**编码引论第三次大作业**

**量化编码、熵编码与联试**

**实验报告**

姓 名：张鹤龄

班 级：无04

学 号：2020010735

完成时间：2022/12/10

目录

[一、量化编码 2](#_Toc121586890)

[均匀量化器： 3](#_Toc121586891)

[4个不同步长的均匀量化器： 3](#_Toc121586892)

[JPEG/H.261量化器： 3](#_Toc121586893)

[非均匀量化器： 3](#_Toc121586894)

[二、熵编码 3](#_Toc121586895)

[使用提供的码表进行变长编码 3](#_Toc121586896)

[对Huffman编码，逃逸码的理解与设计 4](#_Toc121586897)

[Huffman编码的基本原理 4](#_Toc121586898)

[逃逸码理解和设计 5](#_Toc121586899)

[编写代码思路： 5](#_Toc121586900)

[逃逸码对性能的影响测试 6](#_Toc121586901)

[单符号Huffman 6](#_Toc121586902)

[双符号Huffman 7](#_Toc121586903)

[生成新的单符号和双符号的码本 8](#_Toc121586904)

[单符号 8](#_Toc121586905)

[双符号 11](#_Toc121586906)

[验证自己Huffman编码的正确性 14](#_Toc121586907)

[三、联试和参数选择 14](#_Toc121586908)

[实验模型与流程 14](#_Toc121586909)

[信道模型 14](#_Toc121586910)

[信道编码 15](#_Toc121586911)

[信源编码 15](#_Toc121586912)

[参数选择 16](#_Toc121586913)

[尽量降低要传输的符号数 17](#_Toc121586914)

[尽量提升抗噪声性能 17](#_Toc121586915)

[尽量提升重建图像质量 17](#_Toc121586916)

# 一、量化编码

## **均匀量化器**：

### 4个不同步长的均匀量化器：

固定blockOption参数为1（分块大小为）

压缩比公式如下：

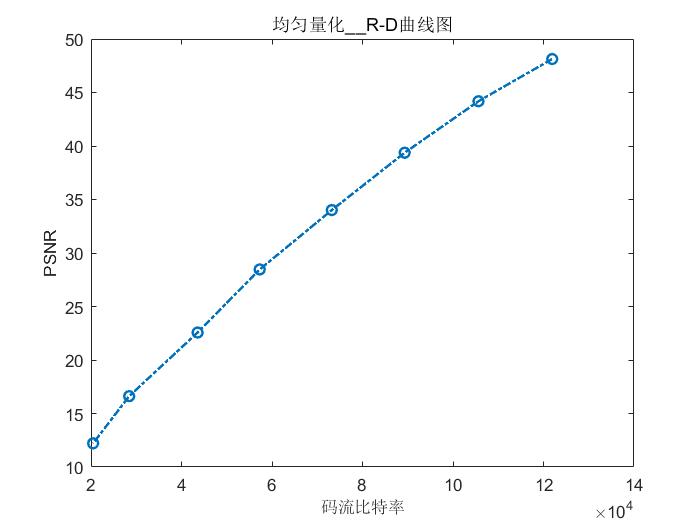
数据统计如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **量化步长** | **输入图像比特数** | **量化后比特数** | **压缩比** | **PSNR** |
| 1 | 131072 | 121925 | 0.930214 | 48.130804 |
| 2 | 131072 | 105634 | 0.805923 | 44.185893 |
| 4 | 131072 | 89326 | 0.681503 | 39.372564 |
| 8 | 131072 | 73220 | 0.558624 | 34.011764 |
| 16 | 131072 | 57283 | 0.437035 | 28.465559 |
| 32 | 131072 | 43556 | 0.332306 | 22.568138 |
| 64 | 131072 | 28440 | 0.216980 | 16.621698 |
| 128 | 131072 | 20466 | 0.156143 | 12.210713 |

处理后图片的效果展示如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 均匀量化步长1 | 均匀量化步长2 | 均匀量化步长4 | 均匀量化步长8 |
|  |  |  |  |
| 均匀量化步长16 | 均匀量化步长32 | 均匀量化步长64 | 均匀量化步长128 |
|  |  |  |  |

将其比特率和PSNR绘制在Rate-Distortion(R-D)曲线中：



可见量化步长越大，量化后图像的质量越低，压缩率越高。

### JPEG/H.261量化器：

一张图片中，人眼对不同频率分量的感知是不同的，对低频分量的感知比对高频分量的感知强的多。因此，如果我们想保证较高的压缩率，一个直观的思路就是丢弃高频分量。

JPEG编码利用了这一点。在JPEG编码中，我们先将待处理图像分块，再对其进行离散余弦变换，分析出从低频到高频的各分量值，将高频分量舍弃。舍弃高频分量的程度决定了压缩比。

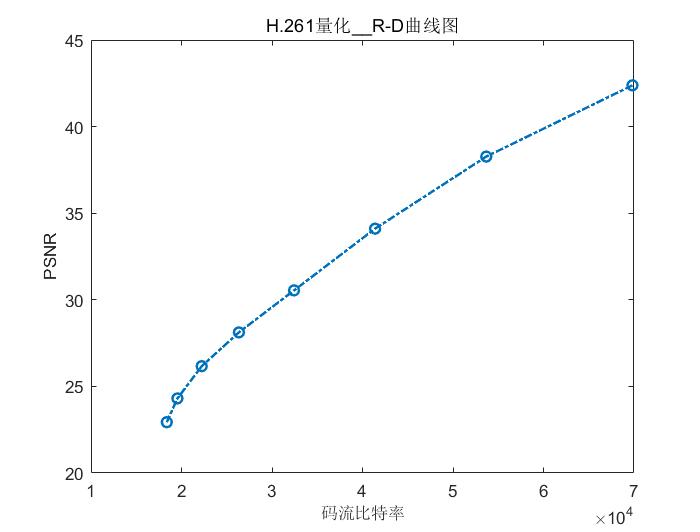
用H.261量化，数据统计如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **量化参数QP** | **输入图像比特数** | **量化后比特数** | **压缩比** | **PSNR** |
| 1 | 131072 | 69833 | 0.532784 | 42.382440 |
| 2 | 131072 | 53675 | 0.409508 | 38.268714 |
| 4 | 131072 | 41392 | 0.315796 | 34.107686 |
| 8 | 131072 | 32434 | 0.247452 | 30.543202 |
| 16 | 131072 | 26340 | 0.200958 | 28.118693 |
| 32 | 131072 | 22233 | 0.169624 | 26.167948 |
| 64 | 131072 | 19546 | 0.149124 | 24.305434 |
| 100 | 131072 | 18377 | 0.140205 | 22.934823 |

处理后图片的效果展示如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **H.261量化参数1** | **H.261量化参数2** | **H.261量化参数4** | **H.261量化参数8** |
|  |  |  |  |
| H.261量化参数16 | H.261量化参数32 | H.261量化参数64 | H.261量化参数100 |
|  |  |  |  |

将其比特率和PSNR绘制在Rate-Distortion(R-D)曲线中：



无论是从图片的直观感受，还是PSNR或者量化后比特数的情况来看，H.261量化的效果都是更优的。

### 非均匀量化器：

通过资料的查询，这里采用Lloyd Max算法，来得到非均匀量化的分层界值。这个算法的核心思想在于，依据一幅图像具体的灰度值分布的概率密度函数，按总的量化误差最小的原则来进行量化：具体做法是对图像中像素灰度值频繁岀现的灰度值范围,量化间隔取小些，区分更加精细，而对那些像素灰度值极少出现的范围,则量化间隔取大些，区分粗糙一些。（[参考博客](https://www.cnblogs.com/pphy1884/p/4594084.html)）

最佳量化器的条件可以写成：

即为量化重建值，而为决策边界。

算法具体流程如下：

1. 计算像素值的概率分布。（采用加和再归一化）
2. 初始化所有决策边界和量化重构值。
3. 迭代开始，每一轮开始，都用上面的两个公式依次更新所有决策边界和重建水平。
4. 然后计算MSE：，因为本题像素值都离散值，所以实际MSE计算如下：。其中j表示循环的轮数。
5. 如果，其中是设定的阈值，就退出循环。否则回到3。

这部分代码封装为Lloyd\_Max.m函数。

固定blockOption为2，对不同的分层数进行测试：

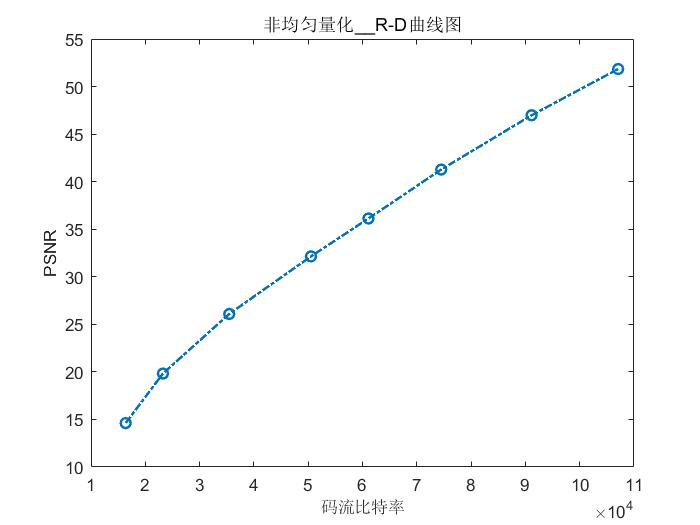
数据统计如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **分层参数QP** | **输入图像比特数** | **量化后比特数** | **压缩比** | **PSNR** |
| 1 | 131072 | 16384 | 0.125000 | 14.617876 |
| 2 | 131072 | 23251 | 0.177391 | 19.825877 |
| 4 | 131072 | 35451 | 0.270470 | 26.089822 |
| 8 | 131072 | 50518 | 0.385422 | 32.144286 |
| 16 | 131072 | 61095 | 0.466118 | 36.127191 |
| 32 | 131072 | 74497 | 0.568367 | 41.273627 |
| 64 | 131072 | 91136 | 0.695312 | 46.981431 |
| 128 | 131072 | 107082 | 0.816971 | 51.847039 |

处理后图片的效果展示如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **分层1** | **分层2** | **分层4** | **分层8** |
|  |  |  |  |
| **分层16** | **分层32** | **分层64** | **分层128** |
|  |  |  |  |

将其比特率和PSNR绘制在Rate-Distortion(R-D)曲线中：



与JPEG对比，Lloyd算法在同等PSNR下有较高的比特率，在同等比特率下PSNR较低。

Lloyd\_max算法根据图像的像素灰度分布决定量化区间，可以使图像的灰度PSNR在给定的量化阶数下局部最优。但是，由于其在空域对每个像素单独进行压缩，没有考虑到人眼对图片的反应特性，因此压缩效果并不尽如人意。另外，由于其仅对灰度值进行量化，要想提高压缩比，就必须降低量化阶数，从而增大图片的失真。（严格来讲这些并不是lloyd\_max算法的缺点，而是空域量化的缺点。我们完全可以用lloyd\_max算法决定JPEG中频域分量的量化阶梯，从而可能提升JPEG的性能）

# 二、熵编码

## 使用提供的码表**进行变长编码**

可调节参数：

vlcRadio: 单符号变长编码/双符号变长编码

blockOption：改变VLC中的Slice高度

注意因为这里进行的都是熵编码，且没有经过信道，因此编解码过程是不可能出现错误的。因此这里只选择一个blockOption，量化的选项设置为0（均匀量化），量化步长为1，使用单符号和双符号，进行编码，解码，来作为示范。我们使用lena图片。

单符号：

blockOption:1 slice\_height:4

uniform quantization quant\_step=1

one symbol coding

input image bit:131072

Processed image bit:121925

Quantization success!

One symbol encode: slice\_height:4

One symbol encode: Encode success! Total bits:191452

One symbol decode: slice\_idx:0

One symbol decode: slice\_idx:1

One symbol decode: slice\_idx:2

One symbol decode: slice\_idx:3

One symbol decode: slice\_idx:4

One symbol decode: slice\_idx:5

One symbol decode: slice\_idx:6

One symbol decode: slice\_idx:7

One symbol decode: slice\_idx:8

One symbol decode: slice\_idx:9

One symbol decode: slice\_idx:10

One symbol decode: slice\_idx:11

One symbol decode: slice\_idx:12

One symbol decode: slice\_idx:13

One symbol decode: slice\_idx:14

One symbol decode: slice\_idx:15

One symbol decode: slice\_idx:16

One symbol decode: slice\_idx:17

One symbol decode: slice\_idx:18

One symbol decode: slice\_idx:19

One symbol decode: slice\_idx:20

One symbol decode: slice\_idx:21

One symbol decode: slice\_idx:22

One symbol decode: slice\_idx:23

One symbol decode: slice\_idx:24

One symbol decode: slice\_idx:25

One symbol decode: slice\_idx:26

One symbol decode: slice\_idx:27

One symbol decode: slice\_idx:28

One symbol decode: slice\_idx:29

One symbol decode: slice\_idx:30

One symbol decode: slice\_idx:31

One symbol decode: All correct!!! Total bits:191452 PSNR:48.130804

save the processed image correctly.

save the reconstructed image correctly.

双符号：

blockOption:1 slice\_height:4

uniform quantization quant\_step=1

two symbol coding

input image bit:131072

Processed image bit:121925

Quantization success!

Two symbol encode: Slice\_height:4

Two symbol encode: Encode success! Total bits:164875

Two symbol decode: slice\_idx:0

Two symbol decode: slice\_idx:1

Two symbol decode: slice\_idx:2

Two symbol decode: slice\_idx:3

Two symbol decode: slice\_idx:4

Two symbol decode: slice\_idx:5

Two symbol decode: slice\_idx:6

Two symbol decode: slice\_idx:7

Two symbol decode: slice\_idx:8

Two symbol decode: slice\_idx:9

Two symbol decode: slice\_idx:10

Two symbol decode: slice\_idx:11

Two symbol decode: slice\_idx:12

Two symbol decode: slice\_idx:13

Two symbol decode: slice\_idx:14

Two symbol decode: slice\_idx:15

Two symbol decode: slice\_idx:16

Two symbol decode: slice\_idx:17

Two symbol decode: slice\_idx:18

Two symbol decode: slice\_idx:19

Two symbol decode: slice\_idx:20

Two symbol decode: slice\_idx:21

Two symbol decode: slice\_idx:22

Two symbol decode: slice\_idx:23

Two symbol decode: slice\_idx:24

Two symbol decode: slice\_idx:25

Two symbol decode: slice\_idx:26

Two symbol decode: slice\_idx:27

Two symbol decode: slice\_idx:28

Two symbol decode: slice\_idx:29

Two symbol decode: slice\_idx:30

Two symbol decode: slice\_idx:31

Two symbol decode: All correct!!!

Total bits:164875

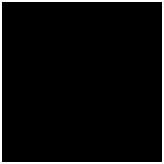
PSNR:48.130804

save the processed image correctly.

save the reconstructed image correctly.

图片：

从左往右分别是量化后图像，编码解码后，以及两个图像之间的difference。可以发现确实没有变化。

## 对Huffman编码，逃逸码的理解与设计

### **Huffman编码的基本原理**

在信源处观察一段时间内发送的信息（或直接拿到全部要发送的信息），统计各种符号出现的频率，将统计分布作为概率分布。之后，

一、取出所有符号中概率最小的两个，将其合并成新的符号，新符号的概率等于被合并的两个符号概率之和。

二、重复上一步，直到仅剩一个符号。

三、按照合并的次序可以得到一株huffman树，每个原始符号都对应一个叶子节点。从根节点开始，将左子树根节点记0，右子树根节点记1，这样依次标记每个节点。

四、从根节点开始，到每个符号对应的叶子节点有唯一路径，按照路径顺序串联节点标记，即得到符号对应的huffman码。

Huffman码是前缀码，即任何码字都不是另一个码字的前缀，因此也是即时码；

Huffman码是变长编码；

Huffman码不损失信息量，因此是熵编码；

在所有可能的唯一可译码中，Huffman码平均码长最短，因此也是最佳码。

### **逃逸码理解和设计**

考虑一个八符号（1，2，3，…，8），服从均匀分布的信源。

对于这个信源进行huffman编码，由完全二叉树节点和自然数一一对应的性质，huffman树是一株完全二叉树。也就是说，进行Huffman编码，相对于直接传输信号的二进制表示，无法获得任何增益；并且huffman编码还需要额外传输码表。

如果仅仅是这样，仍然不需要添加逃逸码，因为总体的符号规模不大。但是考虑以下的信源：

符号集为{0，1，2，3，……，2n}，P(0)=0.5,P(m)=2-n-1。(m≠0)

若n的规模很大，则对这样一个信源进行huffman编码的代价是巨大的。然而，从另一个方面，因为0的概率仍然很大，又不能完全舍弃huffman编码，因此只能采用逃逸码的方案，即如果被编码的符号不是0则“逃逸”，编码结果为逃逸码+符号的二进制表示。这样，既节省了传输码本的大小和译码的时间成本，也充分利用了huffman码的优秀能力。

### **编写代码思路：**

完成Huffman编码的关键点在于构建出Huffman树，根据Huffman树分配每个码字对应的Huffman编码。我对于逃逸码的处理是：设定一个概率的阈值，在统计完符号的概率分布后，将小于这个阈值的符号们统一使用逃逸码进行编码。为了得到标记逃逸码开始的那一串编码，我将逃逸码的符号视作一个新的符号，概率是所有用逃逸码编码的符号的概率之和，然后参与到Huffman编码的过程当中。

首先，先分析单符号的Huffman编码：

1. 获得符号的概率分布：对于单符号而言，我将输入的数组转为行向量，然后调用tabulate函数进行统计。
2. 根据设定的escape\_prob，将概率小于等于escape\_prob的符号们合并成为一个，他们之后将会被用逃逸码编码。在单符号编码的时候，单个像素点的像素取值是0-255，因此我将逃逸码视作一个新的符号的时候，将其标记为256。
3. 然后开始生成Huffman树。具体的过程是，我每次将最小的两个符号合并成一个符号，概率相加。注意，这个符号，注意到像素点的取值都是大于等于0的，因此我这里采用负数来标记。第一个合并的我就标记为-1，以此类推。注意，这里的负数的绝对值刚好可以标记合并的顺序，因此我分层存储每次合并的左右孩子，在解树的时候，如果是负数，我就接着找它的孩子，如果是正数，就说明是我需要的符号。
4. 从Huffman树中解出Huffman编码。依照（3）中所说的过程解树，并且把逃逸码的码字解出来。
5. 使用（4）中得到的码字，对输入的序列进行编码。

其次我们来分析双符号的，双符号的比单符号的在实现上复杂一些，但是基本思想是一样的，只是把相邻的两个符号作为了一个新的符号进行统计。

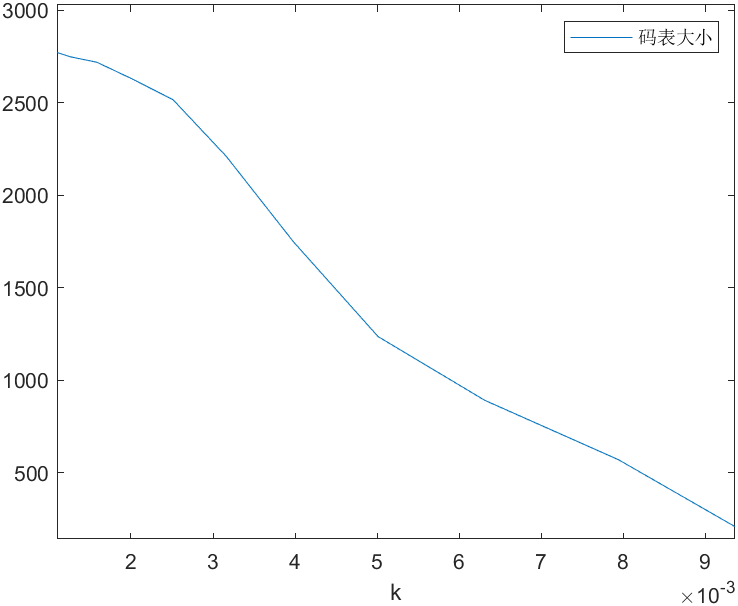
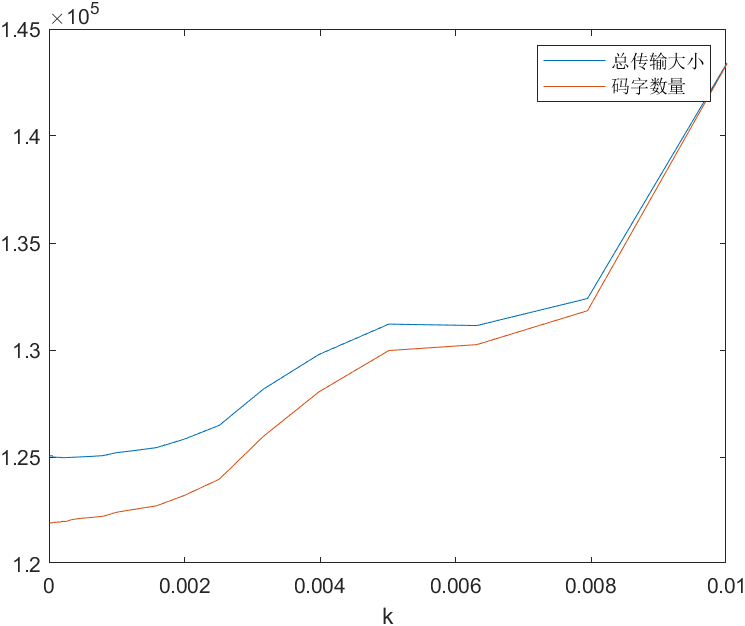
1. 获得符号的概率分布：对于单符号而言，我将输入的数组转为行向量，然后两个两个提取，统计数目。 为了可以复用单符号中的函数，我将统计完的结果排序，之后从上到下用1，2，3……顺序的一一对应。
2. 根据设定的escape\_prob，将概率小于等于escape\_prob的符号们合并成为一个，他们之后将会被用逃逸码编码。在多符号编码的时候，对应到了一串正整数序列1，2……L。因此我将逃逸码视作一个新的符号的时候，将其标记为L+1。
3. 然后开始生成Huffman树。具体过程和单符号中的实现相同。
4. 从Huffman树中解出Huffman编码，同时解出逃逸码码字。和单符号相同。
5. 使用（4）中得到的码字，对输入的序列进行编码。

## **逃逸码对性能的影响测试**

使用lena图片。

### **单符号Huffman**

横坐标为escape\_prob，纵坐标为码表，码字，总传输的比特数大小。



一方面，逃逸码破坏了huffman码的最优性，使得编码后比特率变大；另一方面，逃逸码减少了码表的传输量，减小了总比特率（如上图）。这两种因素相互作用，使得总传输大小总体上上升（如上图），但也存在下降的区间（如下图）

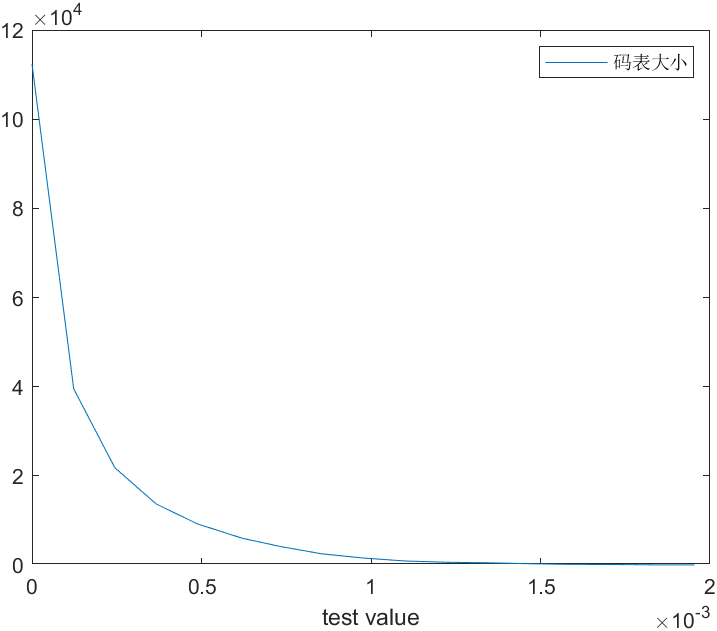
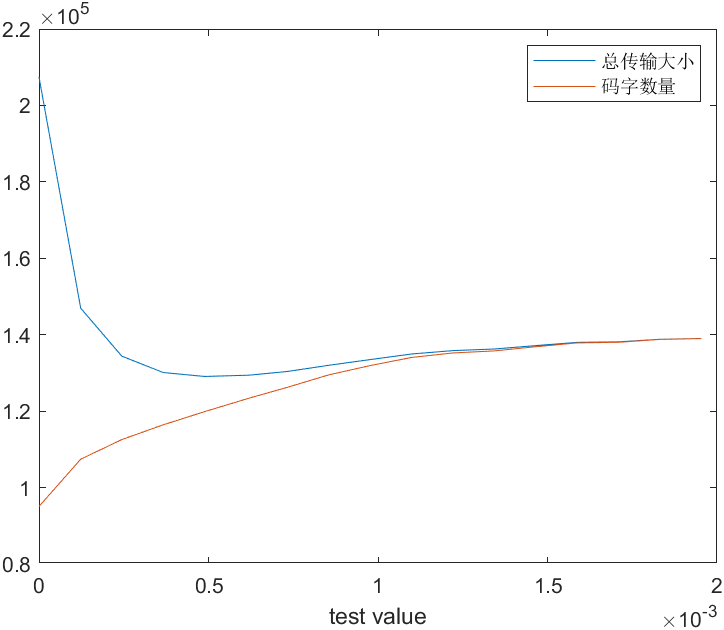


总传输大小的最小值并不是在escape\_prob=0的时候取到，而是在处取到。可以知道，我们采用逃逸码编码确实可以在一定程度上减小总传输的大小（这里我们考虑只传输一次，而不是复用码表）。

### **双符号Huffman**

其中test value即 escape\_prob。

我们可以观察到和单符号中相同的性质：随着escape\_prob的增大，码表的大小在不断的减小。而码字数量在不断地上升。总传输大小在一开始在下降，之后又开始上升。我们同样可以观察到，总传输大小的最小值并不是在escape\_prob=0的时候取到，这也验证了采用逃逸码编码确实可以在一定程度上减小总传输的大小。

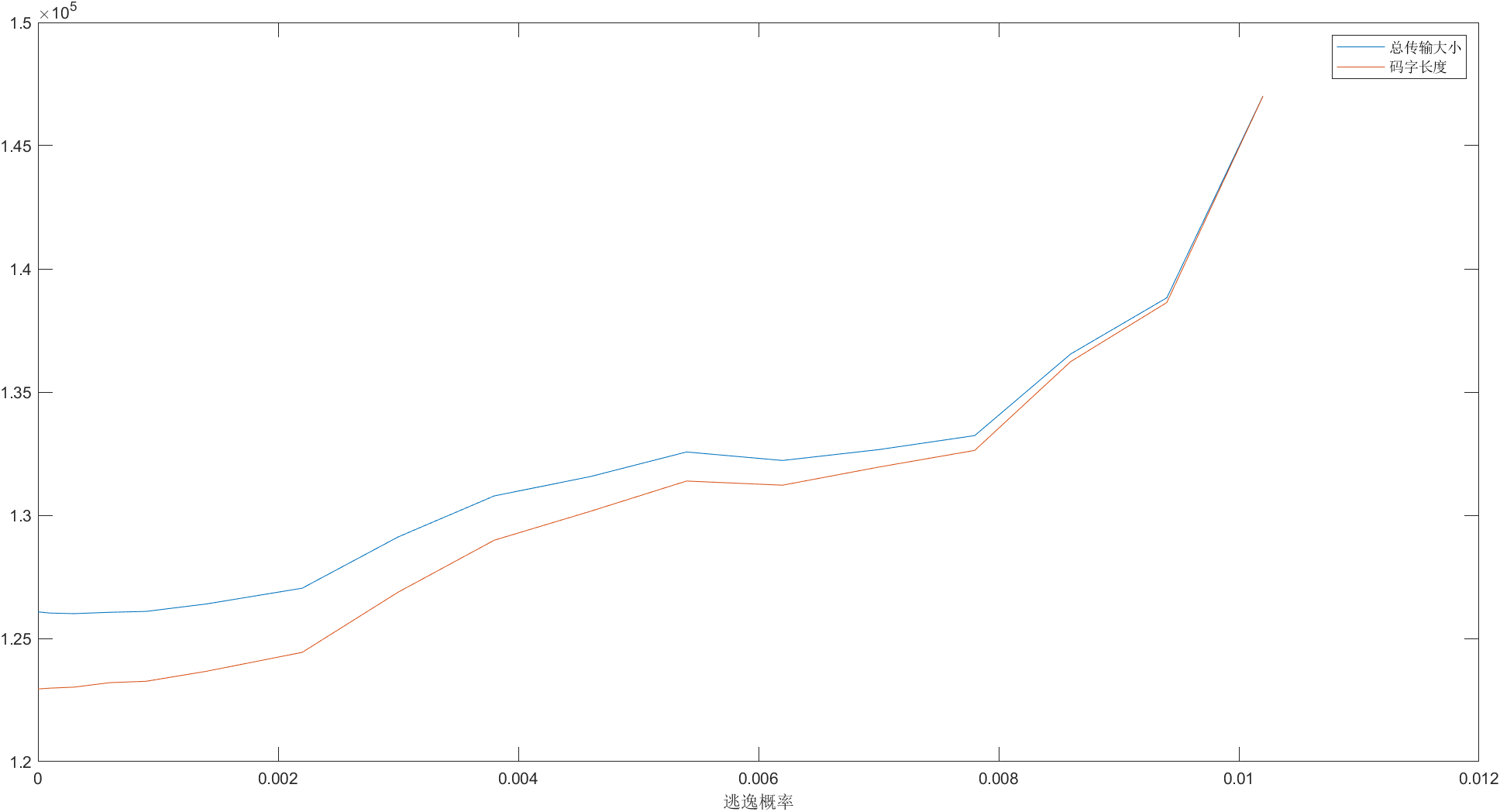


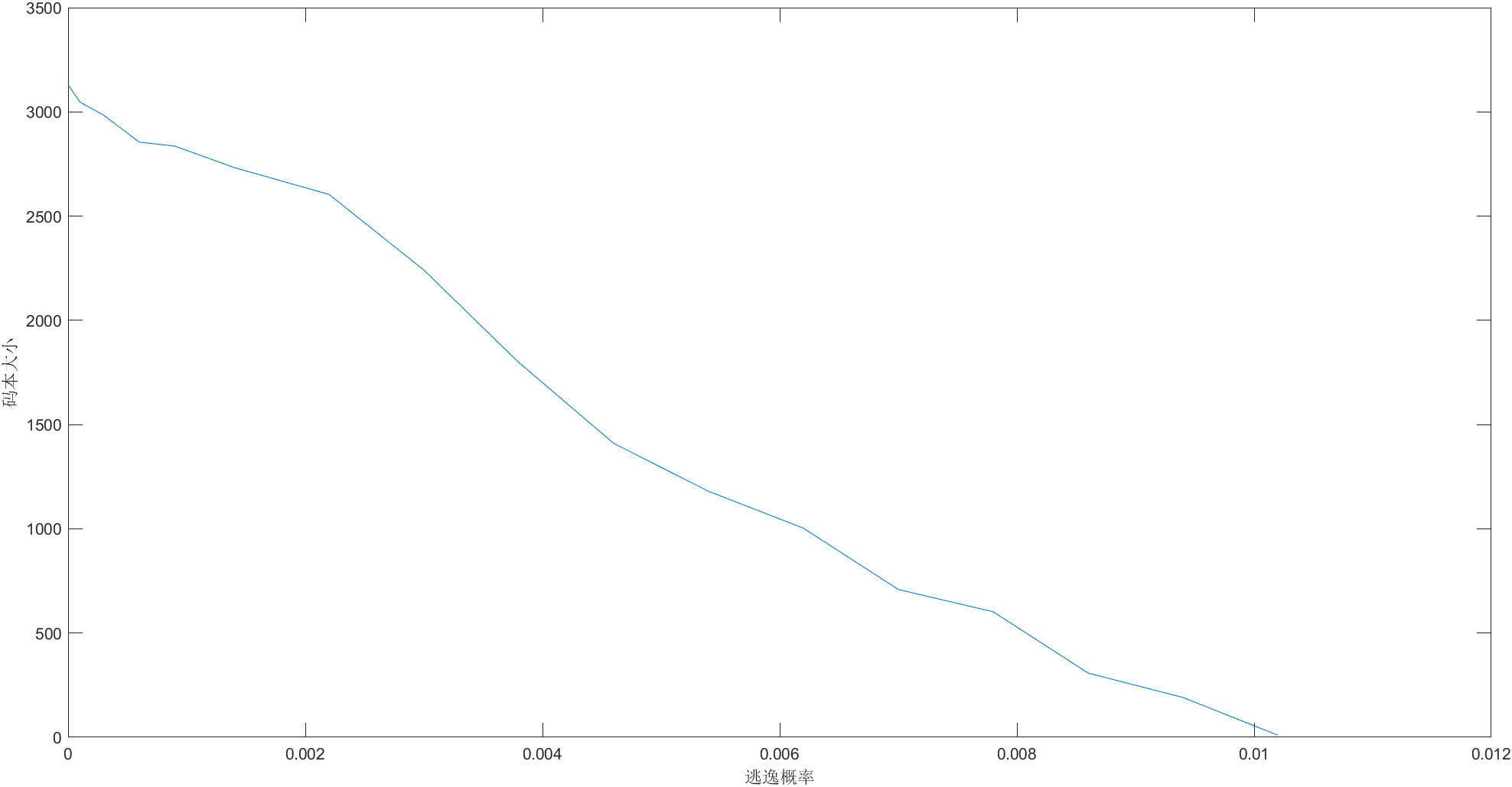
## 生成新的单符号和双符号的码本

### **单符号**

|  |  |  |  |
| --- | --- | --- | --- |
| 逃逸概率 | 总传输大小 | 码表符号数 | 码表长度 |
| 0 | 126080 | 203 | 3131 |
| 1e-4 | 126030 | 199 | 3047 |
| 3e-4 | 126007 | 196 | 2984 |
| 6e-4 | 126062 | 189 | 2855 |
| 9e-4 | 126097 | 188 | 2836 |
| 0.0014 | 126398 | 182 | 2733 |
| 0.0022 | 127039 | 174 | 2604 |
| 0.0030 | 129122 | 151 | 2241 |
| 0.0038 | 130786 | 122 | 1798 |
| 0.0046 | 131570 | 98 | 1410 |
| 0.0054 | 132568 | 80 | 1180 |
| 0.0062 | 132223 | 69 | 1003 |
| 0.0070 | 132663 | 50 | 709 |
| 0.0078 | 133233 | 43 | 602 |
| 0.0086 | 136547 | 23 | 308 |
| 0.0094 | 138826 | 15 | 191 |
| 0.0102 | 147010 | 1 | 10 |

**总传输大小，码字长度，码表长度绘图如下：**





可以发现，这和我们前面绘制的图像几乎相同。我们可以选择不适用逃逸码以达到最小比特率，但考虑到译码复杂度，也可以在0到0.002这个平坦区间选择合适的逃逸概率，以达到比特率和译码性能的折中。

对应码表如下：（最后一行是逃逸码）

156 001101

155 001000

130 000111

129 000100

153 000101

157 000010

131 000011

127 000000

144 000001

152 1011110

48 1011111

126 1011100

128 1011101

143 1011010

154 1011011

148 1011000

146 1011001

101 1010110

141 1010111

99 1010100

151 1010101

161 1010010

158 1010011

125 1010000

142 1010001

147 1001110

53 1001111

140 1001100

145 1001101

133 1001010

50 1001011

160 1001000

102 1001001

49 1000110

103 1000111

52 1000100

150 1000101

139 1000010

45 1000011

51 1000000

54 1000001

135 0111110

159 0111111

136 0111100

138 0111101

132 0111010

149 0111011

47 0111000

100 0111001

104 0110110

46 0110111

124 0110100

162 0110101

109 0110010

122 0110011

137 0110000

98 0110001

123 0101110

119 0101111

134 0101100

106 0101101

165 0101010

121 0101011

117 0100110

163 0100111

55 0101000

97 0101001

57 0100100

167 0100101

96 0100010

105 0100011

95 0100000

107 0100001

118 0011110

120 0011111

44 0011100

173 0011101

94 0011000

114 0011001

56 0010110

164 0010111

58 0010100

43 0010101

175 0010010

166 0010011

93 0001100

116 0001101

11

如果考虑到单符号码表都不大，那我们可以选择逃逸概率3e-4来达到最短的总传输长度，对应码表如下：

156 000001

130 1111010

155 1111011

129 1110110

153 1110111

131 1110100

144 1110010

157 1110011

127 1110000

48 1101111

128 1101010

152 1101011

126 1101101

143 1101000

154 1101001

146 1100100

141 1100000

148 1100001

101 1100011

99 1011110

151 1011111

158 1011010

142 1011000

161 1011001

53 1010110

125 1010111

147 1010100

140 1010000

145 1010001

50 1001110

102 1001100

133 1001101

160 1001010

49 1001000

103 1001001

52 1000100

150 1000101

45 1000111

139 1000010

51 1000000

54 1000001

135 0111110

159 0111111

138 0111101

136 0111011

149 0111001

132 0110110

47 0110100

100 0110101

46 0110010

104 0101101

124 0101010

162 0101011

109 0100110

122 0100111

98 0101001

137 0100100

119 0100010

123 0100000

106 0011111

121 0011100

134 0011101

165 0011010

117 0010110

163 0010111

55 0011000

97 0011001

167 0010100

105 0010010

57 0010011

107 0010000

96 0010001

95 0001110

118 0001010

120 0001011

173 0000110

44 0000100

94 0000000

114 0000001

164 11111110

43 11111100

56 11111101

58 11111010

166 11111001

116 11110010

175 11110011

93 11110001

60 11101010

174 11101011

111 11100010

108 11100011

177 11011100

59 11011000

171 11011001

113 11001110

115 11001111

213 11001100

178 11001010

172 11001011

176 11000100

61 11000101

168 10111010

90 10110110

110 10110111

66 10111000

89 10111001

203 10101010

208 10101011

112 10100100

169 10100101

87 10100110

92 10100111

207 10011110

86 10011111

42 10010110

194 10010111

88 10000110

206 10000111

62 10001101

210 01111000

91 01111001

179 01110101

76 01110000

201 01110001

170 01101110

183 01101111

80 01100110

212 01100111

195 01100010

67 01100011

77 01100000

73 01100001

72 01011110

74 01011111

79 01011100

197 01011000

69 01011001

209 01010000

40 01010001

84 01001010

202 01001011

211 01000110

78 01000111

180 01000010

191 01000011

204 00111100

81 00110110

196 00110111

41 00101010

65 00101011

83 00011111

182 00011000

184 00011001

70 00011010

85 00011011

192 00001110

199 00001111

75 00010000

188 00010001

63 00010010

71 00010011

82 00001010

181 00001011

193 111111110

64 111111111

198 111110111

189 111110000

205 111110001

200 111100000

187 111100001

215 110111010

39 110111011

186 110011010

190 110011011

214 101110110

68 100011001

185 011101001

218 010111010

216 010111011

217 001111010

38 000111100

36 000111101

37 1111101100

219 1011101111

35 1000110000

221 1000110001

222 0111010000

34 0111010001

220 0011110110

33 10111011100

225 00111101110

223 00111101111

226 111110110100

32 111110110101

31 101110111010

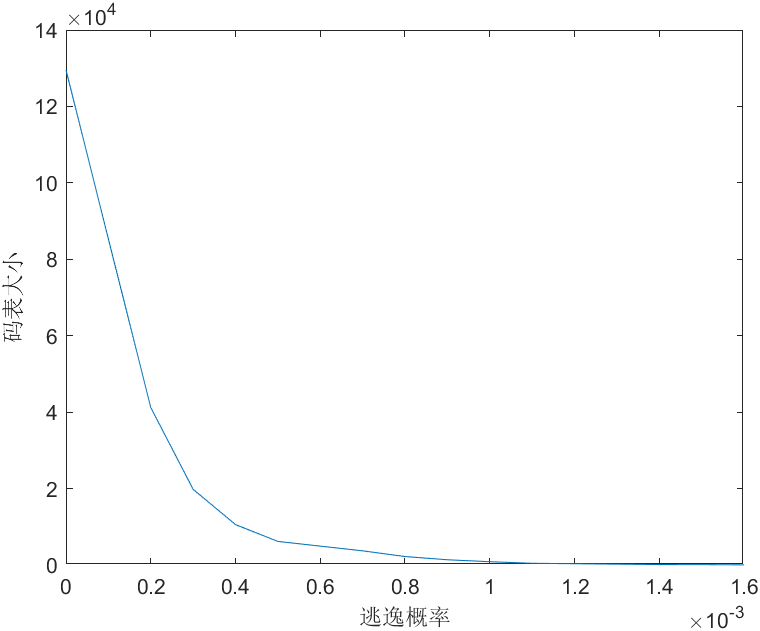
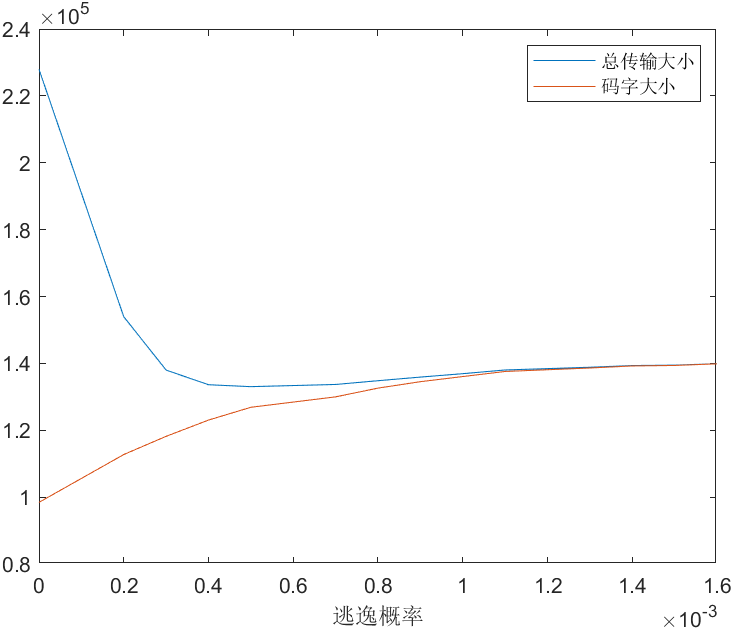
224 101110111011

11111011011

### **双符号**

|  |  |  |  |
| --- | --- | --- | --- |
| 逃逸概率 | 总传输大小 | 码表符号数 | 码表长度 |
| 0 | 227948 | 4779 | 129508 |
| 2e-4 | 153943 | 1558 | 41207 |
| 3e-4 | 137989 | 776 | 19805 |
| 4e-4 | 133615 | 426 | 10573 |
| 5e-4 | 133040 | 255 | 6170 |
| 7e-4 | 133694 | 157 | 3708 |
| 8e-4 | 134814 | 96 | 2212 |
| 9e-4 | 135899 | 61 | 1372 |
| 1e-3 | 136916 | 39 | 858 |
| 0.0011 | 138027 | 21 | 447 |
| 0.0013 | 138839 | 11 | 226 |
| 0.0014 | 139351 | 6 | 118 |
| 0.0015 | 139473 | 5 | 97 |
| 0.0016 | 139905 | 2 | 37 |

总传输大小，码字长度，码表长度绘图如下：



这和我们前面绘制的图像几乎相同。可以看到总传输大小的最小值不是在0处取到，大约是在逃逸概率为5e-4处取得，但是注意此时码表大小也不太小，因此在实际中应该不会选择这个点。

为了达到编码解码代价和编码长度的一个权衡，考虑码表大小的因素之后，我们可以选择逃逸概率为8e-4。

权衡点（8e-4）对应码表如下：

132 134 00111100

129 132 00111101

160 162 01001010

126 126 01001011

130 131 01001000

127 130 01001001

131 128 01001110

155 155 01001111

154 152 01001100

102 103 01001101

131 131 01000010

152 153 01000011

103 102 01000000

48 49 01000001

160 159 01000110

160 158 01000111

128 126 01000100

152 150 01000101

174 173 01011010

101 103 01011011

151 149 01011000

141 141 01011001

142 144 01011110

149 148 01011111

145 146 01011100

146 144 01011101

139 138 01010010

150 147 01010011

144 145 01010000

54 53 01010001

48 48 01010110

98 97 01010111

52 50 01010100

50 50 01010101

150 150 01100010

127 125 01100011

155 157 01100110

156 158 01100111

125 127 01100100

160 160 01100101

155 153 01110010

129 131 01110011

156 154 01110000

127 124 01110001

155 156 01110110

154 154 01110111

153 150 01110100

103 103 01110101

147 148 01101010

143 145 01101011

142 142 01101000

143 141 01101001

54 55 01101110

138 140 01101111

101 102 01101100

177 179 01101101

138 141 01111000

129 129 01111001

153 153 0001000

125 125 0001001

154 155 01111110

128 130 01111111

158 156 01111100

129 128 01111101

158 159 0000010

162 161 0000011

130 128 0000000

156 153 0000001

161 161 0000110

158 158 0000111

52 54 0000100

146 146 0000101

98 101 01111010

145 144 01111011

101 99 0001010

157 156 0001011

127 127 0001110

157 159 0001111

157 157 0001100

161 160 0001101

148 147 0010010

143 144 0010011

154 153 0010000

101 100 0010001

146 145 0010100

130 130 0010101

152 151 0011000

159 158 0011001

154 156 0010110

143 140 0010111

157 155 0011010

131 130 0011011

156 155 0011100

127 128 0011101

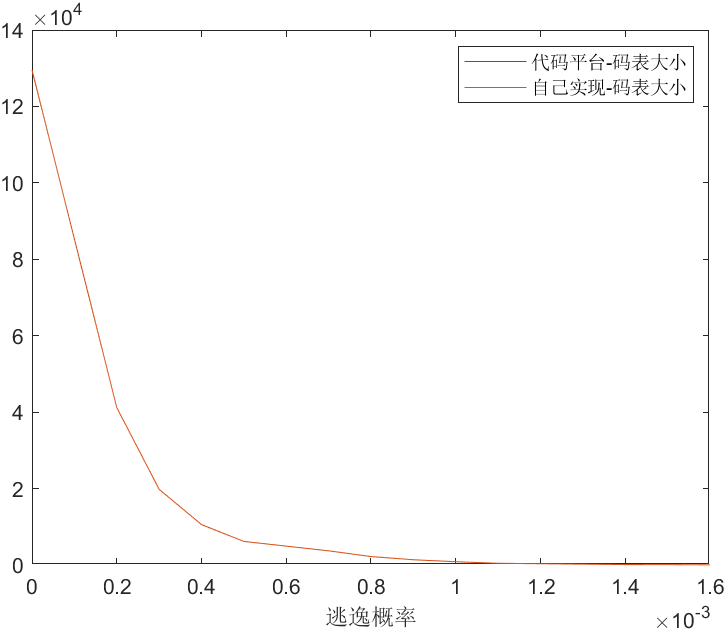
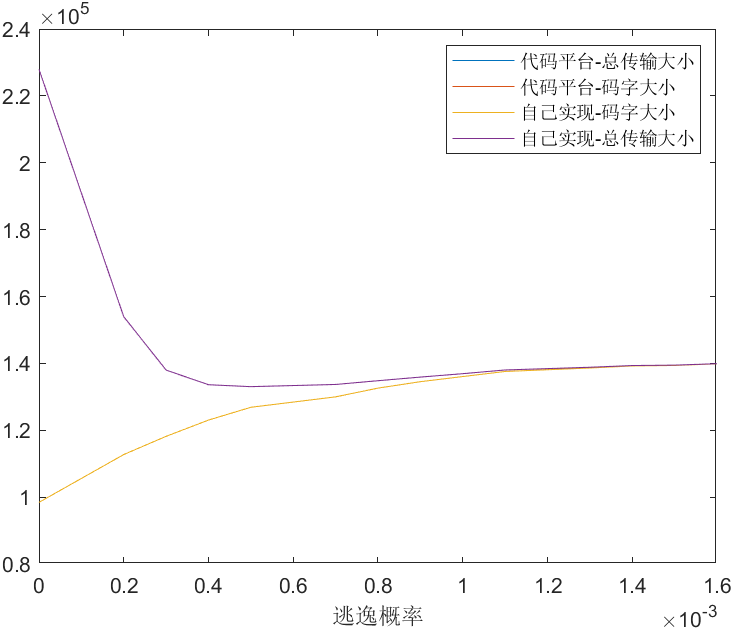
156 156 0011111

158 157 0110000

1

## 验证自己Huffman编码的正确性

下面只验证双符号的，双符号复用了单符号的代码，因此应该正确性相同。注意这里要考虑slice的影响。我使用的blockOption是1，slice\_height=4，因此被分为了32个slice，每个slice的起始编码有32bit，需要在自己实现的统计中加入这些bit。



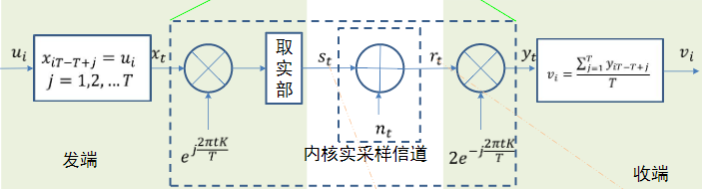
可以发现代码平台上的结果和自己实现的结果完全重合，因此可以验证为正确。

# 三、联试和参数选择

## 实验模型与流程

### 信道模型

本次实验采用波形信道模型，匹配波形为方波，载波频率为，采样频率为。示意图如下：



从发送电平ui到接收电平vi的过程可以等效为电平信道，现计算其噪声功率。在内核信道噪声功率为时，利用我们在实验二中得到的收端计算公式，

设置K=3，T=21，这样上式右侧的第二项将被消除，上式化为：

其中噪声功率在实验二中已经计算过，

将K=3，T=21，代入上式，得到

上式代表噪声功率的总和。由于发送的电平位于单位圆上，因此信噪比可由下式计算：

在实部和虚部两路正交波形上，噪声的功率分配应该是平均的，因此每个方向上的噪声能量为

至此我们得到了电平信道的噪声能量，这在后续软判决时将会用到。

### 信道编码

信道编码采用viterbi编译码模块。如我们在实验一中讨论的，若要采用软判决，则viterbi编码的效率×电平映射bit数应该等于1。因此，在本次实验中，**信道编码参数k**既代表viterbi编码效率的倒数，又代表比特-电平映射数。

Viterbi软判决采用在实验一中给出的软判决公式，到每个符号电平的对数似然值为

其中已经计算出。

### 信源编码

信源编码分为量化和熵编码两部分。我们考虑的方案主要有

量化编码部分：

（1）lloyd\_max量化编码算法

（2）JPEG静止图像量化编码

熵编码部分：

（1）Huffman编码

*值得注意的是，熵编码之前几乎必须要进行量化编码。因为若非如此，待熵编码的信息码字将几乎均匀分布，不仅没有发挥熵编码的长处，反而增加了码表传输的负担。若采用逃逸码，则量化后，需要使用逃逸码的概率将大大减少。*

在选择参数之前，我们应该选择一个合适的信源编码方式。本次实验采用的熵编码方式只有huffman编码，因此选择合适的量化编码方式就尤为重要。下面讨论JPEG编码和lloyd\_max编码的优劣。

固定信道噪声σ=0.5，采用以上的信道、编码模块联试传输图像，测试不同量化方案的性能，

以下是原图、Lloyd\_max和JPEG的接收图案对比：



上面的三幅图像量化方式和比特率依次为：

原图：131 072 bits

JPEG： 69 833 bits

Lloyd：69 329 bits（28阶量化，双符号huffman）

仔细观察会发现lloyd\_max算法中的噪点相当多。不过，如果不对图像质量做过高的要求，它的性能也是可以接受的。

虽然如此，本次实验中我们还是希望探索除JPEG之外量化编码方式的性能，因此采用Lloyd\_max作为量化模块，设置其**量化阶数为n**。

除此之外，本次实验的采取基于条带的VLC编码方式，因此**条带的高度h**也是参数之一;同时huffman编码的单/双符号也需要设置。到此为止，我们获得了本次实验中需要调节的全部四个参数：

**信道编码参数k，决定传输代价和准确率；**

**量化阶数n，决定图像重建质量；**

**条带高度h，决定huffman码错误扩散范围；**

**Huffman编码模式mode，决定huffman编码方式。**

接下来对这四个参数的选择加以讨论。

## 参数选择

我们要达成的主要目标是，

（1）尽量降低要传输的符号数；

（2）尽量提升抗噪声性能；

（3）尽量提升重建图像质量。

接下来就这三点目标决定参数的选择。

### 尽量降低要传输的符号数

要降低符号数，可以选择较小的量化阶数n，选择huffman编码模式为双符号联合编码。(值得注意的是，由于k同时影响编码效率和映射数，因此不会影响符号数)以下为不同参数设置下，lloyd\_max-huffman-viterbi联合编码下，信道中传输的符号数：

|  |  |
| --- | --- |
| **(n,mode)** | **信道传输符号数** |
| (16,1) | 63491 |
| (16,2) | 51837 |
| (32,1) | 75265 |
| (32,2) | 64425 |

可见为降低要传输的符号数，应当尽量选择较小的量化阶数，同时选择huffman编码模式为双符号联合编码。

### 尽量提升抗噪声性能

我们在实验一中已经验证过，相同判决方式下，k=2较k=3有更好的抗噪声性能。理论分析也是如此，电平信道中，在噪声功率一定时，符号电平间距越大，抗噪声性能越强，显然4-PSK抗噪声性能较强。

此外，降低传输符号数也是抗噪声的良好方式。由于我们采用熵编码，只要有一个位置出错就会导致连环出错，因此应尽量保证零传输错误。在每个符号出错概率相同的情况下，显然降低传输符号数可以提升抗噪声性能。

另外，在基于条带的VLC编解码中，加入条带码的代价不是很大，但能将错误限制在小区域内。因此，应当让条带高度h尽量地小（本实验中h=4）。

### 尽量提升重建图像质量

根据以上的分析，我们已经可以选定k=2，h=4，mode=2，只剩下量化阶数n还没有完全确定。增大n可以提升图片质量，但也会增大数据量，同时增大出错的可能性。

在此给出一个n的确定方法：

首先，由于选定n，h和mode之后要传输的符号数就已经确定，在给定信道使用次数m的情况下，先选定最大的n。

之后，由发端向收端传送图像。若收端解码错误，则给发端回复NACK，发端接到此信息后零n=n-α，再次发送图像。这个过程一直持续到收端解码正确并给发端回复ACK为止。

之后的所有图像传输中，让n保持不变，这样在保证正确性和传输速率的前提下，也保证了最高的图片传输质量。