情報工学実験１数理計画法

学生番号 4617043 神保光洋

2018 年 11 月 6 日

1　実験の要旨

c 言語において乱数を発生させる際に使う rand() 関数を用いたものと Mersenne Twister(MT) ををを使いディジタル通信システムと謝り訂正符号の理解を深める。

2　実験の目的

通信システムの「シュミレーション」を通し、ディジタル通信システムと誤り訂正符号の理解を深める。

3　実験の原理

3.1　誤り訂正符号

情報を伝えるとき、できるだけ正確に伝えるための仕組みであり、送信メッセージに助長性を持たせることで通信路で発生した誤りを訂正することが可能である。

3.2　(

２元対称通信路 (BSC)) ２元記号 $\{0,1\}$ が誤り率 $\varepsilon(0 \leq \varepsilon \leq 1)$ で”0”が”1”に誤り、”1”が”0”に誤る。

4　実験の装置あるいは実験手順

確率モデルで表現される通信路（２元対称通信路）を、乱数を用いて実装 rand() 関数、Mersenne Twister(MT) の２種類の擬似乱数生成器を用いる。

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

０以上１以下の乱数を発生させ、乱数が 0.5 以下ならば”0”、0.5 ¡ 乱数ならば”1”とし情報系列 w を生成 (ここの 0.5 はどうでもよく自分は ε を用いた)。

4.2 BSC でん雑音 $e = (e_1, e_2, ..., e_k)$ の発生と受信系列 $y = (y_1, y_2, ..., y_k)$ の生成

4.2.1 BSC での雑音 $e = (e_1, e_2, ..., e_k)$ の発生

０以上１以下の乱数を発生させ、乱数が ε 以下ならば”1”(誤りあり)、ε ¡ 乱数ならば”1”(誤りなし) とし雑音系列 e を生成

4.2.2 受信系列 $y = (y_1, y_2, ..., y_k))$ の生成

情報系列 w と雑音系列 e の各要素の排他的論理和

$$y_i - w_i \oplus e_i, i \in \{1, 2, ..., k\}$$

を計算

4.3 誤りビット数の算出

情報系列 w と受信系列 y の情報ビット w_i と受信ビット y_i が異なる数を数える。

4.4 ビット誤り率 P_e の計算

１３を十分な精度が得られるまで繰り返し、ビット誤り率 P_e を求める。P_e は誤ったビットの総数を送信したビットの総数で割ったものである。

4.5 １４を各 ε について実行する

5 結果

結果は以下のようになった。

表１　ε と P_{dec} の関係

ε	rand() の P_{dec}	MT の P_{dec}
0.000010	0.000000	0.000011
0.000020	0.000020	0.000021
0.000030	0.000028	0.000032
0.000040	0.000035	0.000041
0.000050	0.000050	0.000048
0.000060	0.000068	0.000052
0.000070	0.000045	0.000069
0.000080	0.000090	0.000080
0.000090	0.000085	0.000091
0.000100	0.000115	0.000104

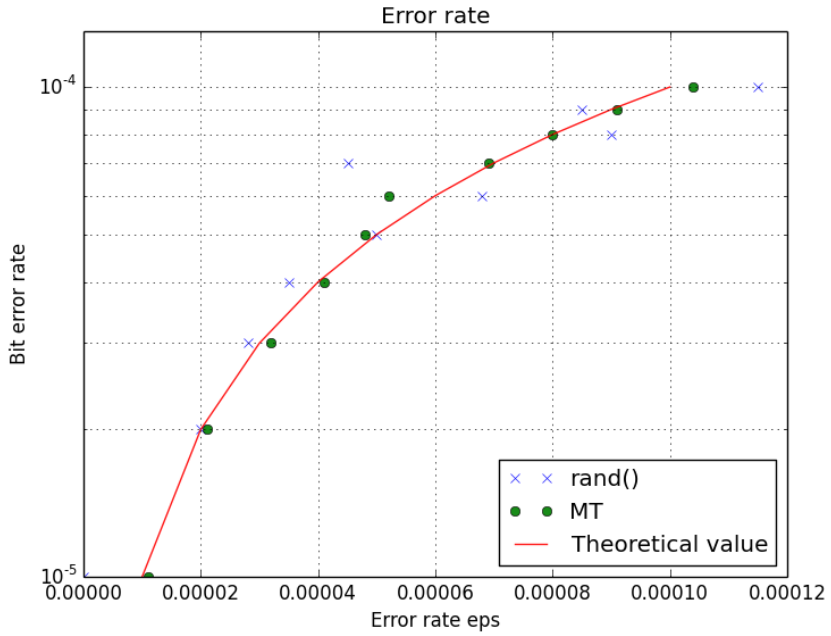

図 1　最急降下法が生成した二次元配列

6　検討・考察

7　検討事項

7.1　シュミレーションによる BER と理論値がほぼ同じ値にな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

1 何度か試行した結果 100 万回以上必要であることがわかる。

7.2　ε を非常に小さくした場合、検討事項 1-1 はどうなるか。

上ののグラフより誤り率が小さいほど理論値に近づいていることが見て取れるため試行回数は 100 万回よりも少なくて良いと考察される。

7.3　rand() と MT の違いは何か、検討事項 1-1 と 1-2 を絡めて考察せよ

MT は rand() よりも周期が大きいため回数を重ねると rand() よりも MT の方がより正確な乱数を生成する。このことより検定事項 1-1 ではシュミレーション回数を大きくしていった際より MT の方が理論値に近くと考察される。

8　結論

rand() は MT に比べ沢山の乱数を生成することに向いていないがコンピュータの記憶容量の消費は MT に比べ小さく少量の乱数生成に向いていると結論付けられる。

9　参考文献

参考文献

[1] やさしい C(第５版)・高橋麻奈

10　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　rand().c

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

int main(void) {
  double ran;

  srand(100);

  ran = rand();

  int w[4];
  int e[4];
  int y[4];

  int i;
  double xi = 0.00001;
  double rnd;

  int j;
  double all_err = 0;

  int k = 4;
```

```
int sim = 100000;
int t;


for (int j = 0; j < 10; j++) {
  for (int t = 0; t < sim; t++) {

    for (i = 0; i < k; i++) {
      rnd = rand();
      if (rnd / RAND_MAX <= xi) {
        w[i] = 1;
      }
      else {
        w[i] = 0;
      }
    }


    for (i = 0; i < k; i++) {
      rnd = rand();
      if (rnd / RAND_MAX <= xi) {
        e[i] = 1;
      }
      else {
        e[i] = 0;
      }
    }

    y[0] = w[0] ^ e[0];
    y[1] = w[1] ^ e[1];
    y[2] = w[2] ^ e[2];
    y[3] = w[3] ^ e[3];

    double err = 0;
    for (i = 0; i < k; i++) {
      if (y[i] != w[i]) {
        err++;
```

```
        }
      }
      all_err = all_err + err;
    }
    double p_e = all_err / (k * sim);
    printf("p_e = %f, xi = %f\n", p_e, xi);
    xi = xi + 0.00001;
    all_err = 0;
  }

  return 0;
}
```

ソースコード 2 MT.c

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

int main() {


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


  int w[4];
  int e[4];
  int y[4];

  int i;
  double xi = 0.00001;
  double rnd;

  int j;
  double all_err = 0;

  int k = 4;
  int sim = 1000000;
```

```
int t;


for(j = 0; j < 10; j++) {
  for (t = 0; t <sim; t++) {
    for (i = 0; i < k; i++) {
      rnd = r_rand(mt);
      if (rnd <= xi) {
        w[i] = 1;
      }
      else {
        w[i] = 0;
      }
    }
    for (i = 0; i < k; i++) {
      rnd = r_rand(mt);
      if (rnd <= xi) {
        e[i] = 1;
      }
      else {
        e[i] = 0;
      }
    }

    y[0] = w[0] ^ e[0];
    y[1] = w[1] ^ e[1];
    y[2] = w[2] ^ e[2];
    y[3] = w[3] ^ e[3];

    double err = 0;
    for (i = 0; i < k; i++) {
      if (y[i] != w[i]) {
        err++;
      }
    }
    all_err = all_err + err;
  }
```

```
    double p_e = all_err / (k * sim);
    printf("p_e = %f, xi = %f\n", p_e, xi);
    xi = xi + 0.00001;
    all_err = 0;
  }
  return 0;
}
```