Les codages/ Code de Hamming (7,4,3) systématique

Objectif du TP

Lors de la transmission de données via des passerelles "classiques" (wifi, protocole IP, ...) il n'est pas rare d'avoir des erreurs de transmission.
Comment détecter et si possible corriger ces erreurs ?

L'objectif est évidemment ambitieux .... nous resterons raisonnables.

Comment détecter les erreurs ? ⇒ introduire des bits de vérifications.

Nous considérons le contexte suivant :
- Nous souhaitons transmettre un message sous la forme d'une suite de bits découpé en blocs de k bits. (k est également une puissance de 2)
- La probabilité de transmission erronée d'un bit est très faible.

Dans la suite, les questions nécessitant de rédiger un code en python sont indiquées par Py . Je propose de traiter en priorité ces questions lors de la séance de TP.

Un outil : pour créer la liste contenant tous les éléments de $\{0,1\}^k$ nous pourrons si besoin utiliser

```
def lstbin(n,k):
    b=bin(n)[2:]
    p=len(b)
    l=[0 for i in range(k-p)]
    for x in b:
        l+=[int(x)]
    return(l)

C=[lstbin(i,k) for i in range(2**k)]
```

Première idée : une somme de contrôle/ Checksum

Nous introduisons un code $(n,k) = (9,8)$ qui fait intervenir la notion de bit de parité et de somme de contrôle. Cette notion sera généralisée dans la suite par la notion de matrice de contrôle.
On considère un message composée de blocs de 8 bits.
A chaque bloc de 8 bits on ajoute un bit "de parité" afin que la somme des bits du nouveau bloc de 9 bits soit pair.
Nous dirons qu'il s'agit d'un code (9,8).

1. Py Ecrire un programme **parite(m)** qui tranforme une message m de longueur 8 en respectant la procédure précédente.

   ```
   import copy as c
   def parite(m):
       l=c.deepcopy(m)
       s=0
       for x in l:
           s+=x
       return(l+[s%2])
   ```

2. Quelle sont les inconvénients d'un telle méthode pour la détection et la correction des erreurs ?
   On une seule se produit nous sommes en mesure de la détecter mais pas de la localiser et encore moins de corriger le message. Si plusieurs erreurs se produisent nous ne sommes même pas sur de les détecter.

Que peut-on/doit-on espérer d'un code ?
- Etre capable de retrouver "rapidement" le message initial dans le message codé.
- Etre capable de détecter et corriger un petit nombre d'erreurs.

– Les algorithmes de codage et décodage et de correction doivent être efficaces.

Comment définir un code ?

Nous considèrerons dans la suite qu'une liste de bits sera décomposée en blocs de longueur k. Un bloc de longueur k sera appelé "message" et représenté en pratique par une liste $[b_0, b_2, \cdots, b_{k-1}]$ et en théorie par une séquence $b_0 \cdots b_{k-1}$ ou un vecteur colonne $\begin{pmatrix} b_0 \\ \vdots \\ b_{k-1} \end{pmatrix}$.

Principe du codage de type (n, k) où $n \geq k$

- Découper le message initial en blocs de longueur k.
- Appliquer la même méthode à chaque bloc en rajoutant à la fin de chaque bloc $n - k$ bits de contrôle
  - sans changer le bloc initial.
    ou
  - en modifiant le bloc initial (Attention de manière à ne pas confondre deux blocs initiaux distincts)

Vocabulaire : lorsque le bloc initial reste inchangé nous parlerons de code systématique.
Un message $b_0 \cdots b_{k-1}$ codé par un code systématique (n,k) sera de la forme $b_0 \cdots b_{k-1} c_0 \cdots c_{n-k-1}$.

Définition formelle d'un code : en identifiant un bloc de k bits à un élément de $\{0,1\}^k$, un code de paramêtres (n, k) est une application **injective** f de $\{0,1\}^k$ dans $\{0,1\}^n$. L'entier n s'appelle la longueur du code f et l'entier k sa dimension.

Pour éviter un excès de technicité, nous assimilerons si nécessaire un code f et l'ensemble des messages obtenus par ce codage :

$$C = \{f(m) \mid m \in \{0,1\}^k\}$$

Exemples :

- Code de parité $(k+1, k)$: un tel code généralise la méthode décrite dans le première partie à un bloc initial de longueur k.
- Code de répétitions $(n, 1)$: il s'agit dans ce cas de répéter simplement n fois chaque bit rencontré.
  Ex : un message $b \in \{0,1\}$ est codé par le code de répétitions (3,1) par le message bbb.

Codage par répétitions et décodage naïf

---

1. Nous nous intéressons dans un premier temps au codage par répétitions $(3, 1)$
   (a) Identifier suivant le nombre d'erreurs les messages possibles de $\{0,1\}^3$.
   - message d'origine 000 :
     - Aucune erreur : 000
     - Une erreur : 001 , 010, 100
     - Deux erreurs : 011, 101, 110
     - Trois erreurs : 111
   - message d'origine 111 :
     - Aucune erreur : 111
     - Une erreur : 011, 101, 110
     - Deux erreurs : 001 , 010, 100
     - Trois erreurs : 000

   (b) En considérant qu'un bit peut être erroné avec probabilité $p < \frac{1}{2}$. Est-il plus probable de recevoir un message avec une erreur ou deux erreurs ?
   (sachant qu'un message avec 3 erreurs ne peut pas être détecté)
   La probabilité qu'un message donné contienne 1 erreur est $3p(1-p)^2$ et la probabilité qu'un message donné contienne 2 erreurs est $3p^2(1-p)$. Sachant que $p < \frac{1}{2}$ $(p < 1-p)$ nous obtenons qu'il est plus probable qu'un message ne contienne qu'une seule erreur.

   (c) Proposer une méthode permettant de corriger les messages contenant une seule erreur.
   Lorsque une ou deux erreurs se produisent le message reçu n'est pas un message du code nous pouvons ainsi détecter le problème. Lorsque le message contient une erreur il suffit pour corriger de prendre le message le plus "proche" dans un sens que nous allons définir avec la distance de Hamming. Nous pourrions également

corriger deux erreurs avec une méthode adaptée.
Sachant que la probabilité de recevoir un message avec deux erreurs est plus faible que celle de recevoir un message n'en contenant qu'une seule, nous préférons nous concentrer sur la correction d'une erreur.

Définition : une distance pour "mesurer" le nombre d'erreurs/ Poids et distance de Hamming
Lorsque m_1 et m_2 sont deux messages de longueur k : $d_1, d_2 \in \{0,1\}^k$.
- La distance de Hamming entre m_1 et m_2, notée $d(m_1, m_2)$ est le nombre de bits distincts de m_1 et m_2.
- Le poids (de Hamming) de m_1 est sa distance à 0. i.e. le nombre de bits non nuls de m_1.

2. Py Ecrire des fonctions **dist(m_1, m_2)** et **pds(m)** qui donne la distance entre deux messages m_1 et m_2 et le poids d'un message m.

```
def dist(m1,m2):
    if len(m1)!=len(m2):
        return('les listes ne sont pas compatibles')
    else:
        s=0
        for i in range(len(m1)):
            s+=(m1[i]+m2[i])%2
    return(s)

def pds(m):
    l=[0 for x in m]
    return(dist(m,l))
```

3. Py Ecrire une fonction **distm(m,C)** qui donne la distance du message m au code C ainsi que la liste des messages réalisant cette distance minimale.

```
def distm(m,C):
    d=len(m)+1
    op=[]
    for x in C:
        if dist(x,m)<d:
            d=dist(x,m)
            op=[c.deepcopy(x)]
        elif dist(x,m)==d:
            op+=[c.deepcopy(x)]
    return(d,op)
```

Décodage naïf pour un code C : (le code n'est pas nécessairement "par répétitions")
Nous considérons un code (n,k) décrit par C ou f. Un décodage naïf se réalise de la manière suivante :
Nous recevons un message $m \in \{0,1\}^n$ nous décodons m en renvoyant le message $m' \in C$ réalisant la distance minimale avec m.
Il suffit ensuite de retrouver le message d'origine par injectivité du codage f.
(Dans le cas d'un codage systématique, il suffit de prendre le préfixe de longueur k du mot m' obtenu)

4. Py Ecrire une fonction **deconaif(m)** qui décode un message m de longueur n codé par répétitions $(7,1)$.

```
def deconaif(m):
    C=[[1 for i in range(7)],[0 for i in range(7)]]
    return(distm(m,C)[1])
  # nous remarquons que le 7 permet de garantir l'unicité du message
```

5. Evaluer la complexité d'un tel décodage pour un codage systématique (n,k). (la complexité sera évaluée en nombre de comparaisons)
Sachant que nous devons tester la distance avec tous les 2^k messages du code et que nous devons effectuer n comparaisons pour déterminer une distance d'un message de longueur n. Nous obtenons une complexité dans le pire des cas en $\mathcal{O}(n2^k)$
6. Identifier les problèmes d'un tel décodage.
Comme nous venons de le déterminer à la question précédente, un coût d'un tel algorithme est exponentiel en la dimension du message.... ce n'est pas raisonnable. De plus, d'après la multiplicité des messages pouvant réaliser ce minimum, le message obtenu par cette méthode n'a aucune raison d'être le message d'origine.

Généralités : distance minimale d'un code et codes linéaires

Distance minimale d'un code :
On appelle distance minimale d'un code donné par C associé f la plus petite distance non nulle entre deux messages de C.
Dans la suite, pour un code C nous préciserons les paramètres (n, k, d) où
- k est la longueur d'un bloc/ message d'origine.
- n est la longueur d'un message codé.
- d est la distance minimale du code C.

Nous considérons un code C de paramètres (n, k, d).

1. Capacité de détection : montrer que le plus grand entier p tel qu'on soit toujours capable de détecter qu'un message est erroné est égal $d - 1$.
   Sachant que la distance minimale d'un code est la plus petite distance entre 2 messages du code. Si un message m transmis contient p erreurs avec $p < d$ la distance de m à C est strictement inférieure à p ce qui permet d'identifier l'existence d'une erreur.
2. Capacité de correction : montrer que le plus grand entier p tel qu'on soit toujours capable de corriger au moins p erreurs est égal $\lfloor \frac{d-1}{2} \rfloor$.
   Si m est un message contenant p erreurs tel que $2p < d$ alors le code peut corriger les p erreurs de m. En effet si m et m' sont deux mots du code à distance inférieure ou égale à p du message reçu, alors $d(m, m') \leq 2p < d$ donc $m = m'$.
3. Etablir la distance minimale d'un code par répétitions $(n, 1)$.
   Est-ce que les valeurs précédentes sont compatibles avec l'étude préliminaire ?
   Nous obtenons pour un tel code $d = n$. En particulier nous pouvons détecter un message comportant au plus $n - 1$ erreurs et corriger un message contenant au plus $\lfloor \frac{n-1}{2} \rfloor$ erreurs.
   Dans le cas $n = 3$, nous pouvons ainsi corriger des messages contenant au plus 1 erreurs.
4. Code de parité :
   (a) Py Ecrire une fonction **distmini(C)** qui donne la distance minimale d'un code C. Identifier à l'aide de Python la distance minimale d'un code à l'aide d'un bit de parité (9,8).

```
def lstbin(n,k):
    b=bin(n)[2:]
    p=len(b)
    l=[0 for i in range(k-p)]
    for x in b:
        l+=[int(x)]
    return(l)

k=8
C=[lstbin(i,k) for i in range(2**k)]
Code98=[parite(x) for x in C]

def distmini(Code):
    d=len(Code[0])
    for x in Code:
        l=c.deepcopy(Code)
        l.remove(x)
        d=distm(x,l)[0]
    return(d)
```

   (b) Montrer de manière théorique que la distance minimale d'un code à l'aide d'un bit de parité (9,8) est 2.
   Nous obtenons en considérant les messages 000000000 et 000000011 que la distance est inférieur à 2.
   Il suffit de montrer que 2 messages m et m' distincts du code ont une distance supérieure à 2. 2 messages distincts du code proviennent de 2 messages d'origine distincts. La distance entre les 2 messages d'origine est donc d'au moins 1.
   Si cette distance est égale au moins à 2 la distance entre m et m' est nécessairement au moins égale à 2.
   Si cette distance est égale à 1 les mots différents d'un bit et par suite leur bit de parité est nécessairement distinct. Donc la distance entre m et m' est égale à 2. Dans tous les cas nous obtenons que la distance est supérieure à 2.

Définition : code linéaire
Un code linéaire C associé f de paramètres (n,k) est dit linéaire lorsqu'il existe une matrice G de rang k de $\mathcal{M}_{n,k}(\mathbb{Z}/2\mathbb{Z})$ telle que

$$C = \{G \times m \mid m \in \{0,1\}^k\}$$

La matrice G est appelée la matrice génératrice du code. (où $\times$ est le produit matriciel)
Remarque : Le fait que la matrice G soit de rang k garantit l'injectivité de l'application $f : m \mapsto Gm$.

5. Décrire la forme générale d'une matrice génératrice d'un code linéaire systématique.
Sachant que le message d'origine est inchangé dans le message transmis. Les k premiers bits restent donc inchangés. Une telle matrice est donc de la forme $G = \begin{pmatrix} I_k \\ G_1 \end{pmatrix}$ où $G_1 \in \mathcal{M}_{n-k,k}(\mathbb{Z}/2\mathbb{Z})$

6. Montrer que les codes à l'aide du bit de parité (9,8), par répétitions $(n,1)$ sont des codes linéaires systématiques.
Les matrices G_1 associées sont égales à $G_1 = (1, \cdots, 1)$ et $G_1 = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$

7. Montrer que la distance minimale d'un code linéaire C est le plus petit poids non nul d'un message de C.
Retrouver la distance minimale pour les codes à l'aide du bit de parité (9,8), par répétitions $(n,1)$.
Nous avons $dist(m,m') = pds(m+m')$. En considérant un code est linéaire C, nous avons $\forall m, m' \in C, m \pm m' \in C$ donc $C = \{m + m' \mid m, m' \in C\}$ ce qui donne le résultat attendu.

Nous admettrons la **borne de Singleton** : La distance minimale d d'un code linéaire de dimension k de longueur n vérifie l'inégalité

$$d \leq n + 1 - k$$

Définition : matrice de contrôle
Soit C un code linéaire (n,k) de matrice génératrice G. On appelle matrice de contrôle de C toute matrice de $\mathcal{M}_{n-k,n}(\mathbb{Z}/2\mathbb{Z})$ vérifiant

$$Hm = 0 \iff m \in C$$

Le vecteur Hm s'appelle le syndrome du message m.

8. Soit $G = \begin{pmatrix} I_k \\ G_1 \end{pmatrix}$ une matrice génératrice d'un code linéaire (n,k) systématique montrer qu'une matrice de contrôle est donnée par

$$H = \begin{pmatrix} G_1 & I_{n-k} \end{pmatrix}$$

Sachant que

$$\begin{pmatrix} G_1 & I_{n-k} \end{pmatrix} m = G_1 \begin{pmatrix} m_0 \\ \vdots \\ m_{k-1} \end{pmatrix} + \begin{pmatrix} m_k \\ \vdots \\ m_{n-1} \end{pmatrix} = G_1 \begin{pmatrix} m_0 \\ \vdots \\ m_{k-1} \end{pmatrix} - \begin{pmatrix} m_k \\ \vdots \\ m_{n-1} \end{pmatrix}$$

En notant $m_1 = \begin{pmatrix} m_0 \\ \vdots \\ m_{k-1} \end{pmatrix}$ le message d'origine nous obtenons l'équivalence

$$Hm = 0 \iff G_1 m_1 = \begin{pmatrix} m_k \\ \vdots \\ m_{n-1} \end{pmatrix} \iff Gm_1 = m$$

Ce qui donne l'équivalence $Hm = 0 \iff m \in C$.

9. Py Ecrire une fonction **mult(M,X)** qui donne la multiplication matricielle de la matrice M et du vecteur colonne X (dans $\mathbb{Z}/2\mathbb{Z}$). Quelle est la complexité en nombre d'opérations arithmétiques élémentaires de la multiplication précédente ?

```
def mult(M,X):
    colonne=len(M[0])
    ligne=len(X)
    c=[]
    if ligne!=colonne:
        return(' matrices incompatibles')
    else:
```

```
        for k in range(len(M)):
            s=0
            for j in range(colonne):
                    s+=M[k][j]*X[j]
            c+=[s%2]
    return(c)
```

D'après la fonction précédente, le coût est dominé par 2×colonne× Nbre de lignes de M. Lorsque que la matrice M est de taille (n,k) le coût est donc $\mathcal{O}(nk)$.

10. Py Ecrire une fonction **codlinsys(G1,m)** qui donne le codage linéaire de m à l'aide de la matrice génératrice $\begin{pmatrix} I_k \\ G_1 \end{pmatrix}$. (vous pourrez tester cette fonction sur une code par parité (9,8)).

```
def codlinsys(G1,m):
    return(m+mult(G1,m))

k=8
C=[lstbin(i,k) for i in range(2**k)]
Code98=[parite(x) for x in C]

G1=[[1,1,1,1,1,1,1,1]]
print([codlinsys(G1,C[k]) for k in range(2**k)]==Code98)
```

11. Py Ecrire une fonction **detect(G1,m)** qui détecte une erreur dans le codage linéaire de m à l'aide de la matrice génératrice $\begin{pmatrix} I_k \\ G_1 \end{pmatrix}$.

```
def detect(G1,m):
    k=len(G1[0])
    n=len(m)
    l2=[m[i] for i in range(k,n)]
    l1=[m[i] for i in range(k)]
    return(mult(G1,l1)==l2)
```

Code de Hamming (7,4) systématique

Code de Hamming (7,4) : Un code de Hamming systématique de paramètres (7,4) est le code

$$f : a_1a_2a_3a_4 \in \{0,1\}^4 \to a_1a_2a_3a_4b_5b_6b_7 \in \{0,1\}^7$$

où $b_5 = a_1 + a_2 + a_3$, $b_6 = a_1 + a_2 + a_4$, $b_4 = a_1 + a_3 + a_4$.
Nous noterons C ce code dans la suite.

1. Montrer C est un code linéaire dont on déterminera la matrice génératrice.
   La matrice G_1 associé à ce code est
$$G1 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

2. Py Ecrire la fonction **Ham74(m)** qui donne le message codé du message m et montrer que ses paramètres sont (7,4,3). (vous pouvez évidemment utiliser Python)

```
 def Ham74(m):
    G1=[[1,1,1,0],[1,1,0,1],[1,0,1,1]]
    return(codlinsys(G1,m))

k=4
C=[lstbin(i,k) for i in range(2**k)]
Code=[Ham74(C[k]) for k in range(2**k)]

print(distmini(Code))
```

3. Montrer qu'une matrice de contrôle de C est donnée par

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Il suffit de relire les questions précédentes.

Nous remarquons que les colonnes de la matrice H contient tous les messages de 3 bits possibles.

Nous rappelons que la capacité de correction pour code de distance minimale d est $\lfloor \frac{d-1}{2} \rfloor$. Dans le codage traité, nous avons $d = 3$ ce qui implique que $\lfloor \frac{d-1}{2} \rfloor = 1$. Nous ne pouvons donc espérer proposer un algorithme corrigeant un message contenant plus de 2 erreurs.

4. **Correction d'erreurs :**

Lorsque nous recevons le message m de taille 7 pour identifier l'existence d'une erreur il suffit d'effectuer le calcul Hm.

- Si $Hm = 0$: le message transmis m ne contient pas d'erreur. Pour récupérer le message d'origine, il suffit de prendre les 4 premiers bits du message m.
- Si $Hm \neq 0$: le message transmis m contient au moins une erreur.
  - Si le message contient une unique erreur : le vecteur Hm est un vecteur colonne de 3 bits. Il est donc nécessairement égal à une colonne C_i de H pour $i \in [\![1, 7]\!]$. Nous en déduisons que l'erreur vient de la i-ème valeur de m.
  - Si le message contient plusieurs erreurs : utiliser l'algorithme naïf pour décoder le message.

(a) Justifier de la méthode décrite précédemment dans le cas d'une erreur.
Si le message codé m est entaché d'une seule erreur e. Nous recevons le message $m' = m + e$. Nous obtenons en particulier $Hm' = Hm + He = He$. Or e est par hypothèse un vecteur de la base canonique de $\{0, 1\}^n$, sachant que les colonnes de H sont distinctes identifier He parmi les vecteurs colonnes de H revient à identifier e ce qui donne par suite l'indice de la valeur altérée.

(b) Py Ecrire une fonction **decHam74(m)** qui donne le message d'origine qui a été codé et transmis à l'aide du code de Hamming (7,4,3) en m.
(vous pourrez si nécessaire introduire une fonction que permet d'additionner deux matrices)

```
# on transpose ... plus simple pour comparer les vecteurs lignes
def transpose(M):
    ligne=len(M)
    col=len(M[0])
    M1=[[0 for i in range(ligne)] for j in range(col) ]
    for i in range(ligne):
        for j in range(col):
            M1[j][i]=M[i][j]
    return(M1)

# Le i eme vecteur de la base canonique

def ei(i):
    l=[0 for k in range(7)]
    if i>0:
        l[i-1]=1
    return(l)

# Addition de 2 matrices

def plusliste(m1,m2):
    l=[]
    if len(m1)!=len(m2):
        return('matrice non compatibles')
    else:
        for i in range(len(m1)):
            l+=[(m1[i]+m2[i])%2]
```

```
    return(l)

# decodage

def decHam74(m):
    H=[[1,1,1,0]+[1,0,0],[1,1,0,1]+[0,1,0],[1,0,1,1]+[0,0,1]]
    l=mult(H,m)
    M=transpose(H)
    ind=0
    for i in range(len(M)):
        if M[i]==l:
            ind=i
            break
    return(plusliste(m,ei(i+1))[:4])

# On teste
k=4
C=[lstbin(i,k) for i in range(2**k)]
Code=[Ham74(C[k]) for k in range(2**k)]

j=3
i=4
print(Code[j],decHam74(plusliste(Code[j],ei(i))))
```