Université de Liège
Faculté des Sciences Appliquées

Théorie de l'information et du codage

Notes relatives aux travaux pratiques

Année académique 2006-2007

Table des matières

1 Introduction **2**
1.1 Outils mis à disposition 2
1.2 Organisation 2

2 Algorithme de Viterbi **3**

3 Turbo-codes **5**

4 Correction d'erreurs en rafale **7**

5 Description du modèle turbo_wgn **8**

6 Description du modèle bch_interleaver_markov **10**

7 Définir un encodeur convolutionnel dans Matlab **12**

8 Pilotage de Simulink par script **13**

1 Introduction

Ces notes reprennent l'ensemble des informations nécessaires aux travaux pratiques sur ordinateur du cours de théorie de l'information et du codage.
Pour l'année académique 2006-2007, le travail porte essentiellement sur l'étude du codage de canal et fait appel à Matlab.
Le but de ces travaux pratiques est de permettre aux étudiants de manipuler de façon concrète les notions théoriques introduites dans ce cours grâce à différentes expériences.

1.1 Outils mis à disposition

Deux modèles illustrant des techniques de codage de canal sont proposés pour la réalisation de ce travail. Il s'agit de fichiers de type .mdl, qui s'ouvrent dans Matlab. Les fichiers .mdl sont des fichiers Simulink. Simulink est un composant de Matlab qui permet de notamment de construire et simuler des systèmes sous forme de blocs-diagrammes.

Pour faire fonctionner les modèles, il est nécessaire de disposer des composants Matlab suivants : Communications toolbox, Simulink, Communications Blockset, Signal Processing Blockset.

1.2 Organisation

- Le travail doit être réalisé par groupe de deux étudiants.
- Une fois constitué, le groupe enverra un e-mail à l'adresse **bdf@montefiore.ulg.ac.be** signalant les noms des étudiants constituant ce groupe, ainsi qu'une adresse e-mail. Les modèles Simulink (qui fonctionnent sous Matlab) nécessaires à la réalisation du travail seront envoyés à cette adresse.

On demande aux étudiants de remettre un rapport écrit décrivant, discutant et justifiant de façon suffisamment détaillée les résultats obtenus par expérimentation. Le but est de montrer qu'on a bien compris les notions du cours sur lesquelles porte le travail.

Les changements apportés aux fichiers Simulink, tels que la modification des paramètres des blocs, doivent être clairement détaillés, de façon à pouvoir reconstituer les simulations à partir des fichiers initialement transmis.

Le rapport du groupe doit être remis par e-mail à **bdf@montefiore.ulg.ac.be** et une copie papier du rapport doit être remise à la date limite.

La date limite de remise des travaux est fixée au jour de l'examen oral.

2 Algorithme de Viterbi

Soit $(\mathcal{X}_t, \mathcal{Y}_t)$ $(t = 1, 2, \ldots)$ une chaîne de Markov cachée d'ordre 1, invariante dans le temps, à deux états 1 et 2, et deux symboles observables 1 et 2.
Cette chaîne admet la matrice de transition

$$\mathbf{\Pi} = \begin{bmatrix} 0.1 & 0.9 \\ 0.3 & 0.7 \end{bmatrix},$$

dont l'élément ij donne la probabilité $P(\mathcal{X}_{t+1} = j | \mathcal{X}_t = i)$.
Les probabilités de l'état initial sont données par

$$\pi = [0.4 \quad 0.6];$$

l'élément i de π vaut la probabilité $P(\mathcal{X}_1 = i)$.
A chaque instant t, un symbole $\mathcal{Y}_t$ est émis. Il y a deux symboles possibles, 1 et 2. Les probabilités d'émission des symboles dépend de l'état de la chaîne à l'instant t considéré. Plus précisément, on donne la matrice d'observation

$$\mathbf{\Sigma} = \begin{bmatrix} 0.8 & 0.2 \\ 0.45 & 0.55 \end{bmatrix},$$

dont l'élément ij donne la probabilité $P(\mathcal{Y}_t = j | \mathcal{X}_t = i)$.

On demande

1. De dessiner le réseau bayésien qui représente les relations entre les variables impliquées aux 3 premiers instants, c'est-à-dire $\mathcal{X}_1, \mathcal{Y}_1, \mathcal{X}_2, \mathcal{Y}_2, \mathcal{X}_3, \mathcal{Y}_3$.

2. D'écrire une fonction MATLAB

   ```
   function [xx,yy] = simMC(T,pi_init,trans,sigm)
   ```

   capable de générer une suite d'états $\mathcal{X}_t$ et d'observations $\mathcal{Y}_t$ de la chaîne, de l'instant 1 à l'instant T. La fonction prendra en argument `T` le nombre de pas de temps à simuler, `pi_init` le vecteur π, `trans` et `sigm` les matrices de transition et d'observation. La fonction renverra un vecteur colonne `xx` peuplé de 1 et de 2 pour les états, et un vecteur colonne `yy` peuplé de 1 et de 2 pour les symboles émis.
   Suggestion : comment générer les symboles 1 et 2 avec une probabilité p_1, p_2 à partir des réalisations d'une variable aléatoire à distribution uniforme dans l'intervalle [0,1] ?
   La fonction `rand` effectue des tirages (pseudo) aléatoires uniformément dans l'intervalle [0,1] (aide avec `help rand`).

3. D'écrire un script qui lance la génération de N suites d'états $\mathcal{X}_t$ et d'observations $\mathcal{Y}_t$ de longueur T pour construire les estimations $\tilde{\pi}$, $\tilde{\mathbf{\Pi}}$ et $\tilde{\mathbf{\Sigma}}$ de la distribution de probabilité de l'état initial, des matrices de transition et d'observation, à partir des fréquences d'apparition des états (qu'on suppose ici et ici seulement observés) et des symboles émis.
   Ce script devrait permettre de vérifier le bon comportement de la fonction `simMC` en comparant les estimations au vecteur π et aux matrices $\mathbf{\Pi}$ et $\mathbf{\Sigma}$.
   Assurez-vous que c'est-ce bien le cas (en choisissant des valeurs N, T suffisamment élevées).
   Vérifiez que les fréquences sont correctement calculées en choisissant des valeurs N, T très faibles afin de pouvoir recompter les transitions et les observations par examen visuel de `xx` et `yy`.

4. D'implémenter l'algorithme de Viterbi (décrit au chapitre 5 du cours théorique) pour l'estimation des états à partir des obervations.
   Autrement dit, écrire une fonction

```
function ee = viterbiMC(yy,pi_init,trans,sigm)
```

qui renvoie l'estimation `ee` des états $\mathcal{X}_1, \ldots, \mathcal{X}_T$ selon les observations $\mathcal{Y}_1, \ldots, \mathcal{Y}_T$ reportées dans `yy`, connaissant la distribution initiale des états et les matrices de transition et d'observation de la chaîne de Markov cachée.

5. De tester les performances de votre implémentation de l'algorithme de Viterbi dans les conditions l'exercice.
On propose le protocole suivant :

i. Initialiser le générateur de nombres aléatoires : `rand('state',0);`

ii. Générer une suite d'états `xx` et d'observations `yy` de longueur $T = 10$ à l'aide de `simMC`.

iii. Générer l'estimation des états `ee` à l'aide de `viterbiMC`.

iv. Estimer le taux d'erreur en comptant le nombre de fois qu'un état estimé ne correspond pas à l'état réel de la chaîne, et en divisant ce nombre par T.

v. Estimer le taux d'erreur moyen en répétant plusieurs fois les étapes 2 à 4 pour différentes suites générées, afin de moyenner les taux obtenus en 4.

Comparez le taux d'erreur moyen obtenu au taux d'erreur moyen que vous obtiendriez en utilisant à la place de `viterbiMC` la fonction naïve `naiveMC` sur la même suite d'observation `yy`, pour obtenir une estimation naïve `ee_naive` des états :

```
function ee_naive = naiveMC(yy)
ee_naive = yy;
```

6. De reproduire le protocole du point précédent pour différentes longueurs T_k de séquences. Portez le taux d'erreur moyen relatif à `viterbiMC` et celui relatif à `naiveMC` sur un même graphique, en fonction des longueurs T_k (allez au moins jusqu'à $T = 1000$).
Constatez-vous une dégradation des performances de `viterbiMC` avec la longueur des séquences ? Est-ce que les performances de `naiveMC` se mettent à dépasser celles de `viterbiMC` ?

7. Si les performances de `naiveMC` se dégradent avec T, c'est anormal ! Proposez alors une modification de votre fonction `viterbiMC` remédier au problème. Testez votre nouvelle version selon le protocole du point précédent.
Suggestion : votre implémentation se comporte-t-elle bien numériquement ?
Si les performances de votre fonction `viterbiMC` ne se dégradent pas avec T, cette question est sans objet.

8. De justifier de façon théorique le taux d'erreur moyen de `naiveMC` obtenu pour des longueurs de séquences suffisamment longues (disons T=2000).
Suggestion : pour quelles raisons l'hypothèse des séquences longues va-t-elle simplifier le calcul théorique du taux d'erreur moyen ?

9. Finalement, après avoir mené vos expériences, que pensez-vous de l'algorithme de Viterbi appliqué à une chaîne de Markov cachée d'ordre 1 ?
Synthétisez vos réflexions en abordant, par exemple, la question de la sensibilité à la bonne estimation des paramètres $(\pi, \mathbf{\Pi}, \mathbf{\Sigma})$, les performances auxquelles vous vous attendriez pour une chaîne de Markov d'ordre supérieur à 1, etc.
Alternativement, vous pourriez aussi tester d'autres paramètres $(\pi, \mathbf{\Pi}, \mathbf{\Sigma})$ de la chaîne, observer l'effet sur l'écart de performance entre `viterbiMC` et `naiveMC`, et justifier les résultats obtenus.
Le but est de montrer que vous cherchez à comprendre ce que vous observez sur vos simulations.

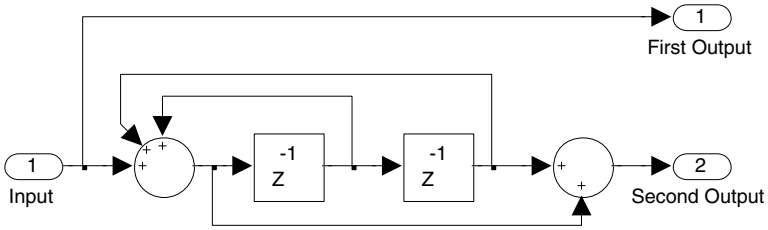
3 Turbo-codes

Pour cette question vous disposez d'un modèle SIMULINK (fichier turbo_wgn.mdl).
Grâce à lui, vous pouvez simuler le comportement d'un système source/encodeur/canal bruité/décodeur reposant sur les turbo-codes, et obtenir les performances du système en terme de taux d'erreur par bit source suivant différents paramètres.
Le modèle se présente sous la forme d'un bloc-diagramme à ouvrir dans MATLAB. Un double-clic sur un bloc permet de déterminer les éventuels paramètres du bloc. La sélection d'un bloc, suivie du choix de l'article "Look under mask" dans le menu "Edit" de la fenêtre du modèle, permet d'ouvrir une nouvelle fenêtre avec le contenu du bloc.

Lisez la description du modèle en section 5 et la façon dont un encodeur convolutionnel est spécifié dans Matlab en section 7. Comprenez bien les figures 1 et 2 de la section 5 qui montrent comment le modèle est construit.
Consultez éventuellement la section 8 pour apprendre comment modifier les paramètres et lancer des simulations sans passer par l'interface graphique de SIMULINK.

Pour cette question, on utilisera l'encodeur convolutionnel récursif

```
t2 = poly2trellis(3,[7 5],[7])
```



qui est déjà spécifié dans `turbo_wgn.mdl` en paramètre.

On demande :

10. De générer une table (dite table du code) qui donne, pour chaque message de 4 bits, le code correspondant. Le code peut être obtenu au moyen de la fonction `convenc` de MATLAB, dont les arguments sont le message et l'encodeur convolutionnel à utiliser.
Par exemple, pour générer le code `c` relatif à 0000, il faut entrer dans la fenêtre de commande

```
c = convenc([0 0 0 0],t2)
```

après avoir défini `t2` au moyen de

```
t2 = poly2trellis(3,[7 5],[7]);
```

Présentez cette table de code et vérifiez en examinant la table que le code est systématique (dans le sens suivant : on peut retrouver les bits du message sur certains bits bien déterminés du mot de code).

11. De calculer la réponse impulsionnelle de l'encodeur grâce à la fonction `convenc` (faire un graphique en bâtonnets). Comparer le schéma de `t2` au schéma de la figure 15.11 du cours théorique. Comparer la réponse impulsionnelle de `t2` à l'expression de la section 15.11.1.1 du cours théorique :

$$h(D) = \frac{1 + D + D^2 + D^3}{1 + D^3} = (1 + D + D^2 + D^3)(1 + D^3 + D^6 + D^9 + \ldots + D^{3k} + \ldots).$$

12. De déduire du schéma de la figure 2 (voir section 5) la puissance du signal envoyé sur le canal (AWGN Channel). Sur la figure 2, on voit que le bloc "Unipolar to Bipolar Converter" convertit les bits 0 et 1 en réels -1 et +1, au moment de la transmission sur le canal.

13. De donner le mode d'utilisation du canal, et de dire en quoi cela influe sur la capacité de celui-ci.
Fixer le paramètre `noiseVar`, la variance du bruit du canal, à 1.
Si le canal qu'on notera $\mathcal{C}$ était remplacé par un modèle de canal symétrique binaire $\mathcal{C}'$ (section 9.1.3.4 du cours théorique), quelle serait la probabilité d'erreur p de la matrice de transition du canal $\mathcal{C}'$?
Notez bien que dans le cours théorique, cette matrice doit se lire

$$\begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix}$$

pour être cohérente avec la figure correspondante du cours ; p est bien la probabilité que la sortie du canal soit 1 (resp. 0) lorsque l'entrée est 0 (resp.1).
Evaluer la capacité du canal $\mathcal{C}$. Evaluer revient ici à donner des bornes inférieures et supérieures sur la capacité, à partir des capacités des deux autres modes d'utilisation du canal.
Comparer au débit du turbo-code.

14. De relever les taux d'erreur par bit obtenus pour des messages de longueur N=4, 128 ou 1024 bits. Considérer 1, 3, 9, 13 ou 19 itérations. Il y a donc en tout 15 taux à relever. Présenter les résultats graphiquement. Commenter les résultats obtenus.

4 Correction d'erreurs en rafale

Pour cette question, vous disposez d'un modèle SIMULINK (fichier `bch_interleaver_markov.mdl`). Le modèle `bch_interleaver_markov` illustre les idées des sections 15.5.4.9 et 15.5.4.10 du cours théorique. On cherche un codage de canal qui détecte et corrige des erreurs en rafale, i.e. des erreurs sur plusieurs bits consécutifs du message qui transite par le canal. Pour modéliser un canal introduisant des erreurs en rafale, on utilise une chaîne de Markov de mémoire 1 dont l'état à l'instant t est noté $\mathcal{N}_t$ et prend sa valeur dans $\{0, 1\}$. Si on note $\mathcal{X}_t$, et $\mathcal{Y}_t$ respectivement l'entrée 0/1 et la sortie 0/1 du canal à l'instant t, on a la relation

$$\mathcal{Y}_t = (\mathcal{X}_t + \mathcal{N}_t) \bmod 2$$

pour modéliser le comportement du canal bruité. Le fait d'utiliser une chaîne de Markov de mémoire 1 seulement n'est pas une très bonne idée pour bien représenter des erreurs en rafale, mais permet néanmoins de déjà mettre en évidence certaines caractéristiques de ce type de bruit.

Lisez la description du modèle et de ses paramètres en section 6 et comprenez bien les figures correspondantes.

Considérez un canal (binaire) dont la matrice de transition du bruit additif binaire (cf section 6) est

$$\mathbf{\Pi} = \begin{bmatrix} 0.95 & 0.05 \\ 0.50 & 0.50 \end{bmatrix}.$$

Ainsi, si le bruit généré selon ce modèle est la séquence `[0 0 0 1 1 1 0 1 1 1 0 1]`, et que l'entrée du canal est le signal `[1 1 1 1 1 1 0 0 0 0 0 0]`, on obtiendra le signal `[1 1 1 0 0 0 0 1 1 1 0 1]` à la sortie du canal.

On demande :

15. De calculer la distribution stationnaire $\boldsymbol{\pi_0}$ de la matrice de transition du bruit $\mathbf{\Pi}$.

16. De calculer la capacité du canal. La capacité du canal avec mémoire (finie) est définie par

$$C = \lim_{n\to\infty} \max_{P(\mathcal{X}^n)} \frac{I(\mathcal{X}^n; \mathcal{Y}^n)}{n}.$$

17. D'écrire un script capable, à partir du signal `channelnoise` obtenu à l'issue d'une simulation du modèle `bch_interleaver_markov`, de compter le nombre d'erreurs de longueur 1,2,... introduites par le canal. Le script commencera par

```
channelnoise = reshape(channelnoise', 1, numel(channelnoise));
```

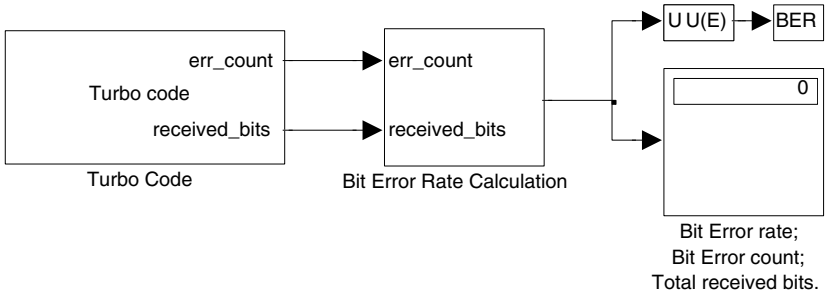
car le signal `channelnoise` arrive sous forme de matrice. Le script doit renvoyer un vecteur dont le ième élément est le nombre d'erreurs de longueur i. Par exemple, la séquence `[0 0 0 1 1 1 0 1 1 1 0 1]` correspond à l'introduction de deux erreurs de longueur 3, et d'une erreur de longueur 1 ; le script renverra donc `[1 0 2]`.

18. De comparer la distribution des erreurs de longueur 1,2,... du canal, dont la matrice de transition du bruit est $\mathbf{\Pi}$, et initialisé selon sa distribution stationnaire $\boldsymbol{\pi_0} = [1-p, p]$, à la distribution des erreurs de longueur 1,2,... d'un signal de bruit généré par un canal symétrique binaire (de probabilité d'erreur p).

19. D'expérimenter différentes valeurs du nombre de blocs `nB` et de codes (N,K) obtenus en modifiant `K`. Régler le paramètre `p_ini` pour initialiser le bruit du canal selon sa distribution stationnaire. Il n'est pas nécessaire de prendre des valeurs de K telles que N devient trop grand (disons au-delà de N=127).

20. De commenter les résultats de simulation obtenus.

FIG. 1 – Le modèle `turbo_wgn` calcule le taux d'erreur par bit source d'un système reposant sur les turbo-codes.



5 Description du modèle turbo_wgn

Le modèle `turbo_wgn` permet de tester l'encodeur et le décodeur de turbo-code de la section 15.11 du cours théorique. Des messages de N bits (tous équiprobables) émis par une source sont codés puis transitent par un canal à bruit blanc gaussien additif. Le décodage a ensuite lieu en sortie du canal. Un taux d'erreur est calculé en comparant les symboles émis par la source aux sorties obtenues à l'issue du processus de décodage.
Il existe différents paramètres pour fixer le type de source, de canal, et d'encodeur et de décodage. On peut les modifier en double-cliquant sur le bloc "Turbo Code" représenté figure 1.

Les paramètres sont les suivants.

i. `N` fixe le nombre de bits qui composent le message destiné à être encodé.

ii. `noiseVar` fixe la variance du bruit blanc gaussien du canal par lequel le message encodé transite.

iii. `t` est la structure qui définit l'encodeur convolutionnel récursif du turbo-code, typiquement obtenue via la fonction Matlab `poly2trellis` (voir section 7).

iv. `r0` est un entier positif qui fixe l'état initial commun de l'entrelaceur et du désentrelaceur.

v. `numIter` donne le nombre d'itérations à effectuer lors du décodage.

Le contenu du bloc "Turbo Code" peut être affiché. Il faut sélectionner ce bloc, puis choisir "Look Under Mask" dans le menu "Edit". Une fenêtre s'ouvre, contenant les blocs repris à la figure 2. Tous les entrelaceurs et désentrelaceurs utilisent `r0` comme état initial. Tous les encodeurs et décodeurs utilisent `t` comme treillis d'encodeur convolutionnel.

Le bloc "Bit Error Rate Calculation" permet d'afficher le taux d'erreur du système (BER : Bit Error Rate), obtenu en divisant le nombre de bits erronés obtenus en sortie du système (Bit Error count) par le nombre total de bits obtenus en sortie du système (Total received bits). En double-cliquant sur ce bloc, on peut régler les paramètres suivants, qui déterminent en fait deux critères d'arrêt possibles (à utiliser seuls ou combinés) pour la simulation.

i. `maxNumErrs` : nombre d'erreurs détectées au-delà duquel la simulation s'arrête.

ii. `maxNumBits` : nombre de bits envoyés au-delà duquel la simulation s'arrête.

Pour ne pas tenir compte d'un de ces critères d'arrêt, on fixe la valeur correspondante à `Inf`. Pour que la source génère sur une simulation un nombre M de messages de longueur N envoyés, on donne à `maxNumBits` la valeur M×N.

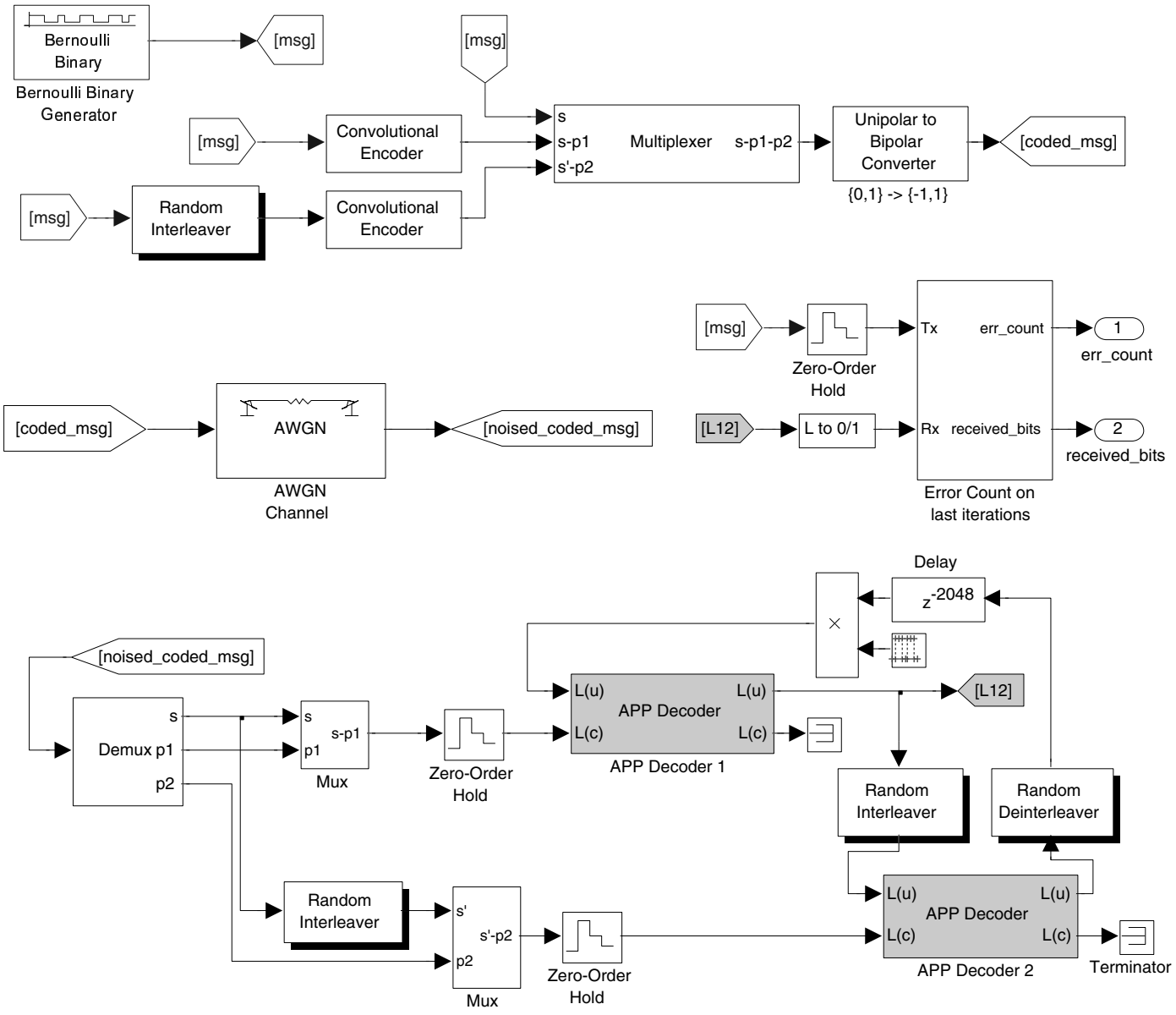
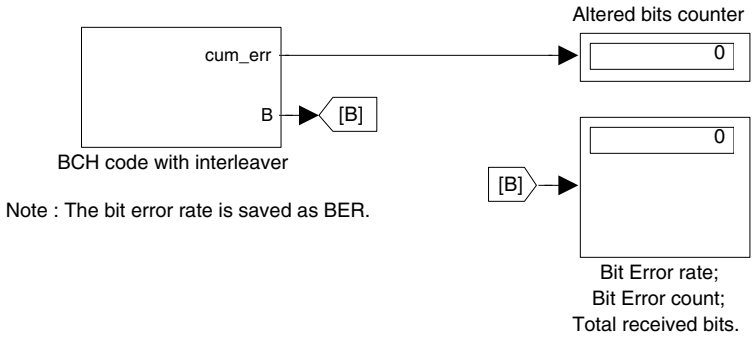


FIG. 2 – Le bloc Turbo Code contient l'encodeur (haut), le canal (milieu gauche), le décodeur (bas), et le comparateur entre messages transmis et reçus (milieu droit). Les flèches [msg], [L12],... permettent de propager les signaux correspondants. Les blocs "Zero-Order Hold" bloquent l'émission d'un nouveau message tant que le nombre d'itérations prescrit n'est pas réalisé au niveau du décodeur.

FIG. 3 – Le modèle `bch_interleaver_markov` calcule des taux d'erreur par bit. Le canal introduit des erreurs en rafale. Le nombre total de bits altérés par le canal au cours de la simulation est affiché par " Altered bits counter".



6 Description du modèle bch_interleaver_markov

Le modèle `bch_interleaver_markov` (figure 3) permet de mettre en évidence l'intérêt d'une stratégie d'entrelacement lors de l'encodage afin de lutter contre des erreurs en rafale. Cette stratégie est décrite en commentaire de la figure 4.
Pour modéliser un canal introduisant des erreurs en rafale, on utilise une chaîne de Markov de mémoire 1, à deux états 0/1. La probabilité d'introduire une erreur sur le symbole i sera plus grande si une erreur a été introduite sur le symbole $i-1$. Du point de vue du canal, introduire une erreur sur un bit revient à lui ajouter un 1 ; ceci se traduira par le fait que la chaîne de Markov se trouve dans l'état 1. La matrice de transition de la chaîne de Markov, que l'on appelera matrice de transition du bruit du canal, s'écrira

$$\mathbf{\Pi} = \begin{bmatrix} p_0 & 1-p_0 \\ p_1 & 1-p_1 \end{bmatrix}$$

avec p_0 la probabilité d'émettre un 0 (pas d'erreur) sachant que l'émission précédente était 0 (pas d'erreur), et p_1 la probabilité d'émettre un 0 (pas d'erreur) sachant que l'émission précédente était 1 (erreur).

Les paramètres du modèle sont modifiés en double-cliquant sur le bloc "BCH code with interleaver" (figure 3) :

i. `nB` est le nombre de fragments (ou blocs) en lequel le message est découpé (nB correspond à j dans le cours).

ii. `K` est la longueur en bits de chaque fragment. Le message a donc une longueur totale de `nB`×`K` bits.

iii. `p_ini` donne la probabilité initiale pour le canal d'émettre un premier symbole 0 (pas d'erreur).

iv. `p0` donne la probabilité p_0 de la matrice de transition $\mathbf{\Pi}$.

v. `p1` donne la probabilité p_1 de la matrice de transition $\mathbf{\Pi}$.

vi. `seed` fixe l'état initial du générateur aléatoire du canal.

vii. `maxNumBits` fixe le nombre de bits que la source doit générer avant que la simulation ne stoppe.

A l'issue de la simulation, le modèle exporte le signal de bruit généré par le canal dans la variable `channelnoise`.

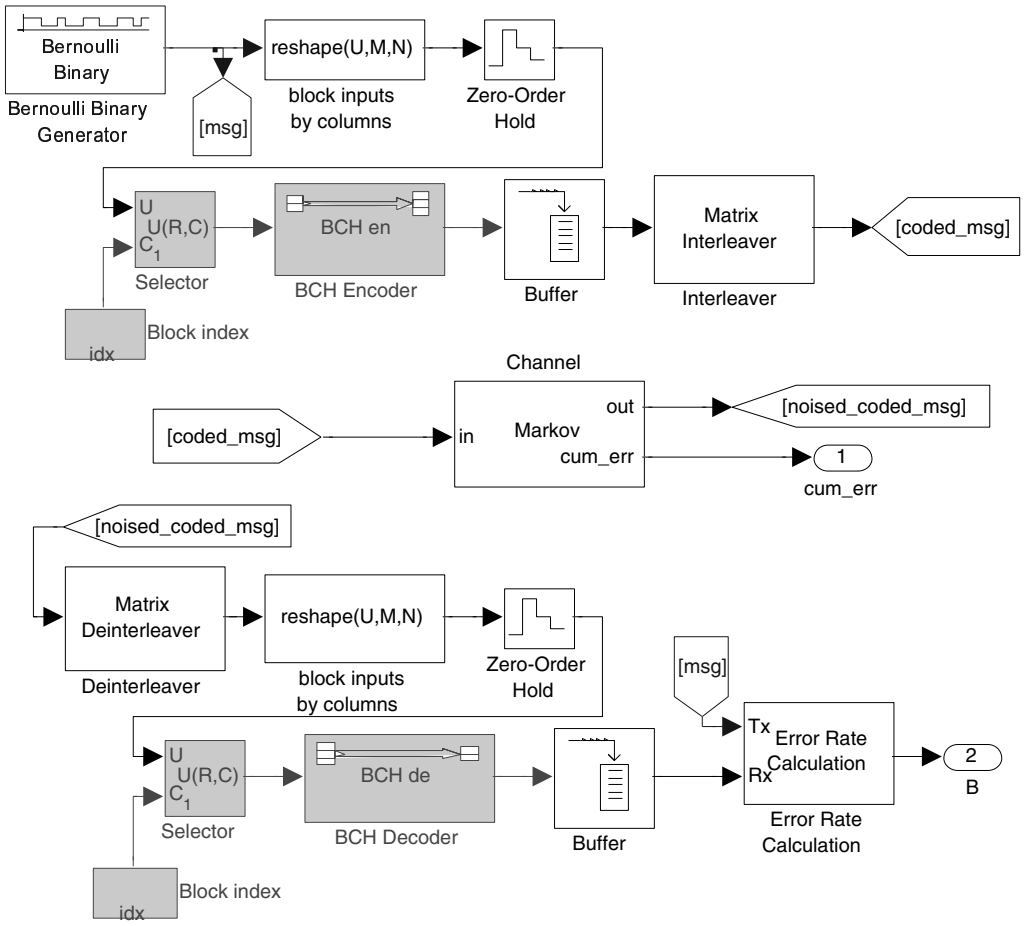


FIG. 4 – Le bloc “BCH code with interleaver” contient l’encodeur (haut), l’ajout d’un bruit binaire à mémoire, (milieu), le décodeur (bas). Les blocs “Reshape” décomposent le long message initial en fragments plus courts, qui sont successivement traités par un même encodeur BCH. Le bloc “idx” est un compteur qui indique le numéro du fragment en cours de traitement. Les fragments codés s’accumulent dans un buffer. Lorsque le dernier fragment a été encodé, le bloc “Interleaver” prend le premier bit de chaque fragment (dans l’ordre), puis le deuxième (toujours dans l’ordre des fragments), et ainsi de suite jusqu’au dernier. Les blocs “Zero-Order Hold” bloquent la génération d’un nouveau message tant que tous les fragments n’ont pas été traités. En pratique, le codage des fragments s’exécuterait en parallèle ; le traitement séquentiel permet de changer facilement le nombre de fragments sans altérer la structure du modèle Simulink.

Le contenu du bloc “BCH code with interleaver” est commenté à la figure 4. Il peut être affiché en sélectionnant le bloc et en choisissant “Look Under Mask” dans le menu “Edit”.

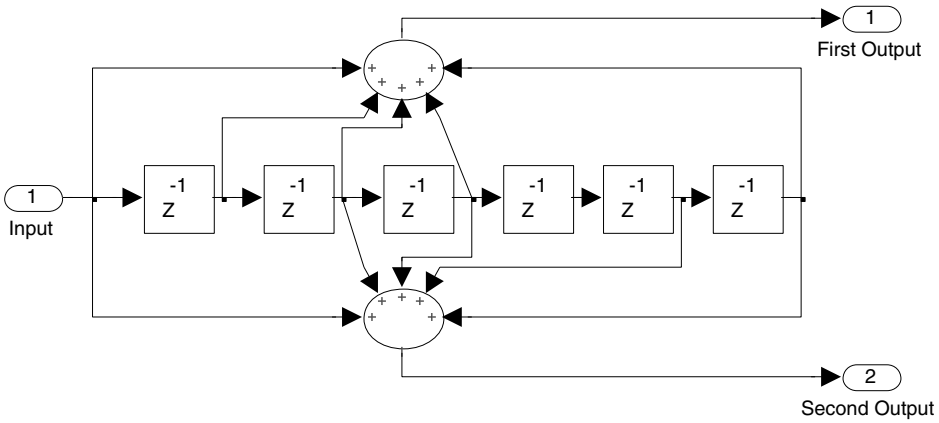
Pour une longueur de mot K, on sait que les possibilités de longueurs N pour le mot codé sont en nombre limité. Par exemple, si K=7, il n'existe de code BCH que pour N=15 et N=63. Si K=12 il n'existe aucun N (donc pas de possibilité de codage BCH). De plus, à un code (N,K) est associé une capacité de détection d'erreur en rafale, et une capacité de correction d'erreur en rafale.

Lorsque l'utilisateur modifie le paramètre `N`, le bloc “BCH code with interleaver” appelle une fonction `checkKN` (fournie). Le bloc peut ainsi afficher une longueur `N` et une capacité de correction d'erreur `errCorrCap`. A noter que `[N,errCorrCap] = checkKN(K)` ne fait que parcourir une liste[1] de triplets `(N, K ,errCorrCap)`, et sélectionne le triplet tel que `N` est le plus élevé mais inférieur ou égal à 127 (pour limiter la complexité du code), s'il y a deux `N` pour un `K` donné.

Remarque. Si vous pilotez Simulink par script (section 8, *non* nécessaire à la réalisation du travail), et que vous modifiez `K`, vous devrez vous-même modifier la valeur de `N` (directement spécifiée ou obtenue en appelant `checkKN`).

7 Définir un encodeur convolutionnel dans Matlab

L'aide de Matlab détaille la procédure à suivre pour définir la structure d'un encodeur tel que



au moyen d'un appel à la fonction `poly2trellis` :

```
t6 = poly2trellis(6+1,[171 133]).
```

Le lecteur intéressé peut se reporter à la section “Communication Toolbox/Error-Control Coding/Polynomial Description of a Convolutional Encoder”ainsi qu'à la section “Communication Blockset/Getting started/Building Models/Building a Convolutional Code Model/Blocks in the Models” de l'aide Matlab.

Pour la réalisation de ce travail, les schémas des encodeurs ainsi que les commandes `poly2trellis` correspondantes sont donnés. Si vous désirez tester d'autres schémas, vous pouvez lire les sections d'aide signalées.

[1] N'hésitez pas à parcourir par vous-même cette liste en ouvrant checkKN.m.

8 Pilotage de Simulink par script

Pour lancer une simulation, le plus simple est de régler les paramètres des blocs, puis de cliquer sur le bouton Play (▶) situé dans la barre d'outils de la fenêtre du modèle Simulink.
A l'issue d'une simulation, le taux d'erreur est placé dans une variable BER accessible dans le workspace de Matlab.
Mais il devient vite laborieux de modifier les paramètres des simulations au travers de l'interface graphique de Simulink lorsque plusieurs simulations successives doivent être lancées.

Pour ceux qui voudraient piloter les modèles Simulink au moyen d'un script Matlab, voici quelques recettes utiles.

Important : le pilotage par script n'est pas indispensable à la réalisation du travail.

- Pour ouvrir un modèle Simulink :
  i. En supposant que le fichier Simulink est `mysyst.mdl`, exécuter `open mysyst`.
- Pour modifier un paramètre :
  i. Ouvrir le modèle Simulink.
  ii. Cliquer sur le bloc dont on veut modifier un paramètre.
  iii. Dans la fenêtre de commande, exécuter `block_name = gcb` qui a pour effet d'attribuer à la variable `block_name` le chemin d'accès du dernier bloc cliqué. Prendre note de la chaîne de caractère `block_name` pour usage ultérieur. Dans la suite, on suppose qu'elle vaut `mysyst/myblock`.
  iv. A supposer que l'on veuille attribuer au paramètre `N` du bloc la valeur `16`, exécuter `set_param(block_name, 'N', num2str(16))`.
- Pour lancer une simulation :
  i. Exécuter `sim('mysyst')`.

Dans le script Matlab,

```
open mysyst
set_param('mysyst/myblock', 'N', num2str(16))
sim('mysyst')
```

est la séquence complète.

A toute fin utile, signalons aussi que `old_N = get_param('mysyst/myblock', 'N');` permet de sauver dans `old_N` la valeur de `N`.

Bon travail.

□