

# レポート提出票

科目名: 情報工学実験2

実験テーマ: 実験テーマ3 情報通信シミュレーション

実施日: 2020年 12月 14日

学籍番号: 4619055

氏名: 辰川力駆

共同実験者:

_____	_____
_____	_____
_____	_____
_____	_____

## 1 実験概要

(7,4) ハミング符号による符号化、復号を行うプログラムを作成して、ハミング符号の特徴を理解する。

## 2 目的

## 3 原理

## 4 実験手順

1.  $K = 4$  ビットの情報  $w$  を乱数を用いて生成
2. 符号化により、 $w$  から  $x$  を生成
3. 乱数を用いて  $N = 7$  ビットの雑音ベクトル  $e$  を生成
4.  $y = x \oplus e$  を計算
5.  $y$  から復号して  $\hat{x}$ 、 $\hat{w}$  を得る
6.  $w$  と  $\hat{w}$  を比較し、ビット誤り数をカウント
7. 1 から 6 を SIM 回行い、ビット/ブロック誤り率を計算

## 5 実験結果

ソースコードは付録に記述した。そのソースコードを実行した結果を下記に示す。

図 1: プログラムの実行結果

## 6 検討

### 6.1 課題 1

なぜ (7,4) ハミング符号は 1 個誤りを訂正できるか。

ハミング符号の生成多項式は  $g(x) = x^3 + x + 1$  であるから、今回の冗長は 3 ビットなので、情報系列  $\mathbf{w} = (x_1, x_2, x_3, x_4)$  としたとき、冗長  $\mathbf{c} = (c_1, c_2, c_3)$  は次のようになる。

$$c_1 = x_1 \oplus x_2 \oplus x_3$$

$$c_2 = x_2 \oplus x_3 \oplus x_4$$

$$c_3 = x_1 \oplus x_2 \oplus x_4$$

よって、情報系列  $\mathbf{w} = (x_1, x_2, x_3, x_4)$  に対して、冗長  $\mathbf{c} = (c_1, c_2, c_3)$  は 1 対 1 対応に決まる。

したがって、

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

の転置を右から書けた値もただ一つに決まるので、シンδροーム  $\mathbf{s} = (s_1, s_2, s_3)$  は、ただ一つに決まる。

したがって、誤り箇所が分かるので 1 個の誤りを訂正できる。

### 6.2 課題 2

2, 3, ... 個の誤りが発生するとどうなるか。

$\mathbf{s} = \mathbf{y}H^T$  において、この計算式でシンδροームを計算するが、 $\mathbf{y}$  が誤りを 2 つや 3 つ発生していても、シンδροーム  $\mathbf{s}$  は 3 ビットしか得られない。つまり、 $2^3 = 8$  通りより、誤りがないもしくは 1 ~ 7 ビット目が誤っているとしか判定できない。したがって、1 個より大きい誤りが発生した場合は訂正はできない。

## A 付録

ソースコード 1: kadai3\_3.cpp

```
1 //4619055 辰川力駆
2 #include <random> // 乱数生成
3 #include <stdio.h>
4 #include <iostream>
5 #include <iomanip>
6
7 using namespace std;
8
9 #define N 7 ///符号化後のビット数
10 #define K 4 ///デジタル情報の分けるブロックのビット数
11 #define seed 55 ///学籍番号下2桁
12 #define SIM 1000000 ///シミュレーション回数
13
14 mt19937 mt(seed); ///メルセンヌ・ツイスタ
15
16 int main()
17 {
18     uniform_real_distribution<double> rand_real(0, 1);
19     normal_distribution<double> rand_n(0, 0.3);
20
21     int w[K]; ///4ビットの情報w
22     int x[N]; ///7ビットの符号語x
23     int e[N]; ///7ビットの雑音ベクトルe
24     int y[N]; ///7ビットの受信語y
25     int s[3]; ///シンドロームs
26     int bit_count = 0; ///シミュレーションごとの毎回のビット誤り数を計算
27     int total_bit_count = 0; ///誤ったビットの総数
28     int total_block_count = 0; ///ブロック単位の誤りの総数
29     int EstimationPosition; ///誤り位置推定場所
30     double ep; ///誤り率
31
32     cout << "#_SIM:" << SIM << endl;
33     cout << "#BSC の誤り率_#復号後ビット誤り率_#復号後ブロック誤り率" << endl;
34
35     for (int k = 0; k < 5; k++)
36     {
37         ep = 0.005 + k * 0.004;
38         total_bit_count = 0;
39         total_block_count = 0;
40
41         for (int sim = 0; sim < SIM; sim++)
42         {
43             bit_count = 0;
44
45             for (int i = 0; i < K; i++) ///K=4 ビットの情報 w を乱数を用いて生成
46             {
47                 w[i] = rand_real(mt) * 2;
48             }
```

```

49
50     for (int i = 0; i < K; i++) ///符号化により, w から x を生成
51     {
52         x[i] = w[i];
53     }
54     x[N - K + 1] = w[0] ^ w[1] ^ w[2];
55     x[N - K + 2] = w[1] ^ w[2] ^ w[3];
56     x[N - K + 3] = w[0] ^ w[1] ^ w[3];
57
58     for (int i = 0; i < N; i++) ///乱数を用いてN=7ビットの雑音ベクトル
        e を生成
59     {
60         if (rand_real(mt) <= ep)
61         {
62             e[i] = 1;
63         }
64         else
65         {
66             e[i] = 0;
67         }
68     }
69
70     for (int i = 0; i < N; i++) ///x と e の排他的論理和を計算して y とする
71     {
72         y[i] = x[i] ^ e[i];
73     }
74
75     s[0] = y[0] ^ y[1] ^ y[2] ^ y[4]; ///y から復号して x ハット、w ハットを得る
76     s[1] = y[1] ^ y[2] ^ y[3] ^ y[5];
77     s[2] = y[0] ^ y[1] ^ y[3] ^ y[6];
78
79     int point = 0;
80     for (int i = 0; i < 3; i++)
81     {
82         point += s[i] * pow(2, 2 - i);
83     }
84     switch (point)
85     {
86     case 5:
87         EstimationPosition = 1;
88         break;
89     case 7:
90         EstimationPosition = 2;
91         break;
92     case 6:
93         EstimationPosition = 3;
94         break;
95     case 3:
96         EstimationPosition = 4;
97         break;
98     case 4:
99         EstimationPosition = 5;

```

```

100         break;
101     case 2:
102         EstimationPosition = 6;
103         break;
104     case 1:
105         EstimationPosition = 7;
106         break;
107     default:
108         EstimationPosition = -1;
109         break;
110     }
111     if (EstimationPosition != -1)
112     {
113         y[EstimationPosition - 1] = y[EstimationPosition - 1] ^ 1;
114     }
115
116     for (int i = 0; i < K; i++) //w と wハットを比較し、ビット誤り数をカウン
117         ト (つまり 4ビットまでを見れば良い)
118     {
119         bit_count += w[i] ^ y[i];
120     }
121
122     total_bit_count += bit_count;
123
124     if (bit_count != 0)
125     {
126         total_block_count += 1;
127     }
128     cout << ep;
129     cout << fixed << setprecision(8) << "          " << (double)total_bit_count /
130         (K * SIM);
131     cout << fixed << setprecision(6) << "          " << (double)
132         total_block_count / SIM << endl;
133     cout.unsetf(ios::fixed); ///体裁を整えている
134 }
135 return 0;
136 }

```