TP: Représentation des signaux binaires.

Objectifs : Ce TP est relatif aux différentes méthodes de codage d'une information binaire, et à la transmission en bande de base de cette information. Les grandes lignes de ce TP sont l'étude des

- méthodes de codage en ligne qui sont couramment utilisées en communications numériques ;
- densités spectrales de puissance associées à ces divers codes en ligne ;
- causes de distorsion du signal dues à un canal de communication numérique ;
- effets d'interférences entre symboles (IES) et de bruit de canal à partir du diagramme de l'oeil.

1 Simulation d'un message binaire - Codage en ligne

On s'interessera tout au long de ce TP à des informations binaires.

a) Ecrire une fonction Matlab *binary* permettant de générer aléatoirement une séquence binaire de n bits. Compléter cette fonction par une autre fonction *waveplot* permettant la représentation graphique de ce message binaire généré avec un débit binaire R_b (**rand , plot**). (On pourra prendre comme valeur par défaut pour la suite $Rb = 1\text{k}\,\text{b/s}$).

On s'intéresse maintenant aux représentations possibles de ce cette séquence binaire. On parle pour cela de codage ligne.

b) Ecrire une fonction *wavegen* qui associe à une séquence binaire b de débit R_b le signal *sig* correspondant codé en ligne avec la fréquence d'échantillonnage f_s pour chacun des codes en ligne suivant :

- Code **Unipolaire NRZ** (Non Return to Zero) :

$$x(t) = \begin{cases} +V\ , & t \in [kT_b, (k+1)T_b]\ , \quad \text{si } \alpha_k = 1 \\ 0\ , & t \in [kT_b, (k+1)T_b]\ , \quad \text{si } \alpha_k = 0 \end{cases}$$

- Code **Polaire NRZ** : c'est la version antipolaire du code unipolaire NRZ :

$$x(t) = \begin{cases} +V\ , & t \in [kT_b, (k+1)T_b]\ , \quad \text{si } \alpha_k = 1 \\ -V\ , & t \in [kT_b, (k+1)T_b]\ , \quad \text{si } \alpha_k = 0 \end{cases}$$

– Code **Unipolaire RZ** :

$$x(t) = \begin{cases} \begin{cases} +V\ , & t \in [kT_b, (k+1/2)T_b] \\ 0\ , & t \in [(k+1/2)T_b, (k+1)T_b] \end{cases} , & \text{si } \alpha_k = 1 \\ 0\ , & \text{si } \alpha_k = 0 \end{cases}$$

– Code **Polaire RZ** (Return to Zero) : ici, le signal "revient" à zero à chaque demi-période :

$$x(t) = \begin{cases} \begin{cases} +V\ , & t \in [kT_b, (k+1/2)T_b] \\ 0\ , & t \in [(k+1/2)T_b, (k+1)T_b] \end{cases} , & \text{si } \alpha_k = 1 \\ \begin{cases} -V\ , & t \in [kT_b, (k+1/2)T_b] \\ 0\ , & t \in [(k+1/2)T_b, (k+1)T_b] \end{cases} , & \text{si } \alpha_k = 0 \end{cases}$$

– Code **Manchester** : c'est un code de transition antipolaire :

$$\begin{aligned} x(t) &= \begin{cases} \text{"front descendant"} & , \text{ si } \alpha_k = 1 \\ \text{"front ascendant"} & , \text{ si } \alpha_k = 0 \end{cases} \\ &= \begin{cases} \begin{cases} +V\ , t \in [kT_b, (k+1/2)T_b] \\ -V\ , t \in [(k+1/2)T_b, (k+1)T_b] \end{cases} & , \text{ si } \alpha_k = 1 \\ \begin{cases} -V\ , t \in [kT_b, (k+1/2)T_b] \\ +V\ , t \in [(k+1/2)T_b, (k+1)T_b] \end{cases} & , \text{ si } \alpha_k = 0 \end{cases} \end{aligned}$$

– Code **AMI** (Alternate Mark Inversion) : ce code est similaire au code Unipolaire NRZ, mais utilisant cette fois un alphabet ternaire $a_n \in \{-V, 0, +V\}$: lorsque $\alpha_k = 0$, $x(t) = 0$ sur $[kT_b, (k+1)T_b]$, tandis que $\alpha_k = 1$ est codé alternativement par $x(t) = V$ et $x(t) = -V$.

Pour générer l'aphabet ternaire β_k, on pourra utiliser la relation de récurrence :

$$\begin{cases} \beta_n = \alpha_n\, s_{n-1} \\ s_n = (1 - 2\alpha_n) s_{n-1}\,, \end{cases}$$

correctement initialisée.

La fonction *wavegen* prendra comme arguments le signal binaire original ainsi qu'une chaîne de caractères "linecode" permettant le choix du codage en sortie.

2 Densité spectrale de puissance (dsp) pour les codes en ligne

On rappelle que, si $a(t)$ est le signal à l'entrée d'un codeur, $x(t)$ le signal en sorti du codeur (i.e. le signal codé en ligne), et $h(t)$ la réponse impulsionnelle du codeur, alors la

densité spectrale de puissance s'exprime par la relation

$$\gamma_x(f) = \gamma_a(f)\,|H(f)|^2\ ,$$

où

$$\gamma_a(f) = \frac{\sigma_a^2}{T} + \frac{2}{T}\sum_{k=1}^{\infty}\Gamma_a(k)\,\cos(2\pi k f T) + \frac{m_a^2}{T^2}\sum_{k=1}^{\infty}\delta(f-\frac{k}{T})$$

avec,

- m_a est la moyenne des a_k
- σ_a^2 est la variance des a_k
- $\Gamma_a(k)$ est la fonction d'auto-corrélation centrée des a_k.

Pour chaque code en ligne utilisé dans la partie précédente, exprimer la densité spectrale de puissance théorique, puis, avec Matlab, générer une séquence binaire b, et calculer numériquement cette densité spectrale de puissance. On pourra utiliser les fonctions matlab *psd* et *pwelch*.

On note :

- F_{pi} : le $i^{\text{ème}}$ maximum du spectre,
- F_{ni} : le $i^{\text{ème}}$ zéro du spectre,
- B_T la bande passante du signal.

Remplir le tableau suivant :

$R_b =$ kb.s^{-1}	F_{p1}	F_{p2}	F_{n1}	F_{n2}	B_T
Unipolaire NRZ					
Polaire NRZ					
Unipolaire RZ					
Bipolaire RZ					
Manchester					

Faire les commentaires qui s'imposent pour les caractéristiques spectrales de chaque code. En déduire l'intérêt, avantages et inconvénients, de chaque code.

Indication pour le code AMI. L'expression de la densité spectrale de puissance pour le code AMI est un peu plus complexe à obtenir que celle des autres codes.

On pourra, pour effectuer le calcul, montrer (et utiliser) les relations ($E\{\dot{}\}$ désignant l'espérance) :

$$\begin{aligned}
&E\{\alpha_n\} = 1/2 \Longrightarrow E\{1-2\alpha_n\} = 0\\
&E\{\alpha_n^2\} = 1/2 \Longrightarrow E\{(1-2\alpha_n)^2\} = 1\\
&E\{\alpha_n\alpha_k\} = 1/4\ ,\ \text{si } n \neq k\,.
\end{aligned}$$

On obtient alors, en utilisant la relation de récurrence

$$\beta_n = \alpha_n(1-2\alpha_{n-1})\cdots(1-2\alpha_{n-m})\cdots\ ,$$

les corrélations :

$$\Gamma_\alpha(k) = \begin{cases} 1/2\ , & \text{si } k = 0 \\ -1/4\ , & \text{si } k = \pm 1 \\ 0\ , & \text{si } |k| \geq 1 \end{cases}$$

3 Simulation d'un canal de transmission

La fonction matlab *Canal.m* donnée à la fin de cet énoncé permet de simuler un canal de transmission.

Syntaxe de la fonction : sig_{out} = Canal(sig_{in}, gain, noise, bandwidth), où,

- sig_{in} est le signal à l'entrée du canal,
- *gain* est le gain (atténuation) du canal,
- *noise* représente la puissance du bruit additif (bruit blanc Gaussien),
- *bandwidth* est la bande passante du canal,
- et bien sûr, sig_{out} est le signal obtenu à la sortie du canal.

Etudier pour chaque code en ligne vu précédemment l'influence des caractéristiques du canal de transmission sur le signal transmis.

Commenter les effets du bruit et de la bande passante du canal sur le signal. Expliquer ensuite ces phénomènes.

4 Diagramme de l'oeil

Les conséquences du bruit et du filtrage peuvent être mises en évidence en observant le signal de sortie avec la représentation dite du diagramme de "l'oeil". Ce diagramme est obtenu avec de multiples balayages et est synchronisé avec le signal d'horloge (le signal d'horloge provient soit d'une source exterieure, soit du signal de sortie du canal ; il est envoyé à l'entrée "external trigger" d'un oscilloscope à mémoire). La largeur de balayage est plus grande que la période de la séquence de données binaires $T_b = 1/R_b$. Dans la simulation, le diagramme de l'oeil est obtenu avec un balayage d largeur égale à $2T_b$.

La fonction matlab *eyediagram* permet de tracer le diagramme de l'oeil d'un signal.

Pour chaque code en ligne utilisé jusqu'ici, représenter le diagramme de l'oeil du signal original, puis du signal transmis par un canal de bande passante B_w et de bruit additif de puissance P_b.

Commenter les diagrammes pour chaque code.

On considére les caractéristiques du canal : $P_b = 0.01$ W et $B_w = 1$ kHz.

Pour quel code en ligne, le diagramme de l'oeil est-il "acceptable". Commenter et expliquer.

```
function [out, t] = Canal(x,gain,noise_power,bandwidth,fs)

f=bandwidth;
if (nargin == 4), fs=10E3;end

%-------------------------------------------------------------------------
% Default values for parameters used in filter design
%-------------------------------------------------------------------------

ripple     = 0.1; % Allowable ripple, in decibels
filt_order = 8; % Filter order
fpoints    = 256; % Used to display channel response
Ts         = 1/fs; % Sampling period;
no_sample  = length(x); % Number of input samples;
time_t     = [0:(no_sample-1)]*Ts; % instances where the samples defined;
if (gain < 0), error('GAIN must be a non-negative parameter.'); end

%-------------------------------------------------------------------------
% set the cut-off frequencies, and check consistency
%-------------------------------------------------------------------------

if (length(f) == 1)  % Low-pass type channel
   fc = f;
   passband = [fc]/(fs/2);
   if ( fc >= fs/2 )
      fprintf('Cut-off frequency must be less than %6.2f [kHz].\n', fs/2000 );
      error('');
   end
elseif (length(f) == 2)   % Band-pass type channel
   fl = min(f); fu = max(f);
   passband = [fl, fu]/(fs/2);
   if ( fu >= fs/2 )
      fprintf('Upper cut-off frequency must be less ...
         than %6.2f [kHz].\n', fs/2000 );
      error('');
   end
end

%-------------------------------------------------------------------------
% Determine & modify filter coefficients
%-------------------------------------------------------------------------
Td      = 0; % To simplify let delay time Td=0, you can modify by adding
% Td as the last argument to CHANNEL function.
```

```
[Bc,Ac] = cheby1(filt_order, ripple, passband);
delay   = fix(Td/Ts); % required time delay
Bc      = [zeros(1,delay) Bc]; % modify numerator for "delay"
Bc      = sqrt(gain) * Bc; % modify numerator for "gain"
x       = [x(:)' zeros(1,delay)];

%-------------------------------------------------------------------------
% Output routines
% ===============
% If no output arguments, plot the magnitude response, otherwise
% compute the channel output vector "y"
%-------------------------------------------------------------------------

if (nargout == 0)

   f     = fs/(2*fpoints) * (0:fpoints-1);
   Hc    = freqz(Bc,Ac,fpoints);
   mag   = abs(Hc);
   phase = angle(Hc);

   subplot(121), plot( f, 20*log10(mag) ),
   title('Magnitude response'),
   xlabel('Frequency (Hz)'),ylabel('Mag [dB]'), grid on;

   subplot(122), plot( f, unwrap(phase) ),
   title('Phase response'),
   xlabel('Frequency (Hz)'),ylabel('Phase'), grid on;

else

   y   = filter(Bc,Ac,x);
   r   = randn(size(y));
   out = y + sqrt(noise_power)*r;
   if( nargout == 2), t = time_t; end

end
```