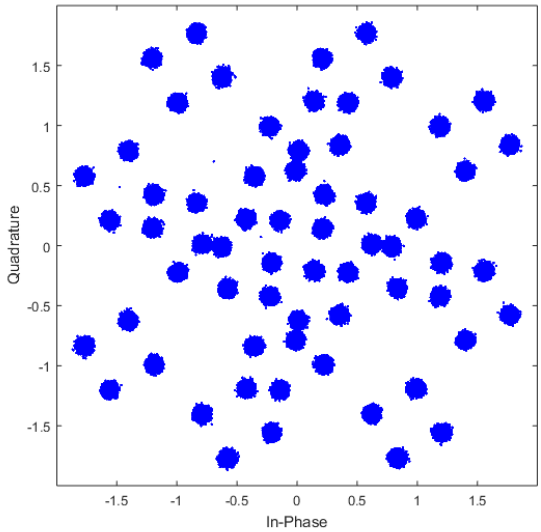
Département Sciences du Numérique

Parcours Télécommunications & **Réseaux**

Travaux pratiques

Egalisation dans le domaine temporel

Auteur : C. Poulliat



Version 1.0 du
2 décembre 2020

1 Introduction

Ce TP sera effectué à l'aide du logiciel Matlab. Les thèmes abordés sont :

- Egalisation temporelle ZF,
- Egalisation temporelle MMSE,
- Egalisation non linéaire basée treillis.

2 Transmission sur canal sélectif en fréquence : égalisation temporelle

On considère dans un premier la transmission sur un canal sélectif en fréquence. On considérera le modèle de transmission bande de base. On utilisera à l'émetteur une modulation de type MAQ à M états avec mapping de type Gray. Le nombre d'états considérés pourra être variable de $M = 4$ à $M = 64$ états.

Pour la génération de symboles QAM pour $M > 4$, on pourra utiliser $\mathrm{qammod}(.)$ donnée par

```
function y = qammod(x, M, varargin)
%QAMMOD Quadrature amplitude modulation
%
%   Y = QAMMOD(X,M) outputs the complex envelope of the modulation of the
%   message signal X using quadrature amplitude modulation. M is the
%   alphabet size and must be an integer power of two. The message signal X
%   must consist of integers between 0 and M-1. X can be a scalar, a
%   vector, a matrix or an array with 3 dimensions.
%
%   Y = QAMMOD(X,M,SYMBOL_ORDER) specifies how the function maps an integer
%   or group of log2(M) input bits to the corresponding symbol. If
%   SYMBOL_ORDER is set to 'gray', then the function uses a Gray-coded
%   ordering. If SYMBOL_ORDER is set to 'bin', then the function uses a
%   natural binary-coded ordering. If SYMBOL_ORDER is an integer-valued
%   vector with M elements, the function uses the ordering specified by
%   this vector. This vector must have unique elements in the range [0,
%   M-1]. The first element of this vector corresponds to the top-leftmost
%   point of the constellation, with subsequent elements running down
%   column-wise, from left to right. The last element corresponds to the
%   bottom-rightmost point. The default value is 'gray'.
%
%   Y = QAMMOD(X,M,...,Name,Value) specifies additional name-value pair
%   arguments described below:
%
%   'InputType'          One of the strings: 'integer', or 'bit'. 'integer'
%                        indicates that the message signal is integer
%                        valued between 0 and M-1. 'bit' indicates that the
%                        message signal is binary (0 or 1). In this case,
%                        the number of rows (dimension 1) must be an
%                        integer multiple of log2(M). A group of log2(M)
%                        bits are mapped onto a symbol, with the first bit
%                        representing the MSB and the last bit representing
%                        the LSB. The default value is 'integer'.
%
%   'UnitAveragePower'   A logical scalar value. If true, the QAM
%                        constellation is scaled to average power of 1. If
%                        false, the QAM constellation with minimum distance
%                        of 2 between constellation points is used. The
%                        default value is false.
%
%   'OutputDataType'     Output fixed-point type as a signed, unscaled
%                        numerictype object in MATLAB simulation, and as a
%                        signed, scaled numerictype object during C code or
%                        MEX generation. When this argument is not
%                        specified, if the input datatype is double or
%                        built-in integers, the output datatype is double;
%                        if the input datatype is single, the output
%                        datatype is single. When the input is fixed-point,
%                        this parameter must be specified.
```

```
%
%   'PlotConstellation'  A logical scalar value. If true, the QAM
%                        constellation is plotted. The default value is
%                        false. The input X is processed and the modulated
%                        signal is returned in output Y.
%
% Example 1:
%    % 32-QAM modulation. Default: Integer input, Gray coding, minimum
%    % distance of 2 between constellation points
%    x = (0:31)';
%    y = qammod(x, 32);
%
% Example 2:
%    % 16-QAM modulation, with LTE specific symbol mapping and constellation
%    % scaled to average power of 1. Default: Integer input
%    x = randi([0, 15], 20, 4, 2);
%    lteSymMap = [11 10 14 15 9 8 12 13 1 0 4 5 3 2 6 7];
%    y = qammod(x, 16, lteSymMap, 'UnitAveragePower', true);
%
% Example 3:
%    % 64-QAM modulation with binary mapping, bit input and signed
%    % fixed-point output data type with 16 bits of word length and 10
%    % bits of fraction length. Default: minimum distance of 2 between
%    % constellation points
%    x = randi([0, 1], 10*log2(64), 3);
%    y = qammod(x, 64, 'bin', 'InputType', 'bit', 'OutputDataType', numerictype↩
    (1,16,10));
%
% Example 4:
%    % Visualize the constellation for 16-QAM modulation, with gray
%    % mapping, bit input and constellation scaled to average power of 1.
%    x = randi([0, 1], log2(16), 1);
%    y = qammod(x, 16, 'InputType', 'bit', 'UnitAveragePower', true, '↩
    PlotConstellation', true);
%
%   See also QAMDEMOD, MODNORM.
```

2.1 Canal de transmission

Le signal émis est transmis dans un canal sélectif en fréquence dont on pourra prendre la réponse impulsionnelle du canal équivalent comme suit (canaux "test" de Proakis) :

- $h_A = [0.04, -0.05, 0.07, -0.21, -0.5, 0.72, 0.36, 0, 0.21, 0.03, 0.07]$,
- $h_B = [0.407, 0.815, 0.407]$,
- $h_C = [0.227, 0.46, 0.688, 0.460, 0.227]$,

On étudiera également des canaux de type

$$\overline{h}(z) = 1 - az^{-1}, \text{où } |a| < 1.$$

On ajoutera ensuite un bruit Gaussien à la chaîne de transmission précédente. On pourra utiliser plusieurs puissances de bruit que l'on calculera en fonction de E_s/N_0 (rapport signal sur bruit par symbole codé). On rappelle que la variance du bruit à appliquer sur les voies en phase et quadrature du bruit complexe $n_e(t)$ s'écrit en fonction du E_s/N_0 souhaité de la manière suivante :

$$\sigma_{n_I}^2 = \sigma_{n_Q}^2 = \frac{\sum_n |h(n)|^2 \sigma_s^2}{2E_s/N_0}, \tag{1}$$

où σ_s^2 représente la variance des symboles s_n, $h(n)$ la réponse impulsionnelle du canal discret équivalent et E_s/N_0 le rapport signal à bruit par symbole à l'entrée du récepteur. Le modèle de réception est alors donné par

$$y[n] = h * x[n] + b[n].$$

Avant de considérer les aspects récepteurs, on analysera les signaux reçus :

1. Regarder les symboles reçus sans bruit. Que pouvez vous dire ?
2. Quel est l'influence du bruit ?
3. Regarder les densités spectrales de puissance (DSP) à l'émission et à la réception avec et sans bruit. Justifier les résultats observés.

2.2 Egaliseurs temporels à structure non contrainte

On réalisera en suite une comparaison des performances des égaliseurs de type ZF et MMSE sans contrainte en comparant les réponses impulsionnelles des filtres égaliseurs et leurs performances en terme de taux d'erreur binaire. Quand le canal est d'inverse stable et causal, le calcul de l'égaliseur ZF non contraint peut être réalisé de manière directe par la fonction *filter* de Matlab. Dans le cas contraire et pour le cas MMSE non contraint, il est difficile d'avoir accès à une implémentation aisée avec Matlab qui permet de gérer une réponse impulsionnelle non causale. Nous réaliserons une version approchée en considérant un calcul direct de la réponse impulsionnelle de ces filtres sur un nombre de coefficients suffisamment grand pour considérer cette approximation valide. Pour ce faire, on calculera cette réponse impulsionnelle à l'aide la fonction *ComputeRI* fournie. Pour visualiser le diagramme des pôles et des zéros, on utililisera

```
function [ RI ] = ComputeRI( N, r, p, k )
%Compute an approximation of an infinite impulse response of length 2*N-1.
%Only the causal number of taps is required.
```

Cette fonction calcule directement une approximation à taille finie de la réponse impulsionnelle *stable* du filtre en distinguant les pôles de module supérieur ou inférieur à 1. La reconstruction se base sur la décomposition de la fonction de transfert du filtre non contraint obtenu à partir de la fonction *residuez* de Matlab qui permet de déterminer les éléments de cette décomposition en éléments simples.

```
function [ r, p, k ] = residuez( b, a, t )
%RESIDUEZ Z-transform partial-fraction expansion.
%   [R,P,K] = RESIDUEZ(B,A) finds the residues, poles and direct terms
%   of the partial-fraction expansion of B(z)/A(z),
%
%      B(z)       r(1)                r(n)
%      ---- = ------------ +...  ------------ + k(1) + k(2)z^(-1) ...
%      A(z)   1-p(1)z^(-1)       1-p(n)z^(-1)
%
%   B and A are the numerator and denominator polynomial coefficients,
%   respectively, in ascending powers of z^(-1). R and P are column
%   vectors containing the residues and poles, respectively. K contains
%   the direct terms in a row vector. The number of poles is
%      n = length(A)-1 = length(R) = length(P)
%   The direct term coefficient vector is empty if length(B) < length(A);
%   otherwise,
%      length(K) = length(B)-length(A)+1
%
%   If P(j) = ... = P(j+m-1) is a pole of multiplicity m, then the
%   expansion includes terms of the form
%          R(j)              R(j+1)                      R(j+m-1)
%      -------------- + ------------------   + ... + ------------------
%      1 - P(j)z^(-1)   (1 - P(j)z^(-1))^2            (1 - P(j)z^(-1))^m
%
%   [B,A] = RESIDUEZ(R,P,K) converts the partial-fraction expansion back
%   to B/A form.
%
%   Warning: Numerically, the partial fraction expansion of a ratio of
%   polynomials represents an ill-posed problem. If the denominator
%   polynomial, A(s), is near a polynomial with multiple roots, then small
%   changes in the data, including roundoff errors, can make arbitrarily
%   large changes in the resulting poles and residues. Problem
%   formulations making use of state-space or zero-pole representations
%   are preferable.
%
%   % Example:
%   %   Compute the partial fraction expansion of the following transfer
%   %   function H(z) = (1 + 2z^-1) / (1 - z^-1 + 2z^-2).
%
%   num = [1  1];                % Numerator coefficients
%   den = [1 -1  2];             % Denominator coefficients
%   [r,p] = residuez(num,den)    % H(z) = r(1)/(1-p(1)z^-1) + ...
%                                %        r(2)/(1-p(2)z^-1)
```

Le travail à réaliser est donc le suivant pour différents canaux :

1. Implémenter et évaluer les performances de égaliseurs ZF et MMSE non contraints en terme de taux d'erreurs binaires. La syntaxe est alors suivante pour le cas ZF :

```
Nzf=200;
[r, p, k]=residuez(1, hc);
[w_zfinf]=ComputeRI( Nzf, r, p, k );
```

Dans le cas MMSE, on aura une implémentation du type

```
deltac=zeros(1,2*Lc-1);
  deltac(Lc)=1;
  Nmmse=200;%causal part
  [r, p, k]=residuez(fliplr(conj(hc)),(conv(hc,fliplr(conj(hc)))+(sig2b/sigs2↩
      )*deltac));
  [w_mmseinf]=ComputeRI( Nmmse, r, p, k );
```

2. Comment se comporte le filtre MMSE à faible et fort rapport signal sur bruit ? Est-ce cohérent avec la théorie ?
3. Evaluer le rapport signal signal à bruit après égalisation et le biais pour le cas MMSE. Est-ce cohérent avec ce qui est attendu ?

2.3 Egaliseurs temporels à structure RIF.

Le travail à réaliser est le suivant pour différents canaux :

1. Implémenter et évaluer les performances de égaliseurs ZF et MMSE contraints en terme de taux d'erreurs binaires. On implémentera les versions optimisées en délai que l'on comparera avec des versions non optimisées. Pour l'implémentation, on pourra utiliser la fonction *toeplitz* de Matlab pour générer la matrice de convolution du canal.

```
H=toeplitz([hc(1) zeros(1,Nw-1)]',[hc, zeros(1,Nw-1)]);
```

```
function t = toeplitz(c,r)
%TOEPLITZ Toeplitz matrix.
%   TOEPLITZ(C,R) is a non-symmetric Toeplitz matrix having C as its
%   first column and R as its first row.
%
%   TOEPLITZ(R) is a symmetric Toeplitz matrix for real R.
%   For a complex vector R with a real first element, T = toeplitz(r)
%   returns the Hermitian Toeplitz matrix formed from R. When the
%   first element of R is not real, the resulting matrix is Hermitian
%   off the main diagonal, i.e., T_{i,j} = conj(T_{j,i}) for i ~= j.
```

2. Comment se comporte le filtre MMSE à faible et fort rapport signal sur bruit ? Est-ce cohérent avec la théorie ?
3. Comment se comporte les filtres avec la taille ?
4. Evaluer le rapport signal signal à bruit après égalisation et le biais pour le cas MMSE. Est-ce cohérent avec ce qui est attendu ?

2.4 Egaliseur Maximum de vraisemblance

On réalisera la mise en oeuvre d'une égalisation par l'égaliseur de maximum de vraisemblance qui est réalisé par la mise en oeuvre de l'algorithme de Viterbi sur le treillis associé au canal équivalent bande de base $h[n]$. Pour ce faire on utilisera les fonctions natives fournies par Matlab. Un exemple de code Matlab mettant en oeuvre ces fonctions est alors donné par le texte ci-après pour la génération et les déclarations des paramètres :

```
%Simulation parameters
%

%modulation parameters
M = 4; %Modulation order
Ns = 500;
N= log2(M)*Ns;

%Viterbi decoding parameters
const = qammod((0:M-1)',M); %reference Gray QPSK constellation
tblen = 16; %Traceback depth
nsamp = 1; %Oversampling rate
preamble = [];
postamble =[];

%Channel Parameters
hc = [0.623; 0.489+0.234i; 0.398i; 0.21];

%Message generation
%
bits= randi([0 1],N,1);
s = qammod(bits, M,'InputType','bit','UnitAveragePower',true);

%Channel output
z = filter(hc,1,s);
```

Après addition d'un bruit Gaussien b, on a le code suivant en réception :

```
s_ml = mlseeq(y,hc,const,tblen,'rst',nsamp,[],[]);
```

Le travail à réaliser est le suivant :

1. Implémenter et évaluer les performances de l'égaliseur par Maximum de vraisemblance ;
2. Comparer aux égaliseurs linéaires étudiés précédemment.