2021春现代通信原理第五次实验报告

——利用卷积码进行信道编码与解码

PB19071509 王瑞哲

>>>实验目的

- 掌握信道编码的思想和基本原理
- 掌握卷积码的编码和解码原理
- 掌握Matlab软件模拟进行信道编解码仿真的基本过程

>>>实验原理

一、信道编码

信道编码的本质是在信源的信息编码中加入一定数量的冗余符号（或称监督符号），使其满足一定的约束条件。码字一旦在传输过程中发生错误，信息符号与监督符号之间的约束关系就会被破坏。在接收端（接收端），根据既定的规则检查这种约束关系，从而达到发现和纠正错误的目的。

最基本的信道编码方案为线性分组码。线性分组码主要有两大缺点：一是在译码过程中必须等待整个码字全部接收到之后才能开始进行译码，二是需要精确的帧同步，从而导致时延较大、增益损失大。卷积码是线性分组码的特例，改善了线性分组码的缺点，使得无线通信性能得到了跳跃式的发展。后来还出现了Turbo码、极化码等，使信道编码效率逐渐接近香农极限。

二、卷积码编码方案

卷积码由 3 个整数 n，k，N 描述。 k/n 也表示编码效率（每编码比特所含的信息量）；但 n 与线性分组码中的含义不同，不再表示分组或码子长度； N 称为约束长度，表示在编码移位寄存器中 k 元组的级数。卷积码不同于分组码的一个重要特征就是编码器的记忆性，即卷积码编码过程中产生的 n 元组，不仅是当前输入 k 元组的函数，而且还是前面 N-1 个输入 k 元组的函数。实际情况下，n 和 k 经常取较小的值，而通过的 N 变化来控制编码的能力和复杂性。

卷积码的编码方法可以利用下面的方法来表示：

- **连接图表示**

  下图表示一个约束长度 K=3 的（2，1）卷积译码器，模 2 加法器的数目为 n=2 ，因此编码效率 k/n = 1/2。在每个输入比特时间上，1 位信息比特移入寄存器最左端的一级，同时将寄存器中原有比特均右移一级，接着便交替采样两个模 2 加法器，得到的码元就是与该输入比特相对应的分支字。对每一个输入信号比特都重复上述采样过程。

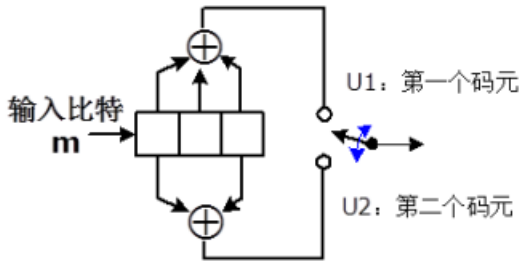


图 1 卷积码编码器（编码效率$1/2$，$K=3$）

- **状态图**

  卷积编码器属于有限状态机的器件。“有限”表明状态机制只有有限个不同的状态。有限状态机的状态可以用设备的当前输入和最少的信息量，来预测设备的输出。状态提供了有关过去序列过程及一组将来可能输出序列的限制，下一状态总是受到前一状态的限制。如下图所示，方框内的状态表示寄存器最右端 N-1 级的内容，状态间的路径表示由此状态转移时的输出分支字。对应于两种可能的输入 bit，从每个状态出发只有两种转移。

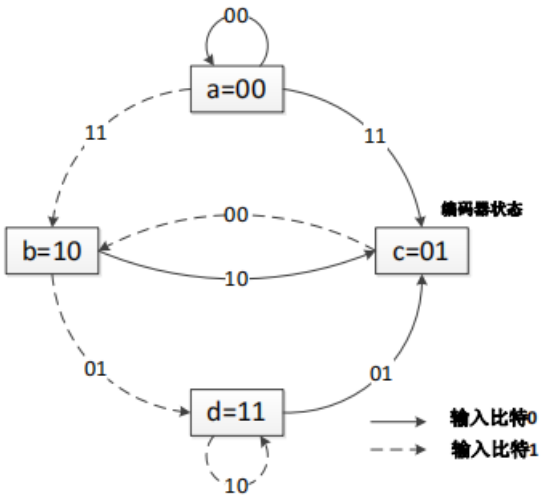


图 2 状态转移图

- **编码器网格图**

  在上述状态转移图 基础上，加入了时间尺度，以动态地描述输入序列的编码过程。网格图利用了结构上的重复性，从而能够更加方便地描述编码器。

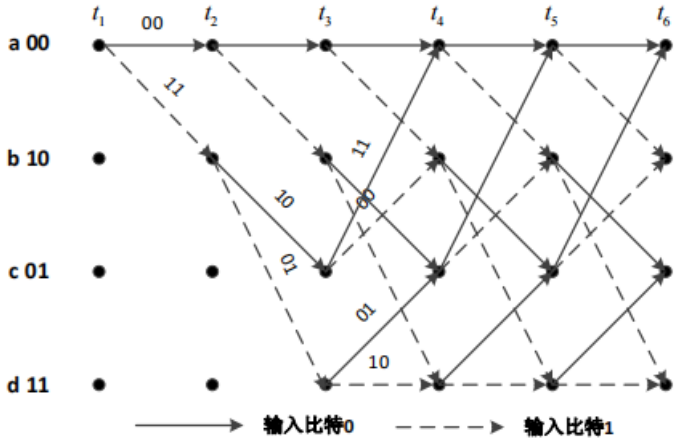


图 3 编码器网格图

三、卷积码解码方案——维特比Viterbi译码算法

维特比译码算法由维特比在 1967 年提出。维特比算法的实质是最大似然译码，但它利用了编码网格图的特殊结构，从而降低了计算的复杂性。该算法包括计算网格图上在时刻到达各个状态的路径和接受序列之间的相似度，或者说距离。维特比算法考虑的是，去除不可能成为最大似然选择对象的网格图上的路径，即如果有两条路径到达同一状态，则具有最佳量度的路径被选中，成为幸存路径。对所有状态都将进行这样的选路操作，译码器不断在网格图上深入，通过去除可能性最小的路径实现判决。

网格图中每个时刻 ti 上有 2^(K-1) 个状态，K为约束长度，每种状态都可经两条路径到达。维特比译码包括计算到达每个状态的两条路径的路径量度，并舍弃其中一条路径。在时刻 ti，算法对 2^(K-1) 个状态（节点）都进行上述计算，然后进入时刻 ti+1，并重复上述过程。在一个给定的时刻，各状态的幸存路径量度就是该状态在该时刻的状态量度。

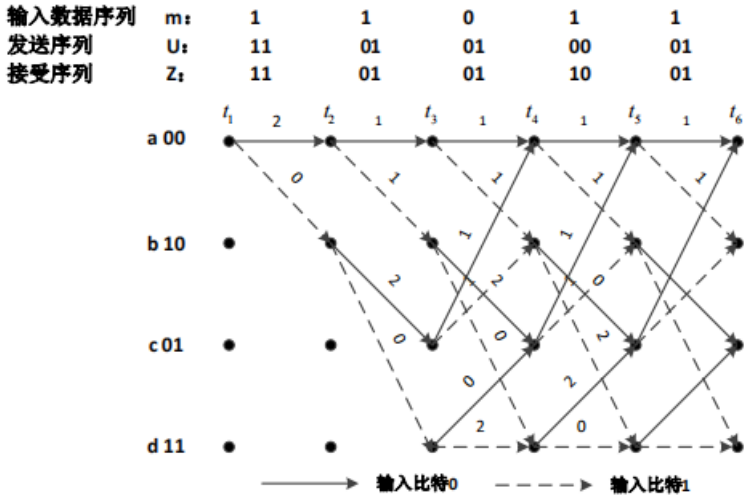


图 4 译码器网格图

>>>实验内容

一、在 MATLAB 上设计一个（2，1，3）卷积编码器和对应的采用维特比译码算法的译码器。编码器的生成多项式为：

$$g_1(X) = 1 + X + X^2$$
$$g_2(X) = 1 + X^2$$

将编码器的输出经过一个高斯白噪声信道的结果作为译码器的输入，观察比较译码器输出和编码器输入，了解卷积码的容错性，并计算译码结果的误比特率。

解：该项目主要由主函数、卷积编码函数和卷积解码函数组成。主函数部分如下：

```
% 测试主函数
clear
n = 2; k = 1; N = 3;              % (2,1,3)卷积码
G_x = [1, 1, 1; 1, 0, 1];         % 生成矩阵

Num = 50000;                                    % 信源长度
B = randi(2,1,Num)-1;                           % 生成随机信源序列(0,1)
Encode = conv_encode(B, G_x);                   % 卷积编码

snrlist = 0:0.5:10 ;
error_rate0 = zeros(1, length(snrlist));
error_rate1 = zeros(1, length(snrlist));
```

```
for i = 1:length(snrlist)
    snr = snrlist(i);
    % 不经过信道编码，直接加噪并简单门限判决
    B_d = Gnoisegen(B, snr);
    B_d(B_d > 0.5) = 1;
    B_d(B_d < 0.5) = 0;
    error_rate0(i) = mae(B-B_d);
    % 经过信道编解码
    Encode_Noise = Gnoisegen(Encode, snr);          % 信道加噪
    Encode_Noise(Encode_Noise > 0.5) = 1;
    Encode_Noise(Encode_Noise < 0.5) = 0;
    Decode = conv_decode(G_x, k, Encode_Noise);     % 卷积解码
    error_rate1(i) = mae(B-Decode);                    % 计算误码率
end
figure(1)
semilogy(snrlist, error_rate0, 'b', 'linewidth',2); hold on
semilogy(snrlist, error_rate1, 'r', 'linewidth',2);
grid on;
xlabel('信噪比 SNR / dB');ylabel('误码率 BER');
title('采用卷积码的信道编解码方案与不采用信道编码性能对比');
legend('无信道编码', '(2,1,3)卷积码');
```

主函数主要实现了模拟信源序列的编码、加噪和解码过程，并与不经过信源编码，直接加噪解码的性能（误码率）进行对比。

其中，信源编码函数为：

```
function [Output] = conv_encode(Input, G_x)
% Conv_encode 卷积码编码器函数
%   输入Input：待编码序列  G_x：生成矩阵  输出Output：编码后序列
    len = length(Input);
    k = 1;                                  % 表示每次对k个码元进行编码
    [n,N] = size(G_x);                      % G_x为矩阵，行数n表示一个输入码元拥有几个输出，列数N表示每次监督的输入码元数
    Output = zeros(1, n*(len+N-1));     % 输出序列长度为（输入序列长度+约束长度-1）*（一个输入码元对应输出长度）
    Input_add0 = [zeros(1,N-1), Input, zeros(1,N+1)];      % 在输入序列头尾补0，方便卷积输出和寄存器清洗

    % 循环每一位输入符号，获得输出矩阵
    register = fliplr(Input_add0(1,1:N));     % 水平翻转，倒序输入
    for i = 1:len+N-1
        % 生成每一位输入符号的n位输出
        Output(n*i-(n-1):n*i) = mod(register*G_x.',2);     % 模2加

        % 更新寄存器序列+待输出符号（共N个符号）
        register = [Input_add0(i+N), register];     % 添加新符号
        register(end) = [];                         % 挤掉旧符号
    end
end
```

上面这段代码模拟生成了一段移位寄存器，利用输入的生成矩阵的形式，模拟卷积码的输出过程。最终输出的序列长度为n*(len+N-1)，其中n为编码效率的倒数，len为输入序列长度，+N-1是因为在编码过程的最后要挤入N-1个0以将移位寄存器内所含的原序列信息全部计算输出。

信源解码函数为：

```
function [ decode_output ] = conv_decode( g,k,decode_input )
% (n,k,L)卷积Viterbi译码器
%   g               n个生成矢量排列形成的卷积码生成矩阵:g = [g1;g2;...;gn]
%   k               编码位数
%   decode_input    输入码流

    N = size(g,2);                                  % 约束长度N
    n = size(g,1);                                  % 编码输出位数n
    number_of_states = 2^(k*(N-1));                 % 网格图的状态数

    input = zeros(number_of_states);                % 输入矩阵
    nextstate = zeros(number_of_states,2^k);        % 状态转移矩阵
    output = zeros(number_of_states,2^k);           % 输出矩阵
    decode_input = [decode_input,zeros(1, n/k)];    % 补充最末尾的输出序列（长度为n/k
的全0列）
    %------ 对各个状态进行运算，得到输入矩阵、状态转移矩阵与输出矩阵 -----%
    for s = 0:number_of_states-1
        % 对前一时刻状态到下一时刻状态之间的各条支路进行运算
        for t = 0:2^k-1
            % next_state_function函数产生移存器跳转到的下一状态及当前时刻编码器内容
            [next_state,memory_contents] = next_state_function(s,t,N,k);
            % 从上至下表示当前状态s0,s1,s2……
            % 从左至右表示下一状态s0,s1,s2……
            % 内容为经由支路编号
            input(s+1,next_state+1) = t;                % 输入矩阵
            % 各条支路编码输出
            branch_output = rem(memory_contents*g',2);
            % 从上至下表示当前状态s0,s1,s2……
            % 从左至右为经由支路编号0,1,2……
            % 内容为下一时刻状态s
            nextstate(s+1,t+1) = next_state;            % 状态转移矩阵
            % 从上至下表示当前状态s0,s1,s2……
            % 从左至右表示经由支路编号0,1,2……
            % 内容为相应分支输出编码
            output(s+1,t+1) = bin2dec(branch_output);   % 输出矩阵
        end
    end
    %-------------开始译码，得到幸存状态矩阵 -------------%
    % 状态度量矩阵
    % 第一列为当前时刻各状态的路径度量
    % 第二列为下一时刻各状态的路径度量（即更新后的状态度量）
    state_metric = zeros(number_of_states,2);
    % 网格深度
    depth_of_trellis = length(decode_input)/n;
    decode_input_matrix = reshape(decode_input,n,depth_of_trellis);
    % 幸存状态矩阵
    survivor_state = zeros(number_of_states,depth_of_trellis+1);
    % 各个状态的初始路径度量
    for i =1:N-1
        % 网格图从全零状态出发，直到所有状态都有路径到达
        for s = 0:2^(k*(N-i)):number_of_states-1
            % 对前一时刻状态到下一时刻状态之间的各条分支进行运算
            for t = 0:2^k-1
                % 分支度量
                branch_metric = 0;
                % 将各分支的编码输出以二进制形式表示
                bin_output = dec2bin(output(s+1,t+1),n);
                for j = 1:n
```

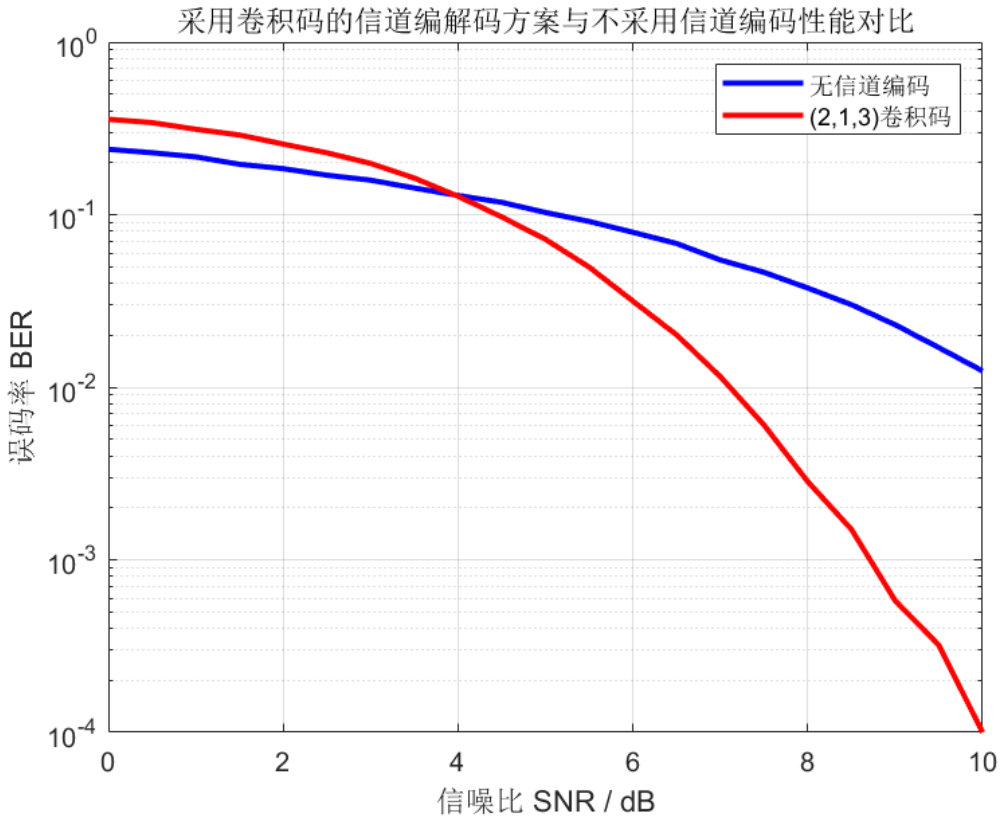
```
                    % 分支度量的计算
                    branch_metric = branch_metric +
metric_hard(decode_input_matrix(j,i),bin_output(j));
                end
                % 各个状态路径度量值的更新
                % 下一时刻路径度量=当前时刻路径度量+分支度量
                state_metric(nextstate(s+1,t+1)+1,2) = state_metric(s+1,1) +
branch_metric;
                % 幸存路径的存储
                % 一维坐标表示下一时刻状态
                % 二维坐标表示该状态在网格图中的列位置
                % 内容为当前时刻状态
                survivor_state(nextstate(s+1,t+1)+1,i+1) = s;
            end
        end
        % 对所有状态完成一次路径度量值计算后
        % 状态度量矩阵第一列（当前状态路径度量）
        % 与第二列（下一状态路径度量）对换
        % 方便下一时刻继续迭代更新
        state_metric = state_metric(:,2:-1:1);
    end
    % 各个状态的路径度量更新
    for i = N:depth_of_trellis-(N-1)
        % 记录某一状态的路径度量是否更新过
        flag = zeros(1,number_of_states);
        for s = 0:number_of_states-1
           for t = 0:2^k-1
               branch_metric = 0;
               bin_output = dec2bin(output(s+1,t+1),n);
               for j = 1:n
                  branch_metric = branch_metric +
metric_hard(decode_input_matrix(j,i),bin_output(j));
               end
               % 若某状态的路径度量未被更新
               % 或一次更新后的路径度量大于本次更新的路径度量
               % 则进行各状态路径度量值的更新

if((state_metric(nextstate(s+1,t+1)+1,2)>state_metric(s+1,1)+branch_metric) ||
flag(nextstate(s+1,t+1)+1) == 0)
                   state_metric(nextstate(s+1,t+1)+1,2) = state_metric(s+1,1)+
branch_metric;
                   survivor_state(nextstate(s+1,t+1)+1,i+1) = s;
                   % 一次更新后flag置为1
                   flag(nextstate(s+1,t+1)+1) = 1;
               end
           end
        end
        state_metric = state_metric(:,2:-1:1);
    end
    % 结尾译码：网格图回归全零状态
    for i = depth_of_trellis-(N-1)+1:depth_of_trellis
    flag = zeros(1,number_of_states);
    %上一比特存留的状态数
        last_stop_states = number_of_states/(2^((i-depth_of_trellis+N-2)*k));
        % 网格图上的各条路径最后都要回到同一个全零状态
        for s = 0:last_stop_states-1
            branch_metric = 0;
            bin_output = dec2bin(output(s+1,1),n);
```

```
            for j = 1:n
               branch_metric = branch_metric+
metric_hard(decode_input_matrix(j,i),bin_output(j));
            end
            if((state_metric(nextstate(s+1,1)+1,2) >
state_metric(s+1,1)+branch_metric) || flag(nextstate(s+1,1)+1) == 0)
                state_metric(nextstate(s+1,1)+1,2) = state_metric(s+1,1)+
branch_metric;
                survivor_state(nextstate(s+1,1)+1,i+1) = s;
                flag(nextstate(s+1,1)+1) = 1;
            end
        end
        state_metric = state_metric(:,2:-1:1);
    end
    %----------- 根据幸存状态矩阵开始逐步向前回溯，得到译码输出 ----------%
    sequence = zeros(1,depth_of_trellis+1);
    % 逐步向前回溯
    for i = 1:depth_of_trellis
       sequence(1,depth_of_trellis+1-i) =
survivor_state(sequence(1,depth_of_trellis+2-i)+1,depth_of_trellis+2-i);
    end
    % 译码输出
    decode_output_matrix = zeros(k,depth_of_trellis-N);
    for i = 1:depth_of_trellis-N
        % 由输入矩阵得到经由支路编号
        dec_decode_output = input(sequence(1,i)+1,sequence(1,i+1)+1);
        % 将支路编号转为二进制码元，即为相应的译码输出
        bin_decode_output = dec2bin(dec_decode_output,k);
        % 将每一分支的译码输出存入译码输出矩阵中
        decode_output_matrix(:,i) = bin_decode_output(k:-1:1)';
    end
    % 重新排列译码输出序列
    decode_output = reshape(decode_output_matrix,1,k*(depth_of_trellis-N));
end
```

上面这段代码模拟进行了维特比译码过程，主要步骤包括：

- **计算状态转移图**：根据生成矩阵，计算出有限状态机的所有可能状态，以及在各类状态转移的过程中所输出的结果；
- **计算幸存状态矩阵**：根据最大似然准则，在网格图中依据状态转移图和接收到的序列，依次计算每条状态转移路径的汉明距离，根据最大似然准则只保留到达各状态的多条路径中汉明距离最小的那条。每输入一个待解码码元，状态机内所有状态的最短到达路径都需要进行逐步计算、比对与更新。
- **结尾译码**：接收序列的末尾，需要使网格图全部回归零状态，这也对应着编码过程中所添加的（N-1）位。在这个过程中不会输入1码元，网络经过N-1步转移后，各状态均回到全0状态。
- **译码输出**：根据幸存状态矩阵开始逐步向前回溯，利用状态转移图的映射关系得到每一步所接受的码元是0还是1。直到回到序列起始位置，重新排列即得译码输出序列。

为了直观展示结果，选取了0到10，步长为0.5的多个信噪比数值，并将结果绘制为图像如下：



由图像可见，在高信噪比区域，(2,1,3)卷积码的误码率表现要显著高于无信道编码的情况，但在低信噪比区域，有信道编码时的误码率表现反而不如无信道编码的情况。该结果与理论分析结果相吻合。

>>>实验总结

通过本次实验，我掌握了信道编码的思想和基本原理，更加熟悉了卷积码的编码和解码原理，同时学到了利用matlab软件模拟进行信道编解码仿真的基本过程。

在本次实验中，还需利用matlab中许多其他函数。例如：

- `y = bin2dec(x)`：二进制转十进制函数。在解码过程中，涉及到各个状态（以01序列表示）和汉明距离的关系，以及最大似然判决等；matlab中使用该函数可以很方便地将输入01序列转换为对应的十进制数；
- `y = dec2bin(x, len)`：与上函数类似，可以实现十进制转二进制，其中 `len` 为输出二进制序列的长度；
- `Encode_Noise(Encode_Noise > 0.5) = 1; Encode_Noise(Encode_Noise < 0.5) = 0;`：这段代码是为了实现信道加噪后的简单判决，因为维特比译码过程必须针对全为01的序列，因此加噪过程可以简化为对输入序列直接进行0/1的改变，所以这里做了一个十分简易的门限判决。如果能结合BPSK、2FSK等调制解调过程与最佳接收机与门限判决过程，相信在相同信道信噪比的条件下，误码率会进一步降低。
- `y=mod(x,2)`：取模函数，因为编解码过程中的加法均为模2加，或称为异或，即1+1=0。

总之，本次实验紧接着通信原理理论课进行，在认识到信道编码的卷积码算法理论基础后，立即就可以通过matlab仿真实验来巩固这一理论，令我取得了很大收获。同时，在程序输出图像时，我一开始还以为程序有问题，因为直觉告诉我有信道编码时的误码率表现（红色曲线）显然是一定优于无信道编码时的表现的（蓝色曲线），一直检查代码也没发现问题，后来查阅课本才发现在低信噪比区域，有信道编码时的误码率表现反而不如无信道编码的情况，两者是互相吻合的。这也进一步加深了我对于信道编码的意义和影响的理解。