MATLAB CODE FOR SIMULATION OF TURBO CODES

The following MATLAB FUNCTION simulates Turbo Coding system, which are also known as parallel concatenated convolutional codes. These MATLAB function implement the SOVA algorithms based on the system model of iterative turbo code decoder:

- Bin_state.m, utility to convert an integer to a vector of binary bits.
- Demultiplex.m, serial to parallel demultiplex at the receiver.
- Encode bit.m, two (Recursive Systematic Convolutional) encoder to encode
- Encoder.m, turbo encoding process.
- Int state.m, utility to convert from vector of bits to integer.
- Rsc_encode.m, encodes a block of data using a RSC encoder.
- Trellis.m, set up trellis for a given code generator.
- Turbo sys demo.m, turbo encoding and turbo decoding system simulation.
  Topmost function. Start here! (An error corrected for SOVA application!)

The information bits are randomly generated which assume a uniform distribution. For each frame of bits, a random interleaver is generated for it. Thus the code performance stands for the average performance among all interleavers. I did set the following parameters by myself:

- Decoding algorithm
- Signal to noise ratio (Eb/N0)
- Code generator
- Frame size
- Puncture/unpuncture
- Maximum number of iterations
- Number of frame errors to terminate
Turbo encoding and turbo decoding system simulation.

% Interleavers are randomly generated for each frame.
% clear all
% Write display messages to a text file
% Choose decoding algorithm
dec_alg = input(' Please enter the decoding algorithm. (1:SOVA, default: 1)  ');
if isempty(dec_alg)
    dec_alg = 1;
end
% Frame size
L_total = input(' Please enter the frame size (= info + tail, default: 192)   ');
if isempty(L_total)
    L_total = 192;  % information bits plus tail bits
end
% Code generator
g = input(' Please enter code generator: ( default: g = [1 0 1; 1 1 1 ] )      ');
if isempty(g)
    g = [ 1 0 1; 1 1 1 ];
end
%g = [1 1 0 1; 1 1 1 1];
%g = [1 1 1 1; 1 0 0 0 1];
%g = [1 0 1 0 1 1; 1 1 1 1 0 1];
[n,K] = size(g);
m = K - 1;
nstates = 2^m;
%puncture = 0, puncturing into rate 1/2;
%puncture = 1, no puncturing
puncture = input(' Please choose punctured / unpunctured (0/1): default 0     ');
if isempty(puncture)
puncture=0
% code rate
rate=1/(2+puncture);

% Fading amplitude; a=1 in AGWN channel
a=1

% Number of iterations
niter=(Please enter number of iterations for each frame: default 5); If isempty(niter)
niter=5
dd

% Number of frame errors to count as a stop criterion
ferrlim=input(Please enter number of frame errors to terminate: default 15); if isempty(ferrlim)
ferrlim=15;
end

EbNodb=input(Please enter Eb/No in dB: default [2.0]); If isempty(EbNodb)
EBnOdb=[2.0]: End

fprintf(\nFrame size=%ld\n\n); for i=1:n
for j=1:k
fprintf(\%6d\n\n);
if puncture==0
fprintf(\texttt{Punctured, code rate=1/2})\n;
else
fprintf(\texttt{Unpunctured, code rate=1/3})\n;
end
fprintf(\texttt{Iteration number=\%6d})\n;\n\texttt{ ecstatic frame errors=\%6d})\n;ferrlim)\n;
fprintf(\texttt{Eb / No (dB)=})\n;\n\texttt{ EbNo})\n;\nend
fprintf(\texttt{======================================})\n;\n\texttt{+ + + Please be patient. Wait a while to get the result.+ + +})\n;\nfor Nen=1:length(EbNodb)
en=10^(EbNo(nEN)/10); \% convert Eb/No from unit db to normal numbers
L_c=4*a*en*rate; \% reliability value of the channel
Sigma=1/sqrt(2*rate*en); \% standard deviation of AWGN noise
\% Clear bit error counter and frame error counter
Ers(Nen,1:niter)=zeros(1,niter);
ferr(nEN,1:niter)=zeros(1,niter);

nframe=0; \% clear counter of transmitted frames
while nframe(niter)<ferrlim
nframe=nframe+1;
x=round(rand(1,L_total-m)); \% info. bits
[temp, alpha] = sort(rand(1, L_total));  % random interleaver mapping
en_output = encoderm(x, g, alpha, puncture);  % encoder output(+1/-1)

r = en_output + sigma * randn(1, L_total*(2+puncture)); % received bits
yk = demultiplexer(r, alpha, puncture) % demultiplex to get input for decoder 1 and 2

% scale the received bits
rec_s = 0.5*L_c*yk;

% Initialize extrinsic information
L_e(1:L_total) = zeros(1, L_total);

For iter = 1:niter
% Decoder one
L_a(alpha) = L_e  % a priori info.
L_all = sova0(rec_s(2,:), g, L_a, 2);  % complete info.
L_e = L_e - 2*rec_s(2, 1:2:2*L_total) - L_a  % extrinsic info.

% Estimate the info. bits
xhat(alpha) = (sign(L_all)+1)/2;

% Number of bits errors in current iteration
err(itr) = length(find9xhat(1:L_total-m)~=x));

% count frame errors for the current iteration
if err(itr) > 0
nferr(Nen, iter) = nferr(nEN, iter)+1;
end
end %iter

%Total number of bit errors for all iterations
errs(Nen,1:niter)=errs(Nen,1:niter)+ err(1:niter);

if re(nframe,3)= =0
    nferr( nEN, niter)==ferrlim
% Bit error rate
ber( nEN,1:niter)=errs( nEN, 1:niter)/nframe/(L_total-m);
%frame error rate
fer(nEN, 1 :niter(nEN,1:niter(nEN,1:niter)/nframe;

%Display intermediate results in process
fprintf(********** Eb/N0= %5.2f db***********
        nEb/N0(nEN));
fprintf('Frame size=%d, rate 1/%, rate 1%d.
        nframe, nferr(nEN, niter));
fprintf(' %d frames transmitted, %d frames in error.
        nframe, nferr(nEN, niter));
fprintf(' Bit Error Rate (from iteration 1 to iteration %d):
        niter);
for i=1:niter
    fprintf(' %8.4e  ber(nEN,i));
end
fprintf(';
fprintf('Frame Error Rate (from iteration 1 to iteration %d):
        niter);
for i=1:niter
    fprintf(' %8.4e  fer(nEN, i));
end
fprintf(';
fprintf('********************************************

% save intermediate results
save turbo_sys_demo EbNodb ber fer
end
Utility to convert an integer to a vector of binary bits.

Function bin_state=bin_state(int_state, m)

% converts a vector of integer into a matrix; the i-th row is the binary form
% of m bits for the i-th integer

for j=1: length( int_state )
for I = m:-1:1
state(j,m-i+1) = fix( int_state(j)/ (2^(i-1)) ) ;
int_state(j) = int_state(j) _ state(j,m-1)*2^(i-1);
end
end

Serial to parallel demultiplexer at the receiver

Function subr = demulplex9r, alpha, puncture);
% At receiver end, serial to parallel demultiplex to get the code word of each
% encoder
% alpha: interleaver mapping
% puncture =0: use puncturing to increase rare to ½
% puncture = 1; unpunctured, rate 1/3
% Frame size, which includes info. bits and tail bits
L_total =length9r)/(2+puncture0;
% Extract the parity bits for both decoders
If puncture ==1  % unpuncturd
For I = 1:L_total
  x_sys(i) = r(3*(i-1)+1);
  for j =1:2
    subr(j,2*i) = r(3*i-1+j);
  end
end
else  % unpunctured
  for i=1:L_total
    x_sys(i) = r(i-1)=1);
    for j =1:2
      subr9j,2*i)=0;
    end
    if rem(I,2*i) = r(2*i);
      else
        subr(2,2*i) = r(2*i);
      end
  end
end
end

% Extract the systematicbits for both decoders
for j =1: L_total
  % for decoder one
  subr(1,2*(j-1)+) = x_sys(j);
  % For decoder two: interleave the systematic bits
  subr(2,2*(j-1)+) = x_sys(alpha(j));
end
Two RSC (Recurcive Systematic Convolutional) encoder to encode frames of information bits.

Function [output, state] = encode_bit(g, input,state)

% This function takes as an input a single bit to be encoded
% as well as the coefficients of the generator polynomials and
% the current state vector
% It returns as output n encoded data bits, where 1/n is the
% code rate.
% the rate is 1/n
% k is the constraint length
% m is the amount of memory
[n,k] = size(g);
M = k-1;

% determine the next output bit
For i=1:n
 Output(i) = g(i,1)*input
 For j = 2:k
  Output(i) = xor(output(i),g(i,j)*(j-1));
 end;
end

state = [input, state(1 :m-1)];
Turbo encoding process

Function en_output = encoderm( x, g, alpha, puncture )
% uses interleaver map alpha
% if puncture = 1, unpunctured produces a rate 1/3 output of a fixed length
% if puncture = 0, punctured, produces a rate ½ output
% multiplexer chooses odd check bits from RSC1
% and even check bits from RSC2
% determine the constraint length (k), memory (m)
% and number of information bits plus tail bits.

[n,k] = size(g);
m = k-1;
L_info = length(x)
L_total = L_info + m;

% generate the codeword corresponding
% end = 1, perfectly terminated;
input = x;
output1 = rsc_encode(g,input,1);

% make a matrix with first few corresponding to info sequence
% second row corresponding to RSC #1's check bits.
% third row corresponding to RSC #2's check bits.

y(1,:) = output(1:2:2*L_total);
y(2,:) = output(2:2:2*L_total);

% interleaver input to the second encoder
for i=1:L_total
input1(1:i) = y(1, alpha(i));
output2 = rsc_encode(g, input1(1,1:L_total,-1 );
y(3,:) = output2(2:2:2*L_total);

% parallel to serial multiplex to get output vector
% puncture = 0: rate increase from 1/3 to 1/2;
% puncture = 1: unpunctured, rate = 1/3;

if puncture > 0 % unpunctured
  for I = 1:L_total
    for j = 1:3
      en_output(1,3*(i_1)+j) = y(j,i);
    end
  end
else     % punctured into rate 1/2
  for I = 1:L_total
    en_output(1,n*(i-1)+1) = y(1,i);
    if rem(i,2)
      % odd check bits from RSC1
      en_output(1,n*i) = y(3,i);
    end
  end
end

% antipodal modulation: +1/-1
En_output = 2 * en_output - ones(size(en_output));
Utility to convert from vector of bits to integer

```matlab
function int_state = int_state( state )
% converts a row vector of m bits into an integer (base10

[dummy, m] = size( state );

for I = 1:m
    vect(i) = 2^(m-1);
end

int_state = state*vect';
```

Encodes a block of data using a RSC encoder

```matlab
Function y = rsc_encode(g, x, terminate)
% encodes a block of data x (0/1) with a recursive systematic
% convolutional code with generator vectors in g, and
% returns the output in y (0/10).
% if terminate >0, the trellis is perfectly terminated
% if terminate <0, it is left unterminated;

% determine the constraint length (k), memory (m), and rate (1/n0
% and number of information bits.
[n,k] = size(g);
m = k-1;
if terminate >0
    L_info = length(x);
    L_total = L_info + m;
Else
```
\[
L_{\text{total}} = \text{length}(x);
\]
\[
L_{\text{total}} = L_{\text{total}} - m;
\]
\]

% initialize the state vector
State = zeros(1,m);

% generate the codeword
for I = 1:L_total
if terminated < 0 \&\& (terminated > 0 \&\& i \leq L_{\text{info}})
d_k = \text{rem}(g(1:2:k)\cdot \text{state}, 2);
end

a_k = \text{rem}(g(1:2:k)\cdot d_k \cdot \text{state}, 2);
[output_bits, state] = encode_bit(g, a_k, state);
% since systematic, first output is input bit
Output_bits(1,1) = d_k;
y(i-1) = 1:n*i = output_bits;
end
SOVA-Soft Output Viterbi Algorithm

Function L_all = sova(rec_s,g,L_a, ind_dec)
% this function implements Soft Output Viterbi Algorithm in trace back mode
% input
% rec_s: scaled received bits.rec_s(k) =0.5 *L_c(k) * k(k)
% L_c =4* a*Es/No, reliability value of the channel
% y: received bits
% g: encoder generator matrix in binary form, g(1,: for f feedback, g(2,:for feedforward
% L_a: a priori information about the info. bits. From the previous
% component decoder
% ind_dec: index of the component decoder.
% =1:component decoder 1; The trellis is terminated to all zero state
% =2: component decoder 2; The trellis is not perfectly terminated.
% output:
% L_all:log (P(x =1 │y)) / (P(x =-1 │y))
% frame size, info. + tail bits
L_total = length(L_a);
[n,k] = size(g);
m =k-1
nstates = 2^m;
infty =1e10;
% SOVA window size. Make decision after δDelay. Decide bit k when received bits
% for bit (k +delta) are processed. Trace back from (k+delta) to k.
Delta =30
% set up the trellis defined by g.
[next_out, next_state, last_out,last_state] = trellis(g);

% Initials path metrics to infty
for t =1:L_total+1
for state=1:nstates
    path_metric(state,t)=-Inf;
end
end

% trace forward to complete all the path metrics
Path_metric(1,1) =0:
For t=1:L_total
    Y=rec_s(2*t-1:2*t);
    For state=1:nstates
        Sym0 =last_out(state,1:2);
        Sym1 =last_out(state,3:4);
        Sym0 =last_state(state,1);
        Sym1 =last_state(state,2);
        Mk0 =y*sym0\cdot L_a(t)/2 + path_metric(state0,t);
        Mk1 =y*sym1\cdot L_a(t)/2 + path metric(state1,t);
        if Mk0>Mk1
            path_metric(state,t+1)=Mk0;
            Mdiff(state,t+1) =Mk0 - Mk1;
            prev_bit(state,t+1)=0
        else
            path_metric(state(state,t+1)=Mk1;
            Mdiff(state,t+1)=Mk1-Mk0;
            prev_bit(state,t+1)=1;end
    end
end
end
end

% for decoder1 trace back from all zero state,
% for decoder two, trace back from the most likely state
If ind_dec ===1
M1state(L_total+1)=1;
else
M1state(L_total+1)=find path_metric(:,L_total+1)==max(path_metric(:,L_total+1))
end

% Trace back to get the estimated bits, and the most likely path
For t=L_total:-1:1
Est(t) =prev_bit(m1state(t+1), est(t)+1);
end

% Find the minimum delta that corresponds to a competition path with different info. bit estimation.
% Give the soft output
for t=1:L_total
llr =Infry;
for i=0:delta
if t+i<L_total+1
bit=1-est(t+1);
temp_state=last_state(m1state(t+i+10,bit+1)
for j=i:-1:0
bit=prev_bit(temp_state,t+j+1);
temp_state=last_state(temp_state, bit+1);
end
if bit~=est(t)
llr=min(llr,Mdiff(m1state(t+i+1),(t+i+1));
end
end
end
end
L_all(t)=(2*est(t)-1)*llr;
End
Set up the trellis for a given code generator

Function [next_out, next_state, last_out, last_state] = trellis(g)
% set up the trellis given code generator g
% g given in binary matrix form. E.g. g=[1 0 1 ;1 1 1];
% next_out(I,1:2): trellis next_out (systematic bit; parity bit) when input=0, state=I;
% next_out(i,j)= -1 or 1
% next_out(I,3:4): trellis next_out (systematic bit; parity bit) when input=1, state=I;
% next_state(i,1): next state when input=0, state=I, next_state(i,i)=1,
% next_state(i,2): next state when input=1, state=I;
% last_out(I,1:2): trellis last_out (systematic bit; parity bit) when input=0, state=I;
% last_out(i,j)= -1 or 1
% last_out(I,3:4): trellis last_out (systematic bit; parity bit) when input=1, state=I;
% last_state(i,1): previous state that comes to state I when info. bit=0;
% last_state(i,2): previous state that comes to state I when info. bit=1;
[n,k]=size(g);
m=k-1
max_state=2^m;
% setup next_out and next_state matrices for systematic code
For state=1:max_state
state_vector=bin_state(state-1,m);
% when receive a 0
d_k=0
a_k=rem(g(1,:)*[0 state_vector]’ ,2);
[out_0, state_0]=encode_bit(g, a_k.state_vector)’ ,2);
[out_1(1)=1;
next_out(state,:) = 2*[out_0 out_1]_1;
next_state(state,:)=[(int_state_0)+1](int_state(state_1)+1)];
end

% find out which two previous state can come to present state
last_state=zeros(max_state,2);
for bit=0:1
for state=1:max_state
last_state(next_state(state,bit+1),bit+1)=state;
last_out(next_state(state,bit+1), bit*2+1:bit*2+2)=next_out(state,bit*2+1:bit*2+2);
end
end