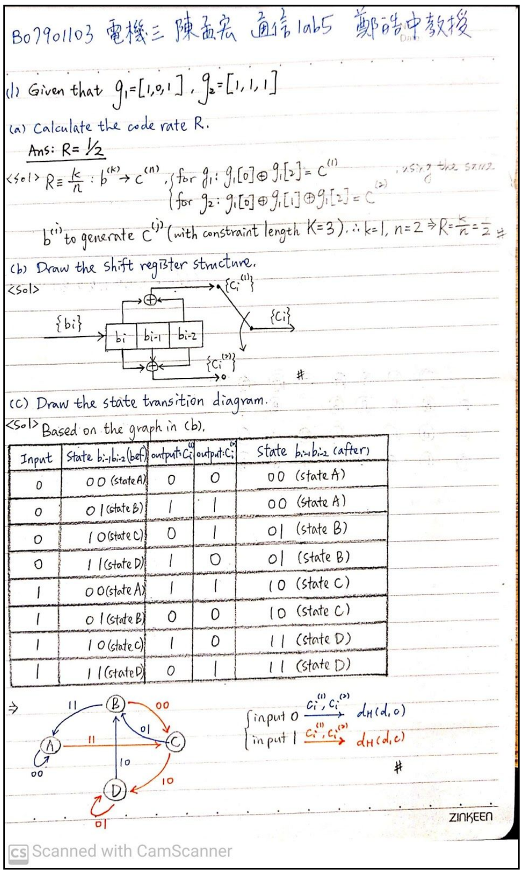
Lab5: Channel Coding

B07901103 電機三 陳孟宏

<Problem 1>

- **There are also .jpg file at the submitted folder.**



B07901103 電機三 陳孟宏 通信lab5 鄭皓中教授

(1) Given that $g_1=[1,0,1]$, $g_2=[1,1,1]$

(a) Calculate the code rate R.

Ans: $R=1/2$

<Sol> $R\equiv\frac{k}{n}: b^{(k)}\to c^{(n)}$, $\begin{cases}\text{for } g_1: g_1[0]\oplus g_1[2]=c^{(1)}\\ \text{for } g_2: g_1[0]\oplus g_1[1]\oplus g_1[2]=c^{(2)}\end{cases}$ using the same

$b^{(i)}$ to generate $c^{(j)}$ (with constraint length K=3). ∴ $k=1$, $n=2 \Rightarrow R=\frac{k}{n}=\frac{1}{2}$ #

(b) Draw the shift register structure.

<Sol>

(c) Draw the state transition diagram.

<Sol> Based on the graph in (b).

Input	State $b_{i-1}b_{i-2}$ (bef)	output: $c_i^{(1)}$	output: $c_i^{(2)}$	State $b_i b_{i-1}$ (after)
0	00 (state A)	0	0	00 (state A)
0	01 (state B)	1	1	00 (state A)
0	10 (state C)	0	1	01 (state B)
0	11 (state D)	1	0	01 (state B)
1	00 (state A)	1	1	10 (state C)
1	01 (state B)	0	0	10 (state C)
1	10 (state C)	1	0	11 (state D)
1	11 (state D)	0	1	11 (state D)

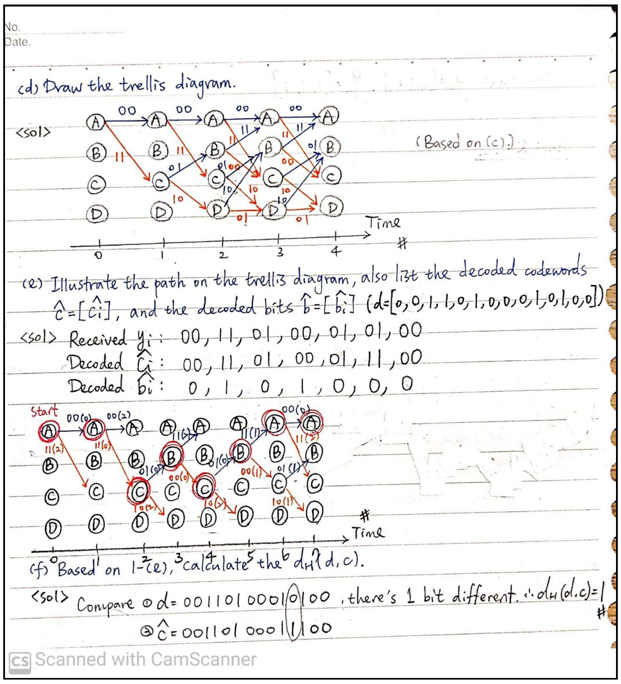
⇒

$\begin{cases}\text{input } 0 \xrightarrow{c_i^{(1)}, c_i^{(2)}} d_H(d,0)\\ \text{input } 1 \xrightarrow{c_i^{(1)}, c_i^{(2)}} d_H(d,c)\end{cases}$

#

ZINKEEN

Scanned with CamScanner



No.
Date.

(d) Draw the trellis diagram.

<Sol>

(e) Illustrate the path on the trellis diagram, also list the decoded codewords $\hat{c}=[\hat{c}_i]$, and the decoded bits $\hat{b}=[\hat{b}_i]$ ($d=[0,0,1,1,0,1,0,0,0,1,0,1,0,0]$)

<Sol> Received y_i: 00, 11, 01, 00, 01, 01, 00

Decoded $\hat{c}_i$: 00, 11, 01, 00, 01, 11, 00

Decoded $\hat{b}_i$: 0, 1, 0, 1, 0, 0, 0

(f) Based on 1-(e), calculate the $d_H(d,c)$.

<Sol> Compare ① $d=00110100010100$, there's 1 bit different. ∴ $d_H(d,c)=1$ #

② $\hat{c}=00110100011100$

Scanned with CamScanner

<Problem 2>

- **Construct a FSM table by declaring global variables and doing xor. (general case of K == 3)**

```
global A_0 A_1 B_0 B_1 C_0 C_1 D_0 D_1;
A_0 = '';
A_1 = '';
B_0 = '';
B_1 = '';
C_0 = '';
C_1 = '';
D_0 = '';
D_1 = '';
a_0 = [0, 0, 0];
b_0 = [0, 0, 1];
c_0 = [0, 1, 0];
d_0 = [0, 1, 1];
a_1 = [1, 0, 0];
b_1 = [1, 0, 1];
c_1 = [1, 1, 0];
d_1 = [1, 1, 1];
```

(initialize)

```
for j = 1:size(impulse_response, 2)
    if (impulse_response(i, j) == 1)
        g = [g, j];
    end
end
```

(if g(i) == 1, do xor)

```
if (length(g) == 1)
    A_0 = [A_0, int2str(a_0(g(1)))];
    A_1 = [A_1, int2str(a_1(g(1)))];
    B_0 = [B_0, int2str(b_0(g(1)))];
    B_1 = [B_1, int2str(b_1(g(1)))];
    C_0 = [C_0, int2str(c_0(g(1)))];
    C_1 = [C_1, int2str(c_1(g(1)))];
    D_0 = [D_0, int2str(d_0(g(1)))];
    D_1 = [D_1, int2str(d_1(g(1)))];
end
```

(if only 1 bit needs to do xor, xor the bit with ‘0’)

```
for k = 2:length(g)
    a = xor(a, a_0(g(k)));
    b = xor(b, b_0(g(k)));
    c = xor(c, c_0(g(k)));
    d = xor(d, d_0(g(k)));
    aa = xor(aa, a_1(g(k)));
    bb = xor(bb, b_1(g(k)));
    cc = xor(cc, c_1(g(k)));
    dd = xor(dd, d_1(g(k)));
end
```

(else do xor with others)

```
A_0 = [A_0, int2str(a)];
A_1 = [A_1, int2str(aa)];
B_0 = [B_0, int2str(b)];
B_1 = [B_1, int2str(bb)];
C_0 = [C_0, int2str(c)];
C_1 = [C_1, int2str(cc)];
D_0 = [D_0, int2str(d)];
D_1 = [D_1, int2str(dd)];
```

(finally we can get the output bits)

- **(a) Encode binary data**
  - **result** of encoding {1, 0, 1, 1, 0}, which is the same as lecture ppt.

```
>> lab5_q2
Encoded data:
111001100110010
```

(mine)

Input bits b_i	1	0	1	1	0
Coded bits c_i	111	001	100	110	010

(lecture ppt)

  - **main function**: input a binary sequence, and call the function “FSM_Table” to output the encoded bits sequence.

```
% conv_enc
function enc_data = conv_enc(bin, imp)
    % coded bits
    enc_data = [];
    % initial state
    K = size(imp, 2);
    state = '';
    for i = 1:K-1
        state = [state, '0'];
    end
    % For every input bits
    for i = 1:length(bin)
        [state, enc_data] = FSM_Table(state, bin(i), enc_data);
    end
end
```

  - **subfunction**: FSM_Table(finite state machine table): Based on the constructed table to output a list containing “state b(i-1)b(i-2)” and “encoded bits”, dividing the scenario into 4 parts -- state A/B/C/D.

Input	State $b_i b_{i-2}$ (Before)	Output $c_i^{(1)}$	Output $c_i^{(2)}$	Output $c_i^{(3)}$	State $b_i b_{i-2}$ (After)
0	00 (State *A*)	0	0	0	00 (State *A*)
0	01 (State *B*)	0	1	1	00 (State *A*)
0	10 (State *C*)	0	0	1	01 (State *B*)
0	11 (State *D*)	0	1	0	01 (State *B*)
1	00 (State *A*)	1	1	1	10 (State *C*)
1	01 (State *B*)	1	0	0	10 (State *C*)
1	10 (State *C*)	1	1	0	11 (State *D*)
1	11 (State *D*)	1	0	1	11 (State *D*)

(state table)

```matlab
% State A: 00
if (strcmp(state, '00') == 1)
    if (bits == 0)
        s = '00';
        e = [enc_string, A_0];
        return;
    end
    if (bits == 1)
        s = '10';
        e = [enc_string, A_1];
        return;
    end
end
```

(State A)

```matlab
% State B: 01
if (strcmp(state, '01') == 1)
    if (bits == 0)
        s = '00';
        e = [enc_string, B_0];
        return;
    end
    if (bits == 1)
        s = '10';
        e = [enc_string, B_1];
        return;
    end
end
```

(State B)

```matlab
% State C: 10
if (strcmp(state, '10') == 1)
    if (bits == 0)
        s = '01';
        e = [enc_string, C_0];
        return;
    end
    if (bits == 1)
        s = '11';
        e = [enc_string, C_1];
        return;
    end
end
```

(State C)

```
% State D: 11
if (strcmp(state, '11') == 1)
    if (bits == 0)
        s = '01';
        e = [enc_string, D_0];
        return;
    end
    if (bits == 1)
        s = '11';
        e = [enc_string, D_1];
        return;
    end
end
```
(State D)

- **(b) Decode the received y(i)**
  - **result** of decoding "d"

$$\boldsymbol{d} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

```
Decoded data:
     0     0     1     1     1     0     1
```

  - **result** of 1-(e) via Matlab & handwritting ({0, 1, 0, 1, 0, 0, 0})

```
>> lab5_q2b_verify_1e
Decoded data of 1-(e):
     0     1     0     1     0     0     0
```
(Matlab)

(e) Illustrate the path on the trellis diagram, also list the decoded codewords $\hat{c}=[\hat{c}_i]$, and the decoded bits $\hat{b}=[\hat{b}_i]$ $(d=[0,0,1,1,0,1,0,0,0,1,0,1,0,0])$

<sol> Received y_i: 00, 11, 01, 00, 01, 01, 00

Decoded $\hat{c}_i$: 00, 11, 01, 00, 01, 11, 00

Decoded $\hat{b}_i$: 0, 1, 0, 1, 0, 0, 0

(handwritting)

  - **main function**: 4 state -- A/B/C/D, call the function "ham_dis", which outputs "[next_state, encoded bits, hamming_distance]", if next input '0' or '1' all has the same hamming distance, it will set the next_state = 2 and call another function "next_ham" to compare next route and determine the current way.

```matlab
% State A
dist = 0;
if (strcmp(state, '00') == 1)
    [next_state, state, dist] = ham_dis(bits, A_0, A_1, '00');
    if (next_state == 2)
        [next_state, state, dist] = next_ham(bits_next, A_0, A_1, C_0, C_1, '00');
        dec_data = [dec_data, next_state];
        continue;
    else
        dec_data = [dec_data, next_state];
        continue;
    end
end
```

(State A)

```matlab
% State B
if (strcmp(state, '01') == 1)
    [next_state, state, dist] = ham_dis(bits, B_0, B_1, '01');
    if (next_state == 2)
        [next_state, state, dist] = next_ham(bits_next, A_0, A_1, C_0, C_1, '01');
        dec_data = [dec_data, next_state];
        continue;
    else
        dec_data = [dec_data, next_state];
        continue;
    end
end
```

(State B)

```matlab
% State C
if (strcmp(state, '10') == 1)
    [next_state, state, dist] = ham_dis(bits, C_0, C_1, '10');
    if (next_state == 2)
        [next_state, state, dist] = next_ham(bits_next, B_0, B_1, D_0, D_1, '10');
        dec_data = [dec_data, next_state];
        continue;
    else
        dec_data = [dec_data, next_state];
        continue;
    end
end
```

(State C)

```matlab
% State D
if (strcmp(state, '11') == 1)
    [next_state, state, dist] = ham_dis(bits, D_0, D_1, '11');
    if (next_state == 2)
        [next_state, state, dist] = next_ham(bits_next, B_0, B_1, D_0, D_1, '11');
        dec_data = [dec_data, next_state];
        continue;
    else
        dec_data = [dec_data, next_state];
        continue;
    end
end
```

(State D)

- **subfunction 1**: ham_dis, if there is a bit different from the correct decoded bit, accumulating it by 1. Divide it into 2 scenario -- input ‘0’ and ‘1’

```
for i = 1:length(bits)
    if (strcmp(bits(i), str_1(i)) == 0)
        diff_0 = diff_0 + 1;
    end
    if (strcmp(bits(i), str_2(i)) == 0)
        diff_1 = diff_1 + 1;
    end
end
```

```
% decode to 0
if (diff_0 < diff_1)
    next_state = 0;
    dist = diff_0;
    % State A
    if (strcmp(state, '00') == 1)
        output = '00';
    end
    % State B
    if (strcmp(state, '01') == 1)
        output = '00';
    end
    % State C
    if (strcmp(state, '10') == 1)
        output = '01';
    end
    % State D
    if (strcmp(state, '11') == 1)
        output = '01';
    end
end
```

```
% decode to 1
if (diff_0 > diff_1)
    next_state = 1;
    dist = diff_1;
    % State A
    if (strcmp(state, '00') == 1)
        output = '10';
    end
    % State B
    if (strcmp(state, '01') == 1)
        output = '10';
    end
    % State C
    if (strcmp(state, '10') == 1)
        output = '11';
    end
    % State D
    if (strcmp(state, '11') == 1)
        output = '11';
    end
end
```

- **subfunction 2**: next_ham, according to the next two possible nodes two other routes respectively, record their hamming distance and recursively call.

```
[n_1, s_1, d_1] = ham_dis(bit, str_1_1, str_1_2, sta);
[n_2, s_2, d_2] = ham_dis(bit, str_2_1, str_2_2, sta);
if (d_1 < d_2)
    n = n_1;
    s = s_1;
    d = d_1;
elseif (d_1 > d_2)
    n = n_2;
    s = s_2;
    d = d_2;
end
```

- **(c) Simulation**
  - Generate 100003 (>100000) a random (0, 1) with the equiprobability.

```
r = mod(reshape(randperm(1*100002), 1, 100002), 2);
```

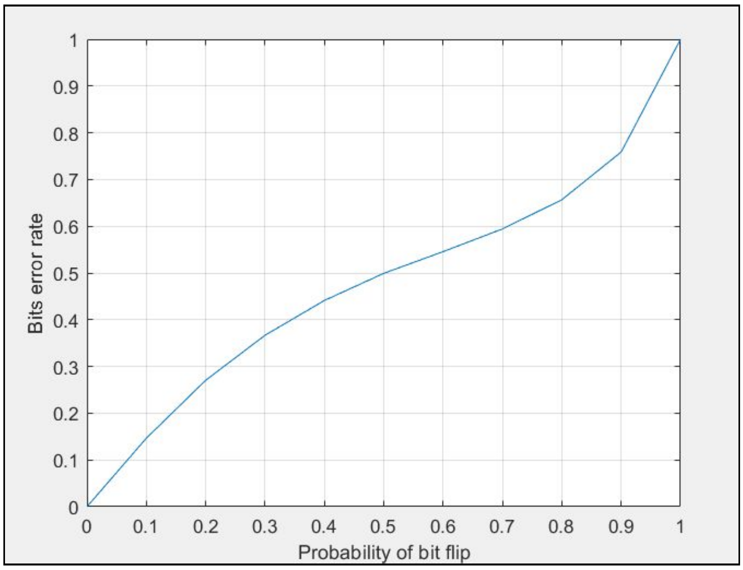
- Encode the list with my FIR and convert the string bits to binary bits.

```
e = conv_enc(r, impulse_response);
e_double = [];
for i = 1:length(e)
    temp = str2double(e(i));
    e_double = [e_double, temp];
end
```

- Use the build-in function “bsc()” and “biterr()” to flip the encoded bits with a probability p=[0, 1] (for every 0.1). Calculate the bit error rate

```
% Bit Error
p = 0;
bit_error = [];
for i = 1:11
    p = (i-1)*0.1;
    BSC = bsc(e_double, p); % Binary symmetric channel
    dec_r = conv_dec(BSC, impulse_response);
    [num, err] = biterr(r, dec_r);
    bit_error = [bit_error, err];
end
```

- Plot the result.

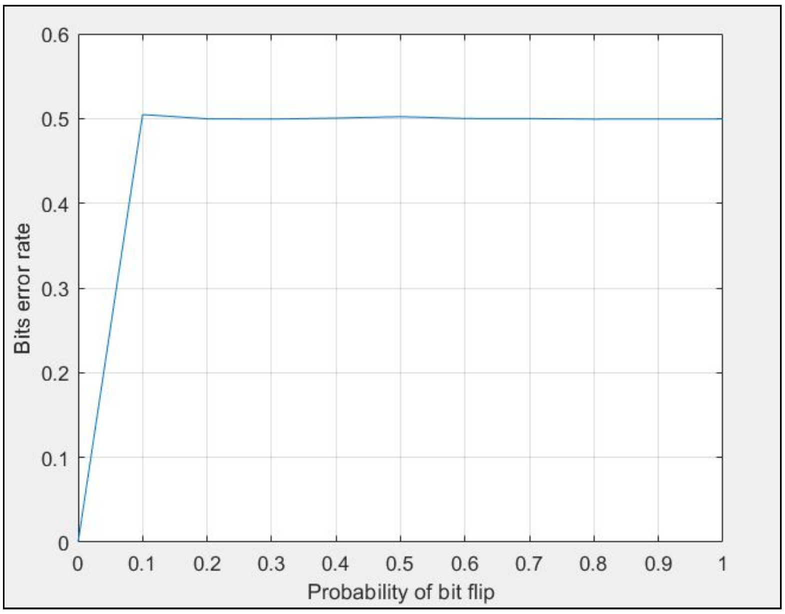


<Problem 3>

- **Simulate as 2-(c) with the given generators (random 100002 (>100000) numbers).** We can find that for 2-(c), the bit error rate will increase until 1 (i.e.

all the bits will be flipped), while in 3, the bit error rates are oscillated around p=0.5.

$$\mathbf{g}_1 = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}, \qquad \mathbf{g}_2 = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}.$$



(result)