République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique
Université de 8 Mai 1945 – Guelma
Faculté des Sciences et de la Technologie
Département d'Electronique et Télécommunications

Polycopié de Travaux Pratiques
Pour la matière :

Codage et Compression

Destiné aux étudiants :
1ième Année Master en Systèmes de Télécommunications et Réseaux de Télécommunications

Présenté par :
Dr. GUEBGOUB Nassima

2025

TABLES DES MATIERES

Avant-propos

L'objectif de la compression de données est de réduire le nombre de bits utilisés pour stocker ou transmettre des informations. Les techniques de compression de données peuvent être divisées en deux grandes familles:

- Sans perte (Conservatives) comme le codage de Huffman et le codage de Shannon-Fano
- Avec perte (Non conservatives) comme par exemple la compression des images fixes par la norme JPEG basée sur la transformée en Cosinus Discrète DCT (Discret Cosine Transform) et la norme JPEG2000 basée sur la transformée en ondelettes discrète DWT (Discret Wavelet Transform).

L'objectif du codage de canal est de protéger les données compressées issues du codage de source contre les erreurs de transmission pouvant se produire sur le canal de transmission. Il existe différentes familles de codes correcteurs d'erreurs: les codes en blocs comme les codes de Hamming (linéaires ou cycliques) et les code convolutifs.

Dans le cadre, des travaux pratiques programmés pour la première année Master, filières : Systèmes et Réseaux de Télécommunications, matière TP Codage et Compression, ce support permet de Familiariser l'étudiant avec les techniques de codage et de compression comme le codage source, le codage canal et la compression des images fixes.

La mise en œuvre de ces manipulations permet à l'étudiant de mieux :

- Comprendre les principes algorithmes de la compression sans pertes tel que l'algorithme de Huffman et l'algorithme de Shannon Fano.
- Appréhender leurs applications à la compression d'une source numériques et textuelle.
- Évaluation de leurs performances en termes d'efficacité et taux de compression.
- Comprendre le fonctionnement d'une chaîne de transmission numérique complète, incluant les étapes de codage source, codage canal, et transmission.
- Étudier les impacts des bruits et des erreurs introduits par un canal binaire symétrique (CBS) et un canal additif Gaussien blanc (AWGN) sur la qualité de la transmission.
- Analyser et comparer l'efficacité des mécanismes de codage canal tels que les codes de Hamming linéaires/cycliques et le code BCH (Bose-Chaudhuri-Hocquenghem).
- Mesurer la performance globale de la chaîne en termes de taux d'erreur binaire (BER) et qualité des données reconstruite (SNR).

- Implémenter et optimiser la Transformée en Cosinus Discrète unidimensionnelle (DCT-I ou DCT-1D), bidimensionnelle (DCT-II ou DCT-2D) et rapide sous MATLAB, tout en évaluant leurs performances pour la transformation d'images fixes.
- Acquérir des compétences pratiques en implémentation et évaluation de la compression/décompression d'images par la norme JPEG (Joint Photographic Experts Group).
- Étudier le standard de compression d'images fixes par la norme JPEG2000 basé sur les ondelettes (DWT) (Discrete Wavelet Transform), en examinant les principales étapes de décomposition, de quantification et de reconstruction.
- Évaluer les avantages et les limites de cette approche.
- Comparer les performances des deux méthodes (DCT et ondelettes) en termes de taux de compression, de qualité d'image et de temps de calcul.

TP 1 : Etude et simulation du Codage de Huffman

I. Objectif du TP

- Compression/décompression d'une source numérique, un texte en utilisant l'algorithme de Huffman.
- Évaluer les performances du codage Huffman, incluant l'efficacité et le taux de compression.
- Utiliser certaines fonctions de Matlab.

II. Rappels théoriques

II.1. Le codage de Huffman

Le codage de Huffman est un algorithme de compression de données sans perte, développé par **David Albert Huffman**. Il est basé sur les probabilités d'apparition des symboles d'une source, à partir desquelles le code est déterminé. Ce code est un code préfix, c'est-à-dire qu'aucun mot du code ne peut être le préfixe d'un autre. Le principe du codage de Huffman repose sur l'attribution de codes plus courts aux symboles les plus fréquents et de codes plus longs aux symboles moins fréquents. Les deux symboles les moins fréquents auront des codes de même longueur.

II.1.2. L'efficacité du code

$$E = \frac{H(x)}{\bar{n}} \qquad (1)$$

Avec $\bar{n}$ la longueur moyenne de code et *H(x)* l'entropie de la source

$$H(x) = -\sum P_i \, log_2(p_i) \qquad (2)$$

$$\bar{n} = \sum n_i \, p_i \qquad (3)$$

II.1.3. Taux de compression :

$$\tau = \frac{\textbf{Taille du fichier compressé}}{\textbf{Taille du fichier initial}} \qquad (4)$$

II.1.4. Algorithme principale

L'algorithme de Huffman utilise une structure arborescente. Les probabilités d'apparition des symboles sont d'abord triées par ordre décroissant. Ensuite, les deux symboles ayant les plus faibles probabilités sont combinés en un nouveau symbole dont la

probabilité est la somme de leurs probabilités respectives. Cette opération est répétée jusqu'à ce qu'il ne reste qu'un seul symbole, construisant ainsi un arbre. Les feuilles de cet arbre représentent les symboles à coder, tandis que les embranchements correspondent aux étapes intermédiaires du codage. Par convention, le fils gauche d'un nœud est étiqueté par un 0 et le fils droit par un 1.

1. **Analyse de fréquence :**
   Compter la fréquence d'apparition de chaque symbole dans les données.
2. **Construction d'une file de priorité :**
   Créer une file de priorité contenant tous les symboles, triés par leur fréquence.
   Chaque symbole est représenté par un nœud.
3. **Construction de l'arbre de Huffman :**
   Tant qu'il y a plus d'un nœud dans la file :
   a. Extraire les deux nœuds de plus faible fréquence.
   b. Créer un nouveau nœud parent avec une fréquence égale à la somme des deux fréquences.
   c. Ajouter ce nouveau nœud à la file.

   Répéter jusqu'à ce qu'il reste un seul nœud, qui devient la racine de l'arbre.
4. **Génération des codes binaires :**
   Parcourir l'arbre depuis la racine. Attribuer :
   Un 0 pour chaque branche gauche.
   Un 1 pour chaque branche droite. Chaque chemin depuis la racine jusqu'à une feuille correspond au code binaire d'un symbole.
5. **Encodage :**
   Remplacer chaque symbole dans les données par son code binaire.

II.1.5. Exemple d'application :

Données à compresser : Une séquence de lettres : **seq= 'cagat aagagaa'**

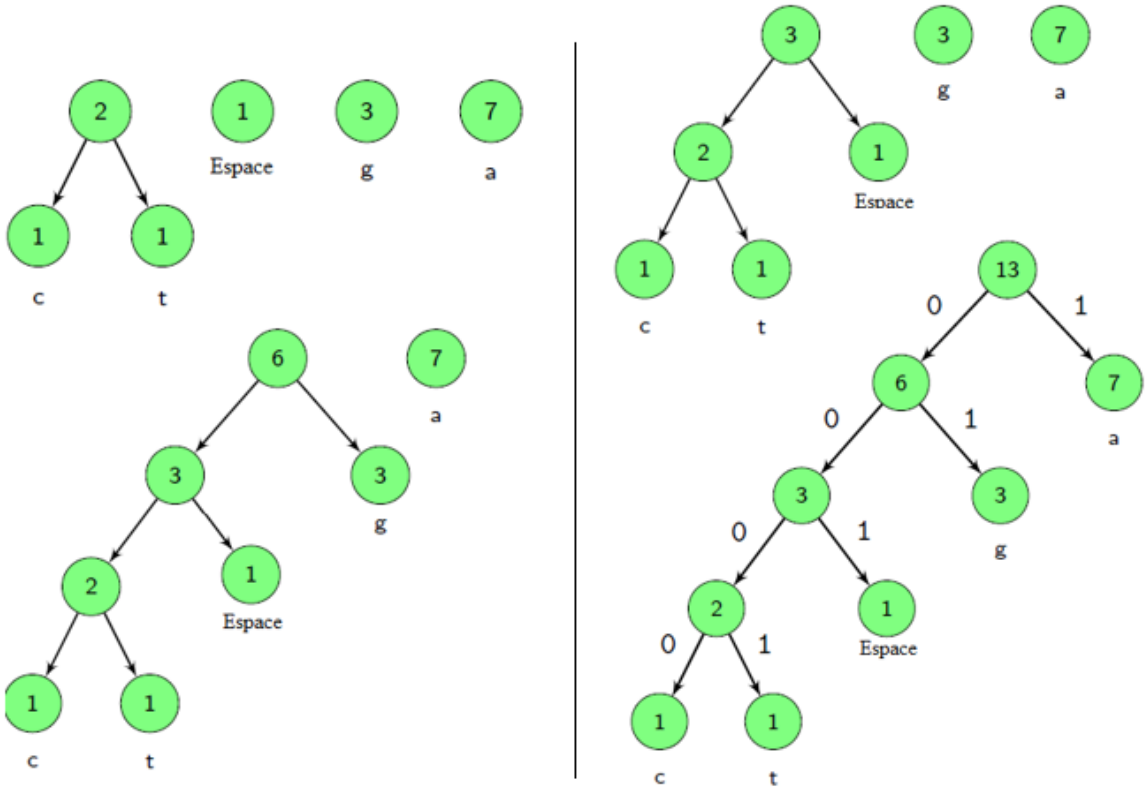
Étape 1 : Analyse de fréquence

Symbole	fréquence
a	3
g	3
espace	1
t	1
c	1

Étape 2 : Construction de la file de priorité

Initialisation avec les fréquences triées :

Étape 3 : Construction de l'arbre de Huffman



Étape 4 : Génération des codes binaires

Caractère	' '	'c'	't'	'g'	'a'
Code	**001**	**0000**	**0001**	**01**	**1**

Étape 5 : Encodage

Les données **seq= 'cagat aagagaa'** deviennent :

0000 1 01 1 0001 001 1 1 01 1 01 1 1

Donc le message est codé sur 24 bits.

Taux de Comp=24/104=0.23=23% (Entrée contient : 13x8 bits= 104 bits)

Remarque:

Pour un cycle de compression/décompression, le décompresseur doit connaître les codes des caractères (dictionnaire) construits par le compresseur, alors pendant la phase de compression, le compresseur écrira donc ces codes en début de fichier compressé, sous un

format à définir, connu du compresseur et du décompresseur. Le fichier compressé aura donc deux parties disjointes :

- Une première partie permettant au décompresseur de retrouver le code de chaque caractère (dictionnaire).
- Une seconde partie contenant la suite des codes des caractères du fichier à compresser.

III. Partie Pratiques

III.1. Compression/Décompression d'une séquence numérique

Considérant l'alphabet de la source Alpha= {1,2, 3} de probabilités P= [0.1 0.1 0.8] et la séquence à compresser : **Seq =3313333323**.

1) Utiliser les fonctions Matlab ***huffmandict*** et ***huffmanenco*** pour compresser la séquence des symboles numériques **Seq**.
2) Calculer l'entropie de la source, la longueur moyenne et l'efficacité du code en utilisant les fonctions Matlab suivantes : ***length, log2, unique***, ***sum***, ***avglen***.
3) Même calcul si le vecteur de probabilités est : P= [0.25 0.25 0.5]
4) Que remarquer vous ?
5) Utiliser la fonction Matlab ***huffmandeco*** pour décompresser la séquence binaire fournie par le compresseur du Huffman.
6) Comparer la taille initiale de la séquence (un octet pour chaque symbole) avec la taille du code fourni par le compresseur de Huffman en calculant le taux de compression.

III.2. Compression/Décompression des données textuelles

Considérant l'alphabet de la source Alpha= {espace, a, b, c, d, e, f, g, h}. Soit le texte à compresser est **msg = ''a bccd aeffhg ga''**.

1) Ecrire un code Matlab permettant d'effectuer les tâches suivantes :

- Calcul de la fréquence d'apparition des caractères
- Calcul des probabilités d'apparition des caractères
- Calcul de l'information propre associée à chaque caractère
- Calcul de probabilité totale
- Calcul de l'entropie

NB) Utiliser les fonctions Matlab suivantes: ***length, unique, sum, log2***

Entête du 1er script :

```
clc;clear all;
seq=input('entrer la séquence');
carac_c=[];
freq_c=[];
p=[];
info_c=[];
```

2) En se basant sur le script précédent écrire un nouveau code Matlab permettant d'effectuer les tâches suivantes :

- ✓ Afficher un message demandant à l'utilisateur de saisir le texte à coder. La chaîne de caractères saisie doit être automatiquement affectée à la variable ***text***.
- ✓ Calculer les probabilités d'occurrence de chaque symbole dans le texte.
- ✓ Calculer l'entropie de la source.
- ✓ Afficher les mots du code correspondant à chaque caractère de l'alphabet en utilisant la fonction ***huffmandict***.
- ✓ Compresser le texte en remplaçant chaque symbole par son code correspondant.
- ✓ Afficher la séquence binaire résultante après le codage.
- ✓ Tester et décompresser cette séquence en utilisant la fonction ***huffmandeco***.
- ✓ Calculer la longueur moyenne du code.
- ✓ Calculer l'efficacité du code.
- ✓ Calculer le taux de compression.

Remarque : Le texte choisi est constitué de n'importe quels symboles de l'alphabet:
Alph= {Espace, a, …, z}.
On pourra utiliser les fonctions Matlab suivantes: ***abs, fprinf, sum, strcat, isequal, save.***

TP2 : Etude et simulation du Codage de Shannon-Fano

I. Objectifs du TP

- Effectuer le codage d'une chaine de symboles en utilisant l'algorithme de Shannon-Fano.
- Calculer les paramètres statistiques de codage Shannon-Fano.
- Comparaison de performance avec le codage de Huffman.

II. Rappels théoriques

II.1. Le codage de Shannon-Fano

Le codage de Shannon-Fano ou codage de Fano-Shannon est un algorithme de compression de données sans perte élaboré par **Robert Fano** à partir d'une idée de **Claude Shannon**. Il s'agit d'un codage entropique produisant un code préfixe très similaire à un code de Huffman, bien que pas toujours optimal, contrairement à ce dernier.

II.1.1. Algorithme de Shannon-Fano

- Classer les différents symboles à coder suivant l'ordre décroissant de leur probabilité.
- Diviser ces symboles en deux sous-groupes de telle sorte que les probabilités cumulées de ces deux sous-groupes soient aussi proches que possible l'une de l'autre.
- On affecte le code « 0 » pour le 1er sous ensemble et le code « 1 » pour l'autre.
- Répéter l'opération jusqu'à obtenir un seul nœud en appliquant les deux premières étapes aux deux sous-ensembles.

III. Parties pratiques

III.1. Compression/Décompression des données textuelles

Considérant le message suivant : **mes=** ''**a bccd aeffhg ga**''.

1) Ecrire un programme MATLAB permettant de :

- ✓ Générer et afficher le dictionnaire à l'aide de la fonction ci-dessous ***ShannonFanoFunc.m*.**
- ✓ Calculer et afficher l'efficacité du code obtenu.
- ✓ Compresser le message **mes** et afficher son code de Shannon-Fano.

- ✓ Évaluer et afficher le taux de compression obtenu.
- ✓ Tester la décompression du code généré pour vérifier la fidélité du message reconstruit.

2) Recompressez le même message à l'aide de l'algorithme de Huffman, basé sur le code développé dans le TP 1. Comparez ensuite les performances des deux algorithmes en termes d'efficacité du code et de taux de compression.

La fonction Matalab *ShannonFanoFunc.m* :

```
function [code1, average_length] = ShannonFanoFunc(p)
set(0,'RecursionLimit',1e4);
% p1 = probability vector
% code1 = corresponds codewords
% average_length is the expected codeword length
% check if p1 is row vector or column vector
if ((sum(p>=0)~=length(p)))
    error('Enter a probability vector');
end
p = p/sum(p);
if(length(p)>2)
[pdes,idx] = sort(p,'descend');
qsum = (2*cumsum(pdes))-1;
[~,idx1] = min(abs(qsum));
    if((idx1>1)&&(idx1<length(pdes)-1))
        q1 = pdes(1:idx1);                  % break into left
half
        q2 = pdes((idx1+1):length(pdes));   % right half
        old_code1 = ShannonFanoFunc(q1);    % recursive call
left
        old_code2 = ShannonFanoFunc(q2);    % recursive call
right
        new_code = [strcat('0',old_code1)
strcat('1',old_code2)]; % code 0 to left
    elseif(idx1==1)
        q1 = pdes(1);
        q2 = pdes(2:length(pdes));
        old_code1 = ShannonFanoFunc(q1);
        old_code2 = ShannonFanoFunc(q2);
        new_code = [old_code1  strcat('1',old_code2)];
    else
        q1 = pdes(1:((length(pdes)-1)));
        q2 = pdes(length(pdes));
```

```
        old_code1 = ShannonFanoFunc(q1);
        old_code2 = ShannonFanoFunc(q2);
        new_code = [strcat('1',old_code1)  old_code2];
    end
    code1(idx) = new_code;
elseif(length(p)==2)
    code1 = {'0', '1'};
else
    code1 = {'0'};
end
length1 = cellfun(@length, code1);
average_length = sum(length1.*p);
end
```

III.2. Compression/Décompression d'un fichier texte

1) Créer un script permettant d'effectuer les tâches suivantes :

- ✓ Lecture du fichier texte à compresser.
- ✓ Calcul des probabilités d'occurrence de chaque symbole dans le texte.
- ✓ Assignation de codes binaires uniques à chaque symbole, basée sur les ensembles obtenus par l'algorithme de Shannon-Fano.
- ✓ Calcul de l'efficacité du code.
- ✓ Compression du texte en remplaçant chaque symbole par son code correspondant.
- ✓ Stockage des codes et des données compressées dans un fichier compressé.
- ✓ Décompression du fichier compressé.
- ✓ Calcul du taux de compression.

Remarque : On pourra utiliser les fonctions Matlab suivantes ***: fopen, fprintf, disp, fclose, textscan.***

2) Donner les avantages et inconvénients des algorithmes de Huffman et Shannon-Fano et de leurs applications dans le domaine de la compression multimédia.

TP3 : Modélisation d'une chaine de transmission avec codage canal et codage source sur un canal binaire puis gaussien

I. Objectif du TP

- Mettre en œuvre des codeurs/décodeurs de canal (codes en bloc) à l'aide du logiciel Matlab.
- Utiliser le logiciel MATLAB avec la Communication Toolbox et l'outil de simulation intégré SIMULINK pour :
  - Développer un premier modèle d'une chaîne de transmission en bande de base sur un canal binaire symétrique (BSC).
  - Développer un deuxième modèle d'une chaîne de transmission en bande de base sur un canal à bruit blanc additif gaussien (AWGN), avec un codage de source en bande de base.
  - Inclure des outils de codage/décodage de canal pour évaluer les performances des différents codeurs (Hamming, BCH, et le code RS (Reed-Solomon)).

II. Rappels théoriques

II.1. Transmission numérique en bande de base

Le but principal d'un système de communication est de transférer l'information de la source vers un utilisateur (destinataire) à travers un canal de transmission. En général, un système de transmission numérique est représenté par le modèle de base suivant :

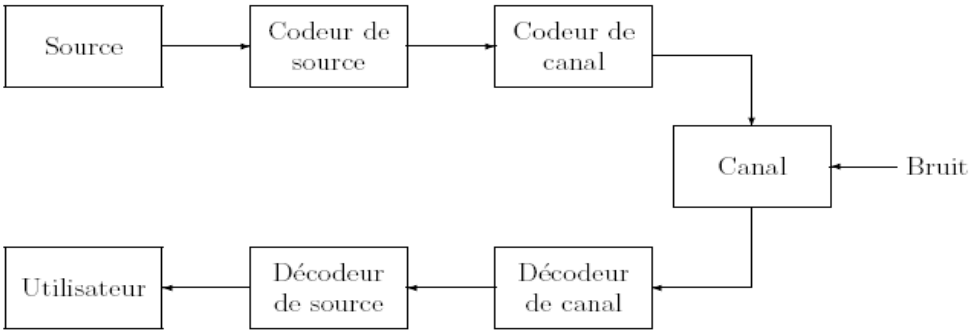


Figure 1 : Chaine de transmission numérique

La transmission de données numériques implique plusieurs étapes, allant de la préparation des données (codage source) à leur protection contre les erreurs (codage canal) pour une transmission fiable sur des canaux bruités. Deux modèles courants de canaux sont :

- **Le canal binaire symétrique (CBS)**
- **Le canal à bruit blanc gaussien (AWGN)**

II.1.1. Le canal binaire symétrique (CBS)

Le canal binaire symétrique (CBS) modèle discret utilisé pour simplifier l'analyse.
Le canal est caractérisé par le lien entre l'entrée et la sortie.

Le schéma complet d'un CBS est donné par :

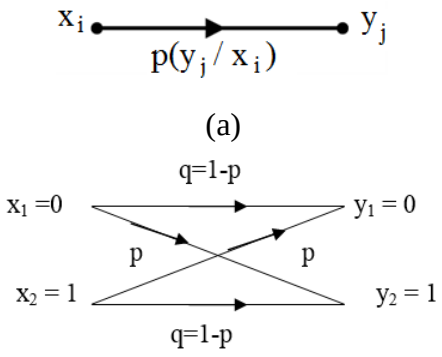


Figure 2 : (a) Modèle mathématique, (b) Canal binaire symétrique (CBS)

II.1.2. Le canal à bruit blanc gaussien (AWGN)

Le canal à bruit blanc gaussien (AWGN) modèle continu qui représente les systèmes physiques réalistes affectés par du bruit thermique.

Modèle mathématique Continu : Le plan du canal de transmission est détaillé sur le schéma bloc ci-dessous :

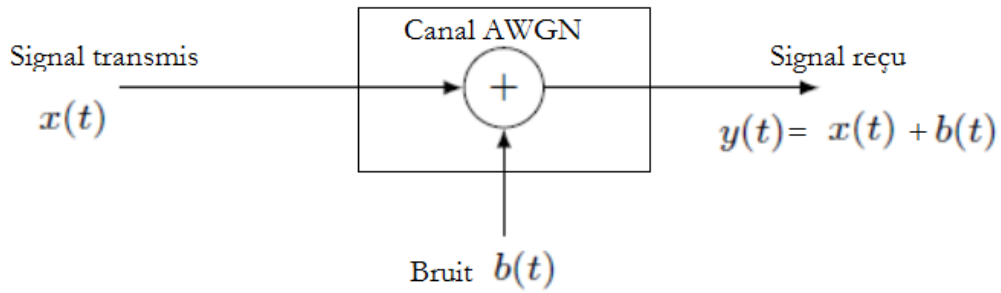


Figure 3 : Modèle d'un canal à bruit blanc gaussien additif (AWGN)

Le canal est modélisé uniquement par un bruit additif gaussien b(t) créant un signal bruite

$$y(t) = x(t) + b(t) \qquad (1)$$

Performance de la chaine de transmission :

Nous pouvons évaluer la performance de la chaine de transmission par le calcul du taux d'erreur par bit (TEB), (BER : bit error rate) donne par :

$$TEB = \frac{Nombre\ de\ bit\ mal\ transmis}{nombre\ de\ bit\ transmis} \qquad (2)$$

II.2. Codes en blocs linéaires

Les codes en bloc linéaire sont une classe de codes correcteurs d'erreurs utilisés dans les systèmes de communication pour détecter et corriger des erreurs survenant pendant la transmission. Un **code linéaire** est un **sous-espace vectoriel** d'un espace vectoriel fini défini sur un **corps fini**, souvent noté **GF(2)**, c'est-à-dire le **Galois Field of order 2** (*corps de Galois d'ordre 2* : C'est le corps fini contenant **deux éléments {0, 1}**, avec les opérations de somme et produit modulo 2.).

Ces codes exploitent les propriétés algébriques pour ajouter de la redondance aux données transmises. Cette redondance introduite permettra à la réception de détecter et/ou de corriger d'éventuelles erreurs de transmission.

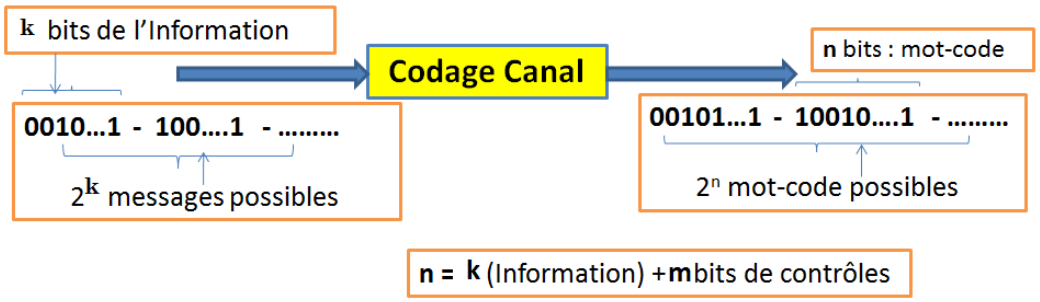


Figure 4 : Principe de code en bloc linéaire

Codeur en bloc linéaire : Un code en bloc est caractérisé par trois paramètres [n, k, d_{min}] où :

- n : Longueur du mot de code (nombre de bits après encodage).
- k : Nombre de bits d'information (données originales).
- d_{min} : Distance minimale entre deux mots de code (mesure de la capacité de correction d'erreurs).
- Le taux de code est donné par R=k/n, avec : $2^k \geq m+k+1 = n+1$
- Les codes linéaires de Hamming où $2^k = m+k+1 = n+1$

Propriétés linéaires :

- Un code est linéaire si la somme (bit à bit, modulo 2) de deux mots de code est encore un mot de code.
- Cela implique qu'un code linéaire contient le mot nul.

II.2.1 Matrice génératrice G

Une matrice de dimension (k×n) utilisée pour générer les mots de code. L'opération de codage est ici réalisée par multiplication matricielle :

$$\boldsymbol{C = i \cdot G} \quad (3)$$

où : ***i*** est le vecteur de données d'entrée (k bits), ***C*** est le mot de code (n bits).

II.2.2. Matrice de contrôle H

Une matrice de dimension (m×n) utilisée pour vérifier si un mot de code est valide :

$$\boldsymbol{H \,.\, v^T = 0} \quad (4)$$

où :

Si le résultat est 0, le mot est valide (un mot de code). Si non, cela indique une erreur.

Cette opération appliquée sur le mot-code reçu ***v'*** permet de calculer le correcteur (ou syndrome) et de détecter les erreurs si :

$$\boldsymbol{H \,.\, v'^T = s \neq 0} \quad (5)$$

H est orthogonal à ***G***, i.e.,

$$\boldsymbol{H \cdot G^T = 0} \quad (6)$$

G et ***H*** peuvent s'écrire sous leur forme systématique ou canonique :

$$G = \left[\begin{array}{c:c} & \\ I_k & A_{k,r} \\ & \end{array}\right] \text{ et } H = \left[\begin{array}{c:c} & \\ A^t{}_{k,r} & I_r \\ & \end{array}\right]$$

où $\boldsymbol{I_k}$ est une matrice identité ou unité et $\boldsymbol{A_{k,r}}$ matrice de parité.

II.2.3. Performances

1. **Capacité de correction d'erreurs :**

   La distance minimale d_{min} détermine :

   - **Détection** : Jusqu'à $\boldsymbol{d_{min} - 1}$ erreurs.
   - **Correction** : Jusqu'à $\boldsymbol{[(d_{min}-1)/2]}$ erreurs.

2. **Taux de redondance :**

   Plus k/n est faible, plus la redondance est élevée.

II.3. Codes en blocs cycliques

Les codes cycliques sont une sous-classe des codes linéaires qui possèdent une structure supplémentaire utile pour la correction des erreurs. Ils sont largement utilisés dans les systèmes de communication et de stockage numérique, notamment pour leur efficacité en termes de décodage et leur facilité de mise en œuvre à l'aide de registres à décalage.

Un code cyclique est un **code linéaire** avec une propriété spéciale :

Si un mot de code $c = [c_0,c_1,\ldots,c_{n-1}]$ appartient au code, alors tout **décalage cyclique** de c appartient également au code.

Par exemple, si $c = [c_0,c_1,c_2,c_3]$, alors $[c_3,c_0,c_1,c_2]$ est aussi un mot de code.

Les codes cycliques sont souvent décrits en termes de **polynômes** sur le corps fini **GF(2) (**Galois Field of order 2**).**

II.3.1. Mot de code comme polynôme

Un mot de code $c=[c_0,c_1,\ldots,c_{n-1}]$ peut être représenté par un polynôme :

$$\boldsymbol{c(x)=c_0 + c_1.x + c_2.x^2 + \ldots + c_{n-1}x^{n-1}} \quad (7)$$

Le décalage cyclique de $\boldsymbol{c(x)}$ correspond à une multiplication par $\boldsymbol{x}$, modulo $\boldsymbol{x^n - 1}$.

II.3.2. Générateur du code

Tout code cyclique peut être défini par un **polynôme générateur** $\boldsymbol{g(x)}$ qui divise $\boldsymbol{x^n - 1}$.

Le polynôme générateur $\boldsymbol{g(x)}$ doit avoir un degré $\boldsymbol{r = n - k}$, où $\boldsymbol{k}$ est la longueur des données d'information.

Un codes cyclique est défini par un polynôme générateur $\boldsymbol{g(x)}$ de degré $\boldsymbol{r}$ et un polynôme de parité $\boldsymbol{h(x)}$ de degré $\boldsymbol{k}$, avec *:*

$$x^n+1=g(x) \,.\, h(x) \quad (8)$$

II.3.3. Codage par multiplication (non systématique)

$$c(x) = m(x) \,.\, g(x) \quad (9)$$

II.3.4. Codage par division (systématique)

$$c(x) = x^r \,.\, m(x) + CRC \quad (10)$$

Avec CRC : Le reste de la division :

$$CRC= [x^r \,.\, m(x)] \,/\, g(x) \quad (11)$$

II.3.5. Décodage par division :

$$v'(x)/g(x)= \hat{m}(x)+s(x) \quad (12)$$

$\Rightarrow$ Si $\boldsymbol{s(x)= 0}$: pas d'erreur,

- Si non il existe des erreurs, et $\boldsymbol{s(x)}$ est utilisé pour localiser et corriger ces erreurs.
- **Correction**: $\boldsymbol{v'(x)_{corrigé} = v'(x) + s(x)}$.

II.4. Codes convolutifs

Les codes convolutifs sont une classe de codes de correction d'erreurs largement utilisés dans les systèmes de communication numériques. Contrairement aux **codes en bloc**, qui traitent des blocs de données indépendants, les **codes convolutifs** introduisent une redondance en faisant dépendre les bits codés non seulement des bits actuels, mais aussi des bits précédents du message d'entrée. Cette relation "à mémoire" permet de détecter et corriger efficacement les erreurs.

Dans un code Convolutif, chaque bloc de n éléments en sortie du décodeur dépend non seulement du bloc composé des k éléments positionnés à l'entrée du décodeur mais aussi des m blocs précédents. Cette famille de codes fait donc appel à un effet de mémoire d'ordre m et la quantité (m+1) s'appelle la longueur de contrainte K du code. De même que pour les codes en blocs, la quantité R =k / n : s'appelle le rendement du code, et si les k éléments d'information présents à l'entrée du décodeur sont effectivement transmis (c'est- à - dire apparaissent explicitement dans le bloc de n éléments), le code est dit systématique.

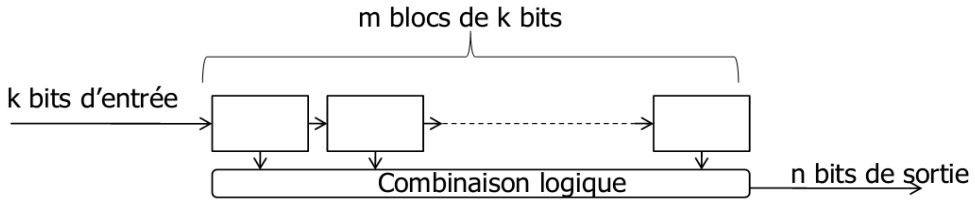


Figure 5 : Principe de base d'un code convolutif

II.4.1. Paramètres du code convolutif

Un code convolutif est caractérisé par :

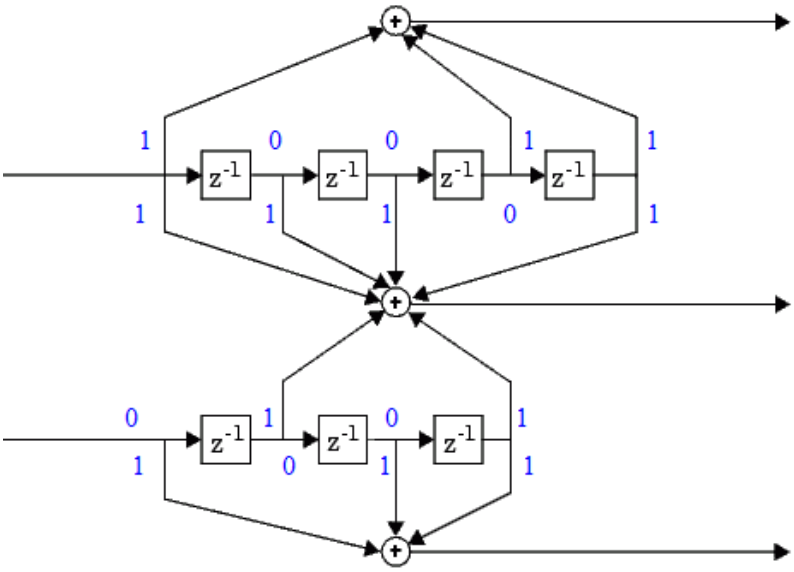
- **Taux de codage ou Rendement (*R=k/n*) :** Le rapport entre les bits d'entrée et les bits de sortie.
- **Contrainte de longueur (m) :** La profondeur de mémoire, c'est-à-dire le nombre de bits précédents qui influencent la sortie.
- **Longueur de contrainte (K=m+1) :** Nombre total de bits utilisés pour coder chaque bit d'entrée.
- **Générateurs polynomiaux ($g_1(x)$, $g_2(x)$,…, $g_n(x)$) :** Définissent les relations entre les bits d'entrée et les bits codés.
- **Schéma de codage :**

  Le codage convolutif est généralement réalisé à l'aide de registres à décalage et d'opérations **XOR**. Un schéma de codage peut être décrit par :

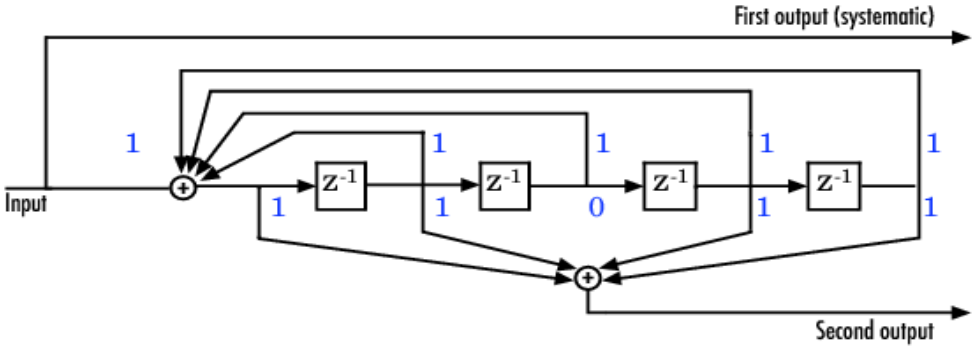
- Un **registre à décalage** contenant **m** bits de mémoire.
- **n générateurs polynomiaux** qui déterminent les combinaisons des bits d'entrée et de mémoire pour produire chaque bit de sortie.

Il existe deux types des Codes Convolutif : non récursif non systématique (NRNSC) et récursif systématique (RSC).

Exemple 1: Code convolutif (NRNSC) de Rendement R=2/3 et m=4



Exemple 2 : Code convolutif systématique récursifs (RSC) de rendement R=1/2 et m=4



II.4.2. Représentation du code convolutif

Représentations numériques :

- Transformée en D ;
- Matrice de génératrice;

Représentations graphiques :

- Diagramme d'état ;
- Arbre ;
- Treillis.

II.4.3. Décodage des codes convolutifs

La contrainte principale du décodage Convolutif réside dans le fait que le mot de code est très long, ce qui a tendance à compliquer le circuit décodeur. Les algorithmes de décodage les plus répandus sont : le **décodage de Viterbi** et le **décodage séquentiel**.
Chacune de ses techniques consiste à trouver un chemin particulier (le message transmis), dans un graphe orienté où on assigne aux branches des métriques ou valeurs de vraisemblance entre les données reçues et les données qui auraient pu être transmises.

III. Partie simulation

III.1. Code de Hamming linéaire et cyclique

III.1.1. Fonctions Matlab utiles

Le codeur de Hamming permet de corriger une erreur parmi n. Le nombre de caractère de contrôle m est lié au nombre de caractère d'information k par la relation suivante :

$$2^m \geq m+k+1=n+1 \qquad (13)$$

- MATLAB gère automatiquement les codeurs de Hamming dans le cas où $2^m = n+1$ par la fonction ***hammgen***

MATLAB propose deux fonctions ***encode*** et ***decode*** permettant d'implanter des codeurs/décodeurs de canal basés sur une matrice de contrôle ou sur un polynôme générateur.

III.1.2. Mise en œuvre

1) En utilisant la fonction ***hammgen***, tracez la courbe donnant le taux d'émission en fonction du nombre de symboles d'information pour un codeur de Hamming. Qu'en concluez-vous ?
2) Quelles sont la matrice génératrice G et de contrôle H, données par MATLAB pour un codeur de Hamming (7,4) ? Commentez la forme des matrices G et H.

- MATLAB donne la fonction ***gen2par*** qui permet de passer de G à H.

3) Calculez, en utilisant la matrice G, le mot-code associé aux symboles i= [1 0 1 0].
Quelle est la distance minimale du code?

- On pourra utiliser les fonctions Matlab suivantes: ***mod, rem, encode*** et ***gfweight***

4) Décodez le vecteur du mot-code : [1 1 0 0 1 0 1], en utilisant la matrice H.

- On pourra utiliser la fonction ***decode***

5) En utilisant le vecteur : [1 1 0 0 **0** 0 1], comme mot_code erroné

Quelles sont les conséquences pour le décodage. En déduire l'intérêt de la table retournée par la fonction ***htruthtb***. Comment va-t-on utiliser cette table ?

6) Ecrire la fonction **hamcode.m** qui retourne un mot-code à partir d'un bloc d'information et de la matrice génératrice G. On remarque que les fonctions ***rem*** et ***mod*** permettent de faire l'opération modulo.

Vérifiez que votre fonction donne le même résultat que la fonction ***encode*** de MATLAB.

7) Ecrivez la fonction **hamdecode.m** qui retourne le mot d'information corrigé à partir du mot-code reçu et de la matrice de contrôle.

- On pourra utiliser les fonctions ***htruthtb, bi2de***.

Vérifiez votre fonction ainsi que la fonction MATLAB ***decode***.

8) Finalement, utilisez un code cyclique C(7,4) (i.e. modifier les programmes précédents de façon à remplacer le code en bloc linéaire par un code cyclique C(7,4)) pour coder et détecter les erreurs éventuelles de transmission. MATLAB donne les fonctions suivantes pour travailler avec les codes cycliques : ***cyclpoly*** et ***cyclgen***. Les fonctions ***encode*** et ***decode*** sont toujours utilisables.

III.2. Codes convolutifs

III.2.1. Fonctions Matlab utiles

- La fonction **poly2trellis** de MATLAB assure la conversion de la description mathématique du code (polynômiale) en treillis, plus facilement manipulable. La fonction **poly2trellis** accepte une description polynomiale d'un encodeur convolutif et renvoie la description de structure de treillis correspondante. Cette sortie peut être utilisée comme entrée pour les fonctions du codage '**convenc'** et décodage '**vitdec'.**

  trel = poly2trellis(ConstraintLength,CodeGenerator)

  trel = poly2trellis(ConstraintLength,CodeGenerator,FeedbackConnection).
- La fonction **convenc** (convolutional encoder) réalise le codage convolutif d'une séquence binaire d'entrée msg à l'aide d'un treillis (trellis) défini par la structure **trel**.

  Le résultat est une séquence binaire codée (code) qui contient plus de bits que le message d'origine — c'est le signal redondant transmis pour permettre la détection et la correction d'erreurs.

  code = convenc(msg, trel);

- La fonction **vitdec** (Viterbi decoder) effectue le décodage du code convolutif à l'aide de l'**algorithme de Viterbi**, qui recherche la séquence la plus probable transmise à travers un canal bruité.

  decoded = vitdec(code, trel, tblen, 'trunc', 'hard');

III.2.2. Mise en œuvre

III.2.2.1. Création de la structure de treillis d'un code convolutif

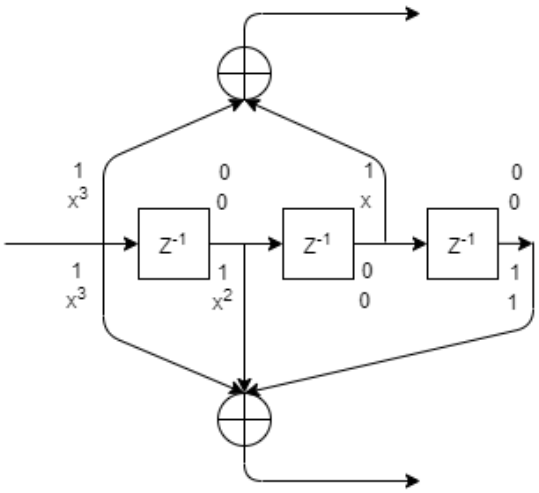
1) Créez une structure en treillis, en définissant le rendement, la longueur de la contrainte et en spécifiant les générateurs des codes en tant que vecteurs de valeurs octales, du code convolutif NSNRC du schéma au-dessous, en relevant les sorties de la fonction **poly2trellis** dans la fenêtre de commande.

2) Utilisez ensuite les fonctions associées à la structure `treillis` pour accéder aux états suivants (Next State) et aux sorties (Out put) :

```
>>trel.nextStates
>>trel.outputs
```

Tapez les commandes suivantes pour visualiser les états suivants et les sorties du code

```
>> commcnv_plotnextstates(trel.nextStates);
>> commcnv_plotoutputs(trel.outputs, trel.numOutputSymbols);
```



NB : Conversion binaire en octal :

1. **Regroupez les bits binaires** par groupes de 3, en commençant par la droite. Si le dernier groupe contient moins de 3 bits, ajoutez des zéros à gauche pour compléter le groupe.
2. **Utilisez une table de conversion** pour obtenir l'équivalent octal de chaque groupe de 3 bits binaires.

Exemple :

Pour le nombre binaire `1110`, vous obtenez les groupes suivants :

```
001 | 110
```

Les équivalents octaux de ces groupes sont : `001` → 1 ; `110` → 6

Ainsi, le nombre binaire `1110` est équivalent à **16** en notation octale.

3) Tapez : **trel = poly2trellis([5 4],[23 35 0;0 5 13])**

- Quelles sont les caractéristiques n, k, K et les générateurs (en binaire) de ce codeur ?
- Si on traçait le graphe modélisant les états de ce codeur, combien y aurait-il d'états ? Combien de flèches partiraient de chaque état ?
- Donner le schéma de ce codeur.

4) Examiner dans la fenêtre de commande les messages d'erreur en sorties des fonctions :

```
>>treillis=poly2trellis(2,[7,5])
>>treillis=poly2trellis(4,[7,5])
```

Quelle est la longueur de contrainte du code ? Comment sont définis les polynômes générateurs pour la fonction **poly2trellis**?

5) Dans la fenêtre de commande MATLAB, récupérez les sorties de la fonction suivante :

```
>> treillis = poly2trellis(3, [7, 5])
```

Utilisez ensuite les fonctions associées à la structure `treillis` pour accéder aux états suivants et aux sorties :

```
>>treillis.nextStates
>>treillis.outputs
```

Tapez les commandes suivantes pour visualiser les états suivants et les sorties du code

```
>> commcnv_plotnextstates(treillis.nextStates);
>> commcnv_plotoutputs(treillis.outputs, treillis.numOutputSymbols);
```

Tracez théoriquement le treillis du code convolutif NSNRC de rendement 1/2 et des polynômes générateurs en notation octale (7,5).

Comparez ensuite les soties obtenues à partir de ce treillis avec celles du codeur NRNSC (3,[7,5]), obtenues lors des étapes précédentes. Que constatez-vous ?

III.2.2.1. Codage et décodage par le code convolutif NSNRC(3,[7,5])

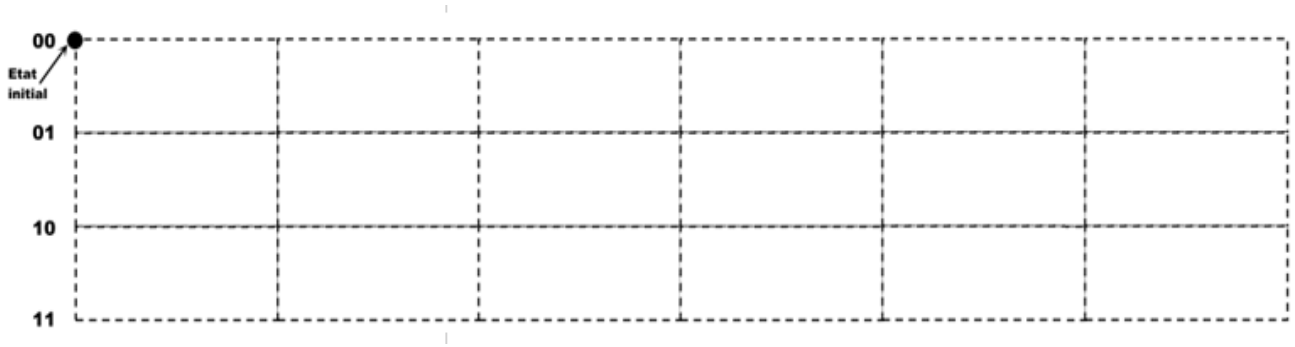
1) Saisir le script ci-dessous et relever ses sorties.

```
clc; clear all;close all;
K=3;
G1=7;
G2=5;
msg=[1 1 0 0 1 0]
```

```
trel=poly2trellis(K,[G1,G2]);
code=convenc(msg,trel)
tblen=length(msg);
decoded=vitdec(code,trel,tblen,'trunc','hard')
```

2) Justifier le résultat du codage obtenu avec la fonction convenc. Utilisez le schéma du treillis précédent pour expliquer les transitions d'états et la génération des bits codés.

3) Justifier le résultat du décodage par application de l'algorithme de Viterbi. Utilisez le schéma du treillis précédent.



IV. Partie SIMULINK

IV.1. Modèle 1 : Transmission sur un canal binaire symétrique (CBS)

Nous allons maintenant simuler la transmission d'un grand nombre de symboles à travers un canal binaire symétrique (CBS) afin d'étudier les probabilités d'erreur et leur évolution en fonction de la qualité du canal, avec et sans l'utilisation du codage de Hamming (7,4).

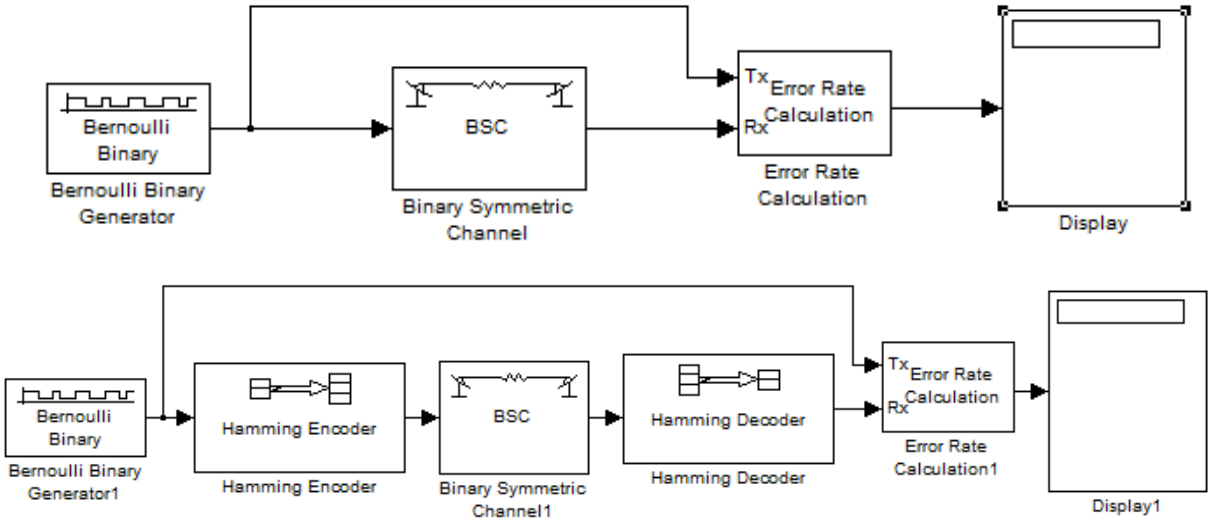
1) Réaliser le deux modèles SIMULINK ci-dessous.

Le modèle est constitué par :

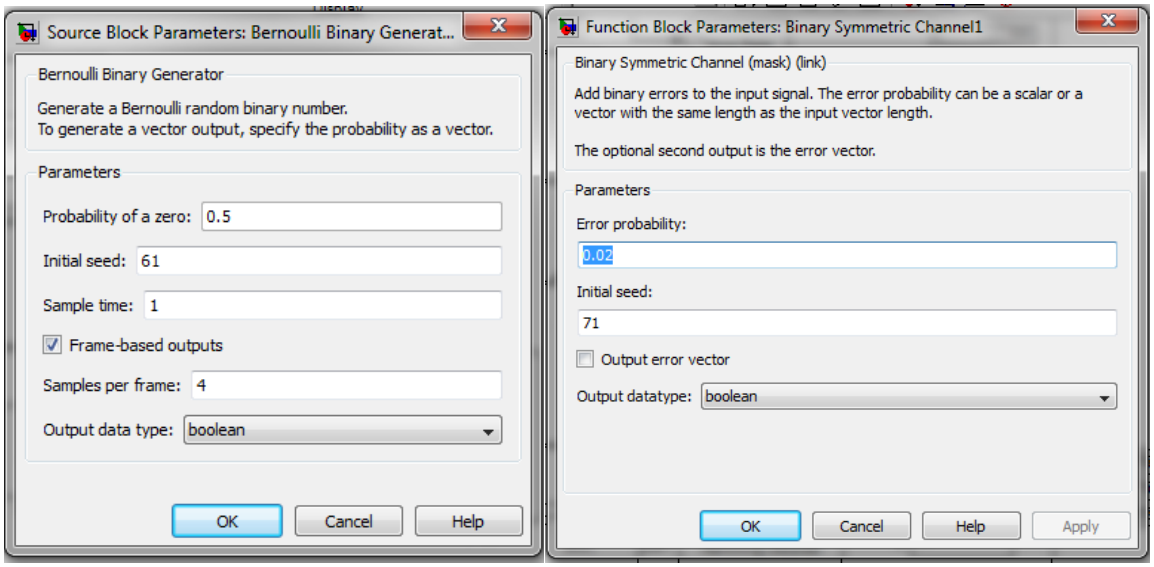
- Un générateur des données aléatoires « Bernoulli Binary Generator ».
- Un canal binaire symétrique « BSC ».
- Un codeur/décodeur de Hamming.
- Un élément de mesure du taux d'erreur binaire « Error Rate Calculation » et un élément de perception « Display ».

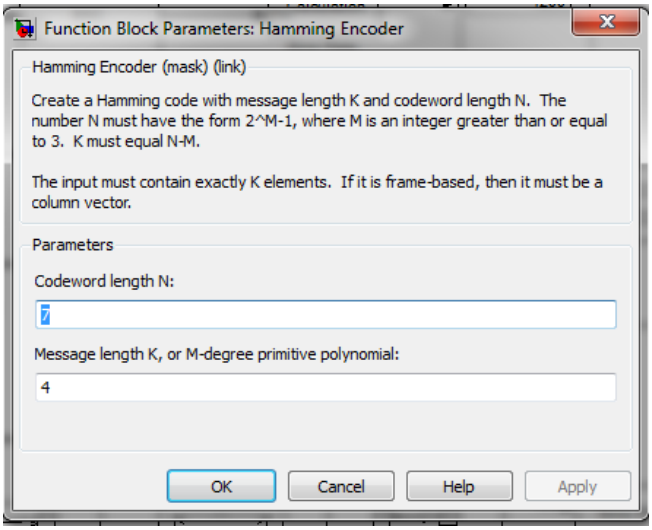
Paramètres de simulation :

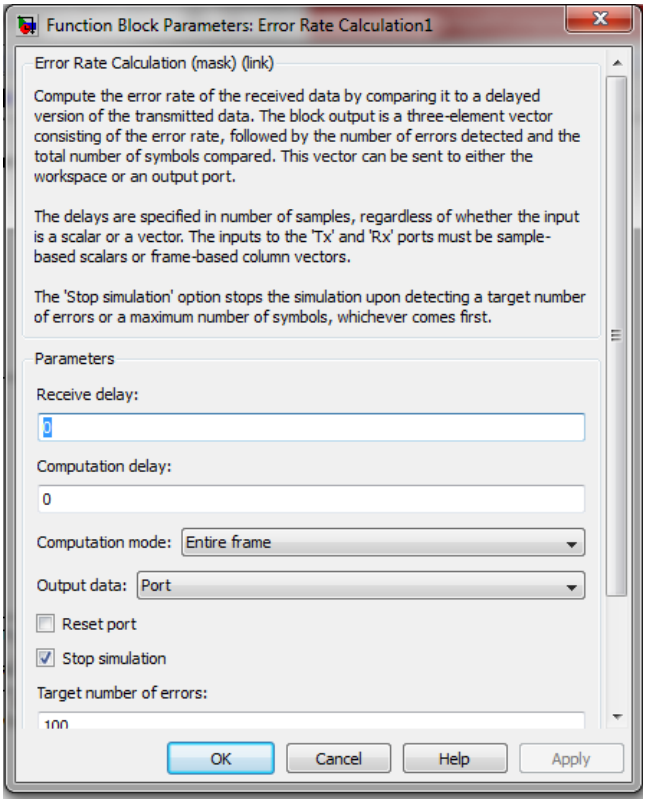
- Bernoulli Binary Generator: Probability of a zero = 0.5, Samples per frame =4
- Hamming Encoder and decoder: Codeword length N = 7, Message length K = 4.
- BSC channel : Error probability = 0.02
- Error-Rate Calculation : Maximum number of symbols 1e6



2) Identifier les différents blocs et regardez leurs paramètres en double clickant dessus, en modifiant les paramètres selon les figures ci-dessous. Lancez la simulation et observez les fenêtres qui s'ouvrent.







3) Paramétrer le taux d'erreur du canal pour qu'il soit juste égal à la capacité de correction du codeur de Hamming implanté. Lancer la simulation. Transmetter 1000 symboles. Donner le nombre d'erreurs non-corrigées, le taux d'erreur après correction. Pourquoi toutes les erreurs n'ont elles pas été corrigées ?

4) Réaliser plusieurs simulations avec différentes probabilités d'erreur du canal (p = 0.01, p = 0.1, etc.). Comenter votre résultats.

5) Comparer les performances du système avec et sans codage de Hamming en termes de taux d'erreur binaire (BER). Analyser les résultats obtenus et discuter de l'importance du codage de canal dans l'amélioration des performances de la transmission numérique.

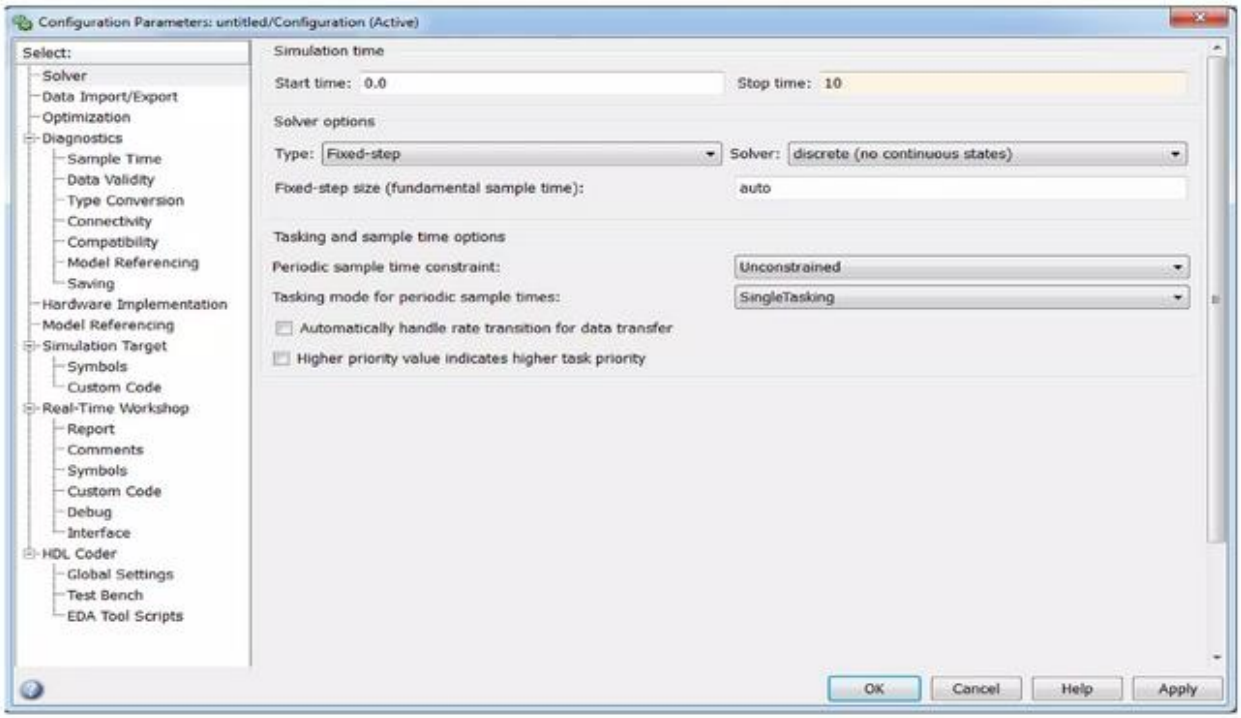
6) Refaire le même travail que précédemment en remplaçant le codeur Hamming (7,4) d'abord par un codeur Hamming (15,11), puis par un codeur BCH (15,5). Comparer les performances des différents codes: Hamming (7,4), Hamming (15,11) et BCH (15,5) en termes de taux d'erreurs binaires (BER). Analyser et comparez les performances de ces codes correcteurs d'erreurs, en tenant compte de leur capacité à corriger des erreurs et de leur efficacité pour différents niveaux de bruit dans le canal de transmission.

Remarque : Configuration de Simulink

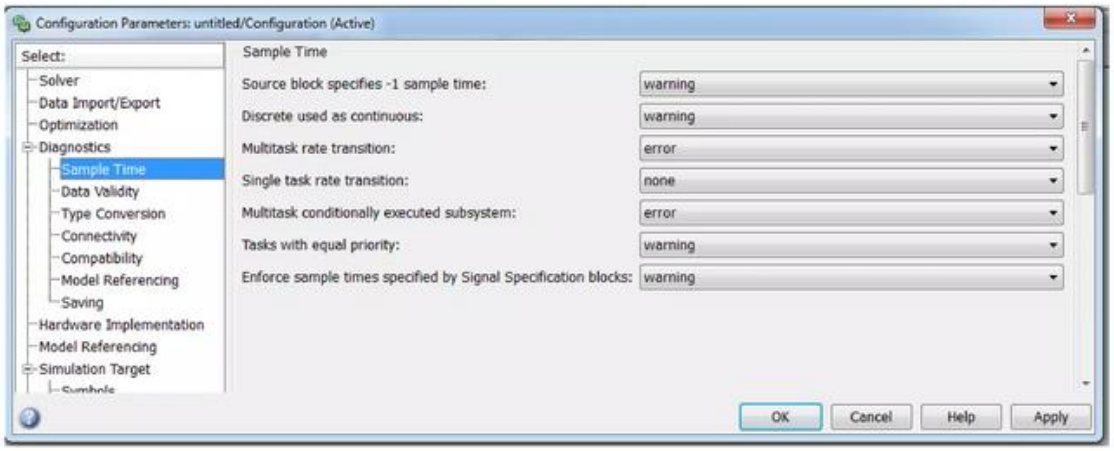
Pour créer un modèle avec Simulink en domaine discret, il faut le configuré de la façon suivante :

- Créer un fichier **Simulink**.
- Aller dans **Simulation** puis **Configuration Paramètres** (ou cliqué sur Ctrl+E) de chaque nouveau modèle, et régler les paramètres comme il est marqué dans cette capture d'écran :

▪ Aller dans **Solver** :



- Puis aller dans **Diagnostic/Sample time** :



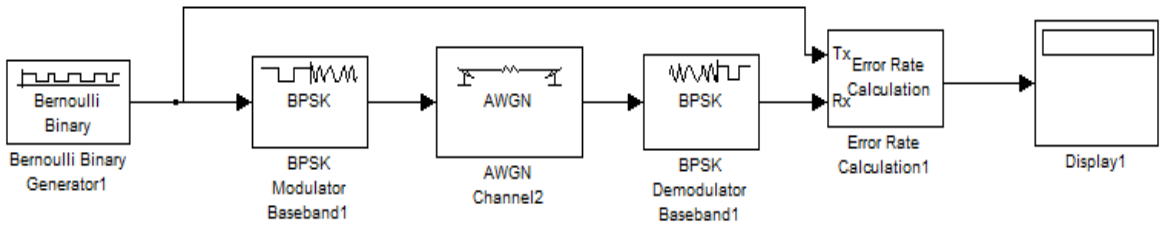
IV.2. Modèle 2 : Transmission sur un canal à bruit blanc gaussien additif (AWGN)

Le deuxième modèle repose sur la transmission d'un grand nombre de symboles à travers un canal à bruit blanc gaussien additif (AWGN). Ce modèle inclut des outils de codage de source « BPSK Modulator Baseband » et de codage de canal afin d'évaluer l'efficacité et les performances des différents codeurs de canal : le code cyclique (31,21) et le code Reed-Solomon (RS).

Le schéma Simulink pour le canal AWGN doit inclure les blocs suivants, avec quelques différences par rapport au modèle précédent avec canal BSC. Le modèle est constitué par :

- Un générateur des données aléatoires « Bernoulli Binary Generator »,
- Un bloc de bande de base du modulateur « BPSK Modulator Baseband »: qui convertit les bits unipolaires (0 et 1) en valeurs bipolaires analogiques bipolaires. Cette étape est nécessaire car le canal AWGN modélise un bruit gaussien sur un signal analogique,
- Un bloc : « AWGN Channel » qui ajoute du bruit gaussien au signal analogique.
- Un bloc : « Bipolar to Unipolar Converter » pour récupérer les bits.
- Un élément de mesure du taux d'erreur binaire « Error Rate Calculation » et un élément de perception « Display ».

1) Réaliser le modèle SIMULINK (sans codage canal) suivant :



Paramètre des blocs :

- Probability of a zero=0.5
- Initial seed=61
- Sample time=1
- Output data type=Double
- Receive delay=0
- Computation delay=0
- Target number of errors=100
- Maximum number of symbols=1e6

a) Lancer la simulation et observer et comenter les resultats obtenus.

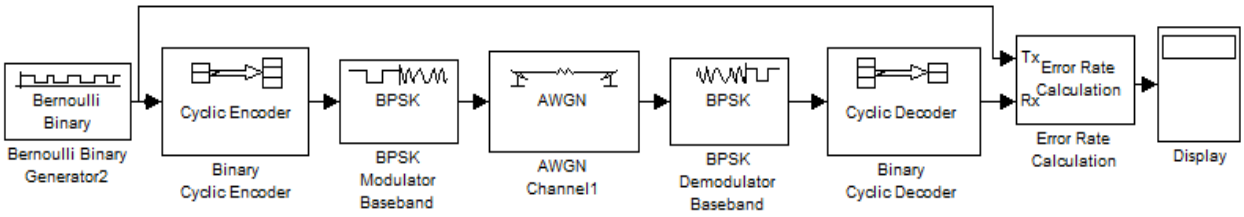
b) Remplisser le tableau ci-dessous en utilsant les paramètres indiqués dans le tableau suivant :

	Error probability	Line seed	SNR
Paramètre 2	0.01	2137	4.2
Paramètre 3	0.01	10	4.2
Paramètre 4	0.01	10	7
Paramètre 5	0.01	10	3

	BER	Number of errors	Number of transmitted bits
Paramètre 2			
Paramètre 3			
Paramètre 4			
Paramètre 5			

c) Discuter les résultats obtenus. Conclusion.

2) Réaliser et configurer le modèle ci-dessous en y ajoutant deux blocs « Cyclic Encoder » et « Cyclic Decoder » permettent de corriger en réception d'éventuelles erreurs de transmission. Si le code cyclique a une longueur de message K et une longueur de mot de code N, alors N doit avoir la forme 2^M-1 pour un entier M supérieur ou égal à 3.



Paramètres de configuration :

- Bernoulli Binary Generator: Probability of a zero = 0.01 , Samples per frame =21
- Binary Cyclic Encoder/decoder: Codeword length N = 31, Message length K = 21.
- AWGN channel : Mode = Signal to Noise Ratio (Eb/No), Symbol period = 21/31
- Error-Rate Calculation : Maximum number of symbols 1e7

a) Exécutez la simulation pour certains niveaux de bruit.

b) Que constatez-vous

c) On sait que : SNR=10*log ($P_S/P_{B)}$

On prend P_S=1 pour simplifier les calculs ; donc $P_B=10^{-SNR/10}$

Remplissez le tableau suivant :

SNR	P_B	BER
-5		
0		
5		
10		
15		
20		
25		

d) Tracer le TBE en fonction du SNR. Comenter votre résultats.

e) Conclure en comparant les deux modèles du canal AWGN avec et sans le codage de canal.

3) Refaire le même travail, en implémentant un encodeur/décodeur RS (Reed-Solomon). Tracer, sur le graphique précédent, le TEB en fonction du SNR. Commenter votre résultat.

TP4 : Implémentation de la DCT rapide à faible complexité arithmétique

I. Objectif du TP

- Implémenter et optimiser la Transformée en Cosinus Discrète (DCT) rapide en MATLAB.
- Evaluer les performances pour transformer des images en 2 dimensions.

II. Rappels théoriques

La transformée en cosinus discrète (DCT : *Discrete Cosine Transform*) est une transformation proche de la transformée de Fourier discrète (DFT). Le noyau de projection est un cosinus et crée donc des coefficients réels, contrairement à la DFT, dont le noyau est une exponentielle complexe et qui crée donc des coefficients complexes.

La DCT est largement utilisée pour la compression de données, notamment dans les formats JPEG et MPEG pour les images et qui utilisent une DCT 2D (bidimensionnelle) sur des blocs de pixels de taille 8×8 (pour des raisons de complexité). Cette transformée offre l'avantage de concentrer la majeure partie de "l'énergie" de l'image ou l'information, dans quelques composants de fréquence (coefficients basses fréquences), idéaux pour le codage.

Il existe plusieurs variante de la DCT, ceux les plus connus sont : la Transformée en Cosinus Discrète unidimensionnelle (DCT I) et bidimensionnelle (DCT II ou DCT 2D) et sa transformé inverse IDCT, nous allons voir la définition de la DCT I, DCT II qui sont la base de la compression JPEG.

II.1. La DCT I

Partons d'une courbe d'équation *y = f(t)*, que l'on remplace par une succession « discrète » de points de coordonnées *(n , f(n))* avec *n* variant de *0* à *N – 1*, ce qui donne *N* points de cette courbe. A partir de là, on calcule sa transformée en cosinus, par la formule :

$$DCT(k) = \frac{2c(k)}{N} \sum_{n=0}^{N-1} f(n) \cos \frac{(2n+1)k\pi}{2N} \qquad (1)$$

$k = 0,1,...,N-1$

$avec : c(0) = 1/\sqrt{2} \;\; et \;\; c(k) = 1 \;\; pour \;\; k \neq 0$

L'intérêt de cette transformation est qu'elle est inversible, et qu'après l'avoir inversée, on retrouve le signal initial, grâce à cette formule :

$$f(n) = \sum_{k=0}^{N-1} c(k) DCT(k) \cos \frac{(2n+1)\pi}{2N} \qquad (2)$$

II.2. La DCT II

Dans la norme de JPEG, la transformée en cosinus discrète est en fait appliquée par bloc de 8x8 pixels dont le nombre de lignes et le nombre de colonnes de l'image entière sont chacun des multiples de 8.

$$DCT(i,j) = \frac{2}{N} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} P(x,y) \cos\left(\frac{(2x+1)i\pi}{2N}\right) \cos\left(\frac{(2y+1)j\pi}{2N}\right) \qquad (3)$$

– N : la largeur d'un bloc, ici N=8.

– i, j : les indices d'un coefficient de la DCT dans un bloc.

– x, y : les indices d'un pixel de l'image dans un bloc

– $DCT(i,j)$: la valeur d'un coefficient dans un bloc.

– $C(x) = \frac{1}{\sqrt{2}}$ si $x = 0$, $C(x) = 1$ sinon.

–$P(x,y)$: **la valeur du pixel aux coordonnées** (x,y).

La transformée inverse est donnée par :

$$P(x,y) = \frac{2}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} C(i)C(j)DCT(i,j) \cos\left(\frac{(2x+1)i\pi}{2N}\right) \cos\left(\frac{(2y+1)j\pi}{2N}\right) \qquad (4)$$

II.3. La DCT Rapide

L'algorithme de la DCT rapide est conçu pour optimiser le calcul de la DCT en diminuant le nombre de multiplications et d'additions requises par rapport à la méthode directe. Nous faisons l'hypothèse que l'image est carrée, de dimension *N*N*. Cela nous permet d'exploiter la propriété de symétrie, ce qui conduit à une expression simplifiée pour le calcul de la DCT :

$$\underline{F} = \underline{C}_{NN} \; \underline{f} \; \underline{C}_{NN}^{T} \qquad (5)$$

Où « *f* » est la matrice image de dimension *N*N* et « *F* » est sa transformée DCT.

« *C* » c'est une matrice *N*N*, ces éléments $C_{NN}(k,l)$ se calcule par la formule suivante :

$$\underline{C}_{NN}(k,l) = \begin{cases} \frac{1}{\sqrt{N}} & l = 0 \\ \sqrt{\frac{2}{N}} \cos\left[\frac{(2k+1)l\pi}{2N}\right] & \text{sinon} \end{cases} \qquad (6)$$

A ce stade, on peut on peut conclure par la formule d'orthogonalité : $C^{-1} = C^{T}$, qui nous permet d'obtenir l'IDCT selon la formule suivante :

$$\underline{f} = \underline{C}_{NN}^{T} \; \underline{F} \; \underline{C}_{NN} \qquad (7)$$

Ces propriétés réduisent considérablement les calculs dans les algorithmes de compression d'image du fait que ces éléments seront déjà calculés à l'avance (calcul de la matrice *C*).

II.4. Propriétés

- ✓ **Orthonormalité**: Les fonctions de base de la DCT sont orthonormées (orthogonales).
- ✓ **Compacité** : La DCT est efficace pour concentrer l'énergie du signal dans quelques coefficients.

III. Parties pratiques

III.1. Implémentation de la DCTI et la DCTII par Matlab

1) Appliquer les fonctions ci-dessous (**dct_1d_fast.m** et **dct_2d_fast.m**) pour calculer les coefficients DCT du vecteur x et d'image I :

```
x = [1, 2, 3, 4];
I = [1, 2, 3, 4;
     5, 6, 7, 8;
     9, 10, 11, 12;
     13, 14, 15, 16];
```

2) Refaire le même calcul, mais cette fois en utilisant les fonctions ***dct*** et ***dct2*** de Matlab. Que constatez-vous ?

3) Comparer le temps d'exécution de la DCT rapide avec la DCT de Matlab. Utiliser les fonctions ***tic*** et ***toc*** de MATLAB pour mesurer les temps d'exécution.

4) Donner le code MATLAB : **dct_1d_inv_fast.m** pour implémenter l'IDCT I (inverse).

5) Donner le code MATLAB : **dct_2d_inv_fast.m** pour implémenter l'IDCT II (inverse).

a) Le code MATLAB **dct_1d_fast.m** pour implémenter la DCTI :

```
function X = dct_1d_fast(x)
    N=length(x);
    X=zeros(1,N);
    for k=1:N
        sum_val=0;
        for n=1:N
            sum_val=sum_val+x(n)*cos(pi*(n-0.5)*(k-1)/N);
        end
        X(k)=sum_val*sqrt(2/N); % Normalisation
        if k==1
```

```
            X(k)=X(k)/sqrt(2);% Normalisation du premier
coefficient
        end
    end
end
```

b) Le code Matlab **dct_2d_fast.m** pour implémenter la DCTII utilisant la fonction **dct_1d_fast.m** précédente :

```
function Y = dct_2d_fast(X)
    [M,N]=size(X);
    Y=zeros(M, N);
    % DCT sur les lignes
    For i=1:M
        Y(i,:)=dct_1d_fast(X(i,:));
    end
    % DCT sur les colonnes
    for j=1:N
        Y(:,j)=dct_1d_fast(Y(:,j))';
    end
end
```

III.2. Transformation d'une image ligne par ligne par la DCTI

Ecrire un script permettant d'effectuer les tâches suivantes :

1) Charger une image en niveaux de gris : Utilisez la fonction ***imread*** pour importer une image en niveaux de gris de votre choix, du toolbox Matlab. Vérifiez que l'image est bien en niveaux de gris avec la fonction ***imshow***. Si non, vous pouvez utiliser une image couleur, puis la transformée en niveau de gris avec la fonction ***rgb2gray***. L'image chargée doit être automatiquement affectée à la variable ***img_gray***.

Instructions Matlab à tester :

```
close all; clear al; clc;
I= imread('peppers.png');
Taille_I=size(I);
img_gray = rgb2gray(I); % Convertir en niveaux de gris si nécessaire
Taille_Img=size(img_gray);
imshow(img_gray);
title('Image Originale');
```

2) Normaliser l'image : Avant d'appliquer la DCT I, normalisez les valeurs des pixels (par exemple, en les échelonnant entre 0 et 1). Utilisez l'instruction suivante:

```
img_normalized = double(img_gray) / 255;
```

3) Appliquer la DCT ligne par ligne : Écrivez une boucle qui parcourt chaque ligne de l'image et appliquez la DCT I à cette ligne. Utilisez la fonction ***dct*** de Matlab.

4) Appliquer la IDCT ligne par ligne : Effectuez l'inverse de la transformée (IDCT I), en parcourant l'image transformée ligne par ligne. Utilisez la fonction ***idct*** de Matlab

5) Afficher les résultats : Affichez les trois images dans la même figure : L'image normalisée, images après la DCT et l'image reconstituée après l'IDCT. Que constatez-vous ?

Instructions Matlab à tester :

```
%Affichafe des résultat
figure (1);
subplot(1,3,1);
imshow(img_normalized);
title('Image originale');
subplot(1,3,2);
imshow(dct_result1);
title('Image après DCT1');
subplot(1,3,3);
imshow(img_reconstructed);
title('image Reconstituée après IDCT1');
```

6) Zoomez sur les coins inférieurs gauches jusqu'à ce que les pixels des bords des deux images, l'originale et la reconstituée, apparaissent. Que remarquez-vous ?

7) Interprétez et discutez les résultats des images obtenues. Concluez.

8) Quelle sont les inconvénients de l'application de la DCT I aux images en niveaux de gris ?

III.3. Transformation d'images entières avec l'application de la DCT II

Modifier le script précédent par les tâches suivantes :

1) Appliquer la DCT II et l'IDCT II : Effectuez la DCT II sur l'image entière, puis appliquez l'IDCT II. Utilisez les fonctions ***dct2*** et ***idct2*** de Matlab.

2) Afficher les résultats : Affichez les trois images : l'originale, la transformée et la reconstituée.

3) Zoomez sur les coins inférieurs gauches jusqu'à ce que les pixels des bords des deux images, l'originale et la reconstituée, apparaissent. Que remarquez-vous ?

4) Interprétez et discutez les résultats des images obtenues. Comparez avec l'application de la DCT I et concluez.

III.4. Transformation d'images par blocs avec l'application de la DCT II

Dans la norme de JPEG, la transformée en cosinus discrète est en fait appliquée par bloc de 8x8 pixels dont le nombre de lignes et le nombre de colonnes de l'image entière sont chacun des multiples de 8.

Ecrire un script Matlab permettant d'effectuer les tâches suivantes :

1) Charger une image en niveaux de gris.
2) Découper ensuite l'image en blocs disjoints de dimension 8x8 puis effectuer la transformée en DCT II à chacun de ces blocs.
3) Afficher les trois images, l'originale, l'image transformée et l'image reconstituée par l'application de l'IDCT II.
4) Comparez visuellement les images obtenues. Où se trouvent les coefficients élevés et les coefficients faibles ? Commentez les différences et comparez avec les résultats précédents.
5) Pourquoi est-il important de travailler avec des blocs de taille fixe lors de l'application de la DCT II?
Remarque : L'instruction ***blkproc*** permet de découper une image en plusieurs blocs de taille [M N]. Ensuite les fonctions ***dct2*** ou ***idct2*** sont appliquées à chacun de ces blocs. Utiliser la fonction ***mesh*** pour visualiser les coefficients DCT en 3D.

Instructions Matlab à tester :

```
d1=blkproc(I,[8 8],'dct2');
d8x8decomp=blkproc(d1,[8 8],'idct2');
```

III.5. Application de la DCT rapide

1) Modifiez le script précédent en y ajoutant le calcul des coefficients DCT par la DCT rapide, sous la forme simplifiée donnée par les équations (5) et/ou (7). Pour cela, utilisez la fonction ***dctmtx*** de MATLAB.
2) Évaluez les performances en termes de temps de calcul et de précision des coefficients DCT pour les deux méthodes. Utilisez les fonctions ***tic, toc*** de Matlab. Que remarquez-vous?
3) Présentez les résultats des tests et analysez les performances en termes de complexité et de temps de calcul. Conclusion

TP5 : Implémentation sous Matlab de la méthode de compression d'images JPEG

I. Objectif du TP

La mise en œuvre d'une méthode de compression d'images de type JPEG (Joint Photographic Expert Group). Afin de faciliter l'implémentation et de mettre en évidence les principales étapes du processus de compression et de décompression, certaines simplifications pourront être apportées par rapport à la norme JPEG complète.

Les principales étapes de l'étude sont les suivantes :

- ✓ Appliquer la transformée en cosinus discrète (DCT) sur des blocs de l'image.
- ✓ Quantifier les coefficients DCT pour simuler l'étape de compression.
- ✓ Reconstruire l'image compressée à partir des coefficients quantifiés.
- ✓ Évaluer les performances du système en termes de taux de compression et de PSNR (*Peak Signal-to-Noise Ratio*) en fonction du niveau de qualité choisi.

II. Rappels théoriques

L'acronyme **JPEG** (**Joint Photographic Expert Group**) provient de la réunion en 1982 d'un groupe d'experts de la photographie, dont le principal souci était de compresser des images au format RGB en images de taille beaucoup moins importante mais dont la qualité n'est que peu affectée. Ceci est particulièrement important en ce qui concerne la transmission d'images par Internet et dans le stockage des données. L'algorithme standard, établit en 1991 est basé principalement sur le codage par **la transformé en cosinus discrète DCT**.

Le processus de Compression/Décompression JPEG est :

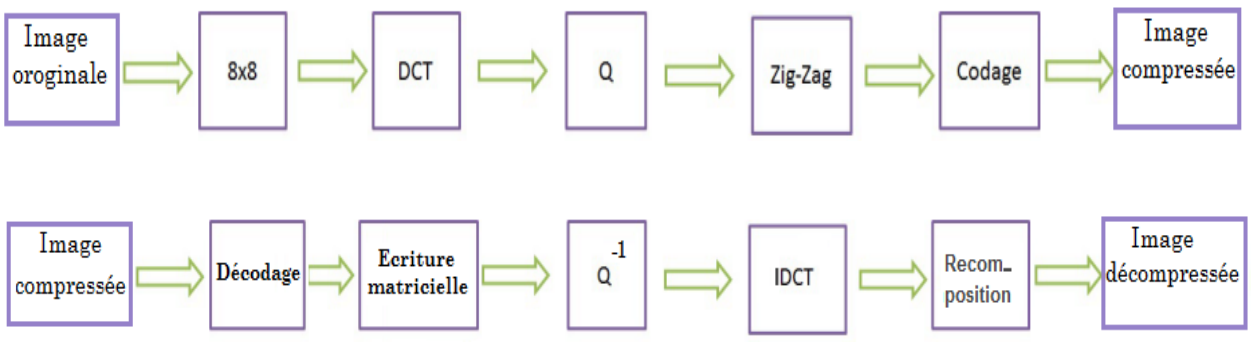


Figure 1 : Processus de Compression/Décompression JPEG

Les étapes principales du codage JPEG sont :

- Transformation du format RGB en Luminance / Chrominance (YCbCr) : L'œil humain est plus sensible à la luminosité de la couleur qu'à la valeur chromatique d'une image.
- Ré-échantillonnage de la chrominance.
- Découpage en blocs 8*8.
- TCD2D (Transformée en Cosinus Discret en deux dimensions) sur les blocs 8*8 : on l'utilise pour séparer les basses et les hautes fréquences et pour la concentration de l'énergie dans les basses fréquences (Coefficients DC).
- Quantification : Pour éliminer les hautes fréquences qui sont moins visibles par l'œil (la plus part des coefficients AC seront arrondis à l'entier prêt qui est souvent 0).
- Lecture Zigzag : Pour la mise en série des données.
- Codage sans pertes : Codage de manière optimale (Codage DPCM pour les coeff. DC, codage RLE pour les coeff. AC et codage de Huffman pour les deux).

III. Partie simulation

III.1. Compression/Décompression d'une image en niveau de gris

L'objectif de cette partie est d'écrire un code MATLAB permettant la compression et la décompression d'une image en niveaux de gris, en suivant les étapes décrites ci-dessous.

III.1.1. Préparation de la compression

1) Charger l'image RGB **'peppers.png'** du toolbox Matlab.

2) Effectuer la transformation en niveau de gris avec la fonction ***rgb2gray***.

Les instructions MATLAB suivantes peuvent être utilisées pour développer le programme :

```
% Chargement de l'image RGB
RGB = imread('peppers.png');
% Conversion en niveaux de gris
I = rgb2gray(RGB);
% Affichage des deux images
figure;
subplot(1,2,1); imshow(RGB); title('Image originale (RGB)');
subplot(1,2,2); imshow(I); title('Image en niveaux de gris');
```

III.1.2. Application de la DCT II

1) Découper ensuite l'image en blocs disjoints de dimension 8x8 puis effectuer la transformée en DCT II (DCT 2D) à chacun de ces blocs.

L'instruction **blkproc** permet de découper une image en plusieurs blocs de taille [M N]. Ensuite la fonction **dct2** est appliquée à chacun de ces blocs. Utiliser les fonctions **idct2** et **imshow** pour visualiser les images et **mesh** pour visualiser les coefficients DCT en 3D.

Instructions Matlab à tester:

```
%Application de la dct par bloc de 8x8 pour Y
d8x8=blkproc(I,[8 8],@dct2);
d8x8decomp=blkproc(d8x8,[8 8],@idct2);
figure(2);subplot(2,2,1);imshow(I,gray(256));
subplot(2,2,2);imshow(d8x8,gray(256));
subplot(2,2,3);mesh(d8x8);
subplot(2,2,4);imshow(d8x8decomp,gray(256));
```

III.1.3. Vérification de la concentration de l'énergie en basse fréquences (Coeff. DC)

1) Représenter l'énergie moyenne de chaque bloc : Pour cela, on crée une fonction **energie.m** qui calcule la moyenne du carré des coefficients de la DCT.

2) Analyse de l'image après suppression des coefficients AC : On crée une fonction **filtpb.m** permettant de ne conserver que les coefficients DC dans chaque bloc de 8×8.

- On calcule l'énergie moyenne avant et après l'application de **filtpb.m** à l'aide de la fonction **energie.m.**
- On affiche le résultat obtenu après le calcul de la transformée inverse (**IDCT-II**).
- On compare les deux images obtenues.

3) Évaluation du pourcentage d'énergie : On identifie le pourcentage moyen d'énergie contenu dans les coefficients DC.

4) Extension du filtrage : De la même manière, on réalise un filtrage en conservant les coefficients DC ainsi que les trois premiers coefficients AC voisins du DC.

6) Qu'observez-vous ? Discutez de vos résultats. Concluez.

Instructions Matlab à tester :

```
%concentration de l'énergie  en basses fréquences après
filtpb(x)
d2=blkproc(d1,[8 8],'filtpb');
```

```
d3=blkproc(d2,[8 8],'idct2');  % dct inverse
figure (3);subplot (2,1,1);imshow(d3,gray(256));
subplot (2,1,2);mesh(d2);
%on mesure l'energie moy par function z=energie(x)
d3=blkproc(d1,[8 8],'energie');
cofACDC=mean(mean(d3))
%on mesure l'energie DC
d4=blkproc(d2,[8 8],'energie');
cofDC=mean(mean(d4))
%energie en %
Ppourcentag=cofDC*100/cofACDC
%on affiche l'energie au fur à mesure avec plus de détail
mask = [1   1   0   0   0   0   0   0
        1   1   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0];

d5=blkproc(d1,[8 8],'P1.*x',mask');
d6=blkproc(d5,[8 8],'idct2');
figure (4);subplot (2,1,1);imshow(d6,gray(256));subplot
(2,1,2); mesh(d5);
d7=blkproc(d5,[8 8],'energie');
cofDC3AC=mean(mean(d7))
%qualité de l'image
Ppourcentag=cofDC3AC*100/cofACDC
```

a) La fonction MATLAB : `energie.m`

```
function z=energie(x)
    xcarr=x.^2;
    z=mean(mean(xcarr)); %moyenne du carré de l'entrée
end
```

b) La fonction MATLAB : `filtpb.m`

```
function z=filtpb(x)
```

```
    z=zeros(8,8);
    z(1,1)=x(1,1); % seul premier composant DC
end
```

III.1.4. Quantifications des coefficients de la DCT

Pour des images en niveaux de gris, la norme JPEG consiste d'abord transformer les niveaux de gris en valeurs entre -128 et 127, ensuite à calculer par bloc de 8x8 les coefficients de la DCT, et enfin à quantifier ces coefficients avec un pas de quantification qui dépend de la position du pixel à l'intérieur du bloc 8x8. La matrice de quantification Q est donnée par la relation suivante :

$$Q(i,j) = 1 + F.(i + j - 1), \text{ où } F \text{ désigne le facteur de qualité } (F>0).$$

Cette étape permet d'éliminer les composantes de haute fréquence, moins perceptibles visuellement.

1) Appliquez une matrice de quantification standard JPEG sur les coefficients DCT en divisant la matrice des coefficients par la table de quantification donnée par la matrice Q50 qui correspond à un facteur de qualité FQ=50% :

```
Q50= [16 11 10 16 24 40 51 61 ;
12 12 14 19 26 28 60 55 ;
14 13 16 24 40 57 69 56 ;
14 17 22 29 51 87 80 62 ;
18 22 37 56 68 109 103 77 ;
24 35 55 64 81 104 113 92 ;
49 64 78 87 103 121 120 101 ;
72 92 95 98 112 100 103 99] ;
```

Il est possible de modifier le compromis qualité de la compression/mémoire requise en modifiant la matrice de quantification Q par l'introduction d'un facteur de qualité FQ qui varie entre 10 et 90

Instructions Matlab à tester :

```
FQ = input('What quality of compression you require - FQ: ');
if FQ > 50
     Q = round(Q50.*(ones(8)*((100-FQ)/50)));
     Q = uint8(Q);
 elseif FQ < 50
     Q = round(Q50.*(ones(8)*(50/FQ)));
     Q = uint8(Q);
 elseif FQ == 50
     Q = Q50;
end
```

III.1.5. Décompression et Reconstruction

Cette étape consiste à déquantifier et à reconstruire l'image à partir des coefficients quantifiés.

1) Effectuer la déquantification, en multipliant les coefficients quantifiés par la même matrice Q utilisée précédemment.

2) Reformez l'image complète à partir des blocs reconstitués en appliquant l'inverse de la DCT (IDCT) à l'aide de la fonction ***idct2*** pour reconstruire les blocs.

3) Affichez l'image d'origine et sa version compressée/décompressée. Comparez l'image originale et l'image compressée en termes de qualité visuelle.

4) Mesurez la qualité de l'image décompressée en termes de EQM (Erreur Quadratique Moyenne) et de PSNR (Peak Signal-Noise Ratio) en visualisant les effets de la qualité sur la compression (par exemple pour **FQ= 10 : 20 : 90**). Que remarquez-vous?

Le PSNR est donné par la formule :

$$PSNR = 10\log_{10}\left(\frac{(2^{bit}-1)^2}{EQM}\right) \quad (1)$$

Où, bit=8 et l'EQM est donnée par :

$$EQM = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left\|I_0(i,j) - I_r(i,j)\right\|^2 \quad (2)$$

et $I_0(i,j)$ et $I_r(i,j)$ l'image originale et l'image décompressée, respectivement.

III.1.6. Codage de Huffman

1) Effectuer le codage de Huffman à l'image transformée et quantifiée. Tout d'abord, il faut supprimer les coefficients nuls dans les matrices quantifiées. Puis calculer la probabilité d'apparition de chaque symbole. Ensuite, construire l'arbre de Huffman à l'aide de la fonction ***huffmandict***, effectuer le codage avec la fonction ***huffmenenco***.

2) Déduire le taux de compression pour un facteur de qualité **FQ= 10 : 20 : 90**.

Que remarquez-vous?

III.2. Compression/Décompression d'une image couleur

En se basant sur le script précédent : Compression / Décompression d'une image en niveaux de gris, effectuer les mêmes étapes pour l'image en couleurs '**peppers.png**'.

III.2.1. Etapes à suivre

1/- Préparation de la compression et conversion vers le plan YCbCr (Y: luminances, Cb et Cr les deux chrominances). Effectuer la conversion vers le plan YCbCr à l'aide de la fonction ***rgb2ycbcr***.

2/- Sous échantillonnage d'un facteur 2 des deux chrominances. Vérifiez rapidement que notre œil n'est pas sensible à un sous échantillonnage d'un facteur 2 de la chrominance (le format 4 :2 :0). Pour la suite, utilisez la fonction ***double()*** à l'image en luminance/chrominance de façon à travailler avec un niveau suffisant de précision.

3/- Découpage en blocs 8*8 .

4/- Application de la transformée DCT II sur chaque canal (Y, Cb, Cr) séparément.

5/- Quantification des coefficients de la DCT : Pour cela utilisez la matrice de quantification précédente pour luminance Y de qualité FQ=50 et une autre matrice pour les deux chrominances donnée par :

```
Q2=[17 18 24 47 99 99 99 99 ;
 18 21 26 66 99 99 99 99;
 24 26 56 99 99 99 99 99;
 47 66 99 99 99 99 99 99;
 99 99 99 99 99 99 99 99;
 99 99 99 99 99 99 99 99;
 99 99 99 99 99 99 99 99;
 99 99 99 99 99 99 99 99];
```

6/- Décompression : Effectuez la déquantification et la DCT inverse des trois canaux (Y, Cb, Cr). Pour cela, il faudra interpoler les valeurs manquantes des deux chrominances (pour revenir, depuis des blocs 4x4, à des blocs 8x8) avec la commande : ***bloc8x8 = kron(bloc4x4,ones(2,2)).*** Utilisez ensuite la manière successive les fonctions ***uint8()*** et ***ycbcr2rgb()*** pour revenir à une image RGB codée sur 8 digits.

7/- Application du codage de Huffman : Effectuer le codage de Huffman à l'image transformée et quantifiée.

8/- Affichage des images et comparaison : Affichez l'image d'origine et sa version compressée/décompressée en déduire le taux de compression après le codage d'Huffman.

III.2.2. Questions

1) Discutez de l'impact de la compression sur la qualité d'image.

2) Identifiez les avantages et inconvénients du schéma de compression JPEG dans ce contexte.

3) Réfléchissez aux domaines d'application de cette méthode de compression.

TP6 : Implémentation sous Matlab d'une méthode de compression d'images à base de la DWT

I. Objectifs du TP

- ✓ Comprendre le principe de la compression d'image par ondelettes avec perte et son fonctionnement dans les standards tels que JPEG2000.
- ✓ Appliquer une DWT (Discrete Wavelet Transform) pour décomposer une image en sous-bandes multi-résolution.
- ✓ Introduire la quantification des coefficients pour réduire le débit tout en contrôlant la perte d'information.
- ✓ Observer la reconstruction progressive de l'image à l'aide de la transformation inverse (IDWT).
- ✓ Expérimenter la compression progressive multi-résolution et analyser l'impact sur la qualité de l'image.
- ✓ Mesurer la performance de la compression via des indicateurs tels qu'EQM **et** PSNR.

II. Rappels théorique

II.1. Introduction

La compression d'images est essentielle pour réduire l'espace de stockage et le temps de transmission.

La DWT (Discrete Wavelet Transform) permet de représenter l'image sous une forme multi-résolution mieux adaptée à la compression, contrairement à la DCT (utilisée dans JPEG) qui traite l'image par blocs 8×8.

II.1. Transformation en Ondelette Discrète (DWT)

Le cœur de JPEG2000 repose sur l'utilisation de la DWT, qui permet de décomposer une image en différentes échelles de détail. Contrairement à la DCT, qui ne fournit qu'une résolution fixe, la DWT offre une représentation multi-résolution de l'image. Les coefficients d'ondelette obtenus permettent de représenter les détails fins ainsi que les grandes structures de l'image.

La théorie des ondelettes a été inventée par le mathématicien hongrois Alfred Haar dans les années 1910. Une ondelette est une transformation de fonction, comme Laplace ou Fourier, qui oscille principalement dans un intervalle restreint. L'évolution des ondelettes

dans le monde mathématique se fait en cherchant à caractériser les différents espaces fonctionnels. Si les mathématiciens ont développé de nouveaux concepts pour les espaces linéaires, les physiciens sont parvenus à une transformation temps-fréquence. Quant à Gabor, il décompose le signal en fréquences, intervalle par intervalle. Cela revient à comparer un segment de signal à des morceaux de courbes oscillantes de différentes fréquences. Les physiciens, travaillant dans le domaine du traitement du signal, représentent des phénomènes physiques par des sommes de translatés.

Nous pouvons donner une définition empirique des ondelettes : les ondelettes sont des fonctions qui respectent certains critères d'orthogonalité nécessaires pour la construction d'une analyse multirésolution. La multirésolution, telle qu'elle est énoncée par Meyer et Mallat, présente les propriétés suivantes :

1. la construction d'une fonction d'échelle est orthogonale à ses translatés par des entiers,
2. le signal à une résolution donnée contient toute l'information du signal aux résolutions grossières,
3. la fonction 0 est le seul objet commun à tous les espaces Vi,
4. n'importe quel signal peut être approximé avec une précision arbitraire.

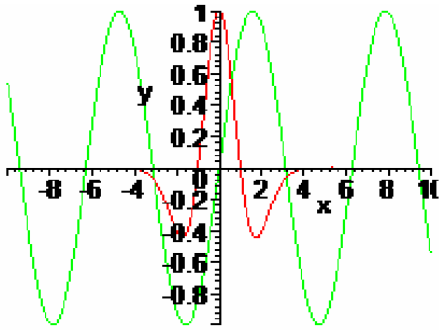


Figure 1 : Représentation d'une sinusoïde et d'une ondelette

Nous pouvons voir sur la figure ci-dessus que, contrairement à la fonction Cosinus (en vert), l'ondelette « Chapeau Mexicain » possède une amplitude variable, et est quasiment nulle en dehors de l'intervalle [-4,4]. Cette variation très locale de la fonction permet néanmoins de savoir précisément ce qui ce passe en n'importe quel endroit du signal original (non transformé). Contrairement au format JPEG, qui décompose une image en blocs de 8x8 pixels, la compression JPEG 2000 transforme chaque ligne horizontale en un signal, qui sera ensuite transformé en somme d'ondelettes. En effet, la variation de couleur et d'intensité de chaque pixel d'une ligne peut être assimilée à la variation de deux signaux. Chaque signal sera ensuite directement transformé en une série d'ondelettes, répétées en différents endroits, et à différentes échelles, pour que la somme décrive le plus exactement le signal original

(figure au-dessous). L'algorithme éliminera les variations les plus infimes pour compresser encore d'avantage l'image.

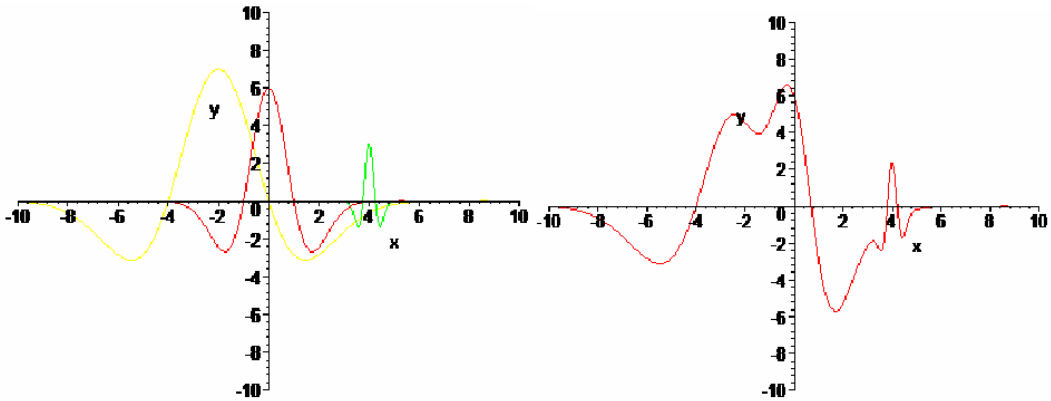


Figure 2 : Exemple de fonctions ondelette et leur somme

La famille des ondelettes construite par dilatation-translation à partir de l'ondelette mère est définie sous la forme:

$$\Psi_{a,b} = \frac{1}{\sqrt{a}} \Psi \left(\frac{x-b}{a}\right) \qquad (1)$$

avec a ≠ 0 et a, b ∈ R, ainsi, toutes les ondelettes ont la même énergie.

a : Paramètre de dilatation b : Paramètre de translation

La transformée continue s'écrit :

$$WT_{f,\psi}(a,b) = < f \left| \psi_{ab} > = \frac{1}{\sqrt{|a|}} \int f(x) \bar{\psi} \left(\frac{x-b}{a}\right) dx \qquad (2)$$

où < > est le produit scalaire. $\bar{\psi}$ désigne le complexe conjugué de ψ.

II.2. Transformée directe 2D-DWT

II.2.1. Principe

Passons maintenant à l'algorithme pyramidal utilisé. La décomposition en coefficients d'ondelettes n'utilise pas une fonction de moyenne, mais s'appuie sur deux filtres. Un filtre passe bas (L) et un filtre passe haut (H). La combinaison de ces filtres permet d'obtenir quatre sous images HH, HL, LH et LL. Ces filtres sont nommés filtres miroirs en quadrature (figure ci-dessous).

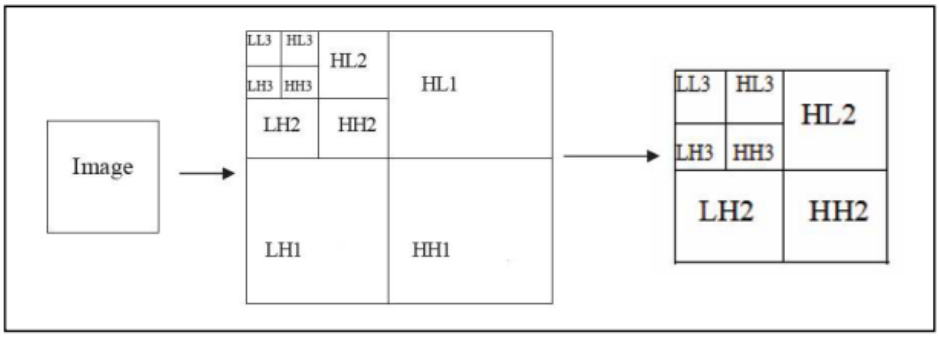


Figure 3 : Principe de la transformation en ondelettes (DWT) multi-résolution

Chacune des quatre images obtenues par la transformation représente des informations bien distinctes.

II.2.2. Décomposition d'une image en Ondelettes

L'image est d'abord décomposée en sous-bandes qui contiennent des informations à différentes échelles. Cela inclut la sous-bande approximative (coefficient moyen) ainsi que les sous-bandes détaillées (coefficients de haute fréquence).

- **Représentation Multi-résolution :** La DWT permet une analyse de l'image à plusieurs résolutions, offrant une meilleure flexibilité pour la compression.

Étape 1 : Filtrage en ligne (horizontale)

Pour chaque ligne x, on calcule :

- Passe-bas : $L(x,y) = \sum_k I(x,y)\ .\ h_L(y-k)$ (3)
- Passe-haut : $H(x,y) = \sum_k I(x,y)\ .\ h_H(y-k)$ (4)

Étape 2 : Filtrage en colonne (verticale)

Ensuite, pour chaque colonne y, on applique les mêmes filtres :

- Passe-bas : $LL(x,y) = \sum_k L(k,y)\ .\ h_L(x-k)$ (5)
- Passe-haut : $LH(x,y) = \sum_k L(k,y)\ .\ h_H(x-k)$ (6)
- Passe-bas : $HL(x,y) = \sum_k H(k,y)\ .\ h_L(x-k)$ (7)
- Passe-haut : $HH(x,y) = \sum_k H(k,y)\ .\ h_H(x-k)$ (8)

Les résultats sont sous-échantillonnés de moitié (facteur 2) dans les deux directions.

Sous-bandes obtenues:

- LL : basse fréquence (approximation).
- LH : détails horizontaux (passe-bas vertical, passe-haut horizontal).
- HL : détails verticaux (passe-haut vertical, passe-bas horizontal).
- HH : détails diagonaux (passe-haut dans les deux directions).

II.3. Transformée inverse 2D-DWT

Pour reconstruire l'image d'origine à partir des sous-bandes LL, LH, HL, HH, on applique les étapes inverses :

1. **Suréchantillonnage** des sous-bandes (×2 dans les deux directions).
2. **Filtrage inverse** avec des filtres passe-bas (h'_L) et passe-haut (h'_H).
3. **Somme des contributions** des sous-bandes.

Étape de reconstruction :

$$L'(x,y) = \sum_k LL(x,k).h'_L(y-k) + \sum_k LH(x,k).h'_H(y-k) \quad (9)$$

$$H'(x,y) = \sum_k HL(x,k).h'_L(y-k) + \sum_k HH(x,k).h'_H(y-k) \quad (10)$$

$$I'(x,y) = \sum_k L'(k,y).h'_L(x-k) + \sum_k H'(k,y).h'_H(x-k) \quad (11)$$

II.4. Représentation matricielle simplifiée

Si l'image I est une matrice, les opérations peuvent être décrites comme suit :

1. **Transformée directe** :

$$\text{2D-DWT} = (L \cdot L^T + L \cdot H^T + H \cdot L^T + H \cdot H^T) \quad (12)$$

2. **Transformée inverse** :

$$\text{2D-IDWT} = L^T \cdot L + H^T \cdot L + L^T \cdot H + H^T \cdot H \quad (13)$$

Ici, L et H représentent les filtrages passe-bas et passe-haut respectivement.

Compression : en conservant principalement la sous-bande LL, qui contient les informations globales, et en supprimant certaines des sous-bandes de détail (LH, HL, HH).

Reconstruction : en utilisant la transformée inverse pour restaurer l'image.

II.5. Types de compression d'image par ondelettes

La compression d'image par ondelettes repose sur la **transformée en ondelettes discrète (DWT)**, qui permet de représenter l'image à différentes résolutions. On distingue plusieurs types de compression selon la méthode utilisée et le niveau de fidélité recherché.

II.5.1. La compression par ondelettes sans perte : utilise des ondelettes entières et ne fait pas appel à une quantification destructive. Elle permet de reconstruire exactement l'image originale après décompression. Cette méthode est utilisée dans JPEG2000 en mode lossless et convient aux applications nécessitant une qualité parfaite, comme l'imagerie médicale ou l'archivage.

II.5.2. La compression par ondelettes avec perte : repose sur l'utilisation d'ondelettes réelles et sur la quantification des coefficients. Les coefficients de faible amplitude sont supprimés afin de réduire la taille des données. Cette approche, utilisée dans JPEG2000 en mode lossy, permet d'obtenir des taux de compression élevés et est largement employée dans les applications multimédia.

II.5.1. La compression progressive par ondelettes : permet une reconstruction graduelle de l'image, de la plus basse résolution vers la plus détaillée. Elle est particulièrement adaptée aux transmissions sur des réseaux à débit limité et aux applications web.

II.5.1. La compression multi-résolution : exploite la structure hiérarchique des ondelettes pour représenter l'image à plusieurs niveaux de résolution. Elle facilite le zoom et l'affichage rapide sans décoder l'image complète, ce qui est utile en imagerie satellite et dans les bases de données d'images.

II.5.1. La compression par seuillage et codage entropique : consiste à annuler les coefficients d'ondelettes de faible amplitude puis à compresser les coefficients restants à l'aide de méthodes comme le codage de Huffman ou arithmétique. Cette étape permet de réduire efficacement le débit binaire final.

II.6. Compression/Décompression par ondelettes (JPEG2000)

Les étapes étudiées, illustrant les standards basés sur les ondelettes tels que JPEG2000, utilisent une compression d'image avec perte combinée à une compression progressive multi-résolution. Grâce à la quantification, une perte d'information contrôlée est introduite, tandis que la reconstruction par transformation inverse permet d'afficher l'image de façon progressive, avec une amélioration graduelle de la qualité et de la résolution.

II.6.1. Compression

- La compression commence par une **transformée en ondelettes discrète (DWT)**, qui décompose l'image en sous-bandes multi-résolution (approximation et détails).
- Les coefficients d'ondelettes obtenus sont ensuite soumis à une **quantification**, permettant de réduire leur précision et d'introduire une perte d'information contrôlée.
- Les coefficients de faible amplitude sont quantifiés de manière plus agressive car ils ont un faible impact perceptuel.
- Un **codage entropique** (Huffman ou arithmétique) est appliqué afin d'éliminer les redondances statistiques et de réduire davantage la taille des données.

II.6.2. Décompression

- Le processus débute par un **décodage entropique** pour récupérer les coefficients quantifiés.
- Une **déquantification** est ensuite réalisée afin de restaurer une approximation des coefficients originaux.
- La **transformation inverse en ondelettes (IDWT)** permet de reconstruire l'image à partir des sous-bandes.

Grâce à la structure multi-résolution des ondelettes, l'image est reconstruite de manière **progressive**, avec une amélioration graduelle de la qualité et de la résolution.

III. Partie simulation

Ecrire un code Matlab permettant d'effectuer les tâches suivantes :

1. Chargement de l'image

- Chargez une image en niveaux de gris et affichez-la.

2. Application de la DWT

- Appliquez la transformation en ondelette discrète sur l'image en utilisant la fonction ***dwt2***.

Instruction Matlab à tester :

```
clc; clear all; close all;
I = imread('peppers.png');
X=rgb2gray(I);
[cA1,cH1,cV1,cD1] = dwt2(X,'db2');
%Les coefficients approximation: matrice CA1
%Les coefficients  de détail horizontale  matrices CH1
%Les coefficients  de détails verticale   matrices CV1
%Les coefficients  de détails diagonale  matrices CD1
```

3. Visualisation des coefficients DWT

- Affichez les coefficients approximatifs et détaillés obtenus après la DWT. Utilisez la fonction ***wcodemat (X)*** pour l'affichage des matrices.
- Appliqué les commandes sur une image en niveau de gris puis sur une image couleur pour afficher tous les coefficients.

Remarque : Utilisez le **Help** du Matlab

```
Y = WCODEMAT(X,NBCODES,'mat',1);
```

4. Quantification des coefficients DWT

- Appliquez une méthode de quantification sur les coefficients DWT. Par exemple, vous pouvez fixer un seuil pour les coefficients.

5. Reconstruction de l'image à partir des coefficients quantifiés

- Calculer l'inverse de la décomposition avec wavelet.
- Calculer les coefficients de la décomposition **DWT.**
- Extraire les coefficients d'approximation et de détails d'ordre 2 en utilisant la fonction ***wavedec2***.
- Afficher les paramètres des filtres (voir le help).
- Utilisez la fonction ***idwt2*** pour reconstruire l'image à partir des coefficients DWT quantifiés.

Instructions Matlab à tester :

```
[C,S] = WAVEDEC2(X,N,Lo_D,Hi_D);
%Lo_D is the decomposition low-pass filter and
%Hi_D is the decomposition high-pass filter.
[F1,F2] = WFILTERS('wname','type');
%Les coefficients approximation: matrice CA1
%Les coefficients  de détail horizontale  matrices CH2
%Les coefficients  de détails verticale   matrices CV2
%Les coefficients  de détails diagonale  matrices CD2
```

6. Compression EZW (Embedded Zerotree Wavelet): EZW est une méthode de compression par ondelettes avec perte, basée sur le concept de zerotree, permettant un codage progressif de l'image en exploitant la dépendance hiérarchique des coefficients.

- En utilisant la commande ***wcompress***, afficher le CR (compression ratio), charger l'image mask asiatique.
- En utilisant la commande ***wcompress***, afficher le CR (compression ratio) et le PPB (Bit-Per-Pixel ratio).
- Faire la compression et la décompression avec ***wcompress***.
- Reprendre la même chose avec nbloop=9;12
- Comparer avec **Biorthogonal** wavelets bior4.4, bior3.7

Instructions Matlab à tester :

```
load mask;
image(X);
axis square;
colormap(pink(255));
title('Original Image: mask');
```

7. Évaluation de la qualité de l'image

- Comparez l'image originale et l'image reconstruite. Calculez l'EQM (Erreur Quadratique Moyenne) et le PSNR (Peak Signal-to-Noise Ratio) pour évaluer la qualité.

8. Comparaison des performances

- Testez l'impact de différents seuils de quantification sur la qualité d'image et le taux de compression. Affichez les résultats pour chaque valeur de seuil choisie.

9. Conclusion

- Discutez des résultats obtenus. Quel est l'impact de la quantification sur la qualité de l'image ? Quels sont les avantages et inconvénients de l'utilisation de la DWT pour la compression d'images ?
- **DWT multi-niveaux :** Implémentez une DWT à plusieurs niveaux et comparez les résultats.
- **Compression couleur :** Étendez l'implémentation pour traiter des images couleur en séparant les canaux R, G, et B.
- Quelle sont les avantage qu'offrant la norme JPEG2000 par rapport à la norme JPEG classique ?
- Donner quelques applications de JPEG2000.
- Quel a été le taux de compression obtenu pour différents seuils de quantification, et comment cela a-t-il affecté la qualité de l'image ?
- Quels indicateurs (comme l'EQM ou le PSNR) avez-vous utilisés pour évaluer la qualité de l'image, et quels résultats avez-vous obtenus ?
- Quelle méthode de quantification a produit les meilleurs résultats en termes de qualité d'image et de taux de compression ?
- Quelles applications pratiques de la compression d'images basée sur la DWT avez-vous identifiées ? Comment cette méthode pourrait-elle être utilisée dans des scénarios réels ?

REFERENCES BIBLIOGRAPHIES

[1] K. Sayood, *Introduction to Data Compression*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann, 2006.

[2] D. Salomon and G. Motta, *Handbook of Data Compression*, 5th ed. London, U.K.: Springer, 2010.

[3] J. G. Proakis and M. Salehi, *Digital Communications*, 5th ed. New York, NY, USA: McGraw-Hill, 2007.

[4] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. Upper Saddle River, NJ, USA: Pearson, 2018.

[5] H. Taub and D. L. Schilling, *Principles of Communication Systems*, 3rd ed. New York, NY, USA: McGraw-Hill, 2008.

[6] G. Strang, *Introduction to Linear Algebra*, 5th ed. Wellesley, MA, USA: Wellesley–Cambridge Press, 2016.

[7] C. Christopoulos, A. Skodras, and T. Ebrahimi, "The JPEG2000 still image coding system: An overview," *IEEE Trans. Consumer Electron.*, vol. 46, no. 4, pp. 1103–1127, Nov. 2000.

[8] S. A. Khayam, *The Discrete Cosine Transform (DCT): Theory and Application*, ECE 802–602: Information Theory and Coding, Academies Course, 2003.

[9] P. Getreuer, *Image Processing with MATLAB*, unpublished manuscript.