Code file

Honour code :-

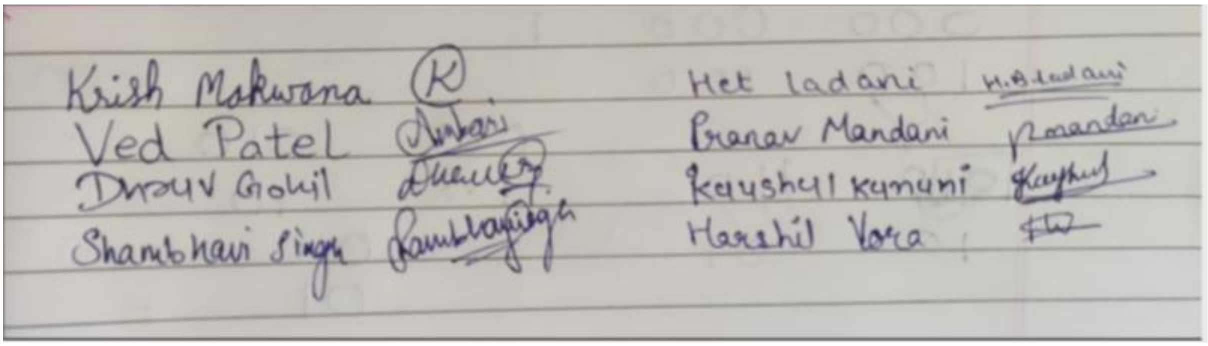
We declare that

→ The work that we are presenting is our own work.

→ We have not copied the work (the code, the results, etc.) that someone else has done.

→ Concepts, understanding and insights we will be describing are our own.

→ We make this pledge truthfully. We know that violation of this solemn pledge can carry grave consequences.



Krish Makwana	K	Het ladani	H.B.ladani
Ved Patel		Pranav Mandani	Pmandani
Dhruv Gohil		kaushal kanani	Kaushal
Shambhavi Singh		Harshil Vora	

INDEX

Encoder

```
function out = encoder(ip, gen)
    [~, len] = size(ip);
    [nglen, glen] = size(gen);
    out = zeros(1, nglen*len); % declare output
    tem = zeros(1, glen-1);    % declare and attach padding
    ip = [tem ip];
    itr = 1;
    [~, len] = size(ip);
    for i = glen:len          % start iterating after the zeros
      for j = 1:nglen        % Iterate over all generator matrices
        jtr = 1;
        sum = 0;
        for k = i-glen+1: i    % Iterate through all the blocks for generator
          if(gen(j, jtr) == 1)   % if gen ==1 and ip == 1 ++sum
            sum = sum + ip(1, k);
          end
          jtr = jtr +1;
        end
```

```
            out(1, itr) = mod(sum, 2); % get 2 mod sum and attach to output
            itr  = itr +1;
         end
      end

end
```

BPSK

```
function out = bpsk(encodedbits)
        out = zeros(1, numel(encoded_bits_2));
        for i = 1:numel(encodedbits)
                 out(i) = 1 - 2*encodedbits(i);        % Map 1 to -1 and 0 to 1
         end
end
```

AWGN

```
function noisy_msg = pass_msg(s,sigma_n)
    [~,col_s]=size(s);
    % Noise Creation
    us = randn([1,col_s]);
    noise=sigma_n * us;
    % Noise AdditionBm1
    noisy_msg=s+noise;
end
```

Hard-Decoder

```
function dec = decoder(ip, gen)
    [~, len] = size(ip);
    [nglen, glen] = size(gen);
    prev = zeros(2^(glen-1), (len/nglen) + 1);   % initialize previous for backtrack
    metric = zeros(2^(glen-1), (len/nglen)+1);  % path metric initialize
    maxi = len+10;             % max value in pathlen
    for i = 1:2^(glen-1)       % go through all metrics and initialize with default
       for j = 1:(len/nglen)+1
           metric(i, j) = maxi;
           prev(i,j) = -1;
       end
    end
    metric(1, 1) = 0;          % metric for 0,0 = 0
    setn = zeros(2^(glen-1), glen-1);          % Initialize states possible
    for i = 1:2^(glen-1)                       % all states are binary representations
       numi = i-1;                             % of numbers from 0 to 2^(glen-1) -1
       ptr = glen-1;
       for j = 1:(glen-1)                      % The numbers are stored in reverse
```

```
        temi = mod(numi, 2);              % binary order but generator decimal
        setn(i, j) = temi;                % is also reverse compensating
        numi = floor(numi/2);
        ptr = ptr-1;
      end
    end
    num = 2^(glen-1);
    powo = 2^(glen-1);
% adding this and dividing the number by 2 will give us the state when we use
branch 1
    for i = 1:len/nglen         % iterate through all input parts
      in = (i-1)*nglen + 1;
      for nu = 0:num-1          % going through each part
        for fir = 0:1           % Is it 0 or 1
          sum = 0;
          for j = 1:nglen       %For each generator
            numi = 0;
            for k = 1:glen-1   %go through generator
              if(gen(j,k) == 1 && setn(nu+1, k) == 1)
                numi = numi+1;
              end
            end
            if(gen(j, glen)== 1 && fir == 1)
              numi=numi+1;
            end
            if(mod(numi,2) ~= ip(1, in+j-1))          % expected and received
              sum = sum+1;                             % values don't match
```

```
               end
            end
            if(fir == 0)                                        % path metric in next state
               ori = metric(floor(nu/2) + 1, i+1);
              metric(floor(nu/2) + 1, i+1) = min(metric(nu+1, i)+sum, metric(floor(nu/2)+1, i+1));
               if(ori ~= metric(floor(nu/2) + 1, i+1))
                  prev(floor(nu/2)+1, i+1) = nu;          % set for backtrack
               end
            else
               temi = powo+nu;
               ori = metric(floor(temi/2) + 1, i+1);
               metric(floor(temi/2) + 1, i+1) =min(metric(nu+1, i)+sum, metric(floor(temi/2)+1, i+1));
               if(ori ~= metric(floor(temi/2) + 1, i+1))
                  prev(floor(temi/2)+1, i+1) = nu;       % set for backtrack
               end
            end
         end
      end
   end
% Back tracking part
   mintu = maxi;
   minti = -1;
   for i = 1:2^(glen-1)                     % get lowest metric from last state
      if(mintu > metric(i, len/nglen +1))
         mintu = metric(i,len/nglen +1);
         minti = i-1;
      end
```

```
    end

    dec = zeros(1, len/nglen);
    j = len/nglen;
    for i = 1:len/nglen              % backtrack from the lowest metric
        if(minti >= 2^(glen-2))       % only then is the MSB 1
            dec(1,j) =1;
        end
        minti = prev(minti+1, j+1);
        j = j-1;
    end
end
```

For soft decision decoding the entire code remains the same only metric computation changes here is it’s code

Soft decision decoding

```
function dec = soft_decoder(ip, gen)
    [~, len] = size(ip);
    [nglen, glen] = size(gen);
    prev = zeros(2^(glen-1), (len/nglen) + 1);
    metric = zeros(2^(glen-1), (len/nglen)+1);
    maxi = len*10000;
    for i = 1:2^(glen-1)
    for j = 1:(len/nglen)+1
        metric(i, j) = maxi;
        prev(i,j) = -1;
    end

end
metric(1, 1) = 0;
setn = zeros(2^(glen-1), glen-1);
for i = 1:2^(glen-1)
    numi = i-1;
    ptr = glen-1;
    for j = 1:(glen-1)
        temi = mod(numi, 2);
        setn(i, j) = temi;
        numi = floor(numi/2);
        ptr = ptr-1;
    end
end
num = 2^(glen-1);
powo = 2^(glen-1);
for i = 1:len/nglen %For Each branch
    in = (i-1)*nglen + 1;
    for nu = 0:num-1% Iterate through all possible states

        for fir = 0:1% Is it 0 or 1
            sum = 0;
```

```
for j = 1:nglen %For each generator
    numi = 0;
    for k = 1:glen-1 %go through generator
        if(gen(j,k) == 1 && setn(nu+1, k) == 1)
            numi = numi+1;
        end
    end
    if(gen(j, glen)== 1 && fir == 1)
        numi=numi+1;
    end
    pred = mod(numi, 2);
    if(pred == 1)
        pred = -1;
    end
    sum = sum + (pred-ip(1, in+j-1))^2;          % The only change when
end                                               % calculating metric
if(fir == 0)
    ori = metric(floor(nu/2) + 1, i+1);
    metric(floor(nu/2) + 1, i+1) = min(metric(nu+1, i)+sum, metric(floor(nu/2)+1, i+1));
    if(ori ~= metric(floor(nu/2) + 1, i+1))
        prev(floor(nu/2)+1, i+1) = nu;
    end
else
    temi = powo+nu;
    ori = metric(floor(temi/2) + 1, i+1);
    metric(floor(temi/2) + 1, i+1) =min(metric(nu+1, i)+sum, metric(floor(temi/2)+1, i+1));
    if(ori ~= metric(floor(temi/2) + 1, i+1))
```

```
                        prev(floor(temi/2)+1, i+1) = nu;
                    end
                end
            end
        end
    end
    mintu = maxi+10000;
    minti = -1;
    for i = 1:2^(glen-1)
        if(mintu > metric(i, len/nglen +1))
            mintu = metric(i,len/nglen +1);
            minti = i-1;
        end
    end
    display(minti);
    dec = zeros(1, len/nglen);
    j = len/nglen;
    for i = 1:len/nglen
        if(minti >= 2^(glen-2))
            dec(1,j) =1;
        end
        minti = prev(minti+1, j+1);
        j = j-1;
    end
end
```

Monte-Carlo Simulations

```
EbN0_dB    = 0:0.5:10;
EbN0_lin   = 10.^(EbN0_dB/10);
num_trials = 1000;
info_len   = 50;
gen2 = [1,1,0,1;1,0,1,1;1,1,1,1];
gen3 = [1,1,1,0,0,1; 1,1,0,1,0,1;1,0,1,1,1,1];
codes = {
  struct('name','r=1/2,K=3 soft','gens',{{[1 1 1],[1 0 1]}},'rate',1/2), ...
  struct('name','r=1/3,K=4 soft','gens',{{[1 0 1 1],[1 1 0 1],[1 1 1 1]}},'rate',1/3), ...
  struct('name','r=1/3,K=6 soft','gens',{{[1 0 0 1 1 1],[1 0 1 0 1 1],[1 1 1 1 0 1]}},'rate',1/3)...
  struct('name','r=1/2,K=3 hard','gens',{{[1 1 1],[1 0 1]}},'rate',1/2), ...
  struct('name','r=1/3,K=4 hard','gens',{{[1 0 1 1],[1 1 0 1],[1 1 1 1]}},'rate',1/3), ...
  struct('name','r=1/3,K=6 hard','gens',{{[1 0 0 1 1 1],[1 0 1 0 1 1],[1 1 1 1 0 1]}},'rate',1/3)
};

ber = zeros(numel(codes)*2, numel(EbN0_dB));
```

```
for c = 1:3  % for all generator matrices

  if(c == 1)
     G = gen;
  end
  if(c == 2)
     G = gen2;
  end
  if(c == 3)
     G = gen3;
  end

 R    = codes{c}.rate;

 for i = 1:numel(EbN0_dB)        %For all energy levels
  gamma  = EbN0_lin(i)*R;
  sigma  = 1/sqrt(gamma);
  errors = 0;

  for t = 1:num_trials                  % monte-carlo simulations
   info = randi([0 1],1,info_len);

   % encode & modulate
   enc  = encoder(info, G);
   bpsk = 1-2*enc;
```

```
      % AWGN + hard decision
      y     = pass_msg(bpsk, sigma);
      hard  = y<0;

      % decode
      dec = soft_decoder(y, G);

      % add errors to get ber
      errors = errors + sum(dec~=info);
    end

    %get ber
    ber(c,i) = errors/(num_trials*info_len);
    fprintf('%s, Eb/N0=%.1f dB → BER=%.3e\n', codes{c}.name, EbN0_dB(i), ber(c,i));
  end
end
```

Thank You