I. Introduction

Le C++ a été créé en 1983 par Bjarne Stroupstrup alors insatisfait des possibilités offertes par le C. Dans ce cours nous utiliseront pas toutes les fonctionnalité offertes par le C++.
Alors pourquoi utiliser le C++ au lieu du C ? Notamment pour les raisons suivantes
- existence de classe
- assignation et libération de mémoires simplifiées
- procédure d'affichage simplifiée
- chaîne de caractère mieux gérées

1 Avantage des classes

Une classe C++ ressemble de très près à une structure en C. Voici une structure C et une classe C++ permettant de représenter un nombre complexe.

```
typedef struct{
  float re;
  float im;
} Point;
```

```
class Complexe{
  float re;
  float im;
};
```

Maintenant supposons que nous voulions crée une fonction retournant vrai si un tt Complexe est réel et faux sinon.
En C, on ferait :

```
bool est_reel(Complexe Z){
return Z.im==0;
}
```

L'appel est a alors de la forme

```
est_reel(z);
```

Des fonctions de ce genre peuvent apparaître plus ou moins n'importe où dans le programme. En C++ nous déclarons un fonction `est_reel` membre de la class `Complexe`

```
class Complexe{
  float re;
  float im;
  bool est_reel();
};
```

Nous définissons alors la fonction par

```
bool Complexe::est_reel(){
return im==0;
}
```

L'appel est a alors de la forme

```
z.est_reel();
```

En résumé le C++ permet d'associer des fonctions aux classes alors que le C ne le permet pas avec les structures. Un autre avantage de C++ est la surcharge d'opérateurs qui permet de donner un sens à la commande

```
z=z+w
```

Ou `z` et `w` sont des `Complexes`.

2 Gestion mémoires

Supposions que nous souhaitons créer dynamiquement un tableau de int de taille 20. En C on écriera

```
int* tab=(int*)malloc(20*sizeof(int));
```

La libération de ce tableau se fera par

```
free(tab)
```

En C++ on écriera

```
int* tab=new int[20];
```

La libération de ce tableau se fera par

```
delete(tab)
```

Le C++ est aussi plus souple sur le passage de tableau en paramètre et sur l'allocation dynamique de mémoire. Ces points seront détaillés ultérieurement si besoin.

3 La fonction cout

Supposons que `i` représente une variable de type entier contenant la valeur 10 et que l'on souhaite afficher à l'écran

```
La variable i a pour valeur 10.
```

En C, nous utiliserions la commande

```
printf("La variable i a pour valeur %d.\n",i);
```

En C++ nous utiliserions la commande

```
cout<<"La variable i a pour valeur "<<i<<"."<<endl;
```

II. Arithmétique

Un processeur 32 bits peut manipuler des valeurs non signées comprises entre

$$0 \quad \text{et} \quad 2^{32} - 1 = 4,294,967,295$$

et des valeurs signées comprises entre

$$-2^{31} = -2,147,483,648 \quad \text{et} \quad 2^{31} - 1 = 2,147,483,647$$

Un processeur 64 bits peut manipuler des valeurs entières non signées comprises entre

$$0 \quad \text{et} \quad 2^{64} - 1 = 18,446,744,073,709,551,615$$

et des valeurs signées comprises entre

$$-2^{63} = -9,223,372,036,854,775,808 \quad \text{et} \quad 2^{63} - 1 = 9,223,372,036,854,775,808.$$

Les nombres entiers utilisés par l'algorithme de chiffrement RSA sont maintenant codés sur 1024 bits. Il faut plus de 300 chiffres pour les écrires. Les entiers naturellement manipulable en C/C++ sont donc de taille insuffisante.

1 Grand entier non signé

Théorème 1.1. *Soit b un entier naturel supérieur ou égale à 2. Alors, pour tout entier n de $\mathbb{N}$, ils existe un unique entier $\ell \geqslant 0$ et des uniques entiers c_k pour $k = 0, \ldots, \ell$ vérifiant :*

$$n = \sum_{k=0}^{\ell} c_k \cdot b^k$$

L'expression $c_\ell c_{\ell-1} \ \ldots c_1 c_0$ est la numération de l'entier n en base b. Les entiers c_i sont appelés chiffres.

Question 1.2. Comment peut-on représenter un entier de taille "pratiquement" infini en C ?

Solution. A l'aide d'un tableau de chiffre.
A partir du choix d'une base b nous allons représenter un entier n sous forme de tableau stockant les chiffres de la numération de n en base b.

Question 1.3. Un tableau de quel type ? (char, short int, int, long int, unsigned char, ...)

Les type en C/C++ ne sont pas bien défini. Le standard garantit seulement

```
1 == sizeof(char) <= sizeof(short int) <= sizeof(short) <= sizeof(int)
                  <= sizeof(long) <= sizeof(long long)
```

Cependant la réalité est que les types `short int` et `int` sont au moins codé sur 2 octets, que le type `long` sur au moins 4 octets et le `long long` sur au moins 8 octets.
Afin de ne pas perdre de mémoire, nous allons utiliser un tableau de type char. Nous allons donc utiliser une numération en base 256.

Question 1.4. Quel est le plus grand entier (non signé) que peut représenter un tableau de type char de longueur ℓ ?

Solution. $256^\ell - 1$
Le plus grand entier représenté par un tableau de type char de taille ℓ est $256^\ell - 1$. Pour pouvoir manipuler un tableau il faut mémoriser sa taille.

Question 1.5. Quel doit être (au minimum) le type de la variable dans laquelle va être stocké la longueur du tableau de char si on veut pouvoir représenter un entier de l'ordre de grandeur de 2^{1024}.

Solution. On a $2^{1024} = 2^{8\times 128} = 256^{128} < 256^{256} - 1$. Un entier de type `char` est donc suffisant.

Exercice 1.6. Proposer une structure `Entier` permettant de représenter des entiers non signés entre 0 et $256^{65535} - 1 = 2^{524280} - 1$.

Exercice 1.7. Ecrire une fonction `void Affiche(Entier e)` affichant la numération en base 256 de l'entier représenter par e.

Question 1.8. Comment sont représenté les nombres 345 et 62345 comme `Entier`.

Exercice 1.9. Ecrire une fonction `Entier versEntier(unsigned int)` permettant d'initialiser un `Entier` à partir d'une valeur de type `unsigned int`.

1.1 Opérations élémentaires

Exercice 1.10. Ecrire une fonction `int estEgal(Entier e,Entier f)` qui retourne 1 si les entiers e et f sont égaux et 0 sinon.

Exercice 1.11. Ecrire une fonction `int estSuperieur(Entier e,Entier f)` qui retourne 1 pour $e \geqslant f$ et 0 sinon.

Exercice 1.12. Ecrire une fonction `int estInferieur(Entier e,Entier f)` qui retourne 1 pour $e \leqslant f$ et 0 sinon.

Exercice 1.13. Faites "comme à la petite école" les additions $144659+56346$ et $56765+67678$.

Question 1.14. Si un `Entier` e et de taille ℓ et un entier `Entier` f est de taille ℓ', quels sont les tailles possibles pour l'`Entier` représentant $e + f$?

Les tailles possibles sont $\max(\ell, \ell')$ et $\max(\ell, \ell') + 1$.

Exercice 1.15. Ecrire une fonction `Entier addition(Entier e,Entier f)` retournant `e+f`.

Exercice 1.16. Faites "comme à la petite école" les multiplications 1446×563 et 567×6768.

Exercice 1.17. Ecrire une fonction `Entier decalage(Entier e)` qui retourne l'entier $e \times 256$.

Exercice 1.18. Ecrire une fonction `Entier multiplicationSimple(Entier e,unsigned char f)` retournant `e×f`.

Exercice 1.19. Ecrire une fonction `Entier multiplication(Entier e,Entier f)` retournant `e×f`.

2 L'anneau $\mathbb{Z}/n\mathbb{Z}$

2.1 Définition

Pour a, b et n des éléments de $\mathbb{Z}$, on note $a \equiv b \mod n$ si n divise $b - a$. On rappel que la relation $\equiv_n$ est une relation d'équivalence, c'est la "congruence modulo n".
Pour $a \in \mathbb{Z}$, on note $[a]_n$ la casse de a modulo n.

Exercice 2.1. Montrer que l'on a $[a]_n = \{a + kn \mid k \in \mathbb{Z}\}$.

Définition 2.2. Un anneau A est un ensemble non vide muni de 2 opérations $+_A : A \times A \to A$, $\cdot_A : A \times A \to A$ vérifiant :
$i)$ $(A, +_A)$ est un groupe commutatif dont l'élément neutre est noté 0_A.
$ii)$ associativité du produit : pour tout x, y, z de A, on a $x \cdot_A (y \cdot_A z) = (x \cdot_A y) \cdot_A z$
$iii)$ existence de neutre pour $\cdot_A$ (noté 1_A) : pour tout $x \in A$, on a $x \cdot_A 1_A = 1_A \cdot_A x = x$.
$iv)$ distributivité de $\cdot$ sur $+$: pour tout $x, y, z \in A$, on a $x \cdot_A (y +_A z) = x \cdot_A y +_A x \cdot_A z$ et $(y +_A z) \cdot_A x = y \cdot_A x +_A z \cdot_A x$.

Définition 2.3. On dit qu'un anneau $(A, +_A, \cdot_A)$ est commutatif si le produit $\cdot_A$ est commutatif : $\forall x, y \in A (x \cdot_A y = y \cdot_A x)$.

Définition 2.4. Soit $\mathbb{Z}/n\mathbb{Z}$ l'ensemble des classes d'équivalences de la congruence modulo n. Muni des opérations $+$ et $\cdot$ définies par

$$[x] + [y] = [x + y] \quad \text{et} \quad [x] \cdot [y] = [x \cdot y]$$

c'est un anneau commutatif

Exercice 2.5. Calculer la table d'addition et de multiplication de $\mathbb{Z}/6\mathbb{Z}$.

2.2 Exponentiation modulaire

Pour a dans $\mathbb{Z}/n\mathbb{Z}$, on pose $x^p = x \cdot ... \cdot x \mod n$.

Exercice 2.6.

Question 2.7. Calculer 4^{10} dans $\mathbb{Z}/13\mathbb{Z}$.

Question 2.8. Proposer un algorithme en pseudo-langage permettant de calculer la puissance k-ème d'un élément x de $\mathbb{Z}/n\mathbb{Z}$.

Question 2.9. Combien de multiplication sont utilisé pour calculer 4^{10} dans $\mathbb{Z}/13\mathbb{Z}$?

Question 2.10. Pouvez-vous trouver un algorithme en utilisant seulement 4 ?

2.3 Eléments inversibles

Définition 2.11. Soit $(A, +_A, \cdot_A)$ un anneau. On dit qu'un élément x de A^* est inversible s'il existe $y \in A$ tel que $x \cdot_A y = y \cdot_A x = 1_A$. L'ensemble des éléments inversibles de A est noté A^*.

Exercice 2.12. Quels sont les éléments inversibles de $\mathbb{Z}/10\mathbb{Z}$?

Proposition 2.13. *Soit* $(A, +_A, \cdot_A)$ *un anneau. Alors* $(A^*, \cdot_A)$ *est un groupe, appellé* groupe multiplicatif *de* A

Définition 2.14. Soient a et b des éléments de $\mathbb{Z}$. Le plus grand diviseur de a et b, noté $\mathrm{pgcd}(a, b)$ est l'entier $d \in \mathbb{N}$ vérifiant
– d divise a et d divise b,
– si d' divise a et d') divise b alors d' divise d.

Exercice 2.15. Calculer $\mathrm{pgcd}(12, 15)$, $\mathrm{pgcd}(25, -9)$ et $\mathrm{pgcd}(16, 6)$.

Proposition 2.16. *Le groupe multiplicatif de* $\mathbb{Z}/n\mathbb{Z}$ *est*

$$(\mathbb{Z}/n\mathbb{Z})^* = \{\overline{x} \mid \mathrm{pgcd}(x, n) = 1\}.$$

2.4 Le pgcd

Ainsi pour savoir si un élément a de $\mathbb{Z}/n\mathbb{Z}$ est inversible il est suffit de calculer $\mathrm{pgcd}(x, n)$.

Exercice 2.17. Soient a et b des éléments de $\mathbb{Z}$. Monter que les relations suivantes sont vraies :

1. $\mathrm{pgcd}(a, b) = \mathrm{pgcd}(b, a)$
2. $\mathrm{pgcd}(-a, b) = \mathrm{pgcd}(a, b)$
3. $\mathrm{pgcd}(a \mod b, b) = \mathrm{pgcd}(a, b)$

A l'aide de la dernière relation de l'exercice précédant, Euclide a donné son nom à un algorithme efficace pour calculer le pgcd de deux entiers.

Exercice 2.18. Ecriver cet algorithme en pseudo-langage.

Grâce à l'algorithme d'Euclide, nous pouvons calculer des pgcd et donc déterminer les éléments inversibles de $\mathbb{Z}/n\mathbb{Z}$. Nous ne pouvons cependant pas donner l'inverse d'un tel élément.

2.5 Identité de Bezout

Théorème 2.19 (Bezout)**.** *Soient* $a, b \in \mathbb{Z}$*, alors il existe deux entiers* u *et* v *tels que* $au+bv = q$*. En particulier, deux entiers* a *et* b *sont premiers entre eux si et seulement si il existe* $u, v \in \mathbb{Z}$ *tels que* $au + vb = 1$.

Nous pouvons utiliser une version *étendue* de l'algorithme d'Euclide afin de calculer les coefficients de Bezout u et v.

Exercice 2.20.

1. Appliquer l'algorithme d'Euclide aux entiers 17 et 48.
2. "Remonter" l'algorithme pour trouver u et v tels que $17u + 48v = 1$.
3. Proposer un algorithme en pseudo-langage prenant en entrée deux entiers a et b et retournant u et v vérifiant $au + bv = \mathrm{pgcd}(a, b)$.

Exercice 2.21. A l'aide du théorème de Bezout et de l'algorithme établi à l'exercice précédent écrire en pseudo-langage écrire un algorithme

1. qui teste si un nombre a de $\mathbb{Z}/n\mathbb{Z}$ est inversible ;
2. qui retourne l'inverse d'un élément inversible de $\mathbb{Z}/n\mathbb{Z}$.

3 Fonction indicatrice d'Euler

Pour tout entier strictement positif n on note traditionnellement $\varphi(n)$ le nombre d'entiers strictement positifs inférieurs à n et premiers avec n. Ce nombre mesure donc la cardinalité de l'ensemble des éléments inversibles de $\mathbb{Z}/n\mathbb{Z}$, soit $\varphi(n) : \#(\mathbb{Z}/n\mathbb{Z})$.

Exercice 3.1.

1. Montrer que si p est un nombre premier on a $\varphi(p) = p - 1$.
2. Monter de plus que si q est aussi un nombre premier mais différents de p, on a $\varphi(pq) = (p-1)(q-1)$.

4 Application à RSA

Voir slides

III. Matrices

1 Nombre rationnel

Question 1.1. Comment représenter un nombre rationnel en `C++` ?

Aller sur ma page web pour télécharger le squelette de classe pour les nombres rationnels. Noter la présence de la fonction `normalise`. Cette fonction à pour but de mettre n'importe quelle fraction sous forme irréductible. Nous supposerons donc que tous nos rationnels sont représentés par une fraction irréductible.

Exercice 1.2. Ecrire en pseudo-langage puis en `C++` des fonctions peramettant

1. d'additionner deux rationnels
2. de soustraire deux rationnels
3. de multiplier deux rationnnels
4. de diviser un rationnel par un rationnel non nul
5. de compare deux ratiopnnels $:=$, $<$, $>$

2 Matrices

Question 2.1. Comment représenter les matrices en `C++`

On pense tout d'abort un tableau de ligne mais cette représentation consomme trop de mémoire. En effet on a un tableau de pointeur sur des tableaus, chacun d'eux représenter une ligne. Pour une matrice de taille (l, c) il nous faut donc un tableau de taille l dont chacune des cases pointes pers un tableu de taille c. Au total cette représentation nécessite $l + lc$ bloc mémoire.
La représentation qu'on utilisera consiste à voir une matrice comme un tableau simple de taille lc. Ainsi la matrice

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

sera représentée par $[1, 2, 3, 4, 5, 6]$. Le coefficient (i, j) est stocké dans la case $(i-1) \times c + j - 1$ du tableau (les indices des coefficients commence à 1 et celui du tableau à 0).
Aller sur ma page web pour télécharger le squelette de classe pour les nombres rationnels.
On suppose qu'on a une classe `Matrice` possédant les focntions membres :
– `get_nb_lignes()` retournant le nombre de lignes de la matrice
– `get_nb_colonnes()` retournant le nombre de colonnes de la matrice
ainsi qu'une fonction permettant d'accéder aux coefficients. Si `M` est une matrice alors `M[l][c]` désigne le coefficient de `M` se trouvant à l'intersection de la ligne `l` et de la colonne `c`.

Exercice 2.2. Faire à la main le produit $\begin{bmatrix} 4 & 5 \\ 1 & 3 \end{bmatrix} \times \begin{bmatrix} 5 & 1 \\ 2 & 1 \end{bmatrix}$.

Exercice 2.3. Ecrire en pseudo-langage puis en `C++` des fonctions permettant

1. d'additionner deux matrices
2. de multiplier une matrice par un scalaire
3. de multiplier deux matrices

Exercice 2.4. Ecrire en pseudo-langage puis en `C++` les fonctions

1. `mul_ligne(Matrice A,usint i,Rationnel r)` effectuant $L_i \leftarrow rL_i$
2. `add_mul_ligne(Matrice A,usint i,usint j,Rationnel r)` effectuant $L_i \leftarrow L_i + rL_j$
3. `ech_ligne(Matrice A,usint i,usint j)` effectuant $L_i \leftrightarrow L_j$

Exercice 2.5. Appliquer la méthode du pivot de Gauß à la matrice

$$\begin{bmatrix} 0 & 1 & -1 & 4 \\ 0 & 2 & 0 & 2 \\ 2 & 4 & -2 & 2 \\ 2 & 1 & -3 & 2 \end{bmatrix}$$

Exercice 2.6. Ecrire une procédure `Gauss(Matrice A)` appliquant la méthode du pivot de Gauß à la matrice A.

3 Calcul sur les matrices

Proposition 3.1. *Le rang d'une matrice est invariant par opérations élémentaires sur les lignes. Le rang d'une matrice échelonnée réduite est le nombre de ligne non nulle.*

Exercice 3.2. Ecrire une fonction `rang(Matrice A)` retorurnant le rang d'une matrice quelconque.

Exercice 3.3. Notons O l'opération élémentaire de ligne permettant de passer d'une matrice A à une matrice B. Donner une formule liant $\det(A)$ et $\det(B)$ lorsque
– $O = L_i \leftrightarrow L_j$
– $O = L_i \leftarrow cL_i$
– $O = L_i + cL_j$
Indication : considérer les matrices associés aux operations élémentaires sur les lignes

Exercice 3.4. A partir de la fonction appliquant le pivot de Gauss à une matrice, ecrire une fonction `det(Matrice A)` retournant le déterminant de la matrice A.

Exercice 3.5. Ecrire une fonction testant si une matrice est inversible.

Exercice 3.6. Ecrire une fonction retournant l'inverse d'une matrice

IV. Espace vectoriel

Les espaces vectoriels qui nous intéressent sont exclusivement les sous-espaces vectoriels de $\mathbb{Q}^n$ pour un certain entier $n \in \mathbb{N}$.

1 Introduction

Question 1.1. Comment représenter un sous-espace vectoriel de $\mathbb{Q}^n$?

Reponse. Par une base de cette sous-espace vectoriel et donc par une famille génératrice de $\mathbb{Q}^n$.

Exercice 1.2. Proposer une structure `EspaceVectoriel` en `C++` permettant de représenter un sous-espace vectoriel de $\mathbb{Q}^n$.

Solution.

```
struct EspaceVectoriel{
  usint dim; //dimension de l'espace vectoriel
  Matrice* base; //tableau de dim Matrices de taille n*1
};
```

Une base est une famille libre et génératrice. Une famille de vecteurs $u_1, ..., u_k$ est libre si

$$\lambda_1 u_1 + ... + \lambda_k u_k \implies \lambda_1 = ... = \lambda_k = 0.$$

Une famille de vecteurs $u_1, ..., u_k$ est une famille génératrice d'un sous-espace vectoriel F de $\mathbb{Q}^n$ si pour tout v de F il existe des rationnels $\lambda_1, ..., \lambda_k$ tels que $v = \lambda_1 u_1 + ... + \lambda_k u_k$. Pour $u_1, ..., u_k$ une famille de vecteurs de $\mathbb{Q}^n$, on note $\mathrm{Vect}_{\mathbb{Q}}(u_1, ..., u_k)$ le sous espace vectoriel de $\mathbb{Q}^n$ engendré par $u_1, ..., u_k$.

Exercice 1.3. Ecrire une fonction

`Matrice famille_to_mat(Matice *F, usint k)`

qui étant donnée une famille `F` $= (u_1, ..., u_k)$ de vecteurs colonne de même taille retourne la matrice

$$\left[\begin{array}{c} u_1 \\ \hline \vdots \\ \hline u_k \end{array}\right].$$

Solution.

```
Matrice famille_to_mat(Matice *F, usint k)
  n=F[0].nb_lignes()
  Matrice R(k,n) //création d'une matrice à k lignes et n colonnes
  Pour i=1 jusqu'à k faire:
    Pour j=1 jusqu'à n faire:
      R[i][j]=(F[i])[j][1]
  Retourner R
```

Exercice 1.4. Ecrire une fonction `bool est_libre(Matrice* F,usint k)` qui teste si une famille F de k vecteurs de $\mathbb{Q}^n$ est libre ou pas.

Solution. Soient $\mathcal{F} = (u_1, ..., u_k)$ une famille de vecteurs de $\mathbb{Q}^n$. Notons F l'espace vectoriel engendré par $\mathcal{F}$. Comme F est engendré par k vecteurs, on a $\dim F \leqslant k$. Si la famille $\mathcal{F}$ n'est par libre, il existe un vecteur de $\mathcal{F}$ qui est combinaison linéaire des autres. Alors F peut être engendré par $k-1$ vecteurs et on a $\dim F \leqslant k-1$. Ainsi la famille $\mathcal{F}$ est libre si et seulement si $\dim F = k$ et donc si et seulement si le rang de

$$\left[\begin{array}{c} u_1 \\ \hline \vdots \\ \hline u_k \end{array}\right]$$

vaut k. D'ou la fonction :

```
bool est_libre(Matrice* F,usint k)
  Matrice M=famille_to_mat(F,k)
  Retourner rang(M)==k
```

Question 1.5. Comment à partir d'une famille $F = (u_1, ..., u_k)$ de vecteur colonnes de $\mathbb{Q}^n$ construire une famille G telle que
– G soit libre
– $\mathrm{Vect}_{\mathbb{Q}}(G) = \mathrm{Vect}_{\mathbb{Q}}(F)$.

Reponse. Il est nécessaire de commencer par résoudre $\lambda_1 u_1 + ... + \lambda_k u_k = 0$.

2 Système linéaire homogène

Un système linéaire $AX = B$ est homogène si $B = 0$ où A est une matrice de taille $m \times n$ et X un vecteur colonne de taille n.
Un système linéaire homogène à toujours au moins une solution, à savoir $X = 0$.

Proposition 2.1. *Soit $AX = 0$ un système linéaire sur $\mathbb{Q}$. L'ensemble des solutions de ce système est $\mathbb{Q}$ espace vectoriel de dimension finie.*

Exercice 2.2. Résoudre le système $AX = 0$ pour

$$A = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix}, \quad X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Solution. En appliquant le pivot de Gaußà la matrice A, on obtient la matrice

$$B = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix}$$

Les solutions de $AX = 0$ sont donc les solution de $BX = 0$ qui sont aussi les solutions de

$$(S) \begin{cases} x_1 - x_3 &= 0 \\ x_2 - x_3 &= 0 \end{cases}.$$

Les solutions du système $AX = 0$ sont donc

$$\left\{ \begin{bmatrix} x_3 \\ x_3 \\ x_3 \end{bmatrix}, \ x_3 \in \mathbb{Q} \right\} = \mathrm{Vect}_{\mathbb{Q}} \left(\begin{bmatrix} x_3 \\ x_3 \\ x_3 \end{bmatrix} \right)$$

La dimension de l'espace des solutions de $AX = 0$ correspond au nombre de colonnes de A qui ne porterons pas de pivot dans la matrice échelonnée réduite obtenue de A.

Exercice 2.3. Ecrire une fonction `usint dim_sol(Matrice A)` qui retourne la dimension de l'espace vectoriel solution de $AX = 0$ pour un certain X.

Solution. Soit B la matrice échelonnée réduite issue de A le nombre de pivot dans B est exactement le rang de B. Le nombre de colonne dans B qui ne contiennent pas de pivot est donc le nombre de colonnes de B moins son rang.

```
usint dim_sol(Matrice A)
  Retourner A.get_nb_colonnes()-rang(A)
```

Exercice 2.4. Ecrire une fonction `EspaceVectoriel resoud(Matrice A)` qui retourne l'espace vectoriel des solutions de $AX = 0$, le choix de X n' a pas d'importance.

Solution. Notons B la matrice échelonné réduite issue de A. Prenons l'exemple de

$$B = \left[\begin{array}{c|c|cc|c|c|cc} 0 & 1 & 3 & 5 & 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 0 & 1 & 0 & 4 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 3 & 7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right], \quad \text{avec } X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix}$$

Les colonnes portant un pivot dans B sont les colonnes $2, 5$ et 6. Les solutions du système $AX = 0$ sont celles de $BX = 0$ et donc celles de

$$\begin{cases} x_2 + 3x_3 + 5x_4 + x_7 + 3x_8 &= 0 \\ x_5 + 4x_7 + 2x_8 &= 0 \\ x_6 + 3x_7 + 7x_8 &= 0 \end{cases}$$

qui s'écrit encore

$$\begin{cases} x_2 &= -3x_3 - 5x_4 - x_7 - 3x_8 \\ x_5 &= -4x_7 - 2x_8 \\ x_6 &= -3x_7 - 7x_8 \end{cases}$$

Les solutions de $AX = 0$ est alors l'espace vectoriel

$$S = \left\{ \begin{bmatrix} x_1 \\ -3x_3 - 5x_4 - x_7 - 3x_8 \\ x_3 \\ x_4 \\ -4x_7 - 2x_8 \\ -3x_7 - 7x_8 \\ x_7 \\ x_8 \end{bmatrix}, \ (x_1, x_3, x_4, x_7, x_8) \in \mathbb{Q}^5 \right\}$$

On obtient donc

$$S = \text{Vect}_{\mathbb{Q}} \left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -3 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -5 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \\ -4 \\ -3 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -3 \\ 0 \\ 0 \\ -2 \\ -7 \\ 0 \\ 1 \end{bmatrix} \right\}$$

Il est essentiel de repérer les paramètres, qui correspondent aux colonnes sans pivot. On va donc créer un tableau `param` de taille le nombre de colonnes de A. Si la colonne c porte un pivot alors `param[c]=0` si la colonne c est le paramètre numéro 4 alors `param[c]=4`. Ainsi dans l'exemple précédent, on aurait `param=[1,0,2,3,0,0,4,5]`.

```
EspaceVectoriel resoud(A)
  B=Gauss(A)
  //Initialisation de l'espace vectoriel solution S
  EspaceVectoriel S
  n=B.get_nb_colonnes()//taille des vecteurs solutions
  r=rang(B)
  S.dim=n-r//dimension de S
  S.base=new Matrice[S.dim]
  Pour i de 1 jusqu'à S.dim faire:
    S.base[i]=Matrice(n,1)//initialisation des vecteurs de base à 0^n
  //Creation du tableau param
  param=tableau d'usint de taille n//l'indice de 1 à n
  pivot=tableau d'usint de taille r//l'indice de 1 à r
  l=1//ligne courante
  np=1//numero parametre
  Pour j de 1 jusqu'à n faire:
    Si B[l][j]==1://pivot trouve
      param[j]=0
      pivot[l]=j//position pivot ligne l
      l++
    Sinon:
      param[j]=np
      //on met un 1 au vecteur de base correspondant au parametre np
      (S.base[np])[j][1]=1
      np++
  //Suite et fin du remplissage des vecteurs de base de S
  Pour i de 1 jusqu'à r faire:
    Pour j de pivot[i]+1 jusqu'à n faire:
     Si param[j]!=0:
         (S.base[param[j]])[pivot[i]][1]=-B[i][j]
  Retourner S
```

3 Système linéaire

Exercice 3.1. Soit $AX = B$ un système linéaire. Notons S l'ensemble des solutions de ce système, S_H l'espace vectoriel des solutions de $AX = 0$. Montrer que s'il existe une solution particulière Y telle que $AY = B$ alors on a $S = \{Y + Z \mid Z \in S_H\}$.

Exercice 3.2. Ecrire une fonction

`Matrice solution_particuliere(Martice A,Matrice B)`

qui retourne, sil elle existe, une solution particulière du système $AX = B$.

Solution. En reprenant l'exemple de la matrice 8×5 précédent, on note qu'il est seulement nécessaire de donner une affectation aux variables correspondant aux pivots. Pour résoudre le système $AX = B$ on applique la méthode du pivot de Gaußà la matrice augmentée $M = [A|B]$ pour obtenir la matrice $M' = [A'|B']$. Notons r le rang de M' et s le rang de A'. Par définition d'une matrice échelonnée réduite on a $s \geqslant r$. Si $s < r$, alors M' contient une ligne $[0....01]$ qui correspond à l'équation $0 = 1$. Le système n'a dons pas de solution.

```
Matrice solution_particuliere(Matrice A,Matrice B)
  l=A.nb_lignes()
  c=A.nb_colonnes()
  Si B.nb_lignes()!=l:
    Erreur "Système mal formé"
  //Création de la matrice augmentée
  M=Matrice(l,c+1)
  Pour i de 1 jusqu'à l faire:
    Pour j de 1 jusqu'à c faire:
      M[i][j]=A[i][j]
     M[i][c+1]=B[i][1]
  N=Gauss(M)
  r=rang(M)
  Si M[r][c]==0:
    Erreur "Pas de solution"
  X=Matrice[c][1]
  //Recherche de pivot
  l=1//ligne courante
  Pour je de 1 jusqu'à n faire:
    Si N[i][j]==1://Pivot trouve
      X[j][1]=N[i][c+1]
   Retourner X
```

Exercice 3.3. Déduire de l'exercice précédent une fonction

`void resoud(Matrice A,Matrice B)`

qui affiche toutes les solutions du système $AX = B$.

4 Opérations sur les espaces vectoriels

Dans cette partie on suppose que E et F sont deux sous-espaces vectoriels de $\mathbb{Q}^n$ donnée par les base $B_E = (u_1, ..., u_p)$ et $B_F = (v_1, ..., v_q)$.

Question 4.1. Soient $F = (w_1, ..., w_k)$ une famille de vecteur. Comment à partir des solutions de $\lambda_1 w_1 + ... + \lambda_k w_k$ trouvé une famille libre G qui engendre le même espace que F ?

Reponse. Notons S l'espace-vectoriel soluition de $\lambda_1 w_1 + ... + \lambda_k w_k$. La dimension de S correspond aux nombres de paramètres libres du système. Posons $d = \dim S$. Sans perte de généralité, nous renommons les vecteurs $w_1, ..., w_k$ de tels sorte que les paramètre de S soit $\lambda_1, ..., \lambda_d$. Soit $j \in \{1, ..., d\}$, il existe une solution de S avec $\lambda_i = 1$ pour $i = j$ et $\lambda_i = 0$ pour $i = 2, ..., d$ avec $i \neq j$. D'où l'existence de $\lambda_{d+1}, ..., \lambda_k$ tels que

$$u_j + \lambda_{d+1} u_{d+1} + ... + \lambda_k u_k = 0,$$

ce qui implique

$$u_j = -\lambda_{d+1} u_{d+1} - ... - \lambda_k u_k.$$

Le vecteur u_j est donc combinaison linéaire des $u_{d+1}, ..., u_k$, quellque soit $j \in \{1, ..., d\}$. D'où $\text{Vect}(F) = \text{Vect}(w_{d+1}, ..., w_k)$. Comme la dimension de $\text{Vect}(F)$ est $k - d$, la famille $(w_{d+1}, ..., w_k)$ est libre.

Exercice 4.2. Ecrire une fonction

`EspaceVectoriel get_espace_vectoriel(Matrice* F,usint k)`

qui retourne l'espace vectoriel engendré par la famille `F` de `k` vecteurs.

Solution. Posons $F = (w_1, ..., w_k)$. On commence par créer la matrice $A = [w_1|...|w_k]$. On calcule la matrice échelonnée réduite B associée à A. On retourne la famille obtenue de F en conservant les w_i d'indice i si la colonne C_i porte un pivot.

```
EspaceVectoriel get_espace_vectoriel(Matrice* F,usint k)
 EspaceVectoriel R
 Si k==0:
   R.dim=0
   Retourner R
  //On est certain que F[0] existe
  n=F[0].get_nb_lignes()
  Matrice A(n,k) //création d'une matrice à n lignes et k colonnes
  Pour j de 1 jusque k faire:
    Si F[j-1].get_nb_lignes()!=n:
      Erreur "Famille non définie"
    Pour i de 1 jusque n faire:
      A[i][j]=(F[j-1])[i][1]
   B=Gauss(A)
   R.dim=rang(B)
   R.base=nouveau tableau de R.dim matrices de taille nx1
   //Recherche des pivots
   l=1//ligne courante
   Pour j de 1 jusque k faire:
     Si B[l][j]==1://pivot trouve
       R.base[l-1]=F[j-1]
       l++
    Retourner R
```

Exercice 4.3. Montrer que

$$E + F = \{u \in \mathbb{Q}^n, \text{tel que } \exists (v, w) \in E \times F \text{ avec } u = v + w\}$$

est un sous-espace vectoriel de $\mathbb{Q}^n$.

Solution. On $0 \in E+F$ car $0 = 0+0$ avec $0 \in E$ et $0 \in F$, donc $E+F$ est non vide. Soient x et y deux éléments de $E+F$ et $\lambda \in \mathbb{Q}$. Par définition de $E+F$, il existe v_x, v_y de E et w_x, w_y de F tels que $x = v_x + w_x$ et $y = v_y + w_y$. On a donc

$$\begin{aligned} x + \lambda y &= v_x + w_x + \lambda(v_y + w_y) \\ &= v_x + w_y + \lambda v_y + \lambda w_y \\ &= (v_x + \lambda v_y) + (w_x + \lambda w_y) \end{aligned}$$

Comme v_x et v_y sont dans E et que E est un espace-vectoriel, on a $v_x + \lambda v_y \in E$. De même, on $w_x + \lambda w_y \in F$. On a donc montrer que $x + \lambda y$ appartient à $E+F$ et donc que $E+F$ est un sous-espace vectoriel de $\mathbb{Q}^n$.

Question 4.4. Donner une famille génératrice de l'espace vectoriel $E+F$.

Reponse. Les vecteurs de $E+F$ sont combinaisons linéaires de vecteur de E et de vecteur de F. Les vecteurs de E sont combinaison linéaire de vecteurs de B_E et les vecteurs de F sont combinaisons linéaires de vecteurs de B_F. Ainsi, les vecteurs de $E+F$ sont combinaison linéaire de vecteurs de $B_E \cup B_f$.

Exercice 4.5. Ecrire une fonction

`EspaceVectoriel somme(EspaceVectoriel E,EspaceVectoriel F)`

qui retourne l'espace vectoriel $E+F$.

Solution.

```
EspaceVectoriel somme(EspaceVectoriel E,EspaceVectoriel F)
 Si E.dim==0:
   Retourner F
 Si F.dim==0:
   Retourner E
 n=E.base[0].get_nb_lignes();
 Si F.base[0].get_nb_lignes()!=n:
   Erreur "Espaces vectoriels incompatibles"
 G=tableau de E.dim+F.dim matrices de taille nx1
 Pour i de 0 jusque E.dim-1 faire:
   G[i]=E.base[i]
 Pour i de 0 jusque F.dim-1 faire:
   G[E.dim+i]=F.base[i]
 Retourner get_espace_vectoriel(G,E.dim+F.dim)
```

Exercice 4.6. Monter que $E \cap F$ est un sous-espace vectoriel de $\mathbb{Q}^n$.

Solution. Soient u et v des vecteurs de $E \cap F$ et λ un rationnel. Montrons que $u + \lambda v$ est un élément de $E \cap F$. Comme u et v sont des vecteurs de E qui est un $\mathbb{Q}$-espace vectoriel, on a $u + \lambda v \in E$ De même, on a $u + \lambda v \in F$. On a donc bine $u + \lambda v \in E \cap F$. L'ensemble $E \cap F$ est donc bien un sous-espace vectoriel de $\mathbb{Q}^n$.

Exercice 4.7. Supposons $n = 3$ et posons $u_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$, $u_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$, $v_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$, $v_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$. Posons $E = \mathrm{Vect}_{\mathbb{Q}}(u_1, u_2)$ et $F = \mathrm{Vect}_{\mathbb{Q}}(v_1, v_2)$. Donner une base, ou au moins une famille génératrice de $E \cap F$.

Solution. Les dimensions possibles pour $E \cap F$ sont $0, 1$ ou 2. Si $\dim(E \cap F) = 2$ alors $E \cap F = E$ et $E \cap F = F$ car $\dim(E) = \dim(F) = 2$. Or le vecteur $\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ appartient à E mais pas à F. On a donc $\dim(E \cap F) = 0$ ou 1. A partir des relations

$$w = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = u_1 - u_2 = v_1 - v_2$$

on a $w \in E \cap F$. On en déduit alors $E \cap F = \mathrm{Vect}_{\mathbb{Q}}(w)$.

Question 4.8. Comment à partir des solutions de $\lambda_1 u_1 + ... + \lambda_p u_p = \mu_1 v_1 + ... + \mu_q v_q$ construire une base de $E \cap F$? On pourra commencer avec les E et F de l'exercice précédent.

Solution. Posons

$$A = [u_1|...|u_k| - v_1|...| - v_\ell] \quad \text{et} \quad X = \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_k \\ \mu_1 \\ \vdots \\ \mu_\ell \end{bmatrix}$$

et notons S l'espace vectoriel solution de $AX = 0$. Si X appartient à S alors $w = \lambda_1 u_1 + ... + \lambda_k u_k$ est dans E. Comme on a aussi $w = \mu_1 v_1 + ... + \mu_\ell v_\ell$, on a $w \in F$ et donc $w \in E \cap F$. Les k premières composantes des vecteurs de S donnent les coefficient des vecteurs de $E \cap F$ en la base de E. On a don $\dim(E \cap F) = \dim(S)$ et que la base de E est donner par la base de S.

Exercice 4.9. Ecrire une fonction

```
EspaceVectoriel intersection(EspaceVectoriel E,EspaceVectoriel F)
```

qui retourne l'espace vectoriel $E \cap F$.

5 Applications linéaires

Question 5.1. Proposer une structure `C++` permettant de représenter une application linéaire de E dans F où E est un sev de $\mathbb{Q}^n$ et F un sev de $\mathbb{Q}^m$.

Reponse. Il nous faut stocker E et F. En posant $k = \dim(E)$ et $\ell = \dim(F)$, l'application est représenter par une matrice de taille $\ell \times k$.

Exercice 5.2. Ecrire une fonction

```
ApplicationLineaire composition(ApplicationLinearire f,ApplicationLineaire g)
```

qui retourne, lorsqu'elle existe, la composée $f \circ g$.

Le rang de l'application linéaire φ de E dans F est la dimension de $\varphi(E)$.

Théorème 5.3. *Pour toute application φ de E dans F, on a*

$$\dim E = \mathrm{rang}(\varphi) + \dim \ker \varphi$$

Exercice 5.4. A partir du théorème du rang, écrire une fonction

```
bool est_injective(Application phi)
```

qui teste si une application linéaire de E dans F est injective ou pas.

Exercice 5.5. A partir du théorème du rang, écrire une fonction

```
bool est_surjective(Application phi)
```

qui teste si une application linéaire de E dans F est surjective ou pas.

Exercice 5.6. Ecrire une fonction

```
EspaceVectoriel kernel(ApplicationLineaire phi)
```

qui retourne le noyau de `phi`.

Supposons que nous ayons implémenter les matrices à coefficients dans $\mathbb{Q}[X]$.

Exercice 5.7. Ecrire une fonction

```
Polynome polynome_caracteristique(ApplicationLineaire f)
```

qui retourne le polynôme caractéristique de f.

6 Diagonalisation

Exercice 6.1. Soient $P(X) = a_n X^n + ... + a_1 X + a_0$ un polynôme de $\mathbb{Q}[X]$. Montrer que si la fraction irréductible $\frac{p}{q}$ est racine de P alors p divise a_0 et que q divise a_n.

V. Codes correcteurs d'erreurs

Les codes correcteurs d'erreurs ont leur source dans un problème très concret lié à la transmission de données. Dans la grande majorité des cas, la transmission de données se fait en utilisant une voyer de communication, la canal de communication, qui n'est pas entièrement fiable. Autrement, les données, lorsqu'elles circulent sur cette voie, sont susceptibles d'être altérée.

Exemple. de canal de communications.
– onde radio (FM, Wifi, Bluetooth, ...)
– fils conducteur (ADSL, Ethernet, ...)
– onde sonore (parole, ...)

Le même phénomène se produit lorsque l'on stocke de l'information sur un support. L'information lue peut être plus ou moins différentes que celles enregistrées.

Exemple. Mémoire vive, disque dur, CDROM, ...

Reprenons l'exemple de communication radio. La présence de parasites sur la ligne va perturber le son de la vois. Il y'a essentiellement deux approches possibles.
– augmenter la puissance de l'émission
– ajouter de la redondance à l'information
Augmenter la puissance a ses limites, pour des raisons diverses : consommation énergétique, nuisance, coût de l'émetteur, ... L'autre solution consiste à ajouter des données, ce qui donne lieu au code des aviateurs qui diront "Alpha Tango Charlie" dans le but de transmettre "ATC" à travers le radio. La séquence " Alpha Tango Charlie ", même avec friture sera plus reconnaissable pour l'oreille humaine qu'un "ATC' déformé.

Un code correcteur peut avoir plusieurs buts de manière non exclusive.
– détecter les erreurs
– corriger les erreurs
Dans le protocole de communication TCP seul la détection est assurée. En effet, la correction est réalisée par une nouvelle demande de transmission du message.
Dans d'autres situation on peut pas faire de nouvelle demande de transmission, c'est le cas notamment pour le stockage de données. Par exemples les codes correcteurs utilisés avec les CD sont conçus pour corrigée jusqu'à 4096 bits consécutifs, ce qui correspond à une rayure de plus d'un millimètre de large.
Les objectifs à atteindre ne sont donc pas du tout les mêmes en fonction de l'utilisation de l'information et du çanal de communication.

Notre modèle est le suivant :
– on considère que le message est une suite de bits
– regroupés par bloc de k bits ($k = 1$, ou 4, ou 8 ...)
– et que chaque bits à une probabilité non nulle d'être inversé.

Pour pouvoir corriger une éventuelle erreur dans un bloc de bits ont doit nécessairement ajouter une information supplémentaire.

Exemple. Code par adjonction d'un bit de parité $(8, 9)$. On découpe notre message initial en bloc de 8 bits. On transforme ensuite chaque bloc en un bloc de 9 bits en ajoutant un bit à la fin de chaque bloc de telle sorte que la somme des bits du nouveaux blocs soit toujours paire.

Exercice 0.1. Coder les blocs 01101011 et 00110101. Que ce passe-t-il si un bit est modifié ? et dans le cas de deux bits.

Solution. La somme des bits de 01101011 donne 5 qui est impaire, il est donc coder en 011010111. La somme des bits de 00110101 donne 4 qui est paire, il est donc coder en 001101010. La modification d'un bit va augmenter ou diminuer de 1 le nombre de bits égaux à 1. La somme des bits sera donc impaire et sera détectée. Dans le cas de deux erreurs, le nombre de bits égaux à 1 va augmenter de 2 ou diminuer de 2 ou rester le même (changement d'un 0 en 1 et d'un 1 en 0. La somme des bits restera donc paire, les erreurs ne peuvent pas être détectées.

Si une erreur des produit, on peut la détecter, mais pas la localiser : on ne peut pas corriger notre bloc, il faut recommencer la transmission. Si d'avantage d'erreurs de produisent, on n'est même pas sûr de détecter le problème.

Qu'attend-on d'un bon code ?
– l'information ne doit pas être trop diluée,
– on doit pouvoir détecter et corriger un nombre raisonnable d'erreurs,
– l'algorithme de codage doit être rapide,
– l'algorithme de décodage aussi.

1 Paramètre d'un code

Un bloc de k bits sera indifféremment appelé bloc, mot ou vecteur. L'ensemble des mots de k bits sera noté $\{0, 1\}^k$. On parlera indifféremment de bits ou de lettres.

Un mot de k bits sera noté $b_1b_2...b_k$ ou éventuellement $\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix}$.

Définition 1.1. . Un code correcteur de paramètre (k, n) est application injective $\varphi : \{0, 1\}^k \to \{0, 1\}^n$ appelé *encodage*. Le paramètre k est la *dimension* du code et n sa *longueur*.

Question 1.2. Pourquoi l'encodage doit-il être injectif ? Existe t-il un lien entre k et n ?

Reponse. Si le codage n'était pas injectif alors ils existeraient deux mots m et m' tels que $\varphi(m) = \varphi(m') = c$. Lors de la réception après transmission sans erreurs du mot c on ne saurait pas dire s'il est le codé de m ou de m'. Comme φ est injective on a que la cardinalité de $\{0, 1\}^k$ est la même que $\varphi(\{0, 1\}^k)$ à savoir 2^k il doit donc y avoir au moins 2^k élément dans $\{0, 1\}^n$ qui est de cardinalité 2^n, ce qui implique $k \leqslant n$.
Dans $\{0, 1\} = \mathbb{Z}/2\mathbb{Z} = \mathbb{F}_2$, on a $1 + 1 = 0$.

Exercice 1.3. En vous inspirant du code vue précédemment, imaginé un code par adjonction d'un bit de parité de paramètres $(k, k + 1)$ pour $k \geqslant 1$.

Solution. On définit l'application φ de $\{0,1\}^k$ dans $\{0,1\}^{k+1}$ en posant

$$\varphi(b_1b_2...b_k) = b_1b_2...b_kb_{k+1}$$

où $b_{k+1} = b_1 + b_2 + ... + b_k$. On a alors

$$b_1 + b_2 + ... + b_{k+1} = b_1 + b_2 + ... + b_k + b_1 + b_2 + ... + b_k = 1 + 2(b_1 + ... + b_k) = 0$$

Définition 1.4. Soit φ un code de paramètre (k, n). L'ensemble $C = \{\varphi(m),\ m \in \{0,1\}^k\}$ est appelé *image* du code φ. Les éléments de C sont les *mots de code* de φ.

Exercice 1.5. Considérons l'application $\varphi : \{0,1\} \to \{0,1\}^3$ définie par $\varphi(0) = 000$ et $\varphi(1) = 111$. Précisée chacune des notions introduite pour ce code. Ce code sera appelé code de répétition pure $(1, 3)$.

Solution. On a $k = 1$, $n = 3$ et $C = \{000, 111\}$.

Exercice 1.6. Préciser ce que peuvent devenir les mots 000 et 111 après $0, 1$ et 2 erreurs. Parmi les mots trouvés repérées ceux qui sont des mots de code pour le code de répétition pure $(1, 3)$. Combien d'erreurs ce code peut-il détecter ? Corriger ?

Exercice 1.7. On a
- après 0 erreurs
  - $000 \to \{000\}$
  - $111 \to \{111\}$
- après 1 erreurs
  - $000 \to \{001, 010, 100\}$
  - $111 \to \{110, 101, 011\}$
- après 2 erreurs
  - $000 \to \{011, 101, 110\}$
  - $111 \to \{100, 010, 001\}$
- après 3 erreurs
  - $000 \to \{111\}$
  - $111 \to \{000\}$

Il peut détecter jusqu'à 2 erreurs mais en corrigé une seule.

Si une, ou même deux erreurs se produisent, le mot reçu n'est pas un mot de code, l'erreur est donc détectée.
Comment corriger ? Si le mot reçu n'est pas un mot de code, la probabilité qu'il se soit produit une erreur est plus importante que celle qui se soit produit deux erreurs. Il est donc plus raisonnable de corriger par le mot de code le plus "proche". On peut alors corriger une erreur mais pas 2.

Définition 1.8. Soient m et m' deux mots de $\{0,1\}^k$. On appelle *distance de Hamming* entre m et m', et on note $d(m, m')$ le nombre de lettres distinctes de m et m'. On appelle *poids de Hamming* de m et on note $w(n)$ le nombre de lettres non nulles de m.

Exercice 1.9. Quel est le poids de Hamming de 01100111. Quelle est la distance de Hamming entre
- 0001001 et 0101001,
- 0000110 et 0001100.

Solution. On a $d(0001001, 0101001) = 1$ et $d(1010101, 0101010) = 7$.

Exercice 1.10. Montrer que pour tout m, m' de $\{0,1\}^k$, on a $d(m,m') = w(m+m')$. En déduire que pour tout m, m' et c de $\{0,1\}^k$ on a $d(m+c, m'+c) = d(m,m')$.

Solution. Soient $m = b_1 b_2 ... b_k$ et $m' = b'_1 b'_2 ... b'_k$ des mots de $\{0,1\}^k$. La distance de Hamming est de nombre de lettres distinctes entre w et w'. On a donc $d(m,m') = \#\{i \in \{1,...,k\} \mid b_i \neq b'_i\}$. Comme $0+0=0, 1+1=0, 0+1=1$ et $1+0=1$ on remarque que deux bits sont différents si et seulement si leur somme vaut 1. On a donc

$$d(m,m') = (b_1+b'_1)+(b_2+b'_2)+...+(b_k+b'_k) = w((b_1+b'_1)(b_2+b'_2)...(b_k+b'_k)) = w(m+m')$$

Définition 1.11. Soit φ un code d'image C. On appelle *capacité de détection* de φ et on note e_d le plus grand nombre d'erreurs que φ permet de détecter quelque soit le message. On appelle *capacité de correction* de φ et on note e_c le plus grand nombre d'erreurs que φ permet de corriger quelque soit le message. On appelle *distance minimale* de φ et on note d_φ la plus petite distance de Hamming non nulle entre deux mots de code.

Proposition 1.12. *On a* $e_d = d_\varphi - 1$ *et* $e_c = \left\lfloor \dfrac{d_\varphi - 1}{2} \right\rfloor$.

Exercice 1.13. Que vaut d_φ, e_d et e_c dans le cas du code de répétition pure $(1,3)$.

Solution. On a $C = \{000, 111\}$ et donc $d_\varphi = d(000,111) = 3$. D'où $e_d = 2$ et $e_c = \lfloor \frac{2}{2} \rfloor = 1$.

Exercice 1.14. Que vaut d_φ, e_d et e_c dans le cas du code de bit de parité $(8,9)$.

Solution. Les mots $m = 000000000$ et $m' = 100000001$ sont deux mots de C. Comme $d(m,m') = 2$ on a $d_\varphi \leqslant 2$. Montrons que d_φ n'est pas 1. Supposons par l'absurde qu'il existe deux mots m et m' de C tels que $d(m,m') = 1$. Posons $m = b_1...b_9$, $m' = b'_1...b'_9$. Comme m et m' sont dans C, on a $b_1 + ... + b_9 = 0$ et $b'_1 + ... + b'_9 = 0$. Comme $d(m,m') = 1$ il existe $i \in \{1,...,9\}$ tel que $b_i \neq b'_i$ et $b_j = b'_j$ pour tout $j \neq i$. On a déjà vue que le relation $b_i \neq b'_i$ est équivalente à $b_i + b'_i = 1$. On a donc $b'_i = b_i + 1$. Ce qui donne

$$0 = b'_1 + ... + b'i + ...b_k = b_1 + ... + b_i + 1 + ... + b_k = 1 + b_1 + ... + b_k = 1 + 0 = 1$$

On a donc nécessairement $d_\varphi = 2$. D'où $e_d = 1$ et $e_c = \lfloor \frac{1}{2} \rfloor = 0$.

2 Codes linéaires

Définition 2.1. Un code φ de paramètre (n,k) est dit linéaire s'il existe une matrice $G \in M_{n,k}(\mathbb{F}_2)$ de rang k, telle que $\forall m \in \{0,1\}^k$, $\varphi(m) = G \times m$. La matrice G est appelée matrice génératrice du code φ.

Exercice 2.2. Les codes répétition pure $(1,3)$ et bit de parité $(8,9)$ sont-ils linéaire ? Si oui, quelles sont leurs matrices génératrices.

Solution. Pour le code de répétition pure $(1,3)$, on a $\varphi(b_1 = b_1 b_1 b_1$. Pour

$$G = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \text{on a} \quad G\begin{bmatrix} b_1 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_1 \\ b_1 \end{bmatrix}$$

ce que l'on veux. Pour le code bit de parité $(8,9)$ on a $\varphi(b_1...b_b) = b_1...b_8b_9$ avec $b_9 = b_1 + ... + b_8$. Pour

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad \text{on a} \quad G \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7 + b_8 \end{bmatrix}$$

qui est ce qu'on veux.

Exercice 2.3. Etudier le code linéaire ayant

$$G = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

comme matrice génératrice. Ce code est de dimension 2 et de longueur 4. On a

$$G \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \; G \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \; G \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \text{ et } G \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

On a donc $C = \{0000, 1011, 0101, 1110\}$. De

$$d(0000, 1011) = 3, d(0000, 0101) = 2, d(0000, 1110) = 3,$$
$$d(1011, 0101) = 3, d(1011, 1110) = 2 \text{ et } d(0101, 1110) = 3,$$

on obtient $d_\varphi = 2$ et donc $e_d = 1$ et $e_c = 0$.

Exercice 2.4. Proposer une structure `C++` permettant de représenter un code linéaire.

Solution. On a seulement besoin d'un matrice.

```
struct CodeCorrecteur{
  Matrice G;
};
```

Proposition 2.5. *Soit φ un code linéaire de paramètre (n,k) alors son image C est un sous-espace vectoriel de $\{0,1\}^n$.*

Exercice 2.6. Montrons que C est un sous-espace vectoriel de $\{0,1\}^n$ qui est un $\mathbb{F}_{\neq}$-espace vectoriel. Soit G la matrice génératirece de φ. Tout d'abord C est non vide car $0 = G \times 0$. Soient c et c' deux élément de C, alors il existent m et m' tels que $c = G \times m$ et $c' = G \times m'$. On a $c + c' = G \times (m + m')$ et donc $c + c'$ appartient à C. Les seuls éléments de $\{0,1\} = \mathbb{F}_2$ sont 0 et 1. Multiplié un vecteur u de C par 0 donne 0 qui est dans C et le multiplié par 1 donne u qui est aussi dans C. L'ensemble C est donc un sous-espace vectoriel de $\{0,1\}^n$.

Proposition 2.7. *Soit φ un code linéaire d'image C. Alors la distance minimale d_φ de φ est égale au plus petit poids non nul d'un mot de C.*

Démonstration. Notons p le plus petit poids non nul d'un mot de C. La distance minimale est la plus petite distance de Hamming entre deux mots de code. Soit c et c' dans C tel que $d_\varphi = d(c, c')$. Comme φ est linéaire $c' - c$ est encore un mot de code. On a donc $d_\varphi = d(0, c - c') = w(c - c')$. Ainsi $d_\varphi \geqslant p$. Soit m un mot de C tel que $w(m) = p$. Alors $d(0, m) = p$. Les mots 0 et m étant des mots de code, on a $d_\varphi \leqslant d(0, m) = p$. On a donc montrer $d_\varphi = p$. □

Proposition 2.8. *Soit φ un code linéaire de paramètre (k, n) et d'image C. Alors on a $d_\varphi \leqslant n - k + 1$. Un code pour lequel on a égalité est dit MDS (Maximum Distance Separable).*

Démonstration. D'après la proposition précédente, il est suffisant de montrer qu'il existe un mot de code de poids inférieur ou égale $n + 1 - k$. Un mot de $\{0, 1\}^n$ dont les $k - 1$ dernière composante sont nulles à un poids inférieur ou égale à $n - k + 1$. Notons D l'espace vectoriel des mots dont les $k - 1$ dernières composantes sont nulles. La dimension de D est le nombre de composantes libres, à savoir $n - k + 1$. Par un résultat générale d'algèbre linéaire on a $n \geqslant \dim(C+D) = \dim(C)+\dim(D)-\dim(C\cap D)$. et donc $n \geqslant k+n-k+1-\dim(C\cap D)$, ce qui implique $\dim(C \cap D) > 0$. Il existe donc un mot non nul dans $C \cap D$. Ce qui signifie qu'il existe un mot de code avec ses k dernières composantes nulles. Le poids de ce mot étant inférieur à $n - k + 1$, on a $d_\varphi \leqslant n - k + 1$. □

Exercice 2.9. Donner un minorant sur la longueur d'un code code linéaire de dimension k détectant d erreurs.

Solution. S'il est de longueur n alors sa distance minimale d_φ est inférieur ou égale à $n - k + 1$. Dans ce cas il détecte $n - k$ erreurs. On a donc $n - k \geqslant d$ et donc $n \geqslant k + d$.

Exercice 2.10. Les codes bits parité $(8, 9)$, répétition pure $(1, 3)$ et celui donnée par matrice génératrice sont-ils MDS ?

Solution. Pour $(8, 9)$, on a montrer $d_\varphi = 2$, qui est égale à $9 - 8 + 1$. N'importe quel code de bit parité est MDS. Pour $(1, 3)$ on a calculé $d_\varphi = 3$, qui est égale à $3 - 1 + 1$. Il est donc bien MDS. Pour le code donnée par matrice génératrice, on a $k = 2$ et $n = 4$. On a calculé $d_\varphi = 2$. Or $n - k+ = 4 - 2 + 1 = 3$, ce code n'est donc pas MDS.

Définition 2.11. Soit φ un code linéaire de matrice génératrice G. On appelle *matrice de contrôle* de φ toute matrice $H \in M_{n-k,n}(\mathbb{F}_2)$ telle que $H.m = \vec{0} \Leftrightarrow m \in C$.

Définition 2.12. Un code φ de paramètre (k, n) est dit systématique si pour tout $m \in \{0, 1\}^k$, le mot m est un préfixe de $\varphi(m)$.

Exercice 2.13. Montrer qu'un code de paramètre (k, n) est systématique si et seulement si sa matrice génératrice est de la forme $\begin{bmatrix} I_k \\ G' \end{bmatrix}$ où G' est une matrix de $M_{n-k,k}(\mathbb{F}_2)$.

Solution. Soient φ un code linéaire systématique et G sa matrice génératrice. Notons $L_1, ..., L_n$ les n lignes de G. Soit $m = \begin{bmatrix} b_1 \\ \vdots \\ b_k \end{bmatrix}$ un mot de $\{0, 1\}^k$. On a

$$G \times m = \begin{bmatrix} L_1 \\ \vdots \\ L_n \end{bmatrix} \times m = \begin{bmatrix} L_1 \times m \\ \vdots \\ L_n \times m \end{bmatrix}$$

Comme φ est systématique m soit être un préfixe de $G \times m$. Ce qui implique $L_i \times m = b_i$ pour $i = 1, ..., k$. On a donc $L_i = \begin{bmatrix} 0 & \dots & 0 & 1 & 0 & \dots & 0 \end{bmatrix}$ où le 1 est en position i. On a donc bien $G = \begin{bmatrix} I_k \\ G' \end{bmatrix}$ où G' est une matrix de $M_{n-k,k}(\mathbb{F}_2)$. Montrons la réciproque. Si $G = \begin{bmatrix} I_k \\ G' \end{bmatrix}$ alors $G \times m = \begin{bmatrix} m \\ G' \times m \end{bmatrix}$ et m est un préfixe de $G \times m$. Le code est donc systématique.

Proposition 2.14. *Soit φ un code systématique de paramètre (k, n) et de matrice génératrice $\begin{bmatrix} I_k \\ G' \end{bmatrix}$ alors la matrice $H = \begin{bmatrix} G' & I_{n-k} \end{bmatrix}$ est une matrice de contrôle de φ.*

Exercice 2.15. Démontrer cette proposition.

Solution. Montrons d'abord que pour tout mode de code c de $\{0,1\}^n$ on a $Hc = 0$, ce qui revient à montrer que $\mathrm{Im}(G)$ est inclus dans $\ker(H)$. Si c est un mot de code alors il existe $m \in \{0,1\}^k$ tel que $c = Gm$. D'où

$$Hc = HGm = \begin{bmatrix} G' & I_{n-k} \end{bmatrix} \begin{bmatrix} I_k \\ G' \end{bmatrix} m = \begin{bmatrix} G'I_k + G'I_{n-k} \end{bmatrix} m = \begin{bmatrix} 2G' \end{bmatrix} m = \begin{bmatrix} 0 \end{bmatrix} m = 0$$

On a donc montrer $\mathrm{Im}(G) \subseteq \ker(H)$. Pour avoir égalité il suffit de montrer l'égalité des dimensions. Par construction de G, on a $C = \mathrm{Im}(G)$ et donc $\dim(C) = \dim(\mathrm{Im}(G)) = \mathrm{rang}(G) = k$. De $H = \begin{bmatrix} G' & I_{n-k} \end{bmatrix}$, on obtient $\dim(\mathrm{Im}(H)) = \mathrm{rang}(H) = n - k$. Par le théorème du rang, on a $\dim\{0,1\}^n = \mathrm{rang}(H) + \dim(\ker(H))$. On a donc $\dim(\ker(H)) = n - (n-k) = k$. On a donc bien $\ker(H) = \mathrm{Im}(G)$ et H est une matrice de contrôle de φ.

Exercice 2.16. Donner la matrice de controle des code bit de parité $(8, 9)$, répétitions pure $(1, 3)$ et de celui donné par matrice génratrice.

Solution. La matrice génératrice du code $(8, 9)$ est

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

une de ses matrices de controle est donc

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

La matrice génératrice du code $(1, 3)$ est

$$G = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

une de ses matrices de controle est donc

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

Une des matrice de controles du code dont la matrice génératrice est

$$G = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

est la matrice

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}.$$

Définition 2.17. Soit φ un code de paramètre (k, n), de matrice génératrice G et de contrôle H. On se fixe un mot de source x (de longueur k). Le mot de code correspondant sera $\varphi(x) = y$. S'il y a eu des erreurs durant la transmission, on recoit z. On appelle *mot erreur* associé à z, le mot e tel que $z = y + e$. On appelle *syndrome* de z le mot Hz.

On vérifie immédiatement qu'on a $Hz = H(y + e) = He$. Le syndrome ne dépend que de la "maladie" (erreur) et non du "patient" (mot a transmettre).

Définition 2.18. L'ensemble des syndromes S_z de z est appellé *classe litérale* de z.

Principe de décodage Soit φ un code linéaire de paramètre (k, n), corrigeant e_c erreurs et de matrice de contrôle H.

1. On recoit le mot z transmis avec de possibles erreurs.
2. On calcul le syndrome s de z par $s = Hz$.
3. Si s vaut 0, on ne détecte pas d'erreur et on retourne $\varphi^{-1}(z)$.
4. Sinon on recherche l'erreur e de plus petit poids possible telle que $He = s$.
5. Si le poid $w(e)$ est inférieur ou égale à e_c, on retourne $\varphi^{-1}(z + e)$.
6. Sinon afficher "Impossible de corriger les erreurs".

Il nous reste à voir comment calculer $\varphi^{-1}(c)$ pour un mot c de $\mathbb{F}_2^n$ (utiliser au 2 et 5) et comment trouver e (ligne 4). Si φ est un code systématique, alors m est le préfixe de longuer k de $\varphi(m)$. Dans ce cas $\varphi^{-1}(c)$ et le préfixe de longueur k de c.

3 Table de dacodage

Pour trouver e, nous allons construire une *table de décodage*. Soit φ un code linéaire de paramètre (k, n). Les matrices de contrôle associé à φ sont de taille $(n - k) \times n$. L'ensemble des syndromes possibles est donc $\mathbb{F}_2^{n-k}$.
Pour calculer la table de décodage de φ, on liste tous les syndrômes de φ. Puis on liste tous les mots z de $\mathbb{F}_2^n$ par poids croissant. Pour chaque z, on calculer Hz. Au syndrome s on associe alors le premier mot z apparu tel que $Hz = s$. On arrete ce procéder dés qu'on associer un mot à chaque syndrome. Le mot associé à un syndrome s est alors l'erreur de poid minimal donnant$s$.

Exercice 3.1. Calculer la table de décodage pour les codes bits de parité $(8, 9)$, répétition pure $(1, 3)$ et celui donner par matrice génératrice. Pour chacun des codes respectivement, décoder les mots recus 101010100, 101 et 1100.

Solution. Pour le code bit de parité $(8, 9)$, on a $H = \begin{bmatrix}1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\end{bmatrix}$. L'ensemble des syndromes est $\mathbb{F}_2^{n-k} = \mathbb{F}_2 = \{0, 1\}$. On liste les mots z de $\mathbb{F}_2^n = \mathbb{F}_2^9$ par poid croissant :

$z \in \mathbb{F}_2^9$	$s = Hz$
000000000	0
000000001	1

On obtient alors la table de décodage

syndrome	erreur
0	000000000
1	000000001

Si on recoit le mot $z = 101010100$. On calcule

$$s = H \times z = \begin{bmatrix}1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\end{bmatrix} \times \begin{bmatrix}1\\0\\1\\0\\1\\0\\1\\0\\0\end{bmatrix} = 0,$$

on retourne alors le préfixe de longueur 8 de z à savoit 10101010.
Pour le code de répétition pure $(1, 3)$, on a $H = \begin{bmatrix}1 & 1 & 0\\1 & 0 & 1\end{bmatrix}$. L'ensemble des syndromes est $\mathbb{F}_2^{n-k} = \mathbb{F}_2^2 = \{00, 01, 10, 11\}$. On liste les mots z de $\mathbb{F}_2^n = \mathbb{F}_2^3$ par poid croissant :

$z \in \mathbb{F}_2^3$	$s = Hz$
000	00
001	01
010	10
100	11

Détaillons le calcul du syndrome de la troizième ligne. On a

$$s = Hz = \begin{bmatrix}1 & 1 & 0\\1 & 0 & 1\end{bmatrix} \times \begin{bmatrix}0\\1\\0\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix}$$

La table de décodage est donc

syndrome	erreur
00	000
01	001
10	010
11	100

Si on recoit le mot $z = 101$. On calcule

$$s = H \times z = \begin{bmatrix}1 & 1 & 0\\1 & 0 & 1\end{bmatrix} \times \begin{bmatrix}1\\0\\1\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix}$$

D'après la table de décodage, l'erreur associé à 10 est 010, on retournr le préfixe de longueur 2 de $z + 010 = 101 + 010 = 111$, à savoir 11.
Le code correteur de matrice génératrice

$$G = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

à la matrice

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

comme matrice de controle. L'ensemble des syndomes est $\mathbb{F}_2^{n-k} = \mathbb{F}_2^2 = \{00, 01, 10, 11\}$. On liste les mots z de $\mathbb{F}_2^n = \mathbb{F}_2^4$ par poid croissant :

$z \in \mathbb{F}_2^4$	$s = Hz$
0000	00
0001	01
0010	10
0100	01
1000	11

La table de décodage est donc

syndrome	erreur
00	0000
01	0001
10	0100
11	1000

Si on recoit le mot $z = 1100$, on calcule

$$s = H \times z = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

D'apres lea table de décodage, l'erreur associée au syndrome 10 est 0100 qui est de poids 1 or ce code ne corrige aucune erreur, on ne peut donc pas finir le décodage.