Références : Demazure, G. Zémor, wikipedia (pour les codes de Hamming binaires non repris ici).

1 TP

On appellera symbole d'information l'unité de base transmise, qu'on supposera appartenir à un corps fini K, par exemple pour un bit un élément de $K = \mathbb{Z}/2\mathbb{Z}$, ou pour un octet un élément du corps à 256 éléments $K = F_{256}$.

On veut coder un message de longueur k avec des possibilités de détection et de correction d'erreurs, pour cela on rajoute des symboles calculés à partir des précédents, on envoie un élément d'un code ayant n symboles.

1.1 Le bit de parité.

On prend $k = 7$ bits et $n = 8$ bits. On compte le nombre de 1 parmi les 7 bits envoyés, si ce nombre est pair, on envoie 0 comme 8ième bit, sinon 1. Au final le nombre de bits à 1 de l'octet (1 octet=8 bits) est pair. On peut ainsi détecter une erreur de transmission si à la réception le nombre de bits d'un octet est impair, mais on ne peut pas corriger d'erreurs. On peut aussi dire que l'octet représente un polynôme à coefficients dans $\mathbb{Z}/2\mathbb{Z}$ divisible par $X + 1$.

Exercice : Écrire un programme Xcas permettant de rajouter un bit de parité à une liste composée de 7 bits. Puis un programme de vérification qui accepte ou non un octet selon sa parité. Vous représenterez l'octet par une liste de bits, avec le délimiteur `poly1[` pour pouvoir effectuer des opérations arithmétiques polynomiales, et vous effectuerez la vérification de deux manières, en comptant le nombre de 1 ou avec l'instruction `rem`.

1.2 Codes linéaires

Définition : On multiplie le vecteur des k symboles par une matrice M à coefficients dans K de taille $n \times k$ et on transmet l'image. Pour assurer qu'on peut identifier un antécédent unique à partir d'une image, il faut que M corresponde à une application linéaire injective, ce qui entraine $n \geq k$. On dit qu'un vecteur de n symboles est un mot du code s'il est dans l'image de l'application linéaire.

Pour assurer l'injectivité tout en facilitant le décodage, on utilise souvent une matrice identité k, k comme sous-bloc de la matrice n, k, par exemple on prend l'identité pour les k premières lignes de M, on ajoute ensuite $n - k$ lignes.

Pour savoir si un vecteur est un mot de code, il faut vérifier qu'il est dans l'image de M. On peut par exemple vérifier qu'en ajoutant la colonne de ses coordonnées à M, on ne change pas le rang de M (qui doit être k).

Exercice : créez une matrice M de taille 7,4 injective. Puis un programme qui teste si un vecteur est un mot de code et en extrait alors la partie avant codage. Vérifiez votre programme avec un vecteur Mv, on doit obtenir un mot de code.
Instructions utiles : `idn` (matrice identité) `ker` (noyau d'une application linéaire), `rank` (rang), `tran` (tranposée), ... Pour créer une matrice, on peut coller les lignes de 2 matrices A et B par `[op(A),op(B)]` ou avec `blockmatrix`.

1.3 Codes polynomiaux

Définition : Il s'agit d'un cas particulier de codes linéaires. On se donne un polynôme $g(x)$ de degré $n-k$, On représente le message de longueur k à coder par un polynôme P de degré $k-1$. On multiplie P par x^{n-k}, on calcule le reste R de la division de Px^{n-k} par g. On émet alors $Px^{n-k} - R$ qui est divisible par g. Les mots de code sont les polynômes divisibles par g.

Exercice : écrire de cette façon le codage du bit de parité. Puis une procédure Xcas de codage utilisant $g = X^7 + X^3 + 1$ (ce polynôme était utilisé par le Minitel). N.B. on obtient le polynôme X^{n-k} sous forme de polynome-liste dans Xcas par `poly1[1,0$(n-k)]`.

1.4 Détection et correction d'erreur

Si le mot recu n'est pas dans l'image de l'application linéaire il y a eu erreur de transmission. Sinon, il n'y a pas eu d'erreur *détectable* (il pourrait y avoir eu plusieurs erreurs qui se "compensent").

Plutôt que de demander la réémission du mot mal transmis (ce qui serait par exemple impossible en temps réel depuis un robot en orbite autour de Mars), on essaie d'ajouter suffisamment d'information pour pouvoir corriger des erreurs en supposant que leur nombre est majoré par N. Si les erreurs de transmissions sont indépendantes, la probabilité d'avoir au moins $N+1$ erreurs dans un message de longueur L est $\sum_{k=N+1}^{L} \binom{L}{k} \epsilon^k (1-\epsilon)^{L-k}$, où ϵ est la probabilité d'une erreur de transmission, c'est aussi `1-binomial_cdf(L,epsilon,N)`. Par exemple, pour un message de 10^3 caractères, chacun ayant une probabilité d'erreur de transmission de 10^{-3}, si on prend $N = 3$, alors la probabilité d'avoir au moins 4 erreurs est

de 0.019 (arrondi par excès) :

```
P(N,eps,L):=sum(comb(L,k)*eps^k*(1-eps)^(L-k),k,N+1,L):;
P(3,1e-3,10^3)
```

ou directement `1-binomial_cdf(1000,1e-3,3).`

Exemple : On ne peut pas corriger d'erreur avec le bit de parité.

1.5 Distances

La distance de Hamming de 2 mots est le nombre de symboles qui diffèrent. (il s'agit bien d'une distance au sens mathématique, elle vérifie l'inégalité triangulaire).

Exercice : écrire une procédure de calcul de la distance de Hamming de 2 mots. En Xcas, la fonction s'appelle `hamdist`.

La distance d'un code est la distance de Hamming minimale de 2 mots différents du code. Pour un code linéaire, la distance est aussi le nombre minimal de coefficients non nuls d'un vecteur non nul de l'image. Pour un code polynomial, la distance du code est le nombre minimal de coefficients non nuls d'un multiple de g de degré inférieur à n.

Exercice : quelle est la distance du code linéaire que vous avez créé plus haut ?

Majoration de la distance du code :

La distance minimale d'un code linéaire est inférieure ou égale à $n-k+1$: en effet on écrit en ligne les coordonnées des images de la base canonique (ce qui revient à transposer la matrice) et on réduit par le pivot de Gauss, comme l'application linéaire est injective, le rang de la matrice est k, donc la réduction de Gauss crée $k-1$ zéros dans chaque ligne, donc le nombre de coefficients non nuls de ces k lignes (qui sont toujours des mots de code) est au plus de $n-k+1$.

Exercice : si votre code linéaire n'est pas de distance 3, modifiez les 3 dernières lignes pour réaliser un code de distance 3. On ne peut pas obtenir une distance $n-k+1=4$ avec $n=7$ et $k=4$ dans $\mathbb{Z}/2\mathbb{Z}$, essayez ! Essayez sur $\mathbb{Z}/3\mathbb{Z}$ et $\mathbb{Z}/5\mathbb{Z}$.

N.B. : Pour les codes non polynomiaux, par exemple convolutifs, la distance n'est pas forcément le paramètre le mieux adapté à la correction d'erreurs.

1.6 Correction au mot le plus proche

Une stratégie de correction basée sur la distance consiste à trouver le mot de code le plus proche d'un mot donné. Si la distance d'un code est supérieure ou égale à $2t+1$, et s'il existe un mot de code de distance inférieure à t au mot donné, alors ce mot de code est unique. On corrige alors le mot transmis en le remplaçant par le mot de code le plus proche.

Exercice : écrivez un programme permettant de corriger une erreur dans un mot dans votre code linéaire.

On dit qu'un code t-correcteur est parfait si la réunion des boules de centre un mot de code et de rayon t (pour la distance de Hamming) est disjointe et recouvre l'ensemble des mots (K^n).

Exercice : votre code linéaire sur $\mathbb{Z}/2\mathbb{Z}$ (celui de distance 3) est-il un code 1-correcteur parfait ?

2 Les codes de Reed-Solomon

Il s'agit de codes polynomiaux qui réalisent la distance maximale possible $n-k+1$. De plus la recherche du mot de code le plus proche peut se faire par un algorithme de Bézout avec arrêt prématuré.

2.1 Théorie

On se donne un générateur a de F_q^* et le polynôme $g(x)=(x-a)...(x-a^{2t})$ (donc $n-k=2t$). Typiquement $q=2^m$ avec $m=8$, a est une racine d'un polynôme irréductible de degré m à coefficients dans $\mathbb{Z}/2$ qui ne divise pas x^l-1 pour l diviseur strict de 2^m-1, en pratique, on factorise le quotient de $x^{2^m-1}-1$ par le ppcm des $x^{(2^m-1)/p}-1$ où p parcourt les diviseurs premiers de 2^m-1 et on en extrait un facteur de degré m.

Distance du code

Si la longueur n d'un mot vérifie $n \leq 2^m-1$, alors la distance entre 2 mots du code est au moins de $2t+1$. En effet, si un polynome P de degré $<n$ est un multiple de g ayant moins de $2t+1$ coefficients non nuls,

$$P(x)=\sum_{k=1}^{2t} p_{i_k} x^{i_k}, \quad i_k < n$$

en écrivant $P(a)=...=P(a^{2t})=0$, on obtient un déterminant de Van der Monde, on prouve qu'il est non nul en utilisant la condition $i_k<n$ et le fait que la première puissance de a telle que $a^x=1$ est $x=2^m-1$.

Correction des erreurs

Soit $c(x)$ le polynome envoyé, $d(x)$ le polynôme recu, on suppose qu'il y a moins de t erreurs

$$d(x) - c(x) = e(x) = \sum_{k=1}^{\nu} \alpha_k x^{i_k}, \quad \nu \leq t$$

On calcule le polynome syndrome :

$$s(x) = \sum_{i=0}^{2t-1} d(a^{i+1})x^i = \sum_{i=0}^{2t-1} e(a^{i+1})x^i$$

on a donc :

$$\begin{aligned} s(x) &= \sum_{i=0}^{2t-1} \sum_{k=1}^{\nu} \alpha_k (a^{i+1})^{i_k} x^i \\ &= \sum_{k=1}^{\nu} \alpha_k \sum_{i=0}^{2t-1} (a^{i+1})^{i_k} x^i \\ &= \sum_{k=1}^{\nu} \alpha_k a^{i_k} \frac{(a^{i_k}x)^{2t} - 1}{a^{i_k}x - 1} \end{aligned}$$

On pose $l(x)$ le produit des dénominateurs (que l'on appelle polynôme localisateur, car ses racines permettent de trouver la position des symboles à corriger), on a alors

$$l(x)s(x) = \sum_{k=1}^{\nu} \alpha_k a^{i_k} ((a^{i_k}x)^{2t} - 1) \prod_{j \neq k, j \in [1,\nu]} (a^{i_j}x - 1) \tag{1}$$

Modulo x^{2t}, $l(x)s(x)$ est donc un polynôme w de degré inférieur ou égal à $\nu - 1$, donc strictement inférieur à t. Pour le calculer, on applique l'algorithme de Bézout à $s(x)$ et x^{2t} (dans F_q), en s'arrêtant au premier reste $w(x)$ dont le degré est strictement inférieur à t (au lieu d'aller jusqu'au calcul du PGCD de $s(x)$ et x^{2t}). Les relations sur les degrés (cf. approximants de Padé et la preuve ci-dessous) donnent alors en coefficient de $s(x)$ le polynôme $l(x)$ de degré inférieur ou égal à t. On en calcule les racines (en testant tous les éléments du corps avec Horner), donc la place des symboles erronés.

Pour calculer les valeurs α_k, on reprend la définition de w, c'est le terme de droite de l'équation (1) modulo x^{2t}, donc :

$$w(x) = \sum_{k=1}^{\nu} \alpha_k a^{i_k} (-1) \prod_{j \neq k, j \in [1,\nu]} (a^{i_j}x - 1)$$

Donc :

$$w(a^{-i_k}) = -\alpha_k a^{i_k} \prod_{j \neq k, j \in [1,\nu]} (a^{i_j}a^{-i_k} - 1)$$

Comme :

$$l(x) = (a^{i_k}x - 1) \prod_{j \neq k, j \in [1,\nu]} (a^{i_j}x - 1)$$

on a :

$$l'(a^{-i_k}) = a^{i_k} \prod_{j \neq k, j \in [1,\nu]} (a^{i_j}a^{-i_k} - 1)$$

Finalement :

$$\alpha_k = -\frac{w(a^{-i_k})}{l'(a^{-i_k})}$$

2.2 Preuve du calcul de l

On avait $s(x)$ avec $\deg(s) <= 2t - 1$, il s'agissait de voir comment la solution u, v, r calculee par Bezout

$$u(x)x^{2t} + v(x)s(x) = r(x) \tag{2}$$

avec arrêt prématuré au 1er reste $r(x)$ de degré $<= t-1$ correspondait à l'équation

$$l(x)s(x) = w(x) mod x^{2t}$$

avec deg$(l) <= t$ et deg$(w) <= t-1$

On a vu que deg$(v) <= t$. On commence par factoriser la puissance de x de degré maximal p dans $v(x)$, et on simplifie (2) par x^p. Quitte à changer v et r, on se ramene donc à :

$$u(x)x^{2t-p} + v(x)s(x) = r(x)$$

avec $v(x)$ premier avec x, deg$(v) <= t-p$ et deg$(r) <= t-1-p$. On simplifie ensuite par le pgcd de $v(x)$ et de $r(x)$ (qui divise $u(x)$ car premier avec x puisqu'on a déjà traité les puissances de x). On a donc, quitte à changer de notation u, v, r tels que

$$u(x)x^{2t-p} + v(x)s(x) = r(x)$$

avec v et r premiers entre eux, v premier avec x, deg$(v) <= t-p$ et deg$(r) <= t-1-p$ (N.B. : $p = 0$ en general)

On observe que $l(x)$ est premier avec x (0 n'est pas racine de l). On raisonne modulo x^{2t-p}, l et v sont inversibles modulo x^{2t-p}, donc

$$s(x) = w(x) * inv(l) \pmod{x^{2t-p}}, \quad s(x) = r(x) * inv(v) \pmod{x^{2t-p}}$$

donc

$$w(x) * inv(l) = r(x) * inv(v) \pmod{x^{2t-p}} \quad \Rightarrow \quad w(x) * v(x) = r(x) * l(x) \pmod{x^{2t-p}}$$

donc $w(x) * v(x) = r(x) * l(x)$ à cause des majorations de degres

D'autre part par construction $w(x)$ est premier avec $l(x)$ (car chacun des facteurs de $l(x)$ divise tous les éléments de la somme définissant $w(x)$ sauf un), donc $l(x)$ divise $v(x)$, et comme $v(x)$ est premier avec $r(x)$, on en déduit que $v(x) = Cl(x)$ où C est une constante non nulle, puis $r(x) = Cw(x)$.

Bezout donne donc (après simplifications du couple $v(x), r(x)$ par son pgcd) le polynome localisateur à une constante près (donc les racines et les positions des erreurs), et on peut calculer les valeurs des erreurs avec v et r car la constante C se simplifie.

2.3 Avec Xcas

Récupérez la session
`http://www-fourier.ujf-grenoble.fr/~parisse/agreg/reed_s.xws`