UNIVERSITÉ DE LIÈGE
FACULTÉ DES SCIENCES APPLIQUÉES

THÉORIE DE L'INFORMATION ET DU CODAGE

NOTES RELATIVES AUX TRAVAUX PRATIQUES

Année académique 2005-2006

Table des matières

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 2005-2006, le travail porte sur l'étude du codage de canaux 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

Trois 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 7. 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.

1.2 Organisation

- Le travail est réalisé par groupe de deux étudiants.
- Une fois constitué, le groupe envoie 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. L'énoncé (c'est-à-dire ce document) et les modèles Simulink nécessaires à la réalisation du travail sont alors 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 **6 février 2006**.

2 Comparaison d'algorithmes de décodage

2.1 Description du modèle conv_bsc

Le modèle `conv_bsc` (figure 1) compare les performances de décodage par les algorithmes BCJR et Viterbi. Un message est codé par un encodeur convolutionnel, transite par un canal symétrique binaire, puis est décodé en parallèle par les deux algorithmes. Les messages décodés sont comparés au message de départ. Le modèle calcule pour chaque algorithme le taux d'erreur par bit (BER : bit error rate). Le BER est le rapport du nombre de bits erronés (à l'issue du décodage) au nombre de bits de envoyés (avant encodage). Le modèle calcule aussi le taux d'erreur par message. Un message de longueur N est erroné si un nombre quelconque de ses N bits est erroné. Le taux d'erreur par message (désigné par MER dans ce texte) est le rapport du nombre de messages erronés reçus au nombre de message envoyés sur le canal.

La figure 1 montre comment `conv_bsc.mdl` se présente lorsqu'on l'ouvre dans Matlab. Lorsqu'on double-clique sur le bloc "BCJR vs Viterbi", on peut régler dans une fenêtre de dialogue les paramètres de simulation suivants.

i. `N` fixe la longueur (en bits) du message généré par la source. Pour chaque nouveau message, le vecteur d'état de l'encodeur convolutionnel est remis à 0.

ii. `p` est la probabilité p de transmettre un bit erroné par le canal symétrique binaire. La matrice de transition du canal s'écrit donc (cf section 9.1.3.4 du cours)

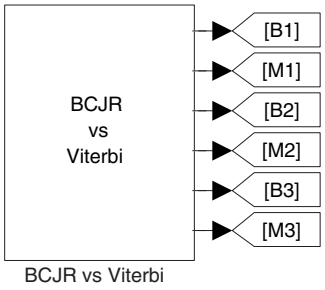
$$\Pi = \begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix}.$$

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

iv. `maxNumMsg` : nombre de messages envoyés au-delà duquel la simulation s'arrête. Pour M messages de longueur N, la simulation s'arrête donc dès que la source a généré $M \times N$ bits.

La figure 2.1 présente le contenu du bloc "BCJR vs Viterbi", affiché en sélectionnant celui-ci puis en choisissant "Look Under Mask" dans le menu "Edit".

FIG. 1 – Le modèle `conv_bsc` calcule les taux d'erreur par bit et par message d'un décodage par BCJR ou par Viterbi. Il vérifie aussi les taux d'erreur obtenus sans codage de canal. Le bouton Play (►) auquel le texte fait référence est sur la barre d'outil (sous celle des menus) dans la fenêtre du modèle.

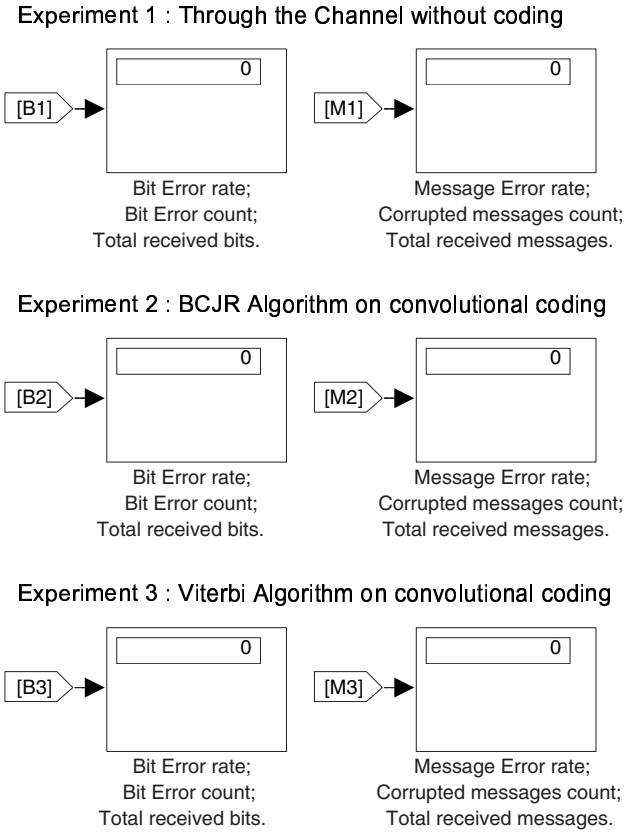


1. Double-click the block "BCJR vs Viterbi"
to see how to change simulation parameters.

2. Right-click the block and choose
"Look under the mask of this subsystem"
in the pop-up menu to see how the system
is decomposed.

3. Start simulation (on this window)
by clicking on the "Play" icon button.

4. At the end of the simulation, the bit
error rates BER and the message error rates
MER are saved in the workspace.



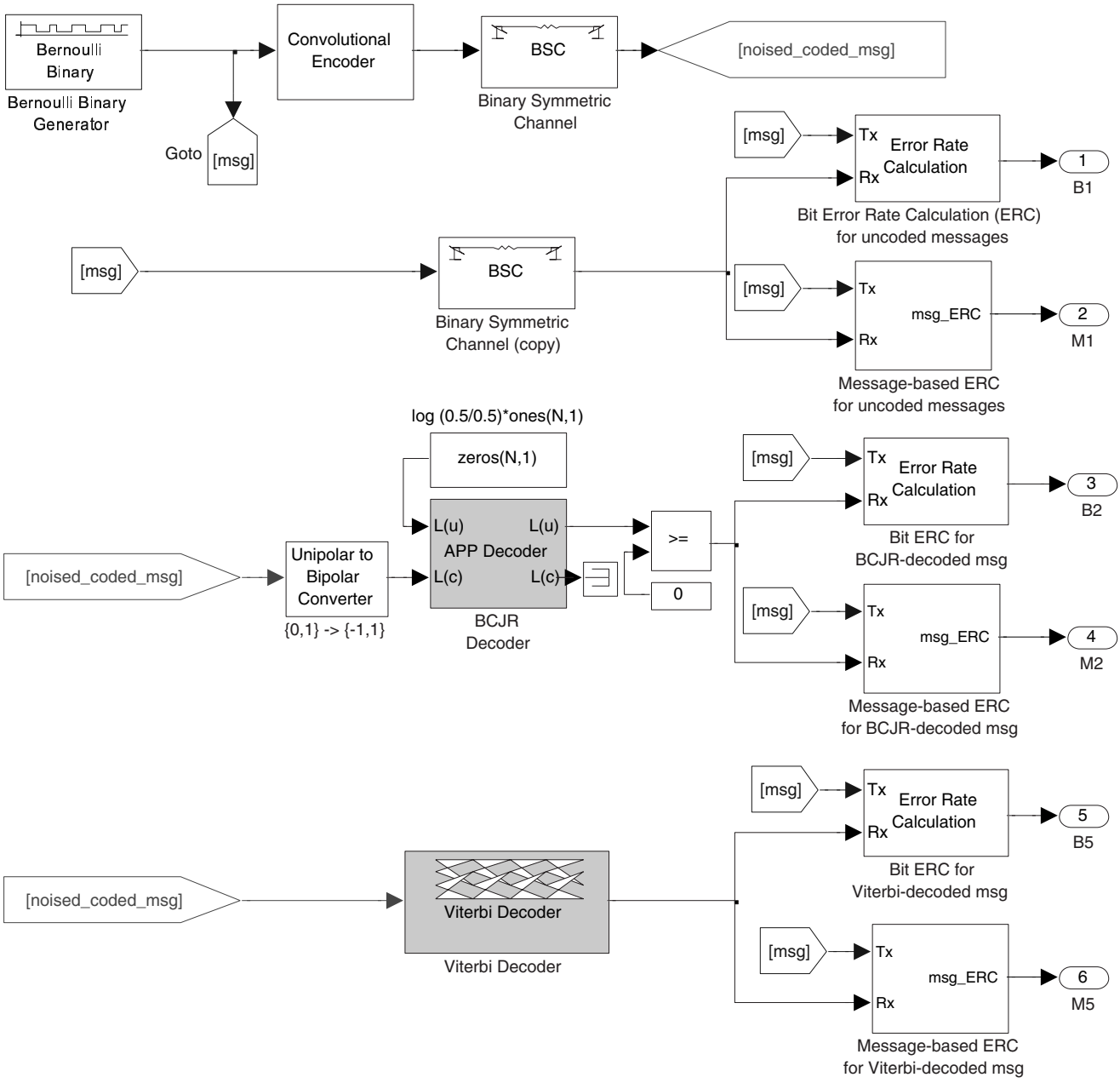
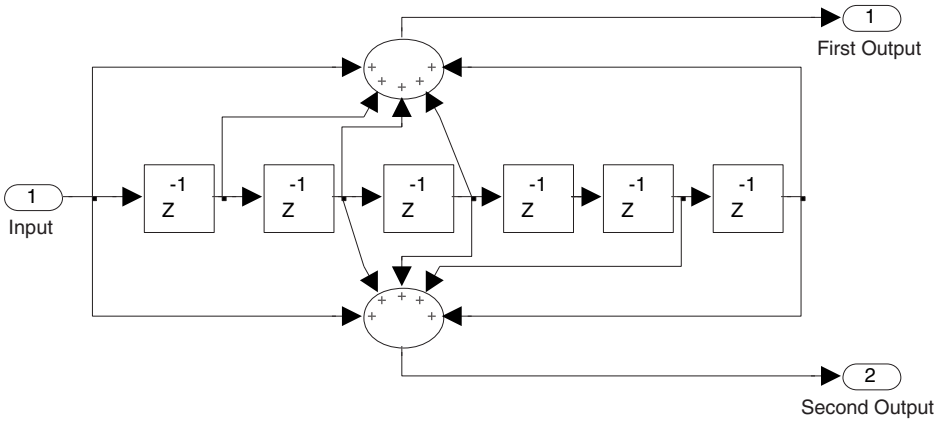


FIG. 2 – Le bloc “BCJR vs Viterbi” implémente trois expériences en parallèle. La première consiste à récupérer le message dès sa sortie de la source, et à l’envoyer tel quel sur le canal symétrique binaire. La deuxième et la troisième partagent le même encodage par un encodeur convolutionnel, suivi du passage dans le canal symétrique binaire (haut de la figure), mais se distinguent par l’algorithme de décodage. Chaque expérience se termine par un calcul des taux d’erreur par bit et par message. Pour le décodage BCJR, associer un bloc de conversion des symboles (0,1) vers (-1,+1) à une entrée nulle pour L(u) dans le bloc “BCJR Decoder” revient à postuler qu’a priori les symboles de sources sont équiprobables (ce qui est le cas). En effet, L(u) est le logarithme du rapport entre les probabilités pour la source d’émettre 1 ou 0, connaissant la sortie de canal L(c). Tout bit de sortie L(u) positif est décodé comme un 1, tout bit négatif comme 0, grâce au comparateur “$>= 0$”.

2.2 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.

Important. 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.

2.3 Questions

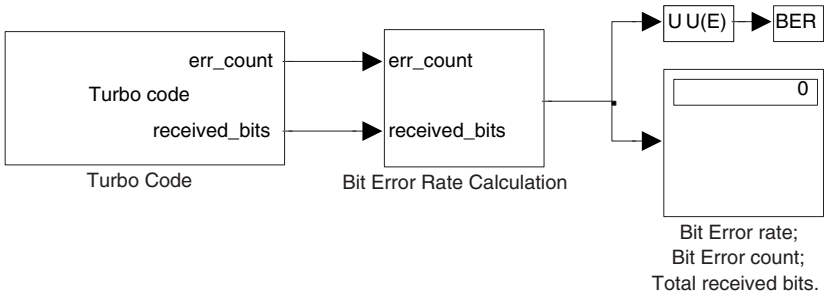
On donne l'encodeur convolutionnel `t6 = poly2trellis(6+1,[171 133])` dont le schéma est à la section 2.2. On demande :

1. De générer une table qui donne, pour les messages de 4 bits, le code correspondant, au moyen de la fonction `convenc` de Matlab. Par exemple, pour générer le code relatif à 0000, il faut entrer

   `c = convenc([0 0 0 0],t6).`

   Le code est-il systématique ?
2. De déduire, à partir de la table, la distance de Hamming du code.
3. De déduire, à partir de la table, le débit du code.
4. De calculer la capacité du canal symétrique binaire pour p = 0.01, 0.05 et 0.1. Jusqu'à quelle valeur de p peut-on espérer transmettre un message avec une probabilité d'erreur arbitrairement petite, si le débit du code est celui de `t6` ?
5. De fixer le paramètre de treillis `t` du bloc "BCJR vs Viterbi" à `poly2trellis(7,[171 133])`, et de lancer des simulations avec N=4 ou N=8, et p = 0.01, 0.05 ou 0.1 (soit en tout 6 simulations). Pour chaque simulation, relever les taux d'erreur par bit et par message des 3 expériences (sans codage de canal, BCJR, Viterbi), soit 6 points par simulation (qui sont chaque fois placés dans les variables `BER` et `MER`). Commenter les résultats obtenus.

FIG. 3 – Le modèle `turbo_wgn` calcule le taux d'erreur par bit d'un système turbo-code.



3 Turbo-codes

3.1 Description du modèle turbo_wgn

Le modèle `turbo_wgn` implémente l'encodeur et le décodeur de turbo-code de la section 15.11 du cours théorique. Le message codé transite sur un canal à bruit blanc gaussien additif. Les paramètres du modèle sont entrés en double-cliquant sur le bloc 'Turbo Code' représenté figure 3.

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`.

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.

Les paramètres ont une portée strictement locale au bloc (et à tout sous-bloc interne).

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 4. Tous les [dés]entrelaceurs utilisent `r0` comme état initial. Tous les [dé]codeurs 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 reçus (Bit Error count) par le nombre total de bits envoyés (Total received bits). En double-cliquant sur ce bloc, on peut régler les paramètres suivants.

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 imposer un nombre M de messages de longueur N envoyés, on donne à `maxNumBits` la valeur $M \times N$.

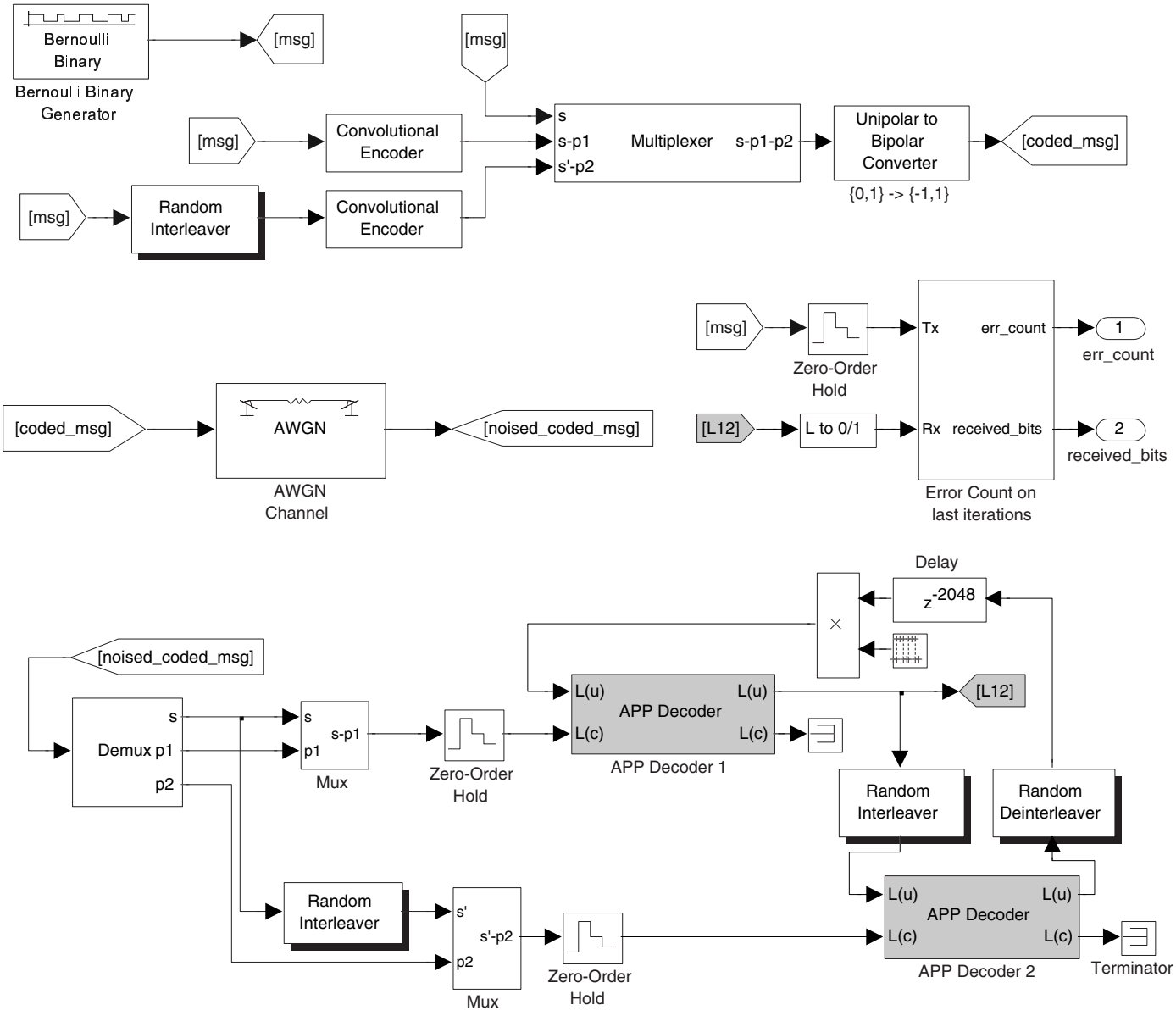
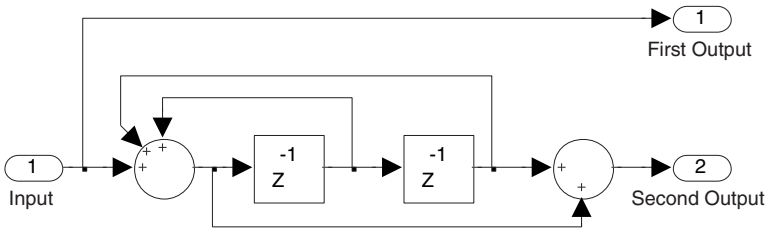


FIG. 4 – Le bloc Turbo Code contient l'encodeur (haut), le canal (milieu gauche), le décodeur (bas), et le comparateur entre messages transmis et reçus (milieu droit). Les flèches [msg], [L12],... permettent de propager les signaux correspondants. Les blocs "Zero-Order Hold" bloquent l'émission d'un nouveau message tant que le nombre d'itérations prescrit n'est pas réalisé au niveau du décodeur.

3.2 Questions

On donne l'encodeur convolutionnel récursif `t2 = poly2trellis(3,[7 5],[7])` :



On demande :

6. De générer la table des codes pour les messages de 4 bits. Vérifier sur cette table que le code est systématique.

7. De calculer la réponse impulsionnelle de l'encodeur grâce à la fonction `convenc`. 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 :

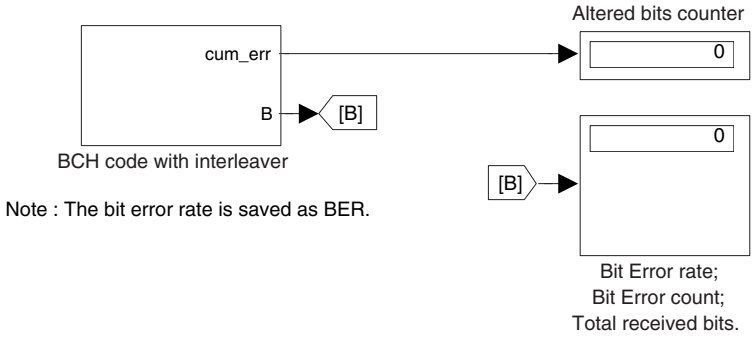
$$h(D) = \frac{1+D+D^2+D^3}{1+D^3} = (1+D+D^2+D^3)(1+D^3+D^6+D^9+\ldots+D^{3k}+\ldots).$$

8. De déduire du schéma de la figure 4 la puissance du signal envoyé sur le canal (AWGN Channel). Sur la figure 4, 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.

9. 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 $\mathcal{C}'$, quelle serait la probabilité d'erreur p de la matrice de transition du canal $\mathcal{C}'$?
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.

10. 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. Commenter les résultats obtenus.

FIG. 5 – 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".



4 Correction d'erreurs en rafale

4.1 Description du modèle bch_interleaver_markov

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.

Le modèle se présente comme sur la figure 5.

Pour modéliser un canal introduisant des erreurs en rafale, on utilise une chaîne de Markov de mémoire 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. La matrice de transition du bruit du canal s'écrira donc

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

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 modèles sont modifiés en double-cliquant sur le bloc "BCH code with interleaver" (figure 5) :

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`$\times$`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 Π.
v. `p1` donne la probabilité p_1 de la matrice de transition Π.
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`.

Le contenu du bloc "BCH code with interleaver" est commenté à la figure 6. Il peut être affiché en sélectionnant le bloc et en choisissant "Look Under Mask" dans le menu "Edit".

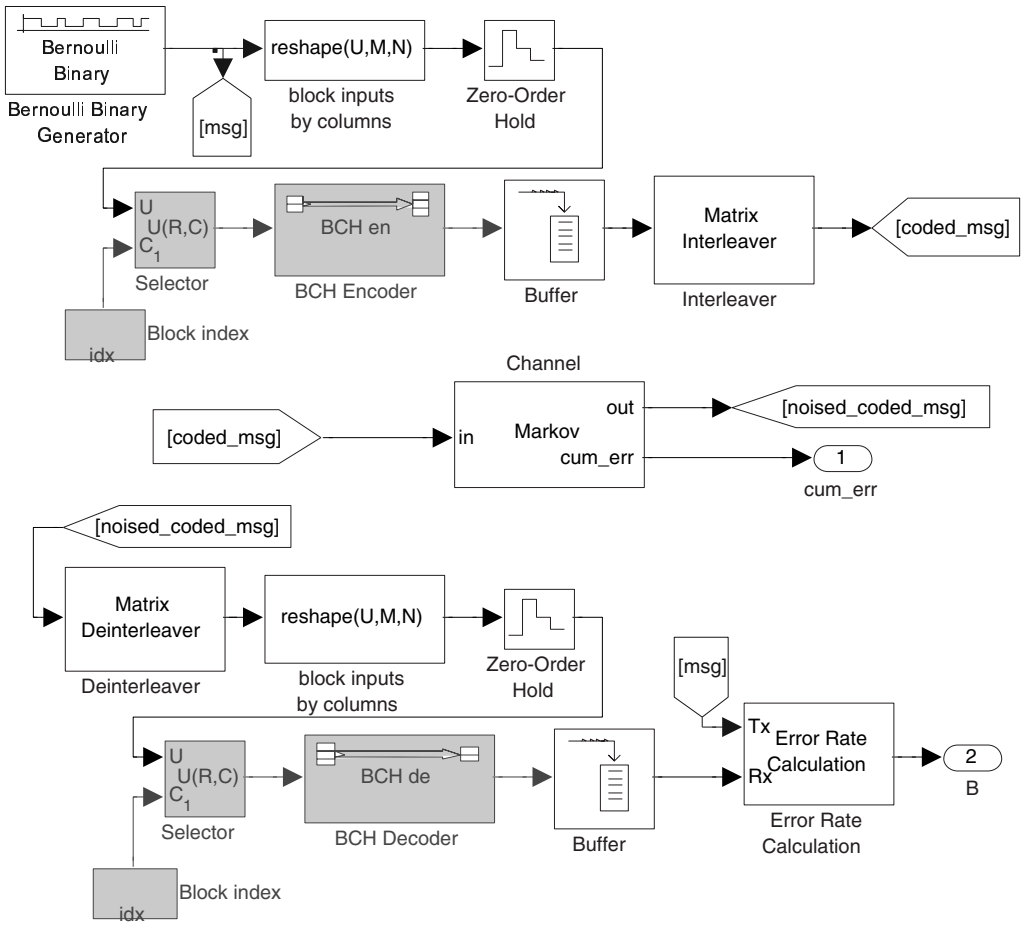


FIG. 6 – Le bloc “BCH code with interleaver” contient l’encodeur (haut), l’ajout d’un bruit binaire à mémoire, (milieu), le décodeur (bas). Les blocs “Reshape” décomposent le long message initial en fragments plus courts, qui sont successivement traités par un même encodeur BCH. Le bloc “idx” est un 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.

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 5, *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`).

4.2 Questions

On donne un canal (binaire) dont la matrice de transition du bruit additif binaire est

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

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 :

11. De calculer la distribution stationnaire π_0 de la matrice de transition du bruit Π.

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

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

13. 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]`.

14. De comparer la distribution des erreurs de longueur 1,2,... du canal, dont la matrice de transition du bruit est Π, et initialisé selon sa distribution stationnaire $\pi_0 = [1-p, p]$, 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).

15. 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).

16. De commenter les résultats de simulation obtenus.

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

5 Annexe : Pilotage de Simulink par script

Il est parfois laborieux d'avoir à modifier les paramètres d'une simulation au travers de l'interface graphique de Simulink. Pour ceux qui voudraient piloter les modèles Simulink au moyen d'un script Matlab, voici la marche à suivre.

Important : le pilotage par script n'est pas nécessaire 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.

□