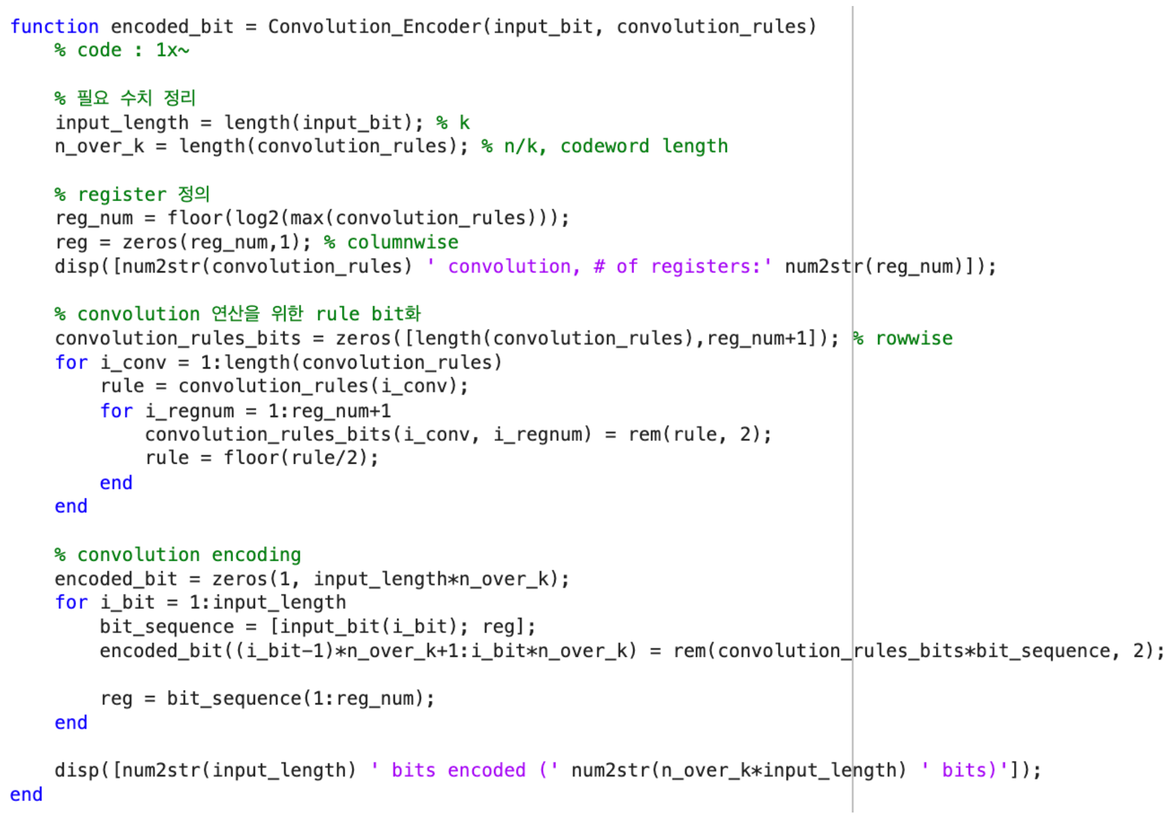
Project (통신시스템)

2021142196 김현수

목차

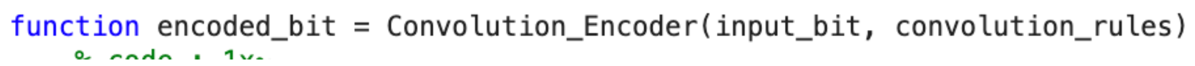
1) Channel encoder

전체 코드



```
function encoded_bit = Convolution_Encoder(input_bit, convolution_rules)
    % code : 1x~

    % 필요 수치 정리
    input_length = length(input_bit); % k
    n_over_k = length(convolution_rules); % n/k, codeword length

    % register 정의
    reg_num = floor(log2(max(convolution_rules)));
    reg = zeros(reg_num,1); % columnwise
    disp([num2str(convolution_rules) ' convolution, # of registers:' num2str(reg_num)]);

    % convolution 연산을 위한 rule bit화
    convolution_rules_bits = zeros([length(convolution_rules),reg_num+1]); % rowwise
    for i_conv = 1:length(convolution_rules)
        rule = convolution_rules(i_conv);
        for i_regnum = 1:reg_num+1
            convolution_rules_bits(i_conv, i_regnum) = rem(rule, 2);
            rule = floor(rule/2);
        end
    end

    % convolution encoding
    encoded_bit = zeros(1, input_length*n_over_k);
    for i_bit = 1:input_length
        bit_sequence = [input_bit(i_bit); reg];
        encoded_bit((i_bit-1)*n_over_k+1:i_bit*n_over_k) = rem(convolution_rules_bits*bit_sequence, 2);

        reg = bit_sequence(1:reg_num);
    end

    disp([num2str(input_length) ' bits encoded (' num2str(n_over_k*input_length) ' bits)']);
end
```

설명 1)

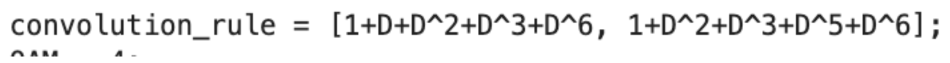


```
function encoded_bit = Convolution_Encoder(input_bit, convolution_rules)
```

input_bit로 1xk의 matrix로 input을 받게 된다.

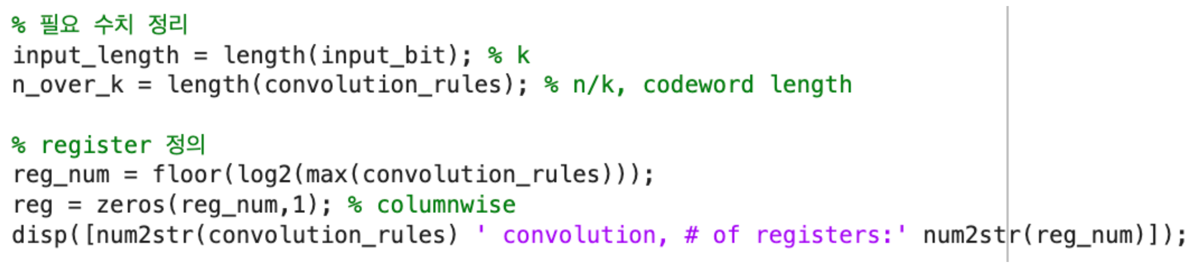
convolution_rules로 generator polynomial을 받게 된다.
받는 형식은 다음과 같다.

(D=2로 정의하였다.)



```
convolution_rule = [1+D+D^2+D^3+D^6, 1+D^2+D^3+D^5+D^6];
```

설명 2)

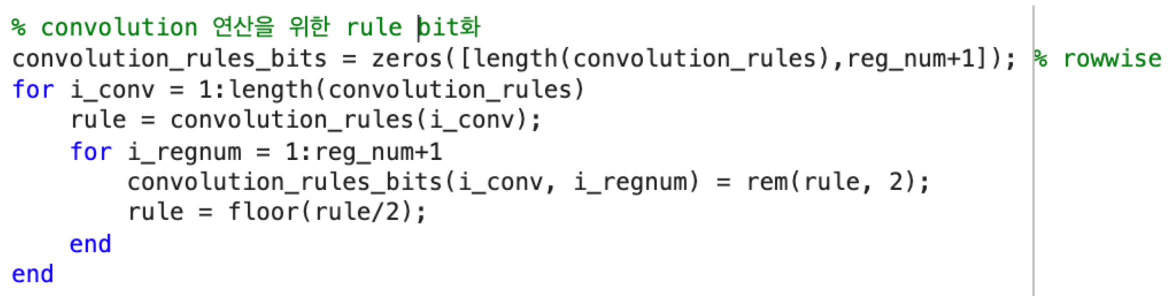


```
% 필요 수치 정리
input_length = length(input_bit); % k
n_over_k = length(convolution_rules); % n/k, codeword length

% register 정의
reg_num = floor(log2(max(convolution_rules)));
reg = zeros(reg_num,1); % columnwise
disp([num2str(convolution_rules) ' convolution, # of registers:' num2str(reg_num)]);
```

input_bit의 길이를 받아오고 convolution_rules(generator polynomial)의 개수를 통해 code rate를 얻어온다.

Convolution encoding을 할 때 필요한 레지스터의 개수를 위와 같이 얻을 수 있고 이를 바탕으로 register를 정의해주었다.

설명 3)



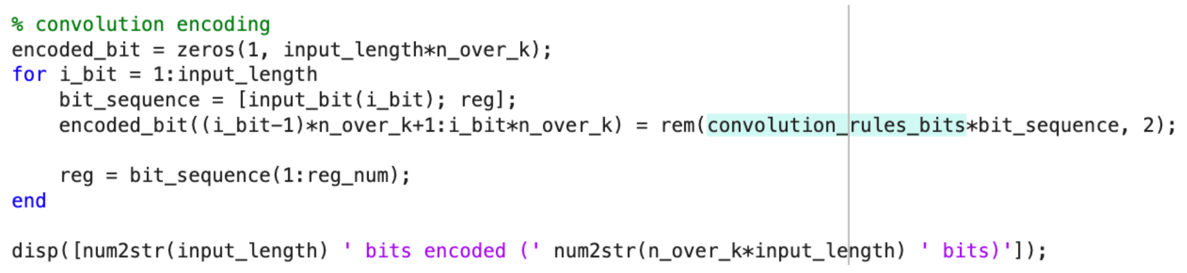
```
% convolution 연산을 위한 rule bit화
convolution_rules_bits = zeros([length(convolution_rules),reg_num+1]); % rowwise
for i_conv = 1:length(convolution_rules)
    rule = convolution_rules(i_conv);
    for i_regnum = 1:reg_num+1
        convolution_rules_bits(i_conv, i_regnum) = rem(rule, 2);
        rule = floor(rule/2);
    end
end
```

받아온 convolution_rules을 2진수로 고쳐주기 위한 코드이다.

[7 5]라면 $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$로 바꾸게 된다.

이와 같은 연산을 진행한 이유는 convolution을 진행할 때 matrix multiply로 진행하기 위해서이다.

설명 3)

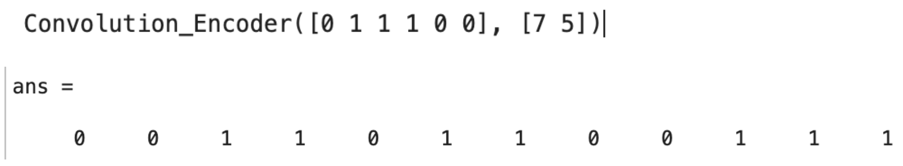


```
% convolution encoding
encoded_bit = zeros(1, input_length*n_over_k);
for i_bit = 1:input_length
    bit_sequence = [input_bit(i_bit); reg];
    encoded_bit((i_bit-1)*n_over_k+1:i_bit*n_over_k) = rem(convolution_rules_bits*bit_sequence, 2);

    reg = bit_sequence(1:reg_num);
end

disp([num2str(input_length) ' bits encoded (' num2str(n_over_k*input_length) ' bits)']);
```

bit_sequence로 convolution에서 사용할 bit와 register상태를 연결해주고 이를 위에서 만든 2진수화한 convolution_rules와 matrix multiply를 통해 연산을 진행한다. 이때 xor 연산이므로 2로 나눈 나머지를 가져와주었다. 이 연산을 모든 비트에 대해 사용하면 convolution encoding을 마치게 된다.

이때 bit_sequence는 column 방향이다.

시연)



```
 Convolution_Encoder([0 1 1 1 0 0], [7 5])
```

```
ans =

     0     0     1     1     0     1     1     0     0     1     1     1
```

2) Modulator

전체 코드

```
function modulated_symbols = moudulator_M_ary_QAM(encoded_bits, M)

    encoded_bits_length = length(encoded_bits); % n
    bit_per_symbols = floor(log2(M));
    symbols_length = encoded_bits_length/bit_per_symbols;
    if floor(bit_per_symbols/2) ~= log2(M)/2
        error('M must be power of power of 2^2');
    end

    [~, ~, bit_to_symbols_lookup_idxp] = graymap(M);
    modulated_symbols = zeros(1,symbols_length);
    bit_sampled = reshape(encoded_bits, bit_per_symbols,symbols_length)';

    for i_sym = 1:symbols_length
        symbol_bits = bit_to_int(bit_sampled(i_sym,:));
        modulated_symbols(i_sym) = bit_to_symbols_lookup_idxp(symbol_bits+1);
    end

    disp([num2str(symbols_length) ' symbols modulated (' num2str(M) '-ary QAM)']);
end
```

설명 1)

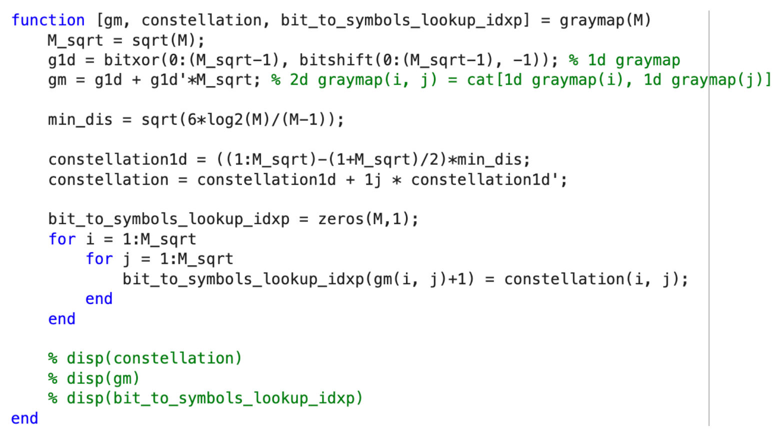
```
function modulated_symbols = moudulator_M_ary_QAM(encoded_bits, M)

    encoded_bits_length = length(encoded_bits); % n
    bit_per_symbols = floor(log2(M));
    symbols_length = encoded_bits_length/bit_per_symbols;
    if floor(bit_per_symbols/2) ~= log2(M)/2
        error('M must be power of power of 2^2');
    end
```

encoded_bits를 받아오고 어떤 QAM을 사용할 것인지 M으로 받아온다.

이를 통해 코드 길이와 심볼당 몇비트로 symbol을 만들어내는지에 대한 정보를 얻어온다.

설명 2)

```
[~, ~, bit_to_symbols_lookup_idxp] = graymap(M);
```

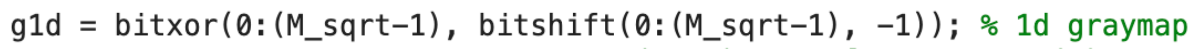


```
function [gm, constellation, bit_to_symbols_lookup_idxp] = graymap(M)
    M_sqrt = sqrt(M);
    g1d = bitxor(0:(M_sqrt-1), bitshift(0:(M_sqrt-1), -1)); % 1d graymap
    gm = g1d + g1d'*M_sqrt; % 2d graymap(i, j) = cat[1d graymap(i), 1d graymap(j)]

    min_dis = sqrt(6*log2(M)/(M-1));

    constellation1d = ((1:M_sqrt)-(1+M_sqrt)/2)*min_dis;
    constellation = constellation1d + 1j * constellation1d';

    bit_to_symbols_lookup_idxp = zeros(M,1);
    for i = 1:M_sqrt
        for j = 1:M_sqrt
            bit_to_symbols_lookup_idxp(gm(i, j)+1) = constellation(i, j);
        end
    end

    % disp(constellation)
    % disp(gm)
    % disp(bit_to_symbols_lookup_idxp)
end
```

M을 기반으로 graymapping을 만들어낸다.

우선 만들고자 하는 graymap은 2 dimension 이기 때문에 예를 들어 16QAM이라면 4bit symbol을 통해 만드는데 2bit를 1dimension에서 만들고 나머지 2bit를 나머지 dimension에서 만들어 cascade하게 연결하였다.

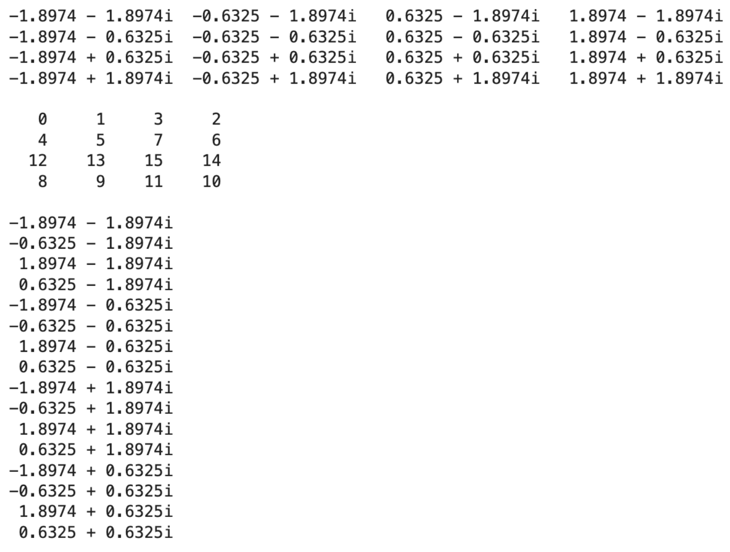
1dimension graymap을 만드는 과정은 다음과 같다.



```
g1d = bitxor(0:(M_sqrt-1), bitshift(0:(M_sqrt-1), -1)); % 1d graymap
```

이를 바탕으로 constellation과 graymap을 만들고 비트 순서대로 constallation을 받아오는 것을 bit_to_symbols_lookup_idxp로 돌려준다.

이를 16QAM으로 실행해보면 다음과 같이 된다.



```
-1.8974 - 1.8974i  -0.6325 - 1.8974i   0.6325 - 1.8974i   1.8974 - 1.8974i
-1.8974 - 0.6325i  -0.6325 - 0.6325i   0.6325 - 0.6325i   1.8974 - 0.6325i
-1.8974 + 0.6325i  -0.6325 + 0.6325i   0.6325 + 0.6325i   1.8974 + 0.6325i
-1.8974 + 1.8974i  -0.6325 + 1.8974i   0.6325 + 1.8974i   1.8974 + 1.8974i

    0     1     3     2
    4     5     7     6
   12    13    15    14
    8     9    11    10

-1.8974 - 1.8974i
-0.6325 - 1.8974i
 1.8974 - 1.8974i
 0.6325 - 1.8974i
-1.8974 - 0.6325i
-0.6325 - 0.6325i
 1.8974 - 0.6325i
 0.6325 - 0.6325i
-1.8974 + 1.8974i
-0.6325 + 1.8974i
 1.8974 + 1.8974i
 0.6325 + 1.8974i
-1.8974 + 0.6325i
-0.6325 + 0.6325i
 1.8974 + 0.6325i
 0.6325 + 0.6325i
```

위부터 순서대로 constellation, graymap, bit_to_symbols_lookup_idxp

설명 3)

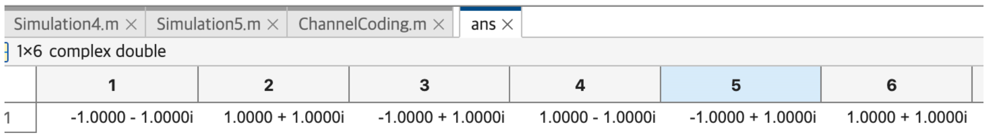
```
    bit_sampled = reshape(encoded_bits, bit_per_symbols,symbols_length)';

    for i_sym = 1:symbols_length
        symbol_bits = bit_to_int(bit_sampled(i_sym,:));
        modulated_symbols(i_sym) = bit_to_symbols_lookup_idxp(symbol_bits+1);
    end

    disp([num2str(symbols_length) ' symbols modulated (' num2str(M) '-ary QAM)']);
end
```

위에서 받은 bit별 constellation을 가지고 encoded bit를 symbol로 바꿔주고 modulation을 진행한다.

이때 symbol energy가 encoded bit기준으로 E_b=1이 되도록 minimum distance를 설정하였으며 encoding에서의 code rate R은 AWGN channel에서 다루도록 하였다.

시연)

```
moudulator_M_ary_QAM([0 0 1 1 0 1 1 0 0 1 1 1], 4)
```



1×6 complex double

	1	2	3	4	5	6
1	-1.0000 - 1.0000i	1.0000 + 1.0000i	-1.0000 + 1.0000i	1.0000 - 1.0000i	-1.0000 + 1.0000i	1.0000 + 1.0000i

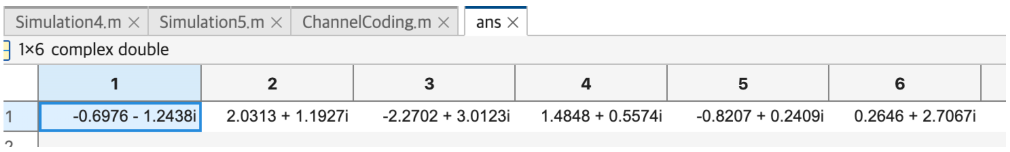
3) AWGN channel

```
%% AWGN Channel

function noised_symbols = AWGN_channel(symbols, SNR, n_over_k)
    disp('AWGN channel added')

    SNR_linear = 10^(SNR/10);

    sigma = sqrt(0.5/SNR_linear*n_over_k);
    noise_bit = randn([1, length(symbols)]) * sigma + 1j*randn([1, length(symbols)]) * sigma;

    noised_symbols = symbols + noise_bit;
end
```

Noise가 N(0, sigma^2)이 되도록 더해주었는데 SNR = E_b_rare/N_0에서 E_b_rare * R = E_b_encoded이고 E_b_encoded를 1로 잡아 simulation할 예정이기에 sigma를 sigma^2=N_0/2의 식을 이용하여 위와 같이 잡았다. 이를 더해주면서 AWGN channel을 구성하였다.

시연)

```
AWGN_channel(moudulator_M_ary_QAM([0 0 1 1 0 1 1 0 0 1 1 1], 4), 5, 2)
```


Simulation4.m × Simulation5.m × ChannelCoding.m × ans ×

1×6 complex double

	1	2	3	4	5	6
1	-0.6976 - 1.2438i	2.0313 + 1.1927i	-2.2702 + 3.0123i	1.4848 + 0.5574i	-0.8207 + 0.2409i	0.2646 + 2.7067i

4-1) Demodulator(Hard)

Hard Decision Decoding을 하기 위해서는 모든 symbol을 정해진 symbol을 찾아서 얻어야 하기 때문에 아래와 같이 진행하였다.

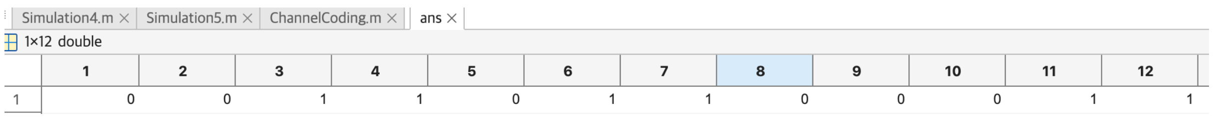
```
function demodulated_bits = hard_demoudulator_M_ary_QAM(symbols, M)
    sym_length = length(symbols);
    symbol_bit_num = log2(M);

    disp([num2str(sym_length) ' symbols demodulated (' num2str(sym_length*symbol_bit_num) 'bits)']);

    min_dis = sqrt(6*sqrt(M)/(M-1));
    [gray_map, constellation, bit_to_symbols_lookup_idxp] = graymap(M);
    symbols = symbols/min_dis;
    symbols = symbols+0.5+0.5j;
    symbols = round(symbols);
    symbols = symbols-0.5-0.5j;
    symbols = symbols-constellation(1,1)/min_dis+1+1j;


    M_sqrt = sqrt(M);
    demodulated_bits = zeros(1, sym_length*symbol_bit_num);
    for idx_sym = 1:sym_length
        sym = symbols(idx_sym);
        i = max(1, min(sqrt(M), round(imag(sym))));
        j = max(1, min(sqrt(M), round(real(sym))));


        demodulated_bits((idx_sym-1)*symbol_bit_num+1:idx_sym*symbol_bit_num) = int_to_bit(gray_map(i, j), symbol_bit_num);
    end
end
```

구현 아이디어는 symbol을 minimum distance로 나누어주어 symbol간의 간격이 1이 되도록 하고 0.5+0.5j를 더해 기존 symbol이 .5단위 인 것을 정수단위로 바꾼 후 round 하여 가장 가까운 symbol을 얻도록 하였다. 이 방식을 사용하면 모든 심볼에 대해 한번에 연산을 진행할 수 있다.

이렇게 얻은 symbol constellation종류를 기반으로 graymap에서 어떤 bit를 modulation하여 얻었는지 구하고 출력한다.

시연)

```
hard_demoudulator_M_ary_QAM(AWGN_channel(moudulator_M_ary_QAM([0 0 1 1 0 1 1 0 0 1 1 1], 4), 5, 2), 4)
```



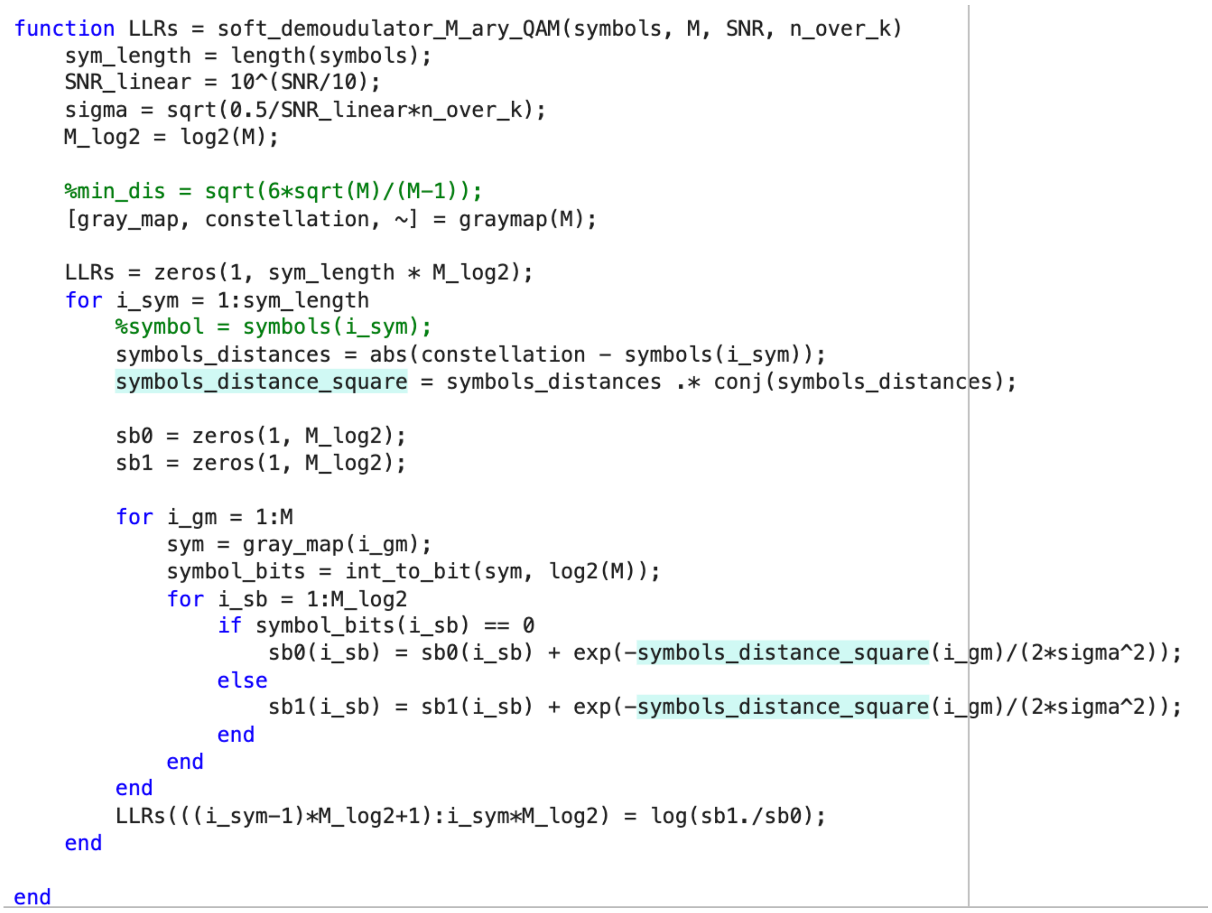
	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	1	1	0	1	1	0	0	0	1	1

10번째 bit가 AWGN에 의해 바뀐 것도 확인할 수 있다.

4-2) Demodulator(Soft)

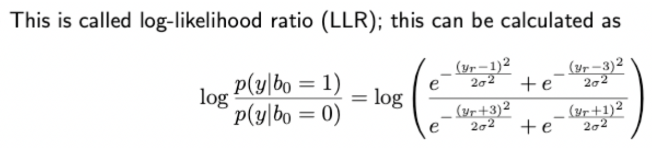
Soft Decision Decoding을 하기 위해서는 symbol별로 구분하는 것이 아닌 비트별로 LLR을 얻어와야한다.

전체 코드



```
function LLRs = soft_demoudulator_M_ary_QAM(symbols, M, SNR, n_over_k)
    sym_length = length(symbols);
    SNR_linear = 10^(SNR/10);
    sigma = sqrt(0.5/SNR_linear*n_over_k);
    M_log2 = log2(M);

    %min_dis = sqrt(6*sqrt(M)/(M-1));
    [gray_map, constellation, ~] = graymap(M);

    LLRs = zeros(1, sym_length * M_log2);
    for i_sym = 1:sym_length
        %symbol = symbols(i_sym);
        symbols_distances = abs(constellation - symbols(i_sym));
        symbols_distance_square = symbols_distances .* conj(symbols_distances);

        sb0 = zeros(1, M_log2);
        sb1 = zeros(1, M_log2);

        for i_gm = 1:M
            sym = gray_map(i_gm);
            symbol_bits = int_to_bit(sym, log2(M));
            for i_sb = 1:M_log2
                if symbol_bits(i_sb) == 0
                    sb0(i_sb) = sb0(i_sb) + exp(-symbols_distance_square(i_gm)/(2*sigma^2));
                else
                    sb1(i_sb) = sb1(i_sb) + exp(-symbols_distance_square(i_gm)/(2*sigma^2));
                end
            end
        end
        LLRs(((i_sym-1)*M_log2+1):i_sym*M_log2) = log(sb1./sb0);
    end

end
```

Hard Decision과 다르게 LLR을 다음과 같이 symbol간의 distance를 이용하여 likelyhood를 계산하여 모두 더하는 식으로 구현하였다.



This is called log-likelihood ratio (LLR); this can be calculated as

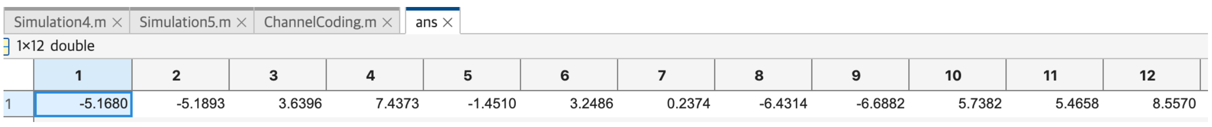
$$\log \frac{p(y|b_0=1)}{p(y|b_0=0)} = \log \left(\frac{e^{-\frac{(y_r-1)^2}{2\sigma^2}} + e^{-\frac{(y_r-3)^2}{2\sigma^2}}}{e^{-\frac{(y_r+3)^2}{2\sigma^2}} + e^{-\frac{(y_r+1)^2}{2\sigma^2}}} \right)$$

강의안 ch7 p.31/86

이를 기준으로 LLR를 계산하고 출력한다.

시연)

```
soft_demoudulator_M_ary_QAM(AWGN_channel(moudulator_M_ary_QAM([0 0 1 1 0 1 1 0 0 1 1 1], 4), 5, 2), 4, 5, 2)
```


Simulation4.m × Simulation5.m × ChannelCoding.m × ans ×

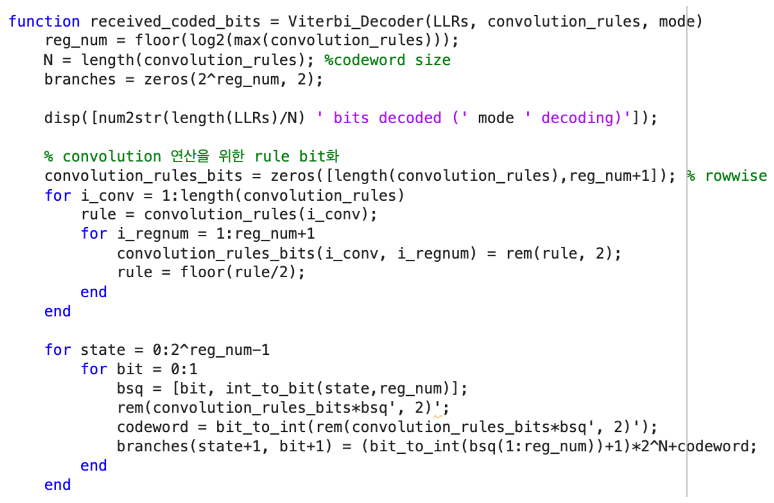
1×12 double

	1	2	3	4	5	6	7	8	9	10	11	12
1	-5.1680	-5.1893	3.6396	7.4373	-1.4510	3.2486	0.2374	-6.4314	-6.6882	5.7382	5.4658	8.5570

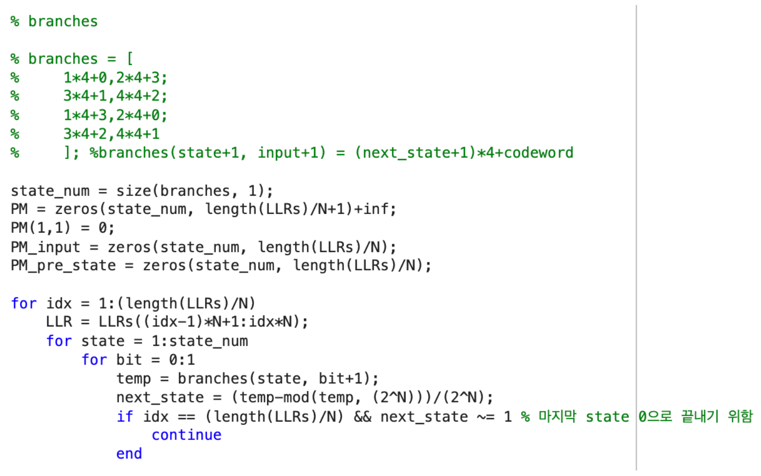
성공적으로 LLR을 얻어낸 것을 확인할 수 있다.

5) Viterbi Decoder

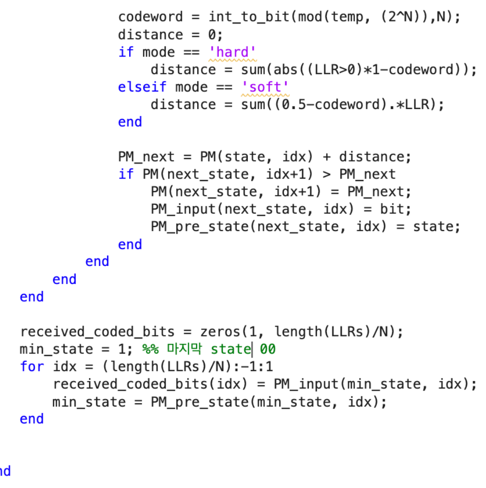
전체 코드



```
function received_coded_bits = Viterbi_Decoder(LLRs, convolution_rules, mode)
    reg_num = floor(log2(max(convolution_rules)));
    N = length(convolution_rules); %codeword size
    branches = zeros(2^reg_num, 2);

    disp([num2str(length(LLRs)/N) ' bits decoded (' mode ' decoding)']);

    % convolution 연산을 위한 rule bit화
    convolution_rules_bits = zeros([length(convolution_rules),reg_num+1]); % rowwise
    for i_conv = 1:length(convolution_rules)
        rule = convolution_rules(i_conv);
        for i_regnum = 1:reg_num+1
            convolution_rules_bits(i_conv, i_regnum) = rem(rule, 2);
            rule = floor(rule/2);
        end
    end

    for state = 0:2^reg_num-1
        for bit = 0:1
            bsq = [bit, int_to_bit(state,reg_num)];
            rem(convolution_rules_bits*bsq', 2)';
            codeword = bit_to_int(rem(convolution_rules_bits*bsq', 2)');
            branches(state+1, bit+1) = (bit_to_int(bsq(1:reg_num))+1)*2^N+codeword;
        end
    end
```



```
    % branches

    % branches = [
    %     1*4+0,2*4+3;
    %     3*4+1,4*4+2;
    %     1*4+3,2*4+0;
    %     3*4+2,4*4+1
    %     ]; %branches(state+1, input+1) = (next_state+1)*4+codeword

    state_num = size(branches, 1);
    PM = zeros(state_num, length(LLRs)/N+1)+inf;
    PM(1,1) = 0;
    PM_input = zeros(state_num, length(LLRs)/N);
    PM_pre_state = zeros(state_num, length(LLRs)/N);

    for idx = 1:(length(LLRs)/N)
        LLR = LLRs((idx-1)*N+1:idx*N);
        for state = 1:state_num
            for bit = 0:1
                temp = branches(state, bit+1);
                next_state = (temp-mod(temp, (2^N)))/(2^N);
                if idx == (length(LLRs)/N) && next_state ~= 1 % 마지막 state 0으로 끝내기 위함
                    continue
                end
```



```
                codeword = int_to_bit(mod(temp, (2^N)),N);
                distance = 0;
                if mode == 'hard'
                    distance = sum(abs((LLR>0)*1-codeword));
                elseif mode == 'soft'
                    distance = sum((0.5-codeword).*LLR);
                end

                PM_next = PM(state, idx) + distance;
                if PM(next_state, idx+1) > PM_next
                    PM(next_state, idx+1) = PM_next;
                    PM_input(next_state, idx) = bit;
                    PM_pre_state(next_state, idx) = state;
                end
            end
        end
    end

    received_coded_bits = zeros(1, length(LLRs)/N);
    min_state = 1; %% 마지막 state 00
    for idx = (length(LLRs)/N):-1:1
        received_coded_bits(idx) = PM_input(min_state, idx);
        min_state = PM_pre_state(min_state, idx);
    end

end
```

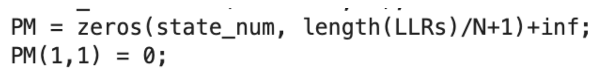
설명 1)

Braches에는 각 state에서 입력 bit별로 출력하는 codeword와 다음 state를 저장하여 추

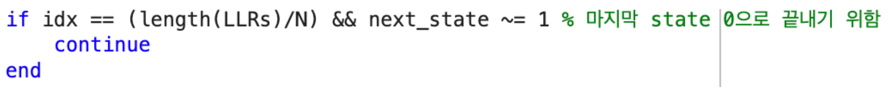
가적인 연산 없이 참조만으로 해결하도록 하였다. 이때 두개의 값을 한번에 저장하기 위해 state가 2의 거듭제곱인 것을 이용하여 그 위의 비트에 codeword를 저장하는 방식을 사용하였다.

설명 2)

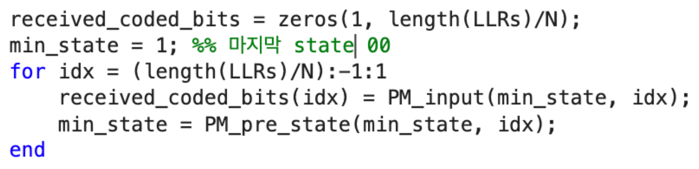
시간 state가 00이고 마지막 state가 00인 것을 이용하기 위해 아래와 같은 코드를 구성하였다.


```
PM = zeros(state_num, length(LLRs)/N+1)+inf;
PM(1,1) = 0;
```

Path Matric의 첫번째의 00 state에 0을 넣고 나머지에 Inf를 넣어주었다.


```
if idx == (length(LLRs)/N) && next_state ~= 1 % 마지막 state 0으로 끝내기 위함
    continue
end
```

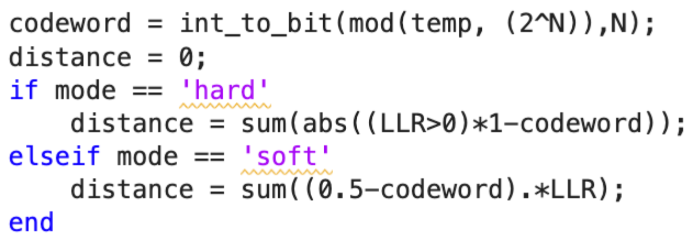
마지막 state가 00이 될 수 있도록 가장 마지막 PM을 만들 때 00이 아니면 수행하지 못하도록 하였다.


```
received_coded_bits = zeros(1, length(LLRs)/N);
min_state = 1; %% 마지막 state 00
for idx = (length(LLRs)/N):-1:1
    received_coded_bits(idx) = PM_input(min_state, idx);
    min_state = PM_pre_state(min_state, idx);
end
```

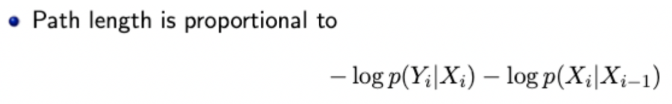
마지막 path를 따라갈 때 state가 00에서 시작하여 이전 state를 따라가도록 하였다.

설명 3)

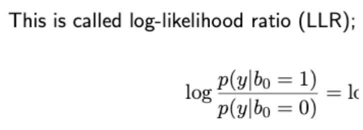
Hard와 Soft에서 distance식을 다르게 지정하여 두가지 모두 사용할 수 있도록 하였다.


```
codeword = int_to_bit(mod(temp, (2^N)),N);
distance = 0;
if mode == 'hard'
    distance = sum(abs((LLR>0)*1-codeword));
elseif mode == 'soft'
    distance = sum((0.5-codeword).*LLR);
end
```

Hard의 경우 Hamming distance를 구하도록 하였고 Soft의 경우 아래 식 두개를 조합하여 만들었다.


- Path length is proportional to

$$-\log p(Y_i|X_i) - \log p(X_i|X_{i-1})$$


This is called log-likelihood ratio (LLR);

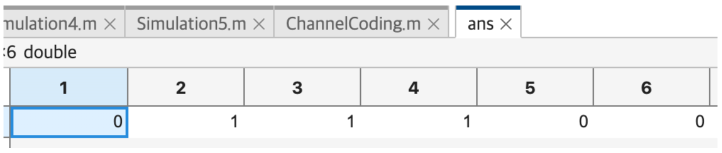
$$\log \frac{p(y|b_0=1)}{p(y|b_0=0)} = \mathrm{l}$$

BM=-LLR*x_i

Codeword에서 0.5를 빼 01이 -0.5 +0.5이 되도록 하였고 BM은 모든 BM에 대해 양수배, constant add가 가능하므로 위 코드와 같이 작성하였다.

시연)

```
LLRs = soft_demoudulator_M_ary_QAM(AWGN_channel(moudulator_M_ary_QAM([0 0 1 1 0 1 1 0 0 1 1 1], 4), 5, 2), 4, 5, 2);
Viterbi_Decoder(LLRs, [7, 5], 'soft')
```


mulation4.m × Simulation5.m × ChannelCoding.m × ans ×

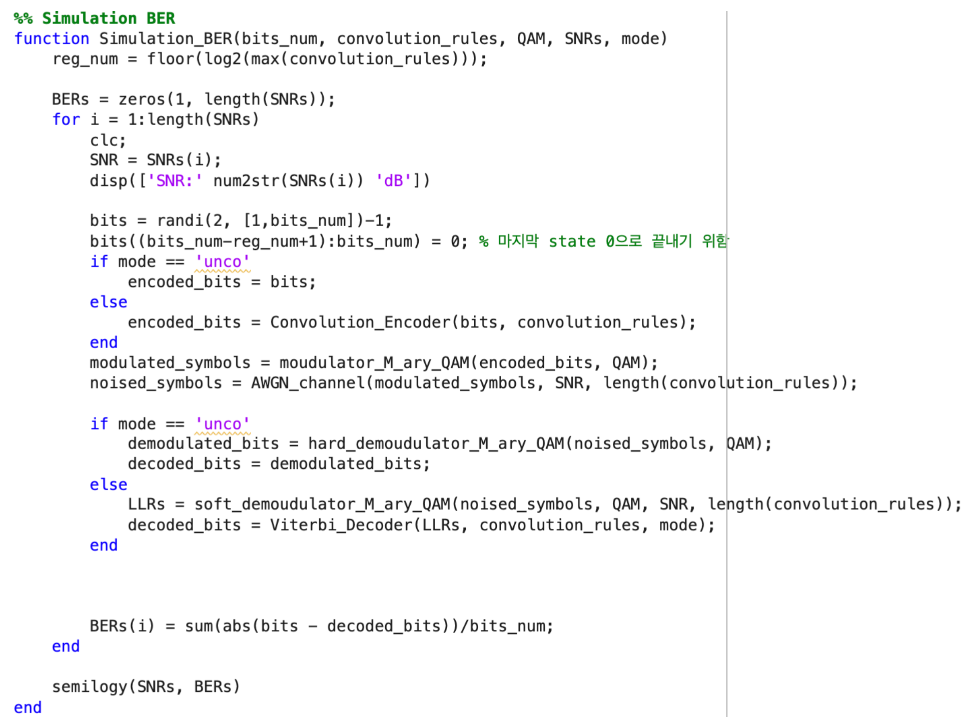
:6 double

1	2	3	4	5	6
0	1	1	1	0	0

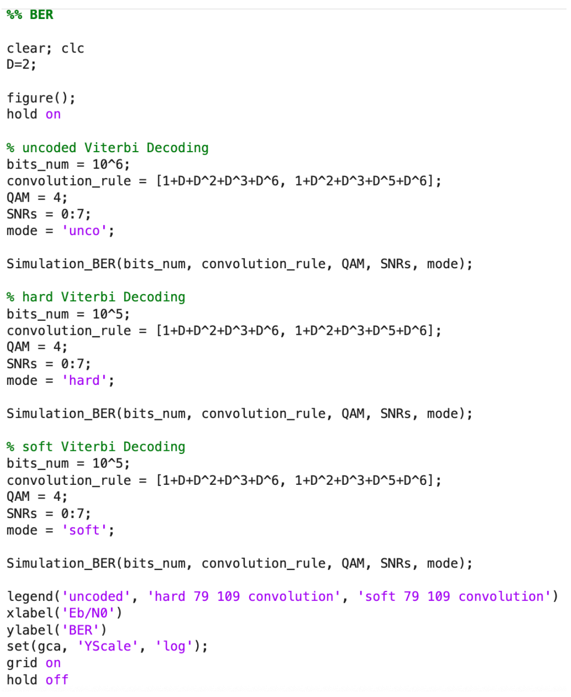
011100을 인코딩한 001101100111을 다시 Viterbi decoder를 거치니 인코딩 전 011100과 동일하게 나온 것을 확인하여 성공적으로 구현하였음을 짐작할 수 있다.

Problem 1) Hard vs. Soft Viterbi Decoding

Simulation 코드

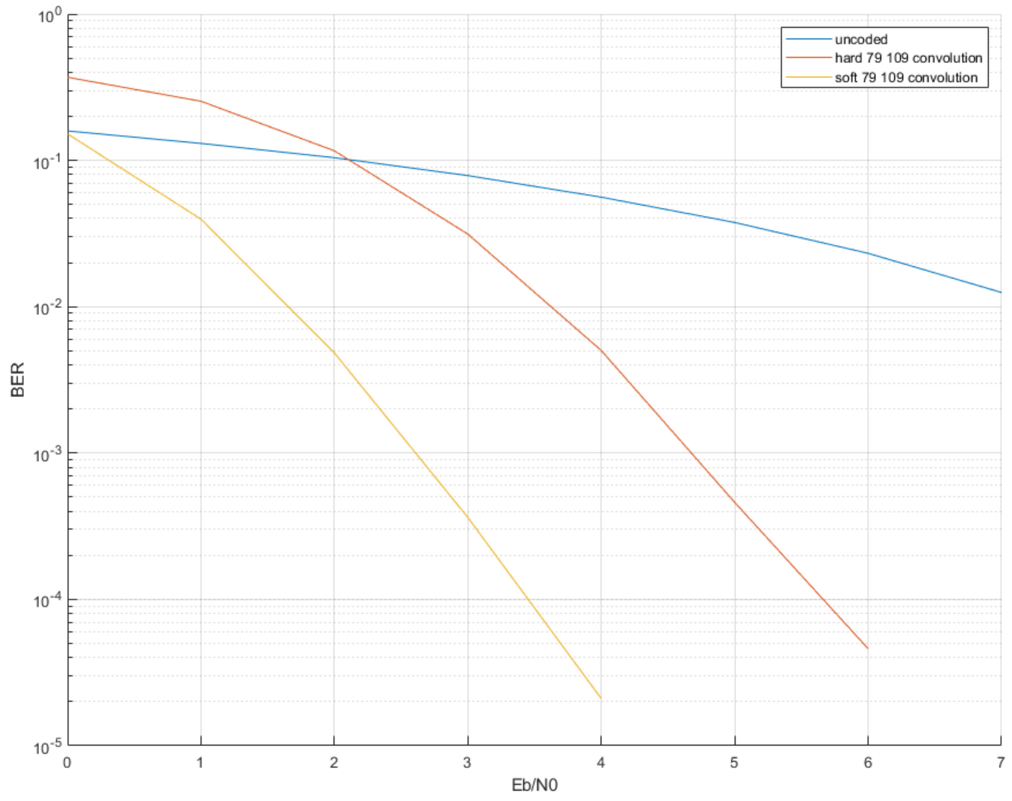


```matlab
%% Simulation BER
function Simulation_BER(bits_num, convolution_rules, QAM, SNRs, mode)
    reg_num = floor(log2(max(convolution_rules)));

    BERs = zeros(1, length(SNRs));
    for i = 1:length(SNRs)
        clc;
        SNR = SNRs(i);
        disp(['SNR:' num2str(SNRs(i)) 'dB'])

        bits = randi(2, [1,bits_num])-1;
        bits((bits_num-reg_num+1):bits_num) = 0; % 마지막 state 0으로 끝내기 위함
        if mode == 'unco'
            encoded_bits = bits;
        else
            encoded_bits = Convolution_Encoder(bits, convolution_rules);
        end
        modulated_symbols = moudulator_M_ary_QAM(encoded_bits, QAM);
        noised_symbols = AWGN_channel(modulated_symbols, SNR, length(convolution_rules));

        if mode == 'unco'
            demodulated_bits = hard_demoudulator_M_ary_QAM(noised_symbols, QAM);
            decoded_bits = demodulated_bits;
        else
            LLRs = soft_demoudulator_M_ary_QAM(noised_symbols, QAM, SNR, length(convolution_rules));
            decoded_bits = Viterbi_Decoder(LLRs, convolution_rules, mode);
        end


        BERs(i) = sum(abs(bits - decoded_bits))/bits_num;
    end

    semilogy(SNRs, BERs)
end
```



```matlab
%% BER

clear; clc
D=2;

figure();
hold on

% uncoded Viterbi Decoding
bits_num = 10^6;
convolution_rule = [1+D+D^2+D^3+D^6, 1+D^2+D^3+D^5+D^6];
QAM = 4;
SNRs = 0:7;
mode = 'unco';

Simulation_BER(bits_num, convolution_rule, QAM, SNRs, mode);

% hard Viterbi Decoding
bits_num = 10^5;
convolution_rule = [1+D+D^2+D^3+D^6, 1+D^2+D^3+D^5+D^6];
QAM = 4;
SNRs = 0:7;
mode = 'hard';

Simulation_BER(bits_num, convolution_rule, QAM, SNRs, mode);

% soft Viterbi Decoding
bits_num = 10^5;
convolution_rule = [1+D+D^2+D^3+D^6, 1+D^2+D^3+D^5+D^6];
QAM = 4;
SNRs = 0:7;
mode = 'soft';

Simulation_BER(bits_num, convolution_rule, QAM, SNRs, mode);

legend('uncoded', 'hard 79 109 convolution', 'soft 79 109 convolution')
xlabel('Eb/N0')
ylabel('BER')
set(gca, 'YScale', 'log');
grid on
hold off
```

결과) SNR= 0~7, QPSK

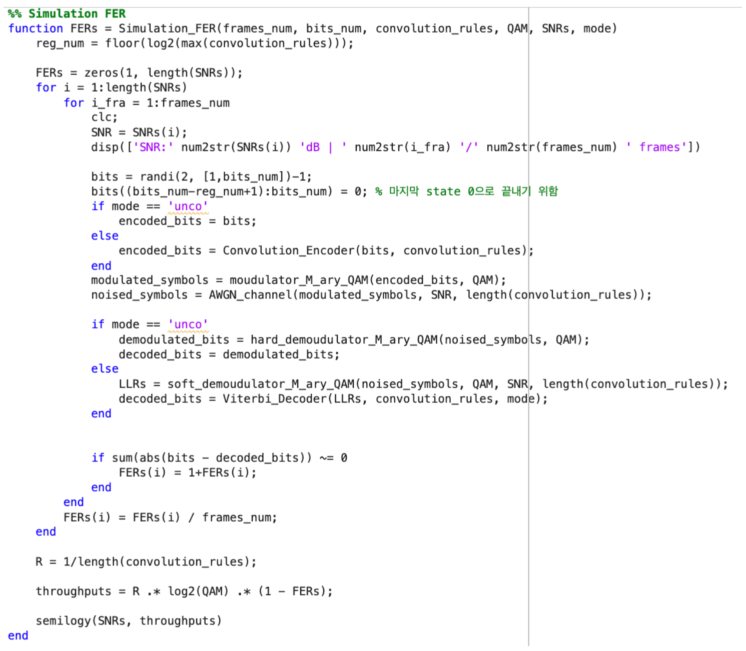


Hard decision decoding 결과 5.6dB 정도의 SNR 이 나오고 soft decision decoding 결과 3.5dB 정도의 SNR 이 나오는 것을 확인할 수 있다.

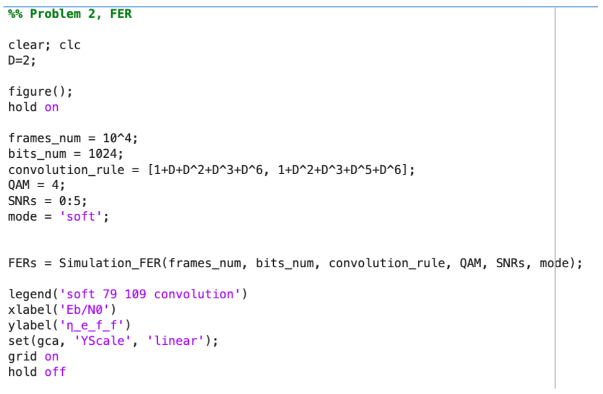
BER 10^-4 에서 대략 2dB 의 code gain 이 생김을 확인할 수 있다.

Problem 2) Throughput Analysis using Frame Error Rate (FER)

Simulation 코드

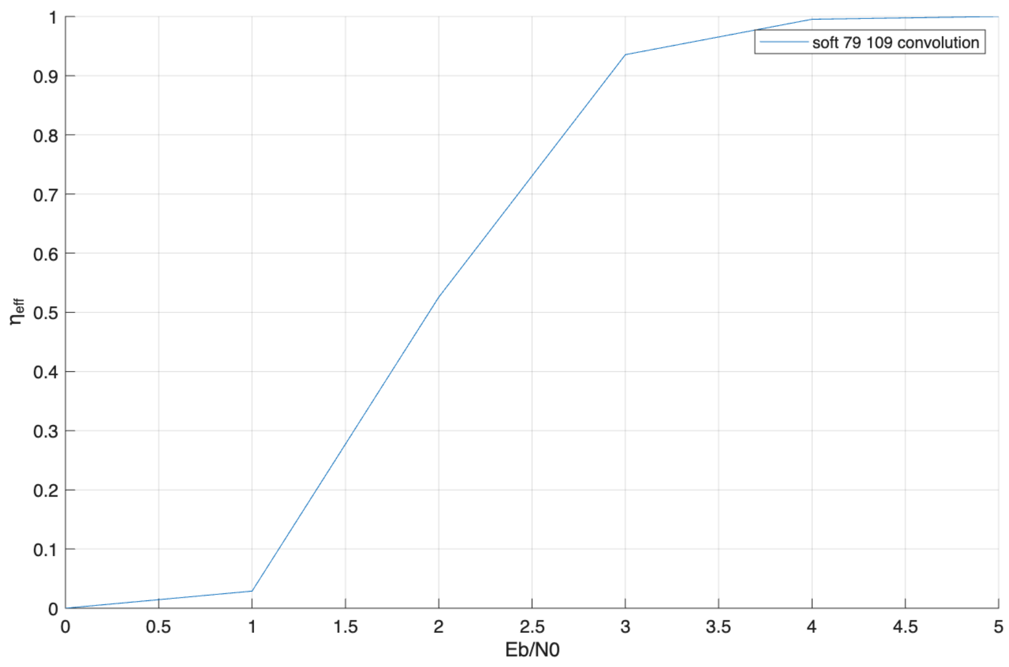


```
%% Simulation FER
function FERs = Simulation_FER(frames_num, bits_num, convolution_rules, QAM, SNRs, mode)
    reg_num = floor(log2(max(convolution_rules)));

    FERs = zeros(1, length(SNRs));
    for i = 1:length(SNRs)
        for i_fra = 1:frames_num
            clc;
            SNR = SNRs(i);
            disp(['SNR:' num2str(SNRs(i)) 'dB | ' num2str(i_fra) '/' num2str(frames_num) ' frames'])

            bits = randi(2, [1,bits_num])-1;
            bits((bits_num-reg_num+1):bits_num) = 0; % 마지막 state 0으로 끝내기 위함
            if mode == 'unco'
                encoded_bits = bits;
            else
                encoded_bits = Convolution_Encoder(bits, convolution_rules);
            end
            modulated_symbols = moudulator_M_ary_QAM(encoded_bits, QAM);
            noised_symbols = AWGN_channel(modulated_symbols, SNR, length(convolution_rules));

            if mode == 'unco'
                demodulated_bits = hard_demoudulator_M_ary_QAM(noised_symbols, QAM);
                decoded_bits = demodulated_bits;
            else
                LLRs = soft_demoudulator_M_ary_QAM(noised_symbols, QAM, SNR, length(convolution_rules));
                decoded_bits = Viterbi_Decoder(LLRs, convolution_rules, mode);
            end


            if sum(abs(bits - decoded_bits)) ~= 0
                FERs(i) = 1+FERs(i);
            end
        end
        FERs(i) = FERs(i) / frames_num;
    end

    R = 1/length(convolution_rules);

    throughputs = R .* log2(QAM) .* (1 - FERs);

    semilogy(SNRs, throughputs)
end
```



```
%% Problem 2, FER

clear; clc
D=2;

figure();
hold on

frames_num = 10^4;
bits_num = 1024;
convolution_rule = [1+D+D^2+D^3+D^6, 1+D^2+D^3+D^5+D^6];
QAM = 4;
SNRs = 0:5;
mode = 'soft';


FERs = Simulation_FER(frames_num, bits_num, convolution_rule, QAM, SNRs, mode);

legend('soft 79 109 convolution')
xlabel('Eb/N0')
ylabel('η_e_f_f')
set(gca, 'YScale', 'linear');
grid on
hold off
```

결과)



SNR 이 작은 곳에서는 FER 이 크게 나오고 η_eff 가 0 에 가깝게 작게 나온다..

SNR 이 큰 곳에서는 FER 이 작게 나오고 η_eff 이 R log_2(M)에 가까워진다.

이처럼 FER 와 η_eff 는 반비례 관계에 있다.

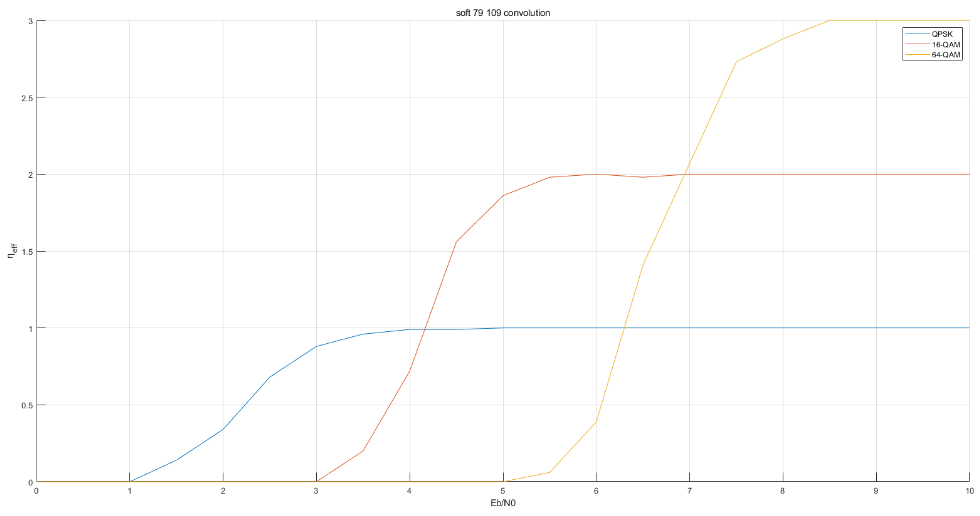
또한 SNR 4dB 에서 부터 η_eff 의 최대인 1 에 매우 근접하게 도달한 것을 보고 4dB 이후부터 신뢰할 수 있는 통신이 가능한 것을 생각할 수 있다.

Problem 3) Adaptive Modulation and Coding (AMC) Efficiency

Simulation code

```
figure();
hold on

% QPSK
frames_num = 10^2;
bits_num = 512*3;
convolution_rule = [1+D+D^2+D^3+D^6, 1+D^2+D^3+D^5+D^6];
QAM = 4;
SNRs = 0:0.5:10;
mode = 'soft';

FERs_QPSK = Simulation_FER(frames_num, bits_num, convolution_rule, QAM, SNRs, mode);

% 16-QAM
frames_num = 10^2;
bits_num = 512*3;
convolution_rule = [1+D+D^2+D^3+D^6, 1+D^2+D^3+D^5+D^6];
QAM = 16;
mode = 'soft';

FERs_16QAM = Simulation_FER(frames_num, bits_num, convolution_rule, QAM, SNRs, mode);

% 64-QAM
frames_num = 10^2;
bits_num = 512*3;
convolution_rule = [1+D+D^2+D^3+D^6, 1+D^2+D^3+D^5+D^6];
QAM = 64;
mode = 'soft';

FERs_64QAM = Simulation_FER(frames_num, bits_num, convolution_rule, QAM, SNRs, mode);

title('soft 79 109 convolution')
legend('QPSK', '16-QAM', '64-QAM')
xlabel('Eb/N0')
ylabel('η_e_f_f')
xlim([0,10])
set(gca, 'YScale', 'linear');
grid on
hold off

% FER plot
figure();
hold on
semilogy(SNRs, FERs_QPSK, SNRs, FERs_16QAM, SNRs, FERs_64QAM);
title('soft 79 109 convolution')
legend('QPSK', '16-QAM', '64-QAM')
xlabel('Eb/N0')
ylabel('FER')
xlim([0,10])
set(gca, 'YScale', 'log');
grid on
hold off
```

결과)



QPSK->16-QAM cross over : SNR=4.2dB

16-QAM->64-QAM cross over : SNR=6.3dB

Appendix(전체 코드)

```
%% Library

function bits = int_to_bit(int, bit_num) % array -> rowwise bits
    int = int';
    bits = zeros(length(int), bit_num);
    for i_regnum = 1:bit_num
        bits(:, i_regnum) = rem(int, 2);
        int = floor(int/2);
    end
end

function ints = bit_to_int(bits) % rowwise bits -> array (rowwise)
    bits_length = length(bits);
    bits_weight = 2.^(0:(bits_length-1));
    ints = bits_weight * bits';
end

function [gm, constellation, bit_to_symbols_lookup_idxp] = graymap(M)
    M_sqrt = sqrt(M);
    g1d = bitxor(0:(M_sqrt-1), bitshift(0:(M_sqrt-1), -1)); % 1d graymap
    gm = g1d + g1d'*M_sqrt; % 2d graymap(i, j) = cat[1d graymap(i), 1d graymap(j)]

    min_dis = sqrt(6*log2(M)/(M-1));

    constellation1d = ((1:M_sqrt)-(1+M_sqrt)/2)*min_dis;
    constellation = constellation1d + 1j * constellation1d';

    bit_to_symbols_lookup_idxp = zeros(M,1);
    for i = 1:M_sqrt
        for j = 1:M_sqrt
            bit_to_symbols_lookup_idxp(gm(i, j)+1) = constellation(i, j);
        end
    end

    % disp(constellation)
    % disp(gm)
    % disp(bit_to_symbols_lookup_idxp)
end
%% Encoders

function encoded_bit = Convolution_Encoder(input_bit, convolution_rules)
    % code : 1x~

    % 필요 수치 정리
    input_length = length(input_bit); % k
    n_over_k = length(convolution_rules); % n/k, codeword length

    % register 정의
    reg_num = floor(log2(max(convolution_rules)));
    reg = zeros(reg_num,1); % columnwise
    disp([num2str(convolution_rules) ' convolution, # of registers:' num2str(reg_num)]);

    % convolution 연산을 위한 rule bit화
    convolution_rules_bits = zeros([length(convolution_rules),reg_num+1]); % rowwise
    for i_conv = 1:length(convolution_rules)
        rule = convolution_rules(i_conv);
        for i_regnum = 1:reg_num+1
            convolution_rules_bits(i_conv, i_regnum) = rem(rule, 2);
            rule = floor(rule/2);
        end
    end

    % convolution encoding
    encoded_bit = zeros(1, input_length*n_over_k);
    for i_bit = 1:input_length
        bit_sequence = [input_bit(i_bit); reg];
        encoded_bit((i_bit-1)*n_over_k+1:i_bit*n_over_k) = rem(convolution_rules_bits*bit_sequence, 2);

        reg = bit_sequence(1:reg_num);
    end

    disp([num2str(input_length) ' bits encoded (' num2str(n_over_k*input_length) ' bits)']);
end

%% Modulator

function modulated_symbols = moudulator_M_ary_QAM(encoded_bits, M)

    encoded_bits_length = length(encoded_bits); % n
    bit_per_symbols = floor(log2(M));
    symbols_length = encoded_bits_length/bit_per_symbols;
    if floor(bit_per_symbols/2) ~= log2(M)/2
        error('M must be power of power of 2^2');
    end

    [~, ~, bit_to_symbols_lookup_idxp] = graymap(M);
    modulated_symbols = zeros(1,symbols_length);
    bit_sampled = reshape(encoded_bits, bit_per_symbols,symbols_length)';

    for i_sym = 1:symbols_length
        symbol_bits = bit_to_int(bit_sampled(i_sym,:));
        modulated_symbols(i_sym) = bit_to_symbols_lookup_idxp(symbol_bits+1);
    end

    disp([num2str(symbols_length) ' symbols modulated (' num2str(M) '-ary QAM)']);
end

%% AWGN Channel

function noised_symbols = AWGN_channel(symbols, SNR, n_over_k)
    disp('AWGN channel added')
    SNR_linear = 10^(SNR/10);

    sigma = sqrt(0.5/SNR_linear*n_over_k);
    noise_bit = randn([1, length(symbols)]) * sigma + 1j*randn([1, length(symbols)]) * sigma;

    noised_symbols = symbols + noise_bit;
end

%% Demodulator

function demodulated_bits = hard_demoudulator_M_ary_QAM(symbols, M)
    sym_length = length(symbols);
    symbol_bit_num = log2(M);

    disp([num2str(sym_length) ' symbols demodulated (' num2str(sym_length*symbol_bit_num) 'bits)']);

    min_dis = sqrt(6*sqrt(M)/(M-1));
    [gray_map, constellation, bit_to_symbols_lookup_idxp] = graymap(M);
    symbols = symbols/min_dis;
    symbols = symbols+0.5+0.5j;
    symbols = round(symbols);
    symbols = symbols-0.5-0.5j;
    symbols = symbols-constellation(1,1)/min_dis+1+1j;


    M_sqrt = sqrt(M);
    demodulated_bits = zeros(1, sym_length*symbol_bit_num);
    for idx_sym = 1:sym_length
        sym = symbols(idx_sym);
        i = max(1, min(sqrt(M), round(imag(sym))));
        j = max(1, min(sqrt(M), round(real(sym))));


        demodulated_bits((idx_sym-1)*symbol_bit_num+1:idx_sym*symbol_bit_num) = int_to_bit(gray_map(i, j), symbol_bit_num);
    end
end

function LLRs = soft_demoudulator_M_ary_QAM(symbols, M, SNR, n_over_k)
    sym_length = length(symbols);
    SNR_linear = 10^(SNR/10);
    sigma = sqrt(0.5/SNR_linear*n_over_k);
    M_log2 = log2(M);

    %min_dis = sqrt(6*sqrt(M)/(M-1));
    [gray_map, constellation, ~] = graymap(M);

    LLRs = zeros(1, sym_length * M_log2);
    for i_sym = 1:sym_length
        %symbol = symbols(i_sym);
        symbols_distances = abs(constellation - symbols(i_sym));
        symbols_distance_square = symbols_distances .* conj(symbols_distances);

        sb0 = zeros(1, M_log2);
        sb1 = zeros(1, M_log2);

        for i_gm = 1:M
            sym = gray_map(i_gm);
            symbol_bits = int_to_bit(sym, log2(M));
            for i_sb = 1:M_log2
                if symbol_bits(i_sb) == 0
                    sb0(i_sb) = sb0(i_sb) + exp(-symbols_distance_square(i_gm)/(2*sigma^2));
                else
                    sb1(i_sb) = sb1(i_sb) + exp(-symbols_distance_square(i_gm)/(2*sigma^2));
                end
            end
        end
        LLRs(((i_sym-1)*M_log2+1):i_sym*M_log2) = log(sb1./sb0);
    end

end

%% Decoder

function received_coded_bits = Viterbi_Decoder(LLRs, convolution_rules, mode)
    reg_num = floor(log2(max(convolution_rules)));
    N = length(convolution_rules); %codeword size
    branches = zeros(2^reg_num, 2);

    disp([num2str(length(LLRs)/N) ' bits decoded (' mode ' decoding)']);

    % convolution 연산을 위한 rule bit화
    convolution_rules_bits = zeros([length(convolution_rules),reg_num+1]); % rowwise
    for i_conv = 1:length(convolution_rules)
        rule = convolution_rules(i_conv);
        for i_regnum = 1:reg_num+1
            convolution_rules_bits(i_conv, i_regnum) = rem(rule, 2);
            rule = floor(rule/2);
        end
    end

    for state = 0:2^reg_num-1
        for bit = 0:1
            bsq = [bit, int_to_bit(state,reg_num)];
            rem(convolution_rules_bits*bsq', 2)';
            codeword = bit_to_int(rem(convolution_rules_bits*bsq', 2)');
            branches(state+1, bit+1) = (bit_to_int(bsq(1:reg_num))+1)*2^N+codeword;
        end
    end

    % branches

    % branches = [
    %     1*4+0,2*4+3;
    %     3*4+1,4*4+2;
    %     1*4+3,2*4+0;
    %     3*4+2,4*4+1
    %     ]; %branches(state+1, input+1) = (next_state+1)*4+codeword
```

```matlab
    state_num = size(branches, 1);
    PM = zeros(state_num, length(LLRs)/N+1)+inf;
    PM(1,1) = 0;
    PM_input = zeros(state_num, length(LLRs)/N);
    PM_pre_state = zeros(state_num, length(LLRs)/N);

    for idx = 1:(length(LLRs)/N)
        LLR = LLRs((idx-1)*N+1:idx*N);
        for state = 1:state_num
            for bit = 0:1
                temp = branches(state, bit+1);
                next_state = (temp-mod(temp, (2^N)))/(2^N);
                if idx == (length(LLRs)/N) && next_state ~= 1 % 마지막 state 0 으로 끝내기 위함
                    continue
                end

                codeword = int_to_bit(mod(temp, (2^N)),N);
                distance = 0;
                if mode == 'hard'
                    distance = sum(abs((LLR>0)*1-codeword));
                elseif mode == 'soft'
                    distance = sum((0.5-codeword).*LLR);
                end

                PM_next = PM(state, idx) + distance;
                if PM(next_state, idx+1) > PM_next
                    PM(next_state, idx+1) = PM_next;
                    PM_input(next_state, idx) = bit;
                    PM_pre_state(next_state, idx) = state;
                end
            end
        end
    end

    received_coded_bits = zeros(1, length(LLRs)/N);
    min_state = 1; %% 마지막 □state 00
    for idx = (length(LLRs)/N):-1:1
        received_coded_bits(idx) = PM_input(min_state, idx);
        min_state = PM_pre_state(min_state, idx);
    end


end

%% Simulation BER
function Simulation_BER(bits_num, convolution_rules, QAM, SNRs, mode)
    reg_num = floor(log2(max(convolution_rules)));

    BERs = zeros(1, length(SNRs));
    for i = 1:length(SNRs)
        clc;
        SNR = SNRs(i);
        disp(['SNR:' num2str(SNRs(i)) 'dB'])

        bits = randi(2, [1,bits_num])-1;
        bits((bits_num-reg_num+1):bits_num) = 0; % 마지막 state 0 으로 끝내기 위함
        if mode == 'unco'
            encoded_bits = bits;
        else
            encoded_bits = Convolution_Encoder(bits, convolution_rules);
        end
        modulated_symbols = moudulator_M_ary_QAM(encoded_bits, QAM);
        noised_symbols = AWGN_channel(modulated_symbols, SNR, length(convolution_rules));

        if mode == 'unco'
            demodulated_bits = hard_demoudulator_M_ary_QAM(noised_symbols, QAM);
            decoded_bits = demodulated_bits;
        else
            LLRs = soft_demoudulator_M_ary_QAM(noised_symbols, QAM, SNR, length(convolution_rules));
            decoded_bits = Viterbi_Decoder(LLRs, convolution_rules, mode);
        end



        BERs(i) = sum(abs(bits - decoded_bits))/bits_num;
    end

    semilogy(SNRs, BERs)
end

%% Simulation FER
function FERs = Simulation_FER(frames_num, bits_num, convolution_rules, QAM, SNRs, mode)
    reg_num = floor(log2(max(convolution_rules)));

    FERs = zeros(1, length(SNRs));
    for i = 1:length(SNRs)
        for i_fra = 1:frames_num
            clc;
            SNR = SNRs(i);
            disp(['SNR:' num2str(SNRs(i)) 'dB | ' num2str(i_fra) '/' num2str(frames_num) ' frames'])

            bits = randi(2, [1,bits_num])-1;
            bits((bits_num-reg_num+1):bits_num) = 0; % 마지막 state 0 으로 끝내기 위함
            if mode == 'unco'
                encoded_bits = bits;
            else
                encoded_bits = Convolution_Encoder(bits, convolution_rules);
            end
            modulated_symbols = moudulator_M_ary_QAM(encoded_bits, QAM);
            noised_symbols = AWGN_channel(modulated_symbols, SNR, length(convolution_rules));

            if mode == 'unco'
                demodulated_bits = hard_demoudulator_M_ary_QAM(noised_symbols, QAM);
                decoded_bits = demodulated_bits;
            else
                LLRs = soft_demoudulator_M_ary_QAM(noised_symbols, QAM, SNR, length(convolution_rules));
                decoded_bits = Viterbi_Decoder(LLRs, convolution_rules, mode);
            end


            if sum(abs(bits - decoded_bits)) ~= 0
                FERs(i) = 1+FERs(i);
            end
        end
        FERs(i) = FERs(i) / frames_num;
    end

    R = 1/length(convolution_rules);

    throughputs = R .* log2(QAM) .* (1 - FERs);

    semilogy(SNRs, throughputs)
end

%% Problem 1, BER

clear; clc
D=2;

figure();
hold on

% uncoded Viterbi Decoding
bits_num = 10^6;
convolution_rule = [1+D+D^2+D^3+D^6, 1+D^2+D^3+D^5+D^6];
QAM = 4;
SNRs = 0:7;
mode = 'unco';

Simulation_BER(bits_num, convolution_rule, QAM, SNRs, mode);

% hard Viterbi Decoding
bits_num = 10^5;
convolution_rule = [1+D+D^2+D^3+D^6, 1+D^2+D^3+D^5+D^6];
QAM = 4;
SNRs = 0:7;
mode = 'hard';

Simulation_BER(bits_num, convolution_rule, QAM, SNRs, mode);

% soft Viterbi Decoding
bits_num = 10^5;
convolution_rule = [1+D+D^2+D^3+D^6, 1+D^2+D^3+D^5+D^6];
QAM = 4;
SNRs = 0:7;
mode = 'soft';

Simulation_BER(bits_num, convolution_rule, QAM, SNRs, mode);

legend('uncoded', 'hard 79 109 convolution', 'soft 79 109 convolution')
xlabel('Eb/N0')
ylabel('BER')
set(gca, 'YScale', 'log');
grid on
hold off

%% Problem 2, FER

clear; clc
D=2;

figure();
hold on

frames_num = 10^4;
bits_num = 1024;
convolution_rule = [1+D+D^2+D^3+D^6, 1+D^2+D^3+D^5+D^6];
QAM = 4;
SNRs = 0:5;
mode = 'soft';


FERs = Simulation_FER(frames_num, bits_num, convolution_rule, QAM, SNRs, mode);

legend('soft 79 109 convolution')
xlabel('Eb/N0')
ylabel('η_e_f_f')
set(gca, 'YScale', 'linear');
grid on
hold off

%% Problem 3, Adaptive Modulation and Coding (AMC) Efficiency

clear; clc
D=2;

figure();
hold on

% QPSK
frames_num = 10^2;
bits_num = 512*3;
convolution_rule = [1+D+D^2+D^3+D^6, 1+D^2+D^3+D^5+D^6];
QAM = 4;
SNRs = 0:0.5:10;
mode = 'soft';

FERs_QPSK = Simulation_FER(frames_num, bits_num, convolution_rule, QAM, SNRs, mode);

% 16-QAM
frames_num = 10^2;
bits_num = 512*3;
convolution_rule = [1+D+D^2+D^3+D^6, 1+D^2+D^3+D^5+D^6];
QAM = 16;
mode = 'soft';
```

```matlab
FERs_16QAM = Simulation_FER(frames_num, bits_num, convolution_rule, QAM, SNRs, mode);

% 64-QAM
frames_num = 10^2;
bits_num = 512*3;
convolution_rule = [1+D+D^2+D^3+D^6, 1+D^2+D^3+D^5+D^6];
QAM = 64;
mode = 'soft';

FERs_64QAM = Simulation_FER(frames_num, bits_num, convolution_rule, QAM, SNRs, mode);

title('soft 79 109 convolution')
legend('QPSK', '16-QAM', '64-QAM')
xlabel('Eb/N0')
ylabel('η_e_f_f')
xlim([0,10])
set(gca, 'YScale', 'linear');
grid on
hold off

% FER plot
figure();
hold on
semilogy(SNRs, FERs_QPSK, SNRs, FERs_16QAM, SNRs, FERs_64QAM);
title('soft 79 109 convolution')
legend('QPSK', '16-QAM', '64-QAM')
xlabel('Eb/N0')
ylabel('FER')
xlim([0,10])
set(gca, 'YScale', 'log');
grid on
hold off
```