COMMUNICATION SATELLITAIRE ET CODE DE REED-SOLOMON

Rapport final

Aurélien BLICQ

Préambule :

Dans le MCOT, je me suis fixé les objectifs suivants :

- Implémenter le code de Reed-Solomon RS(204,188,8)
- Etudier les performances de ce code
- Discuter les modèles retenus

Pour le décodage de ce code, j'ai choisis d'implémenter l'algorithme PGZ car il s'agit du plus répandu et efficace.

J'ai également étudié les performances de correction d'erreurs de ce code et les contraintes que son utilisation fait peser sur un système d'information.

En revanche, je n'ai pu mettre au point aucune expérience simple pour discuter si le modèle du canal binaire symétrique est ou non adapté à la communication satellitaire.

Introduction :

L'implémentation du code de Reed-Solomon m'a amené à devoir implémenter le corps fini à 256 éléments et les polynômes à coefficients sur ce corps. J'ai également dû implémenter un canal bruité : le canal binaire symétrique.

Pour étudier les performances de RS(204,188,8), j'ai développé une procédure de test à partir de mon implémentation de ce code, permettant d'évaluer sa capacité de correction.

Corps principal :

Implémentation du code de Reed-Solomon :

Le code de Reed-Solomon RS(n, k, t) se base sur les propriétés des polynômes à coefficients dans le corps fini à 256 éléments GF(256). En voici le principe :

- Codage : un message M, c'est-à-dire une suite finie d'octets, est interprétée comme un polynôme à coefficients dans GF(256). On dispose donc de la somme, du produit et de la division euclidienne de deux messages.
  On appelle générateur du code de Reed-Solomon et on note g(X) le polynôme à coefficients dans GF(256):

$$g(X) = \prod_{i=1}^{2t} (X - \alpha^i)$$

Où α est le générateur de GF(256).
Alors le mot de code représentant M est $C(X) = M(X) * X^{2t} + CK(X)$, avec $CK(X) = M(X) * X^{2t} \bmod g(X)$ le contrôle de parité associé à M

- Décodage : Le décodage d'un mot reçu R se fait selon le schéma suivant :
  1) Calcul des syndromes $S_i=R(\alpha^i)$ (i=1, 2, …, 2t) par l'algorithme de Horner
  2) Détermination des polynômes σ et ω, les polynômes localisateur et évaluateur d'erreurs par un algorithme basé sur l'algorithme d'Euclide étendu
  3) Détermination des positions des erreurs qui sont les inverses des racines de σ par l'algorithme de la recherche de Chien
  4) Calcul des valeurs des erreurs via une formule explicite en fonction de σ et ω appelée algorithme de Forney

Il est donc nécessaire d'implémenter GF(256) et GF(256)[X].

Usuellement, on définit GF(256) comme suit : Soit P_0 un polynôme irréductible à coefficients dans $\mathbb{Z}/2\mathbb{Z}$. GF(256) est l'ensemble des classes d'équivalence pour la congruence modulo P_0. Dans la suite, on appellera octets les éléments de GF(256)

Dans une première implémentation, j'ai donc défini les polynômes à coefficients dans $\mathbb{Z}/2\mathbb{Z}$ à l'aide des arrays de numpy. Les fonctions principales sont :

- l'addition : on fait la somme terme à terme en appliquant l'opération '%2' à chaque terme
- la multiplication : on utilise la définition du produit de polynômes
- la division euclidienne : on implémente l'algorithme usuel

En prenant pour polynôme irréductible $P_0 = X^8+X^4+X^3+X^2+1$, on peut ainsi aisément implémenter GF(256) : la somme dans GF(256) est la même que dans $\mathbb{Z}/2\mathbb{Z}[X]$, le produit est réalisé en prenant le reste de la division euclidienne par P_0 au produit des polynômes, et on calcul l'inverse de $P \in GF(256)$ en appliquant l'algorithme d'Euclide étendu à P_0 et P.

Pour définir nos messages, donc les polynômes sur GF(256), on définit la somme, le produit et la division euclidienne sur le même principe.

J'ai néanmoins remarqué que cette implémentation entraine un temps de calcul très long, notamment pour le codage d'un mot qui nécessite la division euclidienne d'un mot de 204 octets par un mot de 16 octets et qui, sur mon ordinateur personnel, prenait 1 minute à s'exécuter.

Afin de résoudre ce problème, j'ai donc implémenté les octets d'une manière plus efficace. Dans cette seconde implémentation, un octet est un élément du type numpy.uint8, c'est-à-dire un entier non signé codé sur 8 bits. Un octet est alors confondu avec le nombre dont il est la représentation en base 2. La somme est le ou exclusif bit à bit (réalisé par la fonction numpy.bitwise_xor) et pour le produit et l'inverse, j'ai généré, grâce à ma première implémentation des tables de valeur permettant de réaliser les fonctions suivantes :

- exp : prend un entier k en renvoie α^k où α est un générateur du groupe GF(256)*
- log : prend un octet non nul et renvoie son logarithme en base α
- inv : prend un octet et renvoie son inverse

le produit de a et b est alors exp(log(a)+log(b))
On a ainsi une implémentation bien plus efficace, qui permet un codage en quelques secondes.

Discussion des performances :

L'ajout de redondance dans un message conduit nécessairement à une dilution de l'information. Pour caractériser cette dilution, on définit $\tau=k/n$ le taux d'information du code RS(n, k, t), c'est-à-dire le taux d'octets du message convoyant réellement de l'information. Comme on a $n = k + 2t$, on pourrait se dire qu'on devrait choisir n le plus grand possible, c'est-à-dire $n = 255$ pour un code utilisant GF(256). Néanmoins, la complexité $O(n^2)$ des algorithmes de codage et décodage conduit à un temps de calcul bien plus important comme le montre le tableau suivant :

	RS(204,188)	RS(255,239)
Codage (s)	4	7
Syndromes (s)	0,08	0,1
Algorithme euclidien (s)	0,01	0,02
Chien search (s)	0,02	0,03
Algorithme de Forney (s)	0,001	0,002
Calcul de l'erreur (s)	0,004	0,007
Total décodage (s)	0,1	0,15
Total (s)	**4,1**	**7,15**

Tableau 1 : comparatif des temps d'exécution

On voit qu'un paramètre n petit favorise le débit d'octet. Ainsi, n = 204 réalise un compromis entre le taux d'information et le débit d'octet, et optimise ainsi le débit d'information du code.

Le code de Reed-Solomon permet de corriger parfaitement 8 erreurs et moins. Afin d'étudier son comportement dans le cas où plus de 8 erreurs ont été introduites, on réalise un programme qui génère des erreurs aléatoires sur un mot codé, le décode et test si ces erreurs ont été corrigées, totalement ou en partie, et regarde si des erreurs n'ont pas été ajoutées au mot. On constate ainsi empiriquement, sur un échantillon de 10 000 erreurs, que si plus de 8 erreurs ont été introduites, aucune n'est jamais corrigée entièrement, que dans 10% des cas, certaines erreurs sont corrigées et que dans tous les cas, des erreurs sont ajoutées.

Conclusion :

Après avoir implémenté le code RS(204,188,8), j'ai effectué des tests qui m'ont permis d'établir que les paramètres de ce code permettent d'optimiser le débit d'information de celui-ci et ainsi de le rendre son utilisation moins contraignante.

Ce code possède également une excellente capacité de correction de 8 erreurs sur les octets, même si au-delà, sa capacité de correction est faible.

Ainsi, le code RS(204,188,8) est particulièrement adapté à la communication satellitaire, ce qui explique son utilisation largement repandue.

Bibliographie additionnelle :

[1] Jagadeesh Sankaran, *Reed Solomon Decoder: TMS320C64x Implementation,* texas instrument application report SPRA686, Decembre 2000

[2] John Gill, *EE 387 Notes #7 Handout #24,* Stanford University