

# AWGN 信道模型与调制

小组成员	学号	职务
喻乐	13332024	组长
方锡鑫	13331049	组员
刘建成	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) 学会 Monte-Carlo 仿真

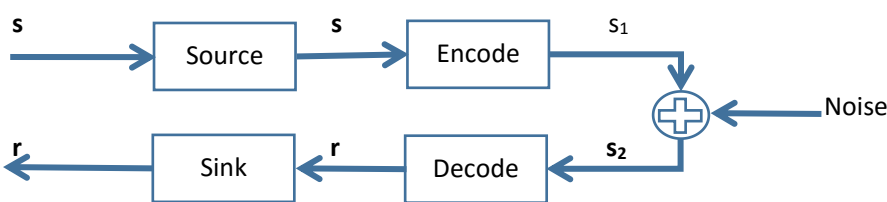
## 二、实验工具

开发语言：C++  
开发工具：Visual Studio, Matlab

## 三、实验内容

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 **s<sub>1</sub>**

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 **s<sub>2</sub>**

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**0**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**

```
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

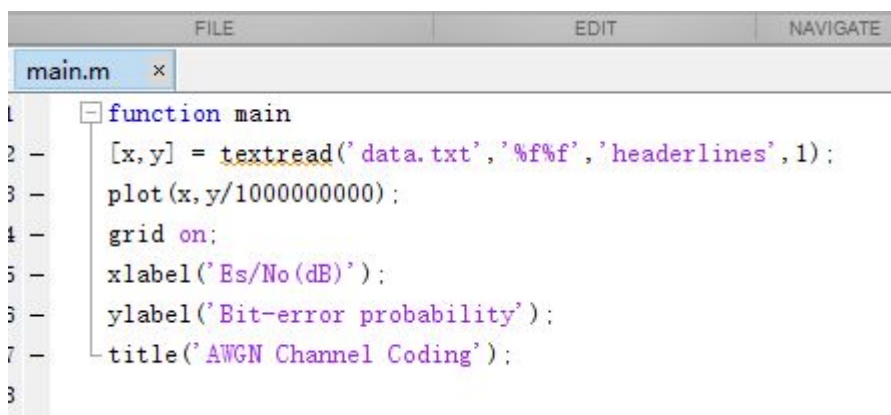
实现步骤：

通过对比产生信号 **s** 与接收信号 **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 程序读取 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。



```
function main
[x,y] = textread('data.txt','%f%f','headerlines',1);
plot(x,y/1000000000);
grid on;
xlabel('Es/No(dB)');
ylabel('Bit-error probability');
title('AWGN Channel Coding');
```

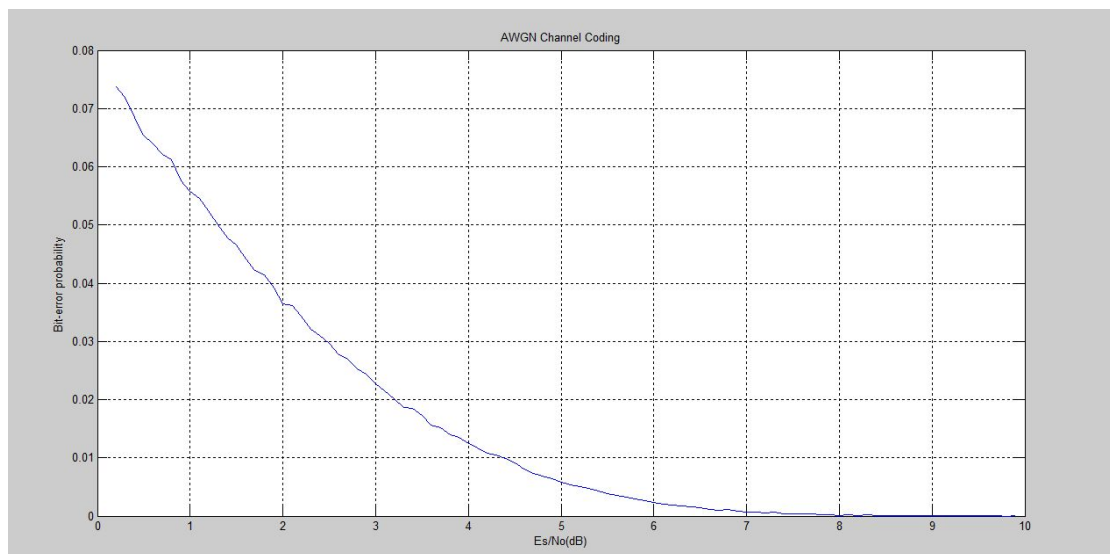
## 四、实验结果

### 1. 预期结果

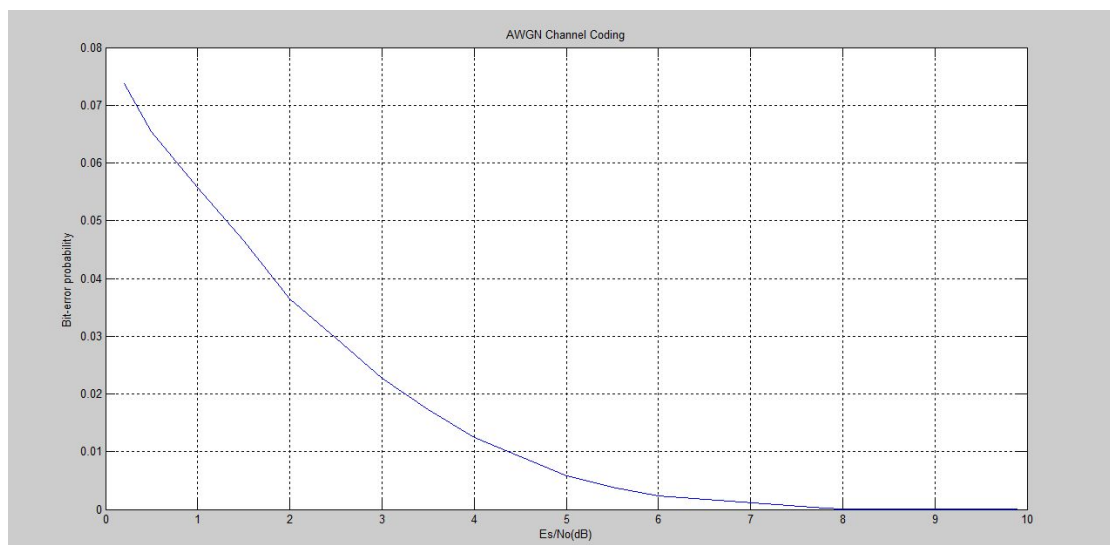
随着  $E/N_0$  的增大，误符号率和误比特率都会随之下降，这是由于高斯分布的方差会随着 SNR 增大而减少，所以对应从 -1 变为 1 的概率（出错概率）也会随之减少。

## 2.实验结果

用 MATLAB 画出误符号率和误比特率随  $E/N_0$  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  $+\infty$  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  $E/N_0$  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。

## 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  $0 \sim 1$  映射到  $-1 \sim 1$  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  $-1 \sim 1$  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,知道最后实现功能。

附: 程序输出结果为:

其中  $X$  为 SNR,  $Y$  对应的是误比特率

```
0.226196
Es/No      Bit-Error Probability
X: 0.20000000000000001,      Y: 73802161*10^(-9)
X: 0.29999999999999999,      Y: 71976769*10^(-9)
X: 0.40000000000000002,      Y: 68480161*10^(-9)
X: 0.5,      Y: 65408587*10^(-9)
X: 0.59999999999999998,      Y: 63865286*10^(-9)
X: 0.69999999999999996,      Y: 62150847*10^(-9)
X: 0.80000000000000004,      Y: 61163551*10^(-9)
X: 0.90000000000000002,      Y: 57574253*10^(-9)
X: 1,      Y: 55714605*10^(-9)
X: 1.10000000000000001,      Y: 54606610*10^(-9)
X: 1.2,      Y: 52489447*10^(-9)
X: 1.3,      Y: 50056986*10^(-9)
X: 1.3999999999999999,      Y: 47731812*10^(-9)
X: 1.5,      Y: 46588995*10^(-9)
X: 1.60000000000000001,      Y: 44127080*10^(-9)
X: 1.7,      Y: 42188292*10^(-9)
X: 1.8,      Y: 41381838*10^(-9)
X: 1.8999999999999999,      Y: 39237511*10^(-9)
X: 1.7,      Y: 42188292*10^(-9)
X: 1.8,      Y: 41381838*10^(-9)
X: 1.8999999999999999,      Y: 39237511*10^(-9)
X: 2,      Y: 36430051*10^(-9)
X: 2.10000000000000001,      Y: 36094552*10^(-9)
X: 2.20000000000000002,      Y: 34272436*10^(-9)
X: 2.29999999999999998,      Y: 32078868*10^(-9)
X: 2.3999999999999999,      Y: 31045056*10^(-9)
X: 2.5,      Y: 29656583*10^(-9)
X: 2.60000000000000001,      Y: 27669939*10^(-9)
X: 2.70000000000000002,      Y: 26923917*10^(-9)
X: 2.79999999999999998,      Y: 25219885*10^(-9)
X: 2.8999999999999999,      Y: 24393929*10^(-9)
X: 3,      Y: 22709208*10^(-9)
X: 3.10000000000000001,      Y: 21478158*10^(-9)
X: 3.20000000000000002,      Y: 20077779*10^(-9)
X: 3.29999999999999998,      Y: 18701537*10^(-9)
X: 3.3999999999999999,      Y: 18422546*10^(-9)
X: 3.5,      Y: 17296234*10^(-9)
X: 3.60000000000000001,      Y: 15671418*10^(-9)
X: 3.70000000000000002,      Y: 15200422*10^(-9)
X: 3.79999999999999998,      Y: 13938371*10^(-9)
X: 3.8999999999999999,      Y: 13534144*10^(-9)
X: 4,      Y: 12428073*10^(-9)
```



```

X: 4.0999999999999996, Y: 11765192*10^(-9)
X: 4.2000000000000002, Y: 10836584*10^(-9)
X: 4.2999999999999998, Y: 10424618*10^(-9)
X: 4.4000000000000004, Y: 9937094*10^(-9)
X: 4.5, Y: 9070595*10^(-9)
X: 4.5999999999999996, Y: 8049696*10^(-9)
X: 4.7000000000000002, Y: 7329332*10^(-9)
X: 4.7999999999999998, Y: 6891583*10^(-9)
X: 4.9000000000000004, Y: 6484565*10^(-9)
X: 5, Y: 5780599*10^(-9)
X: 5.0999999999999996, Y: 5289960*10^(-9)
X: 5.2000000000000002, Y: 4986907*10^(-9)
X: 5.2999999999999998, Y: 4683227*10^(-9)
X: 5.4000000000000004, Y: 4245797*10^(-9)
X: 5.5, Y: 3807673*10^(-9)
X: 5.5999999999999996, Y: 3499573*10^(-9)
X: 5.7000000000000002, Y: 3252639*10^(-9)
X: 5.7999999999999998, Y: 3012739*10^(-9)
X: 5.9000000000000004, Y: 2694867*10^(-9)
X: 6, Y: 2372229*10^(-9)
X: 6.0999999999999996, Y: 2031383*10^(-9)
X: 6.2000000000000002, Y: 1891326*10^(-9)
X: 6.2999999999999998, Y: 1694262*10^(-9)
X: 6.4000000000000004, Y: 1653122*10^(-9)

```

```

X: 6.5, Y: 1393568*10^(-9)
X: 6.5999999999999996, Y: 1199717*10^(-9)
X: 6.7000000000000002, Y: 996600*10^(-9)
X: 6.7999999999999998, Y: 1078984*10^(-9)
X: 6.9000000000000004, Y: 813825*10^(-9)
X: 7, Y: 692027*10^(-9)
X: 7.0999999999999996, Y: 709931*10^(-9)
X: 7.2000000000000002, Y: 576084*10^(-9)
X: 7.2999999999999998, Y: 610637*10^(-9)
X: 7.4000000000000004, Y: 400727*10^(-9)
X: 7.5, Y: 386174*10^(-9)
X: 7.5999999999999996, Y: 426947*10^(-9)
X: 7.7000000000000002, Y: 336864*10^(-9)
X: 7.7999999999999998, Y: 243054*10^(-9)
X: 7.9000000000000004, Y: 250710*10^(-9)
X: 8, Y: 145399*10^(-9)
X: 8.0999999999999996, Y: 195448*10^(-9)
X: 8.1999999999999993, Y: 116258*10^(-9)
X: 8.3000000000000007, Y: 152403*10^(-9)
X: 8.4000000000000004, Y: 102033*10^(-9)
X: 8.5, Y: 73671*10^(-9)
X: 8.5999999999999996, Y: 82041*10^(-9)
X: 8.6999999999999993, Y: 55166*10^(-9)
X: 8.8000000000000007, Y: 71418*10^(-9)

```

```
X: 8, Y: 145399*10^(-9)
X: 8.0999999999999996, Y: 195448*10^(-9)
X: 8.1999999999999993, Y: 116258*10^(-9)
X: 8.3000000000000007, Y: 152403*10^(-9)
X: 8.4000000000000004, Y: 102033*10^(-9)
X: 8.5, Y: 73671*10^(-9)
X: 8.5999999999999996, Y: 82041*10^(-9)
X: 8.6999999999999993, Y: 55166*10^(-9)
X: 8.8000000000000007, Y: 71418*10^(-9)
X: 8.9000000000000004, Y: 20826*10^(-9)
X: 9, Y: 58265*10^(-9)
X: 9.0999999999999996, Y: 17503*10^(-9)
X: 9.1999999999999993, Y: 13218*10^(-9)
X: 9.3000000000000007, Y: 25276*10^(-9)
X: 9.4000000000000004, Y: 22713*10^(-9)
X: 9.5, Y: 12862*10^(-9)
X: 9.5999999999999996, Y: 13586*10^(-9)
X: 9.6999999999999993, Y: 13582*10^(-9)
X: 9.8000000000000007, Y: 0*10^(-9)
X: 9.9000000000000004, Y: 8909*10^(-9)
```