情報工学実験 2 数理計画法 第三回

学生番号 4617043 神保光洋

2018 年 11 月 22 日

1　実験の要旨

実験課題 1 で求めたビット誤り率 P_e より誤り率を小さくする方法を扱う。具体的には、実験課題 2 で作成したハミング符号を用いて実験課題 1 と同様、繰り返しシュミレーションを行って誤り率 (BER)P_{dec} を求める。

2　実験の目的

実験課題 1 で求めたビット誤り率 P_e より誤り率を小さくする。

3　実験の原理

$(n, k) = (7, 4)$ ハミング符号を用いる。$(7, 4)$ ハミング符号とは代表的な誤り訂正符号の一つであり、単一誤りの訂正を可能とする。符号化と呼ばれる操作によって 4 ビットの情報を 7 ビットの符号語に変換し、送信する。

4　実験装置あるいは実験方法

4.1　実行環境

Mojave 10.14.1 zsh 5.3 (x86_64-apple-darwin18.0) g++ 4.2.1

4.2　実験手順

4.2.1　情報系列 $w = (w_1, w_2, ..., w_k)$ の生成

(課題 1)

4.2.2　符号語 $x = (x_1, x_2, ..., x_k)$ の生成

(課題 2)

4.2.3 BSC での雑音 $e = (e_1, e_2, ..., e_n)$ の発生と受信系列 $y = (y_1, y_2, ..., y_n)$ の生成

(課題 1)

4.2.4 ハミング符号の復号

受信系列 y から復号を行い、推定符号語 $\hat{x} = (\hat{x_1}, \hat{x_2}, ..., \hat{x_n})$, 推定情報系列 $\hat{w} = (\hat{w_1}, \hat{w_2}, ..., \hat{w_n})$ を得る。(課題 2)

4.2.5 誤りビット数の算出

情報系列 w と推定情報系列 $\hat{w}$ を比較し誤りビット率を求める (n-k ビットの冗長ビットは含めない) (課題 1)

4.2.6 復号後のビット誤り率 P_dec の計算

１５を十分な精度が得られるまで繰り返し (=SIM 回、自分で適当な値に設定)、復号後のビット誤り率 P_dec を求める。P_dec は誤った推定情報系列ビットの総数を送信した情報系列のビット総数 (k * SIM) で割ったものある。(課題１)

4.2.7 １６を ϵ に対し実行する

5 結果

結果は以下の様になった。

```
^^e2^^9e^^9c g++ Source.c && ./a.out
eps       P_e
0.001000  0.000015
0.002000  0.000040
0.003000  0.000085
0.004000  0.000198
0.005000  0.000278
0.006000  0.000260
0.007000  0.000395
0.008000  0.000500
0.009000  0.000682
0.010000  0.000847
```

誤り率の結果と理論値は以下のグラフの様になった。

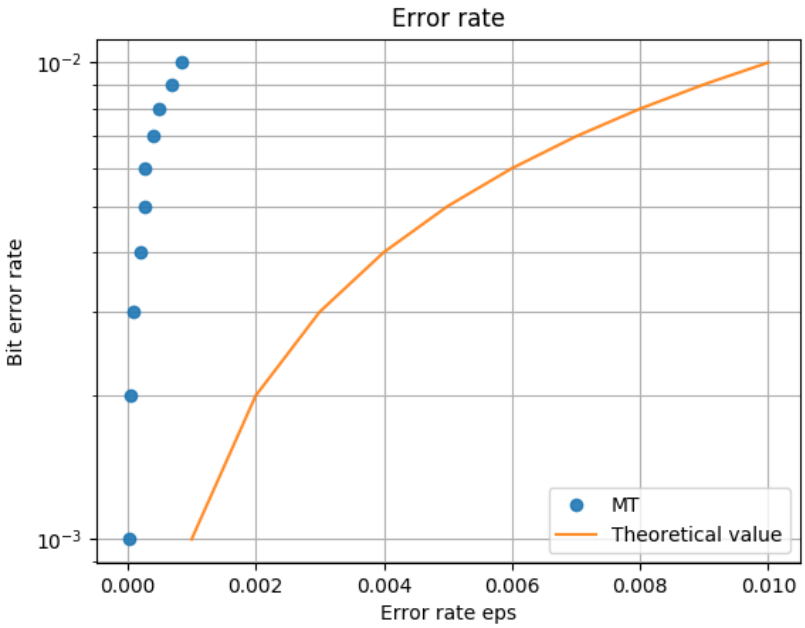

図 1　誤り率の結果と理論値

6　検討・考察

6.1　検討事項 3-1

ϵ が小さいとき、シュミレーションが少ないと BER が 0 となり正しく計測できない場合がある。0 より大きい BER を計測するにはどの程度のシュミレーション回数を実行する必要があるか。

情報工学実験 2 情報シュミレーション第一回目の検討同様少なくとも 100000 回は必要であると検討される。

6.2　検討事項 3-2

ハミング符号を用いたとき、BER がどのように変化するか理論的に考察せよ。

グラフ 1 よりグラフ 1 は対数グラフをとっているが、P_{dec} の値は直線的に変化していることが考察される。これは理論値と比べるとより明らかである。したがって BER の変化は誤り率を e とすると $r \in R$ を用いて

$$P_dec = r^e$$

と近似されると考察される。したがってどのように変化されるかであるが

$$\frac{P_dec}{e} = r^e$$

より変化率 r^e で変化すると考察される。

6.3 検討事項 3-3

ビット誤り率ではなく、ブロック誤り率 (4 ビットの情報系列を 1 ブロックとみなした場合) で計測したシュミレーション結果を示し、理論的に導出した (ブロック単位の) 復号誤り確率と比較せよ。

P_{dec} の値は以下のように導出される。

$$P_{dec} = \frac{\text{誤った情報系列ブロックの総数}}{\text{送信した情報系列のブロックの総数 } (SIM)}$$

結果は以下のようになった。

eps	P_e
0.001000	0.000040
0.002000	0.000090
0.003000	0.000200
0.004000	0.000440
0.005000	0.000630
0.006000	0.000600
0.007000	0.000920
0.008000	0.001190
0.009000	0.001510
0.010000	0.001960

ブロック誤り率を用いた場合の結果のグラフは以下のようになる

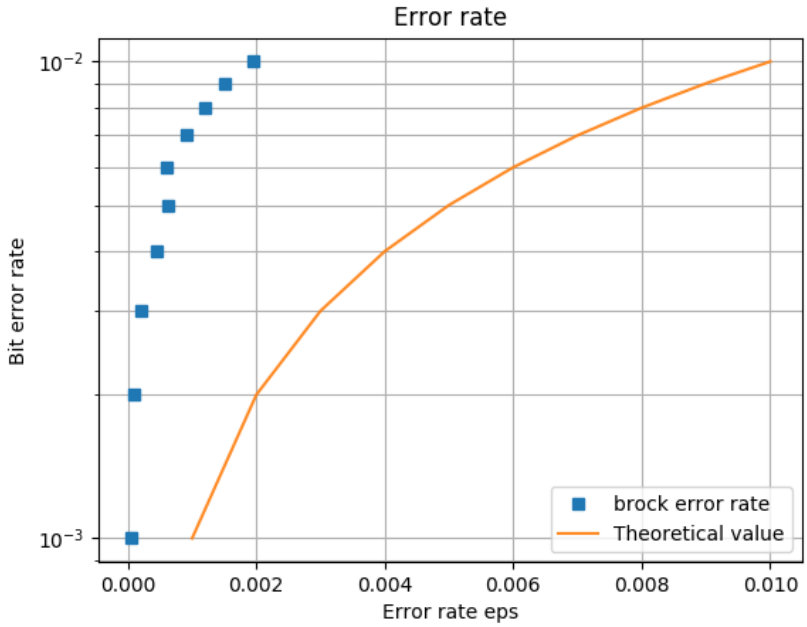

図 2　ビット誤り率、ブロック誤り率比較

グラフより理論値の乖離度合いはビットを用いたものよりも大きいと考察されるこれは $k = 4$ から $k = 1$ になったためと言える。

7 結論

通信路モデルにおいてハミング符号を用いた方法を理解できた。

また random はそもそも c++ のヘッダであり (ヘッダファイルでないことに注意) c++ であるのになぜ拡張子を c 言語にしたかはこれはこの実験の製作者が c 言語、c++ に疎いためと推測される。これは来年再来年に学んでいく学生に有害であるとも推測されるのでここに正しておく。このあたりは評価されたい。

8 参考文献

参考文献

[1] 山本和彦 プログラミング Haskell・オーム社
[2] 高橋麻奈 やさしい C 言語 (第 5 版)・SB クリエイティブ
[3] 植松友彦 情報理論の考え方 ・講談社

9 付録

実験で使用したプログラムは以下のようになる。

ソースコード 1 Source.c

```
#include <stdio.h>
#include <random>

void n2b(int n, int bi[], int bi_len) {
        if (n == 0) {
                return;
        }
        else {
                bi[bi_len++] = n % 2;
                n2b(n / 2, bi, bi_len);
        }
}

void init_bi(int bi[], int n) {
        int i;
        for (i = 0; i < n; i++) {
```

```
                bi[i] = 0;
        }
}

void reverse4(int bi[], int n) {
        int tmp[7];

        int i;
        for (i = 0; i < n; i++) {
                tmp[i] = bi[i];
        }
        init_bi(bi, n);
        bi[3] = tmp[0];
        bi[2] = tmp[1];
        bi[1] = tmp[2];
        bi[0] = tmp[3];
}

void add_c(int bi[]) {
        bi[4] = bi[0] ^ bi[1] ^ bi[2]; // c_0
        bi[5] = bi[1] ^ bi[2] ^ bi[3]; // c_1
        bi[6] = bi[0] ^ bi[1] ^ bi[3]; // c_2
}

void cpy_bit2bits(int bits[16][7], int bi[], int i, int n) {
        int j;
        for (j = 0; j < n; j++) {
                bits[i][j] = bi[j];
        }
}

void make_bits(int bits[16][7]) {
        int n = 7;
        int bi[7];
        int i;
        for (i = 0; i < 16; i++) {
                init_bi(bi, n);
```

```
                n2b(i, bi, 0);
                reverse4(bi, n);
                add_c(bi);
                cpy_bit2bits(bits, bi, i, n);
        }
}

int hum(int bis[16][7], int y[], int w[]) {
        int min_hum = 100000;
        int min_hum_pattern = 0;
        int i, j;

        for (i = 0; i < 16; i++) {
                int hum_num = 0;

                for (j = 0; j < 7; j++) {
                        if (bis[i][j] != y[j]) {
                                hum_num++;
                        }
                }

                if (min_hum > hum_num) {
                        min_hum         = hum_num;
                        min_hum_pattern = i;
                }

        }
        return min_hum_pattern;
}

int count_e(int bis[16][7], int w[], int p_num) {
        int i;
        int e_num = 0;
        for (i = 0; i < 4; i++) {
                if (bis[p_num][i] != w[i]) {
                        e_num++;
                }
```

```
        }
        return e_num;
}

int main(void) {
        std::mt19937 mt(100);
        std::uniform_real_distribution<double> r_rand(0.0, 1.0);

        double ran = r_rand(mt);

        double xi = 0.001;
        int n = 7;
        int k = 4;
        int i;
        int w[7];
        int e[7];
        int y[7];
        int bis[16][7];
        make_bits(bis);

        int sim = 100000;
        int t;
        int all_e = 0;

        int r;
  printf("eps       P_e\n");
        for (r = 0; r < 10; r++) {
                all_e = 0;
                for (t = 0; t < sim; t++) {
                        for (i = 0; i < k; i++) {
                                if (r_rand(mt) <= 0.5) {
                                        w[i] = 1;
                                }
                                else {
                                        w[i] = 0;
                                }
                        }
```

```
                w[4] = w[0] ^ w[1] ^ w[2]; // c_0
                w[5] = w[1] ^ w[2] ^ w[3]; // c_1
                w[6] = w[0] ^ w[1] ^ w[3]; // c_2

                for (i = 0; i < n; i++) {
                        if (r_rand(mt) <= xi) {
                                e[i] = 1;
                        }
                        else {
                                e[i] = 0;
                        }
                }

                for (i = 0; i < n; i++) {
                        y[i] = w[i] ^ e[i];
                }

                // 推定符号語計算
                int min_hum_pattern = hum(bis, y, w);
                all_e = all_e + count_e(bis, w, min_hum_pattern);
        }

        printf("%f %f\n", xi, (double)all_e / (double)k / sim); // ビット誤り率
        xi = xi + 0.001;
    }
    return 0;
}
```