Digital Communication Matlab Assignment 4

Name: Seongjoon Kang

In the check nodes, to estimate $P(c_i = 1)$ we use the following theorem:

Given the n independent binary random variables, C = { $C_1, C_2, ...C_n$ } where $P(C_i = 1) = p_i$,

then, P ($\sum_{i=1}^{n} C_i = 1$) = $\frac{1}{2} - \frac{1}{2}\prod_{i=1}^{n}(1 - 2p_i)$**.**

Whenever we update $P^k(C_i = 1)$ **at** $k'th$ **check node**, we compute probability and send it **to** ith **codeword bits**

$P^k(C_i = 1)$ = P($\sum_{j=1,\ j\neq i}^{m} C_j = 1$) = $\frac{1}{2} - \frac{1}{2}\prod_{j=1,\ j\neq i}^{m}(1 - 2Q^k(C_j = 1))$

In $i'th$ **codeword bites**, we estimate $Q^k(C_i = 1)$ in the same way of the lecture note and send it back **to the** $k'th$ **check node**.

$$Q^k(C_i = 1) = \frac{P_0 \prod_{k=1,\ k\neq i}^{m} P^k(C_i = 1)}{P_0 \prod_{k=1,\ k\neq i}^{m} P^k(C_i = 1) + (1 - P_0) \prod_{k=1,\ k\neq i}^{m} (1 - P^k(C_i = 1))}$$

Here m is the number of check node connected message bit, i

We will iterate the above two procedures to compute $P^k(C_i = 1)$ and $Q^k(C_i = 1)$

And the final probability of $i'th$ bit is given as the following

$$P(C_i = 1) = \frac{P_0 \prod_{k=1}^{m} P^k(C_i = 1)Q^k(C_i = 1)}{P_0 \prod_{k=1}^{m} P^k(C_i = 1)Q^k(C_i = 1) + (1 - P_0) \prod_{k=1}^{m} (1 - P^k(C_i = 1)Q^k(C_i = 1))}$$

Here , P_0 and 1 - P_0 are prior probabilities of one and zero.

In the code, we use n = 500, $r = \frac{3}{4}$ and make matrix H sparser as possible.

The following code is the implementation of LDPC decoding algorithm based on the above formulas.

```
%% homework 4
clear all;

n = 500;
r=3/4;
p = 1/36;
k = n*r;
% Use the predefined function to obtain full rank matrix
H2 = GetFullRankMatrix (n-k, n-k,p);
%Generate H1 in the same way
H1 = GetFullRankMatrix(n-k, k,p);
```

```
H1gf = gf( H1,1);
H2gf = gf (H2,1);
inv_H2 = (inv(H2gf)==1);
H = [H1,H2]; % n-k * n matrix

code_book = double.empty(0,n);
code_book_size = 10000
  for m =1:code_book_size % genenrate 10000 message bits and the corresponding codeword
        % from b_1 to b_k are codewords from c_1 to c_k
        message_bits = randi([0,1],k,1);
        % from c_k+1 to c_n are computed by solving linear equations
        code_bits = inv(H2gf)*H1gf*gf(message_bits,1);
        code_bits = (code_bits == 1);
        code_bits = [message_bits; code_bits]; % total codewords
        code_book = [code_book; code_bits.']; % row vectors are codewords
  end
%%
 decoding_error = 0; rx_error = 0;
 %correct probability set
 prob_set = [0.96,0.965, 0.97,0.975, 0.98,0.985, 0.99, 0.995];
 decoding_error_set = double.empty(0,length(prob_set));
 received_error_set = double.empty(0,length(prob_set));
 norm_set = double.empty(); prev_value = double.empty(); iteration_set = int16.empty();
for p_index=1:length(prob_set)
    p0 = prob_set(p_index);
for trial = 1: 10000 % transmit 10000 independent codewords
      % let's select a codeword from codebooks randomly
      %random_index = randi(code_book_size);
      % or we can just send all codewords of codebook one by one
      rx_codeword = code_book(trial,:);
      original_message = rx_codeword;
      for j = 1: n
           if (rand(1,1) > p0); %% change codewords bits with probability 1-p
             if (rx_codeword(1,j) == 1);
                    rx_codeword(1,j) =0; % if bit is 1, then it becomes 0
                else
                    rx_codeword(1,j) = 1;
             end
           end
      end % end of sending a codeword via BSC channel

     %let's decode the received codewords
    rx_error =rx_error+ sum(xor(original_message, rx_codeword))/n;

     % change the received code word to probabilities of one
     % we only consier the probability that each bit equals 1
    % p(1 | 1) = correct probability; 1 sent, 1 receive
    rx_codeword (rx_codeword == 1) =p0;
    % p(1 | 0) = 1- correct probability; 0 sent, 1 receive
    rx_codeword (rx_codeword == 0) =1-p0;
    p0_set = rx_codeword;
    est_at_codewords = repmat (rx_codeword,n-k,1); %estimation at codewords
    % check node estimation probability of one
    %generate (n-k * n) probability matrix that each row indicates check node index
```

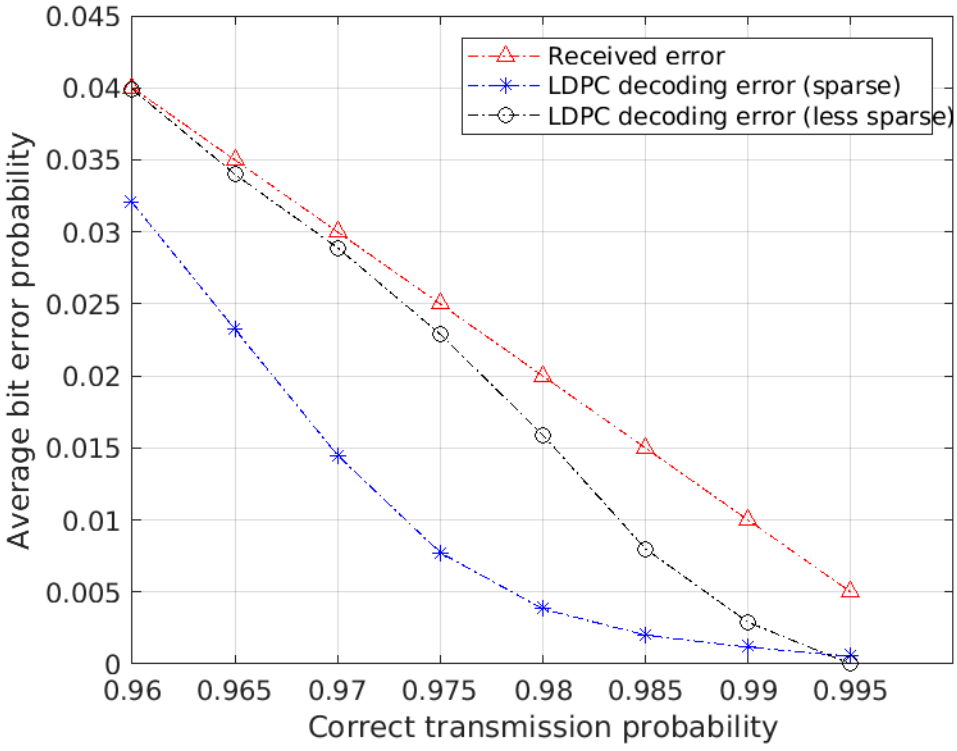
```
    est_at_checknodes = zeros(n-k,n)*1.0; % estimation at checknodes

 for iterate = 1: 100 % iterate 100 times to make probabilities converge
    % update probabilities at each check node
    for i =1 : n-k % there are n-k check nodes
       %each row of parity check matrix, H, is check nodes
       connected_bits = find (H(i,:)==1); % codeword bits connected to check node
           for j = 1:length(connected_bits);
               % update est_at_checknodes(i,check(j)) !
               prod = 1.0;
               for m = 1:length (connected_bits)
                   if m ~= j
                     prod = prod *(1 -2*est_at_codewords(i, connected_bits(m)));
                   end
               end
               % probabilities connected to i'th check node
               est_at_checknodes(i,connected_bits(j)) = 1/2 - 1/2*prod;
           end
    end

   % update probability of one at each of codewords
   for i = 1:n % for each codeword bit
       % find the check nodes that connect codeword bit
       check_nodes = find (H(:,i)==1); % check_nodes(w) is the index of check node
       for j = 1 : length(check_nodes) % for each check node connected to i-th codewor
           prod =1.0; prod_2 = 1.0;
        % compute probability products p_w except p_j
           for w = 1:length(check_nodes)
               if w ~= j
                  prod = prod * est_at_checknodes(check_nodes(w),i);
                  prod_2 = prod_2 * (1-est_at_checknodes(check_nodes(w),i));
               end
           end
             % update and normalize p(1) for each check node
             est_at_codewords(check_nodes(j),i) = p0_set(i)*prod /(p0_set(i)* ...
                 prod + (1-p0_set(i))*prod_2);
       end
   end

 decoded = double.empty();
%check convergence with norm of difference of two consecutive decoding vectors
  if trial == 1 && p0==0.97 % only consider one case since others are the same
       for i =1 : n % for each codeword bit
           % find checknodes connected to codeword
           check_nodes = find (H(:,i)==1);
            prod =1.0; prod_2 = 1.0; w =1;
           % compute probability products p_w
           for w = 1:length(check_nodes)
                 % here check_nodes(w) is the index of check node
                 prod = prod * est_at_checknodes(check_nodes(w),i) * ...
                 est_at_codewords(check_nodes(w),i);
                 prod_2 = prod_2 * (1-est_at_checknodes(check_nodes(w),i))* ...
                 (1-est_at_codewords(check_nodes(w),i));
           end
```

```
            decoded(i) = p0_set(i)*prod/(p0_set(i)*prod +(1-p0_set(i))*prod_2);
        end
            if  length(prev_value) ~= 0
                norm_set =[norm_set,norm(decoded - prev_value)];
                iteration_set = [iteration_set,iterate];
                prev_value = decoded;
            else
                prev_value = decoded;
            end
     end

  end % end of iteration loop

    %after enough iterations, we obtain the final decoded bit
    for i =1 : n % for each codeword bit
        % find checknodes connected to codeword
         check_nodes = find (H(:,i)==1);
         prod =1.0; prod_2 = 1.0; w =1;
         % compute probability products p_w
         for w = 1:length(check_nodes)
               % check_nodes(w) is the index of check node
               prod = prod * est_at_checknodes(check_nodes(w),i) ...
               * est_at_codewords(check_nodes(w),i);
               prod_2 = prod_2 * (1-est_at_checknodes(check_nodes(w),i))* ...
               (1- est_at_codewords(check_nodes(w),i));
          end
            decoded(i) = p0_set(i)*prod/(p0_set(i)*prod +(1-p0_set(i))*prod_2);
     end
    copy_decode = decoded.';
    decoded (decoded >= 0.5) =1; % convert probabilities to bits
    decoded (decoded < 0.5) =0;
    biterror = xor(original_message, decoded);
    decoding_error =decoding_error + sum(biterror)/n;
end % end of trial loop
decoding_error_set = [decoding_error_set,decoding_error / trial];
received_error_set = [received_error_set, rx_error/trial];
decoding_error = 0;
rx_error =0;
end % end of  prob_set loop
%%
decoding_error_set
plot(prob_set, received_error_set, "r^-.");
hold on
grid on
plot(prob_set, decoding_error_set, "b*-.");

xlabel ("Correct transmission probability");
ylabel ("Average bit error probability");
legend ("Received error", "LDPC decoding error");
hold off

figure(2)

plot(iteration_set,norm_set,"b-",'LineWidth',3);
```

```
hold on
grid on
title ("Convergence vs iterations")
ylabel ("Norm of difference (||v_i - v_{i-1}||_2^{2} )")
xlabel("Iterations");

%-------------------------------------------------%
% This is the function returning full rank matrix;
% Rank (H) = min (# of column, # of rows)
```

```
function FullRank_H = GetFullRankMatrix (a,b,p)
     while (1)
        H = rand(a, b);
        H = H > 1-p; % every element has probability p of one
        if gfrank(H)== min(a,b);
            break;
        end
     end
    FullRank_H = H;
end
```

The below figure shows the average bit-error probability as a function of the BSC correct-transmission probability after 10000 trials of sending code words via BSC.



As we can see this plot, LDPC decoding using sparse parity check matrix ($n = 500, p = 1/36$) reduces average bit error rate generated via BSC. But LDPC decoding doesn't work well with less sparse parity check matrix ($n = 500, p = 1/12$) .

From this plot, we observe that LDPC decoding corrects bit errors well when the channel capacity is high enough to support the coding rate and the parity check matrix is sparser.

(correct transmission probability $\geq$ 0.97)

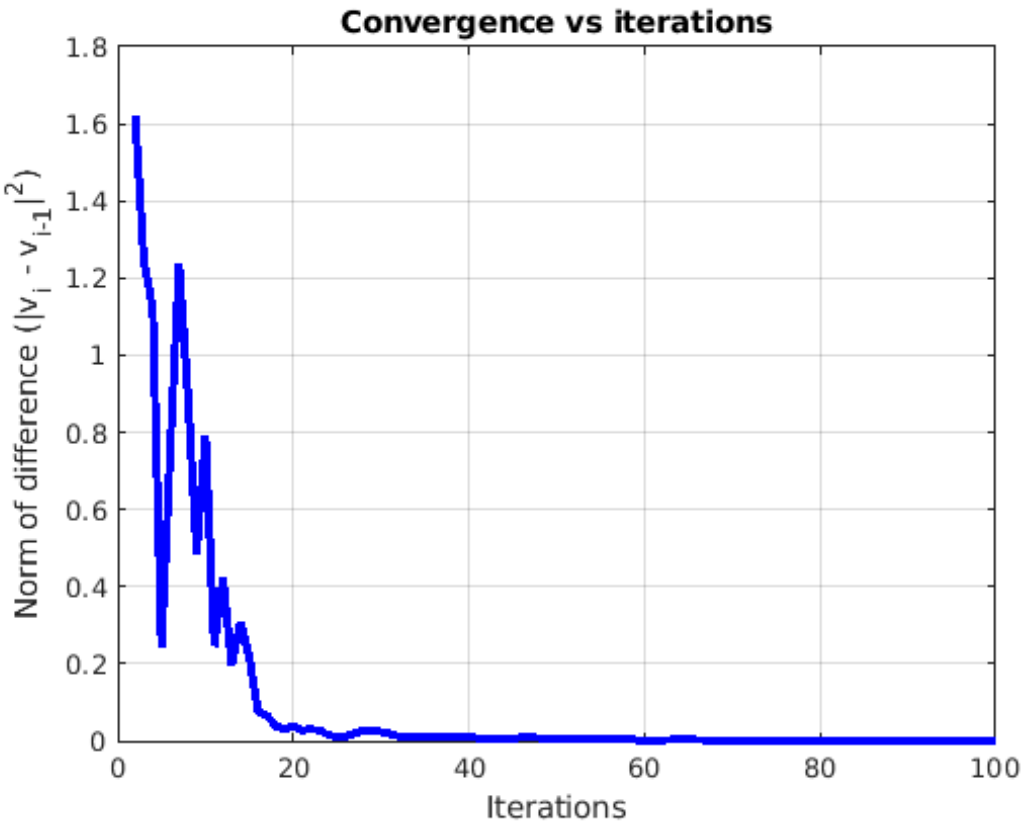
To confirm the convergence of probabilities in LDPC decoding, we employ the vector norm of difference between two consecutive decoding vectors meaning the final probabilities.

Suppose that we obtain a final probability vector, v_i , after $i'th$ iteration.

If the final probabilities converge after some iteration, then $l2$ norm, $\|v_i - v_{i-1}\|_2^2$, will approach zero.

In the matlab code, we compute the norm, $\|v_i - v_{i-1}\|_2^2$, only for one case since other cases have the same tendencies.

We plot it as a function of iterations.



This plot shows that the norm of difference between the consecutive final probabilities approaches 0, which means the final probabilities converge.

The below picture shows some part of the final probability values which converge to roughly 0 or 1.

	1	2	3	4	5	6	7	8	9	10
1	1.0000	0.0000	1.0000	0.0300	0.9994	1.0000	0.9995	0.9700	1.0000	1.0000
2	1.0000	0.0006	1.0000	0.0000	0.0004	0.0004	1.0000	0.0068	0.9994	1.0000
3	0.9994	0.0000	0.0000	0.0000	0.0000	0.9996	0.0000	0.0000	0.0300	1.0000
4	0.0000	0.0001	0.0008	0.0000	1.0000	1.0000	0.0000	0.0000	0.0000	0.0000
5	1.0000	0.0000	1.0000	0.0000	0.0000	1.0000	1.0000	1.0000	0.0000	0.0003
6	0.0034	0.9997	0.9999	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0001
7	0.0001	0.0003	1.0000	0.0000	0.0011	0.9998	0.0000	0.9997	0.0300	1.0000
8	1.0000	1.0000	0.0010	1.0000	1.0000	1.0000	0.0000	1.0000	0.0000	0.0002
9	1.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0300	1.0000	0.0000	1.0000
10	0.0000	0.0016	0.0000	1.0000	0.0000	1.0000	0.9700	1.0000	1.0000	1.0000
11	1.0000	0.0000	0.0068	0.0000	1.0000	1.0000	0.0000	0.0000	0.0007	0.0000
12	0.0067	0.9864	1.0000	1.0000	0.0005	0.0000	0.0000	0.0000	1.0000	0.9993
13	0.9434	0.0000	1.0000	0.0000	0.0003	0.0000	0.0000	0.0000	0.0000	0.0000
14	0.0000	0.0000	0.9993	1.0000	1.0000	1.0000	0.9932	0.0000	1.0000	1.0000
15	1.0000	1.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000	0.9964	0.0000
16	1.0000	1.0000	1.0000	0.0105	1.0000	1.0000	0.0000	1.0000	0.0000	1.0000

We conclude that the final probabilities computed by LDPC decoding algorithm converge to 0 or 1 after sufficient iterations.