij} = \begin{cases} 1 & \text{si } i=j \\ 0 & \text{sinon} \end{cases}$

$\Omega \Omega^* = I \Rightarrow \Omega$  est orthonormée

$$\Omega \vec{y} = \begin{bmatrix} \vec{v}^* \\ \vec{\sigma}_1^* \\ \vec{\sigma}_2^* \\ \vdots \\ \vec{\sigma}_9^* \end{bmatrix} (x_i (\vec{s}_j - \vec{s}_l) + \vec{n})$$

$$= \begin{bmatrix} x_i \|\vec{s}_j - \vec{s}_l\|^2 \\ \vdots \\ 0 \end{bmatrix} + \vec{\theta}^* \vec{n}$$

[ex]

$$\vec{\theta}^* \vec{n} \sim \mathcal{CN}(\theta, N_0 I)$$

$$\text{avec } N_0 = \sigma_n^2$$

On pose ainsi

$$\tilde{y} = \operatorname{Re} \left\{ \frac{(\vec{s}_j - \vec{s}_l)^*}{\|\vec{s}_j - \vec{s}_l\|} \left( \vec{y} - \frac{1}{2} (\vec{s}_j + \vec{s}_l) \right) \right\}$$

$$= x_i \|\vec{s}_j - \vec{s}_l\|^2 + \vec{n}$$

statistique suffisante

$$\text{avec } \vec{n} \sim \mathcal{N}(0, \frac{N_0}{2})$$

On décide alors

$$\tilde{y} \stackrel{H_j}{\geq} 0$$

$\stackrel{H_l}{<} 0$

$$\Rightarrow \tilde{y} = \operatorname{Re} \left\{ \frac{1}{\|\vec{s}_j - \vec{s}_l\|} \left( \vec{s}_j^* \vec{y} - \vec{s}_l^* \vec{y} - \frac{1}{2} \left( \vec{s}_j^* \vec{s}_j + \vec{s}_j^* \vec{s}_l - \cancel{\vec{s}_l^* \vec{s}_j} + \cancel{\vec{s}_l^* \vec{s}_l} \right) \right) \right\} \stackrel{H_j}{\geq} 0$$

$$\Rightarrow \tilde{y} = \operatorname{Re} \left\{ \quad \right\}$$

$$\text{Posons } \varepsilon_i = \vec{u}_i^* \vec{u}_i = \|\vec{u}_i\|^2$$

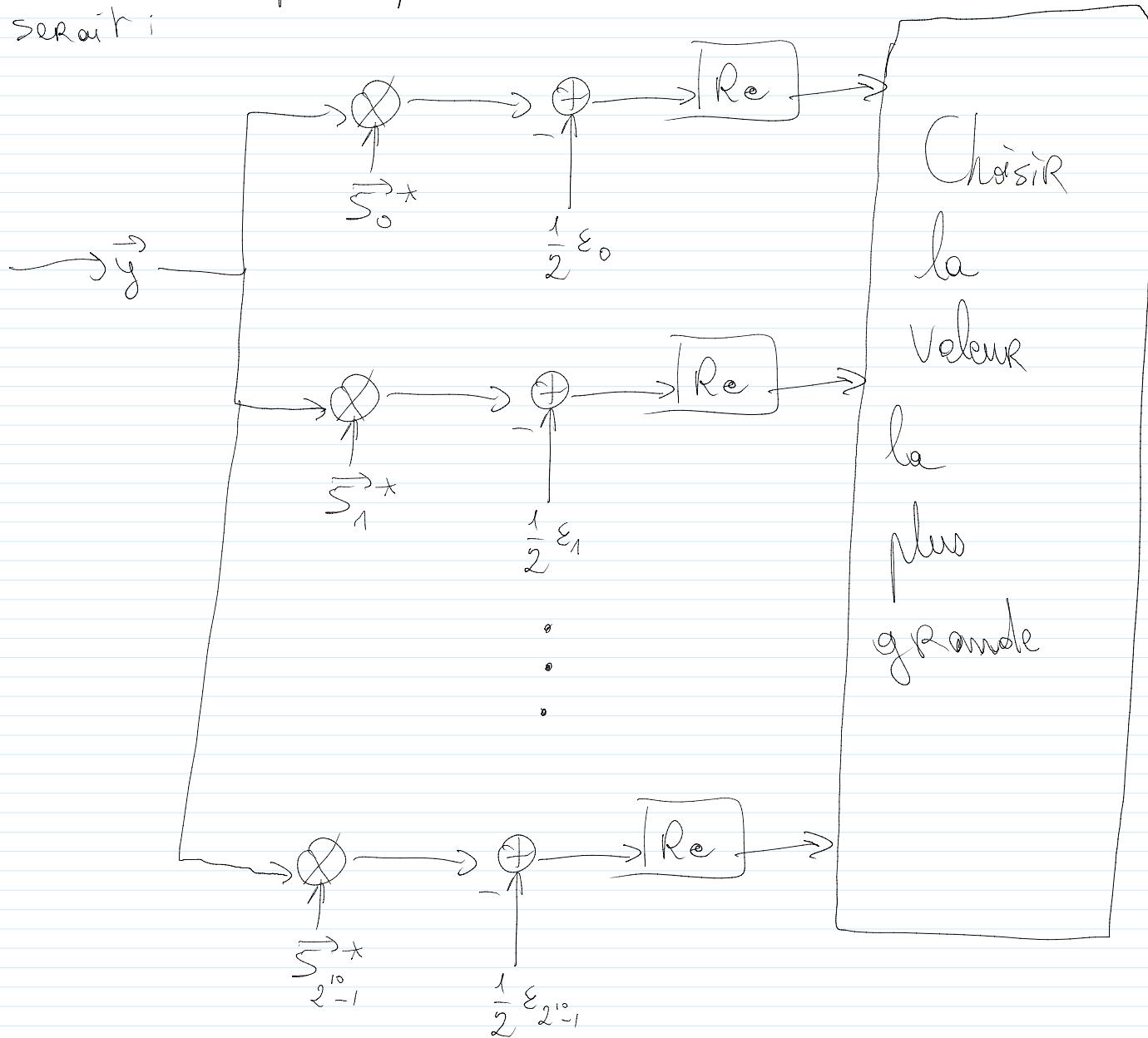
$\sim H_i$

$$\text{Posons } \varepsilon_i = \vec{u}_i^* \vec{u}_i = \|\vec{u}_i\|^2$$

$$\Rightarrow \tilde{y} = \operatorname{Re} \left\{ \frac{1}{\|\vec{s}_j - \vec{s}_l\|} \left( \vec{s}_j^* \vec{y} - \vec{s}_l^* \vec{y} - \frac{1}{2} \varepsilon_j - \frac{1}{2} \varepsilon_l \right) \right\}_{\substack{H_f \\ H_L}} \geq 0$$

calculer toute les paires d'hypothèses  
serait trop coûteux.

Une manière plus simple de procéder  
serait :



$$\text{avec } \{\vec{s}_0, \vec{s}_1, \dots, \vec{s}_{2^{10}-1}\}$$

étant l'ensemble des vecteurs hypothèses possibles.

b) Borne union  $B_u$

$$M = 2^{10}$$

$$B_u = \sum_{i=0}^{2^{10}-1} \pi_i \sum_{\substack{j=0 \\ j \neq i}}^{2^{10}-1} Q\left(\frac{\|\vec{s}_j - \vec{s}_i\|}{2\sqrt{N_0}}\right)$$

Borne distance minimale  $B_{d_{\min}}$

$$B_{d_{\min}} = (2^{10} - 1) \max_{\substack{i,j=0, \dots, 2^{10}-1 \\ i \neq j}} Q\left(\frac{\|\vec{s}_j - \vec{s}_i\|}{2\sqrt{N_0}}\right)$$

c)

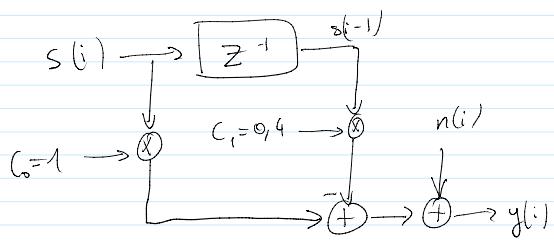
Partie II:

a) on a pour  $D = 0, 1, 2$ :

$$\hat{s}(i-D) = \alpha_0 s(i) + \alpha_1 s(i-1) + \alpha_2 s(i-2)$$

trouver  $\{\alpha_0, \alpha_1, \alpha_2\}$  pour  $D = 0$   
 $D = 1$   
 $D = 2$

Nous avons le casel suivant:



$$\text{tel que } y(i) = s(i) - 0.4 s(i-1) + n(i)$$

$$\text{on a: } \vec{C} = \begin{bmatrix} C_0 \\ C_1 \end{bmatrix} = \begin{bmatrix} 1 \\ -0.4 \end{bmatrix}$$

$$\vec{y} = \begin{bmatrix} y(i) \\ y(i-1) \\ y(i-2) \end{bmatrix}$$

Données:

$$R_s = \tilde{\sigma}_x^2 \mathbb{I} \quad R_n = N \mathbb{I} = \tilde{\sigma}_n^2 \mathbb{I} \quad \bar{n} = 0$$

$$\bar{s} = 0 \quad R_{ns} = 0$$

$$\begin{bmatrix} y(i) \\ y(i-1) \\ y(i-2) \end{bmatrix} = \begin{bmatrix} C_0 & C_1 & 0 & 0 \\ 0 & C_0 & C_1 & 0 \\ 0 & 0 & C_0 & C_1 \end{bmatrix} \begin{bmatrix} s(i) \\ s(i-1) \\ s(i-2) \\ s(i-3) \end{bmatrix} + \begin{bmatrix} n(i) \\ n(i-1) \\ n(i-2) \end{bmatrix}$$

ainsi:

$$\vec{y} = H \vec{s} + \vec{n}$$

avec  $H = \begin{bmatrix} 1 & -0.4 & 0 & 0 \\ 0 & 1 & -0.4 & 0 \\ 0 & 0 & 1 & -0.4 \end{bmatrix}$

$$\hat{s}(i-\Delta) = \omega \vec{y}$$

$$\text{ou } \vec{w} \vec{y} = R_{s(i-\Delta)} \vec{y}$$

$$R_y = E(\vec{y} \vec{y}^*) = E[(H\vec{s} + \vec{n})(H\vec{s} + \vec{n})^*]$$

$$\Rightarrow R_y = \sigma_s^2 H H^* + \sigma_n^2 I$$

$$\begin{aligned} R_{S(i-\Delta)} \vec{y} &= E[s(i-\Delta) \vec{y}^*] \\ &= E[s(i-\Delta)(H\vec{s} + \vec{n})^*] \\ &= E[s(i-\Delta) \vec{s}^*] H^* \\ &= [E[s(i-\Delta) s(i)^*] \dots E[s(i-\Delta) s(i-3)^*]] H^* \end{aligned}$$

Les entrées sont indépendantes

$$\Rightarrow = \left[ 0 \dots \sigma_s^{-2} \dots 0 \right] H^*$$

$$\text{Soit } \vec{e}_\Delta = \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix} \xrightarrow{\Delta}$$

$$\Rightarrow R_{S(i-\Delta)} \vec{y} = \sigma_s^{-2} \vec{e}_\Delta^* H^*$$

$$\Rightarrow S(i-\Delta) = R_{S(i-\Delta)} \vec{y} R_y^{-1} \vec{y}$$

$$= \sigma_s^{-2} \vec{e}_\Delta^* H^* \underbrace{(\sigma_s^{-2} H H^* + \sigma_n^2 I)^{-1}}_W \vec{y}$$

$$\vec{w} = [w_0 \ w_1 \ w_2] = [\alpha_0 \ \alpha_1 \ \alpha_2]$$

$$\text{MMSE} = R_s - K_o R_y K_o^*$$
$$= \tilde{\sigma}_s^2 - \vec{w}^* R_y \vec{w}^*$$
$$= \tilde{\sigma}_s^2 \left( 1 - \tilde{\sigma}_s^2 \vec{e}_o^* H^* R_y^{-1} H \vec{e}_o \right)$$

---

## Table of Contents

Projet Partie I A23 ELE6701A .....	1
Generation des 1024 vecteurs hypotheses .....	1
Generation du bruit n .....	2
Generation de paquets s_i a envoyer .....	2
Vecteur y recu .....	2
Borne Union et Borne Distance Minimale (theorique) .....	2
Commentaire I-b .....	3
Detection ML .....	3
Commentaire question I-e .....	4

# Projet Partie I A23 ELE6701A

Bouh Abdillahi

Matricule : 1940646

Github : <https://github.com/konoDioDA253/ELE6701A>

```
clear all;
clc;
H=[1 0 0 0 0 0 0 0 0 0;
    -0.4 1 0 0 0 0 0 0 0 0;
    0 -0.4 1 0 0 0 0 0 0 0;
    0 0 -0.4 1 0 0 0 0 0 0;
    0 0 0 -0.4 1 0 0 0 0 0;
    0 0 0 0 -0.4 1 0 0 0 0;
    0 0 0 0 0 -0.4 1 0 0 0;
    0 0 0 0 0 0 -0.4 1 0 0;
    0 0 0 0 0 0 0 -0.4 1 0;
    0 0 0 0 0 0 0 0 -0.4 1];

```

## Generation des 1024 vecteurs hypotheses

```
vecteurs_cell = cell(1, 1024);

for i = 1:1024
    % Convertir l'indice en binaire sur 10 bits
    binaire = dec2bin(i-1, 10) - '0'; % Convertir en vecteur binaire

    % Remplacer les '0' par '-1' dans la representation binaire
    binaire(binaire==0) = -1;

    % Stocker le vecteur dans la cellule
    vecteurs_cell{i} = binaire';
end

% Convertir la cellule en une matrice de taille 10x1024
matrice_vecteurs_hypothese_s = cell2mat(vecteurs_cell);
```

---

# Generation du bruit n

Variance desiree

```
variance_desiree = (0.15);

% Nombre de vecteurs a generer
nombre_paquets = 10000; %% VALEUR A MODIFIER
nombre_vecteurs = nombre_paquets;
taille_vecteur = 10;

% Generation des vecteurs de bruit gaussien blanc avec moyenne nulle
vecteurs_bruit_gaussien = sqrt(variance_desiree) *
randn(taille_vecteur, nombre_vecteurs);
```

# Generation de paquets s\_i a envoyer

Nombre de vecteurs aleatoires a generer

```
nombre_vecteurs = nombre_paquets;
taille_vecteur = 10;

% Generation des vecteurs aleatoires
vecteurs_aleatoires_envoyes = randi([1, 2], taille_vecteur,
nombre_vecteurs);
vecteurs_aleatoires_envoyes(vecteurs_aleatoires_envoyes == 2) = -1;
```

# Vecteur y recu

Matrice des vecteurs y

```
y=vecteurs_aleatoires_envoyes + vecteurs_bruit_gaussien;
```

# Borne Union et Borne Distance Minimale (theorique)

```
M = 2^10;
pi_i = 1/M;
Borne_union = 0;
N0 = 0.15;
Qmax = 0;
for i=1:M
    for j=1:M
        if j~=i
            Q = qfunc((1/
(2*sqrt(N0)))*norm(matrice_vecteurs_hypothese_s(:,i)-
matrice_vecteurs_hypothese_s(:,j)));
            Borne_union = Borne_union + pi_i*Q;
            if Q > Qmax
                Qmax = Q;
```

---

```

        end
    end
end
Borne_distance_minimale = 10*Qmax;
disp("La Borne Union vaut : ")
disp(Borne_union)

disp("La Borne distance minimale vaut : ")
disp(Borne_distance_minimale)

La Borne Union vaut :
0.0555

La Borne distance minimale vaut :
0.0491

```

## Commentaire I-b

Nous remarquons que la borne union est inférieure à la borne distance minimale. La probabilité d'erreur se trouve ainsi mieux estimée avec la borne distance minimale.

## Detection ML

```

comparison_matrix=-1000000*ones(1,1024);
indice_guess = -1*ones(1,nombre_paquets);
vecteurs_s_guess = -3434314324*ones(10,nombre_paquets);
error_count_vector = 0;
error_count_symbol = 0;
for i=1:1:nombre_paquets

    for j=1:1:1024
        comparison_matrix(1,j) =
real(transpose(matrice_vecteurs_hypothese_s(:,j))*y(:,i) -
0.5*transpose(matrice_vecteurs_hypothese_s(:,j))*matrice_vecteurs_hypothese_s(:,j)
    end
    [max_value, indice_guess(1,i)] = max(comparison_matrix);
    vecteurs_s_guess(:,i) =
matrice_vecteurs_hypothese_s(:,indice_guess(1,i));
    comp = isequal(vecteurs_aleatoires_envoyes(:,i),
vecteurs_s_guess(:,i));
    difference_per_symbol = nnz(vecteurs_s_guess(:,i) ~= vecteurs_aleatoires_envoyes(:,i));
    error_count_symbol = error_count_symbol + difference_per_symbol;
    if comp
        test = 0;
    else
        error_count_vector = error_count_vector+1;
    disp("error!");
end

```

---

```
end
taux_erreur = error_count_vector/nombre_paquets
taux_erreur_par_symbole = error_count_symbol/(nombre_paquets*10)

taux_erreur =
0.0497

taux_erreur_par_symbole =
0.0051
```

## Commentaire question I-e

Nous observons que ce code prend beaucoup de temps à exécuter du fait de sa grande complexité. En effet, traiter toutes les hypothèses peut s'avérer très fastidieux, surtout pour un nombre important de paquets ou d'observations.

*Published with MATLAB® R2019b*

---

## Table of Contents

Projet Partie II A23 ELE6701A .....	1
G?n?ration des 1024 vecteurs hypoth?ses .....	1
G?n?ration du bruit n .....	2
G?n?ration de paquets s_i ? envoyer .....	2
Vecteur y re?u .....	2
Parametres pour calcul egalisateur MMSE .....	2
Egalisateur MMSE .....	2
Decision MMSE .....	3
Commentaire bii .....	6
Commentaire biii .....	6

# Projet Partie II A23 ELE6701A

Bouh Abdillahi

Matricule : 1940646

Github : <https://github.com/konoDioDA253/ELE6701A>

```
clear all;
clc;
H=[1 0 0 0 0 0 0 0 0 0;
    -0.4 1 0 0 0 0 0 0 0 0;
    0 -0.4 1 0 0 0 0 0 0 0;
    0 0 -0.4 1 0 0 0 0 0 0;
    0 0 0 -0.4 1 0 0 0 0 0;
    0 0 0 0 -0.4 1 0 0 0 0;
    0 0 0 0 0 -0.4 1 0 0 0;
    0 0 0 0 0 0 -0.4 1 0 0;
    0 0 0 0 0 0 0 -0.4 1 0;
    0 0 0 0 0 0 0 0 -0.4 1];
```

## G?n?ration des 1024 vecteurs hypoth?ses

```
vecteurs_cell = cell(1, 1024);

for i = 1:1024
    % Convertir l'indice en binaire sur 10 bits
    binaire = dec2bin(i-1, 10) - '0'; % Convertir en vecteur binaire

    % Remplacer les '0' par '-1' dans la repr?sentation binaire
    binaire(binaire==0) = -1;

    % Stocker le vecteur dans la cellule
    vecteurs_cell{i} = binaire';
end

% Convertir la cellule en une matrice de taille 10x1024
matrice_vecteurs_hypothese_s = cell2mat(vecteurs_cell);
```

---

## Génération du bruit n

Variance désirée

```
variance_desiree = (0.15);

% Nombre de vecteurs ? générer
nombre_paquets = 10000; %% VALEUR A MODIFIER
nombre_vecteurs = nombre_paquets;
taille_vecteur = 10;

% Génération des 1024 vecteurs de bruit gaussien blanc avec moyenne
% nulle
vecteurs_bruit_gaussien = sqrt(variance_desiree) *
randn(taille_vecteur, nombre_vecteurs);
```

## Génération de paquets s\_i à envoyer

Nombre de vecteurs aléatoires ? générer

```
nombre_vecteurs = nombre_paquets;
taille_vecteur = 10;

% Génération des vecteurs aléatoires
vecteurs_aleatoires_envoyes = randi([1, 2], taille_vecteur,
nombre_vecteurs);
vecteurs_aleatoires_envoyes(vecteurs_aleatoires_envoyes == 2) = -1;
```

## Vecteur y résultat

Matrice des vecteurs y

```
y=vecteurs_aleatoires_envoyes + vecteurs_bruit_gaussien;
```

## Paramètres pour calcul égalisateur MMSE

```
Rs = eye(3);
Rn = variance_desiree*Rs;
Delta=[0 1 2];

H=[1 -0.4 0 0;
 0 1 -0.4 0;
 0 0 1 -0.4];
eDeltal = [1;0;0;0];
eDelta2 = [0;1;0;0];
eDelta3 = [0;0;1;0];
Ry = Rs*H*(H') + Rn;
```

## Égalisateur MMSE

```
w_Deltal = (eDeltal')*(H')*Ry^-1 % coefficients pour Delta = 0
```

---

```

w_Delta2 = (eDelta2')*(H')*Ry^-1 % coefficients pour Delta = 1
w_Delta3 = (eDelta3')*(H')*Ry^-1 % coefficients pour Delta = 2

MMSE_Delta1_theorique = 1 - w_Delta1*Ry*(w_Delta1')
MMSE_Delta2_theorique = 1 - w_Delta2*Ry*(w_Delta2')
MMSE_Delta3_theorique = 1 - w_Delta3*Ry*(w_Delta3')

w_Delta1 =
0.8508      0.2865      0.0875

w_Delta2 =
-0.0538      0.8237      0.2515

w_Delta3 =
-0.0271      -0.0888      0.7362

MMSE_Delta1_theorique =
0.1492

MMSE_Delta2_theorique =
0.1547

MMSE_Delta3_theorique =
0.2282

```

## Decision MMSE

```

s_chapeau_iDelta1 = zeros(10,nombre_paquets);

s_chapeau_iDelta2 = zeros(10,nombre_paquets);
% s_chapeau_iDelta3 = zeros(10,nombre_paquets);

% vecteur_guess_s_Delta1 = zeros(10,nombre_paquets);
vecteur_guess_s_Delta2 = zeros(10,nombre_paquets);
% vecteur_guess_s_Delta3 = zeros(10,nombre_paquets);
error_count_symbol_bii = 0;
error_count_vector_bii = 0;
error_count_symbol_biii = 0;
error_count_vector_biii = 0;
for i=1:1:nombre_paquets
    s_chapeau_iDelta1(:,i) = w1*[y(1,i); y(); y()];

```

---

```

    s_chapeau_iDelta2(:,i) = [w_Delta2*[y(2,i); y(1,i); 0];
                                w_Delta2*[y(3,i); y(2,i);
                                y(1,i)];
                                w_Delta2*[y(4,i); y(3,i);
                                y(2,i)];
                                w_Delta2*[y(5,i); y(4,i);
                                y(3,i)];
                                w_Delta2*[y(6,i); y(5,i);
                                y(4,i)];
                                w_Delta2*[y(7,i); y(6,i);
                                y(5,i)];
                                w_Delta2*[y(8,i); y(7,i);
                                y(6,i)];
                                w_Delta2*[y(9,i); y(8,i)];
                                w_Delta2*[y(10,i);
                                y(9,i)];
                                ];

```

---

```

%     s_chapeau_iDelta3(:,i) = w3*y(:,i);

    for j=1:1:10
        if s_chapeau_iDelta1(j,i) < 0
            vecteur_guess_s_Delta1(j,i) = -1;
        else
            vecteur_guess_s_Delta1(j,i) = 1;
        end
%
        if s_chapeau_iDelta2(j,i) < 0
            vecteur_guess_s_Delta2(j,i) = -1;
        else
            vecteur_guess_s_Delta2(j,i) = 1;
        end

        if s_chapeau_iDelta3(j,i) < 0
            vecteur_guess_s_Delta3(j,i) = -1;
        else
            vecteur_guess_s_Delta3(j,i) = 1;
        end
    end

    % question bii
    comp = isequal(vecteurs_aleatoires_envoyes(:,i),
vecteur_guess_s_Delta2(:,i));
    difference_per_symbol_bii = nnz(vecteur_guess_s_Delta2(:,i) ~=
vecteurs_aleatoires_envoyes(:,i));
    error_count_symbol_bii = error_count_symbol_bii +
difference_per_symbol_bii;
    if comp
        test = 0;
    else

```

---

```

        error_count_vector_bii = error_count_vector_bii+1;
%
% disp("error!");
end

% question biii
% On considere que y passe par le module de decision avant de le
% comparer aux vecteurs envoyes
y_post_decision_vecteur = y(:,i);
y_post_decision_vecteur(y_post_decision_vecteur > 0) = 1;    % ?l?ments positifs deviennent 1
y_post_decision_vecteur(y_post_decision_vecteur < 0) = -1;    % ?l?ments n?gatifs deviennent -1
difference_per_symbol_biii = nnz(y_post_decision_vecteur ~= vecteurs_aleatoires_envoyes(:,i));
error_count_symbol_biii = error_count_symbol_biii +
difference_per_symbol_biii;
end

taux_erreur_par_paquet_de_10_symboles_bii = error_count_vector_bii/nombre_paquets
taux_erreur_par_symbole_bii = error_count_symbol_bii/(nombre_paquets*10)

diff_carree = (s_chapeau_iDelta2 - vecteurs_aleatoires_envoyes).^2;
mmse_question_bi = mean(diff_carree(:))

% Pour le mmse biii on considere que y passe directement par le module
de
% d?cision (sans egalisateur)
% Changement des valeurs en fonction du signe
y_post_decision = y;
y_post_decision(y_post_decision > 0) = 1;    % ?l?ments positifs deviennent 1
y_post_decision(y_post_decision < 0) = -1;    % ?l?ments n?gatifs deviennent -1
diff_carree = (y_post_decision - vecteurs_aleatoires_envoyes).^2;
mmse_question_biii = mean(diff_carree(:))

taux_erreur_par_symbole_biii = error_count_symbol_biii/(nombre_paquets*10)

taux_erreur_par_paquet_de_10_symboles_bii =
0.2012

taux_erreur_par_symbole_bii =
0.0218

mmse_question_bi =

```

---

---

0.2015

*mmse\_question\_biii =*

0.0213

*taux\_erreur\_par\_symbole\_biii =*

0.0053

## **Commentaire bii**

On remarque que l'erreur est de l'ordre de 2% pour chaque symbole. Cela repr?sent? une augmentation significative par rapport ? la question I-d o? nous ?tions de l'ordre de 0.4%

## **Commentaire biii**

On remarque que le mmse en II-biii est plus faible qu'en II-bi. Par ailleurs on voit que le taux d'erreur par symbole est de 0.15% pour II-bii et est environ 0.42% en II-biii. L'egalisateur fonctionne donc bel et bien car il donne un taux d'erreur et un mmse plus bas.

*Published with MATLAB® R2019b*

---

## Table of Contents

Projet Partie III A23 ELE6701A .....	1
Generation du bruit n .....	1
Generation de paquets s_i a envoyer .....	2
Vecteur y recu .....	2
Egalisateur LMS (entraînement avec 50 symboles) .....	2
Export de la figure (automatique, eh oui!) .....	5
Egalisateur LMS (entraînement avec 500 symboles) .....	5
Commentaires question b) .....	7

# Projet Partie III A23 ELE6701A

Bouh Abdillahi

Matricule : 1940646

Github : <https://github.com/konoDioDA253/ELE6701A>

```
clear all;
clc;
% Choix du systeme d'exploitation
is_unix = 0; % 0 si votre systeme est windows; 1 sinon (Mac,Ubuntu)
if is_unix
    % Unix
    path = "fig/";
else
    % Windows
    path = "fig\";
end
% Creation du repertoire des figures
mkdir('fig');
% Creation des noms des figures
nom_image = ["Entrainement_IIIa"];
Warning: Directory already exists.
```

## Generation du bruit n

Variance desiree

```
variance_desiree = (0.15);

% Nombre de vecteurs a generer
nombre_paquets = 10000; %% VALEUR A MODIFIER
nombre_vecteurs = nombre_paquets;
taille_vecteur = 10;

% Generation des vecteurs de bruit gaussien blanc avec moyenne nulle
vecteurs_bruit_gaussien = sqrt(variance_desiree) *
randn(taille_vecteur, nombre_vecteurs);
```

---

# Generation de paquets s\_i a envoyer

Nombre de vecteurs aleatoires a generer

```
nombre_vecteurs = nombre_paquets;
taille_vecteur = 10;

% Generation des vecteurs aleatoires
vecteurs_aleatoires_envoyes = randi([1, 2], taille_vecteur,
    nombre_vecteurs);
vecteurs_aleatoires_envoyes(vecteurs_aleatoires_envoyes == 2) = -1;
```

## Vecteur y recu

Matrice des vecteurs y

```
y=vecteurs_aleatoires_envoyes + vecteurs_bruit_gaussien;
```

## Egalisateur LMS (entraînement avec 50 symboles)

```
dataset_size = 50; %% INPUT, A MODIFIER
w=zeros(3,1);
y_list = y(:);
vecteurs_aleatoires_envoyes_list = vecteurs_aleatoires_envoyes(:);
erreur_iterative_list = 233423411*ones(dataset_size,3);
compteur_mu=1;
% Initialisation du vecteur pour stocker les moyennes
moyennes_erreur_iterative_list = zeros(dataset_size, 3);
s_chapeau = 34342342*ones(dataset_size,3);
s_guess = 34342342*ones(dataset_size,3);
s_envoye = vecteurs_aleatoires_envoyes_list(1:dataset_size);
for mu=[0.01 0.05 0.1]
    for i=1:1:dataset_size
        if i == 1
            s_chapeau(i,compteur_mu) = [y_list(i), 0, 0]*w;
            erreur_iterative = 0 - [y_list(i), 0, 0]*w;
            w=w-mu*[y_list(i); 0; 0]*(erreur_iterative);
        end
        if i == 2
            s_chapeau(i,compteur_mu) = [y_list(i), y_list(i-1), 0]*w;
            erreur_iterative = vecteurs_aleatoires_envoyes_list(i-1) -
[y_list(i), y_list(i-1), 0]*w;
            w=w-mu*[y_list(i); y_list(i-1); 0]*(erreur_iterative);
        end
        if i > 2
            s_chapeau(i,compteur_mu) = [y_list(i), y_list(i-1),
y_list(i-2)]*w;
            erreur_iterative = vecteurs_aleatoires_envoyes_list(i-1) -
[y_list(i), y_list(i-1), y_list(i-2)]*w;
```

---

```

        w=w-mu*[y_list(i); y_list(i-1);
y_list(i-2)]*(erreur_iterative);
    end
erreur_iterative_list(i,compteur_mu) = erreur_iterative;

% trouver s apres module de decision :
if s_chapeau(i,compteur_mu) > 0
    s_guess(i,compteur_mu) = -1;
else
    s_guess(i,compteur_mu) = 1;
end

end
% faire la moyenne des derniers elements du vecteurs d'erreur
% pour lisser la courbe :
vecteur_a_moyenner = erreur_iterative_list(:,compteur_mu);
% Nombre d'elements dans le vecteur
n = length(vecteur_a_moyenner);
% Nombre d'elements a considerer avant chaque element a moyenner
elements_avant = 5;
% Calcul des moyennes pour chaque ?lement avec les cinq pr?c?dents
for k = 1:n
    % Indices des ?lements ? considerer
    indices = max(1, k - elements_avant):k;

    % Calcul de la moyenne
    moyennes_erreur_iterative_list(k, compteur_mu) =
mean(vecteur_a_moyenner(indices));
end
compteur_mu = compteur_mu + 1;
end

% Calcul mmse question b
diff_carree = (s_guess(:,1) - s_envoye(:,1)).^2;
mmse_50symboles = mean(diff_carree);

% Calcul taux d'erreur par symbole sur les symboles envoy?s
difference_per_symbol = nnz(s_guess(:,1) ~= s_envoye(:,1)); % mu =
0.01
taux_erreur_par_symbole_50symboles = difference_per_symbol/
dataset_size;

% PLOT L'EVOLUTION DE L'APPRENTISSAGE :
x = 1:1:dataset_size;
fig1 = figure();
subplot(3,1,1);
plot(x,erreur_iterative_list(:,1), 'linewidth', 2);
hold on
plot(x,moyennes_erreur_iterative_list(:,1),'-r', 'linewidth', 2);
grid minor
title('Erreurs iteratives pour mu = 0.01 avec et sans lissage
(50symboles, III-a) ')

```

---

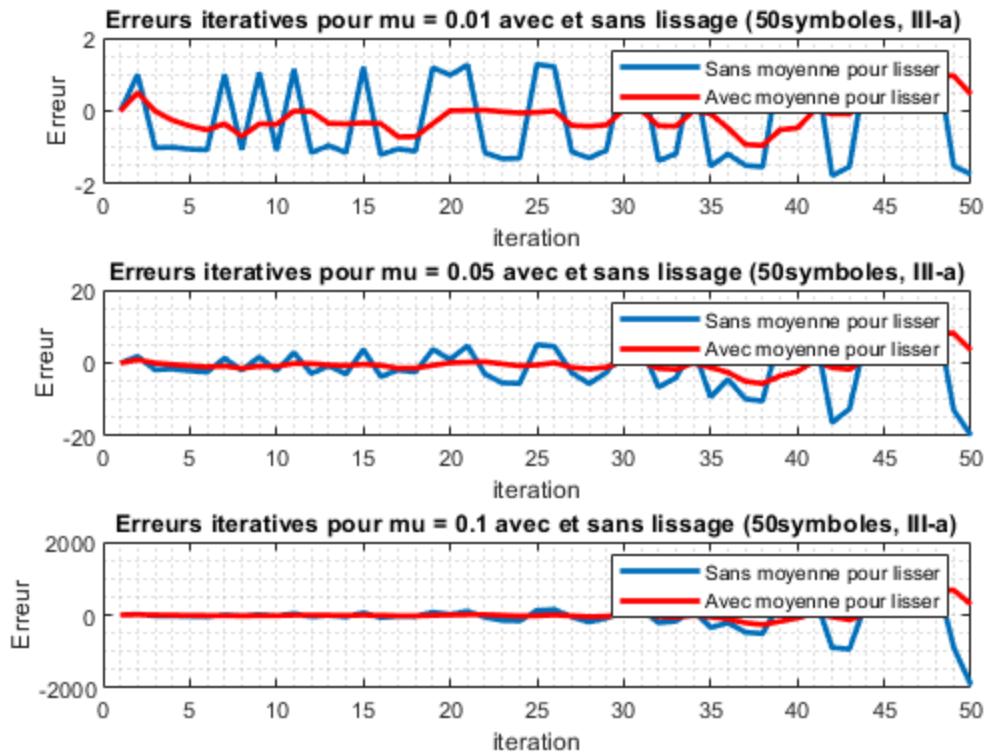
```

xlabel('iteration')
ylabel('Erreur');
legend('Sans moyenne pour lisser', 'Avec moyenne pour lisser')

subplot(3,1,2);
plot(x,erreur_iterative_list(:,2), 'linewidth', 2);
hold on
plot(x,moyennes_erreur_iterative_list(:,2),'-r', 'linewidth', 2);
grid minor
title('Erreurs itératives pour mu = 0.05 avec et sans lissage
(50symboles, III-a)')
xlabel('iteration')
ylabel('Erreur');
legend('Sans moyenne pour lisser', 'Avec moyenne pour lisser')

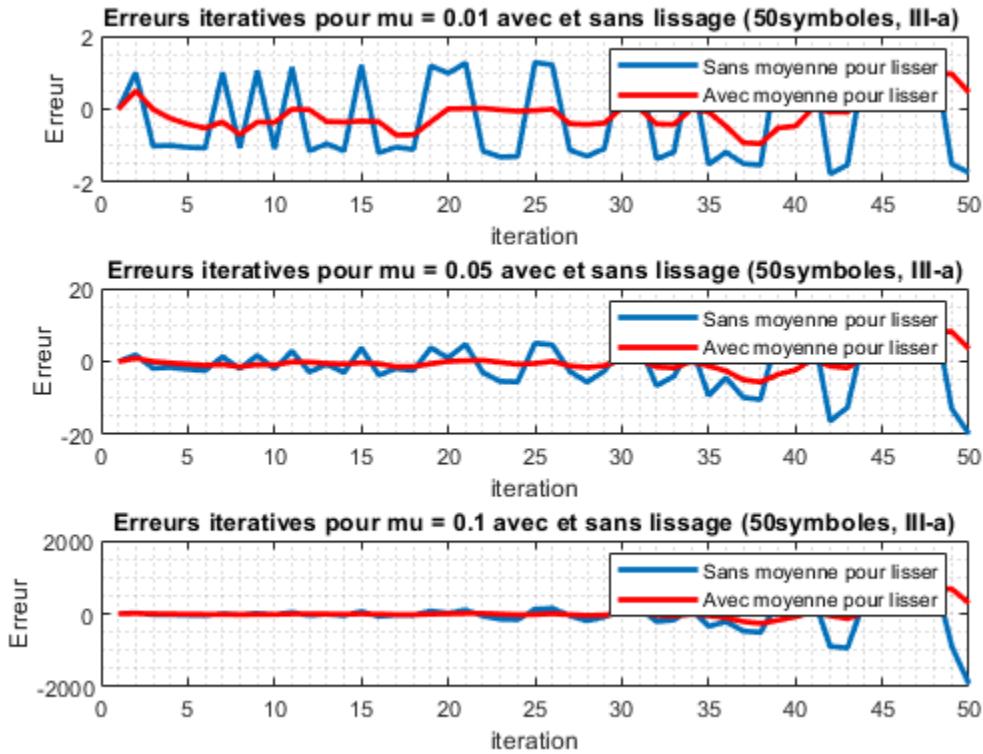
subplot(3,1,3);
plot(x,erreur_iterative_list(:,3), 'linewidth', 2);
hold on
plot(x,moyennes_erreur_iterative_list(:,3),'-r', 'linewidth', 2);
grid minor
title('Erreurs itératives pour mu = 0.1 avec et sans lissage
(50symboles, III-a)')
xlabel('iteration')
ylabel('Erreur');
legend('Sans moyenne pour lisser', 'Avec moyenne pour lisser')

```



# Export de la figure (automatique, eh oui!)

```
%1) en format vectoriel .eps pour les vrais qui utilisent LaTeX  
print(fig1,'-depsc',strcat(path,nom_image,'.eps'))  
%2) en format PNG (haute resolution 600dpi, pour ceux qui utilisent  
Word)  
print(fig1,'-dpng',' -r600',strcat(path,nom_image,'.png'))
```



# Egalisateur LMS (entraînement avec 500 symboles)

```
dataset_size = 500; %% INPUT, A MODIFIER  
w=zeros(3,1);  
y_list = y(:);  
vecteurs_aleatoires_envoyes_list = vecteurs_aleatoires_envoyes(:);  
erreur_iterative_list = 233423411*ones(dataset_size,3);  
compteur_mu=1;  
% Initialisation du vecteur pour stocker les moyennes  
moyennes_erreur_iterative_list = zeros(dataset_size, 3);  
s_chapeau = 34342342*ones(dataset_size,3);  
s_guess = 34342342*ones(dataset_size,3);  
s_envoye = vecteurs_aleatoires_envoyes_list(1:dataset_size);  
for mu=[0.01 0.05 0.1]  
    for i=1:1:dataset_size
```

---

```

    if i == 1
        s_chapeau(i,compteur_mu) = [y_list(i), 0, 0]*w;
        erreur_iterative = 0 - [y_list(i), 0, 0]*w;
        w=w-mu*[y_list(i); 0; 0]*(erreur_iterative);
    end
    if i == 2
        s_chapeau(i,compteur_mu) = [y_list(i), y_list(i-1), 0]*w;
        erreur_iterative = vecteurs_aleatoires_envoyes_list(i-1) -
[y_list(i), y_list(i-1), 0]*w;
        w=w-mu*[y_list(i); y_list(i-1); 0]*(erreur_iterative);
    end
    if i > 2
        s_chapeau(i,compteur_mu) = [y_list(i), y_list(i-1),
y_list(i-2)]*w;
        erreur_iterative = vecteurs_aleatoires_envoyes_list(i-1) -
[y_list(i), y_list(i-1), y_list(i-2)]*w;
        w=w-mu*[y_list(i); y_list(i-1);
y_list(i-2)]*(erreur_iterative);
    end
    erreur_iterative_list(i,compteur_mu) = erreur_iterative;

    % trouver s apres module de decision :
    if s_chapeau(i,compteur_mu) > 0
        s_guess(i,compteur_mu) = -1;
    else
        s_guess(i,compteur_mu) = 1;
    end

end
% faire la moyenne des derniers elements du vecteurs d'erreur
% pour lisser la courbe :
vecteur_a_moyenner = erreur_iterative_list(:,compteur_mu);
% Nombre d'elements dans le vecteur
n = length(vecteur_a_moyenner);
% Nombre d'elements a considerer avant chaque element a moyenner
elements_avant = 5;
% Calcul des moyennes pour chaque ?l?ment avec les cinq pr?c?dents
for k = 1:n
    % Indices des ?l?ments ? considerer
    indices = max(1, k - elements_avant):k;

    % Calcul de la moyenne
    moyennes_erreur_iterative_list(k, compteur_mu) =
mean(vecteur_a_moyenner(indices));
end
compteur_mu = compteur_mu + 1;
end

% Calcul mmse question b
diff_carree = (s_guess(:,1) - s_envoye(:,1)).^2;
mmse_500symboles = mean(diff_carree);
disp("mmse_50symboles");
disp(mmse_50symboles);
disp("mmse_500symboles");

```

---

---

```
disp(mmse_500symboles);

% Calcul taux d'erreur par symbole sur les symboles envoyés
difference_per_symbol = nnz(s_guess(:,1) ~= s_envoye(:,1)); % mu =
0.01
taux_erreur_par_symbole_500symboles = difference_per_symbol/
dataset_size;
disp("taux_erreur_par_symbole_50symboles");
disp(taux_erreur_par_symbole_50symboles);
disp("taux_erreur_par_symbole_500symboles");
disp(taux_erreur_par_symbole_500symboles);

mmse_50symboles
2

mmse_500symboles
1.7440

taux_erreur_par_symbole_50symboles
0.5000

taux_erreur_par_symbole_500symboles
0.4360
```

## Commentaires question b)

Nous remarquons que nous avons des performances moins bonnes en entrainant avec 500 symboles plutot que 50 symboles (mmse et taux d'erreur plus eleve). L'entraînement semble moins performant que les méthodes utilisées dans les parties précédentes. Au vu de la difference entre les taux d'erreurs ici, en comparaison aux taux d'erreurs des parties I et II, il pourrait être intéressant d'essayer un module de décision autre (on pense notamment au module de décision DFS).

*Published with MATLAB® R2019b*