Informatique

TP 05

Classes et codes correcteurs d'erreurs

Sauvegardez sur clé : dossier Info, dossier TP5, fichier TP05.py.

Motivation

Lors de la transmission d'un message à travers un canal de communication, des erreurs peuvent se produire et modifier de petits fragments du message (un grain de poussière sur un CD les provoque déjà). On s'intéresse dans ce TP aux **codes correcteurs d'erreurs** qui permettent de lutter contre ce problème.

On imagine qu'on souhaite transmettre un bloc de n bits. Chaque bit a une probabilité $p \ll 1$ d'être inversé. On essaie de compléter ce bloc en rendant l'information redondante, de sorte qu'on puisse détecter et même corriger les erreurs à condition qu'il s'en produise suffisamment peu. On s'intéresse ici uniquement aux codes **linéaires** :

- le message initial est vu comme une matrice colonne à coefficients dans $\{0,1\}$;
- le message redondant sera un message de m bits obtenus en multipliant le message initial à gauche par une matrice à n lignes et m colonnes à coefficients dans $\{0,1\}$;
- les produits et sommes sont pris modulo 2, on peut le voir comme des opérations à valeurs dans $\mathbb{Z}/2\mathbb{Z}$.

On verra plus en détail le principe du codage et du décodage dans les exercices 4, 5 et 6. En attendant, on a une motivation pour coder $\mathbb{Z}/2\mathbb{Z}$ et des matrices.

Exercice 1. *On commence doucement*

Écrire une classe `F2` implémentant $\mathbb{Z}/2\mathbb{Z}$.

On écrira les méthodes `__init__()`, `__repr__()`, `__add__()`et `__mul__()`.
On pourra écrire d'autres méthodes mais seulement après avoir un peu avancé dans le TP.

Exercice 2. *Un peu plus costaud*

Écrire une classe `Matrice` implémentant les matrices rectangulaires quelconques. Les coefficients seront des objets de n'importe quelle classe qui supporte les opérations algébriques. Cette classe devra :

1. gérer l'initialisation de la matrice grâce à la méthode `__init__()` ;
2. afficher la matrice avec la méthode `__repr__()`
   un affichage rudimentaire pourra faire l'affaire dans un premier temps ;
3. pouvoir tester l'égalité de deux matrices avec les méthodes `__eq__()` et `__ne__()` ;
4. effectuer les somme/produit de deux matrices avec les méthodes `__add__()` et `__mul__()`
   en cas d'incompatibilité de formats, elles provoqueront une erreur.

Exercice 3. *Séquençage d'un message*

Les "messages" que les codes correcteurs d'erreurs permettent de coder sont des matrices de $\mathcal{M}_{n,1}(\mathbb{Z}/2\mathbb{Z})$. Premier objectif : convertir un message clair en suite de blocs de n bits, que l'on verra comme des matrices de $\mathcal{M}_{n,1}(\mathbb{Z}/2\mathbb{Z})$.

Écrire une fonction `decoupage(texte,n)` qui prend comme argument un message `texte` sous forme d'une chaîne de caractères et un entier `n` ; convertit le message en binaire, et le découpe en blocs de n bits sous forme d'une liste de matrices de $\mathcal{M}_{n,1}(\mathbb{Z}/2\mathbb{Z})$. On complètera la dernière matrice avec des 0 si le message n'a pas pour nombre de bits un multiple de n.

Écrire également la fonction `reconstitution(liste,m)` qui reforme un message en clair à partir d'une liste de matrices de $\mathcal{M}_{m,1}(\mathbb{Z}/2\mathbb{Z})$.

$\star\star\star$

Dans la suite, on s'intéresse comme promis aux **codes linéaires** et plus précisément aux **codes sytématiques**.

Un code systématique est une application de la forme $\Phi : \begin{cases} \left(\mathbb{Z}/2\mathbb{Z}\right)^k \longrightarrow & \left(\mathbb{Z}/2\mathbb{Z}\right)^n \\ \begin{pmatrix} a_1 \\ \vdots \\ a_k \end{pmatrix} \longmapsto & \begin{pmatrix} I_k \\ G \end{pmatrix} \times \begin{pmatrix} a_1 \\ \vdots \\ a_k \end{pmatrix} \end{cases}$.

Autrement dit, pour coder un mot (suite de k bits), on recopie le mot en question et on ajoute un certain nombre de bits à la fin, obtenus en multipliant le mot de départ vu comme un vecteur de $\left(\mathbb{Z}/2\mathbb{Z}\right)^k$ par une matrice $G \in \mathcal{M}_{n-k,k}\left(\mathbb{Z}/2\mathbb{Z}\right)$.
On appelera **mot du code** un mot codé c'est-à-dire un élément de l'image de Φ.

On s'intéressera en priorité au **code de Hamming (4,7)** dont la matrice G est : $G = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$.

Autrement dit, un mot $a_1a_2a_3a_4$ est codé par $a_1a_2a_3a_4b_1b_2b_3$ avec $b_1 = a_1+a_2+a_3$, $b_2 = a_1+a_2+a_4$, $b_3 = a_1+a_3+a_4$ (l'addition étant prise modulo 2).

Exercice 4. *Algorithme de décodage naïf pour le code de Hamming.*

---

1. Écrire une fonction `CodageHamming(m)` qui prend comme argument un mot `m`, *i. e.* une matrice de $\mathcal{M}_{4,1}(\mathbb{Z}/2\mathbb{Z})$, et retourne le mot codé par le code de Hamming (4,7) correspondant.

On souhaite maintenant décoder un mot codé après transmission de celui-ci. S'il n'y a pas eu d'erreur, rien de plus simple. Il suffit de tronquer le mot en ne conservant que les 4 premiers caratères. Comment savoir s'il y a eu des erreurs ? Si le mot reçu après transmission n'est pas un mot du code, il est évident qu'il y a eu une ou plusieurs erreurs. Si le mot reçu après transmission est un mot du code, il se pourrait qu'il y ait toutefois eu des erreurs[1] *ayant transformé le mot codé en un autre mot du code ; néanmoins, comme la probabilité d'une erreur est faible, on fera dans ce cas le pari que ce n'est pas ce qui s'est produit, et on considèrera qu'il n'y a pas eu d'erreur.*

2. Écrire une fonction `MotHamming(M)` qui prend comme argument un mot transmis `M`, *i. e.* une matrice de $\mathcal{M}_{7,1}(\mathbb{Z}/2\mathbb{Z})$, et retourne le booléen `true` si `M` est un mot du code et `false` sinon.

Dans la question suivante on implémente l'algorithme de décodage naïf. On procède comme suit : si le mot reçu est un mot du code, il suffit de le tronquer. Sinon, on cherche le mot du code qui présente le moins de bits de différence avec le mot reçu, on fait le pari (car c'est le cas le plus probable) que ce mot a été le mot envoyé et on le tronque.

3. Écrire une fonction `DecodageHamming(M)` qui prend comme argument un mot reçu `M`, *i. e.* une matrice de $\mathcal{M}_{7,1}(\mathbb{Z}/2\mathbb{Z})$, et retourne le mot décodé par le code de Hamming (4,7) avec l'algorithme naïf.

Dans la dernière question on va enfin jouer avec un code correcteur.

4. Écrire une fonction `JouerAvecHamming(p)` qui prend comme argument un flottant `p` qui sera la probabilité qu'un bit soit inversé pendant la transmission (et devra donc être strictement positif mais très petit). Cette fonction ne retournera rien mais aura les effets de bords suivants :
   - faire saisir à l'utilisateur un message en clair ;
   - convertir ce message en liste de matrices de $\mathbb{Z}/2\mathbb{Z}$ à l'aide de la fonction `decoupage(texte,4)` ;
   - simuler une transmission avec erreurs en modifiant chaque coefficient de chaque matrice de la liste avec probabilité `p` ;
   - décoder le message obtenu de deux façons différentes : une première fois en tronquant tout simplement chaque bloc de 7 bits après le $4^{\text{ième}}$ bit ; une seconde fois avec la fonction `DecodageHamming(M)` ; ces messages étant traduits en clair à l'aide de la fonction `reconstitution(liste,4)` ;
   - afficher les deux messages décodés correspondants.

   On pourra écrire une interface graphique ou se contenter des fonctions `input` et `print`.

1. Au moins 3 dans le cas du code de Hamming (4,7).

Exercice 5. *Analyse de l'algorithme naïf. Syndrômes.*

1. En général, pour un code linéaire systématique (k,n), combien de comparaisons de bits devra réaliser l'algorithme de décodage naïf présenté sur l'exemple du code de Hammiung (4,7) dans l'exercice précédent ?

Il est clair que, pour k grand, cet algorithme n'est pas praticable !

Dans la suite, on présente un algorithme plus raisonnable[2]. L'idée-clé est d'introduire la matrice $H = \begin{pmatrix} G & I_{n-k} \end{pmatrix}$, appelée **matrice de contrôle** du code. Celle-ci présente la particularité suivante (on ne demande pas de le montrer) : un mot M est un mot du code si et seulement si $H \times M = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$. On appelle **syndrôme** le mot $S = H \times M$.

- On vient de voir que S est nul si et seulement si M est un mot de code ; l'explication la plus probable est qu'il n'y a pas eu d'erreur ; dans ce cas on décodera donc M en $\Phi^{-1}(M - e_i)$.
- Mais supposons maintenant qu'on ait S non nul, mais $S = C_i$, où C_i est la $i^{\text{ième}}$ colonne de H ; cela signifie (on ne demande pas de le montrer) que le plus probable est qu'une unique erreur se soit produite sur le $i^{\text{ième}}$ bit ; dans ce cas on décodera donc M en $\Phi^{-1}(M - e_i)$.
- Supposons maintenant qu'on ait S non nul, distinct de toutes les colonnes de H, mais $S = C_i + C_j$, la somme des colonnes i et j de H ; cela signifie (on ne demande pas de le montrer) que le plus probable est que deux erreurs se soit produites sur les $i^{\text{ième}}$ et $j^{\text{ième}}$ bits ; dans ce cas on décodera donc M en $\Phi^{-1}(M - e_i - e_j)$.
- Etc.

5. Expliquer pourquoi, dans le cas du code de Hamming (4,7), on peut s'arrêter au second cas de la liste précédente, *i. e.* si le syndrôme n'est pas nul c'est nécessairement une colonne de H. Qu'en déduire sur le code de Hamming (4,7) ?
6. Écrire une fonction `MotHamming2(M)` qui fait la même chose que `MotHamming(M)` mais en utilisant la matrice de contrôle et le syndrôme (donc est beaucoup plus rapide).
7. Écrire une fonction `DecodageHamming2(M)` qui fait la même chose que `DecodageHamming(M)` mais en utilisant la matrice de contrôle et le syndrôme (donc est beaucoup plus rapide). Utiliser la remarque de la question 2.
8. Écrire une fonction `JouerAvecHamming2(p)` qui fait la même chose que `JouerAvecHamming(p)` mais en utilisant la matrice de contrôle et le syndrôme. Comparer les performances en termes de rapidité avec `JouerAvecHamming2(p)`.

Exercice 6. *Si vous avez fait tout le reste.*

Écrire une fonction `Jouer(k,n,G,p)` qui prend comme argument un entier k, un entier $n > k$, une matrice $G \in \mathcal{M}_{n-k,k}(\mathbb{Z}/2\mathbb{Z})$ et un flottant p et fait la même chose que `JouerAvecHamming(p)` mais pour le code systématique correspondant à la matrice G. Cette fonction devra aussi afficher le nombre d_Φ, défini comme étant plus petit nombre de 1 d'un mot du code non nul, ainsi que le nombre e_d d'erreurs maximal qu'on est toujours certain de détecter, et le nombre e_c d'erreurs maximal qu'on est certain de pouvoir corriger pour ce code[3]

2. Qu'on pourrait encore largement optimiser, mais on se contentera de la version présentée ici.
3. On pourra admettre qu'on a $e_d = d_\Phi - 1$ et $e_c = \lfloor \frac{d_\Phi - 1}{2} \rfloor$.