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

2　実験手順

-

3　実験結果

ソースコードは付録に記述した。$\epsilon = 0.0001$ として、SIM を 1000000 回実行した結果は上記のようになった。また、それらの結果をグラフにプロットすると図 1 のようになった。なお、縦軸は対数で表示している。

図 1: ビット誤り率と誤り確率

4　検討

4.1　課題 1

シミュレーションによるビット誤り率と理論値がほぼ同じ値になるにはどの程度のシミュレーション回数を実行する必要があるか。

大数の法則より、シミュレーション回数を増やすほうが、理論値に近づくのは明白である。実際、SIM 回数を 1000000 回から減らすと、誤差が大きくなった。さらに誤差を減らすためには 1000000 回以上行えばよいが、実行するのに時間とてもかかってしまうだろう。

4.2　課題 2

ϵ を非常に小さくした場合、これはどうなるか。

ϵ を実験よりさらに 10 分の 1 の値 ($\epsilon = 0.00001$) とした場合、以下のような結果となった。行うことは同様であったが、誤差の割合を見ると大きくなっていると考えられる。これは、理論値を小さくすることにより相対的に誤差の割合が大きくなっていると考えられる。

4.3 課題 3

rand() と MT の違いは何か。

rand() 関数は環境にもよるが 0 ～ 32767 までの値しか取らない。環境が違っていても少なくとも最大値が存在する。今回は 0 ～ 1 の値を求めるだけに使ったのであまり変わらなかったが、0 ～ 10000 の値を求めるために乱数を使ったりするときは、最大値が存在するから偏りが生じる。よって、ほぼ均等に乱数を生成するならばメルセンヌ・ツイスタを用いるほうが良いと考える。

しかし、メルセンヌ・ツイスタで実行した場合、実行速度が (rand() 関数を用いた時より) 遅くなるので、速度を重視するか、性能を重視するかで使い分けると良いだろう。

A 付録

ソースコード 1: kadai3_rand.cpp

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

using namespace std;

#define SIM 1000000

#define real_rand (double)rand() / RAND_MAX; //
    RAND_MAX で割ることで 0 から 1 を返すようにしている。
#define seed 55 //学籍番号下 2 桁
#define K 4 //系列長

int main()
{
    srand(seed);
    int w[4], e[4], y[4];

    cout << "# SIM:" << SIM << endl;
    cout << "# ep    # BER" << endl;

    double ep = 0;
    int count = 0;
    for (int i = 0; i < 12; i++) //12回で設定
    {
        count = 0;
        ep = 0.0001 * (i + 1);

        for (int s = 0; s < SIM; s++)
        {
            for (int j = 0; j < K; j++)
            {
                double rd = real_rand; //乱数発生
                w[j] = rd * 2; //2倍することで 0.5より大きいか小さいかを判定できる。
            }
            for (int j = 0; j < K; j++)
            {
                double rd = real_rand; //乱数発生
                if (rd <= ep)
                {
                    e[j] = 1;
                }
                else
                {
                    e[j] = 0;
                }
```

```
48                  }
49                  for (int j = 0; j < K; j++)
50                  {
51                      y[j] = w[j] ^ e[j];
52                      count += w[j] ^ y[j];
53                  }
54              }
55              cout << fixed << setprecision(4) << ep;
56              cout.unsetf(ios::fixed);
57              cout << fixed << setprecision(7) << "␣" << (double)count / (K * SIM) <<
                    endl;
58          }
59
60          return 0;
61      }
```