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 7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　実験手順

1. 4 ビットの情報 $\boldsymbol{w}$ を生成
2. 生成多項式 $g(x)$ を用いて符号化
3. 1 ビットの雑音を付与する
4. 受信語 $\boldsymbol{y}$ と検査行列 H からシンドローム $\boldsymbol{s}$ を計算
5. シンドロームから誤り位置を推定する
6. 推定語 $\hat{\boldsymbol{x}}$ を求め、もとの $\boldsymbol{x}$ と値を比較する

3　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。

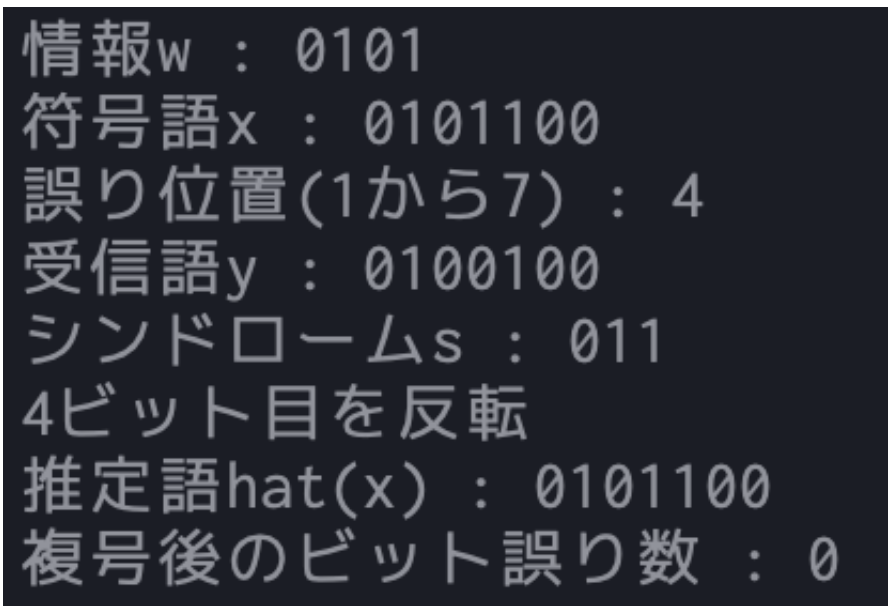

図 1: 情報 = 0101、誤り位置 = 4 の実行結果

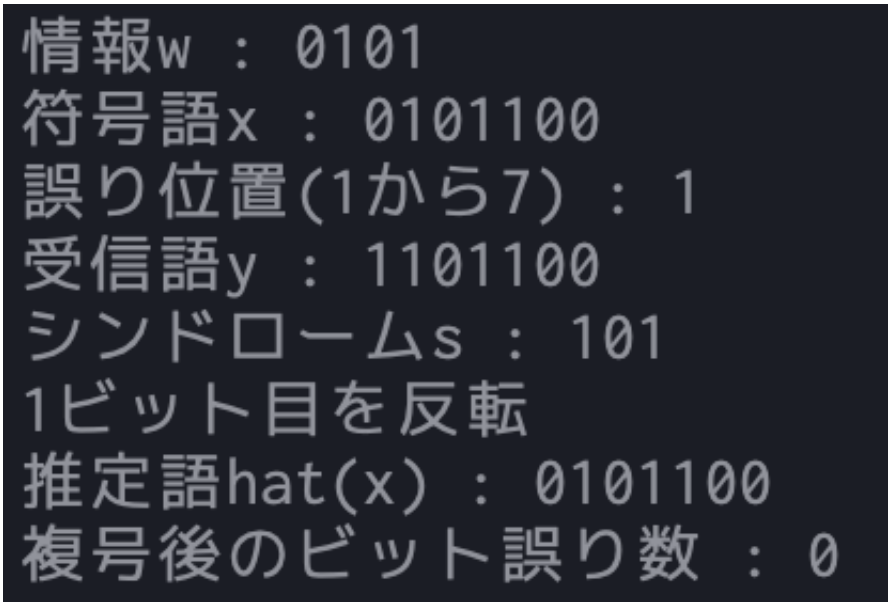

図 2: 情報 = 0101、誤り位置 = 1 の実行結果

4 検討

4.1 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。

4.2 課題 2

2, 3, ... 個の誤りが発生するとどうなるか。

$\boldsymbol{s} = \boldsymbol{y}H^T$ において、この計算式でシンドロームを計算するが、$\boldsymbol{y}$ が誤りを 2 つや 3 つ発生していても、シンドローム $\boldsymbol{s}$ 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_2.cpp

```cpp
//4619055 辰川力駆
#include <random> // 乱数生成
#include <stdio.h>
#include <iostream>
#include <iomanip>

using namespace std;

#define N 7 //符号化後のビット数
#define K 4 //デジタル情報の分けるブロックのビット数

int main()
{
    int w[K] = {0, 1, 0, 1}; //4ビットの情報w
    int x[N]; //7ビットの符号語x
    int y[N]; //7ビットの受信語y
    int ErrorPosition; //誤り位置(1から7)
    int s[3]; //シンドローム s
    int EstimationPosition; //誤り位置推定場所
    int cnt = 0; //復号後のビット誤り数

    cout << "情報 w␣:␣";
    for (int i = 0; i < K; i++)
    {
        cout << w[i];
    }
    cout << endl;

    for (int i = 0; i < K; i++)
    {
        x[i] = w[i];
    }
    x[N - K + 1] = w[0] ^ w[1] ^ w[2];
    x[N - K + 2] = w[1] ^ w[2] ^ w[3];
    x[N - K + 3] = w[0] ^ w[1] ^ w[3];
    cout << "符号語 x␣:␣";
    for (int i = 0; i < N; i++)
    {
        cout << x[i];
    }
    cout << endl;

    cout << "誤り位置(1から7)␣:␣";
    cin >> ErrorPosition;

    cout << "受信語 y␣:␣";
    for (int i = 0; i < N; i++)
    {
```

```
49              y[i] = x[i];
50              if (i == ErrorPosition - 1)
51              {
52                  y[i] = x[i] ^ 1; //誤り位置は反転する
53              }
54              cout << y[i];
55          }
56          cout << endl;
57
58          s[0] = y[0] ^ y[1] ^ y[2] ^ y[4];
59          s[1] = y[1] ^ y[2] ^ y[3] ^ y[5];
60          s[2] = y[0] ^ y[1] ^ y[3] ^ y[6];
61          cout << "シンドローム s␣:␣";
62          for (int i = 0; i < 3; i++)
63          {
64              cout << s[i];
65          }
66          cout << endl;
67
68          int point = 0;
69          for (int i = 0; i < 3; i++)
70          {
71              point += s[i] * pow(2, 2 - i);
72          }
73          switch (point)
74          {
75          case 5:
76              EstimationPosition = 1;
77              break;
78          case 7:
79              EstimationPosition = 2;
80              break;
81          case 6:
82              EstimationPosition = 3;
83              break;
84          case 3:
85              EstimationPosition = 4;
86              break;
87          case 4:
88              EstimationPosition = 5;
89              break;
90          case 2:
91              EstimationPosition = 6;
92              break;
93          case 1:
94              EstimationPosition = 7;
95              break;
96          default:
97              EstimationPosition = -1;
98              break;
99          }
100
```

```
101         if (EstimationPosition == -1)
102         {
103             cout << "誤りはなし" << endl;
104         }
105         else
106         {
107             cout << EstimationPosition << "ビット目を反転" << endl;
108         }
109 
110         y[EstimationPosition - 1] = y[EstimationPosition - 1] ^ 1;
111         cout << "推定語 hat(x)␣:␣";
112         for (int i = 0; i < N; i++)
113         {
114             cout << y[i];
115         }
116         cout << endl;
117 
118         for (int i = 0; i < N; i++)
119         {
120             cnt += x[i] ^ y[i];
121         }
122         cout << "複号後のビット誤り数␣:␣" << cnt << endl;
123 
124         return 0;
125     }
```