Université Paris 8 Année 2022–2023

M1 Mathématiques et applications, parcours ACC

Théorie de l'information — Devoir maison

Transmis le 8 novembre 2022

—

À rendre jusqu'au vendredi 23 décembre 2022

Consignes.

Vous devez **rendre par email** adressé à `julien.lavauzelle@univ-paris8.fr`, jusqu'au vendredi 23 décembre 2022, une archive au format `.zip` contenant les documents suivants :

- votre ou vos **fichiers de programmation**,
- un **rapport** rédigé sur ordinateur, au format `.pdf` **obligatoirement**, comprenant vos réponses aux questions nécessitant des explications.

Vous pouvez utiliser le langage de programmation de votre choix, en vous aidant si besoin de bibliothèques externes (sauf si elles implémentent directement la réponse aux questions...). Les intitulés/paramètres de fonctions à implanter peuvent être adaptés au besoin.

Le **soin** et les **justifications** apportées à vos réponses seront évaluées. On notera également le soin donné à l'implantation (commentaires, lisibilité, etc.).

Codes convolutifs et décodage de Viterbi

Ce devoir a pour but de vous faire implanter et expérimenter les codes convolutifs et leur algorithme de décodage, l'algorithme de Viterbi.

On rappelle qu'un **code convolutif** est la donnée de k **générateurs** $g_1, \dots, g_k$, qu'on peut représenter sous la forme de polynômes $g_i(z) = g_{i,0} + g_{i,1}z + \cdots + g_{i,m}z^d \in \mathbb{F}_2[z]$. On note $n-1$ le plus haut degré des générateurs.

Pour encoder un message $\boldsymbol{x}$ constitué d'une suite de bits $(x_0, x_1, x_2, \dots, x_m, \dots)$ de longueur arbitraire, on rappelle que l'on crée une séquence de k mots de code

$$\begin{aligned} \boldsymbol{c}_1 &= (c_{1,0}, c_{1,1}, \dots, c_{1,m}, \dots) \\ \boldsymbol{c}_2 &= (c_{2,0}, c_{2,1}, \dots, c_{2,m}, \dots) \\ &= \quad \vdots \\ \boldsymbol{c}_k &= (c_{k,0}, c_{k,1}, \dots, c_{k,m}, \dots) \end{aligned}$$

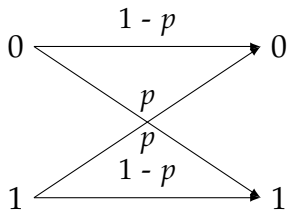
de sorte que pour tout $j \in \{1, \dots, k\}$ et tout $m \geq 0$ on ait (en supposant $x_i = 0$ pour $i < 0$) :

$$c_{j,m} = \sum_{i=0}^{n-1} g_{j,i} x_{m-i} \, .$$

On appelle **registre** les bits de message $(x_m, x_{m-1}, \dots, x_{m-n+1})$ qui permettent de calculer le bit $c_{j,m}$ du mot de code associé au j-ème générateur.

Implantation des premières fonctions

On rappelle que le canal binaire symétrique de paramètre p peut être représenté par le diagramme ci-contre.



Question 1.– Implanter une fonction `bsc(x, p)` qui représente le passage d'un message `x` dans le canal binaire symétrique de paramètre `p`. La fonction prend donc en entrée une suite de bits `x` de longueur arbitraire, ainsi qu'un nombre folttant $p \in [0,1]$, et retourne une suite de bits de même longueur, où chaque bit a été modifié selon une loi de Bernoulli de paramètre `p`.

On considère maintenant un code convolutif représenté par une liste de génarateurs. Plutôt que de passer par le formalisme des polynômes, on va simplement représenter un générateur `gen` par sa liste de coefficients `[gen[0], gen[1], ..., gen[n-1]]`.

De même, l'état d'un registre sera représenté par une liste de longueur n, qui pourra être ordonnée selon votre préférence : `[x[m], x[m-1], ..., x[m-n+1]]` (comme souvent dans le cours) ou `[x[m-n+1], x[m-n+2], ..., x[m]]` (parfois plus facile à manipuler).

Question 2.– Implanter une fonction `apply_generator(state, gen)` qui retourne le bit de sortie associé au générateur `gen` lorsque le registre est dans l'état `state`.

Question 3.– Implanter `encode(x, generators)`, la fonction d'encodage d'un message `x` par les générateurs `generators` d'un code convolutif. S'il y a k générateurs, la fonction retournera donc une liste de k mots de code.

→ *on pourra vérifier que pour le code de polynômes* $1 + z + z^2$ *et* $1 + z$*, et le message* `[0,1,0,1]`*, les mots de codes associés sont* `[[0, 1, 1, 0, 1, 1], [0, 1, 1, 1, 1, 0]]`.

Étant donné une séquence de k mots de codes $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k)$ calculée par encodage d'un message $\boldsymbol{x}$, on souhaiterait maintenant retrouver le message $\boldsymbol{x}$. Commençons par le cas où aucune erreur n'a été introduite sur les mots de code.

Question 4.– Implanter une fonction `decode(list_c, generators)` qui permet de retrouver le message `x` à partir d'une liste de mots de code `list_c` issus d'un code convolutif de générateurs `generators`. On supposera ici qu'il n'y a aucune erreur sur les mots de codes.

→ *on utilisera l'exemple précédent pour vérifier son implantation.*

Question 5.– Tester les fonctions `encode` et `decode`. On vérifiera notamment qu'elles sont bien l'inverse l'une de l'autre, sur des messages tirés aléatoirement.

Algorithme de Viterbi

Dans cette partie, on s'intéresse à la correction d'erreurs intervenues par passage dans le canal. Pour cela, on se réfère à l'algorithme de Viterbi, vu en cours, et dont une descritpion détaillée et des exemples d'exécutions sont disponibles dans les notes de cours.

Question 6.– Implanter l'algorithme de décodage de Viterbi. La fonction prendre en entrée une séquence de mots erronés $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k$ et un ensemble de générateurs du code, et retournera en sortie un message $\boldsymbol{x}$ tel que l'encodage de $\boldsymbol{x}$ est le plus proche possible des $\boldsymbol{y}_i$.

Attention : cette question est plus difficile. Il faudra notamment

- se questionner sur la manière de représenter un état du treillis (par exemple par un entier) ;
- probablement programmer des fonctions auxiliaires (par exemple :distance de Hamming, conversion entier $\longleftrightarrow$ liste de bits).

Pour cette question, il vous est demandé de **bien commenter votre code**, notamment si vous n'êtes pas certain du résultat final.

Question 7.– Grâce à votre implantation, décoder les mots erronés suivants :

$$[[1,0,1,0,1,1,1,0,1,1,1,1,0,1,1,0,0],[1,1,1,1,0,0,1,0,0,1,1,0,1,1,0,0,0]]$$

issus du code de polynômes $g_1(z) = 1+z+z^3$ et $g_2(z) = 1+z^2+z^3$. Combien d'erreurs ont été introduites sur la séquence de mots de codes ?

Taux d'erreur résiduel

Pour mesurer à l'efficacité d'un code convolutif, on s'intéresse au taux d'erreur résiduel par bit (BER, *bit error rate*), c'est-à-dire au nombre de bits mal décodés après passage dans l'algorithme de Viterbi, divisé par le nombre de bits de mesage à transmettre.

Question 8.– Implanter une fonction qui prend en entrée les générateurs `generators` d'un code convolutif, une taille de message `length` et un taux d'erreur `p` d'un canal binaire symétrique, et qui calcule le taux d'erreur résiduel par bit correspondant.

Pour calculer le BER :

— on effectuera entre 100 (minimum) et 1000 (c'est mieux) tests par paramètre à évaluer ;
— les messages seront tirés aléatoirement, et leurs mots de codes associés seront transmis à travers le canal binaire symétrique.

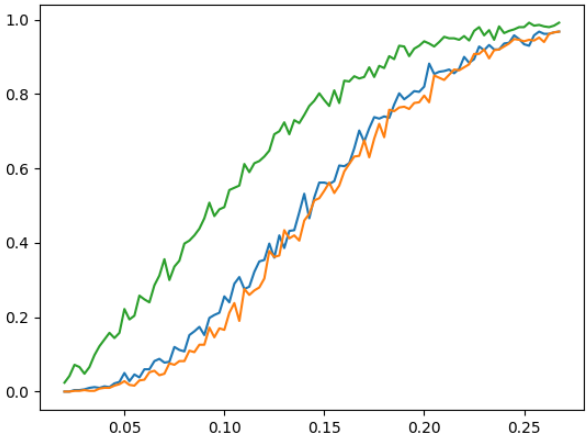
Question 9.– Tracer les courbes de BER en fonction du paramètre d'erreur p, pour les codes suivants (les coefficients de polynômes sont donnés en ordre croissant de degré) :

1. $k = 2$, $n = 4$, et les coefficients de polynômes `[[1,0,0,0,1], [1,1,1,1,1]]`
2. $k = 2$, $n = 4$, et les coefficients de polynômes `[[1,1,1,0,1], [1,0,0,1,1]]`
3. $k = 2$, $n = 4$, et les coefficients de polynômes `[[1,0,0,0,1], [1,0,0,0,0]]`
4. $k = 2$, $n = 5$, et les coefficients de polynômes `[[1,0,1,0,0,1], [1,1,1,0,1,1]]`
5. $k = 2$, $n = 6$, et les coefficients de polynômes `[[1,1,1,1,0,0,1], [1,0,1,1,0,1,1]]`

Notez que le dernier code a été utilisé pour les télécommunications de la sonde spatiale Voyager.

Pour information, vous devriez vous attendre à des courbes de la forme ci-contre (en abscisse, p, en ordonnée, le BER).

(bien entendu, les courbes ci-contre ne sont pas exactement celles attendues)



Question 10.– Selon vos résultats expérimentaux, quel est le « meilleur » code pour $n+1 = 5$? Justifier.

Autour de la distance libre

Dans cette section, on s'intéresse à la notion de **distance libre** d'un code convolutif. Par définition, c'est le poids de Hamming du plus petit message non-nul encodé par le code convolutif :

$$\Delta \coloneqq \min_{\boldsymbol{x} \neq 0} \left\{ \mathrm{wt}(\boldsymbol{c}_1) + \cdots + \mathrm{wt}(c_k) \mid (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k) \text{ est l'encodage de } \boldsymbol{x} \right\}.$$

Question 11.– Quelle est la distance libre du code convolutif à 1 générateur de degré n, constitué du polynôme $g(z) = 1 + z + z^2 + \cdots + z^n = \sum_{i=0}^{n} z^i$?

Question 12.– Quelle est la distance libre du code convolutif à k générateurs de degré maximal k, constitué des polynômes $g_j(z) = z^{j-1}$ pour $j = 1, \ldots, k$?

Question 13.– Démontrer que la distance libre d'un code convolutif à k générateurs de degré n est plus petite que kn.

Question 14.– **[plus difficile]** Démontrer que, si $\boldsymbol{x}$ est tel que le poids de son encodage est égal à la distance libre du code, alors $\boldsymbol{x}$ est nul à partir du rang $n + 1$.

Question 15.– En déduire (en pseudo-code) un algorithme « simple » qui calcule la distance libre d'un code convolutif. On donnera sa complexité en fonction de k et n.

Question 16.– Implanter cet algorithme, puis calculer la distance libre des 3 codes donnés en Question 9.