Université Pierre et Marie Curie

Rapport du projet 4

Algorithme de Berlekamp-Massey pour les codes correcteurs d'erreurs

Phan Nguyet NGUYEN
Mohammed Yacine BERREZOUG

Encadrant : M. Jérémy Berthomieu

25 Mai 2016

Table des matières

1 Introduction

Le but du projet est de présenter l'algorithme de Berlekamp-Massey et son application aux codes correcteurs d'erreurs.
On commence par considérer l'exemple ci-dessous pour comprendre les problèmes apparaissant lors d'une transmission de messages.

Exemple :

On suppose que A veut envoyer des messages à B. Pour chaque envoi, A n'envoie qu'un seul message. A s'attend à ce que le message reçu soit bien celui envoyé. Pendant la transmission, une erreur se produit dans le message envoyé.

Problème 1 : *Détection de l'erreur*
On suppose que le code que A et B utilisent pour communiquer est $\{00, 01, 10, 11\}$ et que A a envoyé 00. Avec l'hypothèse qu'il existe une seule erreur dans le message envoyé, à la réception, B a reçu donc soit 10, soit 01. Ainsi, B n'a aucun moyen pour savoir si le message est le bon.
Pour pouvoir détecter l'erreur, on ajoute *le bit de parité* à la fin de chaque message. Si le nombre du bit 1 dans le message est paire, on ajoute 0, sinon on ajoute 1. Le nouveau code est $\{000, 011, 101, 110\}$. Si A a envoyé 000 et s'il y a qu'une seule erreur, B reçoit donc soit 100, soit 010, soit 001. Donc, B peut savoir que le message reçu n'est pas bon.
Problème 2 : *Correction de l'erreur*
On suppose maintenant que le code utilisé est $\{000, 011, 101, 110\}$ et que A envoie 000. Á la réception, B a reçu 100. Avec l'hypothèse qu'il existe qu'une seule erreur, le bon message est soit 000, soit 101, soit 110. Pourtant, B n'a aucune façon de retrouver le message envoyé à partir de celui reçu.
Pour résoudre ce problème, on peut *répéter* les deux premiers bits, i.e, si A veut envoyer *abc*, A va envoyer donc *abcab*. Le nouveau code cette fois ci est $\{00000, 01101, 10110, 11011\}$. Si B reçoit 10000, les possibilités de message envoyé sont $\{00000, 11000, 10100, 10010, 10001\}$. Donc, le message envoyé est 00000.

Les deux problèmes ci-dessus nous amènent à la recherche d'un code ayant de la capacité de détecter l'erreur et de la corriger. Dans ce projet, on souhaite étudier les codes B.C.H (Bose, Chaudhuri, Hocquenghem) utilisés dans les CD, DVD, SSD. C'est qu'au décodage de ce code apparaîtra le problème suivant :

Problème :
Étant donnée une suite $\mathbf{u} = (u_i)_{i \in N}$.
On cherche une relation de récurrence de l'ordre minimum que la suite satisfait.

Exemple :
Étant donnée la suite $(0, 1, 1, 2, 3, 5, 8, 13)$.

La relation de récurrence est $u_{i+2} = u_i + u_{i+1}$.

Dans ce projet, on étudiera des algorithmes qui permettent de résoudre le **Problème** dont une version matricielle de l'algorithme de Berlekamp-Massey. Ensuite, on présentera la théorie des codes correcteurs d'erreurs et comment le *Problème 1* et le *Problème 2* sont modélisés en mathématiques. A la fin du rapport, on trouvera l'application de l'algorithme de Berlekamp-Massey aux codes correcteurs d'erreurs.

2 Études des algorithmes

2.1 Algorithme naïf

2.1.1 Analyse.

L'algorithme recherché doit retourner une suite $\mathbf{a} = (a_0, a_1, ..., a_{n-1})$ tel que $a_0u_i + a_1u_{i+1} + .. + a_{n-1}u_{i+n-1} = u_{i+n}$.

Exemple :
Étant donnée la suite $(0, 1, 1, 2, 3, 5, 8, 13)$.
La relation de récurrence est $u_{i+2} = u_{i+1} + u_i$.

L'algorithme doit donc retourner (a_0, a_1) tel que : $a_0u_i + a_1u_{i+1} = u_{i+2}$.
Ou,

$$\begin{gathered} a_0 0 + a_1 1 = 1 \\ a_0 1 + a_1 1 = 2 \\ ... \\ a_0 5 + a_1 8 = 13 \end{gathered}$$

Ou

$$\begin{gathered} a_0 0 + a_1 1 = 1 \\ a_0 1 + a_1 1 = 2 \end{gathered}$$

Donc,

$$(a_0, a_1) = (1, 1)$$

L'exemple ci-dessus indique qu'étant donnée la suite $\mathbf{u}$, on peut trouver facilement la suite $\mathbf{a}$ à partir d'un système linéaire. Le problème qui nous reste est la valeur de n.
L'algorithme naïf que l'on propose est de parcourir toutes valeurs possibles de n. Pour chaque valeur de n, on va résoudre un système linéaire :

$$A_n x = b_n,$$

Où

$$A_n = \begin{pmatrix} u_0 & u_1 & \cdots & u_{n-1} \\ u_1 & u_2 & \cdots & u_n \\ \vdots & \vdots & \ddots & \vdots \\ u_{n-1} & u_n & \cdots & u_{2n-2} \end{pmatrix}, b_n = \begin{pmatrix} u_n \\ u_{n+1} \\ \vdots \\ u_{2n-1} \end{pmatrix}$$

et puis vérifier si la solution (s'il y en a une) est bien une relation de récurrence pour la suite (u_i).
On remarque que si la suite $\mathbf{u} = (u_i)_{i \in N}$ est $(u_0, u_1, ..., u_{N-1})$ alors la taille n de la matrice A_n et le vecteur b_n doit garantir que $n \leq \lfloor \frac{N}{2} \rfloor$.
Exemple :
Étant donnée la suite $(0, 1, 1, 2, 3, 5, 8, 13)$.

- Étape 1 : $n = 1$
  Chercher a tel que $a.u_0 = u_1$, i.e $a.0 = 1$.
  Il n'existe pas un tel a.
  Passe à l'étape 2.
- Étape 2 : $n = 2$
  Chercher (a_0, a_1) tel que :

$$\begin{pmatrix} u_0 & u_1 \\ u_1 & u_2 \end{pmatrix} \times \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} u_2 \\ u_3 \end{pmatrix}$$

  ou,

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \times \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

  donc,

$$(a_0, a_1) = (1, 1)$$

  La relation est donc $u_{i+2} = u_i + u_{i+1}$.

- Étape 3 : Tester cette relation sur toute la suite entière.
  On a tester si $\begin{pmatrix} a_0 & a_1 \end{pmatrix} \times \begin{pmatrix} u_i \\ u_{i+1} \end{pmatrix} = u_{i+2}$.
  Elle est satisfaite.
  On retourne $u_{i+2} = u_i + u_{i+1}$.

Algo1 :

if $u_0 \neq 0$ **then**
  $x \leftarrow u_1/u_0$
  if $xu_i = u_{i+1}$ **then**
    return x
  end if
else

$n \leftarrow 2$
while $n \leq \lfloor \frac{N}{2} \rfloor$ **do**

if $\det A_n \neq 0$ **then**
Résoudre $A_n x = b$
if $x_0 u_i + ... + x_{n-1} u_{i+n-1} = u_{i+n}$ est satisfaite pour la suite **u** **then**
return x
else
$n \leftarrow n + 1$
end if
else
$n \leftarrow n + 1$
end if
end while
end if

2.1.2 Un cas générique de Algo1

On considère une suite simple $(1, 2, 3, 5, 8)$.
Algo1 s'arrête en $n = 2$ et retourne la relation $u_{i+2} = u_{i+1} + u_i$. Si on remplace 8 par 7 à la fin de la suite, voir $(1, 2, 3, 5, 7)$, comme la relation n'est plus correcte sur la nouvelle suite, **Algo1** a tendance à incrémenter n par 1, voir $n = 3$, ce qui est impossible pour construire le vecteur b_3,

$$b_3 = \begin{pmatrix} 5 \\ 7 \\ ? \end{pmatrix}$$

L'algorithme Berlekamp-Massey (**AlgoB-M**)que l'on va présenter à la suite permet d'éviter ce problème de **Algo1**.

— Pour la suite $(1, 2, 3, 5, 8)$, **AlgoB-M** retourne le même résultat que **Algo1** : $u_{i+2} = u_{i+1} + u_i$.
— Pour la suite $(1, 2, 3, 5, 7)$, **AlgoB-M** retournera $u_{i+3} = u_i + 2u_{i+1} - 2u_{i+2}$, i.e,

$$5 = 3 + 2 \times 2 - 2 \times 1.$$
$$7 = 5 + 2 \times 3 - 2 \times 2.$$

2.2 Algorithme Berlekamp-Massey

2.2.1 Une autre façon d'interprétation du problème

. On reprend la suite $(1, 2, 3, 5, 8)$.
Dans **Algo1**, la relation de récurrence $u_{i+2} = u_i + u_{i+1}$ est déduit à partir de :

$$\begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix} \times \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix}$$

Dans **AlgoB-M**, l'interprétation sera sous forme :

$$\begin{pmatrix}1 & -1 & -1\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}0 & 0 & 0\end{pmatrix}$$

La raison pour laquelle il est écrit sous cette forme va être mentionnée à la suite.
Le déroulement de **AlgoB-M** appliqué sur la suite $(1, 2, 3, 5, 8)$ est résumé comme suivant :

$$\begin{pmatrix}1\end{pmatrix} \times \begin{pmatrix}1\end{pmatrix} = \begin{pmatrix}1\end{pmatrix}$$

$$\Downarrow$$

$$\begin{pmatrix}1 & -2\end{pmatrix} \times \begin{pmatrix}2 & 3\\ 1 & 2\end{pmatrix} = \begin{pmatrix}0 & -1\end{pmatrix}$$

$$\Downarrow$$

$$\begin{pmatrix}1 & -2 & 1\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}0 & 1 & 1\end{pmatrix}$$

$$\Downarrow$$

$$\begin{pmatrix}1 & -1 & -1\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}0 & 0 & 0\end{pmatrix}$$

Ainsi, le but de **AlgoB-M** est de construire un vecteur annulant la matrice dont chaque colonne correspond à une vérification, ce qui est que si la multiplication du vecteur avec une colonne égale à 0, ce vecteur est une relation récurrente pour les éléments de cette colonne.
On peut observer que dans un premier temps **AlgoB-M** a trouvé le vecteur $\begin{pmatrix}1 & -2\end{pmatrix}$ qui propose une relation sur les deux premiers éléments de la suite, voir $\{1, 2\}$, pourtant la vérification sur $\{2, 3\}$ a échouée, vu que $\begin{pmatrix}1 & -2\end{pmatrix} \times \begin{pmatrix}3\\ 2\end{pmatrix} \neq 0$. En corrigeant le vecteur $\begin{pmatrix}1 & -2\end{pmatrix}$, **AlgoB-M** a trouvé le vecteur $\begin{pmatrix}1 & -2 & 1\end{pmatrix}$, qui est une relation sur les trois premiers éléments de la suite, voir $\{1, 2, 3\}$, mais la vérification sur $\{2, 3, 5\}$ a échouée. **AlgoB-M** continue à corriger $\begin{pmatrix}1 & -2 & 1\end{pmatrix}$ en $\begin{pmatrix}1 & -1 & -1\end{pmatrix}$, ce nouveau vecteur maintenant est correct pour $\{1, 2, 3\}$ et $\{2, 3, 5\}$. La vérification sur $\{3, 5, 8\}$ est aussi satisfaite, car $\begin{pmatrix}1 & -1 & -1\end{pmatrix} \times \begin{pmatrix}8\\ 5\\ 3\end{pmatrix} = 0$ donc **AlgoB-M** s'arrête de corriger ce vecteur et le retourne comme une relation de récurrence pour la suite $(1, 2, 3, 5, 8)$.

2.2.2 Les opérations spéciales de AlgoB-M

On va voir plus en détail comment **AlgoB-M** corrige les vecteurs. Tout d'abord, on présente deux opérations importantes dans **AlgoB-M**.

— Décalage du vecteur à gauche et augmentation simultanée de la taille de la matrice.

$$\begin{pmatrix}1 & -2\end{pmatrix} \times \begin{pmatrix}2 & 3\\ 1 & 2\end{pmatrix} = \begin{pmatrix}0 & \underline{-1}\end{pmatrix}$$

$$\Downarrow$$

$$\begin{pmatrix}1 & -2 & 0\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}\underline{-1} & * & *\end{pmatrix}$$

Ainsi, cette opération permet de déplacer -1 à gauche.

— Décalage du vecteur à droite et augmentation simultanée de la taille de la matrice.

$$\begin{pmatrix}1\end{pmatrix} \times \begin{pmatrix}1\end{pmatrix} = \begin{pmatrix}\underline{1}\end{pmatrix}$$

$$\Downarrow$$

$$\begin{pmatrix}0 & 0 & 1\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}\underline{1} & * & *\end{pmatrix}$$

Ainsi, cette opération permet d'augmenter le taille de vecteur à droite en gardant les éléments existant à gauche.

Le mécanisme de ces deux opérations explique la raison pour laquelle **AlgoB-M** choisit l'interprétation dont on a parlé avant.
Les deux opérations ci-dessus vont permette de corriger le vecteur.
Pour mieux comprendre les fonctionnalités de ces deux opérations, considérons deux questions suivantes.
Question 1. Étant donné que :

$$\begin{pmatrix}1\end{pmatrix} \times \begin{pmatrix}1\end{pmatrix} = \begin{pmatrix}\underline{1}\end{pmatrix}$$

$$\begin{pmatrix}1 & -2\end{pmatrix} \times \begin{pmatrix}2 & 3\\ 1 & 2\end{pmatrix} = \begin{pmatrix}0 & \underline{\mathbf{-1}}\end{pmatrix}$$

Comment peut-on utiliser les deux opérations pour obtenir $\underline{\mathbf{0}}$ *à la place de* $\underline{\mathbf{-1}}$?

— D'abord, on décale -1 à gauche en appliquant la première opération :

$$\begin{pmatrix}1 & -2\end{pmatrix} \times \begin{pmatrix}2 & 3\\ 1 & 2\end{pmatrix} = \begin{pmatrix}0 & \underline{\textbf{-1}}\end{pmatrix}$$

$$\Downarrow$$

$$\begin{pmatrix}1 & -2 & 0\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}\underline{\textbf{-1}} & * & *\end{pmatrix}$$

— On augmente la taille du vecteur $\begin{pmatrix}\underline{1}\end{pmatrix}$ en appliquant la deuxième opération sur $\begin{pmatrix}1\end{pmatrix} \times \begin{pmatrix}1\end{pmatrix} = \begin{pmatrix}\underline{1}\end{pmatrix}$:

$$\begin{pmatrix}0 & 0 & 1\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}\underline{1} & * & *\end{pmatrix}$$

— Il nous reste qu'à multiplier $\begin{pmatrix}\underline{1} & * & *\end{pmatrix}$ par $\frac{-1}{1} = -1$ et puis soustraire les deux vecteurs :

$$\begin{pmatrix}1 & -2 & 0\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}\underline{\textbf{-1}} & * & *\end{pmatrix}$$

—

$$\frac{-1}{1} \times \begin{pmatrix}0 & 0 & 1\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \frac{-1}{1} \times \begin{pmatrix}\underline{1} & * & *\end{pmatrix}$$

$$\begin{pmatrix}1 & -2 & 1\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}\underline{\textbf{0}} & * & *\end{pmatrix}$$

Ainsi, on a réussit à obtenir $\underline{\textbf{0}}$ à place de $\underline{\textbf{-1}}$.

Question 2. On a déjà obtenu que,

$$\begin{pmatrix}1 & -2 & 1\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}0 & \underline{\textbf{1}} & 1\end{pmatrix}$$

Maintenant comment peut-on obtenir $\underline{\boldsymbol{0}}$ à la place de $\underline{\boldsymbol{1}}$?

On peut reprendre les procédures avant avec la première opération appliquée sur $\begin{pmatrix}1 & -2 & 1\end{pmatrix}$ et la deuxième sur $\begin{pmatrix}1\end{pmatrix}$, pourtant $\begin{pmatrix}1 & -2\end{pmatrix} \times \begin{pmatrix}2 & 3\\ 1 & 2\end{pmatrix} = \begin{pmatrix}0 & -1\end{pmatrix}$ nous propose un vecteur de la même forme $\begin{pmatrix}0 & C & *\end{pmatrix}$ (où C est non-nul) que le vecteur $\begin{pmatrix}0 & 1 & 1\end{pmatrix}$.

Applique la deuxième opération sur le vecteur $\begin{pmatrix}1 & -2\end{pmatrix}$ pour augmenter sa taille, on obtient :

$$\begin{pmatrix}0 & 1 & -2\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}0 & -1 & -1\end{pmatrix}$$

On multiplie ce vecteur avec le coefficient et on soustrait les deux vecteurs :

$$\begin{pmatrix}1 & -2 & 1\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}0 & \underline{\mathbf{1}} & 1\end{pmatrix}$$

—

$$\frac{1}{-1} \times \begin{pmatrix}0 & 1 & -2\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \frac{1}{-1} \times \begin{pmatrix}0 & -1 & -1\end{pmatrix}$$

$$\begin{pmatrix}1 & -1 & -1\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}0 & \underline{\mathbf{0}} & 0\end{pmatrix}$$

Ainsi on a réussit à obtenir $\underline{\mathbf{0}}$ à la place de $\underline{\mathbf{1}}$.

Question 3.

A la *Question 2*, on a trouvé la relation de récurrence pour le suite $(1, 2, 3, 5, 8)$. On ajoute $\{13, 21\}$ à la fin de la suite $(1, 2, 3, 5, 8)$.

Comment peut-on veut vérifier si la relation trouvée reste encore correcte sur la nouvelle suite $(1, 2, 3, 5, 8, 13, 21)$ *?*

Il suffit d'appliquer la deuxième opération comme suivant :

$$\begin{pmatrix}1 & -1 & -1\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}0 & 0 & 0\end{pmatrix}$$

$$\Downarrow$$

$$\begin{pmatrix}1 & -1 & -1 & 0\end{pmatrix} \times \begin{pmatrix}5 & 8 & 13 & 21\\ 3 & 5 & 8 & 13\\ 2 & 3 & 5 & 8\\ 1 & 2 & 3 & 5\end{pmatrix} = \begin{pmatrix}0 & 0 & 0 & 0\end{pmatrix}$$

Cela nous dit que la relation est encore correcte sur la nouvelle suite.

2.2.3 Exemples complets sur AlgoB-M

On a vu les ingrédients de **AlgoB-M** et vu comment ils sont utilisés. Les exemples suivants montrent le déroulement complet de cet algorithme.
Exemple 1 **AlgoB-M** appliqué sur la suite $(1, 2, 3, 5, 8)$.

(a) Phase d'initialisation.

$$\begin{pmatrix}1\end{pmatrix} \times \begin{pmatrix}1\end{pmatrix} = \begin{pmatrix}1\end{pmatrix}$$

$$\begin{pmatrix}1 & 0\end{pmatrix} \times \begin{pmatrix}2 & 3\\ 1 & 2\end{pmatrix} = \begin{pmatrix}\underline{\mathbf{2}} & 3\end{pmatrix}$$

(b) Corrige $\underline{\mathbf{2}}$ en 0

$$\begin{pmatrix}1 & 0\end{pmatrix} \times \begin{pmatrix}2 & 3\\ 1 & 2\end{pmatrix} = \begin{pmatrix}\underline{\mathbf{2}} & 3\end{pmatrix}$$

—

$$\frac{2}{1} \times \begin{pmatrix}0 & 1\end{pmatrix} \times \begin{pmatrix}2 & 3\\ 1 & 2\end{pmatrix} = \frac{2}{1}\begin{pmatrix}1 & 2\end{pmatrix}$$

$$\begin{pmatrix}1 & -2\end{pmatrix} \times \begin{pmatrix}2 & 3\\ 1 & 2\end{pmatrix} = \begin{pmatrix}\underline{\mathbf{0}} & \underline{\mathbf{-1}}\end{pmatrix}$$

(c) Corrige $\underline{\mathbf{-1}}$ en 0
Cette partie est déjà faite à la Question 1,

$$\begin{pmatrix}1 & -2 & 0\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}\underline{\mathbf{-1}} & -1 & -2\end{pmatrix}$$

—

$$\frac{-1}{1} \times \begin{pmatrix}0 & 0 & 1\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \frac{-1}{1} \times \begin{pmatrix}1 & 2 & 3\end{pmatrix}$$

$$\begin{pmatrix}1 & -2 & 1\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}\underline{\mathbf{0}} & \underline{\mathbf{1}} & 1\end{pmatrix}$$

(d) Corrige **1** en 0
Cette partie est déjà faite à la Question 2,

$$\begin{pmatrix}1 & -2 & 1\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}\mathbf{0} & \mathbf{1} & 1\end{pmatrix}$$

—

$$\frac{1}{-1} \times \begin{pmatrix}0 & 1 & -2\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \frac{1}{-1} \times \begin{pmatrix}0 & -1 & -1\end{pmatrix}$$

$$\begin{pmatrix}1 & -1 & -1\end{pmatrix} \times \begin{pmatrix}3 & 5 & 8\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}\mathbf{0} & \mathbf{0} & 0\end{pmatrix}$$

(e) Retourne le vecteur $\begin{pmatrix}1 & -1 & -1\end{pmatrix}$

Example 2 Comment **AlgoB-M** arrive-t-il à trouver la relation récurrente pour la suite $(1, 2, 3, 5, 7)$.

(a) Phase d'initialisation.

$$\begin{pmatrix}1\end{pmatrix} \times \begin{pmatrix}1\end{pmatrix} = \begin{pmatrix}1\end{pmatrix}$$

$$\begin{pmatrix}1 & 0\end{pmatrix} \times \begin{pmatrix}2 & 3\\ 1 & 2\end{pmatrix} = \begin{pmatrix}\mathbf{2} & 3\end{pmatrix}$$

(b) Corrige **2** en 0

$$\begin{pmatrix}1 & 0\end{pmatrix} \times \begin{pmatrix}2 & 3\\ 1 & 2\end{pmatrix} = \begin{pmatrix}\mathbf{2} & 3\end{pmatrix}$$

—

$$\frac{2}{1} \times \begin{pmatrix}0 & 1\end{pmatrix} \times \begin{pmatrix}2 & 3\\ 1 & 2\end{pmatrix} = \frac{2}{1} \begin{pmatrix}1 & 2\end{pmatrix}$$

$$\begin{pmatrix}1 & -2\end{pmatrix} \times \begin{pmatrix}2 & 3\\ 1 & 2\end{pmatrix} = \begin{pmatrix}\mathbf{0} & \mathbf{-1}\end{pmatrix}$$

(c) Corrige **-1** en 0

$$\begin{pmatrix}1 & -2 & 0\end{pmatrix} \times \begin{pmatrix}3 & 5 & 7\\ 2 & 3 & 5\\ 1 & 2 & 3\end{pmatrix} = \begin{pmatrix}\mathbf{-1} & -1 & -3\end{pmatrix}$$

—

$$\frac{-1}{1} \times \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 3 & 5 & 8 \\ 2 & 3 & 5 \\ 1 & 2 & 3 \end{pmatrix} = \frac{-1}{1} \times \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & -2 & 1 \end{pmatrix} \times \begin{pmatrix} 3 & 5 & 8 \\ 2 & 3 & 5 \\ 1 & 2 & 3 \end{pmatrix} = \begin{pmatrix} \underline{\mathbf{0}} & \underline{\mathbf{1}} & 0 \end{pmatrix}$$

(d) Corrige $\underline{\mathbf{1}}$ en 0

$$\begin{pmatrix} 1 & -2 & 1 \end{pmatrix} \times \begin{pmatrix} 3 & 5 & 8 \\ 2 & 3 & 5 \\ 1 & 2 & 3 \end{pmatrix} = \begin{pmatrix} \underline{\mathbf{0}} & \underline{\mathbf{1}} & 0 \end{pmatrix}$$

—

$$\frac{1}{-1} \times \begin{pmatrix} 0 & 1 & -2 \end{pmatrix} \times \begin{pmatrix} 3 & 5 & 7 \\ 2 & 3 & 5 \\ 1 & 2 & 3 \end{pmatrix} = \frac{1}{-1} \times \begin{pmatrix} 0 & -1 & -1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & -1 & -1 \end{pmatrix} \times \begin{pmatrix} 3 & 5 & 7 \\ 2 & 3 & 5 \\ 1 & 2 & 3 \end{pmatrix} = \begin{pmatrix} \underline{\mathbf{0}} & \underline{\mathbf{0}} & \underline{\mathbf{\text{-}1}} \end{pmatrix}$$

(e) Corrige $\underline{\mathbf{\text{-}1}}$ en 0

$$\begin{pmatrix} 1 & -1 & -1 & 0 \end{pmatrix} \times \begin{pmatrix} 5 & 7 & ? & ? \\ 3 & 5 & 7 & ? \\ 2 & 3 & 5 & ? \\ 1 & 2 & 3 & ? \end{pmatrix} = \begin{pmatrix} \underline{\mathbf{0}} & \underline{\mathbf{\text{-}1}} & ? & ? \end{pmatrix}$$

—

$$\begin{pmatrix} 0 & 0 & 1 & -2 \end{pmatrix} \times \begin{pmatrix} 5 & 7 & ? & ? \\ 3 & 5 & 7 & ? \\ 2 & 3 & 5 & ? \\ 1 & 2 & 3 & ? \end{pmatrix} = \begin{pmatrix} \underline{\mathbf{0}} & \underline{\mathbf{\text{-}1}} & ? & ? \end{pmatrix}$$

$$\begin{pmatrix} 1 & -1 & -2 & 2 \end{pmatrix} \times \begin{pmatrix} 5 & 7 & ? & ? \\ 3 & 5 & 7 & ? \\ 2 & 3 & 5 & ? \\ 1 & 2 & 3 & ? \end{pmatrix} = \begin{pmatrix} \underline{\mathbf{0}} & \underline{\mathbf{0}} & ? & ? \end{pmatrix}$$

(f) Retourne le vecteur $\begin{pmatrix}1 & -1 & -2 & 2\end{pmatrix}$

Remarque : Dans **AlgoB-M**, après chaque étape, on obtient une formule $xM = v$ où v a la forme $(0, \cdots, 0, C, \cdots)$ avec C non-nul et les vecteurs x tels que $x.M = v$. On constate que les x et C sont utiles quand on veut corriger des éléments non-nul. Dans la description suivant, les x, C sont respectivement sauvegardés dans la matrice X et le vecteur V.

AlgoB-M :

$x \leftarrow (1)$
$M \leftarrow (u[1])$
$v \leftarrow (xM)$
$X \leftarrow (Matrix0)$
$V \leftarrow (Vecteur0)$
$p \leftarrow 0$
while v != 0 **do**
    Décalage x à gauche 1 pas.
    Décalage M 1 pas.
    $v \leftarrow (xM)$
end while
while v != 0 **do**
    $p \leftarrow$ la position du premier non nul de v
    if $X[p] = 0$ **then**
        mettre à jour X,V
        On cherche j tel que $j < p$, j est le plus proche de p
        if j existe **then**
            $p \leftarrow j$
        else
            Décalage x à gauche
            Décalage M
            $v \leftarrow (xM)$
            $p \leftarrow$ la position du premier non nul de v
        end if
    end if
    Décalage $X[p]$ à droite jusqu'à ce que sa taille soit égal à celle de x.
    $x \leftarrow x - (v[p]/V[p]) * X[p]$
    $v \leftarrow (xM)$
end while

3 La théorie du code correcteur d'erreurs

3.1 Modélisation

Avant de présenter des outils mathématiques dans ce domaine, on revient aux problèmes mentionnés au début du rapport. On a vu des problèmes rencontrés et des solutions proposées. Dans cette partir, on chercher à modéliser ces

problèmes et généraliser les solutions.

Problème 1 : *Détection de l'erreur.*
On a vu que le code $\{00, 01, 10, 11\}$ n'a pas de capacité de détecter l'erreur. Pourtant, si on ajoute *le bit de parité*, le nouveau code $\{000, 011, 101, 110\}$ peut le faire. On remarque que la somme des bits d'un message du nouveau code est toujours 0. S'il y une erreur, la somme sera différent de 0, le message reçu ne se trouve donc pas dans le code. Pour généraliser le problème, on va reprendre la propriété de la somme des bits d'un message, plus précisément, on va définir une fonction sur l'ensemble des messages dont valeur vaut 0 quand elle est appliquée sur un bon message, en revanche, quand elle est appliquée sur un message erroné, sa valeur sera différent de 0

Problème 2 : *Correction de l'erreur.*
On a vu que le code $\{110, 101, 100, 110\}$ a de la capacité de détecter l'erreur mais ne peut pas corriger même une erreur. Avec l'hypothèse qu'il y a une seule erreur, le mot envoyé du celui reçu 100 est probablement 000 ou 101 ou 110. Pour trouver la propriété du code utilisé qui cause ce problème, on va se concentrer sur ses mots. Pour cela, on fixe deux mots 000 et 110, et on génère des mots erronés à partir de ces deux mots avec l'hypothèse qu'il y a seulement une erreur.

Message erroné à partir de 000	001	010	100
Message erroné à partir de 110	010	100	111

On trouve que 100 appartient à les deux lignes du tableau, ce qui explique pourquoi on a difficulté de corriger 100.
Pourtant le code $\{00000, 01101, 10110, 11011\}$ que l'on a proposé pour résoudre ce problème n'a pas de cette propriété si on essaie de générer des mots erronés en une erreur.

Message erroné à partir de 00000	10000	01000	00100	00010	00001
Message erroné à partir de 01101	11101	00101	01001	01111	01100
Message erroné à partir de 10110	00110	11110	10010	10100	10111
Message erroné à partir de 11011	01011	10011	11111	11001	11010

On trouve que 10000 n'appartient qu'à la première ligne, ce qui dit que le message 10000 est corrigé en 00000.
On reviendra à ce problème plus tard pour voir quelle propriété du code qui permet de corriger les erreurs.

3.2 Une brève présentation de la théorie code correcteur d'erreurs

Cette section s'inspire de la présentation faite dans *Algèbre discrète et codes correcteurs*, Odile Papini, Jacques Wolfmann, Springer-Verlag Berlin Heidelberg, 1995.

Dans cette partie, on va voir comment des mathématiques sont utilisés pour modéliser le code, la détection l'erreur et la correction l'erreur. La classe de code considérée sera le code cyclique, i.e si $(a_0, ..., a_{n-1})$ est un bon message, alors après décalage vers la droite, le nouveau message $(a_{n-1}, a_0, ..., a_{n-2})$ se trouve encore dans l'ensemble de bon messages. On résume la façon de modéliser le problème :

1. Un message $m = (a_0, ..., a_{n-1})$ est modélisé en un polynôme $m(x) = a_0 + a_1x + ... + a_{n-1}x^{n-1}$ de $\mathbb{K}[x]/(x^n - 1)$.
   Ainsi, quand on veut faire un décalage vers le droite le message $m = (a_0, ..., a_{n-1})$ en $m' = (a_{n-1}, a_0, ..., a_{n-2})$, il suffit d'effectuer une multiplication $m(x)$ par x avec une remarque que $x^n = 1$ dans $\mathbb{K}[x]/(x^n - 1)$, ce qui donne : $xm(x) = a_{n-1} + a_0x + ... + a_{n-2}x^{n-1}$, i.e $xm(x)$ est bien $m'(x)$.

2. La détection est modélisée en une application h de sorte que si $m(x)$ est une représentation d'un bon message, alors $h(m(x)) = 0$. En revanche, si $h(m(x)) \neq 0$ alors $m(x)$ est un message erroné.
3. La correction consiste à déterminer les positions des erreurs et la valeur de l'erreur en ces positions. Alors si l'erreur est modélisée en un polynôme $e(x) = e_0 + e_1x + ... + e_{n-1}x^{n-1}$, il faut d'abord déterminer les i tels que $e_i \neq 0$ et puis déterminer la valeur de e_i.

   — Pour modéliser les positions des erreurs, on aura les *localisateur de la position i :* X_i.

   — Pour déterminer les positions des erreurs, on aura un *polynôme localisateur de l'erreur* de sorte que la relation entre ses racines et les X_i va proposer l'information sur les positions des erreurs.

   Ainsi, le décodage consiste à recevoir le message $y(x)$, vérifier s'il y a des erreurs grâce à l'application h, déterminer l'erreur $e(x)$ (s'il y en a), retourner le message corrigé $m(x) = y(x) - e(x)$.

Dans ce qui suit, ce seront des définitions et des résultats mathématiques nécessaires pour modéliser le problème.

3.2.1 Codes cycliques

Définition 1. *Un code linéaire C de longueur n sur $\mathbb{K}$ et de dimension k est un sous-espace vectoriel de dimension k de $\mathbb{K}^n$. On dira que C est un code(n,k). L'élément de C s'appelle le mot.*

Définition 2. *Un code C est dit cyclique si :*
— *C est un code linéaire.*
— *Si $(x_1, ..., x_n) \in C$, alors $(x_n, x_1, ..., x_{n-1}) \in C$.*

Définition 3. *Soit $\mathbb{K}$ un corps fini et n un entier, $n \geq 1$.*

— *On appelle représentation polynomiale de $\mathbb{K}^n$, l'application θ de $\mathbb{K}^n$ dans $\mathbb{K}[x]/(x^n-1)$ définie par θ : $(a_0,...,a_{n-1}) \mapsto a_0 + a_1x + ... + a_{n-1}x^{n-1}$.*
— *Si $a = (a_0,...,a_{n-1})$, alors le polynôme $a(x) = a_0 + a_1x + ... + a_{n-1}x^{n-1}$ s'appelle la représentation polynomiale de a.*

La deuxième condition dans la définition du code cyclique dit que C est invariant sous le décalage vers la droite. Or un décalage vers la droite dans C est équivalent à une multiplication par x dans $\theta(C)$, $\theta(C)$ est donc invariant sous la multiplication par x, i.e $\theta(C) = x\theta(C)$. Réapplique cet argument, on trouve que $\theta(C) = x\theta(C) = x^2\theta(C) = x^3\theta(C), ...$ Comme C possède d'une structure d'espace vectoriel, $\theta(C)$ l'est aussi. On conclut donc $\theta(C) = (b_0 + b_1x + b_2x^2 + ...)\theta(C)$, autrement dit $\theta(C)$ possède d'une structure d'idéal de $\mathbb{K}[x]/(x^n-1)$.

Théorème 1. *Soit C un code linéaire sur $\mathbb{K}$. Le code C est cyclique si et seulement si $\theta(C)$ est un idéal de $\mathbb{K}[x]/(x^n-1)$.*

Théorème 2. *Soit $\mathbb{K}$ un corps, n non nul, alors tout idéal de $\mathbb{K}[x]/(x^n-1)$ est un idéal principal (i.e un ensemble qui est formé de tous les multiples d'un même élément).*

Théorème 3. *La représentation polynomiale dans $\mathbb{K}[x]/(x^n-1)$ d'un code cyclique est formé de tous les multiples d'un même polynôme.*

Définition 4. *Un tel polynôme s'appelle un générateur du code cyclique.*

Théorème 4. *Chaque code cyclique de longueur n sur $\mathbb{K}$ possède un générateur et un seul qui est un diviseur de x^n-1 dans $\mathbb{K}[x]$, et donc le coefficient dominant es 1.*

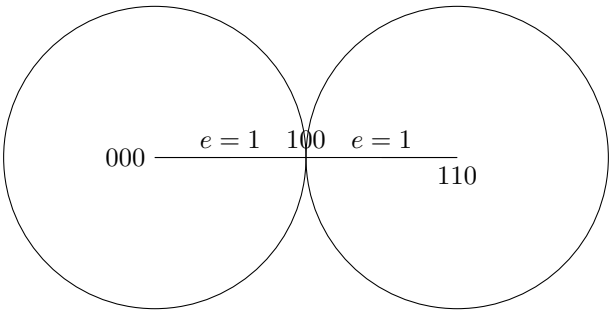
Définition 5. *Un tel générateur s'appelle un générateur de code cyclique C.*

Conclusion : *La représentation polynomiale dans $\mathbb{K}[x]/(x^n-1)$ d'un code cyclique est un multiple du générateur.*

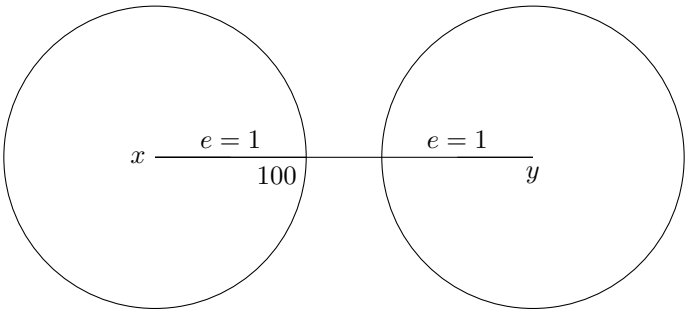
Pour pouvoir corriger e erreurs dans un mot reçu, il nous faut un code ayant la capacité de corriger au plus e erreurs. Dans la partie suivant, on va présenter une telle classe de code particulier de la classe de code cyclique : le code B.C.H (Bose, Chaudhuri, Hocquenghem). C'est dans le décodage de ce code, on trouve l'application de l'algorithme de Berlekamp-Massey.

3.2.2 Code B.C.H

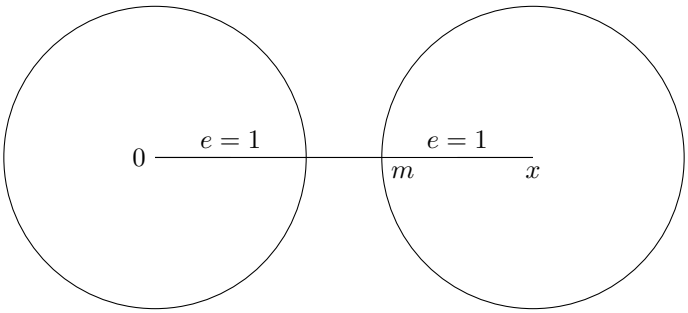
Avant de continuer à introduire les définitions mathématiques, on cherche à comprendre ce qui sont nécessaires pour pouvoir construire un code ayant la capacité de corriger au plus e erreurs.
On rappelle que dans *Problème 2* étudié, 100 se trouve à la fois dans l'ensemble dans mots erroné à partir de 000 et celui de 110. D'autre façon de concevoir le problème,



Si on représente l'ensemble des mots différents de 000 en 1 erreur par un cercle $C(000, 1)$ de centre 000 et de rayon $e = 1$ comme la figure ci-dessus, on comprend que 100 se trouve dans l'intersection de $C(000, 1)$ et $C(110, 1)$, ce qui ramène au problème de ne pas pouvoir corriger le mot reçu. Donc, on s'attend à un code comme suivant :



i.e, $C(x, e)$ et $C(y, e)$ sont disjoints où $x = 000$, $y = 110$, $e = 1$. Par conséquent, 100 se trouve soit dans $C(x, e)$ soit dans $C(y, e)$. Autrement dit, on veut que $d(x, y) \geq 2e + 1$, où $d(x, y)$ est un entier représentant la distance entre x et y. Ainsi, si on veut qu'un code C ait la capacité de corriger e erreurs, il faut que $d(x, y) \geq 2e + 1$ pout tous $(x, y) \in C \times C$.
Si la distance est invariante sous la translation, i.e $d(x, y) = d(x-y, 0)$, alors pour que $d(x, y) \geq 2e+1$ pout tous $(x, y) \in C \times C$, il suffit que $d(x, 0) \geq 2e+1, \forall x \in C$



Dans ce qui suit, on trouvera des notions mathématiques qui généralisent $d(x, y)$ et $d(x, 0)$, ce sont respectivement *la distance de Hamming* et *le poids de Hamming*.

Définition 6. *Soient $x = (x_1, x_2, ..., x_n)$ et $y = (y_1, y_2, ..., y_n)$ deux éléments de $\mathbb{K}^n$. La distance de Hamming entre x et y, noté $d(x, y)$ est défini par : $d(x, y) = card\{i \in \{1, 2, ..., n\} | x_i \neq y_i\}$.*

Définition 7. *La distance minimale d'un code C est la plus petite des distances non nulles entre les mots de C, i.e $d(C) = min\{d(x, y) | (x, y) \in C \times C, x \neq y\}$.*

Définition 8. *Le poids d'un élément $x = (x_1, x_2, ..., x_n)$ est le nombre de ses composantes non nulles. On le note $w(x)$.*

Remarque : $w(x) = d(x, 0)$.

Définition 9. *On appelle le poids du code linéaire C le poids minimum des mots non-nuls de C.*

Remarque : Un code linéaire ayant la capacité de corriger e erreurs si son poids est au moins $2e + 1$.
Donc, si δ est un nombre entier au moins égal à $2e+1$, et si on peut construire un code dont le poids est au moins égal à δ, alors ce code aura la capacité de corriger e erreurs. Pour construire un tel code, il suffit de construire son générateur $g(x)$ de sorte que dans le corps de décomposition de $x^n - 1$, $g(x)$ ait $(\delta - 1)$ racines de forme $\beta^r, \beta^{r+1}, ..., \beta^{r+\delta-2}$ comme dit dans le théorème suivant,

Théorème 5. *Soient :*

i) n, q sont deux entier tels que $pgcd(n, q) = 1$.

ii) C un code cyclique sur $\mathbb{F}_q$, de longueur n et de générateur $g(x)$.

iii) $\mathbb{L}$ le corps de décomposition de $x^n - 1$ sur $\mathbb{F}_q$. β est une racine primitive n^{ime} de l'unité dans $\mathbb{L}$.

iv) δ, r sont des entiers tels que $\delta \geq 2$ et $r \geq 1$.

Si $g(x)$ a $(\delta - 1)$ racines de forme $\beta^r, \beta^{r+1}, ..., \beta^{r+\delta-2}$, alors son poids est au moins égal à δ.

Remarque : Un tel candidat $g(x)$ est le produit des polynômes minimaux de $\beta^r, \beta^{r+1}, ..., \beta^{r+\delta-2}$. Ce candidat ramène à une classe de code comme suivant,

Définition 10. *Un code B.C.H de distance δ est un code dont générateur est le produit des polynômes minimaux de $\beta^r, \beta^{r+1}, ..., \beta^{r+\delta-2}$ pour $r \geq 1$.*

Ainsi, jusqu'à maintenant on a réussit à construire un code pouvant corriger e erreurs. La partie suivant sera pour la détection de l'erreur et la correction de l'erreur sur ce code.

3.2.3 Décodage du code B.C.H

1. Détection de l'erreur.
   Comme expliqué avant, dans cette partie, on s'attend qu'une application h de sorte que si $a(x)$ est une représentation d'un bon mot, alors $h(a(x)) = 0$. En revanche, si $h(a(x)) \neq 0$ alors $a(x)$ est un mot erroné.

On rappelle aussi que la représentation polynomiale dans $\mathbb{K}[x]/(x^n-1)$ d'un code cyclique C est un multiple du générateur. En plus, si C un code B.C.H(n,k) sur $\mathbb{F}_q$, où $pgcd(n,q)=1$, alors les $\beta^r, \beta^{r+1}, ..., \beta^{r+\delta-2}$ sont des racines de $g(x)$. On conclut donc $\beta^r, \beta^{r+1}, ..., \beta^{r+\delta-2}$ sont aussi des racines du bon mot $a(x)$, i.e,

$$a(\beta^{r+i}) = 0 \ \forall i \text{ tel que } 0 \leq i \leq (\delta-2)$$

Ou,

$$a_0 + a_1\beta^{r+i} + ... + a_{n-1}\beta^{(n-1)(r+i)} = 0, \forall i \text{ tel que } 0 \leq i \leq (\delta-2)$$

Écrire sous forme matricielle, il est donc,

$$\begin{pmatrix} 1 & \beta^r & \dots & \beta^{(n-1)r} \\ 1 & \beta^{r+1} & \dots & \beta^{(n-1)(r+1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \beta^{r+\delta-2} & \dots & \beta^{(n-1)(r+\delta-2)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Note H la matrice ci-dessus.

Conclusion : $a(x) \in C$ si et seulement si $H \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$.

2. Correction de l'erreur.
On rappelle que la correction consiste à déterminer les positions des erreurs et la valeur de l'erreur en ces positions.
On a réussit à déterminé l'application H qui permet de détecter s'il y a l'erreur dans le mot reçu. Si la valeur H appliqué sur le mot reçu est égale à 0, on sait que c'est un bon mot, sinon on obtient quand même d'autre information sur l'erreur.

Définition 11. *Soit e capacité de correction de C. Soit $y(x) \in \mathbb{F}_q[x]/(x^n-1)$, on dira que "$y(x)$ est un mot reçu dont erreur est $e(x)$" si $w(e(x)) \leq e$ s'il existe $a(x) \in C$ tel que $y(x) = a(x) + e(x)$.*

Comme $a(x) \in C$, $a(\beta^{r+i}) = 0$, on a donc $y(\beta^{r+i}) = e(\beta^{r+i})$ $\forall i$ tel que $0 \leq i \leq (\delta-2)$.

Autrement dit, $H \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} = H \begin{pmatrix} e_0 \\ e_1 \\ \vdots \\ e_{n-1} \end{pmatrix}$.

Comme H et y sont connus, on peut penser à retrouver e en résolvant un système linéaire. Pourtant comme il n' y a pas toujours l'erreur sur toutes les positions, il existe donc des $e_i = 0$. Donc, pour construire un système linéaire dont les inconnus sont des $e_i \neq 0$, il faut d'abord donc déterminer les

positions des erreurs, i.e les i tels que $e_i \neq 0$.

Définition 12. *i) Les $X_i = \beta^i$ sont appellés les localisateur de la position i avec $0 \leq i \leq n-1$.*

ii) On note I l'ensemble des positions de l'erreur, i.e,

$$I = \{i \text{ tel que } 0 \leq i \leq n-1, e_i \neq 0\}$$

Définition 13. *Soit I l'ensemble des positions d'erreur. On appelle polynôme localisateur de l'erreur, le polynôme $\sigma(x)$ est définie par :*

$$\sigma(x) = \prod_{i \in I}(1 - X_i x).$$

Remarque :

i) $deg\sigma(x) = |I|$

ii) Si β^{-i} est une racine de $\sigma(x)$, alors $e_i \neq 0$. Donc, si on connaît les racines de $\sigma(x)$, on connaît I. Pour cela, il faut d'abord déterminer $\sigma(x)$ et puis résoudre $\sigma(x) = 0$.

On note $\sigma(x) = \sigma_m x^m + \sigma_{m-1}x^{m-1} + ... + \sigma_1 x + 1$ avec $m = |I|$.
Il faut déterminer les coefficients σ_i.

Quand on reçoit le mot $y = (y_0, ..., y_{n-1})$, on peut calculer Hy^t où y^t est la transposée de y. On note $S = (S_0, ..., S_{\delta-2})$ le vecteur tel que $S^t = Hy^t$. S est appelé *le syndrome de y*. La relation suivant permet de déterminer les σ_i à partir des S_j.

Théorème 6. $\sigma_m S_j + \sigma_{m-1}S_{j+1} + ... + \sigma_1 S_{j+m-1} = -S_{j+m+1}$ *,$\forall j$ tel que* $0 \leq j \leq (\delta - m - 2)$

Remarque : Les σ_i définissent une relation de récurrence pour la suite (S_j). C'est ici qu'intervient l'application de l'algorithme de Berlekamp-Massey.
Après avoir déterminé le polynôme $\sigma(x)$ et ses racines β^{n-i}, on connaît I. Le polynôme erreur maintenant peut être écrite sous forme :

$$e(x) = \sum_{i \in I} e_i x^i.$$

Or, $e(\beta^j) = S_j$ avec $r \leq j \leq r + \delta - 2$.
Donc, $\sum_{i \in I} e_i((\beta^j)^i) = S_j$ avec $r \leq j \leq r + \delta - 2$.
On peut réécrire sous forme : $\sum_{i \in I} e_i((\beta^i)^j) = S_j$ avec $r \leq j \leq r + \delta - 2$.

On a donc :

$$\begin{pmatrix} (\beta^{i_1})^r & (\beta^{i_1})^r & \cdots & (\beta^{i_1})^r \\ (\beta^{i_1})^{r+1} & (\beta^{i_1})^{r+1} & \cdots & (\beta^{i_1})^{r+1} \\ \vdots & \vdots & \ddots & \vdots \\ (\beta^{i_1})^{r+\delta-2} & (\beta^{i_1})^{r+\delta-2} & \cdots & (\beta^{i_1})^{r+\delta-2} \end{pmatrix} \begin{pmatrix} e_{i_1} \\ e_{i_2} \\ \vdots \\ e_{i_m} \end{pmatrix} = \begin{pmatrix} S_r \\ S_{r+1} \\ \vdots \\ S_{r+\delta-2} \end{pmatrix}$$

On note :

— E le vecteur dont les composantes sont des e_i avec $i \in I$.
— s le vecteur dont les composantes sont des S_j avec $r \leq j \leq r + \delta - 2$.
— M la matrice dont les colonnes sont $\begin{pmatrix} (\beta^i)^r \\ (\beta^i)^{r+1} \\ \vdots \\ (\beta^i)^{r+\delta-2} \end{pmatrix}$ avec $i \in I$.

Alors les valeur de l'erreur à la i^{ime} position e_i peuvent être trouvées à partir du système linéaire : $ME = s$.

4 Application de l'algorithme Berlekamp-Massey dans le code correcteur d'erreur.

Soit
— C un code B.C.H de distance $(q-1)$ sur $\mathbb{F}_q$.
— β est le générateur de $\mathbb{F}_q^*$.
— le générateur de C, $g(x) = \prod_{j=0}^{l-1}(x - \beta^j)$ avec $l \leq p - 1$.
On envoie $c(x) = g(x)(m_0 + m_1 x + ...)$

Le décodage du message reçu de $c(x)$ est suivant :

1. On reçoit le message $y(x)$. Si $Hy = 0$, le message est $y(x)$, sinon on passe à l'étape suivant.
2. Construire la suite $s_j = y(\beta^j)$ avec $0 \leq j \leq l - 1$.
3. Applique **AlgoB-M** sur la suite s_j pour déterminer les coefficients de la relation récurrente. On en déduit les coefficients pour le polynôme localisateur de l'erreur $\sigma(x)$.
4. Chercher les racines de $\sigma(x) = 0$.
5. On en déduit l'ensemble I et la forme de l'erreur $e(x)$.
6. Résoudre le système $ME = s$. Déterminer $e(x)$.
7. Le message est corrigé : $c(x) = y(x) - e(x)$.

Exemple
Le corps : $\mathbb{F}_5$ dont $\beta = 2$, $\beta^0 = 1, \beta^1 = 2, \beta^2 = 4, \beta^3 = 3$.

2. A la réception, après avoir calculé le syndrome, on obitient :

$$\begin{aligned} s_0 &= y(\beta^0) = 3 \\ s_1 &= y(\beta^1) = 0 \\ s_2 &= y(\beta^2) = 4 \\ s_3 &= y(\beta^3) = 2 \end{aligned}$$

3. **AlgoB-M** donne $s_{i+2} + 2s_{i+1} + 2s_0 = 0$, i.e $2s_0 + 2s_{i+1} = s_{i+2}$.
Donc, $\sigma(x) = 2x^2 + 2x + 1$.
4. Les racines sont 1 et 3, ou sont β^0 et $\beta^3 = \beta^{-1}$

5. I est donc $\{0, 1\}$ et $e(x) = e_0 + e_1.x$
6. Resoudre

$$\begin{pmatrix} \beta^0 & \beta^1 \\ (\beta^0)^2 & (\beta^1)^2 \end{pmatrix} \begin{pmatrix} e_0 \\ e_1 \end{pmatrix} = \begin{pmatrix} s_1 \\ s_2 \end{pmatrix}$$

Donc,

$$\begin{pmatrix} 1 & 2 \\ 1 & 4 \end{pmatrix} \begin{pmatrix} e_0 \\ e_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \end{pmatrix}$$

Donc,

$$\begin{pmatrix} e_0 \\ e_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

$e(x) = 1 + 2x$