项目1-4不要求字数，直接在系统提交。

期末课程论文上交打印稿一份，同时要在系统提交电子稿。

期末课程论文按照模板格式写作，某些章节如果涉及不到可以省略。

每一份电子文档命名格式为：学号-姓名-项目名

期末论文提交截止日期：第十六周周五下午五点。请学委收齐打印版交到办公室。

项目一：信源熵的计算

1、基本要求：编写计算离散随机变量的熵、联合熵、条件熵、互信息的Matlab程序。

2、输入：信源分布，信道转移概率矩阵。其中；

3、输出：信源熵，条件熵，联合熵，互信息。

4、验证程序的正确性。利用下表中的数据进行测试.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| aj | ai | | |  |
| 0 | 1 | 2 |  |
| 0 | 1/4 | 1/18 | 0 | 11/36 |
| 1 | 1/18 | 1/3 | 1/18 | 4/9 |
| 2 | 0 | 1/18 | 7/36 | 1/4 |
|  | 11/36 | 4/9 | 1/4 |  |

5、自学图像熵的相关概念，并应用所学知识，使用matlab求解图像熵。

图像熵计算过程：

1) 输入一幅图像，并将其转换成灰度图像。

2) 统计出图像中每个灰度阶象素概率。

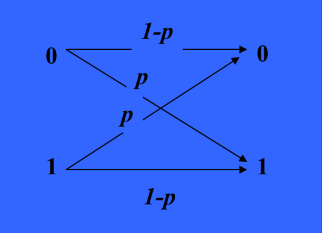
3) 计算出一幅图像的一维熵。

项目二：信道及其容量

本设计要求掌握互信息与二元对称信道BSC的信道容量计算方法。对于二元对称信道的输入概率空间为



信道为



1. 要求程序能够判断信道矩阵是否为对称。
2. 如果是对称矩阵则计算信道容量。绘制当从0到1之间变化时的信道容量曲线。
3. 计算其平均互信息，绘制当从0到1之间变化时的平均互信息熵曲线.

项目三：信源编码

1、随机产生一组不少于1000码元的二进制序列并进行Huffman编码与解码；利用Matlab, C或者其他编程语言计算信源Huffman编码的平均码长和编码效率。

2、选择一篇较长的自然科学文章（英文、不少于10页），以扩展的ASKII码作为初始化字典的词条，即字典的0-255项预置为ASKII的全部8位字符。试用LZW算法将文档压缩，再解压缩。

项目四：信道编码

本设计要求利用MATLAB或其他编程语言生成信号源序列，信道噪声，实现循环码编解码过程。

1. 已知（10，6）系统循环码的生成多项式为：*g*(*x*)=*x*4+*x*+1，请设计该循环码的编码器。
2. 输入随机码元序列长度至少1000位。
3. 按照错误概率Pe随机产生差错图样，得到实际接收码字。
4. 根据接收到的码字进行译码，计算误码率。
5. 调整Pe的大小，画出误码率与Pe之间的关系曲线。

期末项目：Turbo码的仿真实现

给定Turbo码参数：

码率1/3

传输函数*G*(*D*) = [1, g1(D)/g0(D)]

*g*0(*D*) = 1 + *D*2 + *D*3,

*g*1(*D*) = 1 + *D* + *D*3.

移位寄存器的初始值为0



基本设计思路及要求：

1. 读入不少于100字符的文本文件作为数据源，可以将ASCII码转化为二进制再使用。也可以随机产生一组不少于1000码元的二进制序列（提示：randint）。
2. 采用1/3码率的Turbo码，通过高斯白噪声(AWGN)信道，分别采用LOG-MAP和SOVA译码，迭代1-5次。

加入噪声的方法供参考。

方法1：直接仿真二进制序列噪声，原理与循环码相同，即给定Pe，产生一组符合Pe的二进制序列。噪声序列与卷积码序列相加之后送入Viterbi译码器进行硬判决译码。

方法2，用BPSK调制方式调制信号，加入信道噪声（高斯白噪声，给出信噪比SNR）后再经过BPSK解调制后送入Viterbi译码器进行硬判决译码。

（3） 误码统计后输出。

（4）分别画出两种译码算法的误码率曲线。

（5）编码算法需要自己实现，译码算法可以直接调用现有的代码。

附录：例程

% This script simulates the classical turbo encoding-decoding system.

% It simulates parallel concatenated convolutional codes.

% Two component rate 1/2 RSC (Recursive Systematic Convolutional) component encoders are assumed.

% First encoder is terminated with tails bits. (Info + tail) bits are scrambled and passed to

% the second encoder, while second encoder is left open without tail bits of itself.

%

% Random information bits are modulated into +1/-1, and transmitted through a AWGN channel.

% Interleavers are randomly generated for each frame.

%

% Log-MAP algorithm without quantization or approximation is used.

% By making use of ln(e^x+e^y) = max(x,y) + ln(1+e^(-abs(x-y))),

% the Log-MAP can be simplified with a look-up table for the correction function.

% If use approximation ln(e^x+e^y) = max(x,y), it becomes MAX-Log-MAP.

%

% Copyright Nov 1998, Yufei Wu

% MPRG lab, Virginia Tech.

% for academic use only

clear all

% Write display messages to a text file

diary turbo\_logmap.txt

% Choose decoding algorithm

dec\_alg = input(' Please enter the decoding algorithm. (0:Log-MAP, 1:SOVA) default 0 ');

if isempty(dec\_alg)

dec\_alg = 0;

end

% Frame size

L\_total = input(' Please enter the frame size (= info + tail, default: 400) ');

if isempty(L\_total)

L\_total = 300; % infomation bits plus tail bits

end

% Code generator

g = input(' Please enter code generator: ( default: g = [1 1 1; 1 0 1 ] ) ');

if isempty(g)

g = [ 1 1 1;

1 0 1 ];

end

%g = [1 1 0 1; 1 1 1 1];

%g = [1 1 1 1 1; 1 0 0 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;

end

% Code rate

rate = 1/(2+puncture);

% Fading amplitude; a=1 in AWGN channel

a = 1;

% Number of iterations

niter = input(' Please enter number of iterations for each frame: default 5 ');

if isempty(niter)

niter = 5;

end

% Number of frame errors to count as a stop criterior

ferrlim = input(' Please enter number of frame errors to terminate: default 15 ');

if isempty(ferrlim)

ferrlim = 15;

end

EbN0db = input(' Please enter Eb/N0 in dB : default [2.0] ');

if isempty(EbN0db)

EbN0db = [2.0];

end

fprintf('\n\n----------------------------------------------------\n');

if dec\_alg == 0

fprintf(' === Log-MAP decoder === \n');

else

fprintf(' === SOVA decoder === \n');

end

fprintf(' Frame size = %6d\n',L\_total);

fprintf(' code generator: \n');

for i = 1:n

for j = 1:K

fprintf( '%6d', g(i,j));

end

fprintf('\n');

end

if puncture==0

fprintf(' Punctured, code rate = 1/2 \n');

else

fprintf(' Unpunctured, code rate = 1/3 \n');

end

fprintf(' iteration number = %6d\n', niter);

fprintf(' terminate frame errors = %6d\n', ferrlim);

fprintf(' Eb / N0 (dB) = ');

for i = 1:length(EbN0db)

fprintf('%10.2f',EbN0db(i));

end

fprintf('\n----------------------------------------------------\n\n');

fprintf('+ + + + Please be patient. Wait a while to get the result. + + + +\n');

for nEN = 1:length(EbN0db)

en = 10^(EbN0db(nEN)/10); % convert Eb/N0 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

errs(nEN,1:niter) = zeros(1,niter);

nferr(nEN,1:niter) = zeros(1,niter);

nframe = 0; % clear counter of transmitted frames

while nferr(nEN, 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 = demultiplex(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.

if dec\_alg == 0

L\_all = logmapo(rec\_s(1,:), g, L\_a, 1); % complete info.

else

L\_all = sova0(rec\_s(1,:), g, L\_a, 1); % complete info.

end

L\_e = L\_all - 2\*rec\_s(1,1:2:2\*L\_total) - L\_a; % extrinsic info.

% Decoder two

L\_a = L\_e(alpha); % a priori info.

if dec\_alg == 0

L\_all = logmapo(rec\_s(2,:), g, L\_a, 2); % complete info.

else

L\_all = sova0(rec\_s(2,:), g, L\_a, 2); % complete info.

end

L\_e = L\_all - 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 bit errors in current iteration

err(iter) = length(find(xhat(1:L\_total-m)~=x));

% Count frame errors for the current iteration

if err(iter)>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 rem(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) = nferr(nEN,1:niter)/nframe;

% Display intermediate results in process

fprintf('\*\*\*\*\*\*\*\*\*\*\*\*\*\* Eb/N0 = %5.2f db \*\*\*\*\*\*\*\*\*\*\*\*\*\*\n', EbN0db(nEN));

fprintf('Frame size = %d, rate 1/%d. \n', L\_total, 2+puncture);

fprintf('%d frames transmitted, %d frames in error.\n', nframe, nferr(nEN, niter));

fprintf('Bit Error Rate (from iteration 1 to iteration %d):\n', niter);

for i=1:niter

fprintf('%8.4e ', ber(nEN,i));

end

fprintf('\n');

fprintf('Frame Error Rate (from iteration 1 to iteration %d):\n', niter);

for i=1:niter

fprintf('%8.4e ', fer(nEN,i));

end

fprintf('\n');

fprintf('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n');

% Save intermediate results

save turbo\_sys\_demo EbN0db ber fer

end %iter

end %while

end %nEN

diary off

function [next\_out, next\_state, last\_out, last\_state] = trellis(g)

% copyright Nov. 1998 Yufei Wu

% MPRG lab, Virginia Tech

% for academic use only

% set up the trellis given code generator g

% g given in binary matrix form. e.g. g = [ 1 1 1; 1 0 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,...2^m

% 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;

% set up 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);

out\_0(1) = 0;

% when receive a 1

d\_k = 1;

a\_k = rem( g(1,:)\*[1 state\_vector]', 2 );

[out\_1, state\_1] = encode\_bit(g, a\_k, state\_vector);

out\_1(1) = 1;

next\_out(state,:) = 2\*[out\_0 out\_1]-1;

next\_state(state,:) = [(int\_state(state\_0)+1) (int\_state(state\_1)+1)];

end

% find out which two previous states 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

function L\_all = sova(rec\_s, g, L\_a, ind\_dec)

% This function implememts Soft Output Viterbi Algorithm in trace back mode

% Input:

% rec\_s: scaled received bits. rec\_s(k) = 0.5 \* L\_c(k) \* y(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 feedback, g(2,:) for feedforward

% L\_a: a priori information about the info. bits. Extrinsic info. 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) )

%

% Copyright: Yufei Wu, Nov. 1998

% MPRG lab, Virginia Tech

% for academic use only

% 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 'delta' delay. 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);

% Initialize path metrics to -Infty

for t=1:L\_total+1

for state=1:nstates

path\_metric(state,t) = -Infty;

end

end

% Trace forward to compute 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);

state0 = last\_state(state,1);

state1 = last\_state(state,2);

Mk0 = y\*sym0' - L\_a(t)/2 + path\_metric(state0,t);

Mk1 = y\*sym1' + 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,t+1)=Mk1;

Mdiff(state,t+1) = Mk1 - Mk0;

prev\_bit(state,t+1) = 1;

end

end

end

% For decoder 1, trace back from all zero state,

% for decoder two, trace back from the most likely state

if ind\_dec == 1

mlstate(L\_total+1) = 1;

else

mlstate(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(mlstate(t+1),t+1);

mlstate(t) = last\_state(mlstate(t+1), est(t)+1);

end

% Find the minimum delta that corresponds to a compitition path with different info. bit estimation.

% Give the soft output

for t=1:L\_total

llr = Infty;

for i=0:delta

if t+i<L\_total+1

bit = 1-est(t+i);

temp\_state = last\_state(mlstate(t+i+1), bit+1);

for j=i-1:-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(mlstate(t+i+1), t+i+1) );

end

end

end

L\_all(t) = (2\*est(t) - 1) \* llr;

end

function y = rsc\_encode(g, x, terminated)

% Copyright Nov. 1998 Yufei Wu

% MPRG lab, Virginia Tech.

% for academic use only

% 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 (0/1).

% if terminated>0, the trellis is perfectly terminated

% if terminated<0, it is left unterminated;

% determine the constraint length (K), memory (m), and rate (1/n)

% and number of information bits.

[n,K] = size(g);

m = K - 1;

if terminated>0

L\_info = length(x);

L\_total = L\_info + m;

else

L\_total = length(x);

L\_info = L\_total - m;

end

% initialize the state vector

state = zeros(1,m);

% generate the codeword

for i = 1:L\_total

if terminated<0 | (terminated>0 & i<=L\_info)

d\_k = x(1,i);

elseif terminated>0 & i>L\_info

% terminate the trellis

d\_k = rem( g(1,2:K)\*state', 2 );

end

a\_k = rem( g(1,:)\*[d\_k 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(n\*(i-1)+1:n\*i) = output\_bits;

end

function L\_all = logmapo(rec\_s,g,L\_a,ind\_dec)

% Copyright Nov 1998, Yufei Wu

% MPRG lab, Virginia Tech.

% for academic use only

% Log\_MAP algorithm using straightforward method to compute branch metrics

% no approximation is used.

% Can be simplified to Max-Log-MAP by using approximation ln(e^x+e^y) = max(x,y).

% Input: rec\_s: scaled received bits.

% rec\_s = 0.5 \* L\_c \* yk = ( 2 \* a \* rate \* Eb/N0 ) \* yk

% g: code generator for the component RSC code, in binary matrix form.

% L\_a: a priori info. for the current decoder,

% scrambled version of extrinsic Inftyo. of the previous decoder.

% ind\_dec: index of decoder. Either 1 or 2.

% Encoder 1 is assumed to be terminated, while encoder 2 is open.

%

% Output: L\_all: log-likelihood ratio of the symbols. Complete information.

% Total number of bits: Inftyo. + tail

L\_total = length(rec\_s)/2;

[n,K] = size(g);

m = K - 1;

nstates = 2^m; % number of states in the trellis

% Set up the trellis

[next\_out, next\_state, last\_out, last\_state] = trellis(g);

Infty = 1e10;

% Initialization of Alpha

Alpha(1,1) = 0;

Alpha(1,2:nstates) = -Infty\*ones(1,nstates-1);

% Initialization of Beta

if ind\_dec==1

Beta(L\_total,1) = 0;

Beta(L\_total,2:nstates) = -Infty\*ones(1,nstates-1);

elseif ind\_dec==2

Beta(L\_total,1:nstates) = zeros(1,nstates);

else

fprintf('ind\_dec is limited to 1 and 2!\n');

end

% Trace forward, compute Alpha

for k = 2:L\_total+1

for state2 = 1:nstates

gamma = -Infty\*ones(1,nstates);

gamma(last\_state(state2,1)) = (-rec\_s(2\*k-3)+rec\_s(2\*k-2)\*last\_out(state2,2))....

-log(1+exp(L\_a(k-1)));

gamma(last\_state(state2,2)) = (rec\_s(2\*k-3)+rec\_s(2\*k-2)\*last\_out(state2,4))....

+L\_a(k-1)-log(1+exp(L\_a(k-1)));

if(sum(exp(gamma+Alpha(k-1,:)))<1e-300)

Alpha(k,state2)=-Infty;

else

Alpha(k,state2) = log( sum( exp( gamma+Alpha(k-1,:) ) ) );

end

end

tempmax(k) = max(Alpha(k,:));

Alpha(k,:) = Alpha(k,:) - tempmax(k);

end

% Trace backward, compute Beta

for k = L\_total-1:-1:1

for state1 = 1:nstates

gamma = -Infty\*ones(1,nstates);

gamma(next\_state(state1,1)) = (-rec\_s(2\*k+1)+rec\_s(2\*k+2)\*next\_out(state1,2))....

-log(1+exp(L\_a(k+1)));

gamma(next\_state(state1,2)) = (rec\_s(2\*k+1)+rec\_s(2\*k+2)\*next\_out(state1,4))....

+L\_a(k+1)-log(1+exp(L\_a(k+1)));

if(sum(exp(gamma+Beta(k+1,:)))<1e-300)

Beta(k,state1)=-Infty;

else

Beta(k,state1) = log(sum(exp(gamma+Beta(k+1,:))));

end

end

Beta(k,:) = Beta(k,:) - tempmax(k+1);

end

% Compute the soft output, log-likelihood ratio of symbols in the frame

for k = 1:L\_total

for state2 = 1:nstates

gamma0 = (-rec\_s(2\*k-1)+rec\_s(2\*k)\*last\_out(state2,2))....

-log(1+exp(L\_a(k)));

gamma1 = (rec\_s(2\*k-1)+rec\_s(2\*k)\*last\_out(state2,4))...

+L\_a(k)-log(1+exp(L\_a(k)));

temp0(state2) = exp(gamma0 + Alpha(k,last\_state(state2,1)) + Beta(k,state2));

temp1(state2) = exp(gamma1 + Alpha(k,last\_state(state2,2)) + Beta(k,state2));

end

L\_all(k) = log(sum(temp1)) - log(sum(temp0));

end

function int\_state = int\_state( state )

% Copyright 1996 Matthew C. Valenti

% MPRG lab, Virginia Tech.

% for academic use only

% converts a row vector of m bits into a integer (base 10)

[dummy, m] = size( state );

for i = 1:m

vect(i) = 2^(m-i);

end

int\_state = state\*vect';

function [output, state] = encode\_bit(g, input, state)

% Copyright 1996 Matthew C. Valenti

% MPRG lab, Virginia Tech

% for academic use only

% This function takes as an input a single bit to be encoded,

% as well as the coeficients 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)\*state(j-1));

end;

end

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