学籍番号: 4617041 氏名: 白江涼雅

課題１ C言語による乱数発生

1 目的

C 言語の乱数発生手段である rand() 関数と Mersenne Twister(MT) を用いて、それぞれについて理解を深める。

2 実験装置

- windows X シリーズ
- Visualstdio2013

3 結果

以下のになる。また、最後のページに図をまとめた.

表１:100 万回試行 誤り確率 ϵ 10^{-5} から 10^{-4} の時のビット誤り率 P_e

誤り確率 ϵ	rand 関数の P_e	MT の P_e
0.00001	0.000032	0.000011
0.00002	0.000031	0.000021
0.00003	0.000028	0.000032
0.00004	0.000065	0.000041
0.00005	0.000062	0.000048
0.00006	0.000056	0.000052
0.00007	0.000096	0.000069
0.00008	0.000095	0.000080
0.00009	0.000085	0.000091
0.00010	0.000123	0.000104

表 2:100 万回試行 誤り確率 ϵ 10^{-3} から 10^{-2} の時のビット誤り率 P_e

誤り確率 ϵ	rand 関数の P_e	MT の P_e
0.001	0.000978	0.000987
0.002	0.002031	0.001971
0.003	0.003037	0.003001
0.004	0.004000	0.003998
0.005	0.004988	0.004965
0.006	0.005960	0.006013
0.007	0.007015	0.006955
0.008	0.008024	0.008059
0.009	0.009036	0.009045
0.010	0.009936	0.010098

4 検討事項

1. BER と理論値がほぼ同じ値になるには rand 関数は周期があり、2147483648 回行えば良いと考えられる。そのことについては検討事項 3 でしめす.

2. 表１、２をみてわかる通り rand 関数を使った場合に顕著に現れるが、ϵ の値が理論値から大きく外れる. 検討事項 3 で示すが、精度の悪い乱数を使うと周期がでてきてしまうため良い値にならなかった.

3. - rand 関数について
   rand 関数のコードを確認した結果以下のように書かれていることがわかった.

```
static long x=1;
void srand(long s){ x=s; }
long rand() { x=x*1103415245+12345; return x%2147483647; }
```

図 1:UNIX 系 rand 関数のソースコード

これからわかる通り、擬似乱数列の生成式の線形合同法を利用していることがわかる。記憶領域を必要とせず低機能な PC でも動作するメリットがあるが規則的に分布してしまうという特性がある. そして、$2^{31} = 2147483647$ で割ったあまりを出力しているため 2^{31} 周期とわかる. つまり多く試行を重ねるたびにい乱数として扱うことは難しくなってくる。以上の理由から精度の悪い乱数とされている。

- メルセンヌ・ツイスタについて
このコードを私の実力では読むことができなかったため、調べた内容で示す。メルセンヌ・ツイスタは $2^{19937} - 1$ の周期であるため現実的には周期がないと思ってよく、試行回数を常識の範囲内で多く増やしたとしても乱数として扱える。
- 以上のことから検討事項 1,2 のような結果になった.

5 ソースコード

より一般的に読めるように、define の部分は省いた.

```
#include <stdio.h>
#include <stdlib.h>
int main(){
int miss_count;
int Transmission[K],Reception[K],nouse[K];
double ran;
int i,j,k;
double delta = 0;
srand((unsigned)41);
for (k = 1; k < probability; k++){
miss_count = 0;
delta = delta + delta_plus;
for (j = 0; j < SIM; j++){
//乱数４つ生成:送るデータのやつ
for (i = 0; i < K; i++){
ran = (double)rand() / RAND_MAX;
if (ran < 0.5){Transmission[i] = 0;}else{Transmission[i] = 1;}
                        }
//雑音の乱数４つ
for (i = 0; i < K; i++){
ran = (double)rand() / RAND_MAX;
if (ran < delta){noise[i] = 1;}else{noise[i] = 0;}
}

//送られてくるデータ
for (i = 0; i < K; i++){
if (noise[i] == 0){Reception[i] = Transmission[i];}else{
Reception[i]=abs(Transmission[i]-1);
}if (Reception[i] != Transmission[i]){
miss_count++;
}
}
}
printf("deltaが%f のときのビット誤り率は%f\n",
delta, (double)miss_count / (double)(K * SIM));
}
return 0;
}
```

図 2:rand 関数を使った時のソースコード

```
#include <stdio.h>
#include <stdlib.h>
int main(){
int miss_count;
int Transmission[K],Reception[K],noise[K];
double ran;
int i,j,k;
double delta = 0;
std::mt19937 mt(100);
std::uniform_real_distribution<double> r_rand(0.0, 1.0);

for (k = 1; k < probability; k++){
miss_count = 0;
delta = delta + delta_plus;
for (j = 0; j < SIM; j++){
//乱数４つ生成:送るデータのやつ
for (i = 0; i < K; i++){
ran = r_rand(mt);
if (ran < 0.5){Transmission[i] = 0;}else{Transmission[i] = 1;}
}
//雑音の乱数４つ
for (i = 0; i < K; i++){
ran = r_rand(mt);
if (ran < delta){noise[i] = 1;}else{noise[i] = 0;}
}

//送られてくるデータ
for (i = 0; i < K; i++){
if (noise[i] == 0)
{Reception[i]=Transmission[i];}else{Reception[i] = abs(Transmission[i]-1);}
if (Reception[i] != Transmission[i]){miss_count++;}
}
}
printf("deltaが%f のときのビット誤り率は%f\n",
 delta, (double)miss_count / (double)(K * SIM));
}
return 0;
}
```

図 3:mt を使った時のソースコード

6　参考文献

参考文献

[1] 「良い関数、悪い関数」 www001.upp.so-net.ne.jp