BE2 Traitement et protection de l'information

URIEN, PRÉVOST

March 20, 2023

1 Channel coding laboratory

1.1 1. Block codes

1.1.1 1.1. From a generator matrix

We have the following generator matrix:

[273]:
```
%%file genMatrix.m
G = [[1 0 0 1 1 1 0 1 1 1]
     [1 1 1 0 0 0 1 1 1 0]
     [1 1 0 1 1 1 1 0 0 1]];
```

```
Created file '/Users/thomasprevost/github/ProtectInfo/BE2/genMatrix.m'.
```

[274]:
```
genMatrix;
```

1. First, we generate all possible codewords from the generator matrix:

[275]:
```
%%file genCode.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% genCode.m
% Generates all possible codewords of a given generator matrix
% Input: G - generator matrix
% Output: C - codeword matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function C = genCode(G)
    [m,n] = size(G);
    C = zeros(2^n,n);
    fprintf('Generating code matrix of size %d x %d\n',2^n,n);
    for i = 0:2^n-1
        b = de2bi(i,n);
        C(i+1,:) = mod(b*G(1),2);
    end
end
```

```
Created file '/Users/thomasprevost/github/ProtectInfo/BE2/genCode.m'.
```

[276]:
```
C = genCode(G);
```

```
Generating code matrix of size 1024 x 10
```

2. Then, we calculate the minimum distance of the code:

[277]:
```
%%file minDist.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% minDist.m
% Calculates the minimum distance of a given code
% Input: C - codeword matrix
% Output: d - minimum distance
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function d = minDist(C)
    [m,n] = size(C);
    d = n;
    for i = 1:m
        for j = i+1:m
            if sum(rem(C(i,:)+C(j,:),2)) < d
                d = sum(rem(C(i,:)+C(j,:),2));
            end
        end
    end
end
```

```
Created file '/Users/thomasprevost/github/ProtectInfo/BE2/minDist.m'.
```

[278]:
```
d = minDist(C)
```

```
d =

     1
```

3. Finally, we compare the minimum distance of the code with the minimum distance of a parity code with the same parameters. To do so, we first generate the parity code, then we calculate its minimum distance and compare it with the minimum distance of the code.

[279]:
```
%%file parityCode.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% parityCode.m
% Generates a parity code given its parameters
% Input: n - number of bits
%        k - number of information bits
% Output: C - parity code matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [C, G] = parityCode(n,k)
    G = [eye(k) zeros(k,n-k)];
    C = [G rem(G*G',2)];
end
```

```
Created file '/Users/thomasprevost/github/ProtectInfo/BE2/parityCode.m'.
```

[280]:
```
%%file compareToParity.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% compareToParity.m
% Compares the minimum distance of a given code with the minimum distance
% of a parity code with the same parameters
% Input: G - generator matrix
% Output: d - minimum distance
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function d = compareToParity(G)
    [m,n] = size(G);
    C = genCode(G);
    d = minDist(C);
    [Cp,Gp] = parityCode(n,m);
    dp = minDist(Cp);
    fprintf('Minimum distance of the code: %d\n',d);
    fprintf('Minimum distance of the parity code: %d\n',dp);
end
```

```
Created file '/Users/thomasprevost/github/ProtectInfo/BE2/compareToParity.m'.
```

[281]:
```
compareToParity(G);
```

```
Generating code matrix of size 1024 x 10
Minimum distance of the code: 1
Minimum distance of the parity code: 4
```

1.1.2 1.2. Hamming code

1. First, we generate codes of the Hamming coding $C(15, 11)$:

[282]:
```
%%file hammingCode.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% hammingCode.m
% Generates codes of a hamming coding given its parameters
% Input: n - number of bits
%        k - number of information bits
% Output: H - hamming code matrix
%         G - generator matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [H,G] = hammingCode(n,k)
    A = zeros(n-k, k);
    for i = 1:n-k
        A(i,:) = de2bi(i,k);
    end
    G = [eye(k) A.'];
    H = [G rem(G*G',2)];
end
```

```
Created file '/Users/thomasprevost/github/ProtectInfo/BE2/hammingCode.m'.
```

[283]:
```
%%file genHammingCodes.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% genHammingCodes.m
% Generates all possible codes of a hamming coding given its parameters
% Input: n - number of bits
%        k - number of information bits
% Output: C - codeword matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function C = genHammingCodes(n,k)
    [~,G] = hammingCode(n,k);
    C = genCode(G);
end
```

```
Created file '/Users/thomasprevost/github/ProtectInfo/BE2/genHammingCodes.m'.
```

[284]:
```
[~,Gh] = hammingCode(15,11);
Ch = genHammingCodes(15,11);
```

```
Generating code matrix of size 32768 x 15
```

2. Then, we calculate the minimum distance of the Hamming code:

[285]:
```
dh = minDist(Gh)
```

```
dh =

     2
```

NB : the minimum distance of the Hamming code is given to be $d = 3$. We do not have this value, the error being probably due to a mistake in the calculation of the minimum distance.

source_channel_coding

March 20, 2023

1 Source and channel coding

Problème résolu ! Le problème venait du type de " utilisé. Pour définir `X2`, il faut bien utiliser ' et non ".

1.1 Programme II

On convertit ici une entrée au format **`string`** en une entrée au format binaire (soit un tableau de `0` et de `1`).

Entrée: `X2` (string)
Sortie: `X4` (tableau de `0` et de `1`)

[38]:
```
% Programme II
%                ***************
%                *             *
% Message X --> * Source Code * --> X2
%                *             *
%                ***************
% X2 is supposed known

% Warning : The output of LempelZivEnco is a 'string'

% Input : X2
X2 = '01101111010001';

clear X3 X4 Y1 Y2;
X4 = zeros(1, length(X2));

for i = 1:length(X2)
    X4(i) = str2num(X2(i));
end

% X4 is a binary sequence "=" X2
X4
X4(1:10)
X2(1:10)
```

```
X4 =

     0     1     1     0     1     1     1     1     0     1     0     0     0
1


ans =

     0     1     1     0     1     1     1     1     0     1


ans =

    '0110111101'
```

1.2 Channel Code

On calcule ici la distance minimale de Hamming avec la matrice génératrice `G`. Et on vérifie la taille de `X4` pour savoir si on doit ajouter des `0` à la fin de `X4`.

[39]:
```
%                ****************
%                *               *
% Message X --> * Channel Code * --> X5
%                *               *
%                ****************

% Let G be a generator matrix of a block code
G = [1 0 1 0 1 0; 0 1 0 1 0 1]; % repetition
u = [0 1; 1 1; 1 0];
distmin = min(sum(rem(u * G, 2), 2))

% Code X4
% Checking the length of the sequence
L4 = length(X4);
rest = rem(L4, size(G, 1));

if rest
    X4(L4 + 1:L4 + size(G, 1) - rest) = zeros(1, size(G, 1) - rest);
end

size(X4)
```

```
distmin =

     3
```

```
ans =

     1    14
```

Ici, on sépare `X4` en blocs de taille `size(G,1)` (ici vaut 2). Puis on multiplie chaque bloc de code par G, pour les stocker dans `X5`. Enfin, on calcule le code rate entre `X4` et `X5`. On constate que le code rate est égal à la distance minimale de Hamming, calculé précédemment. Cela nous permet d'affirmer que le code est correct.

[40]:
```
% Verification is possible by vec2mat(mat,padded) of Matlab
mat = vec2mat(X4, size(G, 1));
size(mat)
X4(1:10)
mat(1:5, :)

code = rem(mat * G, 2);
X5 = reshape(code', 1, size(code, 1) * size(code, 2));
Rcb = length(X5) / length(X4) % Code rate
```

```
ans =

     7     2


ans =

     0     1     1     0     1     1     1     1     0     1


ans =

     0     1
     1     0
     1     1
     1     1
     0     1


Rcb =

     3
```

1.3 Decode channel

On calcule l'inverse de la matrice `G` et on vérifie que celle-ci est bien son inverse (en multipliant `G` et `Gri`, on obtient bien la matrice identité). Grâce à l'inverse, on peut décoder le message en multipliant `Y1` par `Gri`, une fois `Y1` séparé en blocs de taille `size(G,2)`.

On calcule `Y2` à partir de `Y1`, le message reçu. On constate que `Y2` est égal à `X4`, ce qui prouve que le décodage s'est bien passé.

[41]:
```
%                         ***********
%                         *         *
% Message envoyé X5 --> * Channel * --> Y1
%                         *         *
%                         ***********

% Canal sans bruit ==> Y1 = X5
Y1 = X5;

%                       ******************
%                       *                *
% Message reçu Y1 --> * Decode Channel * --> Y2
%                       *                *
%                       ******************

% Decode Y1
Gri = G' * inv(G * G') % right inverse
G * Gri
Ym = vec2mat(Y1, size(G, 2));
deco = Ym * Gri;

Y2 = reshape(deco', 1, size(deco, 1) * size(deco, 2)); % real vector
Y2 = (Y2 > 0.5); % binary vector after a threshold

BER = sum((Y2 - X4) .^ 2) % Bit Error Rate
```

```
Gri =

    0.3333         0
         0    0.3333
    0.3333         0
         0    0.3333
    0.3333         0
         0    0.3333


ans =

     1     0
```

```
     0     1
```

```
BER =
```

```
     0
```

1.4 Canal II

On introduit des erreurs dans le message `X5` via le canal II. `Y1` contient donc `Nbrerr` erreurs.

[42]:
```
%                       ************
%                       *          *
% Sent Message X5 --> * Canal II * --> Y1
%                       *          *
%                       ************

% Noisy channel ==> Y1 = X5 + noise
e = zeros(size(X5));

for i = 1:length(X5)
    e(i) = (mod(i, size(G, 2)) == 1);
end

Nbrerr = sum(e)
Y1 = rem(X5 + e, 2);
```

```
Nbrerr =
```

```
     7
```

1.5 Decode canal

On calcule `P` car `G` est dans la forme systématique.

[48]:
```
%                           ****************
%                           *              *
% Received Message Y1 --> * Decode Canal * --> Y2
%                           *              *
%                           ****************

% Correction Y1
% Control matrix H
% G is in systematic form
```

```
k = size(G, 1);
n = size(G, 2);
P = zeros(k, n - k);

for i = 1:k
    P(i, :) = G(i, k + 1:end);
end

% H=[P^T I] is a n-k x k matrix
Pt = P';
I = eye(n - k);
H = zeros(n - k, k);

for i = 1:n - k
    H(i, 1:size(Pt, 2)) = Pt(i, 1:end);
    H(i, size(Pt, 2) + 1:end) = I(i, 1:end);
end

rem(G * H', 2) % G x H^T = 0
```

```
Unable to perform assignment because the size of the left side is 1-by-0
and the size of the right side is 1-by-4.
```

[44]:
```
% error check
mat = vec2mat(Y1, size(G, 2));
s = rem(mat * H', 2); % Syndrome
inder = (sum(s, 2) > 0); % error indicator

% error correction
matc = mat; %mat corrected

for i = 1:size(mat, 1)

    if inder(i) % correction
        a = 0;
        b = 0;

        for j = 1:3 %repetition code 2x6
            a = a + mat(i, 2 * j);
            b = b + mat(i, 2 * j - 1);
        end

        for j = 1:3
            matc(i, 2 * j) = ((a / 3) > 0.5);
            matc(i, 2 * j - 1) = ((b / 3) > 0.5);
```

```
        end

    end

end

% Y1 corrected
Y1 = reshape(matc', 1, size(matc, 1) * size(matc, 2));
BER = sum((Y1 - X5) .^ 2) % Bit Error Rate

% Decode Y1
deco = matc * Gri;

Y2 = reshape(deco', 1, size(deco, 1) * size(deco, 2)); % real vector
Y2 = (Y2 > 0.5); % binary vector after a threshold

BER = sum((Y2 - X4) .^ 2) % Bit Error Rate
```

```
Incorrect dimensions for matrix multiplication. Check that the number of
columns in the first matrix matches the number of rows in the second matrix. To
operate on each element of the matrix individually, use TIMES (.*) for
elementwise multiplication.
```

[45]:
```
%                               *****************
%                               *               *
% Channel Decoding Y2 = X2 --> * Decode Source * --> Y3 = X
%                               *               *
%                               *****************

% format string
str = num2str(Y2);

for i = 1:length(Y2)
    Y2s(i) = str(3 * (i - 1) + 1);
end

Y3 = LempelZivDeco(Y2s, Dict2); % Y2s doit être un string

% Attention : La sortie de LempelZivDeco est un 'string'
for i = 1:length(Y3)
    Y4(i) = str2num(Y3(i));
end

BERF = sum((X(1:length(Y3)) - Y4) .^ 2) % Bit Error Rate
```

```
Unrecognized function or variable 'Dict2'.
```