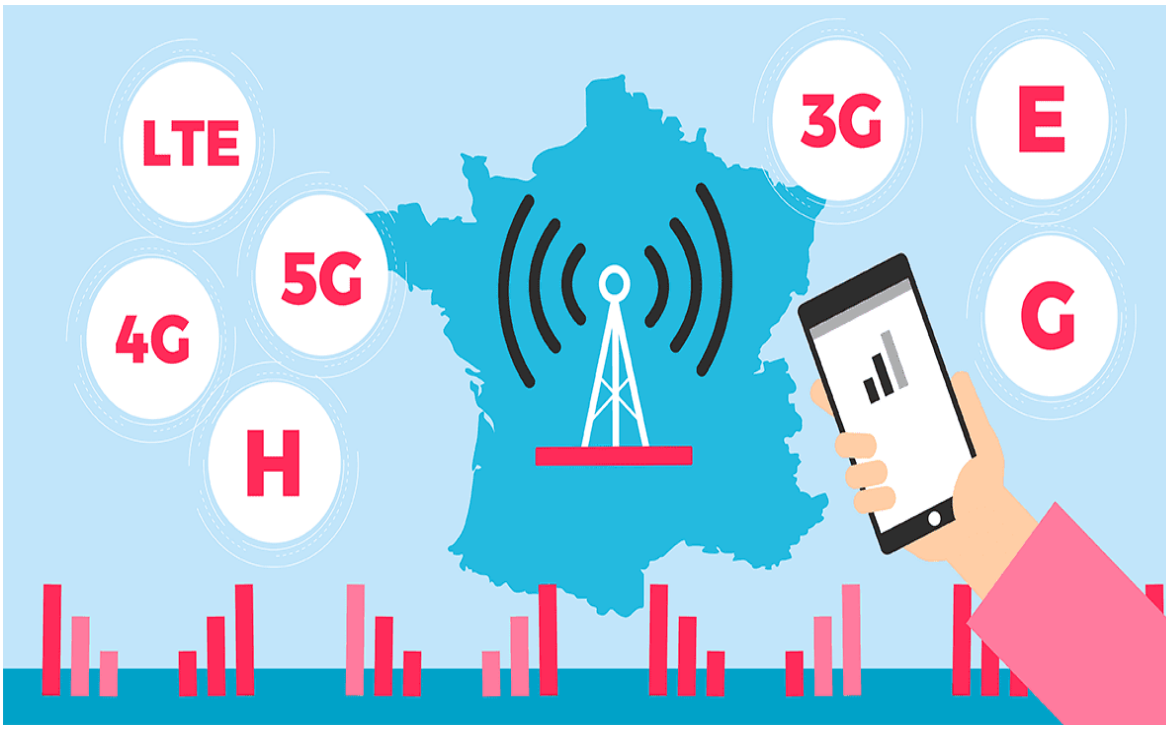
الكلية متعددة التخصصات تازة
جامعة سيدي محمد بن عبد الله بفاس
FACULTÉ POLYDISCIPLINAIRE DE TAZA
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH DE FES

Module « Réseaux mobiles »

~TP : simulation des différents concepts du cours~



Réalisé par :
OUGNI Imane

Encadré par :
Pr. TAZI El bachir

Table des matières

Introduction

Dans ce rapport, nous abordons les concepts fondamentaux de la transmission de données et de la modulation des signaux dans le contexte des communications numériques. À travers des simulations réalisées sous MATLAB, qui est un environnement de calcul scientifique largement utilisé pour la modélisation, l'analyse et la visualisation de systèmes complexes, nous explorons les principales notions telles que le débit binaire, le rapport signal sur bruit (SNR), le taux d'erreur binaire (BER) et la capacité de canal selon la formule de Shannon…

1. Bases de la transmission et de la modulation

1.1. Les bases de transmissions

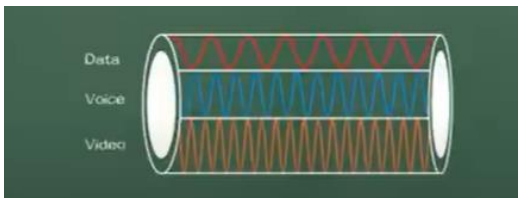
a. baseband vs broadband

Base de bande (baseband) :

Le signal en baseband est transmis tel quel, sans modulation. Il utilise une seule fréquence, généralement proche de zéro. Ce type de transmission est utilisé pour des courtes distances, comme dans les câbles Ethernet où le signal est envoyé directement sans transformation.

Bande large (broadband) :

La transmission broadband permet de faire passer plusieurs signaux en même temps, chacun sur une fréquence différente. Cela permet d'envoyer beaucoup d'informations à la fois. C'est utilisé, par exemple, pour l'accès Internet par câble ou fibre optique.



Bande passante (bandwidth) :

La bande passante est la différence entre la fréquence la plus haute et la plus basse qu'un canal peut transmettre. Plus elle est grande, plus on peut envoyer d'informations.

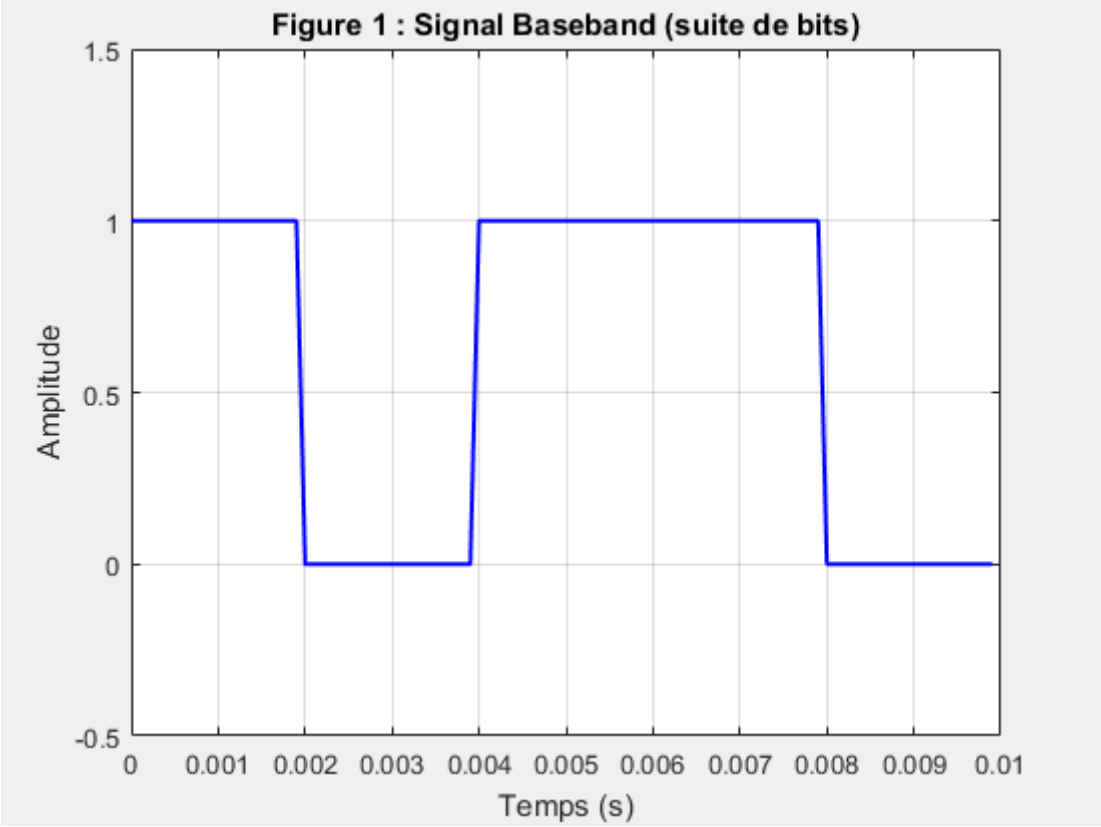
- **Code :**
- **Le fichier : baseband_vs_broadband.m**

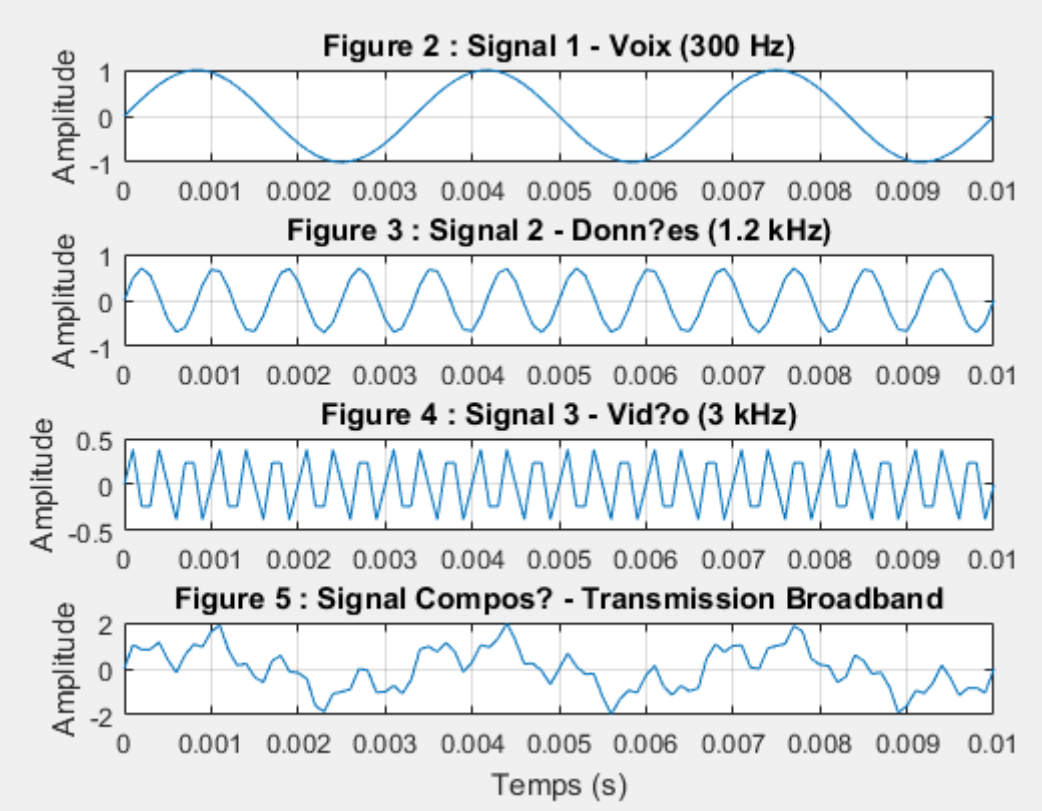
```
clc; clear; close all;

%% 1) SIGNAL BASEBAND : Suite de bits
Fs = 10000;              % Fr?quence d'?chantillonnage (10 kHz)
Tb = 0.002;              % Dur?e d?un bit = 2 ms
bits = [1 0 1 1 0];      % Suite de bits ? transmettre

% G?n?ration du signal baseband
baseband = [];
for i = 1:length(bits)
```

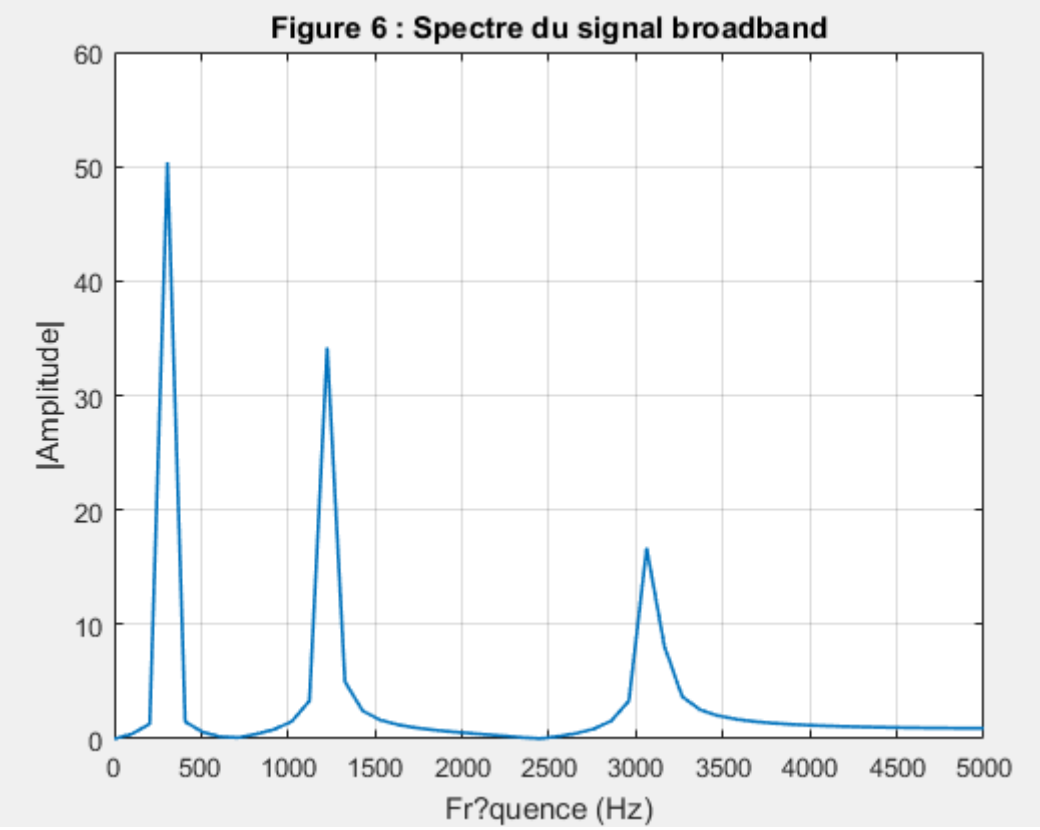
```
    baseband = [baseband bits(i)*ones(1,Fs*Tb)];
end

t1 = (0:length(baseband)-1)/Fs; % Temps associ?

% === Affichage du signal baseband ===
figure;
plot(t1, baseband, 'b', 'LineWidth', 1.5);
title('Figure 1 : Signal Baseband (suite de bits)');
xlabel('Temps (s)');
ylabel('Amplitude');
ylim([-0.5 1.5]); grid on;

%% 2) SIGNAL BROADBAND : Plusieurs signaux sur diff?rentes fr?quences
fs = 10000;             % Fr?quence d'?chantillonnage
t2 = 0:1/fs:0.01;       % 10 ms

% Signal 1 : voix (basse fr?quence)
f1 = 300;               % 300 Hz
s1 = sin(2*pi*f1*t2);

% Signal 2 : donn?es (fr?quence moyenne)
f2 = 1200;              % 1.2 kHz
s2 = 0.7 * sin(2*pi*f2*t2);

% Signal 3 : vid?o (haute fr?quence)
f3 = 3000;              % 3 kHz
s3 = 0.4 * sin(2*pi*f3*t2);

% Signal total broadband = somme des 3
broadband_signal = s1 + s2 + s3;

% === Affichage temporel des signaux ===
figure;
subplot(4,1,1);
plot(t2, s1);
title('Figure 2 : Signal 1 - Voix (300 Hz)');
ylabel('Amplitude'); grid on;

subplot(4,1,2);
plot(t2, s2);
title('Figure 3 : Signal 2 - Donn?es (1.2 kHz)');
ylabel('Amplitude'); grid on;

subplot(4,1,3);
plot(t2, s3);
title('Figure 4 : Signal 3 - Vid?o (3 kHz)');
ylabel('Amplitude'); grid on;

subplot(4,1,4);
plot(t2, broadband_signal);
```

```
title('Figure 5 : Signal Compos? - Transmission Broadband');
xlabel('Temps (s)'); ylabel('Amplitude'); grid on;

%% 3) Analyse fr?quentielle du signal broadband
n = length(broadband_signal);
Y = fft(broadband_signal);
n = length(Y);
Y = abs(Y(1:floor(n/2)));
f = linspace(0, fs/2, length(Y));

% === Affichage du spectre (FFT) ===
figure;
plot(f, Y, 'LineWidth', 1.2);
title('Figure 6 : Spectre du signal broadband');
xlabel('Fr?quence (Hz)');
ylabel('|Amplitude|');
grid on;
```

↳ **Résultat :**







- **Interpretation :**

Le signal baseband transmet directement une suite de bits sous forme de niveaux fixes sans modulation. En revanche, le signal broadband peut transporter plusieurs types de données simultanément (voix, données, vidéo), chacun sur une fréquence différente.Les figures montrent la composition temporelle et le spectre fréquentiel, où l'on observe distinctement les composantes à 300 Hz, 1.2 kHz et 3 kHz.

b. Débit binaire, SNR, BER, latence

Débit binaire (bit rate) :

C’est la quantité de bits transmis par seconde sur un canal, son unité est bit/s (bps).

- Plus le débit est élevé, plus la transmission est rapide.

Rapport signal sur bruit (SNR) :

C’est le rapport entre la puissance du signal utile et celle du bruit, son unité est décibels (dB).

- Plus le SNR est élevé, donc la qualité du signal est meilleure et moins d’erreurs de transmission.

Taux d’erreur (BER) :

C’est le taux d’erreur binaire, c’est-à-dire le nombre de bits reçus incorrectement par rapport au total transmis.

- Plus le BER est faible, meilleure est la fiabilité.

Latence (delay spread) :

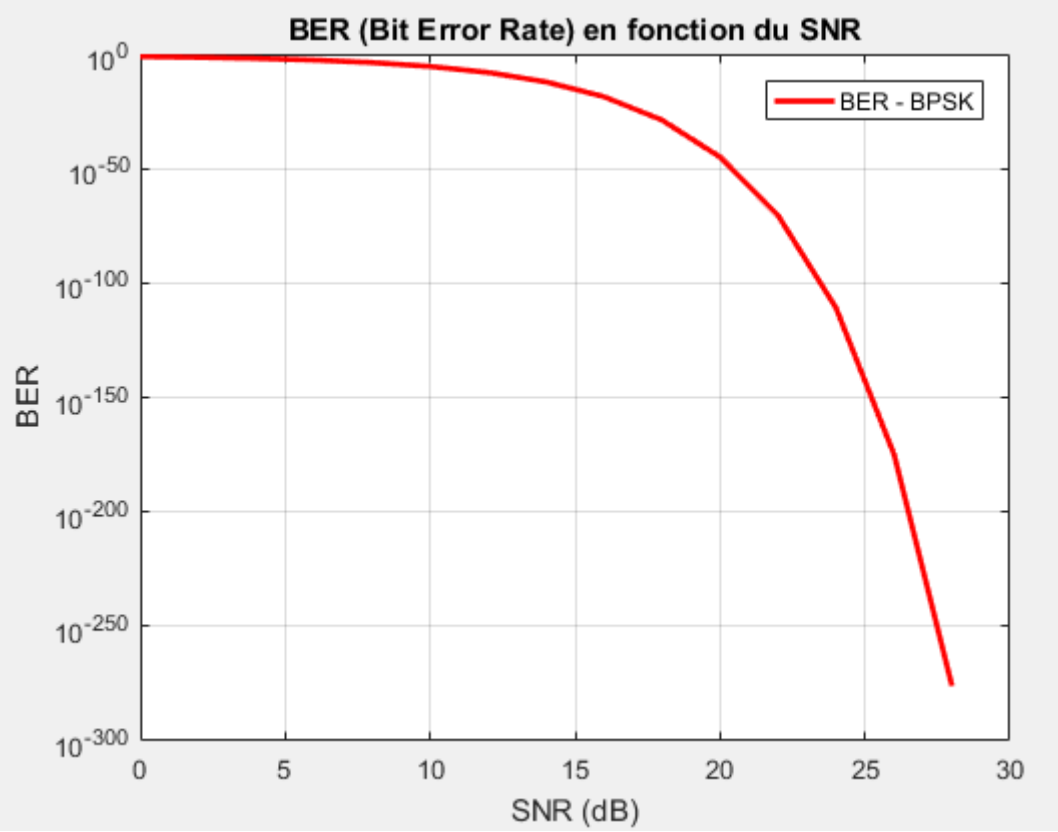
C’est le temps de retard entre l’envoi d’un message et sa réception, son unité est millisecondes (ms).

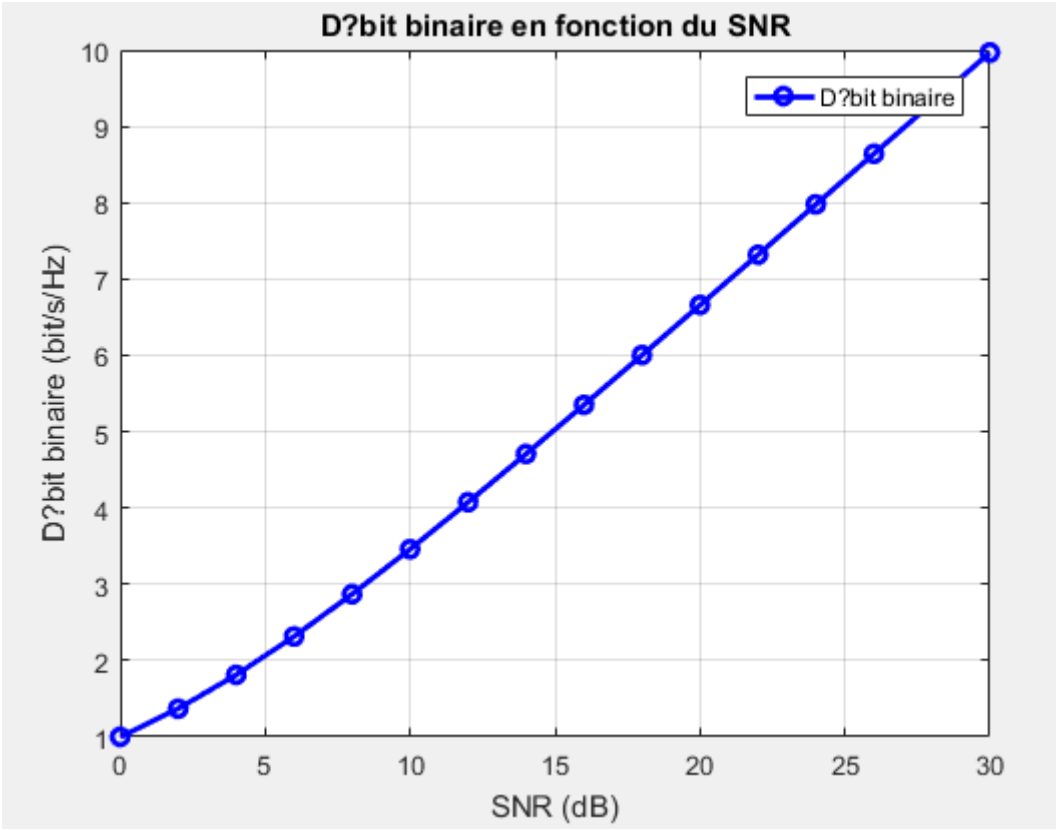
- Faible latence = réponse rapide
- **Code :**
- **Le fichier : debit_BER_SNR_latence.m**

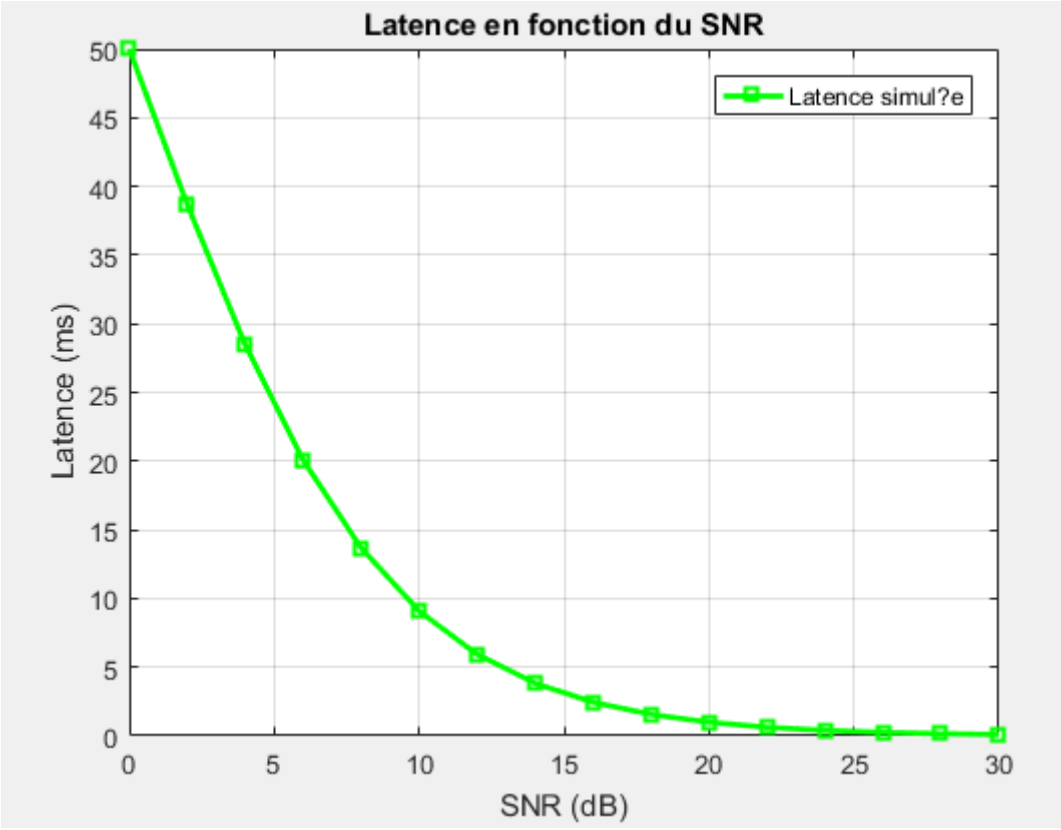
```
clc; clear; close all;

% --- Param?tres simul?s ---
SNR_dB = 0:2:30;              % SNR de 0 ? 30 dB
SNR_linear = 10.^(SNR_dB/10);  % Conversion en ?chelle lin?aire

% --- BER th?orique pour BPSK (modulation binaire simple) ---
BER = qfunc(sqrt(2 * SNR_linear));  % Fonction Q pour le BER BPSK

% --- D?bit binaire simul? en fonction du SNR ---
% Hypoth?se simple : 1 bit/s/Hz * log2(1 + SNR)
bitrate = log2(1 + SNR_linear);     % D?bit binaire normalis?

% --- Latence simul?e : inversement proportionnelle au SNR ---
latence = 100 ./ (1 + SNR_linear);  % en millisecondes

% === AFFICHAGE DES COURBES ===

% 1. Courbe BER vs SNR
figure;
semilogy(SNR_dB, BER, 'r', 'LineWidth', 2);
grid on;
xlabel('SNR (dB)');
ylabel('BER');
title('BER (Bit Error Rate) en fonction du SNR');
legend('BER - BPSK');

% 2. D?bit binaire vs SNR
figure;
plot(SNR_dB, bitrate, 'b-o', 'LineWidth', 2);
grid on;
xlabel('SNR (dB)');
```

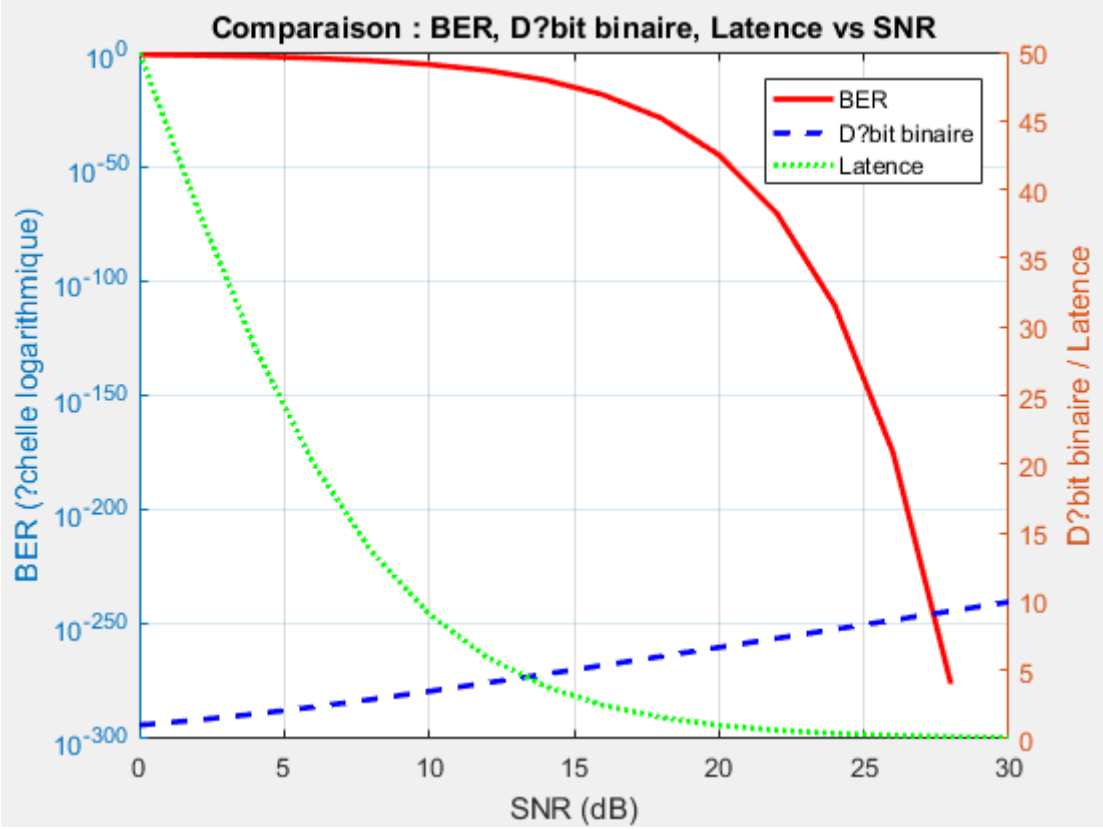
```
ylabel('D?bit binaire (bit/s/Hz)');
title('D?bit binaire en fonction du SNR');
legend('D?bit binaire');

% 3. Latence vs SNR
figure;
plot(SNR_dB, latence, 'g-s', 'LineWidth', 2);
grid on;
xlabel('SNR (dB)');
ylabel('Latence (ms)');
title('Latence en fonction du SNR');
legend('Latence simul?e');

% 4. Courbes combin?es sur une m?me figure
figure;
yyaxis left;
semilogy(SNR_dB, BER, 'r', 'LineWidth', 2);
ylabel('BER (?chelle logarithmique)');

yyaxis right;
plot(SNR_dB, bitrate, 'b--', 'LineWidth', 2);
hold on;
plot(SNR_dB, latence, 'g:', 'LineWidth', 2);
ylabel('D?bit binaire / Latence');

xlabel('SNR (dB)');
title('Comparaison : BER, D?bit binaire, Latence vs SNR');
legend('BER', 'D?bit binaire', 'Latence');
grid on;
```

- **Résultat :**




D?bit binaire en fonction du SNR
D?bit binaire
D?bit binaire (bit/s/Hz)
SNR (dB)
0
5
10
15
20
25
30
1
2
3
4
5
6
7
8
9
10


Latence en fonction du SNR
Latence simul?e
Latence (ms)
SNR (dB)
0
5
10
15
20
25
30
35
40
45
50



↳ **Interpretation :**

- BER vs SNR : plus le SNR augmente, plus le taux d'erreur diminue (moins d'erreurs). Cela montre que la qualité du signal améliore la fiabilité de la transmission.
- Débit binaire vs SNR : avec un meilleur SNR, le débit binaire augmente (canal plus efficace), ce qui signifie une transmission plus rapide.
- Latence vs SNR : la latence diminue quand le SNR augmente, car les erreurs sont moins fréquentes, donc moins de retransmissions (transmission plus rapide).
- Courbe combinée : permet de comparer l'évolution simultanée du BER, du débit et de la latence selon le SNR.

c. Capacité de bande passante – Formule de Shannon

La capacité d'un canal selon shanon, c'est la quantité maximale de données qu'on peut transmettre sans erreur sur un canal en tenant compte du bruit (perturbations), son unité est bits par secondes (bits/s)

$$C=D_{max}\cdot\log_2(1+N/S)$$

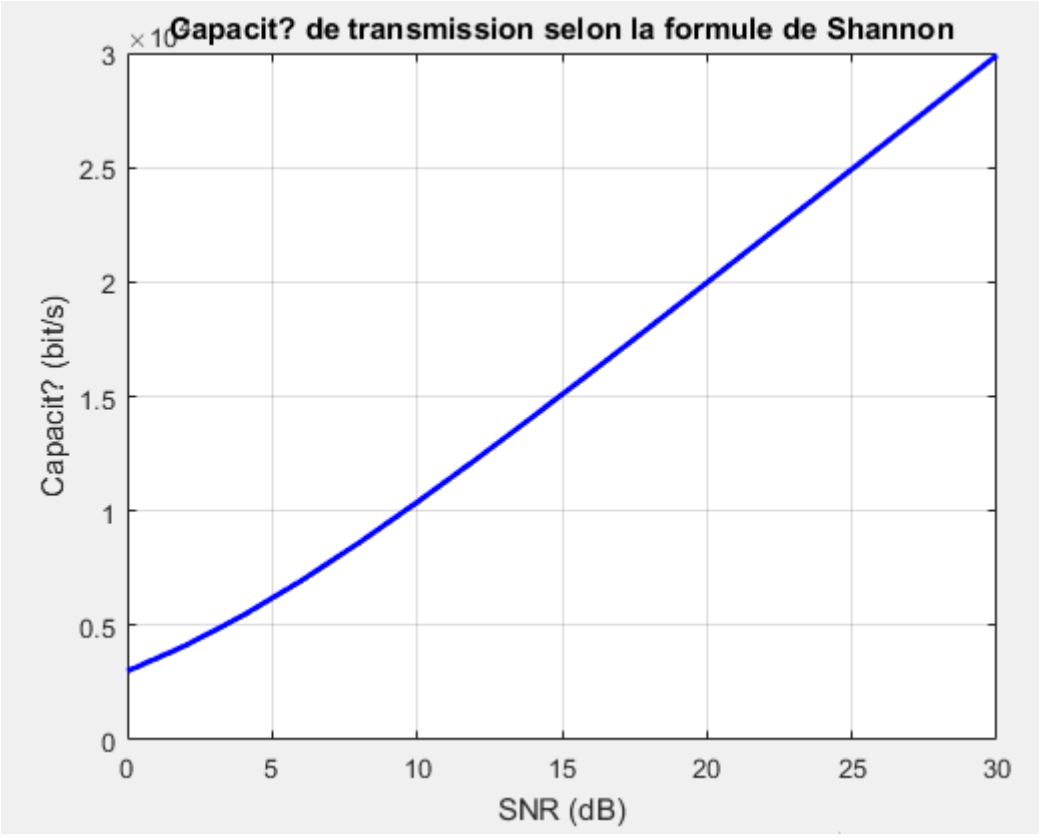
↳ **Code :**

⇨ **Le fichier : formule_shanon.m**

```
clc; clear; close all;

B = 3000;                    % Bande passante en Hz (ex: 3 kHz)
SNR_dB = 0:2:30;             % SNR en dB
SNR_linear = 10.^(SNR_dB/10);     % Conversion en lin?aire

C = B * log2(1 + SNR_linear);     % Capacit? selon Shannon

figure;
plot(SNR_dB, C, 'b', 'LineWidth', 2);
xlabel('SNR (dB)');
ylabel('Capacit? (bit/s)');
```

```
title('Capacit? de transmission selon la formule de Shannon');
grid on;
```

- **Résultat :**



- **Interpretation :**

Ce code utilise la formule de Shannon pour calculer la capacité maximale d'un canal de transmission en fonction du rapport signal sur bruit (SNR). Il montre que plus le SNR augmente, plus la capacité (en bit/s) du canal augmente. La bande passante est fixée à 3000 Hz, et on observe que la capacité dépend directement de la qualité du signal.

d. Signal analogique vs numérique

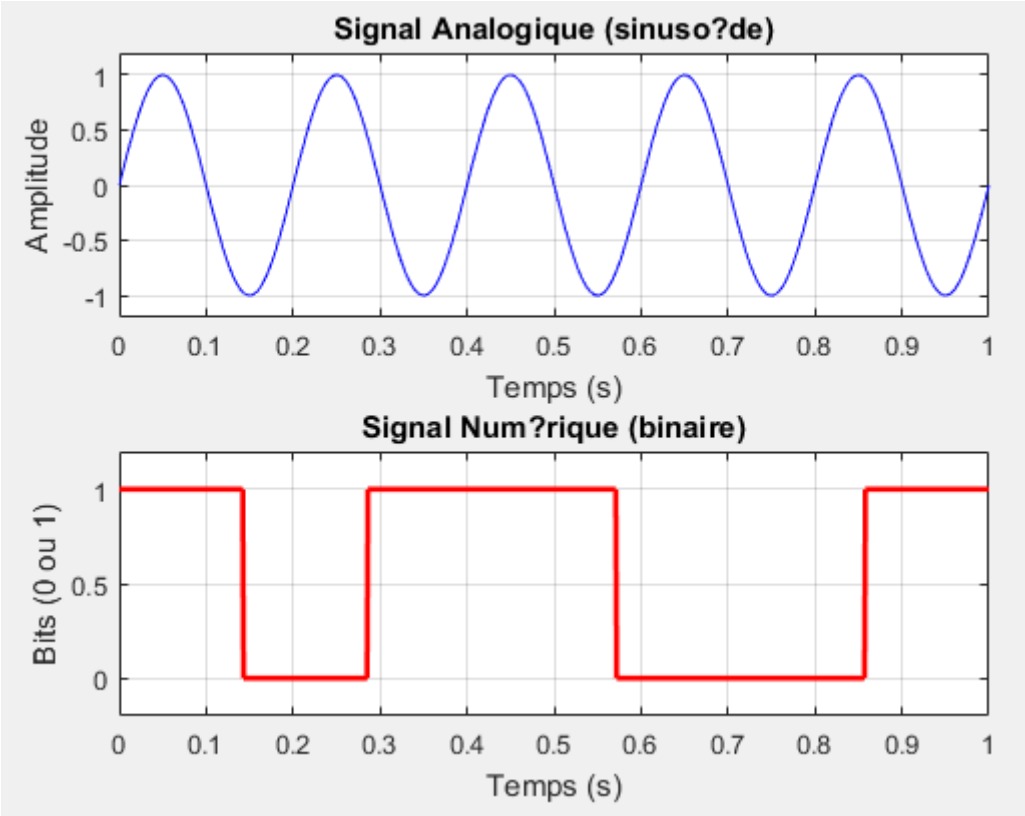
Un signal analogique (onde continue) peut prendre une infinité de valeurs et un signal numérique (signal discret) se resume uniquement en une succession de 0 et de 1.

- **Code :**
  - **Le fichier : analogique_numerique.m**

```
clc; clear; close all;

% --- Param?tres ---
fs = 1000;              % Fr?quence d'?chantillonnage
t = 0:1/fs:1;           % 1 seconde

% --- Signal analogique : sinuso?de continue ---
analog_signal = sin(2*pi*5*t); % 5 Hz

% --- Signal num?rique : suite binaire ---
digital_bits = [1 0 1 1 0 0 1];
bit_duration = 1 / 7;           % Chaque bit dure 1/7 seconde
digital_signal = repelem(digital_bits, length(t)/length(digital_bits));
```

```
% --- Affichage ---
figure;

% Signal analogique
subplot(2,1,1);
plot(t, analog_signal, 'b');
title('Signal Analogique (sinuso?de)');
xlabel('Temps (s)');
ylabel('Amplitude');
grid on;
ylim([-1.2 1.2]);

% Signal num?rique
subplot(2,1,2);
plot(t, digital_signal, 'r', 'LineWidth', 2);
title('Signal Num?rique (binaire)');
xlabel('Temps (s)');
ylabel('Bits (0 ou 1)');
grid on;
ylim([-0.2 1.2]);
```

- **Résultat :**



e. Transmission en serie Vs parallele

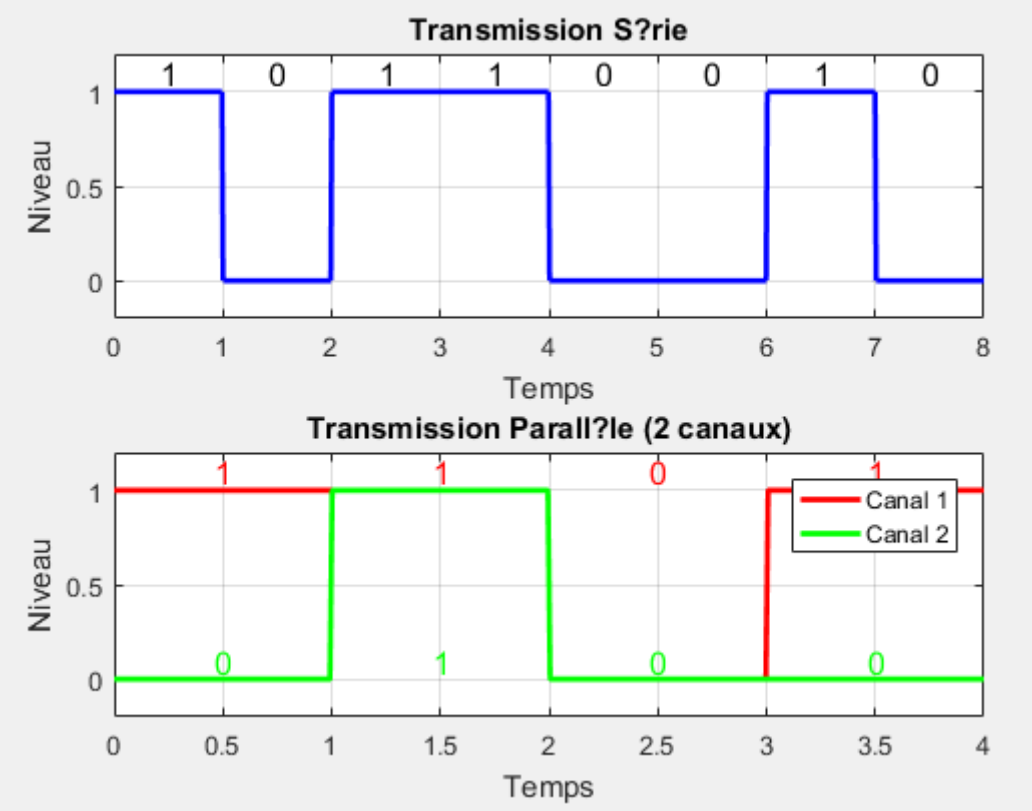
En transmission **série**, les bits sont envoyés un par un sur une seule ligne, ce qui est simple mais plus lent. En transmission **parallèle**, plusieurs bits sont transmis simultanément sur plusieurs lignes, ce qui est plus rapide mais nécessite plus de câblage

- **Code :**
  - **Le fichier : transmission_serie_parallele.m**

```
% Donn?es binaires ? transmettre
bits = [1 0 1 1 0 0 1 0];

% ============================
% TRANSMISSION S?RIE
% ============================
T_bit = 1;                      % dur?e d?un bit
Fs = 100;                       % fr?quence d??chantillonnage pour l?affichage
t_serie = linspace(0, length(bits), length(bits)*Fs);

% Signal s?rie r?p?t? (chaque bit r?p?t? Fs fois)
signal_serie = repelem(bits, Fs);

% ============================
% TRANSMISSION PARALL?LE
% ============================
bits_parallel = reshape(bits, 2, []); % 2 lignes (canaux), plusieurs colonnes
signal_par_1 = repelem(bits_parallel(1,:), Fs);
signal_par_2 = repelem(bits_parallel(2,:), Fs);
t_parallele = linspace(0, size(bits_parallel,2), length(signal_par_1));

% ============================
% AFFICHAGE
% ============================
figure;

% Transmission s?rie
subplot(2,1,1);
plot(t_serie, signal_serie, 'b', 'LineWidth', 2);
ylim([-0.2 1.2]); grid on;
xlabel('Temps');
ylabel('Niveau');
title('Transmission S?rie');

% Ajouter les bits au-dessus de chaque segment
for k = 1:length(bits)
    text(k-0.5, 1.1, num2str(bits(k)), 'FontSize', 12, 'HorizontalAlignment', 'center');
end

% Transmission parall?le
subplot(2,1,2);
plot(t_parallele, signal_par_1, 'r', 'LineWidth', 2); hold on;
plot(t_parallele, signal_par_2, 'g', 'LineWidth', 2);
ylim([-0.2 1.2]); grid on;
xlabel('Temps');
ylabel('Niveau');
title('Transmission Parall?le (2 canaux)');
legend('Canal 1', 'Canal 2');

% Ajouter les bits pour canal 1
for k = 1:size(bits_parallel,2)
```

```
    text(k-0.5, 1.1, num2str(bits_parallel(1,k)), 'Color', 'r', 'FontSize', 12, 'HorizontalAlignment', 'center');
end
% Ajouter les bits pour canal 2
for k = 1:size(bits_parallel,2)
    text(k-0.5, 0.1, num2str(bits_parallel(2,k)), 'Color', 'g', 'FontSize', 12, 'HorizontalAlignment', 'center');
end
```

- **Résultat :**



f. Numérisation d'un signal analogique

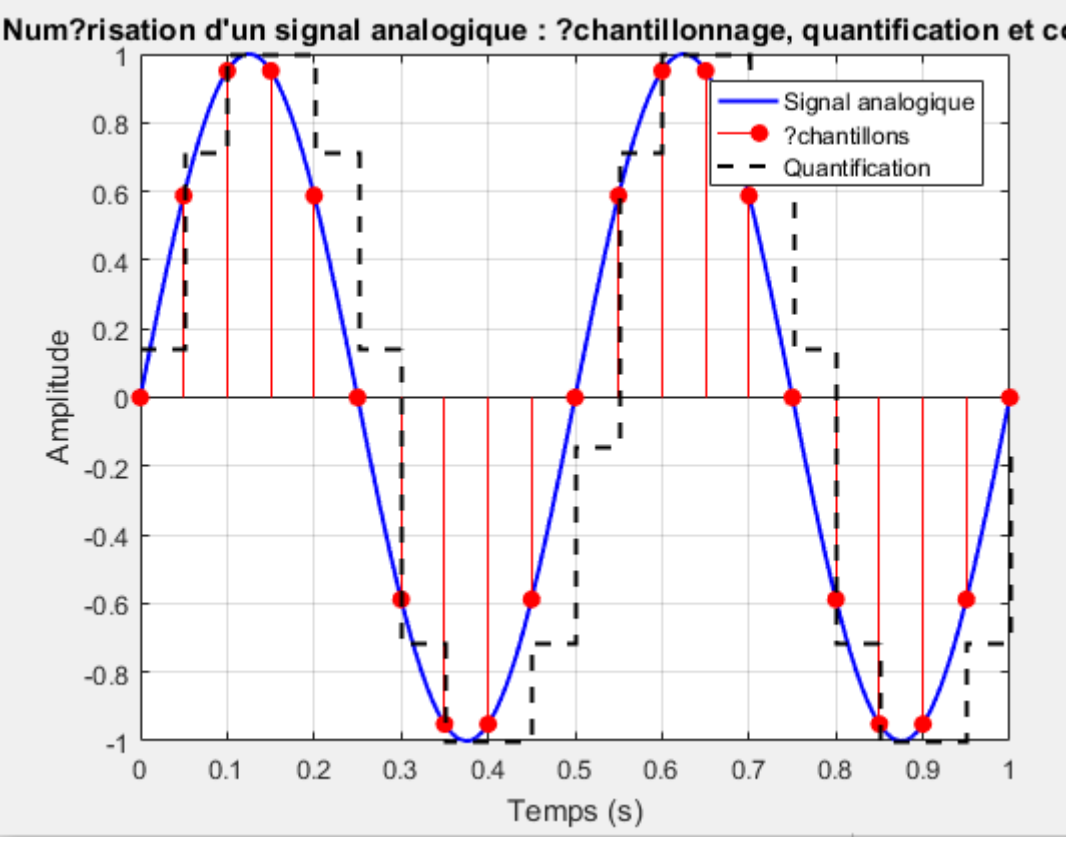
La numérisation consiste à convertir un signal analogique continu en une suite de valeurs numériques. Elle se fait en trois étapes principales : d'abord l'échantillonnage, où le signal est mesuré à des instants discrets ; puis la quantification, qui consiste à arrondir ces mesures à un nombre fini de niveaux prédéfinis ; enfin le codage, où chaque niveau quantifié est converti en un code binaire. Ce processus permet de représenter le signal original sous forme numérique, ce qui facilite son traitement, sa transmission et son stockage.

- **Code :**
  - **Le fichier : numerisation.m**

```
clc; clear; close all;

% Signal analogique (ex. sinuso?de)
f = 2;                    % fr?quence du signal (Hz)
t_continu = 0:0.001:1;    % temps "continu" (pas fin)
x_analogique = sin(2*pi*f*t_continu);  % signal analogique

% === ?CHANTILLONNAGE ===
Fe = 20;                  % fr?quence d'?chantillonnage (Hz)
Te = 1/Fe;
t_ech = 0:Te:1;           % instants d'?chantillonnage
x_echant = sin(2*pi*f*t_ech);  % ?chantillons du signal
```

```
% === QUANTIFICATION ===
nb_niveaux = 8;                    % quantification ? 8 niveaux (3 bits)
x_min = -1; x_max = 1;
q_step = (x_max - x_min)/(nb_niveaux-1);  % pas de quantification
x_quant = round((x_echant - x_min)/q_step) * q_step + x_min;  % quantification uniforme

% === CODAGE BINAIRE ===
indices = round((x_quant - x_min)/q_step);      % indices des niveaux (0 ? 7)
nb_bits = ceil(log2(nb_niveaux));              % nombre de bits par niveau (3 bits)
bin_codes = dec2bin(indices, nb_bits);          % conversion indices -> binaire

% Affichage codes binaires dans la console
fprintf('Echantillon\tAmplitude\tIndice\tCode binaire\n');
for i=1:length(indices)
    fprintf('%d\t\t%.2f\t\t%d\t%s\n', i, x_quant(i), indices(i), bin_codes(i,:));
end

% === AFFICHAGE GRAPHIQUE ===
figure;

% Signal analogique continu
plot(t_continu, x_analogique, 'b', 'LineWidth', 1.5); hold on;

% Points d'?chantillonnage
stem(t_ech, x_echant, 'r', 'filled');

% Signal quantifi? (marche d'escalier)
stairs(t_ech, x_quant, 'k--', 'LineWidth', 1.5);

grid on;
xlabel('Temps (s)');
ylabel('Amplitude');
title('Num?risation d''un signal analogique : ?chantillonnage, quantification et codage');
legend('Signal analogique', '?chantillons', 'Quantification');
```

- **Résultat :**



1.2. Principe de la modulation

a. Amplitude, Fréquence et Phase

Pour un signal sinusoïdal $s(t)=A\cdot \sin(2\pi f_0 t + \phi)$, on va fixer deux paramètres et faire varier le troisième.

Amplitude : mesure l'intensité ou la hauteur d'un signal (une onde), c'est la distance entre le point de repos (le 0) et le sommet ou le creux de l'onde.

Phase : décrit la position d'un point dans un cycle d'onde par rapport à un point de référence. Cela permet de savoir à quel moment un phénomène se produit dans le temps par rapport aux autres.

Fréquence : mesure combien de fois une onde se répète dans un certain temps.

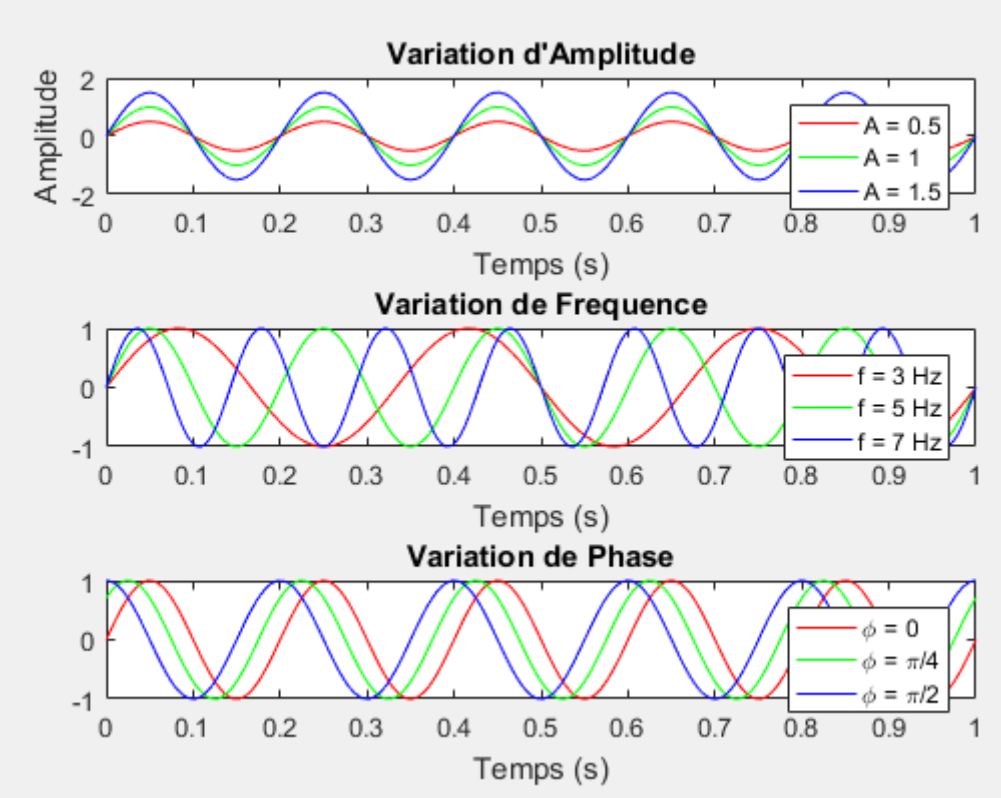
↳ **Code :**

⇨ **Le fichier : Frq_Amp_Phase.m**

```
% Temps
t = 0:0.001:1;  % 1 seconde, r?solution 1 ms

% Valeurs fixes de base
A = 1;      % Amplitude fixe
f = 5;      % Fr?quence fixe (Hz)
phi = 0;    % Phase fixe (radians)

% === 1. Variation d'amplitude ===
A1 = 0.5; A2 = 1; A3 = 1.5;
s1 = A1 * sin(2*pi*f*t + phi);
s2 = A2 * sin(2*pi*f*t + phi);
s3 = A3 * sin(2*pi*f*t + phi);

figure;
```

```
subplot(3,1,1);
plot(t, s1, 'r', t, s2, 'g', t, s3, 'b');
title('Variation d''Amplitude');
legend('A = 0.5','A = 1','A = 1.5');
xlabel('Temps (s)');
ylabel('Amplitude');

% === 2. Variation de fr?quence ===
A = 1;
f1 = 3; f2 = 5; f3 = 7;
s1 = A * sin(2*pi*f1*t + phi);
s2 = A * sin(2*pi*f2*t + phi);
s3 = A * sin(2*pi*f3*t + phi);

subplot(3,1,2);
plot(t, s1, 'r', t, s2, 'g', t, s3, 'b');
title('Variation de Frequence');
legend('f = 3 Hz','f = 5 Hz','f = 7 Hz');
xlabel('Temps (s)');

% === 3. Variation de phase ===
phi1 = 0; phi2 = pi/4; phi3 = pi/2;
f = 5;
s1 = A * sin(2*pi*f*t + phi1);
s2 = A * sin(2*pi*f*t + phi2);
s3 = A * sin(2*pi*f*t + phi3);

subplot(3,1,3);
plot(t, s1, 'r', t, s2, 'g', t, s3, 'b');
title('Variation de Phase');
legend('\phi = 0','\phi = \pi/4','\phi = \pi/2');
xlabel('Temps (s)');
```

- **Résultat :**



b. La modulation

La modulation est une technique utilisée en télécommunication pour adapter un signal d'information (généralement de basse fréquence) à un canal de transmission. Elle consiste à modifier une onde porteuse (signal de fréquence plus élevée) en faisant varier l'une de ses propriétés fondamentales : l'amplitude, la fréquence ou la phase. Cette adaptation permet de rendre le signal plus compatible avec le canal, d'améliorer la fiabilité de la transmission et d'optimiser l'utilisation de la bande passante.

➢ Modulation analogique

Modulation d'Amplitude (AM) : est simple à mettre en œuvre mais reste sensible au bruit et aux interférences, ce qui peut dégrader la qualité de la transmission.

Modulation de Fréquence (FM) : est plus robuste face au bruit que l'AM et est couramment utilisée pour la transmission audio, comme dans la radio FM, offrant une meilleure qualité de signal.

Modulation de Phase (PM) : génère des variations subtiles dans l'onde et est très utilisée dans les communications numériques pour transmettre les données de manière fiable.

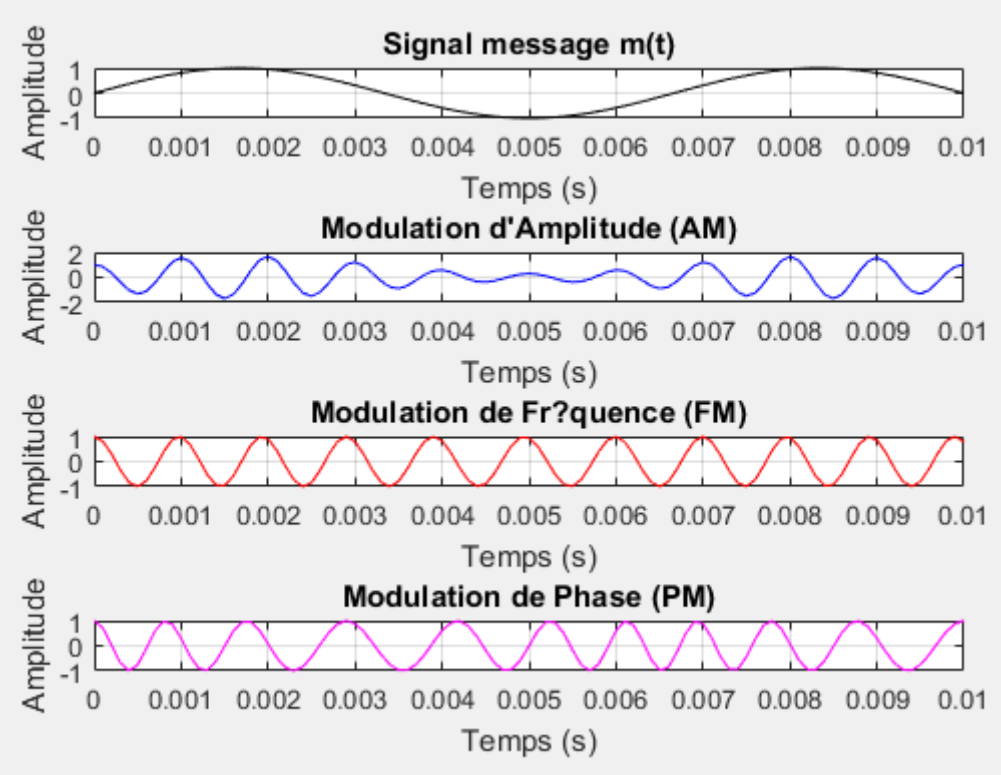
↳ **Code :**

⇨ **Le fichier : modulation_analogique.m**

```
clc; clear; close all;

% Signal d'information (message)
Fs = 10000;             % Fr?quence d'?chantillonnage
t = 0:1/Fs:0.01;        % Temps (10 ms)
fm = 150;               % Fr?quence du message (Hz)
Am = 1;                 % Amplitude du message
m = Am * sin(2*pi*fm*t);  % Signal message

% Porteuse
fc = 1000;              % Fr?quence de la porteuse (Hz)
```

```
Ac = 1;                 % Amplitude de la porteuse

% --- Modulation AM ---
ka = 0.7;               % Indice de modulation AM
AM = (1 + ka*m) .* cos(2*pi*fc*t);

% --- Modulation FM ---
kf = 50;                % Sensibilit? en fr?quence
FM = Ac * cos(2*pi*fc*t + 2*pi*kf*cumsum(m)/Fs);

% --- Modulation PM ---
kp = pi/2;              % Sensibilit? en phase
PM = Ac * cos(2*pi*fc*t + kp*m);

% --- Affichage ---
figure;
subplot(4,1,1);
plot(t,m,'k'); grid on;
title('Signal message m(t)');
xlabel('Temps (s)'); ylabel('Amplitude');

subplot(4,1,2);
plot(t,AM,'b'); grid on;
title('Modulation d"Amplitude (AM)');
xlabel('Temps (s)'); ylabel('Amplitude');

subplot(4,1,3);
plot(t,FM,'r'); grid on;
title('Modulation de Fr?quence (FM)');
xlabel('Temps (s)'); ylabel('Amplitude');

subplot(4,1,4);
plot(t,PM,'m'); grid on;
title('Modulation de Phase (PM)');
xlabel('Temps (s)'); ylabel('Amplitude');
```

- **Résultat :**



➢ Modulation numérique

FSK (Frequency Shift Keying) : chaque bit est représenté par une fréquence différente. Plus robuste que ASK face au bruit.

PSK (Phase Shift Keying) : la phase du signal change pour coder les bits. Très utilisée pour sa bonne efficacité spectrale.

ASK (Amplitude Shift Keying) : l'amplitude varie selon les bits (ex : 1 = fort, 0 = faible). Simple mais très sensible au bruit.

↳ Code :

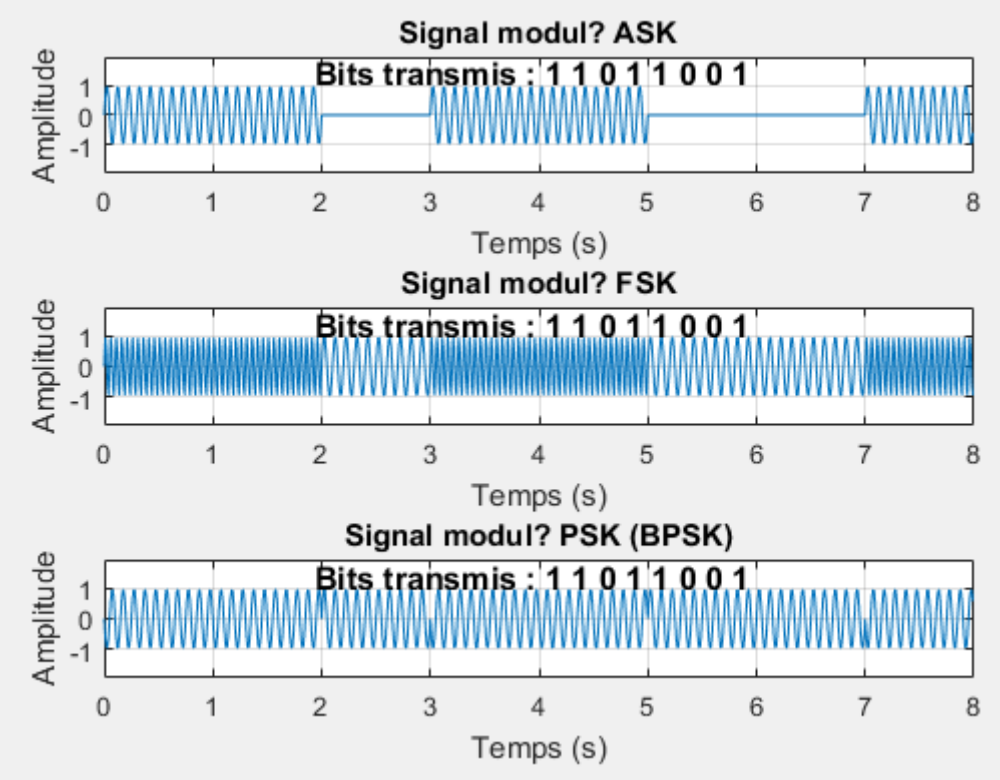
⇨ Le fichier : modulation_numerique.m

```
% Param?tres communs
Tb = 1;                % Dur?e d'un bit (1 seconde)
fs = 100;              % Fr?quence d'?chantillonnage (Hz)
t = 0:1/fs:Tb-1/fs;    % Vecteur temps pour 1 bit
fc = 10;               % Fr?quence porteuse pour ASK et PSK (Hz)
f1 = 10;               % Fr?quence pour bit 0 en FSK (Hz)
f2 = 20;               % Fr?quence pour bit 1 en FSK (Hz)
N = 8;                 % Nombre de bits ? transmettre

% G?n?ration al?atoire de bits (0 ou 1)
bits = randi([0 1], 1, N);

% Initialisation des signaux modul?s
ASK_signal = [];
FSK_signal = [];
PSK_signal = [];

for i = 1:N
```

```
    % Signal ASK
    carrier_ASK = sin(2*pi*fc*t);
    if bits(i) == 1
        mod_ASK = carrier_ASK;
    else
        mod_ASK = zeros(size(carrier_ASK));
    end
    ASK_signal = [ASK_signal mod_ASK];

    % Signal FSK
    if bits(i) == 0
        carrier_FSK = sin(2*pi*f1*t);
    else
        carrier_FSK = sin(2*pi*f2*t);
    end
    FSK_signal = [FSK_signal carrier_FSK];

    % Signal PSK (BPSK) : bit 0 = sin(2*pi*fc*t), bit 1 = -sin(2*pi*fc*t)
    if bits(i) == 0
        mod_PSK = sin(2*pi*fc*t);
    else
        mod_PSK = -sin(2*pi*fc*t);
    end
    PSK_signal = [PSK_signal mod_PSK];
end

% Pr?parer la cha?ne de bits ? afficher
bits_str = sprintf('%d ', bits);

% Vecteur temps total pour N bits
t_total = 0:1/fs:Tb*N-1/fs;

% Affichage dans 3 sous-graphes
figure;

% ASK
subplot(3,1,1);
plot(t_total, ASK_signal);
title('Signal modul? ASK');
xlabel('Temps (s)');
ylabel('Amplitude');
grid on;
x_pos = t_total(round(end/2));
y_pos = max(ASK_signal) + 0.5;
ylim([min(ASK_signal)-1, y_pos + 0.5]);
text(x_pos, y_pos, ['Bits transmis : ' bits_str], 'HorizontalAlignment', 'center', 'FontSize', 12, 'FontWeight',
'bold');

% FSK
subplot(3,1,2);
plot(t_total, FSK_signal);
```

```
title('Signal modul? FSK');
xlabel('Temps (s)');
ylabel('Amplitude');
grid on;
x_pos = t_total(round(end/2));
y_pos = max(FSK_signal) + 0.5;
ylim([min(FSK_signal)-1, y_pos + 0.5]);
text(x_pos, y_pos, ['Bits transmis : ' bits_str], 'HorizontalAlignment', 'center', 'FontSize', 12, 'FontWeight',
'bold');

% PSK
subplot(3,1,3);
plot(t_total, PSK_signal);
title('Signal modul? PSK (BPSK)');
xlabel('Temps (s)');
ylabel('Amplitude');
grid on;
x_pos = t_total(round(end/2));
y_pos = max(PSK_signal) + 0.5;
ylim([min(PSK_signal)-1, y_pos + 0.5]);
text(x_pos, y_pos, ['Bits transmis : ' bits_str], 'HorizontalAlignment', 'center', 'FontSize', 12, 'FontWeight',
'bold');
```

↳ **Résultat :**



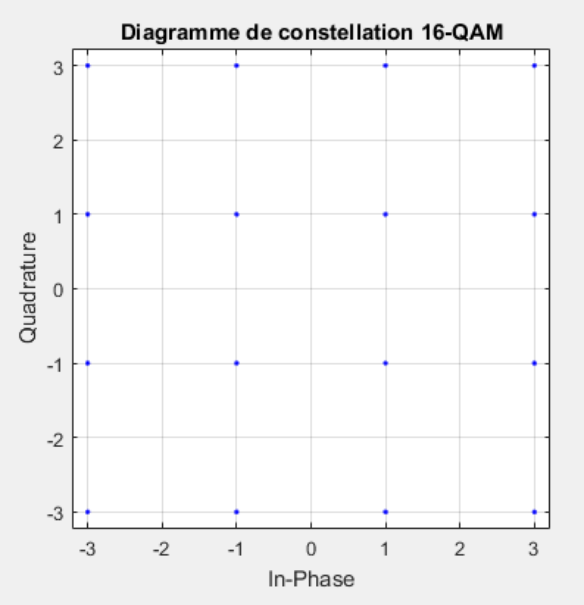
➢ La modulation M-QAM (Quadrature Amplitude Modulation)

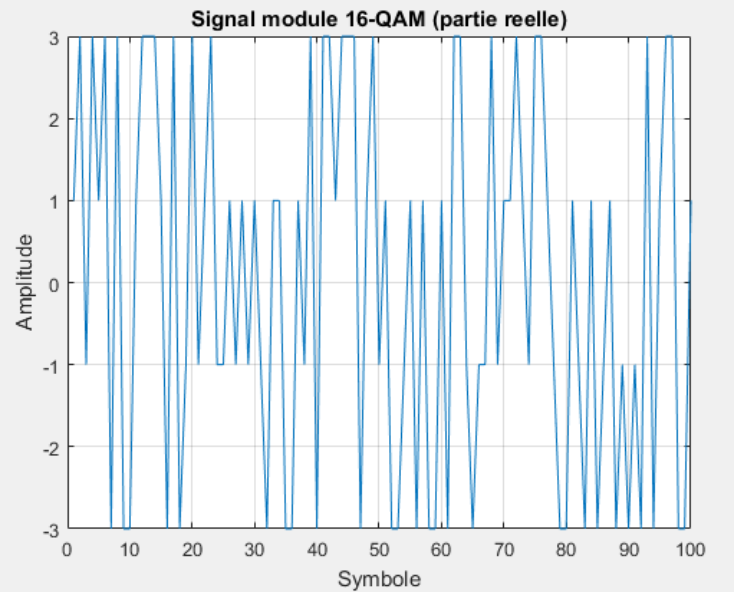
C'est une modulation combinée : elle utilise à la fois l'amplitude et la phase pour représenter les données. Plus la valeur de M (par exemple 16, 64, 256) est grande, plus on peut transmettre de bits par symbole, mais cela demande un meilleur SNR. Elle est très utilisée dans les réseaux modernes (Wi-Fi, 4G, 5G) pour son efficacité spectrale élevée.

- **Code :**
  - **Le fichier : M_QAM.m**

```
% Param?tres
M = 16;                % Ordre de la modulation QAM (16-QAM)
k = log2(M);           % Nombre de bits par symbole
N_sym = 100;           % Nombre de symboles ? transmettre

% G?n?ration al?atoire de bits
bits = randi([0 1], 1, N_sym * k);

% Mapping bits -> symboles (groupes de k bits)
symbols_dec = bi2de(reshape(bits, k, []).', 'left-msb');  % Conversion binaire vers d?cimal

% Cr?ation de l'objet modulateur QAM
qam_mod = comm.RectangularQAMModulator(M, 'BitInput', false);

% Modulation avec step (ou appel? comme une fonction si MATLAB r?cent)
modulated_symbols = step(qam_mod, symbols_dec);

% Affichage de la constellation
figure;
scatterplot(modulated_symbols);
title('Diagramme de constellation 16-QAM');
grid on;

% Affichage des bits transmis (une partie pour lisibilit?)
disp('Quelques bits transmis (par symboles) :');
disp(reshape(bits, k, []).');

% Affichage temporel (partie r?elle)
figure;
plot(real(modulated_symbols));
title('Signal module 16-QAM (partie reelle)');
xlabel('Symbole');
ylabel('Amplitude');
grid on;
```

- **Résultat :**





- **Interprétation :**

La 16-QAM combine la modulation d'amplitude sur deux axes orthogonaux (In-phase I et Quadrature Q) pour transmettre 4 bits par symbole. Le signal est modulé en regroupant les bits en blocs de 4, puis converti en symboles selon une table de constellation. Cette méthode offre un compromis entre débit élevé et robustesse face au bruit. Le diagramme de constellation montre les positions des symboles modulés dans le plan complexe.

➢ M-PSK (Phase Shift Keying)

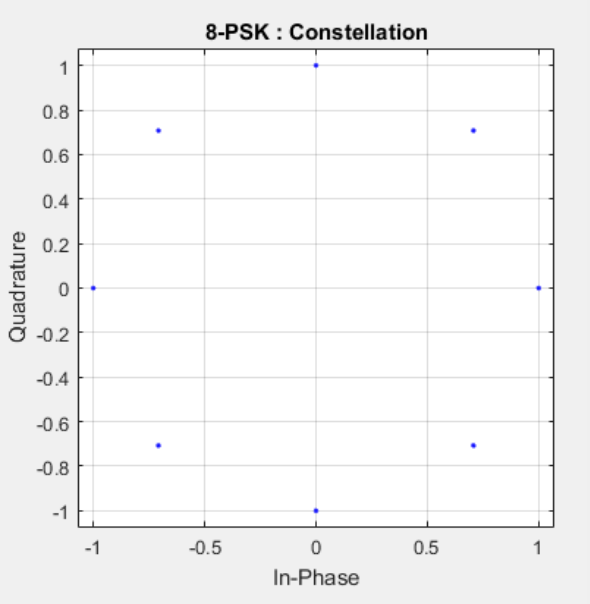
Est une modulation numérique où chaque symbole est représenté par une phase différente d'une porteuse sinusoïdale.

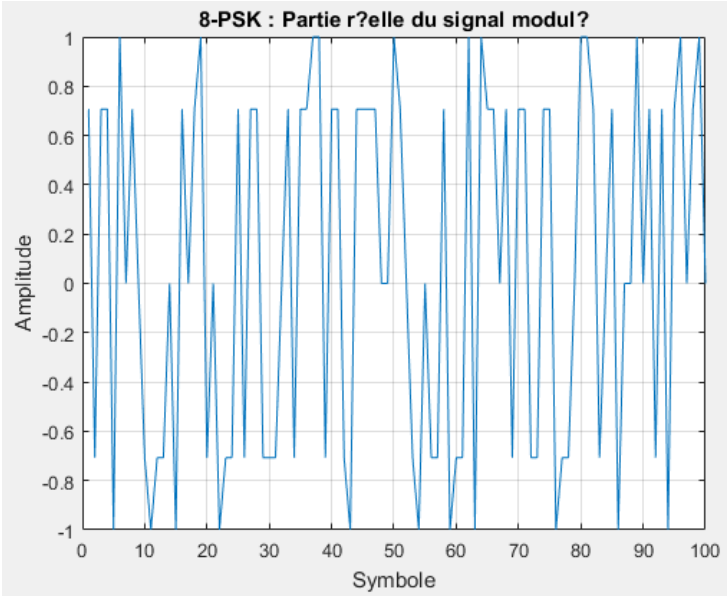
- **Code :**
  - **⇨ Le fichier : M_PSK.m**

```
% Param?tres
M = 8;                      % Ordre de modulation (8-PSK)
k = log2(M);                % Nombre de bits par symbole
N_sym = 100;                % Nombre de symboles

% G?n?ration al?atoire de bits
bits = randi([0 1], 1, N_sym * k);

% Regrouper les bits en symboles d?cimaux
symbols = bi2de(reshape(bits, k, []).', 'left-msb');  % symboles entre 0 et M-1

% Modulation PSK (phase offset = 0)
modulated_symbols = pskmod(symbols, M, 0);  % signal modul? complexe

% Affichage de la constellation
figure;
scatterplot(modulated_symbols);
title([num2str(M), '-PSK : Constellation']);
grid on;

% Affichage temporel (partie r?elle)
```

```
figure;
plot(real(modulated_symbols));
xlabel('Symbole');
ylabel('Amplitude');
title([num2str(M), '-PSK : Partie r?elle du signal modul?']);
grid on;
```

- **Résultat :**





- **Interprétation :**

Dans un **8-PSK**, 8 phases sont utilisées pour encoder 3 bits par symbole.

c. Modulation / démodulation M-QAM

Plus SNR (Eb/No) augmente, plus le BER diminue (meilleure qualité de réception).

Plus M est grand (ex : 1024-QAM), plus la courbe est dégradée :

Car la constellation est plus dense.

Les symboles sont plus proches → plus sensibles au bruit.

4-QAM (ou QPSK) offre la meilleure robustesse au bruit, mais baisse de débit.

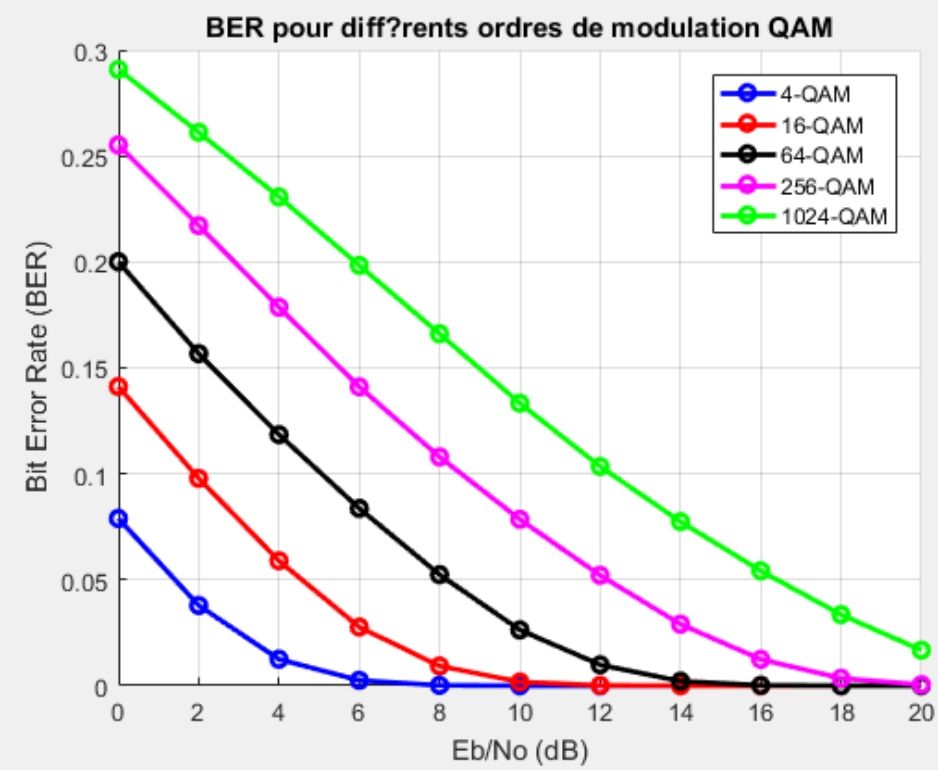
1024-QAM offre un débit très élevé, mais exige un canal très propre (fort Eb/No).

- **Code :**
  - **Le fichier : modulation_M_QAM.m**

```
% Param?tres
EbNo_dB_values = 0:2:20;                % Valeurs de Eb/No en dB
M_values = [4, 16, 64, 256, 1024];      % Ordres de modulation QAM
numBits = 1e6;                          % Nombre de bits simul?s
BER_results = zeros(length(M_values), length(EbNo_dB_values));  % Matrice des BER

for m_idx = 1:length(M_values)
    M = M_values(m_idx);
    k = log2(M);                        % Nombre de bits par symbole

    for e_idx = 1:length(EbNo_dB_values)
```

```
        EbNo_dB = EbNo_dB_values(e_idx);
        EbNo = 10^(EbNo_dB/10);
        EsNo = EbNo * k;

        % G?n?ration des bits al?atoires
        bits = randi([0 1], numBits, 1);

        % Adapter ? un multiple de k
        nBits = floor(numBits / k) * k;
        bits = bits(1:nBits);

        % Regrouper les bits en symboles entiers
        symbols = bi2de(reshape(bits, [], k), 'left-msb');

        % Modulation QAM
        tx = qammod(symbols, M, 'UnitAveragePower', true, 'InputType', 'integer');

        % Bruit AWGN
        noiseSigma = sqrt(1 / (2 * EsNo));
        noise = noiseSigma * (randn(size(tx)) + 1i * randn(size(tx)));
        rx = tx + noise;

        % D?modulation
        demod_symbols = qamdemod(rx, M, 'UnitAveragePower', true, 'OutputType', 'integer');
        received_bits = de2bi(demod_symbols, k, 'left-msb');
        received_bits = received_bits(:);

        % Calcul du BER
        numErrors = sum(bits ~= received_bits);
        BER = numErrors / nBits;
        BER_results(m_idx, e_idx) = BER;

        fprintf('M = %d, Eb/No = %d dB, BER = %.4e\n', M, EbNo_dB, BER);
    end
end

% Tracer les r?sultats
figure; hold on; grid on;
colors = ['b', 'r', 'k', 'm', 'g'];  % Couleurs diff?rentes

for m_idx = 1:length(M_values)
    semilogy(EbNo_dB_values, BER_results(m_idx, :), ['-' colors(m_idx) 'o'], 'LineWidth', 2);
end

xlabel('Eb/No (dB)');
ylabel('Bit Error Rate (BER)');
title('BER pour diff?rents ordres de modulation QAM');
legend(arrayfun(@(x) sprintf('%d-QAM', x), M_values, 'UniformOutput', false));
hold off;
```

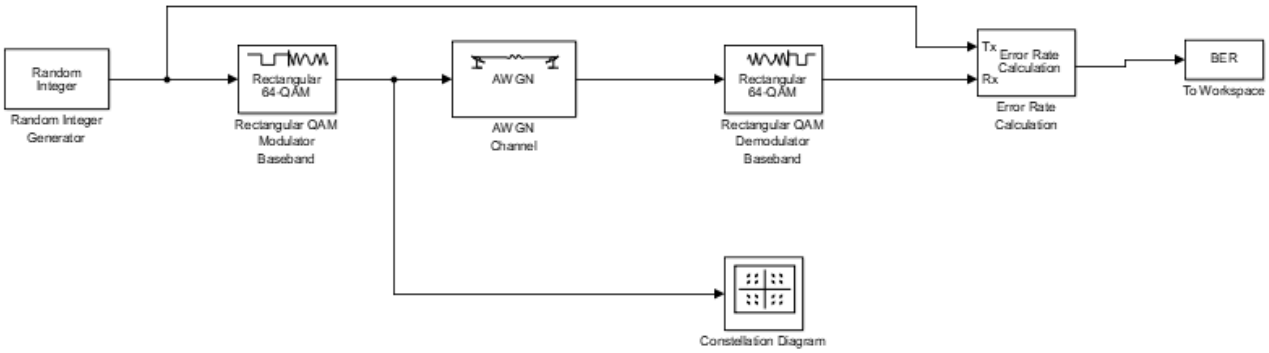
- **Résultat :**




```
Command Window
  M = 1024, Eb/No = 8 dB, BER = 1.6609e-01
  M = 1024, Eb/No = 10 dB, BER = 1.3333e-01
  M = 1024, Eb/No = 12 dB, BER = 1.0345e-01
  M = 1024, Eb/No = 14 dB, BER = 7.7445e-02
  M = 1024, Eb/No = 16 dB, BER = 5.4228e-02
  M = 1024, Eb/No = 18 dB, BER = 3.3557e-02
  M = 1024, Eb/No = 20 dB, BER = 1.6786e-02
```

⇨ **Le fichier : mod_demod_QAM.sxl**

⇨ **Le fichier : demod_mod_QAM.m**



d. Comparaison : Variation du BER en fonction du type de modulation

Ce code simule la transmission de bits modulés en QPSK, 16QAM et 64QAM sur un canal bruité (AWGN), calcule le taux d'erreur binaire (BER) pour différentes valeurs de (SNR), et trace la performance de chaque modulation. Il illustre que des modulations plus complexes offrent un débit plus élevé mais sont plus sensibles au bruit.

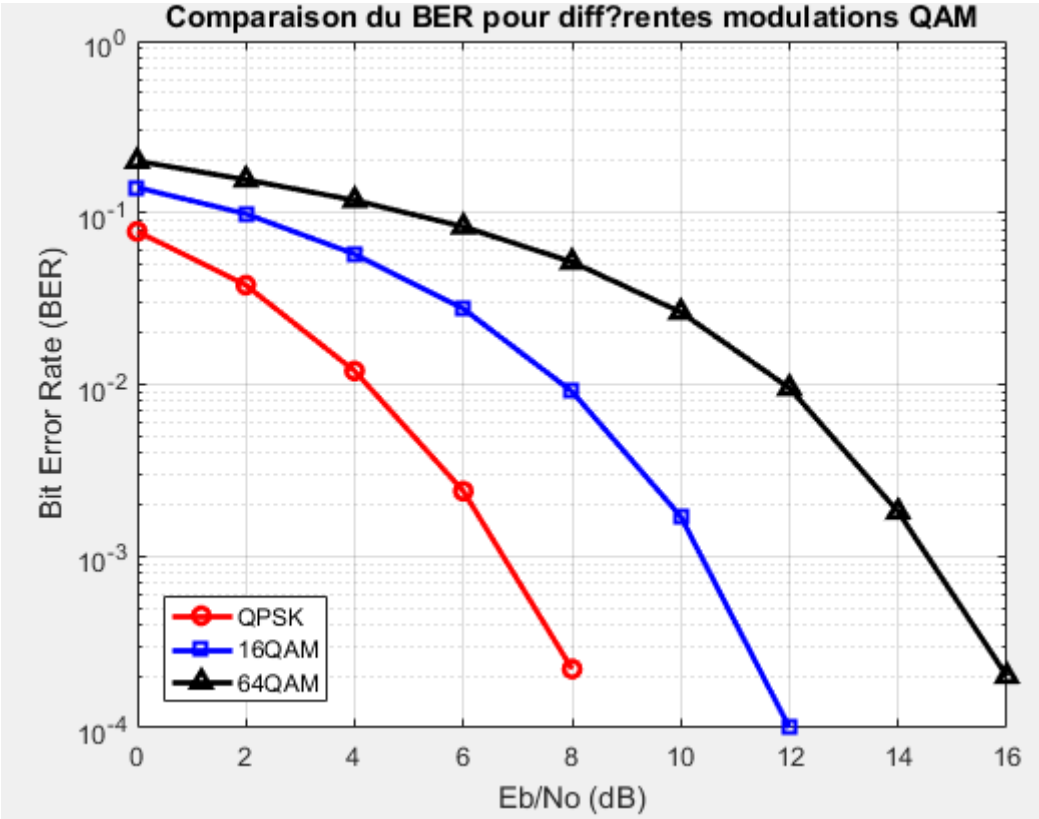
↳ **Code :**

⇨ **Le fichier : comparaison_BER.m**

```
clc; clear; close all;

% Param?tres
EbNo_dB_values = 0:2:20; % Valeurs de Eb/No en dB
M_values = [4, 16, 64]; % Ordres de modulation (QPSK, 16QAM, 64QAM)
modulations = {'QPSK', '16QAM', '64QAM'};

numBits = 1e5; % Nombre de bits ? transmettre
BER_results = zeros(length(M_values), length(EbNo_dB_values)); % Tableau de r?sultats

for m_idx = 1:length(M_values)
    M = M_values(m_idx);
    k = log2(M); % Bits par symbole

    for e_idx = 1:length(EbNo_dB_values)
        EbNo_dB = EbNo_dB_values(e_idx);
        EbNo = 10^(EbNo_dB/10); % Eb/No en valeur lin?aire

        % G?n?ration des bits al?atoires (ajust?s pour ?tre divisibles par k)
        numSymbols = floor(numBits / k);
        bits_tx = randi([0 1], numSymbols * k, 1);

        % Mapping des bits en symboles
        symbols_tx = qammod(bi2de(reshape(bits_tx, k, []).'), M, 'InputType', 'integer', 'UnitAveragePower',
true);

        % Ajout de bruit AWGN
        snr = EbNo_dB + 10*log10(k); % SNR par symbole
        symbols_rx = awgn(symbols_tx, snr, 'measured');

        % D?modulation
        bits_rx = de2bi(qamdemod(symbols_rx, M, 'OutputType', 'integer', 'UnitAveragePower', true), k).';
        bits_rx = bits_rx(:);

        % Calcul du BER
        numErrors = sum(bits_tx ~= bits_rx);
        BER = numErrors / length(bits_tx);
        BER_results(m_idx, e_idx) = BER;

        fprintf('%s, Eb/No=%d dB, BER=%.5e\n', modulations{m_idx}, EbNo_dB, BER);
    end
end

% Trac?
figure;
semilogy(EbNo_dB_values, BER_results(1,:), 'r-o', 'LineWidth', 2); hold on;
semilogy(EbNo_dB_values, BER_results(2,:), 'b-s', 'LineWidth', 2);
semilogy(EbNo_dB_values, BER_results(3,:), 'k-^', 'LineWidth', 2);
grid on;
xlabel('Eb/No (dB)');
ylabel('Bit Error Rate (BER)');
```

```
title('Comparaison du BER pour diff?rentes modulations QAM');
legend(modulations, 'Location', 'southwest');
```

↳ **Résultat :**

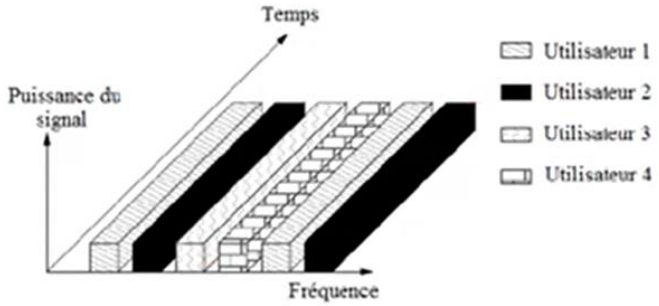


2. Techniques d'accès multiple

2.1. FDMA _ frequency division multiple access

La bande passante totale disponible est divisée en plusieurs sous-canaux fréquentiels distincts, chacun étant attribué à une paire communicante (émetteur-récepteur).

Cette technique est utilisée notamment dans les systèmes 1G et certains systèmes satellites.



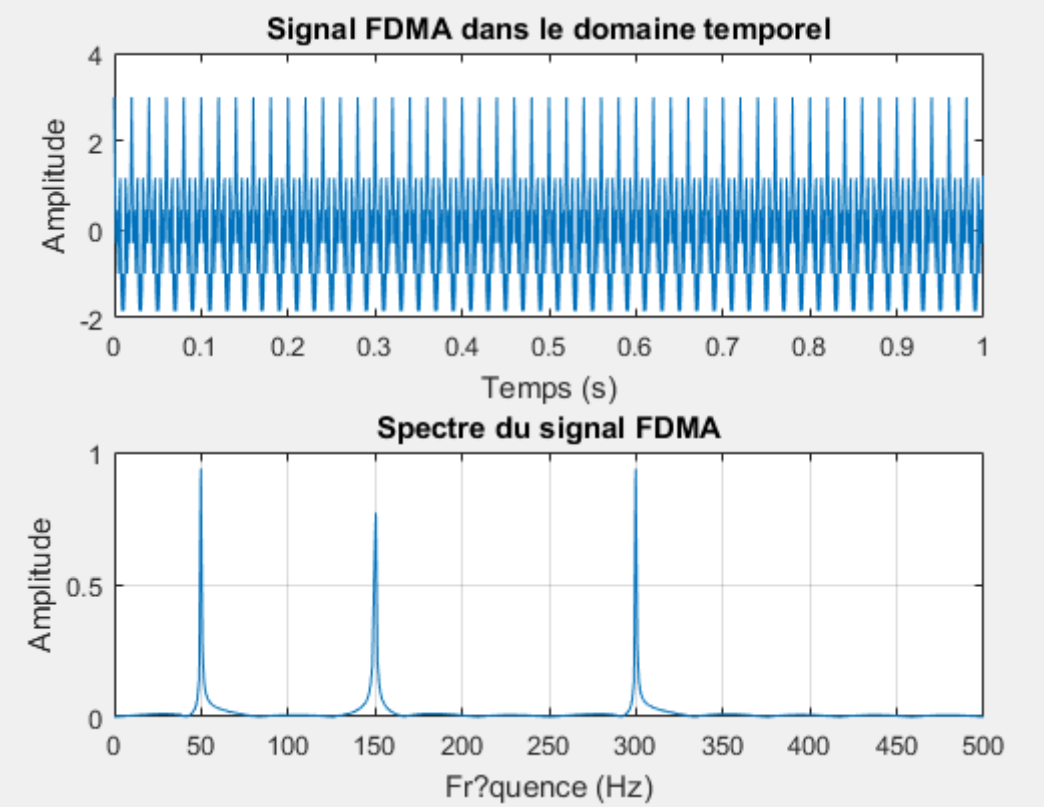
↳ **Code :**

⇨ **Le fichier : FDMA.m**

```
Fs = 1000; % Fr?quence d'?chantillonnage
t = 0:1/Fs:1-1/Fs; % Vecteur temps 1s

% Signaux de trois utilisateurs sur diff?rentes fr?quences porteuses
f1 = 50;  % utilisateur 1
f2 = 150; % utilisateur 2
f3 = 300; % utilisateur 3
```

```
signal1 = cos(2*pi*f1*t);
signal2 = cos(2*pi*f2*t);
signal3 = cos(2*pi*f3*t);

% Signal multiplex? FDMA (somme des signaux)
signal_FDMA = signal1 + signal2 + signal3;

% Spectre du signal multiplex?
figure;
subplot(2,1,1);
plot(t, signal_FDMA);
title('Signal FDMA dans le domaine temporel');
xlabel('Temps (s)');
ylabel('Amplitude');

subplot(2,1,2);
NFFT = 2^nextpow2(length(signal_FDMA));
freq_axis = Fs/2*linspace(0,1,NFFT/2+1);
signal_FDMA_fft = fft(signal_FDMA, NFFT)/length(signal_FDMA);

plot(freq_axis, 2*abs(signal_FDMA_fft(1:NFFT/2+1)));
title('Spectre du signal FDMA');
xlabel('Fr?quence (Hz)');
ylabel('Amplitude');
grid on;
```

- **Résultat :**

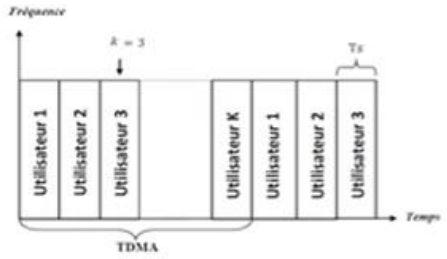


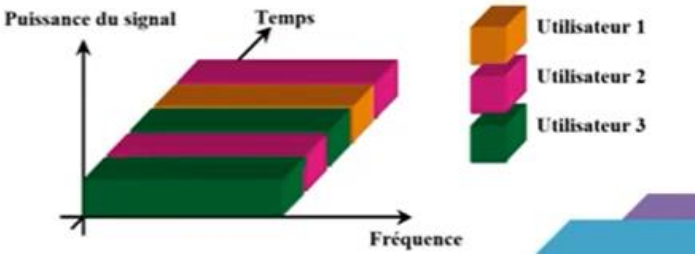
- **Interprétation :**

Signaux cosinus à des fréquences distinctes (50 Hz, 150 Hz, 300 Hz) et affiche leur superposition dans le domaine temporel ainsi que leur séparation claire dans le spectre fréquentiel.

2.2. TDMA _ time division multiple access

Plusieurs utilisateurs partagent la même fréquence porteuse, mais chacun communique à un instant précis dans un créneau temporel (time slot). Les utilisateurs transmettent leurs données à tour de rôle dans une séquence périodique.





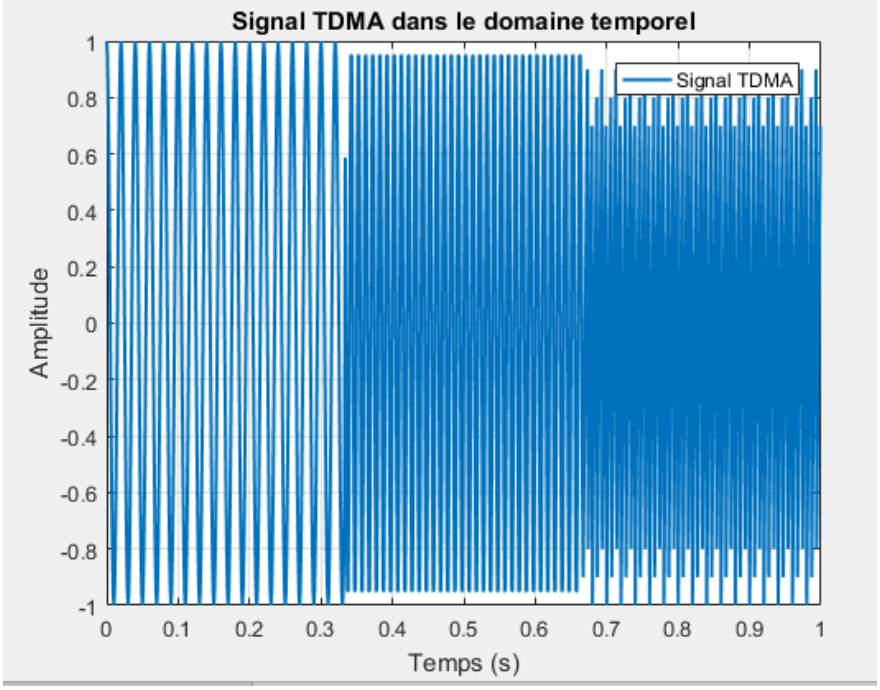
↳ **Code :**

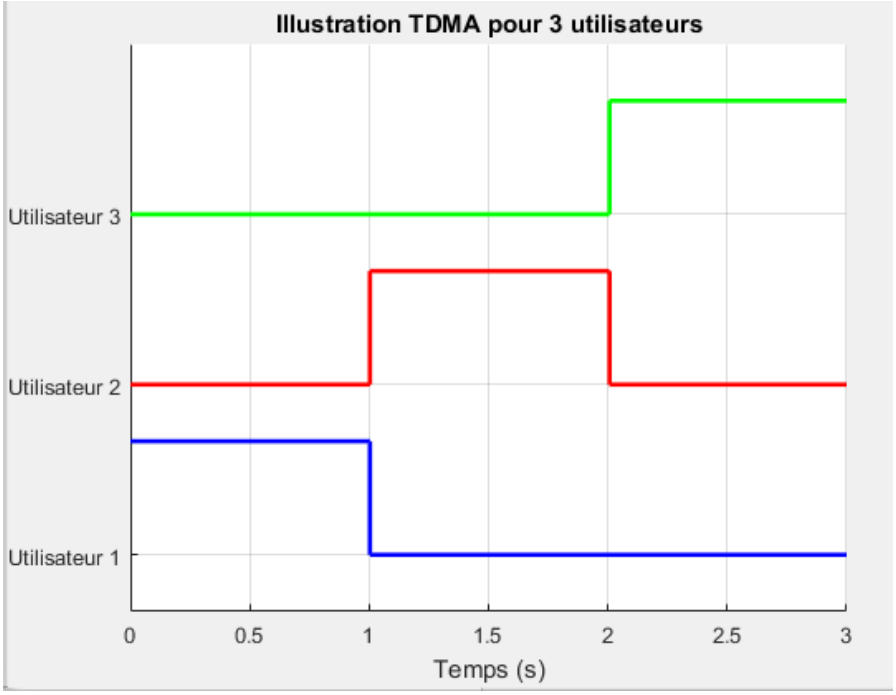
⇨ **Le fichier : TDMA.m**

```
clc; clear; close all;

%% === Premi?re figure : Signal TDMA combin? (cos/sin/triangle) ===
Fs = 1000;              % Fr?quence d'?chantillonnage (Hz)
T = 1;                  % Dur?e totale d'une trame (1 seconde)
t = 0:1/Fs:T-1/Fs;      % Vecteur temps

N_users = 3;                    % Nombre d'utilisateurs
slot_duration = T / N_users;    % Dur?e de chaque slot temporel

% Initialiser le signal TDMA
signal_TDMA = zeros(size(t));

% D?finir les signaux des utilisateurs
user_signals = {
    @(t) cos(2*pi*50*t);          % Utilisateur 1
    @(t) sin(2*pi*100*t);         % Utilisateur 2
    @(t) sawtooth(2*pi*150*t);    % Utilisateur 3
};

% Cr?ation du signal TDMA combin?
for user = 1:N_users
    slot_start = (user-1)*slot_duration;
    slot_end = user*slot_duration;

    slot_idx = (t >= slot_start) & (t < slot_end);
    signal_TDMA(slot_idx) = user_signals{user}(t(slot_idx));
end

% Affichage du signal combin?
figure;
plot(t, signal_TDMA, 'LineWidth', 1.5);
title('Signal TDMA dans le domaine temporel');
```

```
xlabel('Temps (s)');
ylabel('Amplitude');
grid on;
legend('Signal TDMA');

%% === Deuxi?me figure : Structure TDMA (slots carr?s) ===
N_users = 3;          % Nombre d'utilisateurs
T_slot = 1;           % Dur?e d'un slot temporel (en secondes)
N_slots = 3;          % Nombre total de slots
fs = 100;             % Fr?quence d'?chantillonnage
T_total = N_slots * T_slot;

% Vecteur temps
t = linspace(0, T_total, T_total*fs);

% Initialiser les signaux utilisateurs
signals = zeros(N_users, length(t));

% G?n?rer les signaux TDMA par utilisateur
for user = 1:N_users
    idx_start = round((user-1)*T_slot*fs) + 1;
    idx_end = round(user*T_slot*fs);
    signals(user, idx_start:idx_end) = 1;
end

% Affichage structure TDMA
figure;
hold on;
colors = ['b', 'r', 'g'];

for user = 1:N_users
    stairs(t, signals(user,:) + (user-1)*1.5, 'Color', colors(user), 'LineWidth', 2);
end

set(gca, 'YTick', 0:1.5:(N_users-1)*1.5);
set(gca, 'YTickLabel', arrayfun(@(x) sprintf('Utilisateur %d', x), 1:N_users, 'UniformOutput', false));
xlabel('Temps (s)');
title('Illustration TDMA pour 3 utilisateurs');
grid on;
ylim([-0.5, N_users*1.5]);
hold off;
```

↳ **Résultat :**



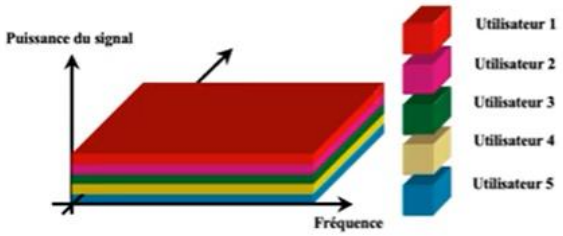


↳ **Interprétation :**

La figure 1 montre plusieurs utilisateurs partagent la même fréquence mais transmettent chacun à leur tour dans des intervalles de temps distincts (time slots), évitant ainsi les interférences et la figure 2 montre la répartition des slots temporels de manière schématique.

2.3. CDMA _ code division multiple access

Permet à plusieurs utilisateurs de partager la même fréquence et de transmettre en même temps. Chaque utilisateur est identifié par un code unique orthogonal. Ainsi, les signaux sont mélangés lors de la transmission, mais peuvent être séparés sans interférence à la réception grâce à l'orthogonalité des codes.



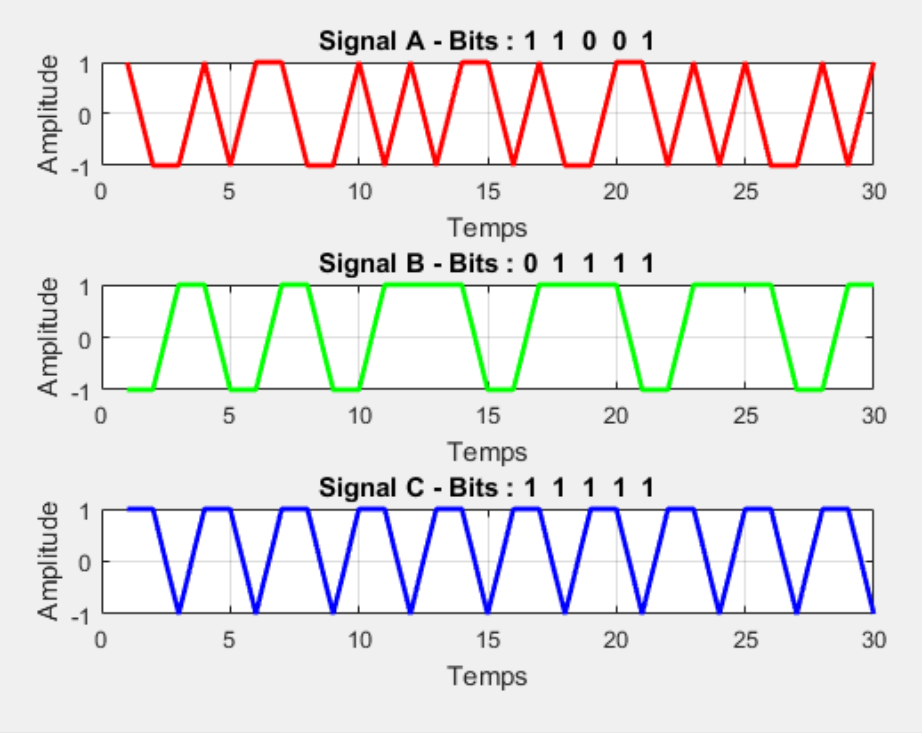
- **Code :**
- **Le fichier : CDMA.m**

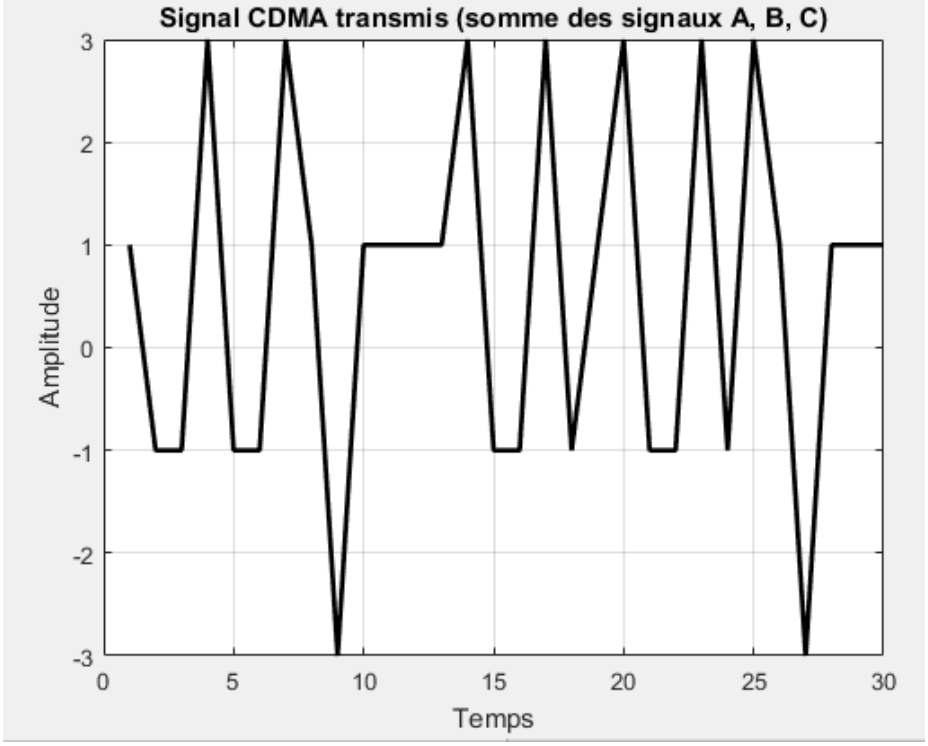
```
clc;
clear all;
close all;

%% 1. Param?tres
Nb_bits = 5;  % nombre de bits par utilisateur
chip_length = 6;

%% 2. Donn?es binaires al?atoires pour A, B, C
data_A = randi([0 1], 1, Nb_bits);
data_B = randi([0 1], 1, Nb_bits);
data_C = randi([0 1], 1, Nb_bits);

%% 3. Codes d??talement orthogonaux (Walsh codes simples)
a_code = [1 -1 -1 1 -1 1];
b_code = [1 1 -1 -1 1 1];
c_code = [1 1 -1 1 1 -1];

%% 4. Codage des bits : bit=1 ? code, bit=0 ? -code
signal_A = [];
signal_B = [];
signal_C = [];

for i = 1:Nb_bits
    if data_A(i) == 1
        signal_A = [signal_A a_code];
    else
        signal_A = [signal_A -a_code];
    end

    if data_B(i) == 1
        signal_B = [signal_B b_code];
    else
        signal_B = [signal_B -b_code];
    end

    if data_C(i) == 1
        signal_C = [signal_C c_code];
    else
        signal_C = [signal_C -c_code];
    end
```

```
end

%% 5. Superposition du signal total (principe du CDMA)
signal_total = signal_A + signal_B + signal_C;

%% 6. D?codage par produit scalaire (correlation avec chaque code)
decoded_A = [];
decoded_B = [];
decoded_C = [];

for i = 1:chip_length:length(signal_total)
    segment = signal_total(i:i+chip_length-1);

    res_A = dot(segment, a_code);
    res_B = dot(segment, b_code);
    res_C = dot(segment, c_code);

    decoded_A = [decoded_A (res_A > 0)];
    decoded_B = [decoded_B (res_B > 0)];
    decoded_C = [decoded_C (res_C > 0)];
end

%% 7. Trac? du signal transmis
figure;
plot(1:length(signal_total), signal_total, 'k', 'LineWidth', 2);
title('Signal CDMA transmis (somme des signaux A, B, C)');
xlabel('Temps');
ylabel('Amplitude');
grid on;

%% 8. Figure 2 : signaux individuels (avant sommation)
t_A = 1:length(signal_A);
t_B = 1:length(signal_B);
t_C = 1:length(signal_C);

figure;

subplot(3,1,1);
plot(t_A, signal_A, 'r', 'LineWidth', 2);
title(['Signal A - Bits : ', num2str(data_A)]);
xlabel('Temps');
ylabel('Amplitude');
grid on;

subplot(3,1,2);
plot(t_B, signal_B, 'g', 'LineWidth', 2);
title(['Signal B - Bits : ', num2str(data_B)]);
xlabel('Temps');
ylabel('Amplitude');
grid on;
```

```
subplot(3,1,3);
plot(t_C, signal_C, 'b', 'LineWidth', 2);
title(['Signal C - Bits : ', num2str(data_C)]);
xlabel('Temps');
ylabel('Amplitude');
grid on;

%% 9. Affichage des r?sultats de d?codage
disp('Bits transmis A :'); disp(data_A);
disp('Bits d?cod?s A  :'); disp(decoded_A);

disp('Bits transmis B :'); disp(data_B);
disp('Bits d?cod?s B  :'); disp(decoded_B);

disp('Bits transmis C :'); disp(data_C);
disp('Bits d?cod?s C  :'); disp(decoded_C);
```

- **Résultat :**





- **Interprétation :**

Plusieurs utilisateurs peuvent partager un même canal simultanément en utilisant des codes orthogonaux pour distinguer leurs transmissions. La corrélation avec ces codes permet ensuite d'extraire les données de chaque utilisateur indépendamment des autres, démontrant ainsi la robustesse du CDMA face aux interférences entre utilisateurs.

Conclusion

Les simulations effectuées sous MATLAB nous ont permis d'observer de manière concrète les effets des différentes techniques de modulation et d'accès multiple sur la transmission des données. La modulation joue un rôle clé dans l'adaptation des signaux aux canaux de communication, tandis que les techniques d'accès permettent le partage efficace de la ressource spectrale entre plusieurs utilisateurs. Les résultats montrent que le choix de la modulation et de la méthode d'accès dépend fortement des contraintes du système (bande passante, bruit, complexité, nombre d'utilisateurs). Ces expérimentations renforcent notre compréhension théorique et soulignent l'importance de la simulation dans l'analyse et la conception des systèmes de communication.