松 山 大 学 論 集
第 34 巻 第 3 号 抜 刷
2022 年 8 月 発 行

擬似乱数生成器（PRNG）のアルゴリズムに関する理論および実装の総合的研究

檀　　裕　也

擬似乱数生成器（PRNG）のアルゴリズムに関する理論および実装の総合的研究

檀　　裕　也

1　はじめに

擬似乱数生成器（Pseudo-Random Number Generator）とは，情報処理の分野でコンピュータが一定の規則に従ってビット列を生成するアルゴリズムである。そのビット列が，一定の規則とは裏腹に，乱数のような統計性を有すると，モンテカルロ法に基づくシミュレーションなど確率的現象の理解に大きな役割を果たす。これまでに，モンテカルロ法によるシミュレーションとして，研究室配属モデル[1]，予測の精緻化に関する数理モデル[2]，ソーシャルネットワーク空間における情報拡散の現象[3,4]およびファイル共有システムの現象[5]について明らかにしてきた。また，量子乱数を用いた研究[6]では，物理乱数としての統計的な性質について量子デバイスを使って検証した。本研究は，これら既存の研究の基礎として利用された擬似乱数生成器のアルゴリズムに関する理論について，最近の IoT や GPU など並列処理の装置を用いて実用化されることを視野に性能を明らかにするものである。

よく知られている擬似乱数生成器（PRNG）のアルゴリズムについて，線形代数によるフレームワークを通じて理解し，新しい PRNG を提案することが研究の目的である。具体的には，CPU 処理時間とメモリ消費量の観点から，IoT や GPU を用いた並列計算による効率化に至るまで，進化したハードウェア性能を踏まえた効率の良い実装を目指して，新しい PRNG の有効性および統計的性質を示したい。

従来の代表的なPRNGのアルゴリズムについて，線形代数の理論で統一的に処理できることを示し，行列の次数や固有値・固有ベクトルなど線形代数のツールが有効に機能することを示した上で，行列演算をGPUで処理することによって処理の高速化を図ることが本研究の学術的な特色・独創的な点である。逆に，性質の良い行列を構成することで，統計的な性質の良い擬似乱数を生成することが可能になると考えられる。新しいPRNGは，擬似乱数の次元や周期において優位性を示すことができれば，その応用として個別具体的なモンテカルロ法に基づくシミュレーションを効果的に実行することが可能となる。

これまでに，擬似乱数生成器として，線形合同法[7]をはじめ，メルセンヌ・ツイスター[8]やXorshift[9]などのアルゴリズムが知られている。これらは，Google Chromeなどの Webブラウザで実装されるなど情報社会において大きな役割を果たしている。その一方で，マルチコアを意識しつつもCPU内部のレジスタ処理を前提とした設計のため，省電力であるIoT機器や大規模行列計算が得意なGPUによる実装は想定されていない。本研究は，省電力であるIoT機器や大規模並列計算が得意なGPU上に実装可能なPRNGアルゴリズムを提案するものである。

なお，本研究は2020年度に交付を受けた松山大学特別研究助成による成果の一部である。その波及効果の一つとして，エルゴード性を利用した無限周期の擬似乱数生成器について2021年度卒業論文[10]にも取り組むことができ，情報処理学会の全国大会において研究の成果[11]を発表した。

2 擬似乱数生成器

コインを投げて表や裏のうちどちらかが決まる現象やサイコロを振って1から6までの整数の目が出る現象は，ともに予測不可能な確率現象である。

細工のされていない完全なコイン投げの場合，表か裏かという結果の予言はできないものの，表が出る確率と裏が出る確率は，ともに1/2である。コイ

ン投げの試行を繰り返したとき，その回数を増やすごとに統計的な性質が現れるものの，各事象は独立であって，次にどちらの面が出るのか予測することは不可能である。

また，不正のない完全なサイコロを使って目を出すと，どの目が出るのかという予言はできないが，１から６までの整数それぞれが出現する確率はいずれも１/６である。コイン投げの現象と同様に，試行回数を増やすと統計的な性質が現れるものの，各事象は独立であって，次にどの目が出るのか予測できない。

このような現象は，予測不可能な数列を生み出すという観点から，乱数の日常的で分かりやすい事例である。何も規則を持たない数列としての乱数列は，数値積分の近似値やモンテカルロ法によるシミュレーションなどコンピュータを用いた情報処理において極めて有用な道具である。統計学の教科書では，今でも付録として乱数表を掲載しているものもある。

しかし，大規模なデータを高速で処理するシステムにおいて，コインを投げたりサイコロを振ったりして乱数を生成することは現実的ではない。そこで，放射線や半導体などの物理現象によって発生するノイズからハードウェア的に乱数を生成することができる。例えば，放射線源から放射されるガンマ線は各事象が独立であって，その頻度はポアソン分布に従うことが知られている。すなわち，放射性核種の半減期という統計的な性質は知られているものの，いつガンマ線が放出されるか知る手掛かりは存在しない。このような乱数を物理乱数と呼ぶ。

現在のコンピュータシステムで物理乱数を用いるとき，半導体の電気的なノイズが用いられることが多い。

一方，ハードウェアから乱数を取り込むことは，デバイスに依存することを意味するため，ソフトウェア的に乱数が発生できないかと考えるのは自然である。そこで，擬似的な乱数列を発生するアルゴリズムが開発されてきた。そのようなものを擬似乱数生成器と呼ぶ。厳密に考えると，擬似乱数は，アルゴリ

ズムに従って生成されるため規則があって，次の値を予測できるものである。しかし，ある程度の乱数性を備えていれば，数値積分の近似値やモンテカルロ法によるシミュレーションなどコンピュータを用いた情報処理に問題のない精度で実行できることから，広く使われている。

本章では，Lehmer によって 1951 年に導入された線形合同法[7] をはじめ，Matsumoto および Nishimura によって 1998 年に開発されたメルセンヌ・ツイスター[8] や Marsaglia によって 2003 年に登場した Xorshift[9] などのアルゴリズムについて紹介し，エルゴード乱数を提案する。

2.1 線形合同法

線形合同法（Linear Congruential Generator）とは，合同式を用いて周期的な擬似乱数を生成するアルゴリズムである。合同式とは，法 M に対して，

$$a = b (\mathrm{mod}.\, M)$$

について，ある整数 n が存在して

$$a - b = nM$$

と書ける関係である。

一般に，初期値 $X_0 = (x_0, x_1, \cdots, x_{k-1})$ に対して，漸化式

$$x_{n+1} = a_k x_n + a_{k-1} x_{n-1} + \cdots + a_1 x_{n-k+1} + a_0 \ (\mathrm{mod}.\, M)$$

によって定義される擬似乱数の無限数列である。ここで，係数 $(a_0, a_1, \cdots, a_{k-1})$ および合同式の法 M は整数であって，条件に応じて擬似乱数の性質が定まる。

例えば，漸化式

$$x_{n+1} = a_0 x_n \ (\mathrm{mod}.\, M)$$

に対して，$M=2^{\beta}$ および $(a_0,\ 2^{\beta})=1$ を仮定すると，$a_0=3,\ 5\ (\mathrm{mod.}\ 8)$ のとき数列 $\{x_n\}$ の周期は $2^{\beta-2}$ である。

また，漸化式

$$x_{n+1}=a_1x_n+a_0\ (\mathrm{mod.}\ M)$$

に対して，$M=2^{\beta}$ を仮定すると，$a_1=1\ (\mathrm{mod.}\ 4)$ および $a_0=1\ (\mathrm{mod.}\ 2)$ のとき数列 $\{x_n\}$ の周期は 2^{β} である。

これらの性質を含めて，一般の漸化式についてよく知られている性質は Jansson[12] などで証明されている。

Lehmer によって 1951 年に導入された線形合同法[7] は古くからモンテカルロ・シミュレーションなどの計算機実験に用いられてきた。最近では，O'Neill[13] など線形合同法の改良されたアルゴリズムが提案されている。

2.2 メルセンヌ・ツイスター

Matsumoto および Nishimura によって 1998 年に開発されたメルセンヌ・ツイスター[8] は，$GF(2)$ の元を要素とする $\omega\times\omega$ の行列 A および B を用いて

$$x_{n+p}=x_q+x_{n+1}A+x_nB\ (\mathrm{mod.}\ M)$$

によって定義される擬似乱数の無限列である。ここで，$x_n\in GF(2)^{\omega}$ であって，$p>q>0$ を仮定する。そのとき，最大周期は $2^{p\omega}-1$ となることが証明されている。特に，周期 $2^{19937}-1$ の MT19937 は広く用いられていて，さまざまな処理系において実装されている。このような長い周期だけでなく，623 次元という高次元に均等分布する性質は線形合同法の弱点を改良するものであって，実用上十分な統計的な乱数性を持っていると考えられている。

2.3 Xorshift

Marsaglia によって 2003 年に登場した Xorshift[9] のアルゴリズムは，排他的

論理和とビットシフトのみの演算で実装できるため，高速に出力することが可能である。

2.4 エルゴード乱数

1997年に杉田洋によって提示された無理数回転を利用した擬似乱数生成法[14]によると，漸化式

$$x_{n+1} = x_n + \alpha \ (\mathrm{mod.}\ 1)$$

によって，区間$[0,\ 1)=\{x \in R;\ 0 \leqq x < 1\}$の擬似乱数系列を得るアルゴリズムである。ここで，法1と取るのは，小数点以下の部分が等しいことを意味する。例えば，

$$\alpha = \frac{1+\sqrt{5}}{2} = 1.618\cdots$$

と黄金比を取ると，比較的性質の良い乱数性が得られることが知られている。

3 代数的拡大体によるエルゴード乱数の実装

有理数体Qに無理数$\sqrt{5}$を添加した拡大体$Q(\sqrt{5})$について考える。その定義は，

$$Q(\sqrt{5}) = \left\{ \frac{a}{b} + \frac{c}{d}\sqrt{5};\ a,\ b,\ c,\ d \in Z,\ bd \neq 0 \right\}$$

であって，代数拡大体$Q(\sqrt{5})$の任意の要素xは，4つの整数$a,\ b,\ c,\ d \in Z$（ただし$bd \neq 0$）を用いて，

$$x = \frac{a}{b} + \frac{c}{d}\sqrt{5}$$

と書くことができる。エルゴード乱数は，適当な初期値x_0に対して，漸化式

$$x_{n+1} = x_n + \frac{1+\sqrt{5}}{2} \ (\mathrm{mod.}\ 1)$$

によって定まることから，

$$\begin{aligned} x_{n+1} &= x_n + \frac{1+\sqrt{5}}{2} \\ &= \frac{a}{b} + \frac{c}{d}\sqrt{5} + \frac{1+\sqrt{5}}{2} \\ &= \left(\frac{a}{b} + \frac{1}{2}\right) + \left(\frac{c}{d} + \frac{1}{2}\right)\sqrt{5} \\ &= \frac{2a+b}{2b} + \frac{2c+d}{2d}\sqrt{5} \end{aligned}$$

のように計算すると，４つの整数 $a,\ b,\ c,\ d \in Z$（ただし $bd \neq 0$）に関する漸化式として，次の結果が得られる：

$$\begin{cases} a_{n+1} = 2a_n - b_n \\ b_{n+1} = 2b_n \\ c_{n+1} = 2c_n - d_n \\ d_{n+1} = 2d_n \end{cases}$$

以上のアルゴリズムを踏まえ，次の C 言語ソースコードとして実装した。

C 言語ソースコード（抜粋）

```
#define SQRT5 2.23606797749978969

int a = 0, b = 1, c = 0, d = 1;

double ergodic(void)
{
    a = ( a << 1 ) + b;
    b <<= 1;
    while( !(a % 2) && !(b % 2) )
        a >>= 1, b >>= 1;
    c = (c << 1) + d;
    d <<= 1;
    while( !(c % 2) && !(d % 2) )
        c >>= 1, d >>= 1;
```

```
    x = (double)a / (double)b + (double)c / (double)d * SQRT5;
    return x - (int)x;
}
```

なお，プログラミング言語 Haskell による実装を含めて，周期のないエルゴード乱数に関する詳細については 2021 年度松山大学経営学部情報コース卒業論文[10] に委ねる。

4 乱数性の統計的検定

アルゴリズムによって生成された擬似乱数列は，一定の規則に従って計算されているとはいえ，乱数として先読みできないような性質を持っていなければならない。そのような性質について定量的に調べるために，いくつかの統計的検定セットが提案されている。

例えば，Knuth[15] は，頻度検定，系列検定（2 次元度数検定），間隔検定，ポーカー検定，札集め検定，順列検定，連の検定，t 個の数の最大値検定，衝突検定，系列相関検定，部分数列に関する検定，およびスペクトル検定の 12 種類を示した。

また，Marsaglia による DIEHARD[16] では，バースデイ空間検定，OPERM5 検定，（31×31）の 2 値行列ランク検定，（32×32）の 2 値行列ランク検定，（6×8）の 2 値行列ランク検定，ビット列検定，OPSO 検定，OQSO 検定，DNA 検定，8 ビット中の文字数検定，特定位置の 8 ビット中の文字数検定，駐車場検定，最小距離検定，3D SPHERES 検定，スクイーズ検定，重なりのある和検定，連の検定，およびクラップス検定の 18 種類を示した。

本章では，一般に最も普及している乱数性の統計的検定法として，NIST Special Publication 800-22[17, 18] を紹介する。この検定法は，1 次元度数検定，ブロック単位の頻度検定，連の検定，ブロック単位の最長連検定，2 値行列ランク検定，離散フーリエ変換検定，重なりのないテンプレート適合検定，重なりのあるテンプレート適合検定，Maurer のユニバーサル統計検定，線形複雑

度検定，系列検定，近似エントロピー検定，累積和検定，ランダム偏差検定，および種々のランダム偏差検定の 15 種類が提案されている。

その他の乱数性の検定については，情報処理振興事業協会セキュリティセンターによる網羅的な調査結果[19] を見よ。

4.1 一様性の検定（monobit_test）

ビット列 $Z=\{Z_1, Z_2, \cdots, Z_n\}\in\{0, 1\}^n$ に対し，0 と 1 が同じ頻度で出現することについて統計的に検定するものである。Z に対する検定統計量

$$T_n=\sum_{j=1}^{n}\frac{2Z_j-1}{\sqrt{n}}$$

は，n が十分大きいと，中心極限定理によって平均値 0 および分散 1 の正規分布 $N(0, 1)$ に近似することができる。そこで，p 値について，両側検定によって $p<0.005$ または $p>0.995$ となったときに 0 と 1 の一様性に関する帰無仮説を棄却する。

4.2 ブロック内一様性の検定（frequency_within_block_test）

ビット列 $Z=\{Z_1, Z_2, \cdots, Z_n\}\in\{0, 1\}^n$ に対し，M ビットずつで区切ったときに，各ブロック内で 0 と 1 が同じ頻度で出現することについて統計的に検定するものである。4.1 節は $M=n$ の特別な場合であると見做せる。

Z に対する検定統計量

$$T_n=4M\sum_{k=1}^{\left[\frac{n}{M}\right]}\left(\sum_{j=1}^{M}\frac{Z_{(k-1)M+j}}{\sqrt{n}}-\frac{1}{2}\right)^2$$

は，n および M が十分大きいと，中心極限定理によって分布が定まる。そこで，p 値について，片側検定によって $p<0.01$ となったときに 0 と 1 のブロッ

ク内一様性に関する帰無仮説を棄却する。

4.3 連による検定（runs_test）

ビット列 $Z=\{Z_1, Z_2, \cdots, Z_n\}\in\{0, 1\}^n$ のうち，0または1の連続した出現頻度について統計的に検定するものである。まず，$Z_j = Z_{j+1}$ のとき $U_j = 0$，および $Z_j \neq Z_{j+1}$ のとき $U_j = 1$ と置いて，統計量

$$Q_n = \sum_{j=1}^{n-1} U_j + 1$$

を作って，Z に対する検定統計量

$$T_n = \frac{Q_n - 2np_n(1-p_n)}{2\sqrt{2n}p_n(1-p_n)}$$

は，n が十分大きいと，中心極限定理によって平均値0および分散1の正規分布 $N(0, 1)$ に近似することができる。ただし，

$$p_n = \frac{1}{n}\sum_{j=1}^{n} Z_j$$

である。そこで，p 値について，両側検定によって棄却点 d を超えたときに連に関する帰無仮説を棄却する。

4.4 ブロック単位の最長連に関する検定（longest_run_ones_in_a_block_test）

ビット列 $Z=\{Z_1, Z_2, \cdots, Z_n\}\in\{0, 1\}^n$ に対し，M ビットずつで区切ったときに，各ブロック内で0または1の連続した出現頻度について，その最長連の出現回数に着目して統計的に検定するものである。

4.5 2値行列の階数を用いた検定（binary_matrix_rank_test）

ビット列 $Z=\{Z_1, Z_2, \cdots, Z_n\}\in\{0, 1\}^n$ から J^2 ずつの部分列で J 次正方行列を生成し，その階数の分布について統計的に検定するものである。$[n/J^2]$ 個の行列で，階数 J，階数 J-1，およびそれ未満のものの個数を求め，行列の階数に関する帰無仮説に対して，自由度2のカイ二乗適合度検定を行う。

4.6 離散フーリエ変換による検定（dft_test）

ビット列 $Z=\{Z_1, Z_2, \cdots, Z_n\}\in\{0, 1\}^n$ の出現パターンに強い周期的な成分が入っていないかについて統計的に検定するものである。周波数領域 $[0, 1]$ を n 等分した周波数点 $\lambda_\ell=\ell/n\,(\ell=1, 2, \cdots, n-1)$ における離散フーリエ変換

$$V_n(\lambda_\ell)=\sum_{j=1}^{n}(2Z_j-1)\exp[2\pi i(j-1)\lambda_\ell]$$

の実数成分と虚数成分に分けて，

$$\mathrm{Re}[V_n(\lambda_\ell)]=\sum_{j=1}^{n}(2Z_j-1)\cos[2\pi i(j-1)\lambda_\ell]$$

$$\mathrm{Im}[V_n(\lambda_\ell)]=\sum_{j=1}^{n}(2Z_j-1)\sin[2\pi i(j-1)\lambda_\ell]$$

を考える。ただし，$i=\sqrt{-1}$ は虚数単位である。このとき，検定統計量

$$\frac{\mathrm{Re}[V_n(\lambda_\ell)]}{\sqrt{n/2}} \quad \text{および} \quad \frac{\mathrm{Im}[V_n(\lambda_\ell)]}{\sqrt{n/2}}$$

は，$\lambda_\ell\neq 0, 1/2$ のとき，n が十分大きいと，中心極限定理によって平均値0および分散1の正規分布 $N(0, 1)$ にそれぞれ独立に近似することができる。

4.7 重複のないテンプレート適合に関する検定 (non_overlapping_template_matching_test)

ビット列 $Z=\{Z_1, Z_2, \cdots, Z_n\}\in\{0, 1\}^n$ のうち，特定の出現パターン $B=\{b_1, b_2, \cdots, b_m\}\in\{0, 1\}^m$ について統計的に検定するものである。部分列 B のことをテンプレートという。ビット列 Z のうち，$j=1$ から $j=m$ までの部分列について一致しているかどうか調べ，一致していなければ $j=2$ から始まる部分列について調べる。もし，一致していたら，重複は避けて，次は $j=m+1$ から始まる部分列について調べる。

4.8 重複のあるテンプレート適合に関する検定 (overlapping_template_matching_test)

ビット列 $Z=\{Z_1, Z_2, \cdots, Z_n\}\in\{0, 1\}^n$ のうち，特定の出現パターン $B=\{b_1, b_2, \cdots, b_m\}\in\{0, 1\}^m$ について統計的に検定するものである。部分列 B のことをテンプレートという。ビット列 Z のうち，$j=1$ から $j=m$ までの部分列について一致しているかどうか調べ，次は $j=2$ から始まる部分列について調べる。重複を許容する点が4.7節の検定とは異なる。

4.9 Maurer のユニバーサル統計検定（maurers_universal_test）

ビット列 $Z=\{Z_1, Z_2, \cdots, Z_n\}\in\{0, 1\}^n$ のうち，特定のパターンの出現間隔について統計的に検定するものである。

4.10 線形複雑度の検定（linear_complexity_test）

ビット列 $Z=\{Z_1, Z_2, \cdots, Z_n\}\in\{0, 1\}^n$ について，線形的な関係から十分な複雑性があるかについて統計的に検定するものである。ビット列 Z を複数のブロックに分割し，それぞれのブロックにおいて

$$Z_j = c_1 Z_{j-1} + c_2 Z_{j-2} + \cdots + c_L Z_{j-L}$$

を満たす係数 $c=\{c_1, c_2, \cdots, c_L\}\in\{0, 1\}^L$ の存在について調べる。

4.11 出現系列による検定（serial_test）

ビット列 $Z=\{Z_1, Z_2, \cdots, Z_n\}\in\{0, 1\}^n$ のうち，すべての m ビット長の出現パターン $B=\{b_1, b_2, \cdots, b_{2^m}\}\in\{0, 1\}^{2^m}$ について統計的に検定するものである。パターンのビットの長さを変えることで，出現頻度がどのように変化するのかを見る。このようにして，パターンの長さと出現の一様性に関する検定を実現する。

4.12 近似エントロピーによる検定（approximate_entropy_test）

ビット列 $Z=\{Z_1, Z_2, \cdots, Z_n\}\in\{0, 1\}^n$ のうち，すべての m ビット長の出現パターン $B=\{b_1, b_2, \cdots, b_{2^m}\}\in\{0, 1\}^{2^m}$ の重複による数え方によって，エントロピーという尺度を使って検定するものである。一般に，エントロピーは，各事象の出現確率 $p=\{p_1, p_2, \cdots, p_n\}$ に対して，

$$E=-\sum_{j=1}^{n} p_j \log p_j$$

を使って計算でき，乱雑性の指標とすることができる。

4.13 累積和による検定（cumulative_sums_test）

ビット列 $Z=\{Z_1, Z_2, \cdots, Z_n\}\in\{0, 1\}^n$ について，確率事象としてのランダム・ウォークの概念を用いて，その乱数性を統計的に検定するものである。ビット列 Z の部分和

$$W_k=\sum_{j=1}^{k}(2Z_j-1)$$

を使って，ランダム・ウォークの振る舞いから見て統計的に自然かどうかで判

断する。

4.14 ランダム偏差検定（random_excursion_test）

ビット列 $Z=\{Z_1, Z_2, \cdots, Z_n\}\in\{0, 1\}^n$ について，確率事象としてのランダム・ウォークの概念を用いて，その乱数性を統計的に検定するものである。ビット列 Z の部分和

$$W_k = \sum_{j=1}^{k}(2Z_j - 1)$$

を使って，累積和の値ごとのサイクルの数について分布を考え，統計的に自然かどうかで判断する。

4.15 種々のランダム偏差検定（random_excursion_variant_test）

ビット列 $Z=\{Z_1, Z_2, \cdots, Z_n\}\in\{0, 1\}^n$ について，確率事象としてのランダム・ウォークの概念を用いて，その乱数性を統計的に検定するものである。ビット列 Z の部分和

$$W_k = \sum_{j=1}^{k}(2Z_j - 1)$$

を使って，全サイクルを通して累積和が特定の値になる回数を用いて，統計的に自然かどうかで判断する。

5 実 験 結 果

第3章で述べたアルゴリズムに従ってエルゴード乱数を生成し，その乱数性について統計的に検定した結果を示す。比較のため，物理乱数による実験結果と合わせて紹介する。なお，実験で用いた計算機のスペックは表1のとおりである。

表 1　実験で用いた計算機のスペック

項　目	性　　能
PC	MacBook Pro（13-inch，M1，2020）
チップ	Apple M1（8 cores）
メモリ	16GB（DDR4）
OS	macOS Big Sur 11. 5. 2
Metal ファミリー	Metal GPUFamily Apple 7

5. 1　計算速度

第 3 章で実装したエルゴード乱数の生成プログラムを実行し，計算時間を測定した。コンパイル時に最適化オプションを付けずに実行ファイルを生成し，計算結果を格納するファイル処理の時間は含めない。比較のため，**/dev/urandom** から読み出せる物理乱数（ソースコードは付録参照）の生成時間を付けた。その結果，ハードウェアに対するアクセスのある物理乱数に比べて 50 倍ほど早い計算速度を実現した。（表 2 ）

表 2　計算速度

	エルゴード乱数	物理乱数
生成ビット列（bit）	32000000	32000000
計算時間（秒）	10. 952	509. 690
計算速度（Gbps）	2.922×10^{6}	6.278×10^{4}

5. 2　乱数性の統計的検定

第 3 章で実装したエルゴード乱数のプログラムから生成されたビット列について，David Johnston による NIST Special Publication 800-22 の実装[20] を用いて乱数性を統計的に検定した。この検定法のうち，ブロック内一様性の検定（frequency_within_block_test），2 値行列の階数を用いた検定（binary_matrix_rank

表 3　エルゴード乱数の統計的検定結果（4 M バイト）

検　定　法	p 値	結果
monobit_test	0. 990973159236	PASS
frequency_within_block_test	0. 0	FAIL
runs_test	0. 930973124344	PASS
longest_run_ones_in_a_block_test	0. 597727248391	PASS
binary_matrix_rank_test	0. 0	FAIL
dft_test	0. 0	FAIL
non_overlapping_template_matching_test	1. 00286438899	PASS
overlapping_template_matching_test	0. 0727848032788	PASS
maurers_universal_test	0. 0	FAIL
linear_complexity_test	0. 712068926252	PASS
serial_test	0. 999996558756	PASS
approximate_entropy_test	0. 999999995337	PASS
cumulative_sums_test	1. 0	PASS
random_excursion_test	0. 0	FAIL
random_excursion_variant_test	0. 0	FAIL

_test），離散フーリエ変換による検定（dft_test），Maurer のユニバーサル統計検定（maurers_universal_test），ランダム偏差検定（random_excursion_test），および種々のランダム偏差検定（random_excursion_variant_test）の 6 種類では良い結果が得られていない。生成されるビット列は四則演算で得られることから手軽ではあるが，十分にエルゴード性が活用されているのか，それとも実装上の改善の余地があるのか，さらに検討が必要である。

参考までに，物理乱数の統計的検定結果を表 4 に示した。十分な統計的精度で乱数性を有していることが確認できる。

表 4　物理乱数の統計的検定結果（4 M バイト）

検　定　法	p 値	結果
monobit_test	0. 586563113257	PASS
frequency_within_block_test	0. 924830087853	PASS
runs_test	0. 300701527782	PASS
longest_run_ones_in_a_block_test	0. 740058992223	PASS
binary_matrix_rank_test	0. 143220790940	PASS
dft_test	0. 288617290293	PASS
non_overlapping_template_matching_test	0. 0531717886703	PASS
overlapping_template_matching_test	0. 878586801048	PASS
maurers_universal_test	0. 492691166460	PASS
linear_complexity_test	0. 923521321232	PASS
serial_test	0. 0749329852117	PASS
approximate_entropy_test	0. 212079644465	PASS
cumulative_sums_test	0. 540284171643	PASS
random_excursion_test	0. 251259489364	PASS
random_excursion_variant_test	0. 287462058551	PASS

6　ま　と　め

本稿は，2020 年度に交付を受けた松山大学特別研究助成「擬似乱数生成器（PRNG）のアルゴリズムに関する理論および実装の総合的研究」によって，幾何学的エルゴード性に基づく新しい乱数のアルゴリズムを提案し，代数的拡大体を用いた実装によって計算リソースを節約できることを示した。さらに，生成されたビット列の乱数性について統計的に検定し，計算速度とともにハードウェア的な特性から得られる物理乱数と比較した。

提案手法の利点としては，ビット列の計算についてシフト演算および論理演算を用いて実装を試みたところである。しかし，int 型整数同士の除算や平方根の展開による double 型浮動小数点演算が含まれることがあって，物理乱数

に比べて 50 倍の速度が出ているものの，高速な Xorshift には及ばないであろうことは容易に想像できる。

もっとも本質的な問題は，乱数性である。NIST SP 800-22 の検定結果を見る限り，多くの改善の余地がある。エルゴード性を用いた生成アルゴリズムは，実装によっては周期を無限大にすることができる。したがって，その乱数性が増せば，どんなに規模の大きなモンテカルロ・シミュレーションであっても実用上の乱数の制約を受けずに任意の精度で実行できることになる。有理数体に添加する無理数として黄金比が最適であるかは証明されていない。エルゴード性の本質によって乱数性を向上させることは今後の課題である。

最後に，本研究に取り組んで得られた最新の成果は，実験に用いたプログラムのソースコードをインターネットで公開するとともに，ACM を含むトップレベルの国際会議や学術専門誌（ジャーナル）に論文が採択される水準まで研究のクオリティを高めたい。そして，2021 年度「卒業論文」における乱数に関する研究活動の中で，檀ゼミ 12 期生の学生たちに感謝するとともに，そのときの議論を通じて見出された課題を解決していくことによって本稿が完成に至ることになったことを付記して締めくくる。

参 考 文 献

［1］ 檀裕也「モンテカルロ法による研究室配属モデルのシミュレーション」松山大学論集，第 19 巻 第 4 号，pp. 75-89.（2007 年）

［2］ 檀裕也「モンテカルロ・シミュレーションによる予測の精緻化に関する数理モデル」松山大学論集，第 23 巻 第 3 号，pp. 315-334.（2011 年）

［3］ Yuya DAN, “Mathematical Analysis and Simulation of Information Diffusion on Networks,” Proceedings of International Symposium on Applications and the Internet（SAINT2011），pp. 550-555.（2011）

［4］ Yuya DAN, “Information Diffusion and Dissipative Effect on Social Networks,” IT Enabled Services, pp. 39-59（Springer）.（2013）

［5］ Yuya DAN and Takehiro MORIYA, “Analysis and Simulation of Power Law Distribution of File Types in File Sharing Systems,” Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation, pp. 116-121.（2011）

［6］ 檀裕也「量子乱数によるモンテカルロ・シミュレーションの理論および実証研究」松山大学論集，第 27 巻 第 4-2 号，pp. 121-152.（2015 年）

［7］ D. H. Lehmer, “Mathematical methods in large-scale computing units,” Annu. Comput. Lab. Harvard Univ. 26, pp. 141-146.（1951）

［8］ M. Matsumoto and T. Nishimura, “Mersenne twister : a 623-dimensionally equidistributed uniform pseudo-random number generator,” ACM Trans. Model. Comput. Simul. 8, 1, pp. 3-30.（January 1998）

［9］ G. Marsaglia, “Xorshift RNGs,” Journal of Statistical Software, Vol. 8, Iss. 14, pp. 1-6.（2003）

［10］ Tatsuki IKEDA, “Ergodic Pseudo-Random Number Generators,” 2021 年度松山大学経営学部情報コース卒業論文（2022 年）

［11］ 池田樹生，檀裕也「Ergodic PRNG－エルゴード性を用いた周期なし擬似乱数生成器－」情報処理学会第 84 回全国大会講演論文集，Vol. 1，7 K-01，pp. 251-252.（2022 年）

［12］ Birger Jansson, *Random number generators*, Victor Pettersons Bokindustri Aktiebolag, Stockholm（1966）

［13］ M. E. O’Neill, “PCG : A family of simple fast spaceefficient statistically good algorithms for random number generation,” Technical report no. HMC-CS-2014-0905. Claremont, CA : Harvey Mudd College, Computer Science Department. Retrieved from https://www.pcg-random.org/pdf/hmc-cs-2014-0905.pdf

［14］ 杉田洋「無理数回転を利用した擬似乱数生成法」数理解析研究所講究録 no. 1011，pp. 77-88.（1997 年）
https://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/1011-7.pdf

［15］ D. Knuth, *The Art of Computer Programming*, seminumerical algorithms, Vol. 3.

［16］ G. Marsaglia, “DIEHARD Statistical Tests,” in CD-ROM.
See also https://webhome.phy.duke.edu/~rgb/General/dieharder.php

［17］ NIST, “A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications,” Special Publication 800-22.（2001）

［18］ A. Rukhin, et al., “A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications,” NIST Special Publication 800-22.（2010）
https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final

［19］ 情報処理振興事業協会セキュリティセンター「電子政府情報セキュリティ技術開発事業：擬似乱数検証ツールの調査開発調査報告書」（2003 年）
https://www.ipa.go.jp/files/000013665.pdf

［20］ David Johnston, A python implementation of the SP800-22 Rev 1a PRNG test suite.
https://github.com/dj-on-github/sp800_22_tests

（以上，URL は 2022 年 3 月 22 日閲覧）

付録．エルゴード乱数の統計的検定結果（出力）

```
Tests of Distinguishability from Random
TEST: monobit_test
  Ones count   = 16000032
  Zeroes count = 15999968
  PASS
  P=0.990973159236
TEST: frequency_within_block_test
  n = 32000000
  N = 99
  M = 323232
  FAIL
  P=-1.80381486512e-109
TEST: runs_test
  prop  0.500001
  tau   0.000353553390593
  vobs  15999755.0
  PASS
  P=0.930973124344
TEST: longest_run_ones_in_a_block_test
  n = 32000000
  K = 6
  M = 10000
  N = 75
  chi_sq = 4.58727131365
  PASS
  P=0.597727248391
TEST: binary_matrix_rank_test
  Number of blocks 31250
  Data bits used: 32000000
  Data bits discarded: 0
  Full Rank Count = 4948
  Full Rank-1Count = 15063
  Remainder Count = 11239
  Chi-Square = 14280.7310041
  FAIL
  P=0.0
TEST: dft_test
  N0 = 15200000.000000
  N1 = 15899259.000000
  FAIL
  P=0.0
TEST: non_overlapping_template_matching_test
  PASS
  P=1.00286438899
```

```
TEST: overlapping_template_matching_test
  B =  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
  m =  10
  M =  1062
  N =  968
  K =  5
  model =  [352, 179, 134, 97, 68, 135]
  v[j] =   [577, 122, 102, 61, 47, 59]
  chisq =  10.0878200778
  PASS
  P=0.0727848032788
TEST: maurers_universal_test
  sum = 29433192.3259
  fn = 10.1893963968
  FAIL
  P=6.28927025011e-179
TEST: linear_complexity_test
  M =  512
  N =  62500
  K =  6
  chisq =  3.73809625045
  P =  0.712068926252
  PASS
  P=0.712068926252
TEST: serial_test
  psi_sq_m   =  0.0302589982748
  psi_sq_mm1 =  0.0164140015841
  psi_sq_mm2 =  0.00782049819827
  delta1     =  0.0138449966908
  delta2     =  0.00525149330497
  P1         =  1.00000000141
  P2         =  0.999996558756
  PASS
P=1.00000000141
P=0.999996558756
TEST: approximate_entropy_test
  n         =  32000000
  m         =  3
  Pattern 1 of 8, count = 4000073
  Pattern 2 of 8, count = 4000018
  Pattern 3 of 8, count = 3999884
  Pattern 4 of 8, count = 3999993
  Pattern 5 of 8, count = 4000018
  Pattern 6 of 8, count = 3999859
  Pattern 7 of 8, count = 3999993
  Pattern 8 of 8, count = 4000162
```

```
  phi(3)     =-4.382027
  Pattern 1 of 16, count = 2000025
  Pattern 2 of 16, count = 2000048
  Pattern 3 of 16, count = 2000005
  Pattern 4 of 16, count = 2000013
  Pattern 5 of 16, count = 1999944
  Pattern 6 of 16, count = 1999940
  Pattern 7 of 16, count = 1999959
  Pattern 8 of 16, count = 2000034
  Pattern 9 of 16, count = 2000048
  Pattern 10 of 16, count = 1999970
  Pattern 11 of 16, count = 1999879
  Pattern 12 of 16, count = 1999980
  Pattern 13 of 16, count = 2000074
  Pattern 14 of 16, count = 1999919
  Pattern 15 of 16, count = 2000034
  Pattern 16 of 16, count = 2000128
  phi(3)     = -5.075174
  AppEn(3)   = 0.693147
  ChiSquare = 0.0138449465226
  PASS
  P=0.999999995337
TEST: cumulative_sums_test
PASS
  PASS
P=1.0
P=1.0
TEST: random_excursion_test
J=113539
x = -4 chisq = 89.603225 p = 0.000000  Not Random
x = -3 chisq = 111.822177 p = 0.000000  Not Random
x = -2 chisq = 96.869463 p = 0.000000  Not Random
x = -1 chisq = 75.409667 p = 0.000000  Not Random
x = 1 chisq = 67.918801 p = 0.000000  Not Random
x = 2 chisq = 93.171841 p = 0.000000  Not Random
x = 3 chisq = 128.435574 p = 0.000000  Not Random
x = 4 chisq = 164.167846 p = 0.000000  Not Random
FAIL: Data not random
 FAIL
P=8.14083367367e-18
P=1.68784284039e-22
P=2.41291308557e-19
P=7.64043981112e-15
P=2.7762689166e-13
P=1.44758724949e-18
P=5.11117070065e-26
```

```
P=1.27958151673e-33
TEST: random_excursion_variant_test
J= 113539
x = -9 count=81869 p = 0.000000  Not Random
x = -8 count=84969 p = 0.000000  Not Random
x = -7 count=89128 p = 0.000000  Not Random
x = -6 count=92880 p = 0.000000  Not Random
x = -5 count=96117 p = 0.000000  Not Random
x = -4 count=99163 p = 0.000000  Not Random
x = -3 count=102542 p = 0.000000  Not Random
x = -2 count=106111 p = 0.000000  Not Random
x = -1 count=109879 p = 0.000000  Not Random
x = 1 count=117042 p = 0.000000  Not Random
x = 2 count=121232 p = 0.000000  Not Random
x = 3 count=125853 p = 0.000000  Not Random
x = 4 count=130469 p = 0.000000  Not Random
x = 5 count=134705 p = 0.000000  Not Random
x = 6 count=139302 p = 0.000000  Not Random
x = 7 count=144690 p = 0.000000  Not Random
x = 8 count=149953 p = 0.000000  Not Random
x = 9 count=155203 p = 0.000000  Not Random
FAIL : Data not random
  FAIL
P=1.87848803949e-58
P=4.71868973349e-54
P=8.19781181084e-46
P=4.79114425244e-39
P=3.65551146481e-34
P=4.06106279976e-30
P=5.6912484366e-25
P=2.26518866447e-19
P=1.58380009624e-14
P=1.96575392505e-13
P=1.15597577787e-20
P=6.84356701538e-31
P=4.12886951423e-41
P=1.34506253174e-49
P=9.71911525971e-60
P=1.82669589251e-73
P=1.18301495571e-86
P=8.49229206273e-100

SUMMARY
-------
monobit_test                          0.990973159236       PASS
frequency_within_block_test           -1.80381486512e-109  FAIL
```

runs_test	0.930973124344	PASS
longest_run_ones_in_a_block_test	0.597727248391	PASS
binary_matrix_rank_test	0.0	FAIL
dft_test	0.0	FAIL
non_overlapping_template_matching_test	1.00286438899	PASS
overlapping_template_matching_test	0.0727848032788	PASS
maurers_universal_test	6.28927025011e-179	FAIL
linear_complexity_test	0.712068926252	PASS
serial_test	0.999996558756	PASS
approximate_entropy_test	0.999999995337	PASS
cumulative_sums_test	1.0	PASS
random_excursion_test	1.27958151673e-33	FAIL
random_excursion_variant_test	8.49229206273e-100	FAIL