LDPC码编译码程序设计

学号：1901213306 姓名：杨庆龙

# 要求

编写(2016, 1008)LDPC码的编码器与译码器，并搭建仿真系统统计误码性能。系统框图如图Fig1所示



Fig1. 仿真系统框图

# 设计内容

该实验涉及的设计内容有以下几项，LDPC码校验矩阵，随机信源，LDPC编码器，BPSK调制解调器，AWGN信道，LDPC码译码器。其中译码器共需要四种，分别为和积算法译码器，最小和算法译码器，归一化最小和算法译码器和偏置最小和算法译码器。

# 编解码原理

## 编码原理

本系统采用码长为2016比特，1/2码率的系统码，其H矩阵如表达式如下所示.



对于系统码，H矩阵可分为[Hp, Hs]两个部分，其中，Hp对应校验比特部分，大小为mbz\*mbz，Hs对应信息比特部分，大小为mbz\*kbz，其中kb=nb-mb。Hp具有如下所示的规则结构:

右上角a表示的矩阵的结构比较特殊，如下所示

由于采用的是系统码，所以求出校验比特p=[p1,p2,……pmbz]之后，再与信息比特s=[s1,s2,……pkbz]拼接在一起即可得到编码结果。又考虑到校验矩阵结构特殊，所以采用如下所示的编码方法，以减小编码计算量。

步骤一：使用信息矢量s，依据如下公式计算出中间结果w

步骤二：使用中间结果w计算出校验矢量p，具体计算公式如下所示

## 解码原理

### 和积算法

#### 初始化

每个比特从信道接收到的信道信息作为初始置信度，即变量节点的初始置信度uj->i=u0->i，其中u0->i表示第i个变量节点从信道接收到的LLR，如图Fig2所示。



Fig2. 初始化节点图

#### 置信度传递

每个变量节点要将自己的置信度传递给具有校验关系的校验节点。第i个变量节点传递给第j个校验节点的置信度计算方法如图Fig3所示，其中，div表示第i个变量节点的度。



Fig3. 变量节点置信度传递图

校验节点依据变量节点传递来的置信度，更新每一个变量节点的置信度。第j个校验节点为第i个变量节点更新的置信度如图Fig4所示，其中djv表示第j个校验节点的度。



Fig4. 校验节点置信度传递图

#### 判决

置信度更新完毕后依据如下关系进行硬判决。

如果所有的校验关系都得到满足，即满足公式Hxt=0。则译码结束，取出靠后的kbz个比特即为信息位的估计值。若校验关系不满足，则返回迭代第二步继续迭代，直到迭代次数上限。

### 最小和算法

最小和算法的流程与和积算法完全一致，唯一的区别在于将和积算法的积运算简化为如下所示的最小和运算。

### 归一化最小和算法

归一化最小和算法建立在最小和算法之上，将较为粗糙的最小和算法更新为带有归一化系数α的最小和算法，具体公式如下所示，以减小更新幅度。

### 偏置最小和算法

偏置最小和算法也是建立在最小和算法之上，在变量节点更新的置信度的基础上引入了偏置量β，具体公式如下所示

# 程序设计与实现

## 语言选择

由于个人对Matlab并不是很熟悉，考虑到助教说过用matlab需要跑一整天，所以用Python可能要一周。加之编解码过程有大量的并行计算，所以使用和C语言性能类似，但对并行化支持更好的golang作为开发语言。

## 随机信源

随机信源模块位于source包内，该模块会等概率地产生给定长度的随机01序列。当序列长度为1e5时，可保证0或1在整个序列内的占比在49.9%至50.1%波动，基本符合信源条件。

## 稀疏矩阵

golang没有提供现成的矩阵运算包，所以需要自行开发。又考虑到校验矩阵比较稀疏，所以在不调用底层CPU指令集的情况下，将其作为稀疏矩阵进行存储和计算将能够极大地减小运算开支，节约时间。

综上，开发了matrix包，以方便后续开发。该包支持稀疏矩阵的建立，行遍历，列遍历以及矩阵乘法。

## LDPC编码器

LDPC编码器位于coder模块的encoder文件中。创建编码器时，需要传入校验矩阵H。调用Encode函数，传入特定长度的01序列，依据实验要求里所给的LDPC码编码过程，递推地得到所需结果。

## 调制解调器

调制解调器主要用于将01比特序列依据BPSK调制规则进行调制，再将解码器解码完成的浮点序列依据BPSK的规则进行解调处理。

调制部分较为简单，将比特0映射为1，比特1映射为-1即可。

解调部分将输入浮点数序列中的负数输入映射为-1，其余均映射为0即可。

## 信道

信道位于channel模块中，db与线性值的互换也由该包以包函数的形式提供。

使用BPSK调制的一大好处是，每个信号的功率都是1，在加噪过程中可以免去测量的部分。加噪具体使用math包中的NorRand函数，生成符合标准正态分布的随机变量，再将其乘上噪声的标准差，最后将结果加到相对应的BPSK序列上即可。

## LDPC解码器

解码器位于coder模块的decoder文件中。LDPC码的解码过程是一个节点迭代更新的过程，所以定义了Node类以实现节点更新的过程。又由于共有四种解码算法，所以需要一个泛型接口以减少工作量。综上，又定义了Recorder接口以解决这个问题，并实现了四个用于校验节点的Recorder和一个用于值节点的Recorder。此外，在进行节点更新时若使用值节点更新校验节点的方法，将会出现竞争等问题，若使用锁，则会导致性能出现明显损失。因此，采用校验节点从值节点读取数值的方式进行更新。

# 仿真结果及分析

## NMS算法的α值选取

在Eb/N0为1db的信道条件下，从0到1.0测试不同的α值，结果如图Fig5所示。从图中可以看到，α为0.7时，性能最优。

A close up of a map

Description automatically generated

Fig5. NMS算法不同α的纠错性能对比(Eb/N0=1db)

## OMS算法的β值选取

在Eb/N0为1db的信道条件下，从0到1.0测试不同的β值，结果如图Fig6所示。从图中可以看到，β为0.5时，性能最优。

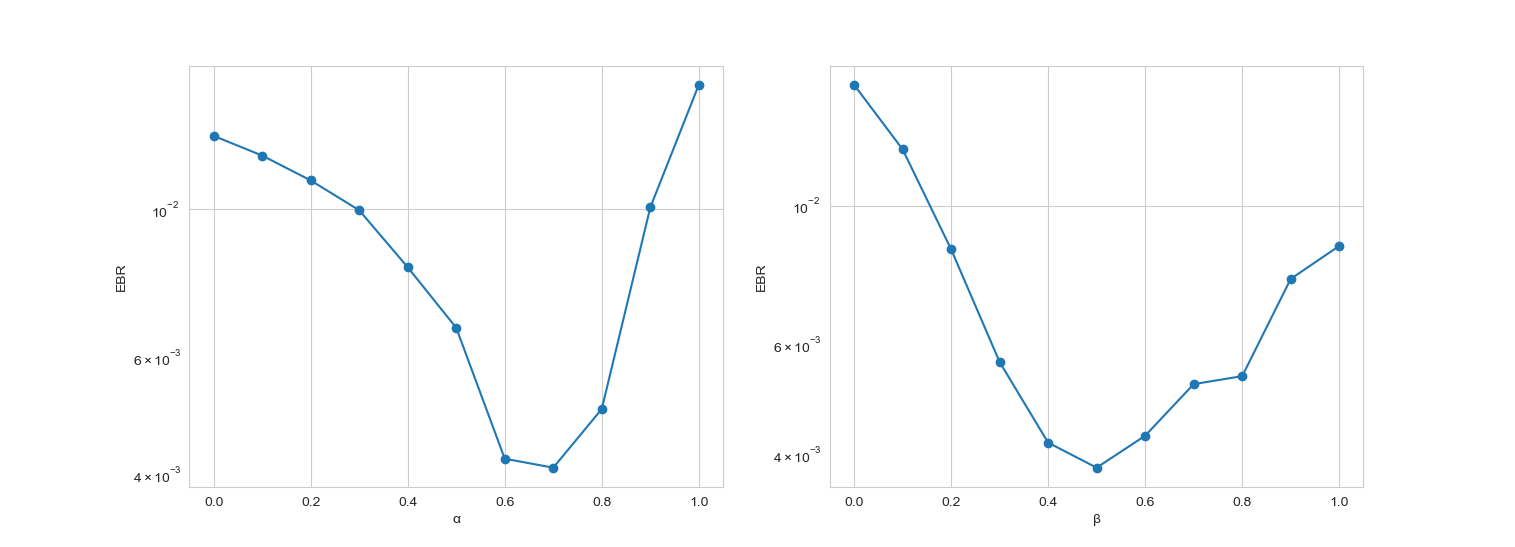


Fig6. OMS算法不同β的纠错性能对比(Eb/N0=1db)

## 纠错性能分析

在[-1db, 2db]的信道条件下，运行四种解码算法，得到的结果如图Fig7所示。从图中可以看到，随着信道条件逐渐变好，四种算法的误码率也在逐渐降低。当Eb/N0大于1db后，SP算法，NMS算法和OMS算法的误码率更是显著下降。在2db的条件下，OMS的EBR甚至优于SP，这主要由于这两种译码算法均已基本不再出错，在这样的情况下，EBR的值更多是与测试使用的数据有关，与两者本身的纠错能力已经关系不大了。

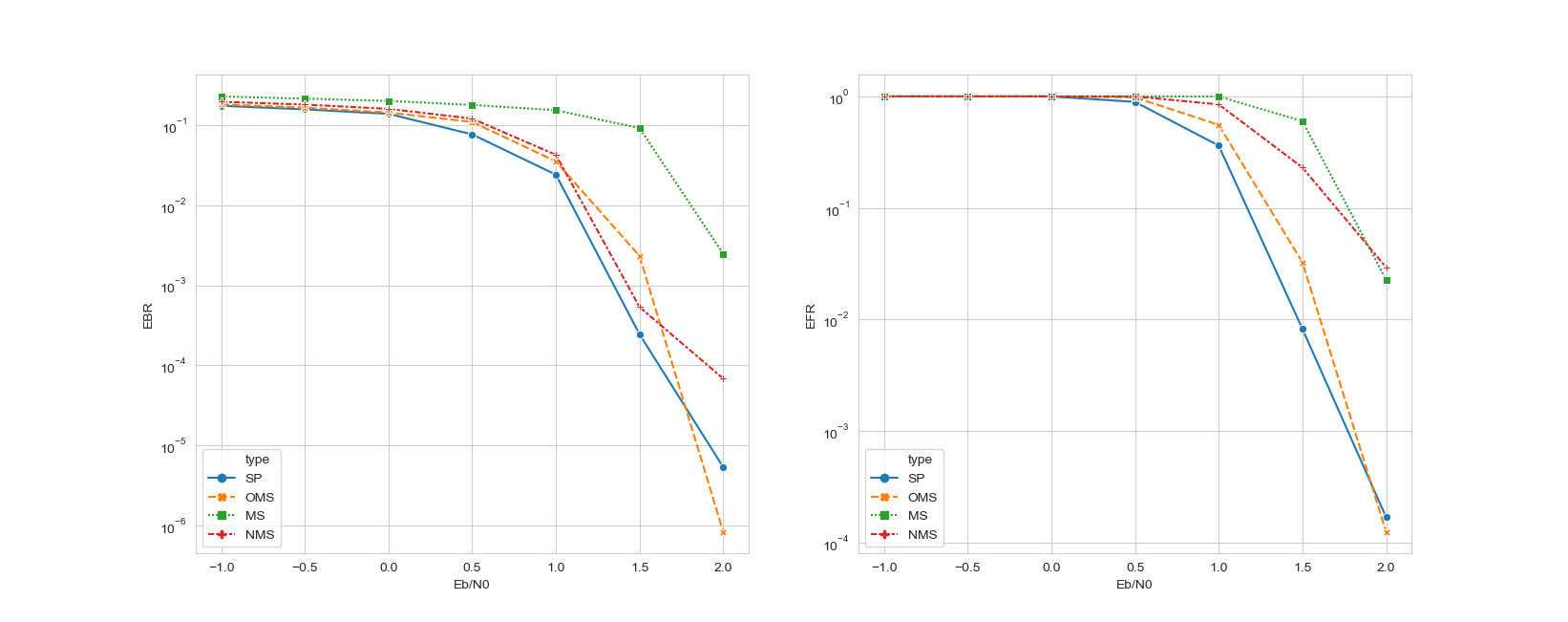


Fig7.四种算法在不同信道条件下的EBR对比

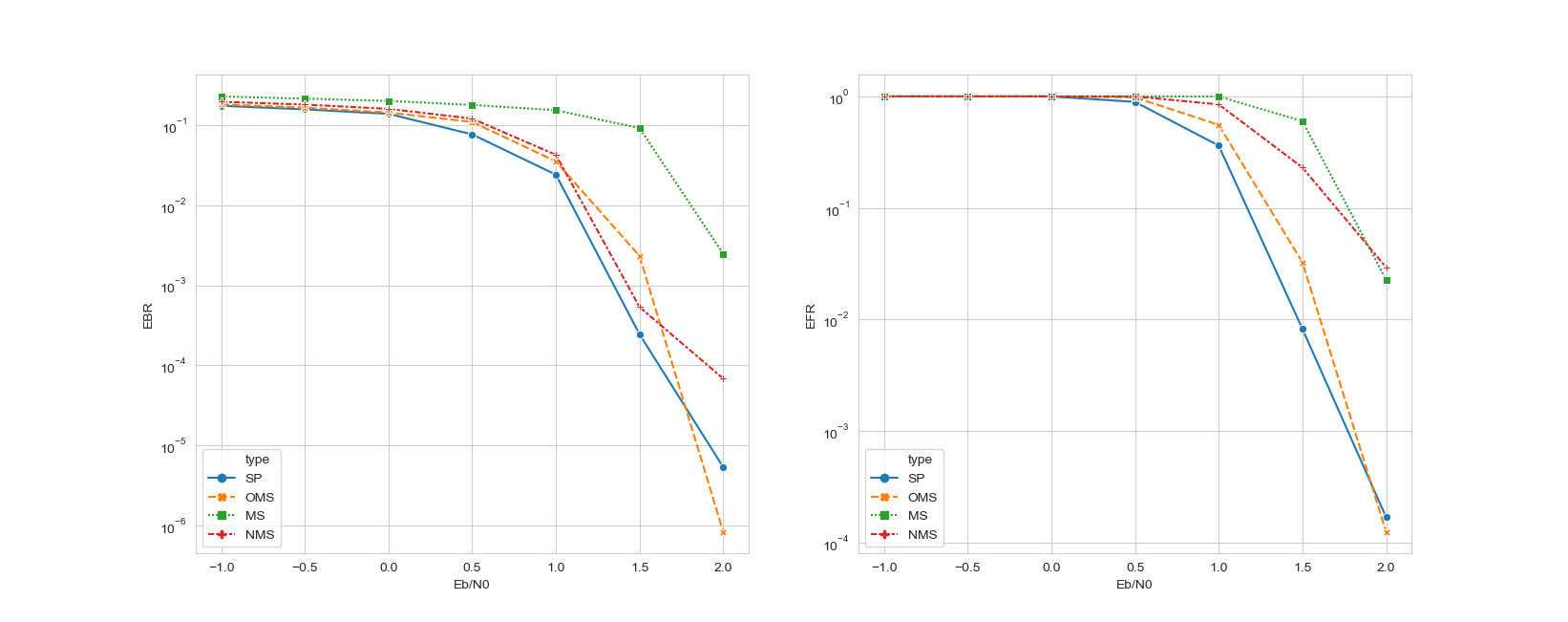


Fig8.四种算法在不同信道条件下的EFR对比

对比来看，MS算法的纠错性能最差，SP算法的纠错性能稳定最优，OMS和NMS在适当的参数选取下纠错性能也十分优秀。其中，OMS算法更是与SP算法相差无几。考虑到SP算法需要计算较为复杂的三角函数，所以在实际的生产中，选用计算复杂度较低，而精度又没有太多损失的OMS算法作为解码算法也不失为一个不错的选择。

## 运行性能分析

使用golang编写的主要目的是为了节约运行时间，实际上，使用golang编写的编解码程序，在型号为i5-8400F的CPU上，运行所有的测试，共处理了53770个数据帧，用时2652887ms约为44.2分钟，比动辄几小时的Matlab略优。

# 项目链接

该大作业的代码与结果均已放到Github上，项目链接为https://github.com/richsoap/Homeworks/tree/master/information/LDPC