学籍番号: 4617041 氏名: 白江涼雅

課題2 ハミング符号を用いた誤り訂正

1 目的

実際にハミング符号のスクリプトをC言語で生成し, ハミング符号についての理解を深める.

2 実験装置

- windows X シリーズ
- Visualstdio2013

3 結果

最後に示すソースコードを動かすと, 以下のような結果になった.

```
情報語:  1  1  0  1
符号語:  1  1  0  1  0  1  0
誤らせるのは何回目 (1~7)？:4
受信語  1  1  0  0  0  1  0
シンドローム  0  1  1
4 ビット目を反転させます
推定後符号語  1  1  0  1  0  1  0
符号語のビット誤り数は 0 です
```

図 1:ハミング符号のプログラム実行結果

4 検討事項

1. なぜ (7,4) ハミング符号は1個誤りを訂正できるか.
   最後にのせているソースコードにあるように生成行列 G=$[I_k A]$ を作った.G に合わせて H=$[A^T I]$ もつくった.G と H の関係は $HG^T = HG^T = 0$ となる値をとる. そうすることで、G に情報語 i をかけてたとしても $iG^T H = 0$ とすることができるためである.
   i に右から G をかけて送信符号 x を作り, 誤り符号 e を付け加えた値を受信信号 y とする. つまり y は $y = iG + e$ となっている.
   受信信号を受け取った側は y に対し H^T をかけ、シンドローム s を得られる. つまり s は $s = (iG + e)H^T$ である.$iGH^T = 0$ であるため、残るのは eH^T である. もし s が [0 0 0] ならば誤りはなく, 誤りがあるのなら誤りに対応するシンドロームがでる.
   といった理由で1個誤りは訂正できる. ここではシンドロームは3行1列であるため,1元で8通りのシンドロームが表わせ、0もしくは1個誤りは1元的で合計8通りであるため、e は一意に定まる.
   すなわち、誤りの数がシンドロームのパターンとすべて異なるように表現できるので、誤りを特定できる.

2. 2 個以上の誤りが発生するとどうなるか.2 個,3 個, ... とするとどうなるか.
   (7,4) ハミング符号で 2 個以上の誤りが発生してしまうと,s の 3 行 1 列では表しきれないため, 想定される何通りかまでは絞れるが, 一意には定まらなくなってしまう.

5 ソースコード

```
#include <stdio.h>
#include <random>
#define gyo 7 //ハミング符号
#define retu 4 //ハミング符号
#define k 3 //シンドロームの長さ

int main(){
int G[gyo][retu];//生成行列
int H[gyo][k]; //検査行列
int w[retu];  //情報系列
int x[gyo];  //送信系列
int y[gyo];  //受信系列
int e[gyo];  //誤り符号
int s[gyo]; //シンドローム生成
double ran;
int i,j;
int tmp;
int miss;

//乱数発生準備
std::mt19937 mt(41);
std::uniform_real_distribution<double> r_rand(0.0, 1.0);


//生成行列で必要な任意にきめるところの生成
G[0][0] = 1; G[0][1] = 0; G[0][2] = 0; G[0][3] = 0;
G[1][0] = 0; G[1][1] = 1; G[1][2] = 0; G[1][3] = 0;
G[2][0] = 0; G[2][1] = 0; G[2][2] = 1; G[2][3] = 0;
G[3][0] = 0; G[3][1] = 0; G[3][2] = 0; G[3][3] = 1;
G[4][0] = 1; G[4][1] = 1; G[4][2] = 1; G[4][3] = 0;
G[5][0] = 1; G[5][1] = 1; G[5][2] = 0; G[5][3] = 1;
G[6][0] = 1; G[6][1] = 0; G[6][2] = 1; G[6][3] = 1;

H[0][0] = 1; H[0][1] = 1; H[0][2] = 1;
H[1][0] = 1; H[1][1] = 1; H[1][2] = 0;
H[2][0] = 1; H[2][1] = 0; H[2][2] = 1;
H[3][0] = 0; H[3][1] = 1; H[3][2] = 1;
```

```
H[4][0] = 1; H[4][1] = 0; H[4][2] = 0;
H[5][0] = 0; H[5][1] = 1; H[5][2] = 0;
H[6][0] = 0; H[6][1] = 0; H[6][2] = 1;

/*
//生成確認
for (i = 0; i < retu; i++){
for (j = 0; j < gyo; j++){
printf("%3d", G[j][i]);
}
printf("\n");
}
printf("\n\n");

for (i = 0; i < k; i++){
for (j = 0; j < gyo; j++){
printf("%3d", H[j][i]);
}
printf("\n");
}
printf("\n\n");
*/
//w の生成
for (i = 0; i < retu; i++){
ran = r_rand(mt);
if (ran < 0.5){
w[i] = 0;
}
else{
w[i] = 1;
}
}
//w の出力
printf("情報語:");
for (i = 0; i < retu; i++){
printf("%3d", w[i]);
}
printf("\n");

//x の生成
for (i = 0; i < gyo; i++){
tmp = 0;
for (j = 0; j < retu; j++){
tmp += w[j] * G[i][j];
}
```

```
if (tmp % 2 == 0){
x[i] = 0;
}
else{
x[i] = 1;
}
}
//x の確認
printf("符号語:");
for (i = 0; i < gyo; i++){
printf("%3d", x[i]);
}
printf("\n");

//誤り e の生成
for (i = 0; i < gyo; i++){
e[i] = 0;
}
printf("誤らせるのは何回目 (1~7)？:"); scanf("%d", &miss);
e[miss-1] = 1;

//送信行列に誤り e を干渉させ, 送信行列をつくる
for (i = 0; i < gyo; i++){
y[i] = (x[i] + e[i]) % 2;
}
//y の確認
printf("受信語");
for (i = 0; i < gyo; i++){
printf("%3d", y[i]);
}
printf("\n");


//s の生成
for (i = 0; i < k; i++){
s[i] = 0;
for (j = 0; j < gyo; j++){
s[i] += y[j] * H[j][i];
}
if (s[i] % 2 == 1){
s[i] = 1;
}else{
s[i] = 0;
}
}
```

```
printf("シンドローム");
for (i = 0; i < k; i++){
printf("%3d", s[i]);
}
printf("\n");

//どこ反転させるかの判定
if (s[0] == 1 && s[1] ==1 && s[2] ==1 ){
miss = 1;
}else if (s[0] == 1 && s[1] == 1 && s[2] == 0){
miss = 2;
}
else if (s[0] == 1 && s[1] == 0 && s[2] == 1){
miss = 3;
}
else if (s[0] == 0&& s[1] == 1&& s[2] == 1){
miss = 4;
}
else if (s[0] == 1&& s[1] == 0&& s[2] == 0){
miss = 5;
}
else if (s[0] == 0&& s[1] == 1&& s[2] == 0){
miss = 6;
}
else if (s[0] == 0&& s[1] == 0&& s[2] == 1){
miss = 7;
}
else{
miss = 0;
}

printf("%d ビット目を反転させます\n",miss);

if (y[miss - 1] == 0){
y[miss - 1] = 1;
}
else{
y[miss - 1] = 0;
}
printf("推定後符号語");
for (i = 0; i < gyo; i++){
printf("%3d", y[i]);
}
printf("\n");
printf("符号語のビット誤り数は 0 です\n");
```

```
}
```

図 2:ハミング符号プログラムソースコード

6　参考文献

参考文献

[1] 実験の際もらったプリント

[2] 情報工学実験 2 (2018)