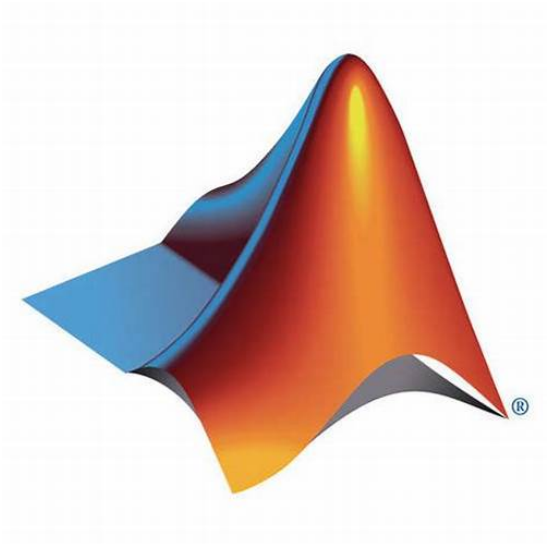
المدرسة الوطنية للعلوم التطبيقية بتطوان
ⵜⵉⵍⵍⵉⵏ ⵜⴰⵏⴰⵎⵓⵔⵜ ⵏ ⵜⵙⵙⵏⵉⵡⵉⵏ ⵜⵉⵜⴰⵡⵉⵏ
Ecole Nationale des Sciences Appliquées de Tétouan

École Nationale des Sciences Appliquées Tétouan

Filière : Cybersecurité et Systèmes Embarqués , GCSE

DSP : Audio Stéganographie

Projet DSP



Rédigé par :

BOUSSETA HATIM

EL OUARDIJI MAROUANE

Encadré par : Pr. ERRACHIDI ZAKARIA

Table des matières

0.1 Introduction

La stéganographie audio est une technique fascinante de dissimulation d'information qui consiste à cacher des données secrètes dans des fichiers sonores sans altération perceptible. Contrairement à la cryptographie qui protège le contenu des messages, la stéganographie se concentre sur la dissimulation de leur existence même. Cette méthode trouve des applications importantes en sécurité de l'information, notamment pour la communication confidentielle et la protection des droits d'auteur.

Dans le cadre de la digital forensics, la stéganographie audio présente un double intérêt : d'une part, elle constitue un moyen sophistiqué d'exfiltration de données que les experts doivent savoir détecter et analyser ; d'autre part, elle permet d'insérer discrètement des informations de traçabilité ou des preuves numériques dans des fichiers audio, facilitant ainsi les enquêtes numériques. Notre projet s'inscrit donc dans cette dynamique, en étudiant et mettant en œuvre une technique de dissimulation par écho, alliant traitement du signal numérique et enjeux liés à la cyber-sécurité et à la criminalistique numérique.

0.1.1 Relation DSP/Audio Stéganographie

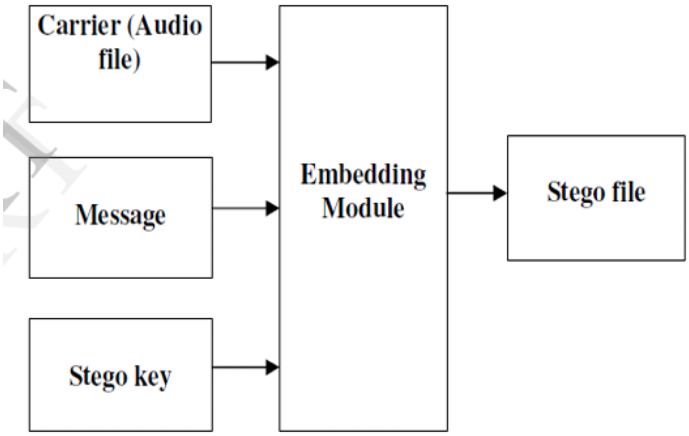
-DSP fournit les outils mathématiques essentiels à la stéganographie audio. Par exemple : l'échantillonnage et la transformée de Fourier, il permet de manipuler les signaux audio dans les domaines temporel et fréquentiel. Les quatres techniques exploitent différemment ces principes : (1) le *LSB Coding* agit directement sur les échantillons temporels en modifiant les bits les moins significatifs, (2) le *Phase Coding* utilise la représentation fréquentielle (TFD) pour altérer les phases de manière imperceptible, tandis que (3) le *Spread Spectrum* applique des modulations pseudo-aléatoires inspirées des communications numériques. Le DSP garantit ainsi à la fois l'efficacité du masquage et la préservation de la qualité audio, tout en permettant des compromis calculables entre capacité, robustesse et imperceptibilité.

0.1.2 Contexte et Enjeux

Dans un monde numérique où la sécurité des communications est cruciale, cette technique trouve des applications variées :

— Marquage numérique de contenus audio
— Communication sécurisée militaire
— Protection des droits d'auteur
— Transmission de données sensibles

0.1.3 Méthodologie :



- **Fichier Audio Porteur (Carrier)**
  - Le fichier audio original qui servira à cacher le message (par exemple un fichier .wav ou .mp3)
  - Doit paraître normal pour ne pas éveiller les soupçons
- **Message à Cacher**
  - Les données secrètes que vous souhaitez dissimuler (texte, image, autre fichier)
  - Souvent compressé et/ou chiffré avant l'insertion
- **Clé Stégo (Stego Key)**
  - Un paramètre optionnel qui contrôle le processus d'insertion
  - Peut spécifier l'emplacement, la méthode de masquage ou servir de mot de passe
- **Module d'Insertion (Embedding Module)**
  - Le cœur du système qui implémente l'algorithme de stéganographie
  - Modifie subtilement le signal audio pour y incorporer le message
- **Fichier Stego Résultant**
  - Le fichier audio contenant le message caché
  - Doit rester perceptuellement similaire à l'original

0.1.4 Techniques Principales

Trois méthodes principales seront explorées dans ce projet :

1. Phase Coding

Cette technique exploite la faible sensibilité de l'oreille humaine aux variations de phase dans les signaux audio. En modifiant judicieusement les composantes phase du spectre fréquentiel, on peut encoder des informations sans affecter notablement la qualité sonore.

2. LSB Coding (Least Significant Bit)

Méthode populaire qui consiste à remplacer les bits les moins significatifs des échantillons audio par des bits de message secret. Bien que simple à implémenter, cette approche présente une vulnérabilité accrue aux traitements audio de base.

3. Spread Spectrum

Inspirée des techniques militaires, cette méthode répartit l'énergie du signal secret sur une large bande fréquentielle selon une séquence pseudo-aléatoire, offrant une robustesse améliorée contre les perturbations extérieures.

4. Echo Hiding

Cette méthode repose sur l'insertion contrôlée d'échos imperceptibles dans le signal audio. En variant des paramètres tels que le délai, l'amplitude et l'atténuation de l'écho, il est possible de coder des bits d'information. Comme l'écho peut être dissimulé sans altérer la qualité perçue, cette technique permet un bon compromis entre transparence et robustesse.

0.2 Objectifs du projet

Dans ce projet de stéganographie audio basée sur le traitement du signal numérique (DSP), nous allons explorer deux aspects essentiels :

- **encodage** : comment cacher un message secret dans un signal audio en utilisant des techniques de traitement du signal.
- **Decodage , La détection et l'extraction** : comment détecter et extraire le message caché à partir du signal audio modifié.

L'ensemble des simulations et des expérimentations seront réalisées à l'aide du logiciel MATLAB, qui permet une analyse et une manipulation efficace des signaux numériques.

Ce projet vise à implémenter et comparer ces trois techniques selon différents critères :

- Capacité de charge utile (bits/s)
- Robustesse aux attaques
- Imperceptibilité audio
- Complexité algorithmique

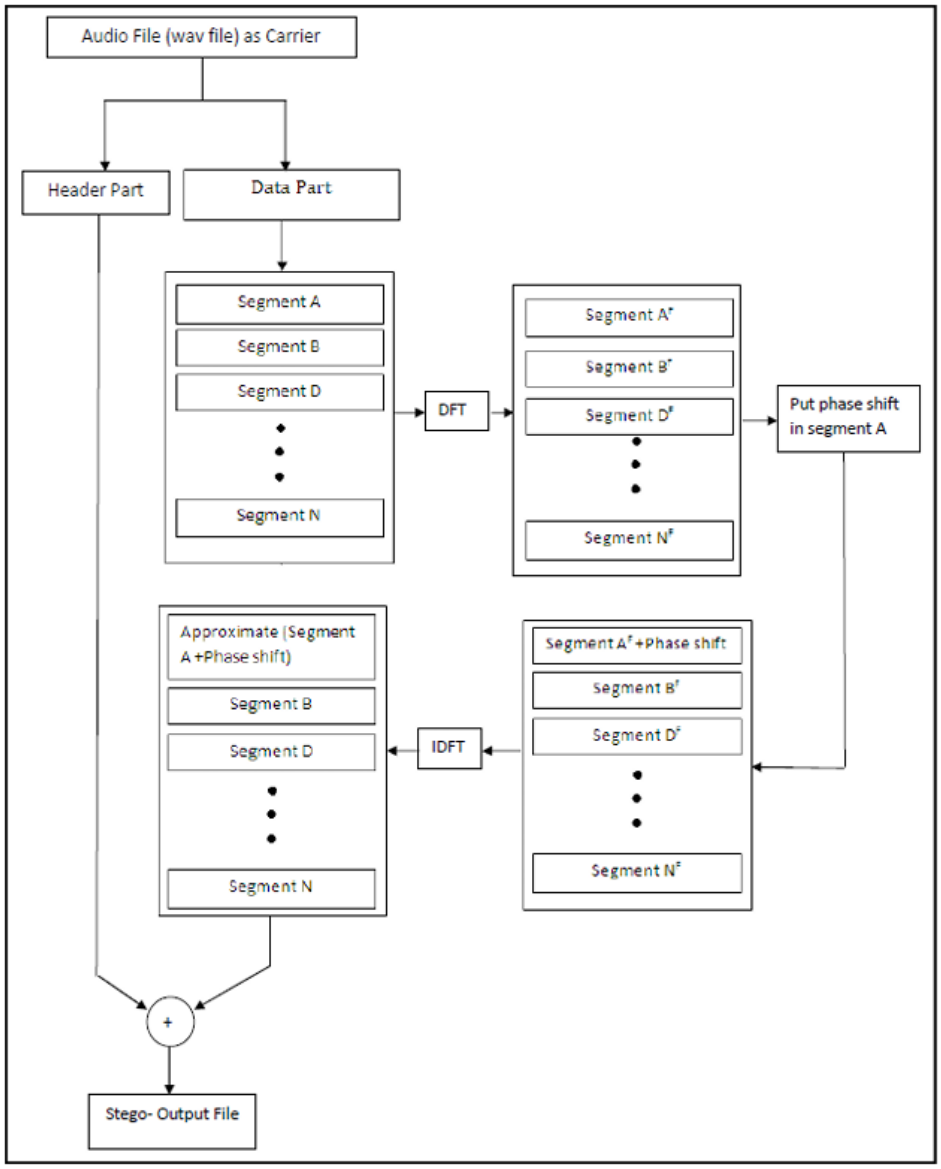
Les développements seront réalisés en MATLAB avec des outils de traitement du signal
.

0.3 Stéganographie par phase coding (codage par phase)

Le fichier audio au format `wav` utilisé comme support est d'abord séparé en deux parties distinctes : l'en-tête et les données audio. Après traitement de la partie données, ces deux segments sont recombinés pour former le fichier `wav` codé.

La partie données est segmentée en blocs temporels, sur lesquels on applique la Transformée de Fourier discrète (TFD) définie par :

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j\frac{2\pi}{N}kn} \quad pour k = 0, 1, \ldots, N-1 \tag{1}$$



Un décalage de phase est appliqué sur le premier segment, le message encodé étant converti en bits comme suit :

$$\text{New Phase} = \begin{cases} \text{Old Phase} + \pi/2 & \text{if message bit} = 0 \\ \text{Old Phase} - \pi/2 & \text{if message bit} = 1 \end{cases}$$

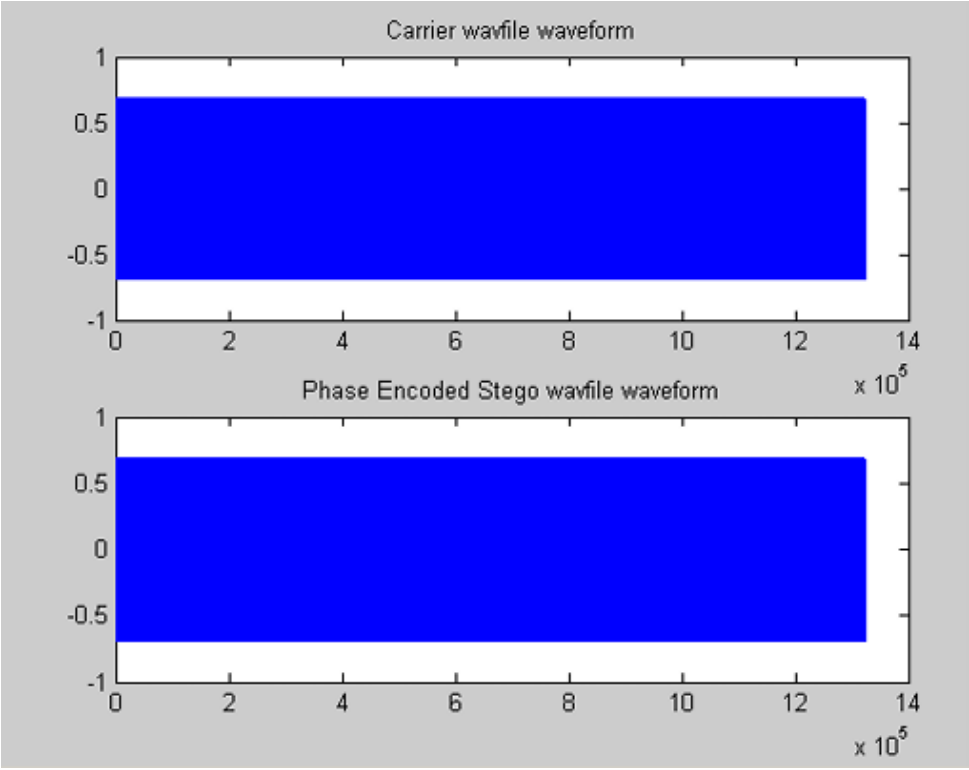
Les données audio sont récupérées en appliquant la transformée de Fourier inverse sur le tableau modifié de nombres complexes, donnée par l'équation suivante :

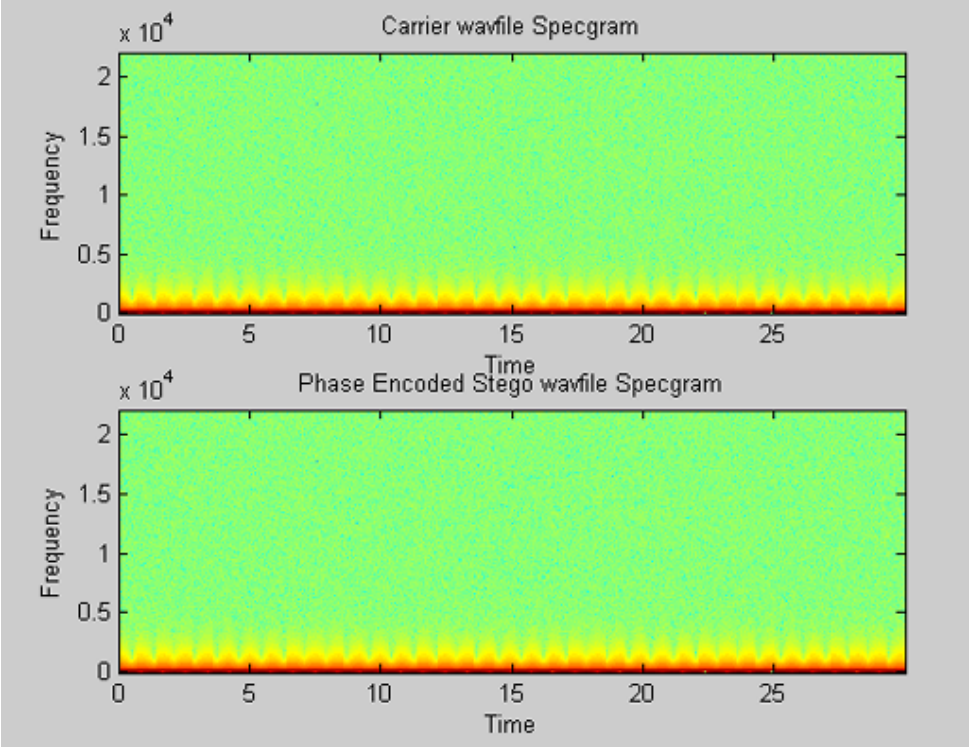
$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \cdot e^{j \frac{2\pi}{N} kn}$$

où $x[n]$ représente le signal temporel reconstruit, $X[k]$ les coefficients complexes modifiés, N la taille de la fenêtre, et $j = \sqrt{-1}$.

Le fichier décodé est ensuite soumis à la transformée de Fourier, et les différences de phase sont extraites à partir du premier segment de données afin de récupérer le message secret.

Ce type de codage pour la stéganographie est préféré par rapport à l'algorithme LSB, où le signal porteur est partiellement corrompu à cause du bruit introduit lors de la modification du bit de poids faible.





0.3.1 Explication des fonctions MATLAB pour le codage et décodage

phase_encode.m | phase_decode.m | lsb_encode.m | lsb_decode.m | echo_encode.m | echo_decode.m | dsss_encode

```m
% Encoding function (modified to work with GUI)
function outputPath = phase_encode(pathToAudio, stringToEncode)
    % Read audio file
    [audioData1, rate] = audioread(pathToAudio);

    % Pad message with '~' to ensure uniform length
    stringToEncode = pad(stringToEncode, 100, 'right', '~');

    % Calculate text length and chunk size
    textLength = 8 * length(stringToEncode);
    chunkSize = int32(2 * 2^ceil(log2(2 * textLength)));
    numberOfChunks = int32(ceil(size(audioData1, 1) / chunkSize));

    % Copy and resize audio data
    audioData = audioData1;
    if size(audioData1, 2) == 1
        audioData(numberOfChunks * chunkSize, 1) = 0;
        audioData = audioData';
    else
        audioData(numberOfChunks * chunkSize, size(audioData, 2)) = 0;
        audioData = audioData';
    end

    % Reshape audio into chunks
    chunks = reshape(audioData(1, :), chunkSize, [])';

    % Apply FFT
    chunks = fft(chunks, [], 2);
    magnitudes = abs(chunks);
    phases = angle(chunks);
    phaseDiff = diff(phases, 1, 1);

    % Convert message to binary
    textInBinary = [];
    for i = 1:length(stringToEncode)
        binStr = dec2bin(uint8(stringToEncode(i)), 8);
        textInBinary = [textInBinary str2num(binStr')'];
    end

    % Convert binary to phase differences
    textInPi = double(textInBinary);
    textInPi(textInPi == 0) = -1;
    textInPi = textInPi * -pi/2;

    % Find middle of chunk
    midChunk = chunkSize/2;

    % Embed message in phase differences
    phases(1, midChunk-textLength+1:midChunk) = textInPi;
    phases(1, midChunk+2:midChunk+1+textLength) = -fliplr(textInPi);

    % Recompute phase matrix
    for i = 2:size(phases, 1)
        phases(i, :) = phases(i-1, :) + phaseDiff(i-1, :);
    end

    % Apply inverse Fourier transform
    chunks = magnitudes .* exp(1j * phases);
    chunks = real(ifft(chunks, [], 2));

    % Reconstruct audio data
    audioData(1, :) = reshape(chunks', 1, []);

    % Save encoded audio
    outputPath = [tempname '.wav'];
    audiowrite(outputPath, audioData', rate);
end
```

FIGURE 1 – Code MATLAB pour l'encodage (phase) : insertion d'un message dans un signal audio.

phase_encode.m | phase_decode.m | lsb_encode.m | lsb_decode.m | echo_encode.m | echo_decode.m | dsss_encode.m

```m
% Decoding function (modified to work with GUI)
function decodedMessage = phase_decode(audioLocation)
    % Read encoded audio
    [audioData, ~] = audioread(audioLocation);

    textLength = 800; % 100 characters * 8 bits
    blockLength = 2 * int32(2^ceil(log2(2 * textLength)));
    blockMid = blockLength/2;

    % Extract phase information
    if size(audioData, 2) == 1
        code = audioData(1:blockLength);
    else
        code = audioData(1:blockLength, 1);
    end

    % Get phases and convert to binary
    codePhases = angle(fft(code));
    codePhases = codePhases(blockMid-textLength+1:blockMid);
    codeInBinary = double(codePhases < 0);

    % Convert binary to characters
    decodedMessage = '';
    powers = 2.^(7:-1:0)';

    for i = 1:8:length(codeInBinary)
        if i+7 <= length(codeInBinary)
            binGroup = codeInBinary(i:i+7);
            decValue = sum(binGroup .* powers);
            if decValue > 0 && decValue <= 127
                decodedMessage = [decodedMessage char(decValue)];
            end
        end
    end

    % Remove padding
    decodedMessage = strrep(decodedMessage, '~', '');
end
```

FIGURE 2 – Code MATLAB pour le décodage (phase) : extraction du message à partir du signal audio modifié.

Fonction de décodage

La fonction `phase_decode` a pour rôle d'extraire le message secret depuis un fichier audio encodé. Son fonctionnement se déroule en plusieurs étapes :

— Lecture des données audio à partir du fichier encodé.
— Sélection d'un segment initial du signal correspondant à la longueur du message codé.
— Application de la transformée de Fourier sur ce segment pour obtenir les phases des coefficients complexes.
— Extraction des phases correspondant aux bits du message, puis conversion de ces phases en valeurs binaires selon leur signe.
— Regroupement des bits par paquets de 8 pour reconstituer les caractères ASCII du message.
— Suppression des caractères de bourrage afin d'obtenir le message original clair.

Fonction de codage

La fonction `phase_encode` permet d'insérer un message dans un fichier audio porteur selon la technique de codage par décalage de phase. Le processus comprend :

— Lecture du fichier audio source.
— Ajout de caractères de bourrage au message pour obtenir une longueur fixe.
— Segmentation du signal audio en blocs adaptés à la taille du message.
— Application de la transformée de Fourier sur chaque bloc pour obtenir modules et phases.
— Conversion du message en binaire, puis en décalages de phase proportionnels.
— Insertion symétrique de ces décalages dans les différences de phase du premier segment pour garantir la cohérence spectrale.
— Reconstruction des phases complètes à partir des différences modifiées.
— Application de la transformée de Fourier inverse pour obtenir un signal audio modifié.
— Sauvegarde du nouveau fichier audio encodé contenant le message caché.

0.3.2 Avantages de la technique

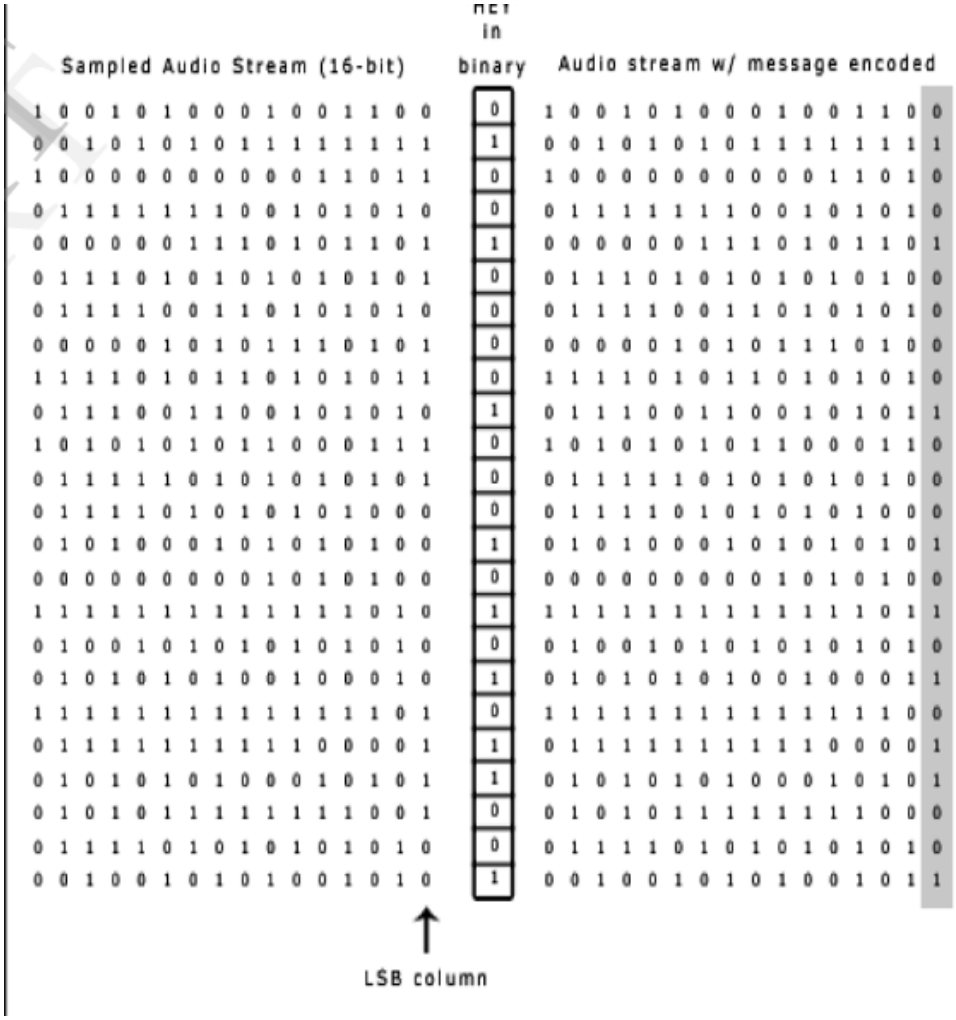
Cette approche exploite la manipulation précise des phases spectrales du signal audio, assurant un codage robuste avec un impact minimal sur la qualité sonore. Elle surpasse ainsi les méthodes classiques basées sur la modification des bits de poids faible (LSB), souvent plus sensibles au bruit et à la dégradation perceptible du signal porteur.

0.4 Stéganographie audio par LSB et modification de bits

0.4.1 Principe fondamental

La stéganographie par LSB (Least Significant Bit) consiste à dissimuler des données dans les bits de poids faible des échantillons audio. Ces bits correspondent aux variations

les plus infimes de l'amplitude du signal, rendant les modifications quasiment indétectables à l'oreille humaine.



0.4.2 Pourquoi utiliser les bits de poids faible ?

Les bits de poids faible sont utilisés car :

— **Perception humaine** : L'oreille humaine est incapable de détecter des changements minimes d'amplitude (ex. : ± 1 sur 65 535 dans un signal 16 bits).

— **Préservation de la qualité** : La modification des bits de poids plus fort engendre des distorsions audibles (cliquetis, pops).

— **Simplicité d'implémentation** : L'algorithme est simple à mettre en œuvre via une manipulation directe des échantillons audio.

0.4.3 Fonctionnement

L'intégration d'un message se fait en plusieurs étapes :

— **Insertion** : Le message secret est converti en binaire. Chaque bit vient remplacer le bit de poids faible d'échantillons audio successifs.

— **Extraction** : La lecture des bits LSB permet de reconstruire le message caché.

— **Compatibilité** : La méthode est efficace avec des formats audio sans perte tels que WAV ou FLAC. Les formats compressés avec perte (MP3) altèrent irrémédiablement les données cachées.

0.4.4 Effets de la modification des bits

— **Bits LSB** : Introduisent un bruit inaudible, idéal pour la discrétion.
— **Bits plus significatifs** : Provoquent des distorsions sonores, pertes de volume ou corruption du fichier.
— **Bits critiques (en-têtes, métadonnées)** : Peuvent rendre le fichier inutilisable.

0.4.5 Avantages et inconvénients

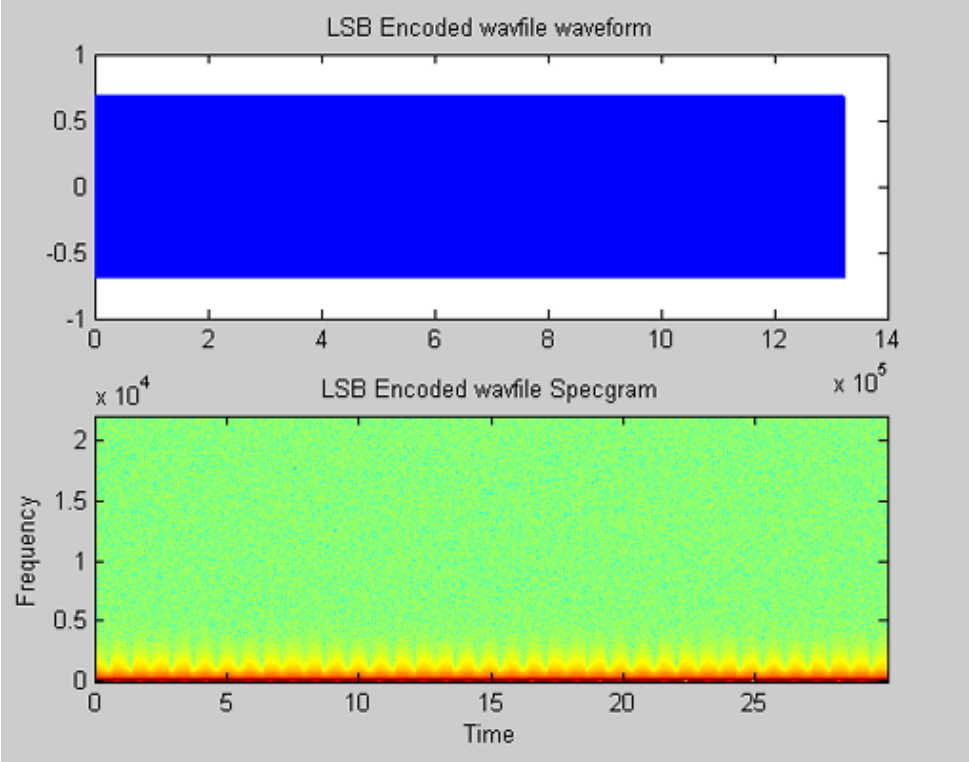
Avantages :
— Méthode simple, rapide et difficile à détecter à l'écoute.
— Compatible avec les formats audio courants.
Inconvénients :
— **Capacité limitée** : Environ 5,5 Ko/s pour un LSB avec un signal à 44,1 kHz.
— **Fragilité** : La compression, le réencodage ou le bruit peuvent détruire les données cachées.
— **Détectabilité** : Des analyses statistiques peuvent révéler des motifs anormaux dans les LSB.

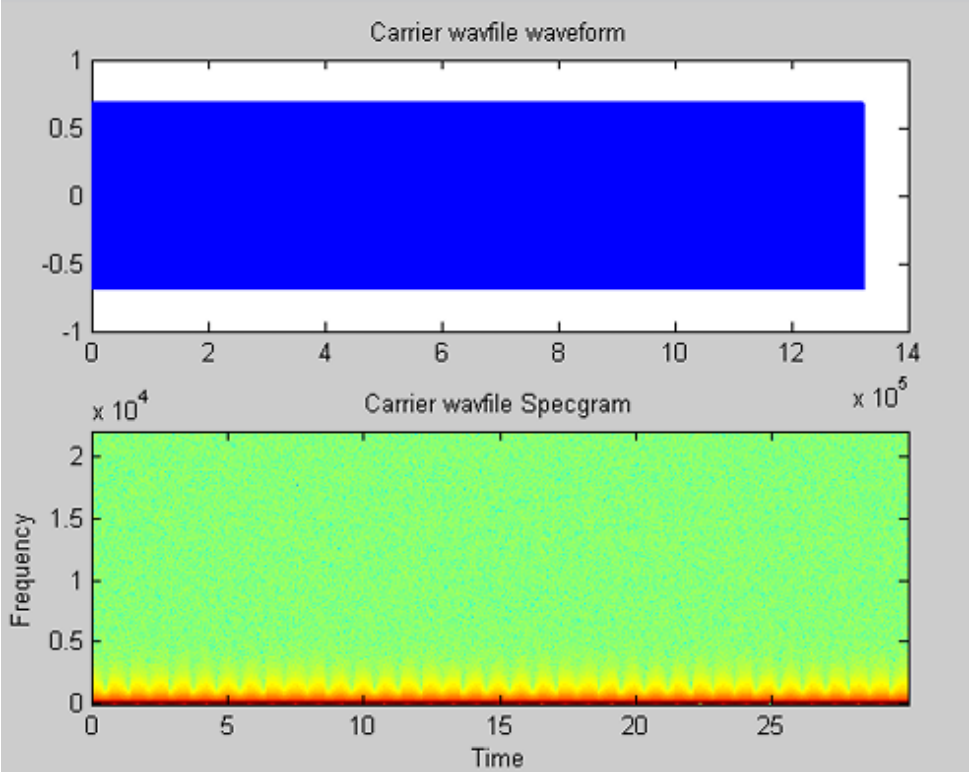
0.4.6 Cas d'utilisation

— **Stéganographie** : Cacher des messages confidentiels dans des fichiers audio.
— **Tatouage numérique** : Intégrer des informations de copyright de manière discrète.

0.4.7 Risques majeurs

— **Perte de données** : Les compressions destructives suppriment les informations dissimulées.
— **Détection** : Des outils spécialisés peuvent identifier les anomalies dans les bits LSB.
— **Dégradation audio** : L'usage excessif de bits (plus de 3-4 bits) peut introduire du bruit audible.





0.4.8 Encodage et décodage LSB en Matlab

L'encodage et le décodage des messages cachés sont réalisés à l'aide de scripts MATLAB personnalisés. La méthode repose sur la modification du bit de poids faible (LSB) de

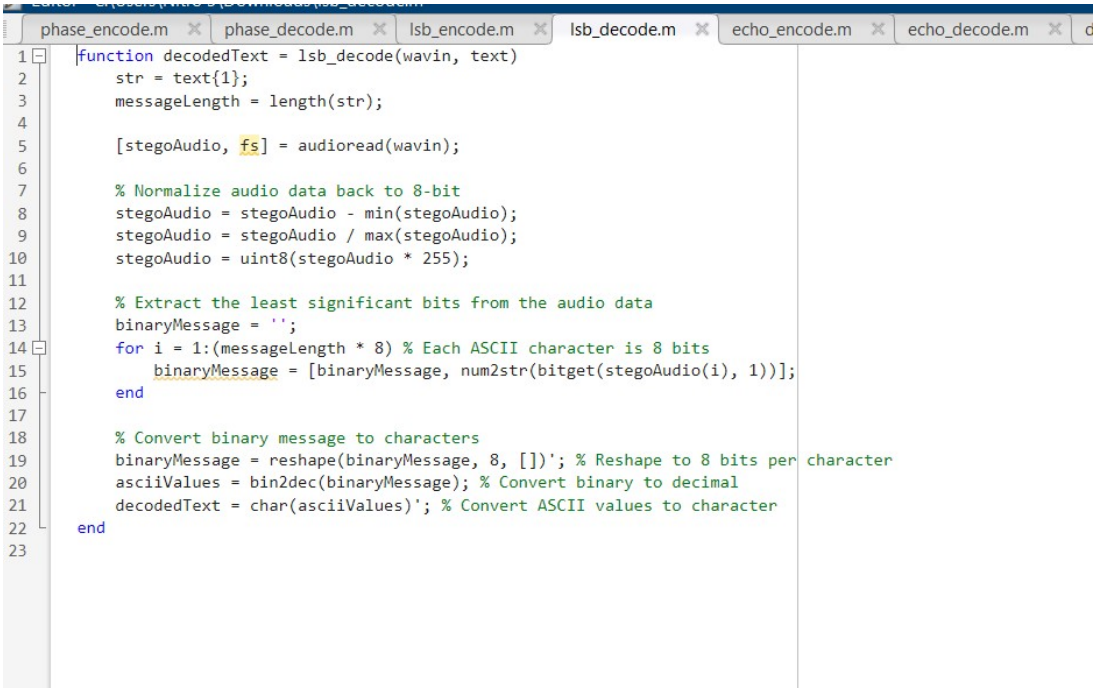


FIGURE 3 – Code MATLAB pour l'encodage LSB : insertion d'un message dans un signal audio.

chaque échantillon audio. Ci-dessous, deux scripts illustrent respectivement le processus d'insertion d'un message texte dans un signal audio, et son extraction.

0.5 Explication des codes Matlab

Encodage (`lsb_encode`) : Ce script lit un fichier audio, le convertit en une échelle de 0 à 255 (8 bits), puis transforme le message à cacher en binaire. Chaque bit du message est ensuite inséré dans le bit de poids faible de chaque échantillon audio. Le fichier audio modifié est ensuite sauvegardé.

Décodage (`lsb_decode`) : Ce script lit le fichier audio contenant le message caché, extrait les bits de poids faible des premiers échantillons (en fonction de la longueur du message), reconstruit la chaîne binaire, puis la convertit en texte lisible.

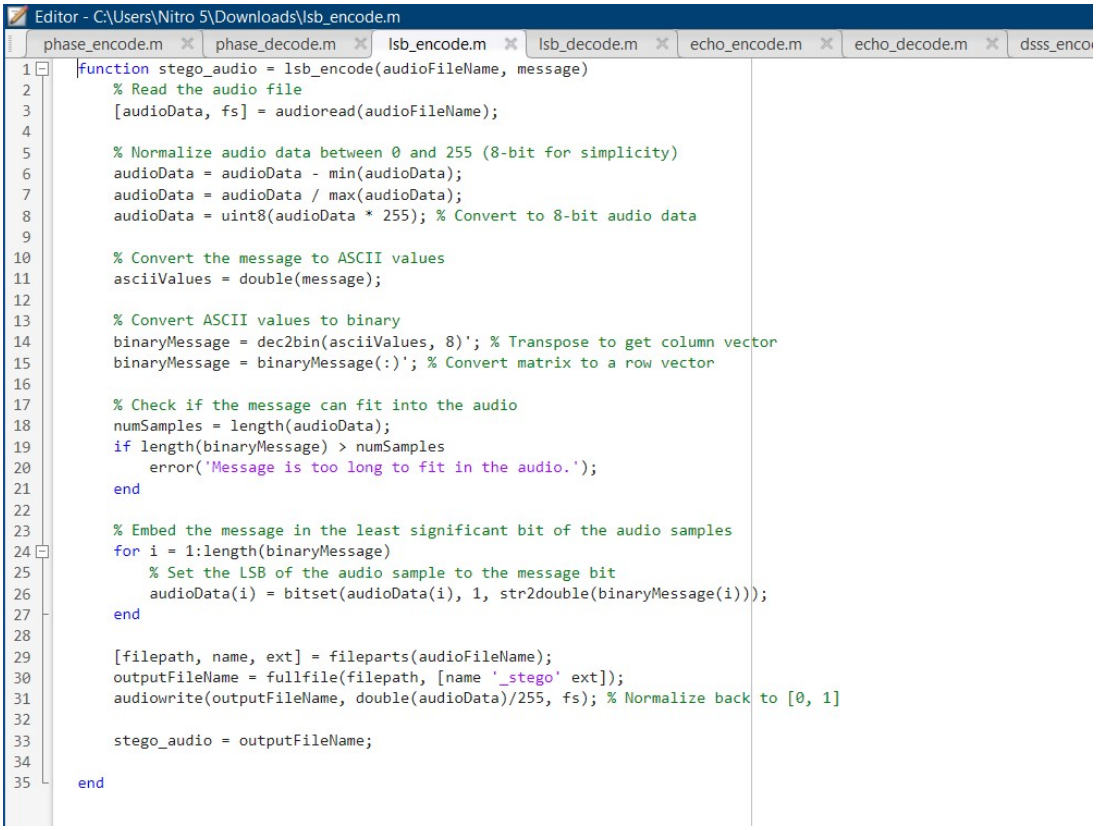


FIGURE 4 – Code MATLAB pour le décodage LSB : extraction du message à partir du signal audio modifié.

0.6 Stéganographie par Étendue de Specter (Spread spectrum) [DSSS]

0.6.1 Principe Fondamental

Contrairement aux méthodes traditionnelles (comme LSB), cette technique répartit le message secret uniformément sur l'ensemble du fichier audio, comparable à la dispersion homogène d'un colorant dans un liquide.

0.6.2 Mécanisme Technique

Processus d'Encodage

1. **Fragmentation du message** :
   — Découpage en éléments binaires
   — Transformation via code de Barker (ex : $c_{11} = \{1, 1, 1, -1, -1, -1, 1, 1, -1, 1, -1\}$)

Sous l'application $0 \to -\mathbf{c}$ et $1 \to \mathbf{c}$, la séquence b est transformée en une séquence w en mappant successivement chaque bit de b vers son image correspondante. La séquence w est ensuite utilisée pour moduler le signal audio en ajoutant δw aux $|w|$ premiers échantillons du signal, où δ est une valeur suffisamment faible pour que le message reste inaudible.

Pour récupérer le message secret, on peut effectuer une convolution de $\mathbf{c}$ avec les échantillons du signal audio stéganographié. Grâce à la faible auto-corrélation de $\mathbf{c}$ avec elle-même, la séquence b peut être reconstruite en analysant les pics de cette convolution.

Processus de Décodage

— Convolution du signal avec le code de Barker
— Détection des pics de corrélation
— Reconstruction du message binaire

0.6.3 Avantages et Inconvénients

Avantages :
— **Difficile à détecter** : aucune structure évidente comme dans LSB.
— **Robuste** : résiste aux compressions et modifications du signal.
— **Sécurisé** : extraction impossible sans la clé.
Inconvénients :
— **Plus lent** : temps de traitement plus long.
— **Capacité réduite** : quantité de données insérables limitée par rapport à LSB.

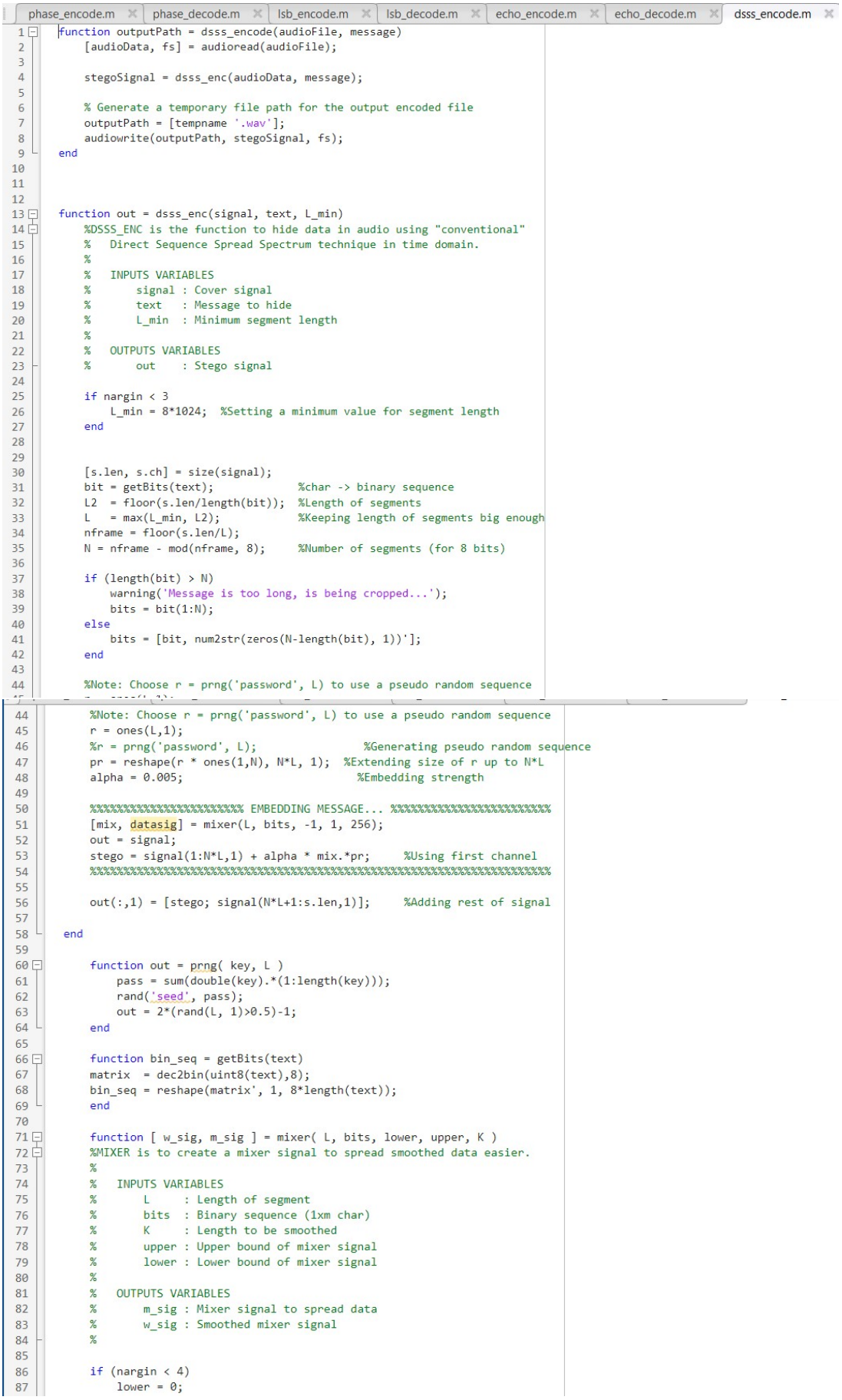
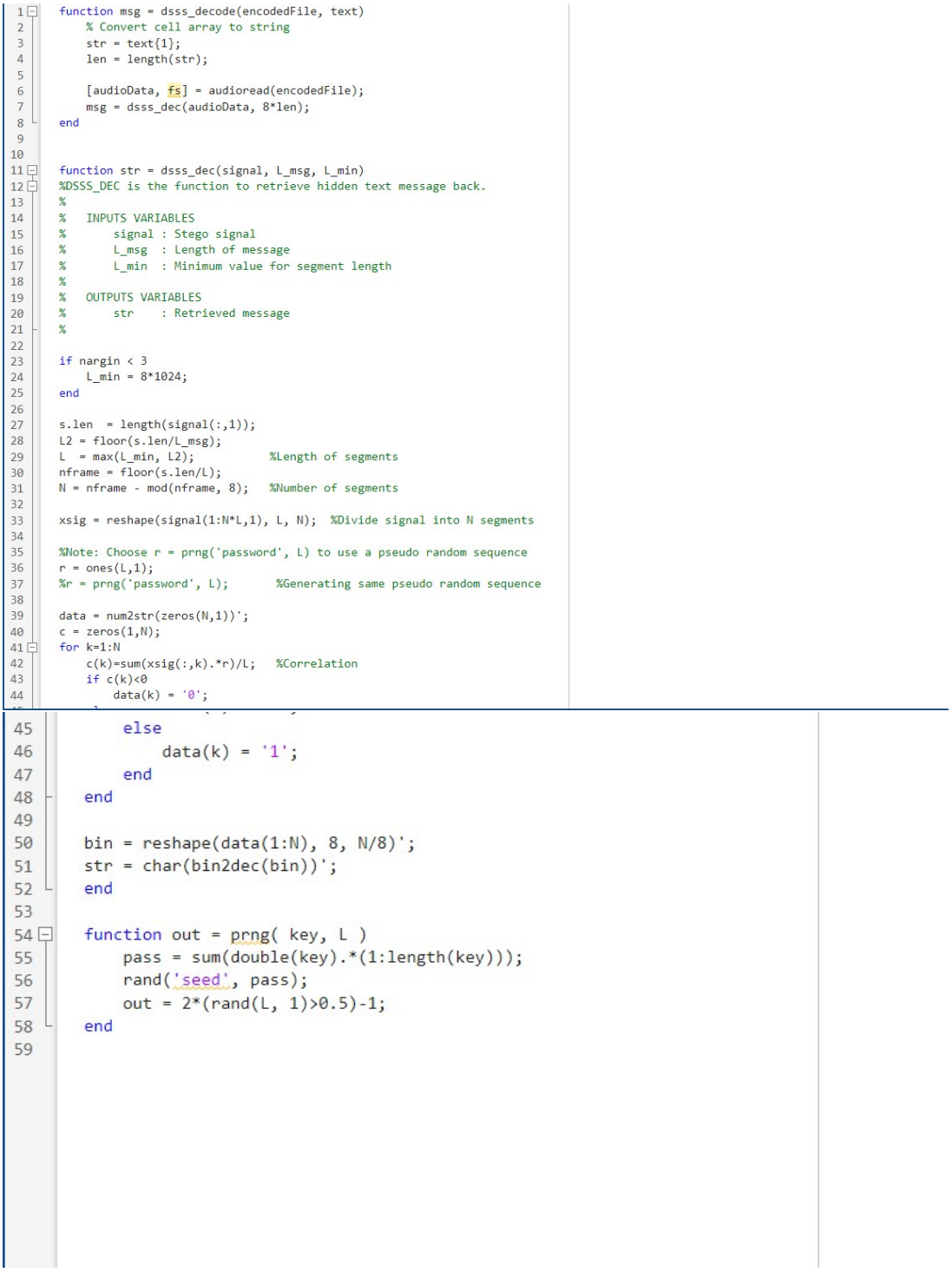


FIGURE 5 – Code MATLAB pour DSSS encodage : insertion d'un message dans un signal audio.



```
function msg = dsss_decode(encodedFile, text)
    % Convert cell array to string
    str = text{1};
    len = length(str);

    [audioData, fs] = audioread(encodedFile);
    msg = dsss_dec(audioData, 8*len);
end


function str = dsss_dec(signal, L_msg, L_min)
%DSSS_DEC is the function to retrieve hidden text message back.
%
%   INPUTS VARIABLES
%       signal : Stego signal
%       L_msg  : Length of message
%       L_min  : Minimum value for segment length
%
%   OUTPUTS VARIABLES
%       str    : Retrieved message
%

if nargin < 3
    L_min = 8*1024;
end

s.len  = length(signal(:,1));
L2 = floor(s.len/L_msg);
L  = max(L_min, L2);           %Length of segments
nframe = floor(s.len/L);
N = nframe - mod(nframe, 8);   %Number of segments

xsig = reshape(signal(1:N*L,1), L, N);  %Divide signal into N segments

%Note: Choose r = prng('password', L) to use a pseudo random sequence
r = ones(L,1);
%r = prng('password', L);       %Generating same pseudo random sequence

data = num2str(zeros(N,1))';
c = zeros(1,N);
for k=1:N
    c(k)=sum(xsig(:,k).*r)/L;   %Correlation
    if c(k)<0
        data(k) = '0';
    else
        data(k) = '1';
    end
end

bin = reshape(data(1:N), 8, N/8)';
str = char(bin2dec(bin))';
end

function out = prng( key, L )
    pass = sum(double(key).*(1:length(key)));
    rand('seed', pass);
    out = 2*(rand(L, 1)>0.5)-1;
end
```

FIGURE 6 – Code MATLAB pour DSSS décodage : extraction du message à partir du signal audio modifié.

0.6.4 Explication des fonctions MATLAB pour la stéganographie DSSS

1. Fonction de décodage : `dsss_decode`

La fonction `dsss_decode` permet d'extraire un message texte caché dans un fichier audio encodé à l'aide de la technique Direct Sequence Spread Spectrum (DSSS).

- **Arguments d'entrée** :
  - Le fichier audio encodé contenant le message caché.
  - Une référence textuelle dont la longueur sert à déterminer le nombre de bits à extraire.
- **Principaux traitements** :
  - Lecture du fichier audio.
  - Calcul du nombre total de bits à récupérer à partir de la longueur de la référence.
  - Appel d'une fonction interne qui réalise la récupération du message en traitant le signal audio par segments.
- **Fonction interne** :
  - Le signal est découpé en segments de taille adaptée.
  - Une séquence pseudo-aléatoire est générée (optionnellement avec une clé).
  - Pour chaque segment, la corrélation entre le signal et la séquence est calculée.
  - La corrélation positive ou négative permet de déterminer les bits (0 ou 1).
  - La suite de bits est ensuite convertie en texte.

2. Fonction d'encodage : `dsss_encode`

La fonction `dsss_encode` permet de cacher un message texte dans un fichier audio en utilisant la technique DSSS.

- **Arguments d'entrée** :
  - Le fichier audio porteur dans lequel le message sera inséré.
  - Le message texte à cacher.
- **Principaux traitements** :
  - Lecture du signal audio porteur.
  - Conversion du message en une séquence binaire.
  - Découpage du signal audio en segments.
  - Génération d'une séquence pseudo-aléatoire (optionnellement avec une clé).
  - Modulation du signal par l'ajout du message diffusé via la séquence pseudo-aléatoire.
  - Recomposition du signal stéganographié et sauvegarde dans un nouveau fichier.

Fonctions auxiliaires utilisées :

- Génération d'une séquence pseudo-aléatoire déterministe à partir d'une clé, utilisée pour moduler et démouler le message.
- Conversion du texte en une séquence binaire.
- Construction d'un signal intermédiaire qui répartit les bits sur les segments du signal audio.

Cette méthode DSSS repose donc sur la modulation du signal audio avec une séquence pseudo-aléatoire pour cacher les bits du message, puis sur une corrélation inverse pour

les extraire, garantissant ainsi une bonne résistance au bruit et une discrétion dans la dissimulation du message.

0.6.5 Comparaison Métaphorique

— **LSB** : Écrire un message secret sur une seule page d'un livre.
— **Spectre Étendu** : Écrire un mot par page à travers tout le livre—beaucoup plus difficile à retrouver.

Quand Utiliser Cette Méthode ?

Cette méthode est recommandée :
— lorsque la sécurité est prioritaire ;
— lorsque le fichier audio est susceptible d'être compressé ou modifié.

0.6.6 Résumé Technique

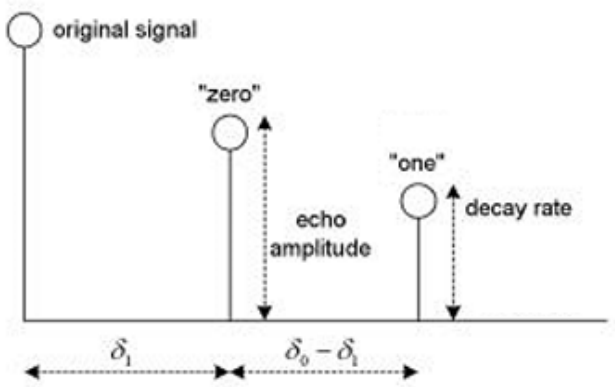
La méthode de spectre étendu consiste à propager un message dans le spectre fréquentiel du signal audio. Le message binaire est d'abord transformé en une séquence plus longue via un code de Barker (par exemple : $c_{11} = \{1, 1, 1, -1, -1, -1, 1, 1, -1, 1, -1\}$). Chaque bit du message est mappé via $0 \to -c$ et $1 \to c$ pour générer une séquence w. Cette séquence w est ensuite utilisée pour moduler les premiers échantillons du signal audio via l'ajout d'un petit facteur δw, de sorte que le message soit inaudible. Pour récupérer le message, on applique une convolution entre le code de Barker et le signal audio modifié. Grâce à la faible auto-corrélation du code de Barker, le message d'origine peut être reconstruit à partir des pics de convolution.

0.6.7 Conclusion

Le spectre étendu dissimule les données en les répartissant dans tout le signal audio, ce qui les rend discrètes, résistantes aux altérations, mais plus complexes à manipuler.

0.7 Stéganographie par (Echo Hiding)

La technique d'insertion par écho consiste à dissimuler un message secret dans un signal audio en introduisant un écho modifié. Cette méthode exploite le fait que lors de la diffusion audio dans différents environnements, des échos naturels sont souvent présents, ce qui rend difficile pour l'oreille humaine de détecter un écho artificiellement inséré.



L'écho est caractérisé par trois paramètres principaux : l'amplitude, le délai (ou retard) et le décalage (offset). Ces paramètres diffèrent selon que le bit du message caché soit un 0 ou un 1.

— **Amplitude** : définit la force de l'écho inséré dans le signal audio.
— **Délai** : indique le temps de retard entre le signal original et l'écho.
— **Décalage** : représente la distance temporelle entre l'écho et le signal original, variant en fonction du bit encodé.

Lorsque ces paramètres sont ajustés avec précision, l'audio modifié (appelé audio stéganographique) reste perceptuellement identique à l'original, assurant ainsi la dissimulation efficace du message.

0.7.1 Principe d'encodage

Le signal audio porteur est divisé en blocs discrets. Chaque bloc est modifié en introduisant un écho dont les caractéristiques dépendent du bit à encoder (0 ou 1). Chaque écho encode donc un bit unique du message secret.

Après la modification, les blocs sont réassemblés pour former le signal audio final contenant le message caché.

0.7.2 Avantages de la méthode

Cette technique présente plusieurs avantages majeurs :

— **Débit de transmission élevé** : la capacité d'insérer de nombreux bits rapidement.
— **Robustesse supérieure** : l'information cachée résiste mieux aux altérations et au bruit.
— **Bonne immunité au bruit** : le message reste récupérable même en présence de perturbations.

0.7.3 Paramètres essentiels

Les trois paramètres clés utilisés pour définir l'écho sont :

— **Amplitude initiale** : utilisée pour définir le niveau d'amplitude du signal porteur.
— **Taux de décroissance** (decay rate) : contrôle la forme et la durée de l'écho.
— **Décalage (offset)** : fixe la distance temporelle entre l'écho et le signal porteur.

encoding :

```m
function outputPath = echo_encode(audioFile, message)
    [audioData, fs] = audioread(audioFile);

    stegoSignal = echo_enc(audioData, message);

    % Generate a temporary file path for the output encoded file
    outputPath = [tempname '.wav'];
    audiowrite(outputPath, stegoSignal, fs);
end


function out = echo_enc(signal, text, d0, d1, alpha, L)
    %ECHO_ENC_SINGLE Echo Hiding Technique with single echo kernel
    %
    %   INPUT VARIABLES
    %       signal : Cover signal
    %       text   : Message to hide
    %       d0     : Delay rate for bit0
    %       d1     : Delay rate for bit1
    %       alpha  : Echo amplitude
    %       L      : Length of frames
    %
    %   OUTPUT VARIABLES
    %       out    : Stego signal


    if nargin < 4
        d0 = 150;     %Delay rate for bit0
        d1 = 200;     %Delay rate for bit1
    end

    if nargin < 5
        alpha = 0.5;  %Echo amplitude
    end

    if nargin < 6
        L = 8*1024;   %Length of frames
    end

    [s.len, s.ch] = size(signal);
    bit = getBits(text);
    nframe = floor(s.len/L);
    N = nframe - mod(nframe,8);      %Number of frames (for 8 bit)

    if (length(bit) > N)
        warning('Message is too long, being cropped!');
        bits = bit(1:N);
    else
        warning('Message is being zero padded...');
        bits = [bit, num2str(zeros(N-length(bit), 1))'];
    end

    k0 = [zeros(d0, 1); 1]*alpha;        %Echo kernel for bit0
    k1 = [zeros(d1, 1); 1]*alpha;        %Echo kernel for bit1

    echo_zro = filter(k0, 1, signal);    %Echo signal for bit0
    echo_one = filter(k1, 1, signal);    %Echo signal for bit1

    window = mixer(L, bits, 0 ,1, 256);  %Mixer signal
    mix = window * ones(1, s.ch);        %Adjusting up to channel size

    %%%%%%%%%%%%%%%%%%%%%%%% EMBEDDING MESSAGE... %%%%%%%%%%%%%%%%%%%%%%%%
    out = signal(1:N*L, :) + echo_zro(1:N*L, :) .* abs(mix-1) ...
                           + echo_one(1:N*L, :) .* mix;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    out = [out; signal(N*L+1:s.len, :)];   %Rest of the signal
end

function bin_seq = getBits(text)
    matrix  = dec2bin(uint8(text),8);
    bin_seq = reshape(matrix', 1, 8*length(text));
end

function [ w_sig, m_sig ] = mixer( L, bits, lower, upper, K )
    %MIXER is the mixer signal to smooth data and spread it easier.
    %
    %   INPUTS VARIABLES
    %       L     : Length of segment
    %       bits  : Binary sequence (1xm char)
    %       K     : Length to be smoothed
    %       upper : Upper bound of mixer signal
    %       lower : Lower bound of mixer signal
    %
    %   OUTPUTS VARIABLES
    %       m_sig : Mixer signal to spread data
    %       w_sig : Smoothed mixer signal
    %


    if (nargin < 4)
        lower = 0;
        upper = 1;
    end

    if (nargin < 5) || (2*K > L)
        K = floor(L/4) - mod(floor(L/4), 4);
    else
```

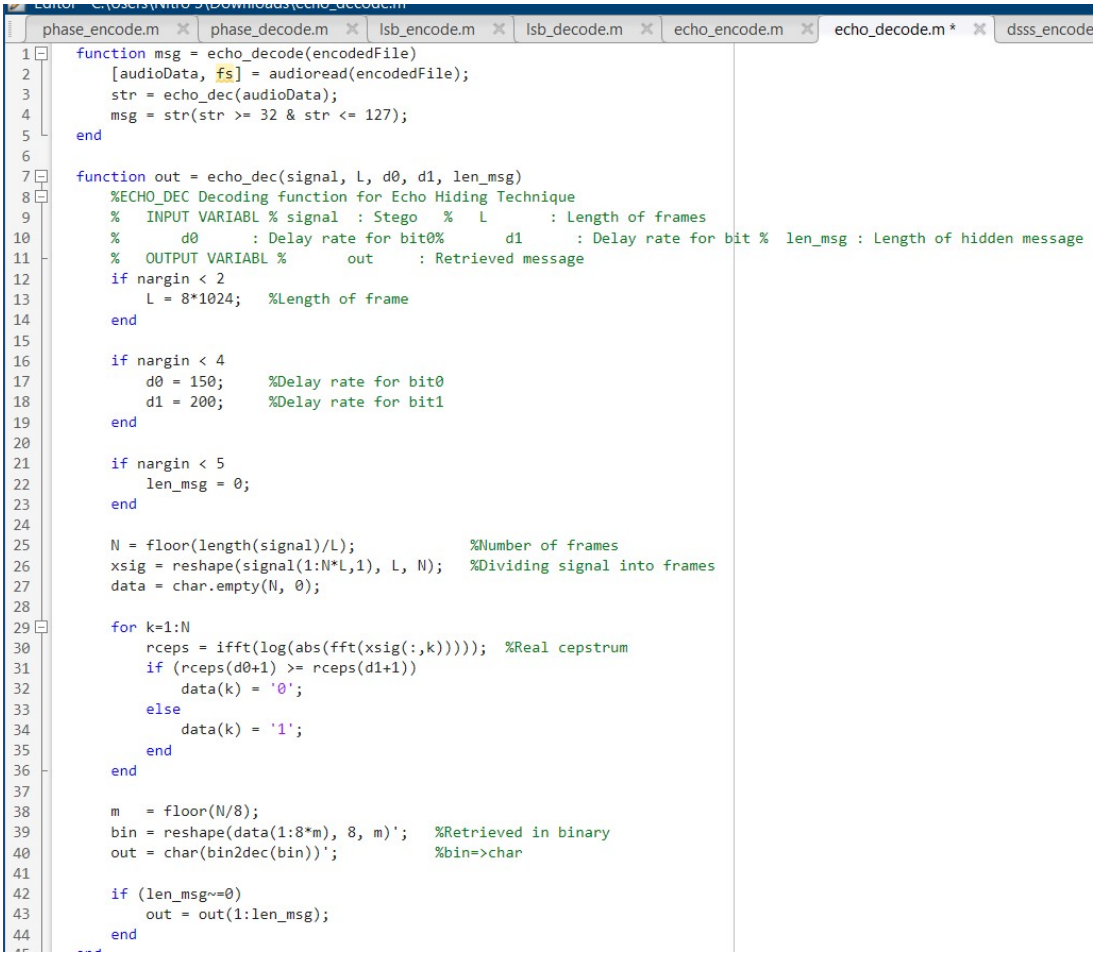


FIGURE 8 – Code MATLAB pour décodage (echo hiding) : extraction du message à partir du signal audio modifié.

0.8 Explication des fonctions `echo_encode` et `echo_enc`

1. Fonction `echo_encode`

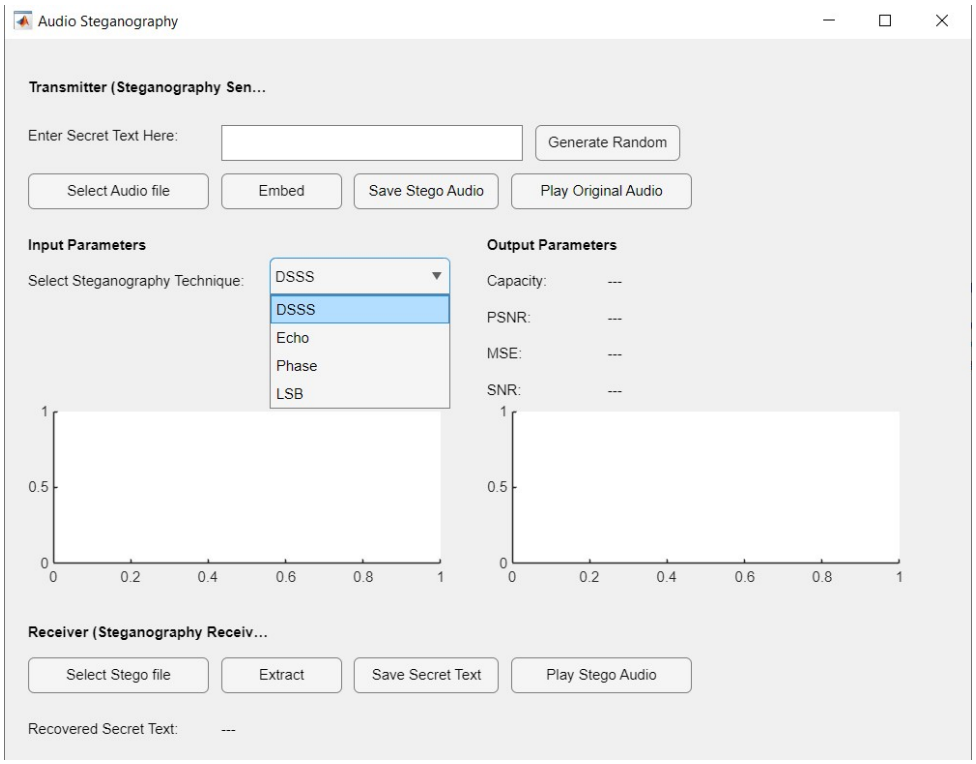
- **Objectif :** Cacher un message texte dans un fichier audio en utilisant la technique de l'écho.
- **Entrées :**
  - `audioFile` : chemin du fichier audio d'entrée.
  - `message` : texte à encoder dans le signal audio.
- **Processus :**
  - Lecture du fichier audio pour récupérer le signal et la fréquence d'échantillonnage.
  - Appel de la fonction `echo_enc` pour insérer le message dans le signal audio.
  - Sauvegarde du signal codé dans un fichier temporaire au format WAV.
- **Sortie :** Chemin du fichier audio contenant le message caché.

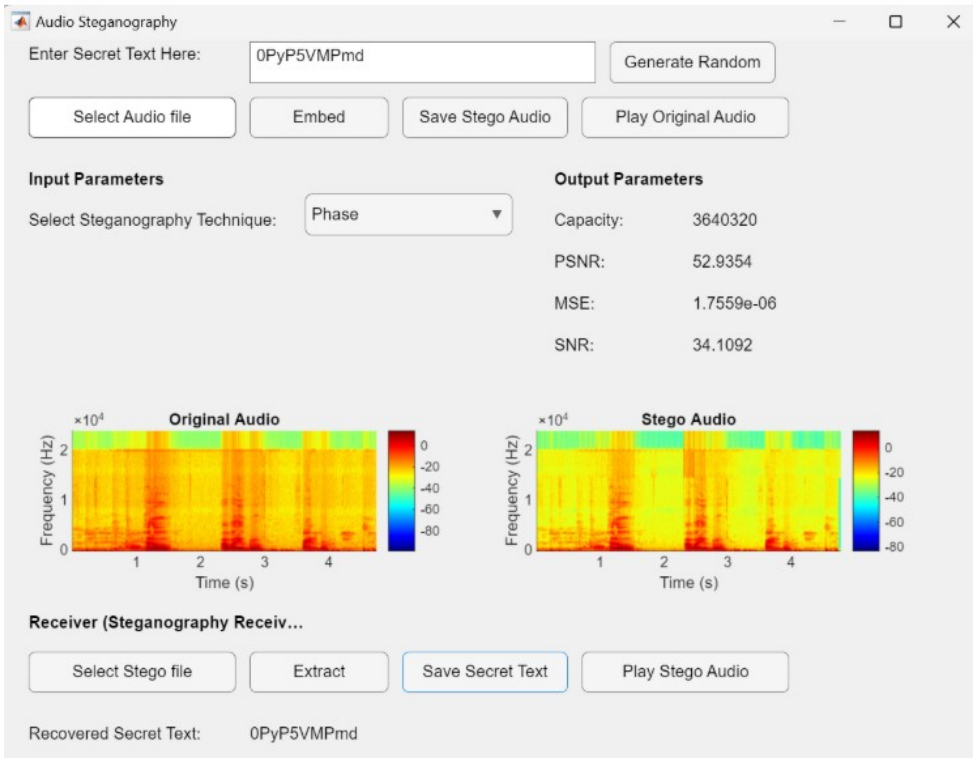
2. Fonction `echo_enc`

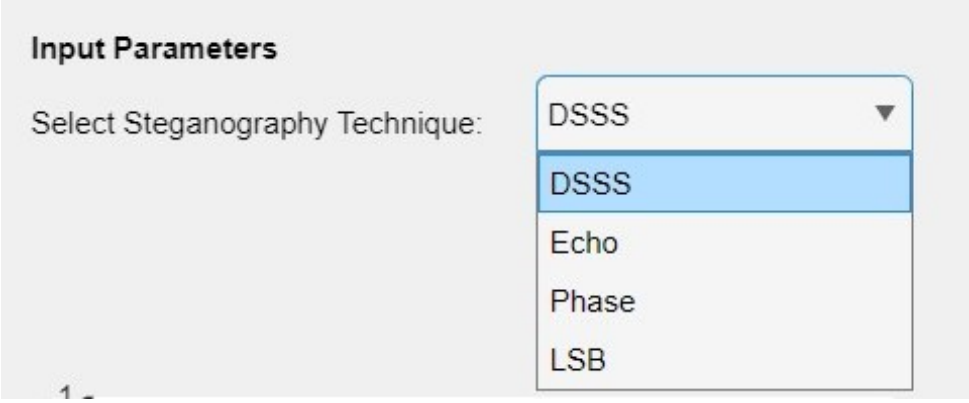
- **Objectif :** Encoder un message binaire dans un signal audio en modulant des échos avec des délais différents.
- **Entrées :**
  - `signal` : signal audio original.
  - `text` : message texte à encoder.
  - `d0`, `d1` (optionnels) : délais des échos pour coder respectivement les bits 0 et 1.
  - `alpha` (optionnel) : amplitude des échos.
  - `L` (optionnel) : longueur des trames audio utilisées pour l'encodage.
- **Processus :**
  - Conversion du texte en une séquence binaire.
  - Division du signal audio en trames.
  - Application de deux filtres d'écho (avec des délais `d0` et `d1`) pour représenter les bits 0 et 1.
  - Création d'un signal de modulation lisse pour répartir l'encodage sur les trames.
  - Combinaison du signal original avec les signaux écho modulés pour obtenir le signal codé.
  - Ajout de la partie du signal audio non traitée à la fin.
- **Sortie :** Signal audio modifié contenant le message caché.

0.9 Application

Le code MATLAB ci-dessous représente l'application principale d'une interface graphique (GUI) pour la stéganographie audio, permettant d'insérer et d'extraire un message secret à l'intérieur d'un fichier audio. L'interface comprend deux parties : l'émetteur (transmetteur) et le récepteur.







— **Interface utilisateur** : L'application utilise la fonction `uifigure` pour créer une fenêtre graphique interactive.
— **Entrée de message** : Un champ de texte permet d'entrer un message secret, avec la possibilité de générer une chaîne aléatoire.
— **Sélection de fichier audio** : L'utilisateur peut choisir un fichier audio .wav à encoder.
— **Techniques de stéganographie** : Plusieurs techniques sont proposées dans un menu déroulant (`DSSS`, `Echo`, `Phase`, `LSB`).
— **Évaluation de performance** : Les métriques telles que PSNR, MSE, SNR, et la capacité sont calculées et affichées.
— **Spectrogrammes** : Deux graphiques affichent les spectrogrammes de l'audio original et de l'audio stéganographié.
— **Réception et extraction** : L'utilisateur peut sélectionner un fichier stégo, en extraire le message et sauvegarder celui-ci.

Le fonctionnement repose sur des fonctions externes (non affichées ici) comme `lsb_encode()`, `dsss_encode()`, etc., qui implémentent les algorithmes de stéganographie.

0.9.1 FULL GUI CODE :

phase_encode.m | phase_decode.m | lsb_encode.m | lsb_decode.m | echo_encode.m | echo_decode.m | dsss_encode.m | dsss_decode.m

```
clc;
clear;

audio_steg_gui();

function audio_steg_gui
    % Create a figure window for the GUI
    fig = uifigure('Position', [100, 100, 800, 600], 'Name', 'Audio Steganography');

    % Input Section (Transmitter side)
    uilabel(fig, 'Text', 'Transmitter (Steganography Sender)', 'Position', [20, 550, 200, 22], 'FontWeight', 'bold');

    uilabel(fig, 'Text', 'Enter Secret Text Here:', 'Position', [20, 510, 150, 22]);
    messageField = uitextarea(fig, 'Position', [180, 500, 250, 30]);

    % Generate Random Text Button
    uibutton(fig, 'Text', 'Generate Random', 'Position', [440, 500, 120, 30], 'ButtonPushedFcn', @(btn, event) generateRandomText());

    uibutton(fig, 'Text', 'Select Audio file', 'Position', [20, 460, 150, 30], 'ButtonPushedFcn', @(btn, event) selectAudio());
    uibutton(fig, 'Text', 'Embed', 'Position', [180, 460, 100, 30], 'ButtonPushedFcn', @(btn, event) encodeAudio(messageField.Value));
    uibutton(fig, 'Text', 'Save Stego Audio', 'Position', [290, 460, 120, 30], 'ButtonPushedFcn', @(btn, event) saveStegoAudio());

     % Parameters for encoding
    uilabel(fig, 'Text', 'Input Parameters', 'Position', [20, 420, 150, 22], 'FontWeight', 'bold');

    % Technique Selector Field
    uilabel(fig, 'Text', 'Select Steganography Technique:', 'Position', [20, 390, 200, 22]);
    techniqueSelector = uidropdown(fig, 'Position', [220, 390, 150, 30], 'Items', {'DSSS', 'Echo', 'Phase', 'LSB'});


    % Output Parameters
    uilabel(fig, 'Text', 'Output Parameters', 'Position', [400, 420, 150, 22], 'FontWeight', 'bold');
    uilabel(fig, 'Text', 'Capacity:', 'Position', [400, 390, 100, 22]);
    capacityLabel = uilabel(fig, 'Position', [500, 390, 100, 22], 'Text', '---');

    uilabel(fig, 'Text', 'PSNR:', 'Position', [400, 360, 100, 22]);
    psnrLabel = uilabel(fig, 'Position', [500, 360, 100, 22], 'Text', '---');

    uilabel(fig, 'Text', 'MSE:', 'Position', [400, 330, 100, 22]);
    mseLabel = uilabel(fig, 'Position', [500, 330, 100, 22], 'Text', '---');

    uilabel(fig, 'Text', 'SNR:', 'Position', [400, 300, 100, 22]);
    snrLabel = uilabel(fig, 'Position', [500, 300, 100, 22], 'Text', '---');

```

```
ax1 = uiaxes(fig, 'Position', [20, 150, 350, 150]);
ax2 = uiaxes(fig, 'Position', [400, 150, 350, 150]);

% Receiver Section
uilabel(fig, 'Text', 'Receiver (Steganography Receiver)', 'Position', [20, 100, 200, 22], 'FontWeight', 'bold');

uibutton(fig, 'Text', 'Select Stego file', 'Position', [20, 60, 150, 30], 'ButtonPushedFcn', @(btn, event) selectStegoFile());
uibutton(fig, 'Text', 'Extract', 'Position', [180, 60, 100, 30], 'ButtonPushedFcn', @(btn, event) decodeAudio());
uibutton(fig, 'Text', 'Save Secret Text', 'Position', [290, 60, 120, 30], 'ButtonPushedFcn', @(btn, event) saveSecretText());

uilabel(fig, 'Text', 'Recovered Secret Text:', 'Position', [20, 20, 150, 22]);
decodedLabel = uilabel(fig, 'Position', [180, 20, 250, 22], 'Text', '---');



% Global variables to store audio paths and results
global audioFile encodedFile decodedMessage;

% Create Play Buttons for Original Audio
uibutton(fig, 'Text', 'Play Original Audio', 'Position', [420, 460, 150, 30], 'ButtonPushedFcn', @(btn, event) playAudio(audioFile));

% Create Play Buttons for Stego Audio
uibutton(fig, 'Text', 'Play Stego Audio', 'Position', [420, 60, 150, 30],'ButtonPushedFcn', @(btn, event) playAudio(encodedFile));

% Function to generate a random alphanumeric string
function generateRandomText()
    characters = ['A':'Z', 'a':'z', '0':'9'];  % Alphanumeric characters
    randomString = characters(randi(numel(characters), [1, 10]));  % Generate random string of length 10
    messageField.Value = randomString;  % Set the value in the message field
end

% Function to play audio
function playAudio(filePath)
    if isempty(filePath)
        uialert(fig, 'Please select an audio file first!', 'No Audio Selected');
        return;
    end

    % Use the sound function to play the audio file
    [y, Fs] = audioread(filePath);
    sound(y, Fs);
end

% Function to select original audio
function selectAudio()
```

```
88
89          % Function to select original audio
90          function selectAudio()
91              [file, path] = uigetfile('*.wav', 'Select an audio file');
92              if isequal(file, 0)
93                  disp('User selected Cancel');
94              else
95                  audioFile = fullfile(path, file);
96                  disp(['User selected ', audioFile]);
97                  [y, Fs] = audioread(audioFile);
98                  plotSpectrogram(ax1, y, Fs, 'Original Audio');
99              end
100         end
101
102         % Function to select stego file
103         function selectStegoFile()
104             [file, path] = uigetfile('*.wav', 'Select a stego file');
105             if isequal(file, 0)
106                 disp('User selected Cancel');
107             else
108                 encodedFile = fullfile(path, file);
109                 disp(['User selected ', encodedFile]);
110                 [y, Fs] = audioread(encodedFile);
111                 plotSpectrogram(ax2, y, Fs, 'Stego Audio');
112             end
113         end
114
115         % Function to save stego audio
116         function saveStegoAudio()
117             [file, path] = uiputfile('*.wav', 'Save Stego Audio As');
118             if isequal(file, 0)
119                 disp('User selected Cancel');
120             else
121                 stegoFilePath = fullfile(path, file);
122                 copyfile(encodedFile, stegoFilePath);
123                 disp(['Stego audio saved to ', stegoFilePath]);
124             end
125         end
126
127         % Function to save secret text
```

```
125         end
126
127         % Function to save secret text
128         function saveSecretText()
129             [file, path] = uiputfile('*.txt', 'Save Secret Text As');
130             if isequal(file, 0)
131                 disp('User selected Cancel');
132             else
133                 textFilePath = fullfile(path, file);
134                 fid = fopen(textFilePath, 'w');
135                 fprintf(fid, '%s', decodedMessage);
136                 fclose(fid);
137                 disp(['Secret text saved to ', textFilePath]);
138             end
139         end
140
141         % Function to encode audio
142         function encodeAudio(message)
143             if isempty(audioFile)
144                 uialert(fig, 'Please select an audio file first!', 'No Audio Selected');
145                 return;
146             end
147
148             % Convert message to string if it's a cell array
149             if iscell(message)
150                 message = strjoin(message, '');
151             end
152
153             % Validate message
154             if isempty(message)
155                 uialert(fig, 'Please enter a message to encode!', 'No Message');
156                 return;
157             end
158
159             %try
160                 % Get the selected technique
161                 selectedTechnique = techniqueSelector.Value;  % Assuming techniqueSelector is your dropdown
162
163                 % Check which technique is selected and call the appropriate function
164                 if strcmp(selectedTechnique, 'DSSS')
165                     encodedFile = dsss_encode(audioFile, message); % Call the DSSS encoding function
166                 elseif strcmp(selectedTechnique, 'Echo')
167                     encodedFile = echo_encode(audioFile, message); % Call the Echo encoding function
168                 elseif strcmp(selectedTechnique, 'Phase')
```

```
            % Check which technique is selected and call the appropriate function
            if strcmp(selectedTechnique, 'DSSS')
                encodedFile = dsss_encode(audioFile, message); % Call the DSSS encoding function
            elseif strcmp(selectedTechnique, 'Echo')
                encodedFile = echo_encode(audioFile, message); % Call the Echo encoding function
            elseif strcmp(selectedTechnique, 'Phase')
                encodedFile = phase_encode(audioFile, message); % Call the Phase encoding function
            elseif strcmp(selectedTechnique, 'LSB')
                encodedFile = lsb_encode(audioFile, message);

            else
                uialert(fig, 'Invalid technique selected!', 'Error');
                return;
            end

            % Plot spectrogram of encoded audio
            [y, Fs] = audioread(encodedFile);
            plotSpectrogram(ax2, y, Fs, 'Stego Audio');

            % Calculate and display metrics
            [mse, psnr, snr, capacity] = calculateMetrics(audioFile, encodedFile);
            mseLabel.Text = num2str(mse);
            psnrLabel.Text = num2str(psnr);
            snrLabel.Text = num2str(snr);
            capacityLabel.Text = num2str(capacity);

            uialert(fig, 'Message successfully encoded.', 'Success', 'Icon', 'success');
        %catch ME
            %uialert(fig, ['Encoding failed: ' ME.message], 'Error', 'Icon', 'error');
        %end
    end



    % Function to decode audio
    function decodeAudio()
        if isempty(encodedFile)
            uialert(fig, 'Please select a stego audio file first!', 'No Audio Selected');
            return;
        end

        % Retrieve the selected technique from the dropdown (or any other UI element)
        technique = techniqueSelector.Value; % Assuming you have a dropdown named techniqueSelector
```

```
202             end
203
204             % Retrieve the selected technique from the dropdown (or any other UI element)
205             technique = techniqueSelector.Value; % Assuming you have a dropdown named techniqueSelector
206
207             try
208                 switch technique
209                     case 'DSSS'
210                         msg = messageField.Value;
211                         decodedMessage = dsss_decode(encodedFile, msg);
212                     case 'Echo'
213                         decodedMessage = echo_decode(encodedFile);
214                     case 'Phase'
215                         decodedMessage = phase_decode(encodedFile);
216                     case 'LSB'
217                         msg = messageField.Value;
218                         decodedMessage = lsb_decode(encodedFile, msg);
219
220
221                     otherwise
222                         uialert(fig, 'Invalid technique selected!', 'Error', 'Icon', 'error');
223                         return;
224                 end
225
226                 decodedLabel.Text = decodedMessage;
227                 uialert(fig, 'Message successfully decoded.', 'Success', 'Icon', 'success');
228             catch ME
229                 uialert(fig, ['Decoding failed: ' ME.message], 'Error', 'Icon', 'error');
230             end
231         end
232
233
234         % Function to plot spectrogram (Fixed version)
235         function plotSpectrogram(ax, y, Fs, titleText)
236             window = hamming(256);
237             noverlap = 250;
238             nfft = 256;
239
240             % Clear the axis
241             cla(ax);
242
243             % Calculate spectrogram
244             [s, f, t] = spectrogram(y, window, noverlap, nfft, Fs);
245
```

```
245
246             % Plot using surface
247             surf(ax, t, f, 10*log10(abs(s)), 'EdgeColor', 'none');
248             view(ax, 0, 90);
249
250             % Set axis properties
251             title(ax, titleText);
252             xlabel(ax, 'Time (s)');
253             ylabel(ax, 'Frequency (Hz)');
254             colorbar(ax);
255
256             % Adjust colormap and axis settings
257             colormap(ax, 'jet');
258             axis(ax, 'tight');
259         end
260
261         % Function to calculate metrics
262         function [mse, psnr, snr, capacity] = calculateMetrics(originalFile, encodedFile)
263             [origAudio, Fs] = audioread(originalFile);
264             [encodedAudio, ~] = audioread(encodedFile);
265
266             % Ensure both signals have the same length
267             minLength = min(length(origAudio), length(encodedAudio));
268             origAudio = origAudio(1:minLength);
269             encodedAudio = encodedAudio(1:minLength);
270
271             % Calculate MSE
272             mse = mean((origAudio - encodedAudio).^2);
273
274             % Calculate PSNR
275             maxVal = max(abs(origAudio));
276             psnr = 20 * log10(maxVal / sqrt(mse));
277
278             % Calculate SNR
279             signalPower = mean(origAudio.^2);
280             noisePower = mean((origAudio - encodedAudio).^2);
281             snr = 10 * log10(signalPower / noisePower);
282
283             % Calculate capacity (in bits)
284             capacity = length(encodedAudio) * 16;  % Assuming 16 bits per sample
285         end
286     end
```