L'objectif de ce projet est d'analyser de petits codes correcteurs d'erreurs linéaires. Un code correcteur d'erreur binaire est une transformation C qui a tout vecteur de bits de taille k fait correspondre un vecteur de bits de taille $n > k$. Les bits rajoutés à ce vecteur permettent de corriger d'éventuelles erreurs lors de la transmission de ce vecteur par exemple par une onde wifi.

Par exemple, un code correcteur d'erreur binaire simple peut se contenter de doubler le vecteur. Ainsi s'il doit transmettre 00111 il transmettra 0011100111. Dans ce cas, il est possible de détecter une éventuelle erreur, mais pas de la corriger. S'il est triplé, alors une simple erreur pourra être corrigée.

Pour des raisons pratiques, il n'est utilisé quasiment que des codes linéaires, ce qui veut dire que C est une fonction linéaire (merci de noter que puisque l'espace utilisé est celui des bits, la loi + correspond au "XOR").

Question 1)

Dans un premier temps, on va considérer que l'on possède une matrice M de taille k par n qui représente un code C. Cette matrice ne contient donc que des "1" et des "0". Que se passe-t-il si la matrice utilisée n'est pas de rang k ?

Question 2)

Pour connaître la capacité de correction d'un tel code, on cherche à connaître le nombre minimum de bits qui différent d'un vecteur à l'autre. C'est ce qu'on appelle la distance minimale:

$$d_{\min} = \min_{v_1, v_2} \sum_{i=1}^{n} C(v_1)_i \neq C(V_2)_i$$

Où $C(V_1)_i \neq C(V_2)_i = 1$ si et seulement si $C(V_1)_i$ et $C(V_2)_i$ sont différents. En effet, si un vecteur subit des erreurs, et que le nombre d'erreurs qu'il a reçues est inférieur à $\lfloor \frac{d_{\min}}{2} \rfloor$, alors il est possible de retrouver le bon mot sans hésitation, celui-ci étant par définition le plus proche en terme de nombre de différences avec le vecteur obtenu.

Montrer que $d_{\min}$ peut être obtenu d'une façon différente, du fait que le code est linéaire:

$$d_{\min} = \min_{v} p(C(v))$$

Où la fonction p est la fonction qui compte le nombre de "1" dans les vecteurs.

Question 3)

Proposer un algorithme qui calcule la distance minimale d'un code donné par sa matrice (en testant tous les vecteurs possibles). Jusqu'à quelles tailles pour n et k est-ce raisonable ?

Question 4)

Proposer un algorithme qui permet, à partir d'un vecteur reçu contenant éventuellement des erreurs, de retrouver le/un mot correspondant le plus proche

dans l'image du code. Dans le cas d'un code avec une grande distance minimale, on fera de nombreux tests pour vérifier que tout est correct.

Question 5)

En générant tous les codes possibles, trouver des codes aux meilleures distances minimales pour des valeurs petites de n et k. Tracer en même temps les courbes de temps pris pour ces recherches.

Question 6)

Ouverture: s'intéresser à des constructions plus riches de codes (LDPC par exemple), regarder quels canaux de transmissions sont possibles et l'impact sur les performances, rechercher des méthodes de décodage plus évoluées...