Détection et correction d'erreurs

On étudie l'envoi d'un message représenté en PYTHON comme une liste de 0 et de 1. On considère que des erreurs de transmission peuvent se produire et on désire pouvoir, en partie, les détecter et même les corriger. Pour cela, on ajoute des informations supplémentaires au message de départ. On prendra garde à faire la différence entre :

— le message à transmettre (liste **M**) ;

— le message effectivement transmis (liste **Mt**), obtenu à partir de **M** en ajoutant des éléments suivant certaines règles ;

— le message reçu (liste **Mr**) qui est de même longueur que **Mt** mais peut en différer suite aux erreurs de transmission ;

— le message décodé à partir du message reçu (liste **Md**). Ce message est de même longueur que **M** et il est identique à **M** lorsqu'il n'y a pas eu d'erreur de transmission, ou lorsque ces erreurs ont pu être corrigées.

I. Parité d'une liste et codage par bit de parité

Si L est un liste constituée de 0 et de 1, on appelle bit de parité de L le nombre $p(L)$ défini par :

$$\begin{aligned} p(L) &= 0 \text{ s'il y a un nombre pair de 1 apparaissant dans } L \\ &= 1 \text{ sinon} \end{aligned}$$

On conviendra que si L est la liste vide, alors $p(L) = 0$. La technique du bit de parité consiste à définir **Mt** à partir de **M** en ajoutant le bit de parité de **M**.

Question 1 : On considère le message `M=[1,0,1,1,1,0]`.

(a) Quelle est la parité de **M** ?

(b) Quel est le message transmis **Mt** correspondant ?

(c) On suppose que `Mr=[1,0,0,1,1,0,0]`, comment peut-on détecter l'erreur de transmission ?

(d) On suppose que `Mr=[0,0,1,1,1,0,1]`, peut-on détecter l'erreur de transmission ?

Question 2 : Écrire une fonction **`parite(L)`** qui calcule la parité de la liste L.

Question 3 : Écrire la fonction **`coder_parite(M)`** qui construit le message **`Mt`** à partir de **`M`**.

Question 4 : Écrire la fonction **`verifier_parite(Mr)`** qui teste si le message reçu **`Mr`** est valide.

II. Codage matriciel

On présente ici un autre type de codage qui permet de corriger une erreur. On suppose que le message à transmettre **M** est de longueur 9 et on dispose les éléments de ce message dans un tableau 3×3 :

$M[0]$	$M[1]$	$M[2]$
$M[3]$	$M[4]$	$M[5]$
$M[6]$	$M[7]$	$M[8]$

On complète ce tableau de la manière suivante :

$M[0]$	$M[1]$	$M[2]$	ℓ_0
$M[3]$	$M[4]$	$M[5]$	ℓ_1
$M[6]$	$M[7]$	$M[8]$	ℓ_2
c_0	c_1	c_2	

où ℓ_i est la parité de la ligne i et c_i est la parité de la colonne i. Le message transmis est alors obtenu en ajoutant à **M** les valeurs $\ell_0, \ell_1, \ell_2, c_0, c_1, c_2$.

Question 5 : On suppose que `M=[1,1,0,0,0,0,0,1,0]`. Quel est le message transmis **Mt** ?

Question 6 : Écrire une fonction **`coder_matrice(M)`** qui construit le message transmis **Mt** à partir de **M**

Question 7 : On suppose que **Mt** et **Mr** diffèrent sur une seule valeur.

(a) Comment savoir si cette différence porte sur l'un des bits $\ell_0, \ell_1, \ell_2, c_0, c_1, c_2$?

(b) Si ce n'est pas le cas, comment retrouver le bit du message où se trouve l'erreur ?

Écrire une fonction **`decoder_matrice(Mr)`** qui renvoie le message **M** de départ, éventuellement après correction (si l'on suppose qu'il y a au plus une erreur).

III. Probabilités

On suppose que pour chaque bit transmis, la probabilité d'être transmis avec erreur (autrement dit d'être inversé) est p, avec $0 < p < 1$. Les erreurs de transmission sont supposées indépendantes.

Question 8 : On envoie un message de longueur n et on note N la variable aléatoire égale au nombre de bits transmis avec une erreur. Quelle est la loi de N ?

Question 9 : On suppose que l'on transmet directement le message `M` (de longueur 9). Quelle est la probabilité qu'il soit transmis sans erreur ?

Question 10 : Quelle est la probabilité que le message `Mt` soit transmis sans erreur ? Avec exactement une erreur ? Avec au plus une erreur ?

Question 11 : Comparer la probabilité que `M` soit transmis sans erreur et la probabilité que `M` puisse être retrouvé à partir de `Mr` avec $p = 10^{-4}$.

Corrections

Q. 1 : Le nombre 1 apparait 4 fois dans **M**, donc la parité de **M** est 0. Le message transmis est donc **Mt=[1,0,1,1,1,0,0]**. Si le message reçu est **Mr=[1,0,0,1,1,0,0]**, on détecte un problème car la parité de **[1,0,0,1,1,0]** n'est pas 0. Si le message reçu est **Mr=[0,0,1,1,1,0,1]**, on ne peut pas détecter l'erreur car la parité de **[0,0,1,1,1,0]** est bien 1 : les deux erreurs de transmission se sont compensées.

Q. 2 : Le principe est de partir d'une parité $p = 0$ et de l'inverser à chaque fois que l'on rencontre un 1. Inverser p revient à effectuer **p=1-p**.

```
def parite(L):
    p=0
    for x in L:
        if x==1:
            p=1-p
    return p
print parite([0,1,1])
print parite([0,1,0])
```

```
0 1
```

Q. 3 :

```
def coder_parite(M):
    Mt=M+[parite(M)]
    return Mt
print coder_parite([1,0,1,1,1,0])
```

```
[1, 0, 1, 1, 1, 0, 0]
```

Q. 4 : Notons **Mr=**$[m_0,\dots,m_{\ell-2},m_{\ell-1}]$ où ℓ est la longueur de **Mr**. Le message **Mr** est valide si, et seulement si, la parité de $[m_0,\dots,m_{\ell-2}]$ est $m_{\ell-1}$.

```
def verifier_parite(Mr):
    M=Mr[:-1]
    p=Mr[-1]
    return parite(M)==p
print verifier_parite([1,0,0,1,1,0,0])
print verifier_parite([0,0,1,1,1,0,1])
```

```
False True
```

Q. 5 : On place dispose les éléments du message dans un tableau et on complète avec les parités :

1	1	0	0
0	0	0	0
0	1	0	1
1	0	0	

Le message transmis est

$$\texttt{Mt=}[1,1,0,0,0,0,0,1,0,\underbrace{0,0,1}_{\ell_0,\ell_1,\ell_2},\overbrace{1,0,0}^{c_0,c_1,c_2}]$$

Q. 6 :

```
def coder_matrice(M):
    l0=parite([M[0],M[1],M[2]])
    l1=parite([M[3],M[4],M[5]])
    l2=parite([M[6],M[7],M[8]])
    c0=parite([M[0],M[3],M[6]])
    c1=parite([M[1],M[4],M[7]])
    c2=parite([M[2],M[5],M[8]])
    Mt=M+[l0,l1,l2,c0,c1,c2]
    return Mt
print coder_matrice([1,1,0,0,0,0,0,1,0])
```

```
[1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0]
```

Q. 7 : On considère le message reçu **Mr** et on le décompose en trois parties :

```
Md=Mr[0:9]
lr=Mr[9:12]
cr=Mr[12:]
```

de sorte que `lr[0]`, `lr[1]`, `lr[2]` représentent les valeurs reçues pour ℓ_0, ℓ_1, ℓ_2 et de même pour `cr[0]`, `cr[1]`, `cr[2]`. La partie **Md** correspond à ce qui devrait être le message **M** envoyé. Calculons les valeurs de $\ell_0, \ldots, c_2$ qu'il aurait fallu recevoir si le message envoyé avait été **Md**. Pour cela on calcule le message**Mdt** qu'il aurait fallu transmettre pour envoyer **Md** :

```
Mdt=coder_matrice(Md)
```

On peut donc récupérer les parités des lignes et colonnes pour **Md** avec :

```
l=Mdt[9:12]
c=Mdt[12:]
```

On va maintenant chercher combien il y a de différences entre les listes `l` et `lr` (on note `nl` le nombre de différences) et l'indice `il` correspondant à cette éventuelle différence :

```
nl=0
il=-1
for i in range(3):
    if l[i]!=lr[i]:
        nl=nl+1
        il=i
```

Si on n'a pas trouvé de différence, alors `il` reste égal à -1. On procède de même avec `cr` et `c` (notations `nc` et `ic`) :

```
nc=0
ic=-1
for i in range(3):
    if c[i]!=cr[i]:
        nc=nc+1
        ic=i
```

Comme on l'a vu en classe, il y a plusieurs possibilités :

— Si **nl** et **nc** sont égaux à 1, alors **il** et **ic** donnent l'indice de la ligne et de la colonne où il y a une modification à apporter. L'indice **k** correspondant dans le message **Md** est **3*il+ic** ;

— Dans les autres cas, l'erreur éventuelle porte sur les bits de parité et il n'y a pas de correction à apporter.

On aura donc les instructions :

```
if il==1 and ic==1:
    k=3*il+ic
    Md[k]=1-Md[k]
return Md
```

On regroupe tout ceci en une fonction :

```
def decoder_matrice(Mr):
    Md=Mr[0:9]
    lr=Mr[9:12]
    cr=Mr[12:]
    Mdt=coder_matrice(Md)
    l=Mdt[9:12]
    c=Mdt[12:]
    nl=0
    il=-1
    for i in range(3):
        if l[i]!=lr[i]:
            nl=nl+1
            il=i
    nc=0
    ic=-1
    for i in range(3):
        if c[i]!=cr[i]:
            nc=nc+1
            ic=i
    if il==1 and ic==1:
        k=3*il+ic
        Md[k]=1-Md[k]
    return Md
```

On fait quelques essais :

```
M=[1,1,0,0,0,0,0,1,0]
Mt=coder_matrice(M)
Mr=list(Mt)
Mr[4]=1-Mr[4]
print decoder_matrice(Mr)
```

```
[1, 1, 0, 0, 0, 0, 0, 1, 0]
```

```
Mr=list(Mt)
Mr[12]=1-Mr[12]
print decoder_matrice(Mr)
```

```
[1, 1, 0, 0, 0, 0, 0, 1, 0]
```

Q. 8 : La loi de N est la loi binomiale $\mathscr{B}(n,p)$. En particulier :

$$\mathbf{P}(N=k) = \left(\begin{smallmatrix} n \\ k \end{smallmatrix}\right) p^k (1-p)^{n-k}$$

Q. 9 : Ici $n=9$. Probabilité que le message soit transmis sans erreur :

$$\mathbf{P}(N=0) = (1-p)^9$$

Q. 10 : Ici $n=15$. Probabilité que **Mt** soit transmis sans erreur :

$$\mathbf{P}(N=0) = (1-p)^{15}$$

Probabilité que **Mt** soit transmis avec exactement une erreur :

$$\mathbf{P}(N=1) = \left(\begin{smallmatrix} 1 \\ 15 \end{smallmatrix}\right) p(1-p)^{14} = 15p(1-p)^{14}$$

Probabilité que **Mt** soit transmis avec au plus une erreur :

$$\mathbf{P}(N \leqslant 1) = \mathbf{P}(N=0) + \mathbf{P}(N=1) = (1-p)^{15} + 15p(1-p)^{14}$$

Q. 11 : Le message **M** peut être retrouvé à partir de **Mt** lorsque **Mt** a été transmis avec au plus une erreur. Il faut donc comparer $(1-p)^9$ et $(1-p)^{15} + 15p(1-p)^{14}$.

```
>>> p=1e-4
>>> (1-p)**9
0.9991003599160126
>>> (1-p)**15+15*p*(1-p)**14
0.9999989509095908
```