疑似乱数についてのメモ

桂田 祐史

2003 年 8 月 12 日, 2012 年 4 月 4 日, 2017 年 4 月 29 日

古すぎて風味が落ちているどころではないのだけど…付録 A に手を入れたので。

目 次

1 はじめに

参考書として以下の二つをあげておく (古いです)。

1. G. E. Forsythe, M. A. Malcolm, C. B. Moler 著 (森 正武 訳); 計算機のための数値計算法, 日本コンピューター協会.
2. D. E. Knuth, The art of computer programming

最近 (2003 年 8 月現在)

> 『間違いだらけの疑似乱数選び』[1] PDF は http://www.soi.wide.ad.jp/class/20010000/slides/03/sfc2002.pdf にある。

を発見した。これを勉強する必要がありそう。メルセンヌ・ツイスターというアルゴリズムを創出した松本眞氏は IBM 科学賞を受賞している (http://www-6.ibm.com/jp/company/society/science/p13th/matsumoto.html を参照)。

…共立出版の bit が元気だった頃ならば、どんなに忙しくても見落とさなかったと思うのだが、どうも浦島太郎になっていたようだ。

[1] http://www.soi.wide.ad.jp/class/20010000/slides/03/1.html

2 UNIX システムに備わっている関数の利用

2.1 心構え

残念ながら、色々な検定にパスするまともな乱数を生成する関数が実際のコンピューター・システムに備わっていることは珍しい。しかし、試し用のデータ生成などには十分利用できる。

2.2 rand(), srand()

UNIX マシンならば、man 3 rand でオンライン・マニュアル読んでみよう。関数 rand() は 0 から RAND_MAX (これは stdlib.h に定義されている) までの整数値を取る、周期 2^{32} の数列の各項を順番に返す[2]。

乱数の種 (seed) を変更するには、 srand(seed); を実行すれば良い。

```
/*
 * test-rand.c
 */

/*
 * rand() は
 *  BSD では 0 から 2^{31}-1 までの範囲の疑似乱数を返す。
 *  SYSTEM V では 0 から 2^{15}-1 までの範囲の疑似乱数を返す。
 *
 */

#include <stdio.h>
#include <stdlib.h> /* rand(),srand() */

int main()
{
  int i, n;
  n = 10;
  printf("start\n");
  for (i = 0; i < n; i++)
    printf("%d\n", rand());
  /* seed を 3 にしてやり直し */
  srand(3);
  printf("start (seed=3)\n");
  for (i = 0; i < n; i++)
    printf("%d\n", rand());
  /* seed を 1 にしてやり直し (最初と同じになる) */
  srand(1);
  printf("start (seed=1)\n");
  for (i = 0; i < n; i++)
    printf("%d\n", rand());

  return 0;
}
```

ゲームのプログラミングなどでは、システムの時刻を読み取って、乱数の種となるデータを生成することがよく行なわれる。シミュレーションをする場合にもそれをしてよいだろうが、追試が出来るように種を記録しておくべきであろう。

[2] RAND_MAX は SYSTEM V では $2^{15}-1 = 32767$, BSD では $2^{31}-1 = 2147483647$ という値を取る。この食い違いは面倒だが…

```
/*
 * test-rand2.c
 */

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

#include <sys/types.h>
#include <sys/time.h>

int main()
{
  int i, n, myseed;
  time_t tloc;

  n = 10;
  time(&tloc);
  myseed = tloc;
  srand(myseed);
  printf("start (seed=%d)\n", myseed);
  for (i = 0; i < n; i++)
    printf("%d\n", rand());

  return 0;
}
```

種を time() で安直に設定するには、(tloc, myseed などを省略して)

```
srand((unsigned)time(NULL));
```

くらいで良いかも。

$[0, 1]$ に正規化したい場合は RAND_MAX で割算すればよいだろう[3]。

[3] RAND_MAX + 1 で割って、$[0, 1)$ に正規化するのが正しいという話を聞いた覚えもある。どちらが正しいか分かったら訂正する。

```
/*
 * test-rand3.c
 */

/*
 * rand() は 0 から RAND_MAX までの範囲の疑似乱数を返す。
 *  ゆえに割算すれば [0,1] の範囲の疑似乱数値を返す。
 */

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

double drand()
{
  return rand() / (double) RAND_MAX;
}

int main()
{
  int i, n;
  n = 10;
  printf("start\n");
  for (i = 0; i < n; i++)
    printf("%f\n", drand());
  return 0;
}
```

2.3 drand48(), lrand48(), srand48()

SunOS, FreeBSD には、48 ビット整数演算を利用した疑似乱数発生関数が備わっている。例えば drand48() は、double 型の、$[0, 1)$ 内の一様疑似乱数を返す。

```
/*
 * test-drand48.c
 */

/*
 * drand48() は [0,1) の範囲の疑似乱数を返す。
 */

#include <stdio.h>
#include <stdlib.h> /* drand48() */
#include <sys/types.h>
#include <sys/time.h>

double drand48();

int main()
{
  int i, n;
  time_t tloc;
  long int myseed;

  n = 10;
  printf("start\n");
  for (i = 0; i < n; i++)
    printf("%f\n", drand48());

  time(&tloc);
  myseed = tloc;
  srand48(myseed);
  printf("start (seed=%ld)\n", myseed);
  for (i = 0; i < n; i++)
    printf("%f\n", drand48());

  return 0;
}
```

2.4 random(), srandom()

SunOS や FreeBSD には rand(), srand() の改良版である random(), srandom() がある。値の範囲は 0 から $2^{31}-1$ で、周期はずっと長い。使い方は大体同じである。ただし、random() の型は long である。

```
/*
 * test-random.c
 */

/*
 * random() は
 *  0 から 2^{31}-1 までの範囲の疑似乱数を返す。
 *  man 3 random 参照。
 *  rand() よりも質の良い疑似乱数を返す。計算速度は 2/3.
 */

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

int main()
{
  int i, n;
  n = 10;
  printf("start\n");
  for (i = 0; i < n; i++)
    printf("%ld\n", random());
  srandom(3);
  printf("start (seed=3)\n");
  for (i = 0; i < n; i++)
    printf("%ld\n", random());
  srandom(1);
  printf("start (seed=1)\n");
  for (i = 0; i < n; i++)
    printf("%ld\n", random());
  return 0;
}
```

2.5 とりあえずのお勧め

```
/*
 * drandom.c --- [0,1] に一様に分布する疑似乱数
 *   ((double) INT_MAX) + 1 で割って [0,1) に分布するとすべきかも
 */

#include <stdlib.h>
#include <limits.h>

double drandom()
{
  return random() / (double) INT_MAX;
}
```

3 ある程度信用出来るプログラム例

(構想倒れ)

4　優良格子点法の話

(構想倒れ)

A　メルセンヌ・ツイスターをインストールしてみる

「Mersenne Twister Home Page」[4]

色々あるけれど、まずは本家の C プログラム mt19937ar.sep.tgz[5] を試してみた。

必要な関数の入っている `mt19937ar.c` をコンパイルして、オブジェクト・ファイルを作り、テスト・プログラム `mtTest.c` をコンパイル＆リンクし、実行結果を付属する `mt19937ar.out` と比較する。

```
cd $SOMEWHERE
tar xzf mt19937ar.sep.tgz
gcc -O -c mt19937ar.c
gcc -o mtTest mtTest.c mt19937ar.o
./mtTest > foo
diff foo mt19937ar.out
```

これで何も表示されなければ成功ということ。

インストールするには、少々強引だが、

```
ar cr libmt19937ar.a mt19937ar.o
ranlib libmt19937ar.a
sudo cp -p libmt19937ar.a /usr/local/lib
sudo cp -p mt19937ar.h /usr/local/include
```

これで C プログラムの中に

```
#include <mt19937ar.h>
```

として、リンクするときに `-lmt19937ar` というリンカー・オプションを指定する。

初期化は

```
  unsigned long init[4]={0x123, 0x234, 0x345, 0x456}, length=4;
  init_by_array(init, length);
```

のように unsigned long int の配列データで初期化するか、

```
  init_genrand((unsigned long)20120328);
```

のように unsigned long int 型データで初期化する。もちろん (`srand()` でやったように)

```
  init_genrand((unsigned long)time(NULL));
```

のように `time()` を使っても良いだろう。

- `unsigned long genrand_int32(void)` 0 から 0xffffffff まで

[4] `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/mt.html`

[5] `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/CODES/mt19937ar.sep.tgz`

- `long genrand_int31(void)` 0 から $2^{31}-1$ まで。
- `double genrand_real1(void)` $[0,1]$
- `double genrand_real2(void)` $[0,1)$
- `double genrand_real3(void)` $(0,1)$
- `double genrand_res53(void)` $[0,1)$

C++ では、新しい規格ではメルセンヌ・ツイスターが含まれているそうだが (もう g++ で使えるの？)、この C バージョンも利用可能である。

(1) C++ プログラム中に

```
extern "C" {
#include "mt19937ar.h"
};
```

のようにインクルードする。

(2) `mt19937ar.c` を C++ プログラムとして利用する。

```
cp -p mt199e7ar.c mt19937ar.cpp
g++ -O -c mt19937ar.cpp
```