AWGN 信道模型与调制 汉明码编码与译码

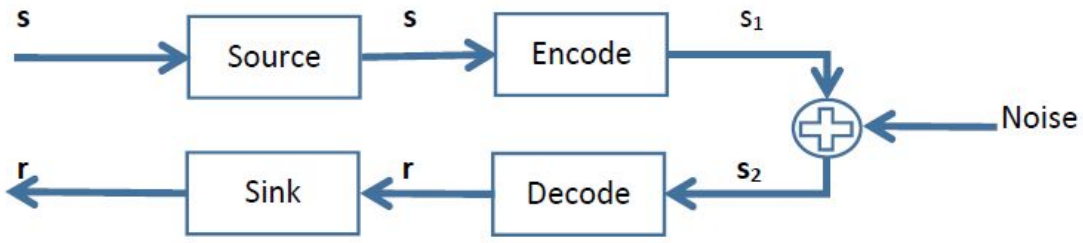
小组成员	学号	职务
喻乐	13332024	组长
方锡鑫	13331049	组员
刘健诚	13331169	组员

一、实验目的

1）掌握编码调制的基本研究手段
2）学会随机数程序的使用
3）学会模块化程序
4）学 Monte-Carl 仿真
5）掌握汉明码的编码与译码原理与方法

二、实验工具
开发语言：C++
开发工具：Visual Studio, Matlab

三、实验内容
A. BPSK 不编码
利用 Monte-Carlo 仿真统计检测的误符号率和误比特率，以下是部分具体步骤：
1.实现流程图



2.重要部分分析
1) 信号产生函数
实现步骤：
A.调用 Uniform 产生均匀分布
B.再对产生的随机数（0~1）跟 0.5 进行比较
C.比 0.5 大则产生 1 信号位，否则产生 0 信号位，这样产生 0/1 概率随机
D.通过一个我们设定的长度循环控制参数，即可产生一个随机数列（0/1）比特串
s
具体代码如下：

```
int* generator(int length) {
        double token;
        CLCRandNum temp;
        for (int i = 0; i < length; i++) {
                token = temp.Uniform();   // generate number between 0~1
                source_arr[i] = token > 0.5 ? 1 : 0;
        }
        return source_arr;
}
```

2) 编码函数

实现步骤：

A.对于传进来的比特串的每一位比特与 0.5 进行比较

B.若比 0.5 大则编码为 1，否则编码为-1，这样可将 0/1 编码为-1/1，方便后面的处理

C.通过一个我们设定的长度循环控制参数，即可编出一串（-1/1）的信号编码 s1

具体代码如下：

```
int *encode(int *src_arr, int length) {
        for (int i = 0; i < length; i++) {
                encode_arr[i] = (src_arr[i] > 0.5) ? 1 : -1;
        }
        return encode_arr;
}
```

3) 信道传输函数（加噪声）

实现步骤：

A.利用 Normal 产生一个高斯噪声

B.将 2）得到的编码信号加上噪声信号

C.通过一个我们设定的长度作为循环控制参数，即可得到一串加上噪声干扰的信号 s2

具体代码如下：

```
double *add_noise(int *encode_arr, int length, double delta) {
        CLCRandNum temp;
        temp.Normal(total_arr, length);
        for (int i = 0; i < length; i++) {
                total_arr[i] = delta*total_arr[i] + encode_arr[i];
        }
        return total_arr;
}
```

4) 译码函数

实现步骤：

A.将 3）得到的信号 θ 的每一位与 0 比较

B.比 0 大则译码为 1，否则译码为 0,

D.通过一个我们设定的长度作为循环控制参数，即可译码得到模拟接收到的信号 r

具体代码如下：

```
int *decode(double *total_arr, int length) {
        for (int i = 0; i < length; i++) {
                decode_arr[i] = total_arr[i] > 0 ? 1 : 0;
        }
        return decode_arr;
}
```

5) 计算误码函数

实现步骤：

通过对比产生信号 s 与接收信 r 每一位是否相同，遇到不同错码数加 1，通过循环，最终得到误码数，除以总的比特数即可得到我们想要的误码率

具体代码如下：

```
int error_num(int *source_arr, int *decode_arr, int length) {
        int error_num = 0;
        for (int i = 0; i < length; i++) {
                if (source_arr[i] != decode_arr[i]) {
                        error_num++;
                }
        }
        return error_num;
}
```

最后我们利用 matlab 程序读 c++程序生成的数据文件，利用大量的数据生成更加准确的曲线。

B. BPSK+Hamming

1） 编码函数

实现步骤：

利用定义的生成矩阵 G 对消息进行编码。将每 4 个比特组成一组消息进行编码(如果不满四位的不进行编码)。利用矩阵乘法，消息的 4 个比特依次与生成矩阵的每一列的对应位相乘，然后进行模 2 加法，得到 7 个比特。7 个比特组成一个汉明码。在这里，我们为了让代码更加泛化，采用了宏定义的方式，用 BLOCK 代表编码前的组长度，用 CODEWORD 代表码字的长度，在（7，4）汉明码中，BLOCK = 4，CODEWORD = 7.

具体代码如下：

```
int* Hamming_coding(int* uncoding_seq, int length) {
    int message_block[BLOCK];
    codeword_length = -1;
    int i, j, k;
    for (i = 0; i < length; ) {
        for (j = 0; j < BLOCK && i < length; j++, i++) {
            message_block[j] = uncoding_seq[i];
        }
        if (i >= length && j < BLOCK) {
            for (k = 0; k < j; k++) {
                codeword_arr[++codeword_length] = message_block[k];
            }
            break;
        }
        for (j = 0; j < CODEWORD; j++) {
            codeword_arr[++codeword_length] = 0;
            for (k = 0; k < BLOCK; k++) {
                codeword_arr[codeword_length] += generate_matrix[k][j] * message_block[k];
            }
            codeword_arr[codeword_length] %= 2;
        }
    }
    codeword_length++;
    codeword_arr[(L / BLOCK + 2)*CODEWORD - 1] = codeword_length;
    return codeword_arr;
}
```

2） 译码函数

实现步骤：

1.利用定义的生成矩阵 G 得到对应的校验矩阵 H。对接收端接收的比特串，计算校正子 s（长度为 3 的向量），也叫做错误模式。

2.若校正子 s 为 0，则认为接收的比特串即为发送的比特串，若校正子不为 0，我们就在对应的陪集首矩阵中相应的错误模式，然后利用错误模式来纠正当前的码字，在这里需要注意的就是所有操作都是向量的模 2 加法，对应着 GF(2),这步操作就是将 7 位的比特串映射到距离它最近（距离为 0 或 1）的码字上。

具体代码如下：

```
int *Hamming_correction(int* error_seq, int length) {
    int error_pattern[CODEWORD - BLOCK];
    int codeword_block[CODEWORD];
    int i, j, k;
    for (i = 0; i < length; ) {
        for (j = 0; j < CODEWORD; j++, i++) {
            codeword_block[j] = error_seq[i];
        }
        if (i >= length && j < CODEWORD) {
            break;
        }
        for (j = 0; j < CODEWORD - BLOCK; j++) {
            error_pattern[j] = 0;
            for (k = 0; k < CODEWORD; k++) {
                error_pattern[j] += codeword_block[k] * check_matrix[j][k];
            }
            error_pattern[j] %= 2;
        }
        int index = 0;
        for (j = 0; j < CODEWORD - BLOCK; j++) {
            index += error_pattern[CODEWORD - BLOCK-j-1]*pow(2,j);
        }
        for (j = 0; j < CODEWORD; j++) {
            error_seq[i - CODEWORD + j] += coset[index][j];
            error_seq[i - CODEWORD + j] %= 2;
        }
    }
    return error_seq;
}
```

3.取译码后的比特串中的后四位作为接收的消息。对应代码如下：

```
int *Hamming_decoding(int* corrected_seq, int length) {
    int codeword_block[CODEWORD];
    message_length = -1;
    int i, j, k;
    for (i = 0; i < length;) {
        for (j = 0; j < CODEWORD && i < length; j++,i++) {
            codeword_block[j] = corrected_seq[i];
        }
        if (i >= length && j < CODEWORD) {
            for (k = 0; k < j; k++) {
                message_arr[++message_length] = codeword_block[k];
            }
            break;
        }
        for (j = BLOCK-1; j < CODEWORD; j++) {
            message_arr[++message_length] = codeword_block[j];
        }
    }
    message_length++;
    message_arr[L + BLOCK * 4 - 1] = message_length;
    return message_arr;
}
```

四、实验结果

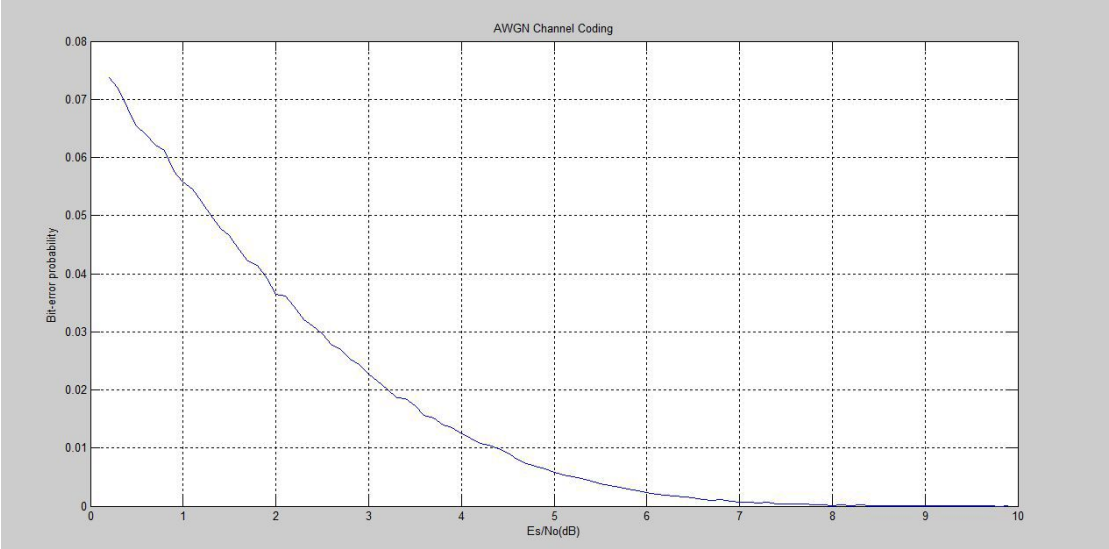
A. BPSK 不编码

1.预期结果

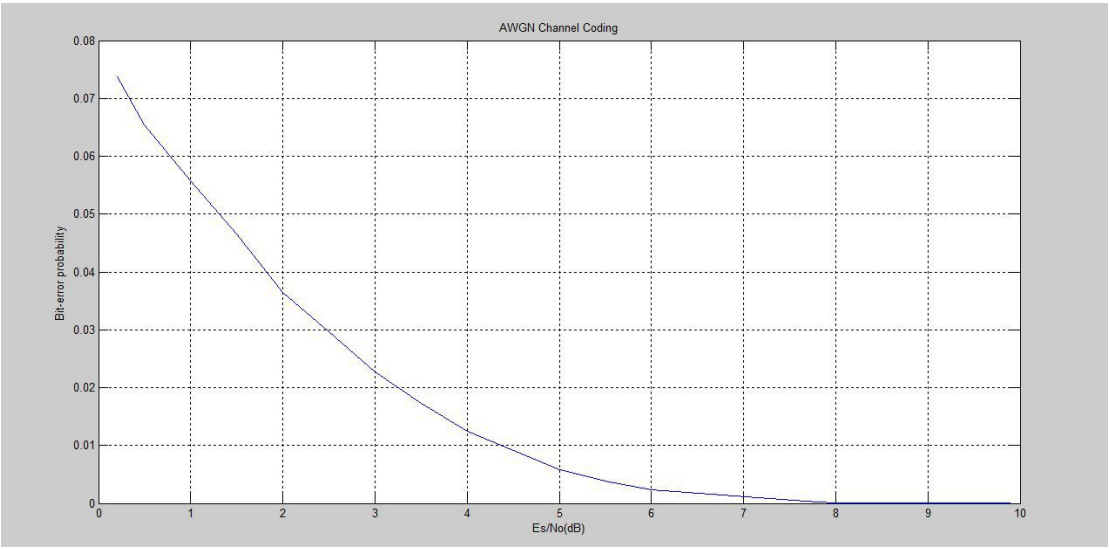
随着E/No 的增大，误符号率和误比特率都会随之下降，这是由于高斯分布的方差会随着SNR增大而减少，所以对应从-1 变为1 的概率（出错概率）也会随之减少。

2.实验结果

用 MATLAB 画出误符号率和误比特率随 E/No 变化而变化的曲线，如下图所示



在这里我们每隔0.1 生成一组（SNR，Bit-error Probability），将数据写入到文件中，然后编写一个matlab 程序从文件中读取数据并在屏幕上画出对应曲线，结果如上图，曲线近乎光滑，其中出现抖动是基于概率论的知识，即在实验次数不是趋于+∞时，概率不等于频率，后来我们将数据增大，得到如下曲线：

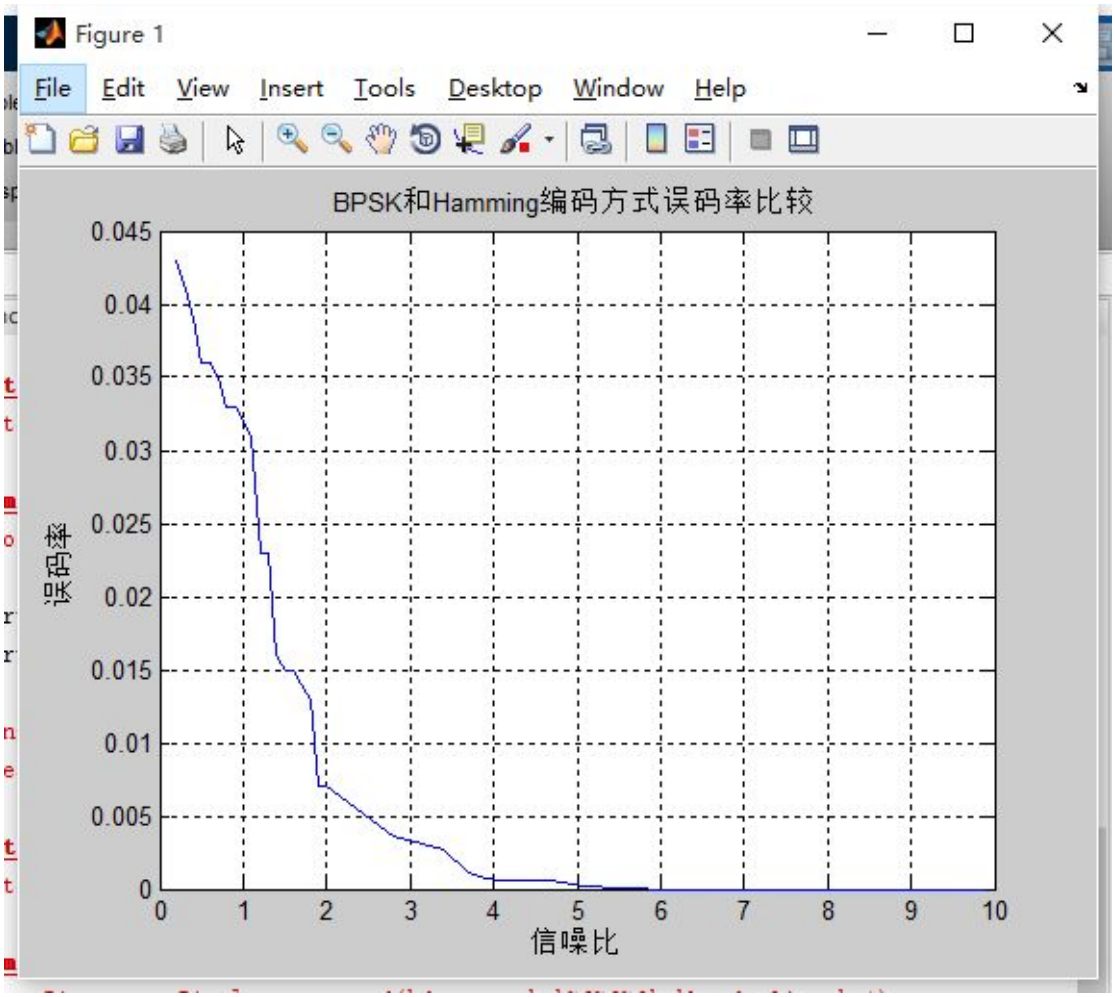


这和我们预期的结果相吻合，即随着E/No 的增大，误符号率和误比特率都会随之下降，这和老师 ppt 上图像相一致。

B. BPSK+Hamming

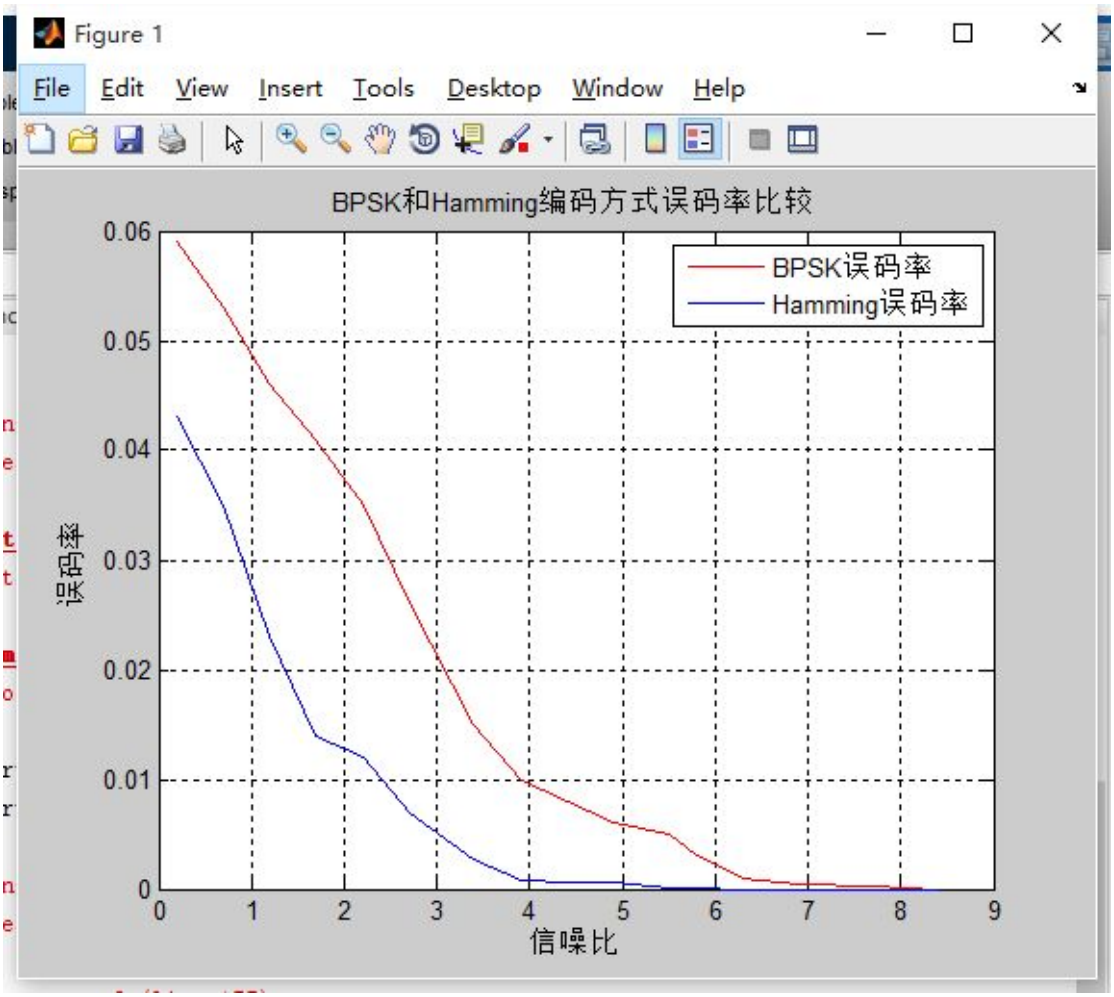
实验结果：

用 MATLAB 画出误符号率和误比特率随 E/No 变化而变化的曲线，如下图所示



由图可知，随着 E/No 的增大，误符号率和误比特率都会随之下降，而且由于 Hamming 自带的纠错功能，下降的幅度会比 BPSK 更加明显，而且会比 BPSK 的编码方式更快的趋向于 0。

C. BPSK 不编码与汉明码编码对比



通过数据对比可知，采用汉明码编码的误码率在相同的信噪比的情况下要优于 BPSK 编码方式，因为在 4 位中发生一个比特错误的概率远远高于在 7 位中同时发生两个比特错误的概率（Hamming 编码可以纠正一个比特错误）。在相同误码率情况下，信噪比增益大约为 1.5，这说明采用 Hamming 编码可以提高可靠信道传输的概率。

五、实验结论

通过这次实验，对于整个编码译码过程有了更深的理解，这次实现的AWGN 信道编码可是说是一种比较简单的编码方式，但是可以有比较好的效果，整个编码过程可以按照信道传输模型分为若干个模块，采用一种自顶向下的方法就可以实现各个模块的功能，再根据题目的具体要求，给定一个SNR，算出一个高斯分布的方差，将它加入到信道噪声中，就可以对应算出在该SNR 下的误比特率。

通过这次实验，也越发感到编码方式在信道传输中的重要性，在老师给出编码文档之前，我也尝试过用自己的想法实现这个实验，当时唯一缺少的就是将0~1 映射到-1~1 这样一个步骤，当时算出的误码率相当高，在根据老师的文档加入了这样一个过程之后，发现误码率明显降低了，后来经过思考，发现-1~1 这样的映射可以避免将标准高斯分布沿x 轴平移，只需要乘上对应的标准差即可。这次实验最深的体会还是一种数学建模的方法，计算机的理论基础是数学，我们根据数学理论构建符合要求的模型，在利用编程的方法将这个模型实现出来，不断根据模型实现效果反过来对模型进行修改，利用类似监督学习的方法不断对模型和编程进行完善，直到最后实现功能。

Hamming编码实验结论

通过编写Hamming码的程序，对于错误纠正和通信中的编码译码方式有了更深的理解，这次的实验只是在上次的实验上加了Hamming.h和Hamming.cpp两个文件，但是前前后后debug了很多次，主要的原因可能是之前编写BPSK时，没有考虑到后面实现更加复杂的编码时能够用到对应的代码，所以BPSK的函数没有实现很好的模块化，而在Hamming码编写中又要重新去实现BPSK相关步骤，就会使工作量变得非常大。所以在这次的程序中，我们虽然只是实现了（7，4）汉明码，但是我们把所有的变量名和方法都抽象出来了，比如我们纠错时候使用的是陪集首的纠错方法，这种方法对于（15，11）或者其他汉明码都适用，而且我们在编写程序时候尽量去使用一些宏定义，这样的话，如果下次要求实现（15，11）汉明码的时候，只需要修改一些宏就好了，这样代码的复用性得到了增强。

```
using namespace std;
#define BLOCK 4
#define CODEWORD 7
#define L 1000
```

```
int *Hamming_correction(int* error_seq, int length) {
    int error_pattern[CODEWORD - BLOCK];
    int codeword_block[CODEWORD];
    int i, j, k;
    for (i = 0; i < length; ) {
        for (j = 0; j < CODEWORD; j++, i++) {
            codeword_block[j] = error_seq[i];
        }
        if (i >= length && j < CODEWORD) {
            break;
        }
        for (j = 0; j < CODEWORD - BLOCK; j++) {
            error_pattern[j] = 0;
            for (k = 0; k < CODEWORD; k++) {
                error_pattern[j] += codeword_block[k] * check_matrix[j][k];
            }
            error_pattern[j] %= 2;
        }
        int index = 0;
        for (j = 0; j < CODEWORD - BLOCK; j++) {
            index += error_pattern[CODEWORD - BLOCK-j-1]*pow(2,j);
        }
        for (j = 0; j < CODEWORD; j++) {
            error_seq[i - CODEWORD + j] += coset[index][j];
            error_seq[i - CODEWORD + j] %= 2;
        }
    }
    return error_seq;
}
```

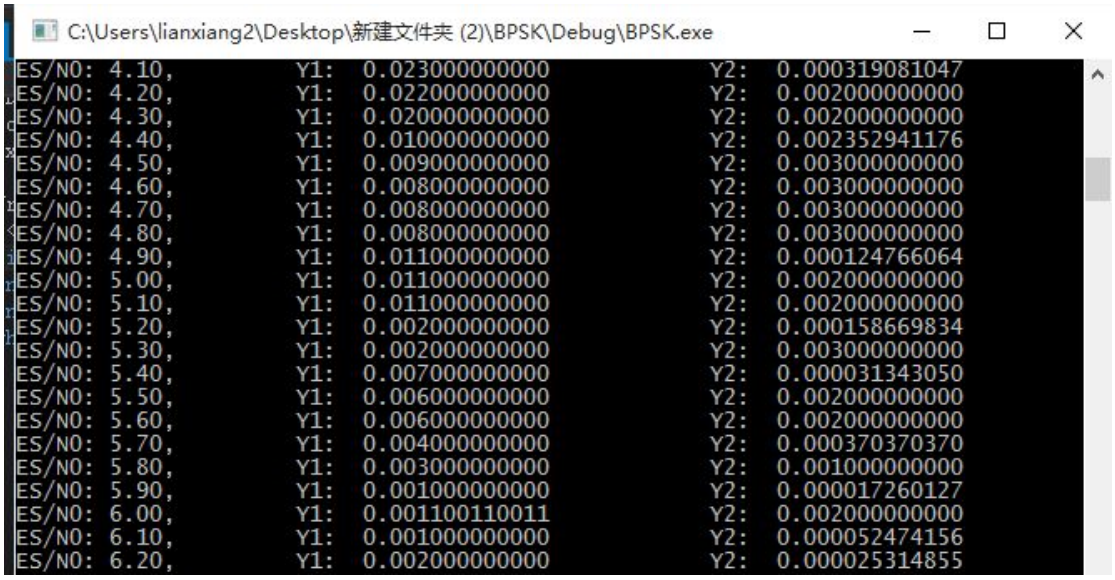
通过这次实现汉明码，同样对于效率和冗余有了更深的理解，汉明码这种方式比BPSK的性能有了很大的增益，但是冗余位数太多，会造成时间上的消耗，而且计算陪集首比较复杂，做矩阵运算复杂度约为n^3。

附：程序输出结果为：
其中 X 为 SNR，Y1 对应的是 BPSK 的误比特率，Y2 对应的是 Hamming 编码的误比特率。

```
   Es/No          BPSK-Bit-Error Rate                  Hamming-Bit-Error Rate
ES/N0: 0.20,          Y1:  0.064000000000          Y2:  0.028000000000
ES/N0: 0.30,          Y1:  0.061000000000          Y2:  0.025000000000
ES/N0: 0.40,          Y1:  0.059000000000          Y2:  0.022000000000
ES/N0: 0.50,          Y1:  0.057000000000          Y2:  0.021000000000
ES/N0: 0.60,          Y1:  0.056000000000          Y2:  0.019000000000
ES/N0: 0.70,          Y1:  0.052000000000          Y2:  0.017000000000
ES/N0: 0.80,          Y1:  0.052000000000          Y2:  0.017000000000
ES/N0: 0.90,          Y1:  0.050000000000          Y2:  0.015000000000
ES/N0: 1.00,          Y1:  0.048000000000          Y2:  0.015000000000
ES/N0: 1.10,          Y1:  0.057000000000          Y2:  0.020196078431
ES/N0: 1.20,          Y1:  0.053000000000          Y2:  0.024000000000
ES/N0: 1.30,          Y1:  0.052000000000          Y2:  0.022000000000
ES/N0: 1.40,          Y1:  0.051000000000          Y2:  0.020000000000
ES/N0: 1.50,          Y1:  0.050000000000          Y2:  0.015000000000
ES/N0: 1.60,          Y1:  0.048000000000          Y2:  0.008000000000
ES/N0: 1.70,          Y1:  0.048000000000          Y2:  0.008000000000
ES/N0: 1.80,          Y1:  0.046000000000          Y2:  0.008000000000
ES/N0: 1.90,          Y1:  0.045000000000          Y2:  0.008000000000
ES/N0: 2.00,          Y1:  0.044000000000          Y2:  0.008000000000
ES/N0: 2.10,          Y1:  0.042000000000          Y2:  0.008000000000
ES/N0: 2.20,          Y1:  0.039000000000          Y2:  0.008000000000
ES/N0: 2.30,          Y1:  0.035000000000          Y2:  0.006000000000
ES/N0: 2.40,          Y1:  0.033000000000          Y2:  0.004000000000
```

C:\Users\lianxiang2\Desktop\新建文件夹 (2)\BPSK\Debug\BPSK.exe

```
ES/N0: 2.30,          Y1:  0.035000000000          Y2:  0.006000000000
ES/N0: 2.40,          Y1:  0.033000000000          Y2:  0.004000000000
ES/N0: 2.50,          Y1:  0.028000000000          Y2:  0.004000000000
ES/N0: 2.60,          Y1:  0.025000000000          Y2:  0.002000000000
ES/N0: 2.70,          Y1:  0.022000000000          Y2:  0.002000000000
ES/N0: 2.80,          Y1:  0.022000000000          Y2:  0.002000000000
ES/N0: 2.90,          Y1:  0.017000000000          Y2:  0.002840909091
ES/N0: 3.00,          Y1:  0.016000000000          Y2:  0.003000000000
ES/N0: 3.10,          Y1:  0.014000000000          Y2:  0.003000000000
ES/N0: 3.20,          Y1:  0.012000000000          Y2:  0.003000000000
ES/N0: 3.30,          Y1:  0.012000000000          Y2:  0.003000000000
ES/N0: 3.40,          Y1:  0.015000000000          Y2:  0.004529411765
ES/N0: 3.50,          Y1:  0.015000000000          Y2:  0.005000000000
ES/N0: 3.60,          Y1:  0.015000000000          Y2:  0.005000000000
ES/N0: 3.70,          Y1:  0.013000000000          Y2:  0.002000000000
ES/N0: 3.80,          Y1:  0.012000000000          Y2:  0.002000000000
ES/N0: 3.90,          Y1:  0.005000000000          Y2:  0.000395100751
ES/N0: 4.00,          Y1:  0.005000000000          Y2:  0.002000000000
ES/N0: 4.10,          Y1:  0.023000000000          Y2:  0.000319081047
ES/N0: 4.20,          Y1:  0.022000000000          Y2:  0.002000000000
ES/N0: 4.30,          Y1:  0.020000000000          Y2:  0.002000000000
ES/N0: 4.40,          Y1:  0.010000000000          Y2:  0.002352941176
ES/N0: 4.50,          Y1:  0.009000000000          Y2:  0.003000000000
ES/N0: 4.60,          Y1:  0.008000000000          Y2:  0.003000000000
微软拼音 半 :
```


C:\Users\lianxiang2\Desktop\新建文件夹 (2)\BPSK\Debug\BPSK.exe
ES/NO: 4.10, Y1: 0.023000000000 Y2: 0.000319081047
ES/NO: 4.20, Y1: 0.022000000000 Y2: 0.002000000000
ES/NO: 4.30, Y1: 0.020000000000 Y2: 0.002000000000
ES/NO: 4.40, Y1: 0.010000000000 Y2: 0.002352941176
ES/NO: 4.50, Y1: 0.009000000000 Y2: 0.003000000000
ES/NO: 4.60, Y1: 0.008000000000 Y2: 0.003000000000
ES/NO: 4.70, Y1: 0.008000000000 Y2: 0.003000000000
ES/NO: 4.80, Y1: 0.008000000000 Y2: 0.003000000000
ES/NO: 4.90, Y1: 0.011000000000 Y2: 0.000124766064
ES/NO: 5.00, Y1: 0.011000000000 Y2: 0.002000000000
ES/NO: 5.10, Y1: 0.011000000000 Y2: 0.002000000000
ES/NO: 5.20, Y1: 0.002000000000 Y2: 0.000158669834
ES/NO: 5.30, Y1: 0.002000000000 Y2: 0.003000000000
ES/NO: 5.40, Y1: 0.007000000000 Y2: 0.000031343050
ES/NO: 5.50, Y1: 0.006000000000 Y2: 0.002000000000
ES/NO: 5.60, Y1: 0.006000000000 Y2: 0.002000000000
ES/NO: 5.70, Y1: 0.004000000000 Y2: 0.000370370370
ES/NO: 5.80, Y1: 0.003000000000 Y2: 0.001000000000
ES/NO: 5.90, Y1: 0.001000000000 Y2: 0.000017260127
ES/NO: 6.00, Y1: 0.001100110011 Y2: 0.002000000000
ES/NO: 6.10, Y1: 0.001000000000 Y2: 0.000052474156
ES/NO: 6.20, Y1: 0.002000000000 Y2: 0.000025314855