MTToolBox で $\mathbb{F}_2$ 線形疑似乱数生成器を作る

斎藤睦夫（広島大学）

September 23, 2013

数学ソフトウェアとフリードキュメント XVII

This study is granted in part by JSPS Grant-In-Aid #21654004, #23244002, #24654019.

目次

1. はじめに
2. 疑似乱数生成器
   - 疑似乱数列
   - $\mathbb{F}_2$ 線形疑似乱数生成器
3. 周期
   - 出力列の周期
   - 最小多項式
   - 原始多項式
   - メルセンヌ素数
4. 均等分布性
   - 均等分布次元
   - v ビット精度均等分布次元
5. MTToolBox
   - MTToolBox の概要
   - サンプル

はじめに

Mersenne Twister(MT) [2](1998 松本, 西村) は広く使われている疑似乱数生成器である。

MTToolBox [3] は MT などの疑似乱数生成器の開発に使われた技術をツール化したものである。
MTToolBox のユーザーは、要望する

- メモリサイズ
- 品質
- 速度
- 並列性

に合った疑似乱数生成器を設計することができる。

以下では、疑似乱数生成器の開発上で使われる概念と技術を説明する。

疑似乱数列

真の乱数列 定義はあるが、そのまま応用するのは難しい。

物理乱数 物理現象を元にして作られた乱数列。

疑似乱数 確定的なアルゴリズムから作られた数列を乱数列とみなしたもの。

- 真の乱数ではない。
- 再現性がある。

- 暗号用疑似乱数。
- **シミュレーション用疑似乱数。**
  （モンテカルロ法用疑似乱数）

今日の話は、シミュレーション用疑似乱数について。

$\mathbb{F}_2$ 線形疑似乱数生成器

疑似乱数生成器は、無入力有限オートマトンである。

定義

無入力有限オートマトンは、

- 有限集合 S:状態空間, O:出力の集合
- 初期状態 $s_0 \in S$
- 状態遷移関数 $f : S \longrightarrow S$
- 出力関数 $o : S \longrightarrow O$

の組である。

疑似乱数生成器の状態遷移は、

$$s_0, s_1 = f(s_0), s_2 = f(s_1), \ldots$$

となり、出力列は、

$$o(s_1), o(s_2), \ldots$$

となる。

$\mathbb{F}_2$ 線形疑似乱数生成器

定義

- ビット $\{0,1\}$ を 2 元体 $\mathbb{F}_2$ と同一視する。
- w-ビット整数を $\mathbb{F}_2^w$ 上の横ベクトルと同一視する。(出力集合 O)
  w は 32, 64 または 128 など.
- 計算機のメモリ中の w-ビットの要素 N 個から成る配列を線形空間 $(\mathbb{F}_2^w)^N$ とみなす。(状態空間 S)

状態遷移関数と出力関数が $\mathbb{F}_2$ 線形写像なら、その疑似乱数生成器は $\mathbb{F}_2$ **線形疑似乱数生成器**と呼ぶことにする。

$\mathbb{F}_2$ 線形疑似乱数生成器

$\mathbb{F}_2$ 線形疑似乱数生成器には以下の性質がある。

- 長い周期であっても周期の計算が（比較的）簡単にできる。
- 均等分布次元（後述）の計算が可能。
- 高速に乱数生成が可能。
- **暗号用疑似乱数生成器としては、容易に解読される。**

これらの性質は、シミュレーション用疑似乱数生成器としては、望ましい。
以下では、$\mathbb{F}_2$ 線形疑似乱数生成器について説明する。

Mersenne Twister

$\mathbb{F}_2$ 線形疑似乱数生成器の例として Mersenne Twister を挙げる。
MT19937 の漸化式は $N = 624$ として、

$$\begin{aligned} \mathbf{w}_i &= g(\mathbf{w}_{i-N}, \ldots, \mathbf{w}_{i-1}) \\ &= g(\mathbf{w}_{i-N}, \mathbf{w}_{i-N+1}, \mathbf{w}_{i-N+M}) \\ &= (\mathbf{w}_{i-N} | \mathbf{w}_{i-N+1}) A + \mathbf{w}_{i-N+M}, \end{aligned}$$

ここで $(\mathbf{w}_{i-N}|\mathbf{w}_{i-N+1})$ は $\mathbf{w}_{i-N}$ の 1 ビット MSB と $\mathbf{w}_{i-N+1}$ の 31 ビットの LSB の結合を示し、A は (32×32) 行列で、$\mathbf{w}A$ が数回のビット演算で計算可能なものである。

状態空間は $(\mathbf{w}_{i-N}, \ldots, \mathbf{w}_{i-1})^*$, 初期状態は $(\mathbf{w}_0, \ldots, \mathbf{w}_{N-1})^\dagger$。

---

*$\mathbf{w}_{i-N}$ は MSB の 1 ビットのみ
†$\mathbf{w}_0$ は MSB の 1 ビットのみ

出力列の周期

初期状態 s_0 を固定した疑似乱数生成器の出力列を $\mathbf{x}_i$ で表すと、状態空間が有限であることから、出力列は必ず繰り返し部分を持つ、このことを出力列は**周期的**であるという。

$$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n_0}, \mathbf{x}_{n_0+1}, \ldots, \mathbf{x}_{n_0+p} = \mathbf{x}_{n_0}, \mathbf{x}_{n_0+1}, \cdots$$

$n \geq n_0$ となるすべての n について $\mathbf{x}_{n+p} = \mathbf{x}_n$ を満たす最小の $p > 0$ を出力列の**周期**という。
$n_0 = 1$ の時は、すべての m について、$\mathbf{x}_{m+p} = \mathbf{x}_m$ が成り立つ。
このとき出力列は**純周期的**という。

$\mathbb{F}_2$ 線形疑似乱数生成器の周期の性質

- $\mathbb{F}_2$ 線形疑似乱数生成器の周期は、状態空間の要素数-1 を越えない。（状態空間が n ビットなら、$2^n - 1$ を越えない）
- $\mathbb{F}_2$ 線形疑似乱数生成器の周期が状態空間の要素数-1 のとき、最大周期という。
- 最大周期なら、純周期的である。

最小多項式

$\mathbb{F}_2$ 線形疑似乱数生成器の出力関数として、1 ビット出力関数を選んだとする。そして出力列を $x_0, x_1, x_2, \ldots, (x_i \in \mathbb{F}_2)$ とすると

$$x_a = c_1 x_{a-1} + c_2 x_{a-2} + \cdots + c_k x_{a-k}$$

がすべての a について成り立つような $c_1, \ldots, c_k \in \mathbb{F}_2$ の最小の列が存在する。このとき、

$$P(t) = t^k + c_1 t^{k-1} + \cdots + c_{k-1} t + c_k$$

を出力列 x_i の**最小多項式**という。

最小多項式の性質

- 最小多項式 $P(t)$ は、初期状態 s_0 および出力関数に依存する。
- 最小多項式 $P(t)$ の次数は、状態空間の次元を越えない。

原始多項式

$Q(t) \in \mathbb{F}_2[t]$ が原始的とは、$Q(t)$ が既約かつ $t \in \mathbb{F}_2[t]/Q(t)$ が乗法群 $(\mathbb{F}_2[t]/Q(t))^\times$ を生成することである。

最小多項式と原始多項式の関係

最小多項式 $P(t)$ が原始多項式なら、出力列の周期 p は

$$p = 2^{\deg(P(t))} - 1$$

となる。
さらに、$\deg(P(t)) = \dim(S)$ なら、周期は $s_0 (\neq \mathbf{0})$ の選び方によらない。

最小多項式の求め方と原始性の判定法

最小多項式を求めるアルゴリズムに Berlekamp-Massey 法 (1967 Berlekamp, 1969 Massey) がある。

入力 最小多項式の次数の二倍以上の長さの数列。

アルゴリズム 多項式版ユークリッドの互除法。

効率 効率よく計算可能。

$Q(t)$ が原始多項式であることの判定法

既約判定 整数における素数判定と比べて効率的に計算が可能。

t が乗法群の生成元か $2^{deg(Q(t))} - 1$ の**素因数分解**が必要。

素因数分解が出来ていれば、効率的に計算が可能。

メルセンヌ素数

- $2^m - 1$ で表される数をメルセンヌ数という。
- メルセンヌ数が素数の時、**メルセンヌ素数**という。
- メルセンヌ素数の指数部 m を**メルセンヌ指数**ということにする。
- メルセンヌ素数は GIMPS プロジェクトにより大きなものが求められている。

メルセンヌ素数の利用

状態空間の大きさをメルセンヌ指数 m にすると、

- 最小多項式の次数は最大で m になる
- **最小多項式の次数が m で既約なら、周期は $2^m - 1$ になる。**
- ある周期が $2^m - 1$ なら、周期は（**0** 以外の）初期状態に依存しない。

状態空間の大きさをメルセンヌ指数 m にし、最小多項式が既約で次数が m となるような $\mathbb{F}_2$ 線形疑似乱数生成器を作れば、周期が最大で初期値によらない。

MTToolBoxによるサポート

MTToolBoxは、利用者の設計した$\mathbb{F}_2$線形疑似乱数生成器に対して、

- 最小多項式の計算
- 最小多項式の既約性判定
- 原始性の判定
- 最大周期となるような状態遷移関数の探索

をサポートする。

均等分布次元

疑似乱数生成器の出力が最大周期になっている場合、w ビットの出力列 $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots$ について、周期 p の 1 周期分について、連続する k 個を 1 組にした列と k 個の w ビットの $\mathbf{0}$ からなる和集合を考える。

$$(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k), (\mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_{k+1}), \ldots, (\mathbf{x}_p, \mathbf{x}_1, \ldots, \mathbf{x}_{k-1}), (\underbrace{\mathbf{0}, \mathbf{0}, \ldots, \mathbf{0}}_{k \text{ 個}})$$

こうして得られる $p+1$ 個の wk ビットがすべての wk ビットパターンを尽くすような**最大の k を w ビット均等分布次元**といい、このとき疑似乱数生成器の出力は、**k 次元 w ビット均等分布する**という。

この性質が成り立つとき、線形写像の性質より、wk ビットパターンはどれも $(p+1)$ 個の中で同じ回数ずつ出現する。
出力列については、全部 0 というビットパターンは 1 回少なく出現する。

v ビット精度均等分布次元

疑似乱数生成器の出力の均等性を上位ビットについてみることが実用上重要である。

w ビットの出力列の上位 v ビットだけを切り出した出力列についての均等分布の次元 k を **v ビット精度均等分布次元**といい、$k(v)$ で表す。このとき疑似乱数生成器の出力は、**v ビット精度で $k(v)$ 次元均等分布する**という。

$\mathbb{F}_2$ 線形疑似乱数生成器の周期を $2^m - 1$ とすると、$k(v) \leq \lfloor m/v \rfloor$ という理論的上限がある。理論的上限との差を $d(v)$, $d(v)$ の $1 \leq v \leq w$ に渡る和を Δ とする。

$$d(v) = \lfloor m/v \rfloor - k(v)$$

$$\Delta = \sum_{v=1}^{w} d(v)$$

v ビット精度均等分布次元の例

v ビット精度均等分布次元の例として、Mersenne Twiseter[2](1998 松本, 西村) のひとつ MT19937 の v ビット精度均等分布次元の一部を挙げる。

v	$k(v)$	$d(v)$	$\cdots$	v	$k(v)$	$d(v)$
1	19937	0		25	623	174
2	9968	0		26	623	143
3	6240	405		27	623	115
4	4984	0	$\cdots$	28	623	89
5	3738	249		29	623	64
6	3115	207		30	623	41
7	2493	355		31	623	20
8	2492	0		32	623	0

$$\Delta = 6750$$

v ビット精度均等分布次元の計算

- 均等分布次元の計算は、線形写像のランクを計算すればよい。しかし、状態空間 S が大きい場合、ランクの計算ですら困難になる。
- PIS 法 [1](2011 原瀬) を用いると、高速に計算できる。 ($O(2w\dim(S)^2)$)
- v ビット精度均等分布次元は、出力関数 o を変えることによって、向上させることが出来る。
- 出力関数 o を変更するパラメータをテンパリングパラメータという。

MTToolBox によるサポート

- MTToolBox は PIS 法による v ビット精度均等分布次元の計算をサポートし、それを改善する。
- MTToolBox はテンパリングパラメータの探索をサポートする。

MTToolBox の概要

MTToolBox は

- $\mathbb{F}_2$ 線形疑似乱数生成器を開発するためのツールである。
- C++ で書かれている。
- 周期の計算をサポートする。
- 最大周期となるパラメータの探索をサポートする。
- 均等分布次元の計算をサポートする。
- テンパリングパラメータの探索をサポートする。

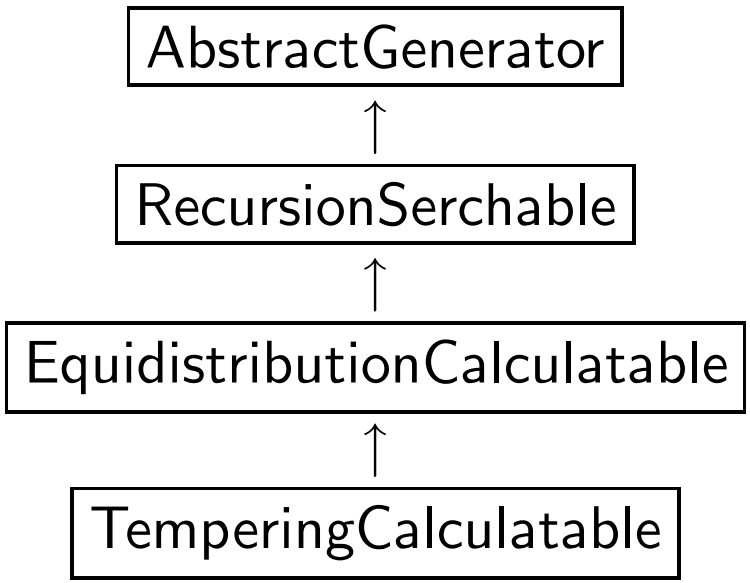
また、

- SFMT, MTGP, TinyMT[‡] の開発に使用したツールを整理したもの。
- MTGP, TinyMT の Dynamic Creator（パラメータ自動生成プログラム）をサンプルに含む。

[‡]SFMT: SIMD 用 MT, MTGP:GPU 用 MT, TinyMT:小型 MT

MTToolBox の概要

疑似乱数生成器のクラス階層（抽象クラス）



周期の計算が可能

状態遷移関数のパラメータの探索が可能

均等分布次元の計算が可能

テンパリングパラメータの計算が可能

MTToolBox の利用者は上記クラスのサブクラスを作成することによって、該当する計算が可能になる。

MTToolBoxの概要

アルゴリズム

- AlgorithmRecursionSearch 状態遷移パラメータの探索
- AlgorithmEquidistribution 均等分布次元の計算
- AlgorithmRecursionAndTempering 状態遷移パラメータとテンパリングパラメータの探索

原始多項式判定アルゴリズム

- AlgorithmIrreducible $\mathbb{F}_2$ 多項式の既約判定
- AlgorithmPrimitive $\mathbb{F}_2$ 多項式の原始性判定

テンパリングパラメータ探索アルゴリズム

- AlgorithmPartialBitPattern TinyMT Dynamic Creator で使用しているテンパリングパラメータ探索
- AlgorithmBestBits MT Dynamic Creator で使用しているテンパリングパラメータ探索

サンプル

XorShift 128 の周期を求める (XorShift クラス)

```
 1  class XorShift : public AbstractGenerator<uint32_t> {
 2  public:
 3      XorShift(uint32_t v) { seed(v); }
 4      uint32_t generate() {
 5          uint32_t t = (x ^ (x << a));
 6          x = y; y = z; z = w;
 7          return w = (w ^ (w >> c)) ^ (t ^ (t >> b));
 8      }
 9      void seed(uint32_t v) {
10          w = ~v;
11          x = y = z = v;
12      }
13      int bitSize() const { return 128; }
14  private:
15      enum {a = 5, b = 14, c = 1};
16      uint32_t x, y, z, w;
17  };
```

サンプル

XorShift 128 の周期を求める (main)

```
18 int main() {
19     XorShift xs(1);
20     GF2X poly;
21     minpoly<uint32_t>(poly, xs);
22     cout << "degree = " << deg(poly) << endl;
23     // 2^128 - 1 の素因子
24     const char * factors128_1[] = {
25         "3", "5", "17", "257", "641", "65537", "274177", "6700417",
26         "67280421310721", NULL};
27     if (isPrime(poly, 128, factors128_1)) {
28         cout << "period is 2^128 -1." << endl;
29     } else {
30         cout << "period is unknown." << endl;
31     }
32     return 0;
33 }
```

サンプル

XorShift 128 の v ビット精度均等分布次元を求める (XorShift クラス)

```
1  class XorShift : public EquidistributionCalculatable<uint32_t> {
2  public:
3      XorShift(uint32_t v) { seed(v); }
4      XorShift(const XorShift& src) :
5          EquidistributionCalculatable<uint32_t>() {
6          x = src.x; y = src.y; z = src.z; w = src.w;
7      }
8      XorShift * clone() const { return new XorShift(*this);}
9      uint32_t generate();// 省略
10     uint32_t generate(int outBitLen) {
11         uint32_t mask = 0;
12         mask = (~mask) << (32 - outBitLen);
13         return generate() & mask;
14     }
15     bool isZero() const { return x == 0 && y == 0 &&
16             z == 0 && w == 0; }
17     void setZero() { x = y = z = w = 0; }
```

サンプル

XorShift 128 の v ビット精度均等分布次元を求める (XorShift クラス)

```
18      void add(EquidistributionCalculatable<uint32_t>& other) {
19          XorShift* that = dynamic_cast<XorShift *>(&other);
20          if (that == 0) {
21              throw std::invalid_argument(
22                  "the␣adder␣should␣have␣the␣same␣type␣as␣the␣
                        addee.");
23          }
24          x ^= that->x; y ^= that->y; z ^= that->z; w ^= that->w;
25      }
26      void seed(uint32_t v);
27      int bitSize() const { return 128; }
28      void setUpParam(AbstractGenerator<uint32_t>& generator){}
29      const std::string getHeaderString(){return ""; }
30      const std::string getParamString(){return ""; }
31  private:
32      enum {a = 5, b = 14, c = 1};
33      uint32_t x, y, z, w;
34  };
```

サンプル

XorShift 128 の v ビット精度均等分布次元を求める (main)

```
35  int main() {
36      XorShift xs(1);
37      AlgorithmEquidistribution<uint32_t> eq(xs, 32);
38      int veq[32];
39      int delta = eq.get_all_equidist(veq);
40      int bitSize = xs.bitSize();
41      for (int i = 0; i < 32; i++) {
42          cout << "k(" << dec << setw(2) << (i + 1) << "):"
43              << setw(3) << veq[i] << "␣␣d(" << setw(2) << (i + 1)
                    << "):"
44              << setw(3) << (bitSize / (i + 1)) - veq[i] << endl;
45      }
46      cout << "delta:" << delta << endl;
47      return 0;
48  }
```

省略なしのサンプルコードは MTTools の samples/XORSHIFT ディレクトリにある。

まとめ

$\mathbb{F}_2$ 線形疑似乱数生成器の重要な性質

- 出力列の周期は初期状態（$s_0 \in S$）に依存する。
- 最小多項式の次数が $m = \dim(S)$ かつ原始多項式なら最大周期 $2^m - 1$ を達成する。
  このとき、
  - 周期などの性質は、**0** 以外の初期状態に依存しない。
  - v ビット精度均等分布次元の計算が可能となる。

MTToolBox は

- 最大周期となる状態遷移パラメータの探索をサポートする。
- v ビット精度均等分布次元を改善するテンパリングパラメータの探索をサポートする。

ご清聴ありがとうございました。

参考文献

S. Harase.
An efficient lattice reduction method for $\mathbf{F}_2$-linear pseudorandom number generators using Mulders and Storjohann algorithm.
Journal of Computational and Applied Mathematics, Submitted, January.

M. Matsumoto and T. Nishimura.
Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator.
ACM Trans. on Modeling and Computer Simulation, 8(1):3–30, January 1998.
http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html.

M. Saito and M. Matsumoto.
MTToolBox, 2013.
https://github.com/MSaito/MTToolBox.