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

平成15年2月

情報処理振興事業協会

セキュリティセンター

目 次

1　はじめに

乱数列の性質を調べるためには、統計的な手法を使ってその性質を解析するという手段が一般的である。このため、様々な統計的検定法が提案されている。しかしながら、提案されている検定方法の数はきわめて多く、また、乱数検定ツールや文献によって、採用している検定の種類や数はまったく異なっているのが現状である。そこで、本調査・開発では、既存の乱数検定法や乱数検定ツールを調査し、数学的な考察や実際に乱数生成アルゴリズムからの出力に対して検定を行うことにより、有効な検定とそうでないものとを分類し乱数列を検定する上で必要最小限のもので構成されるミニマムセットを選び出すのが目的である。本調査・開発では、ミニマムセットを導出に用いる乱数生成アルゴリズムに、NIST のツール [5] および DIEHARD [7] に付属しているものを使用している。

なお、統計的な乱数検定は、真にランダムな系列の持つべき種々の性質の一部を保障するものであり、各種検定に合格したからといって、検定対象の乱数列の性質の十分性を示しているわけではないことに注意する必要がある。

以下、本調査報告書の構成を述べる。2 章では、既存の乱数検定法や乱数検定ツールの概要を述べる。3 章では、2 章で調査した各種乱数検定法を検定の目的ごとに分類し、それぞれの検定の目的および数学的根拠を明確にする。4 章では、3 章で分類した検定の中から有効な検定を選びミニマムセットを導出し、導出したミニマムセットの妥当性を検証する。

2 擬似乱数評価の現状

2.1 The Art of Computer programming, 準数値算法/乱数

2.1.1 検定法の概要

Knuth の著書 The Art of Computer programming, 準数値算法/乱数 [1] では、以下の 12 種類の乱数の検定方法が示されている。

K1 頻度検定

K2 系列検定 (2 次元度数検定)

K3 間隔検定

K4 ポーカー検定

K5 札集め検定

K6 順列検定

K7 連の検定

K8 t 個の数の最大値検定

K9 衝突検定

K10 系列相関検定

K11 部分数列に関する検定

K12 スペクトル検定

文献 [1] では、基本となる乱数を区間 $[0,1)$ 上を分布する実数列

$$< u_n >= u_0, u_1, u_2, \cdots$$

としており、0 と $d-1$ の間で分布する乱数を扱うときは、$< u_n >$ を

$$< U_i >= \lfloor du_i \rfloor$$

と変形した系列

$$< U_n >= U_0, U_1, U_2, \cdots$$

を扱っている。したがって、上記の検定には、

- 区間 $[0,1)$ 上を分布する乱数列 $< u_n >$ に適用する検定
- 0 と $d-1$ の間で分布する乱数列 $< U_n >$ に適用する検定

の 2 種類があり、$< U_n >$ には (そのままでは) 適用できないものや、乱数のアルファベットが 0 と 1 の場合 ($d = 2$) には、意味がないものなどがある。

2.2 FIPS PUS 140-2

2.2.1 検定法の概要

暗号モジュールのセキュリティ要件が書かれている FIPS PUS 140-2 [2] では、下記の 4 つの検定が採用されている。

F1 1 次元度数検定 (monobit test)

F2 ポーカー検定 (poker test)

F3 連の検定 (runs test)

F4 最長連の検定 (longest runs test)

文献 [2] では、0 と 1 からなる 20000 ビットの乱数列だけが検定対象である。良い乱数かどうかは、検定結果が予め定められた範囲に含まれているかどうかで判断する。

なお、2002 年 12 月に発行された FIPS PUS 140-2 の Change Notice 2 [3] では、上記の検定の記述が削除されている。

2.3 乱数

2.3.1 検定法の概要

伏見著の乱数 [4] では、以下の 7 つの検定法が採用されている。

R1 1 次元度数検定

R2 2 次元度度数検定

R3 ポーカー検定

R4 衝突検定

R5 OPSO 検定

R6 間隔検定

R7 連の検定

文献 [4] の検定も、文献 [1] と同様に、

- 区間 $[0, 1)$ 上を分布する乱数列 $< u_n >$ に適用する検定
- 0 と $d-1$ の間で分布する乱数列 $< U_n >$ に適用する検定

の 2 種類に分類することができる。

2.4 NIST Special Publication 800-22

2.4.1 検定法の概要

NIST Special Publication 800-22 [5] では、以下の 16 種類の検定法が採用されている。

N1 1 次元度数検定 (frequency test)

N2 ブロック単位の頻度検定 (frequency test within a block)

N3 連の検定 (runs test)

N4 ブロック単位の最長連検定 (test for longest run of ones in a block)

N5 2 値行列ランク検定 (binary matrix rank test)

N6 離散フーリエ変換検定 (discrete fourier transform (spectral) test)

N7 重なりのないテンプレート適合検定 (non-overlapping template matching test)

N8 重なりのあるテンプレート適合検定 (overlapping template matching test)

N9 Maurer のユニバーサル統計検定 (Maurer's universal statistical test)

N10 Lempel-Ziv 圧縮検定 (Lempel-Ziv compression test)

N11 線形複雑度検定 (linear complexity test)

N12 系列検定 (serial test)

N13 近似エントロピー検定 (approximate entropy test)

N14 累積和検定 (cumulative sums (cusums) test)

N15 ランダム偏差検定 (random excursions test)

N16 種々のランダム偏差検定 (random excursions variant test)

NIST で採用されているすべての検定は、0 と 1 からなる乱数列を対象としている。また、NIST の検定では、各検定ごとに p-value が得られる。p-value とは、真の乱数生成器が検定を行っている系列よりも乱数らしからぬ系列を生成する確率である。例として、頻度検定の場合を考える。このとき、p-value は以下のように求める。

1. $X_1, X_2, \cdots, X_n$ を $1, -1$ の中の値をとる n 個の確率変数とし、$S_n = X_1 + X_2 + \cdots + X_n$ とする。
2. 系列が真の乱数生成器からの出力ならば、

$$\begin{aligned} \mu &= 0 \\ \sigma^2 &= n \end{aligned}$$

となるので、中心極限定理より、

$$\lim_{n\to\infty} P\left(\frac{S_n}{\sqrt{n}} \leq z\right) = \Phi(z)$$

となる。なお、

$$\Phi(z) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx$$

は標準正規分布の累積分布関数である。

3. 統計量 $s = |S_n|/\sqrt{n}$ を考える。このとき、

$$P\left(s \leq z\right) = 2\Phi(z) - 1$$

が得られ、

$$p\text{-value} = 2[1 - \Phi(s)]$$

である。なお、NIST では、$\Phi(z)$ のかわりに

$$erfc(z) = \int_{z}^{\infty} \frac{2}{\sqrt{\pi}} e^{-x^2} dx$$

を用いているため、

$$p\text{-value} = erfc(s/\sqrt{2})$$

となる。

NIST のツールでは、p-value< 0.01 のときに良い乱数ではないと判断する。

なお、統計量がカイ 2 乗統計量の場合、p-value は、

$$\int_{z}^{\infty} \frac{1}{2\Gamma(\frac{N}{2})} \left(\frac{t}{2}\right)^{\frac{N}{2}-1} e^{-\frac{t}{2}} dt$$

により求める。ただし、N は χ^2 分布の自由度であり、

$$\Gamma(\alpha) = \int_{0}^{\infty} x^{\alpha-1} e^{-x} dx$$

である。

NIST では、1 本の標本系列に対する仮説検定で乱数生成アルゴリズムを評価するのは無意味であるという考え方から、複数の標本系列 (NIST では 1000 程度を推奨している) に対し検定を行い、

1. p-value の一様性
2. p-value が 0.01 より大きくなる割合

から乱数列の評価を行う。1. では、得られた p-value が区間 $[0, 1)$ で一様に分布しているかどうかを調べるために、$[0, 1)$ を 10 の区間に分割し、分割した区間ごとの頻度が一様になっているかどうかをカイ 2 乗検定にて検定する。カイ 2 乗検定により得られた p-value が 0.0001 以上ならば、乱数列は良い乱数であると判断する。また、2. では、標本の数を m としたとき、0.01 以上となる p-value の数の割合が

$$0.99 \pm \sqrt{\frac{0.99 \times 0.01}{m}}$$

の範囲に入っている場合は、乱数列は良い乱数であると判断する。

2.4.2 ツールの概要

NIST のツールでは、検定対象の乱数としてバイナリ形式のデータと '0' と '1' からなる ASCII 形式のデータが入力可能である。ソースコードは NIST のホームページより入手可能である。また、NIST のツールには、指定された乱数のデータを検定する機能の他に、以下のアルゴリズムによる乱数生成機能があり、生成した乱数列を検定することも可能である。

1. G Using SHA-1(FIPS 186 で定められた SHA-1 ベースの乱数生成法)
2. Linear Congruential
3. Blum-Blum-Shub
4. Micali-Schnorr
5. Modular Exponentiation
6. Quadratic Congruential I
7. Quadratic Congruential II
8. Cubic Congruential
9. XOR
10. ANSI X9.17 (3-DES)
11. G Using DES(FIPS 186 で定められた DES ベースの乱数生成法)

検定を行う場合には、

1. 乱数列の (1 本の) 長さ
2. 乱数列の本数
3. 乱数列のファイル名、または、用意されている乱数生成アルゴリズムの番号
4. 実行する検定
5. 検定ごとの各種パラメータ
6. 入力ファイルの形式 (入力データがファイルの場合)

を指定する必要がある。検定結果は、finalAnalysisReport というファイルにまとめられる。

2.5 DIEHARD

2.5.1 検定法の概要

DIEHARD [7] では、以下の 18 個の検定方法を採用している。

D1 バースデイ空間検定 (birthday spacings test)

D2 OPERM5 検定 (overlapping 5-permutation test)

D3 (31×31) の 2 値行列ランク検定 (binary rank test for 31x31 matrices)

D4 (32×32) の 2 値行列ランク検定 (binary rank test for 32x32 matrices)

D5 (6×8) の 2 値行列ランク検定 (binary rank test for 6x8 matrices)

D6 ビット列検定 (bitstream Test)

D7 OPSO 検定 (OPSO(Overlapping-Pairs-Sparse-Occupancy) test)

D8 OQSO 検定 (OQSO(Overlapping-Quadruples-Sparse-Occupancy) test)

D9 DNA 検定 (DNA test)

D10 8 ビット中の文字数検定 (count-the-1's test on a stream of bytes)

D11 特定位置の 8 ビット中の文字数検定 (count-the-1's test for specific bytes)

D12 駐車場検定 (parking lot test)

D13 最小距離検定 (minimum distance test)

D14 3DSPHERES 検定 (3d-spheres test)

D15 スクイーズ検定 (squeeze test)

D16 重なりのある和検定 (overlapping sums test)

D17 連の検定 (runs test)

D18 クラップス検定 (craps test)

DIEHARD で採用されているすべての検定は、0 と 1 からなる乱数列を対象としており、乱数列の長さは、10MByte から 11MByte 程度必要である。DIEHARD では、18 種類の検定から 200 以上の p-value が得られる。検定対象が良い乱数列の場合には、得られた p-value は $[0, 1)$ に一様に分布し、また、良くない乱数列の場合には、p-value の分布に偏りが生じる。なお、DIEHARD では、多数の p-value が得られるだけで、検定対象を良い乱数と判断するための基準は述べられていない。

2.5.2 ツールの概要

DIEHARD を実行する場合には、

1. 入力ファイル名
2. 出力ファイル名
3. 実行する検定

を指定する必要がある。なお、入力は、バイナリ形式のファイルに限られる。

DIEHARD は、'0' と '1' からなる ASCII 形式のデータをバイナリ形式のデータに変換するプログラムや以下のアルゴリズムにより乱数を生成するプログラムと共にホームページにて公開されている。

1. Multiply-with-carry (MWC) generator (MWC generator)
2. MWC generator on pairs of 16 bits(MWC1616 generator)
3. "Mother of all random number generators"(MOTHER generator)
4. KISS generator
5. Simple but very good generator COMBO (COMBO generator)
6. Lagged Fibonacci-MWC combination ULTRA (ULTLA generator)
7. Combination MWC/subtract-with-borrow (SWB) generator (MWC/SWB generator)
8. Extended congruential generator (EXCONG generator)
9. Super-Duper generator (SUPERDUPER generator)
10. Subtract-with-borrow generator (SWB generator)
11. Specified congruential generator
12. 31-bit generator ran2 from Numerical Recipes (RAN2 generator)
13. Specified shift-register generator
14. System generator in Microsoft Fortran (MSRAN generator)
15. Lagged-Fibonacci generator
16. Inverse congruential generator (INVCONG generator)

DIEHARD のソースコードは、

(a) Fortoran 言語で書かれたソース

(b) (a) を C 言語に変換したソース

(c) はじめから C 言語で書かれたソース

の 3 種類が公開されている。また、

(i) (a)、(b) を DOS(Windows) 上でコンパイルした実行形式

(ii) (a)、(b) を Linux 上でコンパイルした実行形式

(iii) (a)、(b) を Sun 上でコンパイルした実行形式

も同時に公開されている。(b) と (c) では使用している検定のパラメータ等が異なるため、入力が同じでも検定結果は異なる。また、(i) と (iii) も入力が同じでも検定結果が異なっている。これは、エンディアン等のプラットフォームに依存する部分を考慮して開発されていないことが原因と考えられる。なお、(c) もプラットフォームに依存する部分を考慮していないため、異なるプラットフォームでは入力が同じでも検定結果は異なる。

2.6 他の文献およびツール

2.1-2.5 で述べた乱数検定に関する文献ツール以外にも PLAB の乱数検定 [8]、乱数検定ツール ENT [9]、ストリーム暗号やブロック暗号の評価ツール Crypt-X [10] などがある。PLAB の乱数検定は、シミュレーション用の乱数に対する検定方法が示されている。ENT は、暗号やシミュレーションで使う乱数生成器や圧縮アルゴリズムなどの評価に使うツールである。Crypt-X には、バイナリのビット列を検定する stream cipher tests、鍵生成器 (key generator) から出力を検定する key generator tests、および、ブロック暗号からの出力を検定する block cipher tests の 3 種類の試験が用意されている。stream cipher tests では、以下の検定を行う。

- '0' と '1' の出現度数の偏りを調べる (Frequency)
- １つ前のビットとの相関を調べる (Binary Derivative)
- 重なりのない固定長の部分列の一様性を調べる (Change Point)
- 連長の出現度数の偏りを調べる (Runs)
- パターンの出現性を調べる (Sequence Complexity)
- 線形複雑度を調べる (Linear Complexity)

key generator tests では、以下の検定を行う。

- '0' と '1' の出現度数の偏りを調べる (Frequency)
- １つ前のビットとの相関を調べる (Binary Derivative)
- サブブロックの出現性を調べる (Sub-blocks)
- ブロックのエントロピーを求めその偏りを調べる (Entropy)

そして、block cipher tests では、以下の検定を行う。

- '0' と '1' の出現度数の偏りを調べる (Frequency)
- １つ前のビットとの相関を調べる (Binary Derivative)
- サブブロックの出現性を調べる (Sub-blocks)
- 入力を 1 ビット変えたときに出力の半分のビットが変化しているかを調べる (Avalanche Criteria, Avalanche Variable)

3　各種検定の数学的考察

本章では、2 章で述べた各種検定を

- ブロック単位の頻度に関する検定
- パターンの出現に関する検定
- 状態遷移・ランダムウォークに関する検定
- 一様性・圧縮可能性に関する検定
- 周期性に関する検定
- その他の検定

の 6 つに分類し、それぞれの検定の目的および数学的根拠を明確にする。

3.1　ブロック単位の頻度に関する検定

3.1.1　1 次元度数検定

乱数列の文字の出現頻度を求めその度数分布が一様になっているかどうかをカイ 2 乗検定にて検定する。乱数列が 0 と 1 からなる列の場合、求めるクラスは 2 個である。

入力	$\{U_i\}$	:	0 から $d-1$ の中から値をとる乱数列
	n	:	乱数列の長さ（乱数の個数）
出力	χ_0^2	:	カイ 2 乗統計量
処理内容	ステップ 1	:	$U_i = 0, 1, 2, \cdots, d-1$ となる個数を数え、それぞれ $n_0, n_1, n_2, \cdots, n_{d-1}$ とおく。
	ステップ 2	:	すべての i に対し $p_i = 1/d$ として、カイ 2 乗統計量 χ_0^2 を計算する。χ_0^2 は d 個の和で、自由度 $d-1$ の χ^2 分布に従う。

3.1.2　2 次元度数検定

乱数列を 2 文字組みに分割し、各組の度数分布が一様になっているかどうかをカイ 2 乗検定にて検定する。乱数列が 0 と 1 からなる列の場合、求めるクラスは 4 個である。

入力	$\{U_i\}$	:	0 と $d-1$ の中から値をとる乱数列
	n	:	乱数列の長さ（乱数の個数）
出力	χ_0^2	:	カイ 2 乗統計量
処理内容	ステップ 1	:	U_i を順に 2 つずつペア (U_{2j}, U_{2j+1}) とする。n が奇数だった場合は、最後の乱数 U_n は捨てる。
	ステップ 2	:	$(U_{2j}, U_{2j+1}) = (0,0), (0,1), (0,2), \cdots, (0, d-1), \cdots, (d-1, 0), (d-1, 1), \cdots, (d-1, d-1)$ となる個数を数え、それぞれ $n_0, n_1, n_2, \cdots, n_{d-1}, \cdots, n_{d^2-1}$ とおく。
	ステップ 3	:	すべての i に対し $p_i = 1/d^2$ として、カイ 2 乗統計量 χ_0^2 を計算する。χ_0^2 は d^2 個の和で、自由度 d^2-1 の χ^2 分布に従う。

なお、乱数列の検定を行う場合より高次元まで検定を行ったほうが良いが、次元が高くなるにつれて、クラスの数、検定に要する計算時間とも大幅に増加する。乱数列が 0 と 1 からなる列の場合、乱数列の長さを n とすると、各クラスの度数の期待値が小さいと正しく検定できないため、次元 m は

$$m \leq \lfloor \log_2 n \rfloor - 7$$

を満たすように選ぶ必要がある。

3.1.3 連の検定 (SP800-22)

0 と 1 からなる乱数列に対する検定法である。乱数列 $\{X_i\}$ の中で、$X_i \neq X_{i+1}$ となっている所の個数を求め、その数の偏りを調べる。乱数列が 0 と 1 からなる列の場合、求めるクラスは 2 個である。

入力	$\{X_i\}$	:	0 と 1 の中から値をとる乱数列
	n	:	乱数列の長さ（乱数の個数）
出力	p-value	:	正規分布統計量からの p-value
処理内容	ステップ 1	:	入力乱数列の 1 の比率 $\pi = \dfrac{\sum_{j=1}^{n} X_j}{n}$ を求める。$\lvert\pi - 1/2\rvert \geq 2/\sqrt{n}$ ならば、検定を終了する。
	ステップ 2	:	統計量 $V_n(obs) = \sum_{k=1}^{n-1} r(k) + 1$ を求める。ただし、$X_i = X_{i+1}$ のときは $r(k) = 0$ とし、$X_i \neq X_{i+1}$ のときは $r(k) = 1$ とする。
	ステップ 3	:	中心極限定理より、連の総数は平均 $2n\pi(1-\pi)$、分散 $8n\pi^2(1-\pi)^2$ の正規分布に従うので、この分布から p-value を計算する。

3.1.4 ポーカー検定

0 と $d-1$ の中から値をとる乱数列を 5 文字組みに分割し、各部分列を (1)5 個同種、(2)3 個同種と対、または 4 個同種、(3) 対 2 個、または 3 個同種、(4) 対 1 個、(5) すべて異種の 5 個のクラスに割り当て、その度数をカイ 2 乗検定にて検定する。

入力	$\{U_i\}$	:	0 と $d-1$ の中から値をとる乱数列
	n	:	乱数列の長さ（乱数の個数）
出力	χ_0^2	:	カイ 2 乗統計量
処理内容	ステップ 1	:	U_i を順に連続する 5 つずつの組 $(U_{5j}, U_{5j+1}, U_{5j+2}, U_{5j+3}, U_{5j+4})$ とする。n が 5 の倍数でない場合は、最後の半端な乱数は捨てる。
	ステップ 2	:	5 つ組の数字の組み合わせ（順序は問わない）が次の 5 種類の形となる個数を数え、それぞれ n_0, n_1, n_2, n_3, n_4 とおく。(1)5 個同種、(2)3 個同種と対、または 4 個同種、(3) 対 2 個、または 3 個同種、(4) 対 1 個、(5) すべて異種。（本来のポーカー検定では (2) と (3) を 2 つに分け、7 種類で行っている。）
	ステップ 3	:	各 i に対し $p_i = \frac{{}_dP_{i+1}}{d^5} \left\{ \begin{array}{c} 5 \\ i+1 \end{array} \right\}$ として、カイ 2 乗統計量 χ_0^2 を計算する。χ_0^2 は 5 個の和で、自由度 4 の χ^2 分布に従う。ここで ${}_dP_i = d(d-1)(d-2)\cdots(d-i+1)$、$\left\{ \begin{array}{c} 5 \\ i \end{array} \right\}$ は相異なる 5 個の要素からなる集合をちょうど i 個の空でない部分集合に分割する仕方の数である。

なお、FIPS 140-2 の検定のポーカー検定は、上記のポーカー検定ではなく、4 次元の度数検定を行っている。

3.1.5 重なりのないテンプレート適合検定

0 と 1 からなる乱数列を 8 つのブロックに分割し、各ブロックごとに m ビットの窓を先頭からスライドさせ、窓と m 文字のテンプレートが適合する回数を調べる。8 ブロックそれぞれの適合回数をカイ 2 乗検定にて検定し適合回数の偏りを調べる。

入力	$\{X_i\}$	:	0 と 1 の中から値をとる乱数列
	n	:	乱数列の長さ（乱数の個数）
	m	:	テンプレートの長さ
	B	:	m ビットのテンプレート
出力	p-value	:	カイ 2 乗統計量からの p-value
処理内容	ステップ 1	:	乱数列を長さ $M = 131,072 = 2^{17}$ の $N = 8$ 個のブロックに分割する。
	ステップ 2	:	ブロック j のテンプレートの適合回数を W_j とする。m ビットの窓をブロックの先頭にセットし、テンプレートが適合しないときは、窓を 1 ビットずらし、適合するときは窓を m ビットずらす。
	ステップ 3	:	中心極限定理より、各ブロックの適合回数はは平均 $\mu = (M-m+1)/2^m$、分散 $\sigma^2 = M\left(\frac{1}{2^m} - \frac{2m-1}{2^{2m}}\right)$ の正規分布に従うことを用いて、カイ 2 乗統計量 $\chi^2(obs) = \sum_{j=1}^{N} \frac{(W_j - \mu)^2}{\sigma^2}$ を求める。
	ステップ 4	:	$\chi^2(obs)$ は、自由度 N の χ^2 分布に従うので、この分布から p-value を計算する。

NIST のツールでは、テンプレートの長さを 2 から 10 まで選択できるが、9 または 10 が推奨されている。なお、NIST のツールでは、乱数列の長さが 1,000,000 の場合のみ検定することができる。

3.1.6 重なりのあるテンプレート適合検定

0 と 1 からなる乱数列を 968 個のブロックに分割し、各ブロックを m 文字のテンプレートが適合する回数により 6 個のクラス割り当て、その度数をカイ 2 乗検定にて検定し適合回数の偏りを調べる。

入力	$\{X_i\}$	:	0 と 1 の中から値をとる乱数列
	n	:	乱数列の長さ（乱数の個数）
	m	:	テンプレートの長さ
	B	:	m ビットのテンプレート
出力	p-value	:	カイ 2 乗統計量からの p-value
処理内容	ステップ 1	:	乱数列を長さ $M = 1032$ の $N = 968$ 個のブロックに分割する。
	ステップ 2	:	ブロック j のテンプレートの適合回数を W_j とする。m ビットの窓をブロックの先頭から 1 ビットずつずらして行き、テンプレートと窓の適合回数を数える。
	ステップ 3	:	適合回数 0 のクラスを f_0、1 のクラスを f_1、2 のクラスを f_2、3 のクラスを f_3、4 のクラスを f_4、5 以上のクラスを f_5 とし、ブロックの適合回数 W_j から各クラスの度数を求める。
	ステップ 4	:	カイ 2 乗統計量 $\chi^2(obs) = \sum_{i=0}^{5} \frac{(f_i - N\pi_i)}{N\pi_i}$ を求める。クラス f_i に対応する確率 π_i は、 $\pi_0 = e^{-\eta}$ $\pi_1 = \frac{\eta}{2} e^{-\eta}$ $\pi_2 = \frac{\eta e^{-\eta}}{8}[\eta + 2]$ $\pi_3 = \frac{\eta e^{-\eta}}{8}\left[\frac{\eta^2}{6} + \eta + 1\right]$ $\pi_4 = \frac{\eta e^{-\eta}}{16}\left[\frac{\eta^3}{24} + \frac{\eta^2}{2} + \frac{3\eta}{2} + 1\right]$ $\pi_5 = 1 - (\pi_0 + \pi_1 + \pi_2 + \pi_3 + \pi_4)$ となる。ただし、$\eta = \lambda/2,\ \lambda = (M - m + 1)/2^m$ である。
	ステップ 5	:	$\chi^2(obs)$ は、自由度 5 の χ^2 分布に従うので、この分布から p-value を計算する。

NIST のツールでは、テンプレートの長さとして 9 または 10 が推奨されている。なお、NIST のツールでは、乱数列の長さが 1,000,000 の場合のみ検定することができる。

3.1.7 スクイーズ検定

0 と 1 からなる乱数列を 32 ビット単位で分割し、各 32 ビット Y を $u = Y/2^{32}$ と $[0, 1)$ の小数に変換する。k の初期値を $k = 2^{31}$ とし、$k \leftarrow \lfloor k \times u \rfloor$ を繰り返し、$k = 1$ となるまでの繰り返し回数に応じて各部分列を 43 個のクラスに割り当て、その度数をカイ 2 乗検定にて検定する。

入力　$\{X_i\}$　：　0 と 1 の中から値をとる乱数列

出力　p-value　：　カイ 2 乗統計量からの p-value

処理内容　ステップ 1　：　くり返し回数を、6 以下、7, 8, ⋯, 46, 47, 48 以上の 43 個のクラスに分ける。

ステップ 2　：　X_i を列の先頭から順に 32 ビットずつ取り出し、32 ビットの整数からなる新しい列 $\{Y_i\}$ を作る。

ステップ 3　：　ステップ 2 の Y_i に対し、$v_i = Y_i/2^{32}$ とおき、小数列 $\{v_i\}$ を作る。

ステップ 4　：　$k = 2^{31}$ とおく。

ステップ 5　：　$k \leftarrow \lfloor k \times v_i \rfloor$ とする。

ステップ 6　：　$k = 1$ となるまでステップ 5 をくり返す。

ステップ 7　：　ステップ 4 のくり返し回数を j とし、各クラスの度数に加える。

ステップ 8　：　ステップ 4〜7 を 100,000 回くり返す。各クラスの度数を $f_0, f_1, \cdots, f_{42}$ とする。

ステップ 9　：　カイ 2 乗統計量 χ_0^2 を計算する。期待値の値は、次の表の値を用いる。χ_0^2 は自由度 42 の χ^2 分布に従うので、この分布から p-value を計算する。

〜6	7	8	9	10	11
21.03	57.79	175.54	467.32	1107.83	2367.84
12	13	14	15	16	17
4609.44	8241.16	13627.81	20968.49	30176.12	40801.97
18	19	20	21	22	23
52042.03	62838.28	72056.37	78694.51	82067.55	81919.35
24	25	26	27	28	29
78440.08	72194.12	63986.79	54709.31	45198.52	36136.61
30	31	32	33	34	35
28000.28	21055.67	15386.52	10940.20	7577.96	5119.56
36	37	38	39	40	41
3377.26	2177.87	1374.39	849.70	515.18	306.66
42	43	44	45	46	47
179.39	103.24	58.51	32.69	18.03	9.82
48〜					
11.21					

3.1.8　クラップス検定

0 と 1 からなる乱数列を 32 ビット単位で分割し、各 32 ビット Y を $u = \lfloor Y/2^{32} \times 6 \rfloor + 1$ と変換する。u を Craps におけるサイコロの値とみなして Craps を 20000 回行い、1 回の勝負がつくまでの繰り返し回数に応じて各部分列を 21 個のクラスに割り当て、その度数をカイ 2 乗検定にて検定する。また、勝率の偏りも調べる。

入力	$\{X_i\}$	:	0 と 1 の中から値をとる乱数列
出力	p-value	:	正規分布統計量からの p-value（勝った回数に対する）
	p-value	:	カイ 2 乗統計量からの p-value（1 回の勝負がつくまでのくり返し回数）
処理内容	ステップ 1	:	くり返し回数を $1, 2, 3, \cdots, 19, 20, 21$ 以上の 21 個のクラスに分ける。
	ステップ 2	:	X_i を列の先頭から順に 32 ビットずつ取り出し、32 ビットの整数からなる新しい列 $\{Y_i\}$ を作る。
	ステップ 3	:	ステップ 2 の Y_i に対し、$v_i = \lfloor Yi/2^{32} \times 6 \rfloor + 1$ とおき、1 から 6 までの整数からなる列 $\{v_i\}$ を作る。v_i は craps におけるサイコロの値に相当する。
	ステップ 4	:	ステップ 3 でのサイコロの値により craps を 200,000 回行い、勝った回数 w を数える。w は勝率 $p = 244/495$ としたとき、平均 $200000p$、分散 $200000p(1-p)$ の正規分布に従うので、この分布から p-value を計算する。
	ステップ 5	:	またステップ 4 で同時に 1 回の勝負がつくまでのくり返し回数を各クラスの度数に加えていく。クラスの度数を $f_1, f_2, \cdots, f_{21}$ とする。
	ステップ 5	:	$f_1, f_2, \cdots, f_{21}$ に対し、カイ 2 乗統計量 χ_0^2 を計算する。χ_0^2 は自由度 20 の χ^2 分布に従うので、この分布から p-value を計算する。

3.1.9 8 ビット中の文字数検定

0 と 1 からなる乱数列の 40 ビットの部分列を各バイト中の 1 の個数に応じて各バイトを文字に置き換えてその 5 文字組の 3625 個のクラスに割り当て、その度数をカイ 2 乗検定にて検定し、文字の出現頻度の偏りを調べる。

入力　$\{X_i\}$　：　0 と 1 の中から値をとる乱数列

出力　p-value　：　カイ 2 乗統計量からの p-value

処理内容　ステップ 1　：　X_i を列の先頭から順に 1 バイトずつに区切る。

ステップ 2　：　各バイトに対し、1 の個数を数え、個数が 2 以下、3, 4, 5, 6 以上に応じてそれぞれ文字 A, B, C, D, E に置き換え、A, B, C, D, E の 5 文字からなる新しい列 $\{S_i\}$ を作る。

ステップ 3　：　A, B, C, D, E の 5 種類に対し、5 文字の組み合わせにより $5^5 = 3625$ 個のクラスに分ける。また 4 文字の組み合わせに対して $5^4 = 625$ 個のクラスに分ける。

ステップ 4　：　$\{S_i\}$ の列の先頭から順に 5 文字ずつを組にし、組の種類に応じてクラスの度数に加えていく。各クラスの度数を $f_{5,0}, f_{5,1}, \cdots, f_{5,3624}$ とする。5 文字組は、$(S_1, S_2, S_3, S_4, S_5), (S_2, S_3, S_4, S_5, S_6)$ のように 1 バイトずつずらして重ねながら作っていく。

ステップ 5　：　ステップ 4 と同時に、$\{S_i\}$ の列の先頭から順に 4 文字ずつを組にし、組の種類に応じてクラスの度数に加えていく。各クラスの度数を $f_{4,0}, f_{4,1}, \cdots, f_{4,624}$ とする。4 文字組みは、5 文字組と同様に 1 バイトずつずらして重ねながら作っていく。

ステップ 6　：　f_5, f_4 に対して、それぞれカイ 2 乗統計量 $\chi^2_{5,0}, \chi^2_{4,0}$ を計算する。$\chi^2_{5,0}$ は自由度 3624 の、$\chi^2_{4,0}$ は自由度 624 の χ^2 分布に従う。

ステップ 7　：　$\chi^2_0 = \chi^2_{5,0} - \chi^2_{4,0}$ を計算する。χ^2_0 は自由度 $5^5 - 5^4 = 2500$ の χ^2 分布に従うので、この分布から p-value を計算する。

3.1.10　特定位置の 8 ビット中の文字数検定

0 と 1 からなる乱数列の 160 ビットの部分列から特定位置の 8 バイトを選び、各バイト中の 1 の個数に応じて各バイトを文字に置き換えてその 5 文字組の 3625 個のクラスに割り当て、その度数をカイ 2 乗検定にて検定し、文字の出現頻度の偏りを調べる。

入力　$\{X_i\}$　：　0 と 1 の中から値をとる乱数列

出力　p-value　：　カイ 2 乗統計量からの p-value（25 個）

処理内容　ステップ 1　：　X_i を列の先頭から順に 32 ビットずつに区切る。

ステップ 2　：　各 32 ビットから 1 バイト取り出し、各バイトに対して 1 の個数を数え、個数が 2 以下、3, 4, 5, 6 以上に応じてそれぞれ文字 A, B, C, D, E に置き換え、A, B, C, D, E の 5 文字からなる新しい列 $\{S_i\}$ を作る。

ステップ 3　：　A, B, C, D, E の 5 種類に対し、5 文字の組み合わせにより $5^5 = 3625$ 個のクラスに分ける。また 4 文字の組み合わせに対して $5^4 = 625$ 個のクラスに分ける。

ステップ 4　：　$\{S_i\}$ の列の先頭から順に 5 文字ずつを組にし、組の種類に応じてクラスの度数に加えていく。各クラスの度数を $f_{5,0}, f_{5,1}, \cdots, f_{5,3624}$ とする。5 文字組は、$(S_1, S_2, S_3, S_4, S_5), (S_2, S_3, S_4, S_5, S_6)$ のように 1 バイトずつずらして重ねながら作っていく。

ステップ 5　：　ステップ 4 と同時に、$\{S_i\}$ の列の先頭から順に 4 文字ずつ組にし、組の種類に応じてクラスの度数に加えていく。各クラスの度数を $f_{4,0}, f_{4,1}, \cdots, f_{4,624}$ とする。4 文字組みは、5 文字組と同様に 1 バイトずつずらして重ねながら作っていく。

ステップ 6　：　f_5, f_4 に対して、それぞれカイ 2 乗統計量 $\chi^2_{5,0}, \chi^2_{4,0}$ を計算する。$\chi^2_{5,0}$ は自由度 3624 の、$\chi^2_{4,0}$ は自由度 624 の χ^2 分布に従う。

ステップ 7　：　$\chi^2_0 = \chi^2_{5,0} - \chi^2_{4,0}$ を計算する。χ^2_0 は自由度 $5^5 - 5^4 = 2500$ の χ^2 分布に従うので、この分布から p-value を計算する。

ステップ 8　：　ステップ 1〜7 を 25 回くり返す。各回の違いは、32 ビットから取り出す 1 バイトの位置で、1 回目は先頭から 8 ビット、2 回目は 2 ビット目から 8 ビット、⋯、25 回目は 25 ビット目から 8 ビットと行う。

3.1.11　順列検定, OPERM5 検定

0 と 1 からなる乱数列を 160 ビットの部分列を 32 ビット単位の 5 文字組みとみなしたときの、5 文字組の大小関係により、120 個のクラスに割り当て、その度数の分布が一様になっているかをカイ 2 乗検定にて検定する。

入力	$\{X_i\}$	:	0 と 1 の中から値をとる乱数列
出力	p-value	:	カイ 2 乗統計量からの p-value（2 個）
処理内容	ステップ 1	:	5 個の数を組にしたときの大小関係を $5! = 120$ 個のクラスに分ける。
	ステップ 2	:	X_i を列の先頭から順に 32 ビットずつ取り出し、32 ビットの整数からなる新しい列 $\{Y_i\}$ を作り、Y_i を 5 個ずつ組にする。
	ステップ 3	:	ステップ 2 を 1,000,000 回くり返し、組の大小関係に応じてクラスの度数に加えていく。各クラスの度数を $f_0, f_1, \cdots, f_{119}$ とする。5 個組は、$(Y_1, Y_2, Y_3, Y_4, Y_5), (Y_2, Y_3, Y_4, Y_5, Y_6)$ のように 32 ビットずつずらして重ねながら作っていく。
	ステップ 4	:	f と共分散行列からカイ 2 乗統計量 χ_0^2 を計算する。χ_0^2 は自由度が共分散行列のランクと等しい 99 の χ^2 分布に従うので、この分布から p-value を計算する。
	ステップ 5	:	ステップ 3、4 を 2 回くり返す。

3.1.12 (6×8) の 2 値行列ランク検定

0 と 1 からなる乱数列の 156 ビットの部分列から (6×8) の 2 値行列を構成し、各部分列を行列のランクに応じて 3 個のクラスに割り当て、その度数をカイ 2 乗検定にて検定しランクの偏りを調べる。

入力　$\{X_i\}$　：　0 と 1 の中から値をとる乱数列

出力　p-value　：　カイ 2 乗統計量からの p-value（25 個）
出力　p-value　：　KS 検定による p-value

処理内容　ステップ 1　：　行列のランクを、4 以下、5, 6 の 3 個のクラスに分ける。
ステップ 2　：　X_i を列の先頭から順に 32 ビットずつに区切る。
ステップ 3　：　各 32 ビットから 8 ビットを取り出し、8 ビットの整数からなる新しい列 $\{Y_i\}$ を作る。
ステップ 4　：　連続する 6 個の Y_i を順に並べて $GF(2)$ 上の 6×8 行列を作る。6 個の組は $(Y_1, Y_2, \cdots, Y_6), (Y_7, Y_8, \cdots, Y_{12})$ のように重ね合わせずに作っていく。
ステップ 5　：　ステップ 4 で得られた行列のランクを計算し、ランクに応じてクラスの度数に加えていく。各クラスの度数を f_0, f_1, f_2 とする。
ステップ 6　：　ステップ 2〜5 を 100,000 回くり返す。
ステップ 7　：　カイ 2 乗統計量 χ_0^2 を計算する。各クラスの確率の値は、次の表の値を用いる。χ_0^2 は自由度 2 の χ^2 分布に従うので、この分布から p-value を計算する。

〜4	5	6
0.009443	0.217439	0.773118

ステップ 8　：　ステップ 2〜7 を 25 回くり返す。各回の違いは、32 ビットから取り出す 8 ビットの位置で、1 回目は 1 ビット目から 8 ビット、2 回目は 2 ビット目から 8 ビット、⋯、25 回目は 25 ビット目から 8 ビットと行う。
ステップ 9　：　ステップ 8 で得られた p-value に対して、$[0, 1)$ に一様に分布しているかどうか KS 検定を行い、その p-value を計算する。

3.1.13　(31×31) の 2 値行列ランク検定

0 と 1 からなる乱数列の 1024 ビットの部分列から (31×31) の 2 値行列を構成し、各部分列を行列のランクに応じて 3 個のクラスに割り当て、その度数をカイ 2 乗検定にて検定しランクの偏りを調べる。

入力　$\{X_i\}$　：　0 と 1 の中から値をとる乱数列

出力　p-value　：　カイ 2 乗統計量からの p-value

処理内容　ステップ 1　：　行列のランクを、28 以下、29, 30, 31 の 4 個のクラスに分ける。

ステップ 2　：　X_i を列の先頭から順に 32 ビットずつに区切る。

ステップ 3　：　各 32 ビットから LSB1 ビットを除いた 31 ビットを取り出し、31 ビットの整数からなる新しい列 $\{Y_i\}$ を作る。

ステップ 4　：　連続する 31 個の Y_i を順に並べて $GF(2)$ 上の 31×31 行列を作る。31 個の組は $(Y_1, Y_2, \cdots, Y_{31}), (Y_{32}, Y_{33}, \cdots, Y_{62})$ のように重ね合わせずに作っていく。

ステップ 5　：　ステップ 4 で得られた行列のランクを計算し、ランクに応じてクラスの度数に加えていく。各クラスの度数を f_0, f_1, f_2, f_3 とする。

ステップ 6　：　ステップ 2〜5 を 40,000 回くり返す。

ステップ 7　：　カイ 2 乗統計量 χ_0^2 を計算する。各クラスの確率の値は、次の表の値を用いる。χ_0^2 は自由度 3 の χ^2 分布に従うので、この分布から p-value を計算する。

〜28	29	30	31
0.0052854502	0.1283502644	0.5775761902	0.2887880952

3.1.14　(32×32) の 2 値行列ランク検定

0 と 1 からなる乱数列の 1024 ビットの部分列から (32×32) の 2 値行列を構成し、各部分列を行列のランクに応じてを 3 個のクラスに割り当て、その度数をカイ 2 乗検定にて検定しランクの偏りを調べる。

入力　$\{X_i\}$　：　0 と 1 の中から値をとる乱数列

出力　p-value　：　カイ 2 乗統計量からの p-value

処理内容　ステップ 1　：　行列のランクを、29 以下、30, 31, 32 の 4 個のクラスに分ける。

ステップ 2　：　X_i を列の先頭から順に 32 ビットずつ取り出し、32 ビットの整数からなる新しい列 $\{Y_i\}$ を作る。

ステップ 3　：　連続する 32 個の Y_i を順に並べて $GF(2)$ 上の 32×32 行列を作る。32 個の組は $(Y_1, Y_2, \cdots, Y_{32}), (Y_{33}, Y_{33}, \cdots, Y_{64})$ のように重ね合わせずに作っていく。

ステップ 4　：　ステップ 3 で得られた行列のランクを計算し、ランクに応じてクラスの度数に加えていく。各クラスの度数を f_0, f_1, f_2, f_3 とする。

ステップ 5　：　ステップ 2〜4 を 40,000 回くり返す。

ステップ 6　：　カイ 2 乗統計量 χ_0^2 を計算する。各クラスの確率の値は、次の表の値を用いる。χ_0^2 は自由度 3 の χ^2 分布に従うので、この分布から p-value を計算する。

〜28	29	30	31
0.0052854502	0.1283502644	0.5775761902	0.2887880952

NIST のツールでもこの検定を行うが、ステップ 5 の繰り返し回数が行っている。

3.1.15　ブロック単位の最長連検定

0 と 1 からなる乱数列を M ビット単位で分割し、最長連の長さに応じて各部分列を 7 個のクラスに割り当てその度数をカイ 2 乗検定にて検定し最長連の長さの偏りを調べる。なお、M は乱数列の長さによって決まる。

入力　$\{X_i\}$　：　0 と 1 の中から値をとる乱数列
　　n　：　乱数列の長さ（乱数の個数）

出力　p-value　：　カイ 2 乗統計量からの p-value

処理内容　ステップ 1　：　乱数列を長さ M のブロックに分割する。ただし、ブロックサイズは、$128 \leq n < 6272$ のときは $M = 8$、$6272 \leq n < 750,000$ のときは $M = 128$、$750,000 \leq n$ のときは $M = 10000$ とする。

ステップ 2　：　$M = 8$ のときは、各ブロックを 1 の最長連の長さに応じて、f_0 から f_3 の $K = 4$ 個のクラスに分割し、各クラスの度数を求める。クラスに対応する長さおよびクラスの確率は

クラス	f_0	f_1	f_2	f_3
度数	〜1	2	3	4〜
確率	0.2148	0.3672	0.2305	0.1875

となる。また、$M = 128$ のときは、以下のような $K = 6$ 個のクラスに割り当てる。

クラス	f_0	f_1	f_2	f_3	f_4	f_5
度数	〜4	5	6	7	8	9〜
確率	0.1174	0.2430	0.2493	0.1752	0.1027	0.1124

となる。また、$M = 10000$ のときは、以下のような $K = 7$ 個のクラスに割り当てる。

クラス	f_0	f_1	f_2	f_3	f_4	f_5	f_6
度数	〜10	11	12	13	14	15	16〜
確率	0.0882	0.2092	0.2483	0.1933	0.1208	0.0675	0.0727

ステップ 3　：　カイ 2 乗統計量 $\chi^2(obs) = \sum_{i=0}^{K} \frac{(f_i - N\pi_i)^2}{N\pi_i}$ を求める。

ステップ 4　：　$\chi^2(obs)$ は、自由度 $K-1$ の χ^2 分布に従うので、この分布から p-value を計算する。

FIPS 140-2 の最長連検定は、長さ 20000 ビットの 1 ブロックに対する検定である。

3.1.16　ブロック単位の度数検定

0 と 1 からなる乱数列を m ビット単位で分割し、各部分列の 1 の出現頻度の偏りを調べる。

入力　$\{X_i\}$　：　0 と 1 の中から値をとる乱数列
　　n　：　乱数列の長さ（乱数の個数）
　　M　：　ブロックの長さ

出力　p-value　：　カイ 2 乗統計量からの p-value

処理内容　ステップ 1　：　乱数列を長さ M のブロックに分割し、$N = \lfloor n/M \rfloor$ とする。
　　ステップ 2　：　ブロック i の 1 の度数 $\pi_i = \dfrac{\sum_{j=1}^{M} X_{(i-1)M+j}}{M}$ を求める。
　　ステップ 3　：　カイ 2 乗統計量 $\chi^2(obs) = 4M \sum_{i=0}^{M} (\pi_i - 1/2)^2$ を求める。
　　ステップ 4　：　$\chi^2(obs)$ は、自由度 N の χ^2 分布に従うので、この分布から p-value を計算する。

3.2　パターンの出現に関する検定

3.2.1　連の検定

区間 $[0, 1)$ 上を分布する乱数列を上昇連、下降連で分割し、部分列の長さでクラスを定義する。部分列を長さに応じてクラスに割り当て、その度数をカイ 2 乗検定にて検定し、連の偏りを調べる。なお、乱数列が 0 と 1 からなる乱数の場合は、区間 $[0, 1)$ 上に分布する乱数列への変換が必要である。

入力　$\{X_i\}$　：　0 と 1 の中から値をとる乱数列

出力　$p-$value　：　KS 検定による p-value（2 個）

処理内容　ステップ 1　：　連の長さを 0, 1, 2, 3, 4, 5 以上の 6 個のクラスに分ける。
　　ステップ 2　：　X_i を列の先頭から順に 32 ビットずつ取り出し、32 ビットの整数からなる新しい列 $\{Y_i\}$ を作る。
　　ステップ 3　：　ステップ 2 の Y_i に対し、$v_i = Y_i/2^{32}$ とおき、小数列 $\{v_i\}$ を作る。
　　ステップ 4　：　$\{v_i\}$ に対して v_1 から順に上昇連の長さをカウントし、長さに応じてクラスの度数に加えていく。クラスの度数を $f_{u,0}, f_{u,1}, \cdots, f_{u,5}$ とおく。下降連についても同様に長さをカウントし、クラスの度数を $f_{d,0}, f_{d,1}, \cdots, f_{d,5}$ とおく。
　　ステップ 5　：　f_u, f_d それぞれについて共分散行列からカイ 2 乗統計量 $\chi^2_{u,0}, \chi^2_{d,0}$ を計算する。$\chi^2_{u,0}, \chi^2_{d,0}$ はそれぞれ自由度 5 の χ^2 分布に従うので、この分布からそれぞれ p-valuep_u, p_d を計算する。
　　ステップ 6　：　ステップ 4, 5 を 10 回くり返し、得られた 10 個の p-value p_u, p_d に対して、$[0, 1)$ に一様に分布しているかどうかそれぞれ KS 検定を行い、その p-value を計算する。

3.2.2 衝突検定

乱数列の k 個の文字の組み合わせを k 次元座標上の点とみなし、セルに配置していく。衝突回数でクラスを定義し、すでに点が入っているところに入ったら、衝突として衝突回数を増やして行き、セルを衝突回数に応じてクラスに割り当て、その度数をカイ 2 乗検定にて検定し衝突回数の偏りを調べる。

入力	$\{u_{i,j}\}$	:	複数の乱数列
	n	:	各乱数列の長さ（乱数の個数）
	b	:	乱数列の個数
出力	χ_0^2	:	カイ 2 乗統計量
処理内容	ステップ 1	:	衝突の起こった回数の値を ν 個のクラスの分ける。例えば $0 \sim 4, 5 \sim 10, 11 \sim 15, 15$ 以上の 4 個に分ける。
	ステップ 2	:	k 次元単位超立方体を d^k 個の等体積のセルに分割する。
	ステップ 3	:	$u_{i,j}$ を順に連続する k 個ずつの組 $(u_{i,km}, u_{i,km+1}, \cdots, u_{i,km+k-1})$ とする。n が k の倍数でない場合は、最後の半端なビットは捨てる。
	ステップ 4	:	k 個の数字の組み合わせを k 次元座標上の点と思い、セルに配置していく。すでに点が入っているところに入ったら、衝突として衝突回数に 1 を加える。この結果、衝突回数 N_i を得る。
	ステップ 5	:	ステップ 3, 4 を b 回繰り返す（i を動かす）。衝突回数 $N_1, N_2, \cdots, N_b$ が得られる。
	ステップ 6	:	$N_1, N_2, \cdots, N_b$ をステップ 1 で定めたクラスに分ける。各クラスに入った個数を $n_0, n_1, \cdots, n_\nu$ とおく。
	ステップ 7	:	衝突回数 N の確率分布を $Pr[N=c] = \frac{d^k(d^k-1)\cdots(d^k-n/k+c+1)}{d^n} \left\{ \begin{array}{c} n/k \\ n/k-c \end{array} \right\}$ として p_i を計算し、カイ 2 乗統計量 χ_0^2 を計算する。χ_0^2 は ν 個の和で、自由度 $\nu - 1$ の χ^2 分布に従う。

衝突検定は、駐車場検定、最小距離検定、3DSPHERES 検定のベースとなる検定であるため、ミニマムセットの候補には加えない。

3.2.3 駐車場検定

0 と 1 からなる乱数列を 32 ビット単位で分割し、各 32 ビット Y を $u = 100Y/2^{32}$ と変換する。変換した乱数列の 2 文字の組み合わせを 2 次元座標上の点とみなし、12000 個の点をプロットしていく。プロットした点とこれまでにプロットされた点との距離を計算し、すべての点との距離が 1 より大きいときは、成功として成功回数の偏りを調べる。

入力	$\{X_i\}$	:	0 と 1 の中から値をとる乱数列
出力	p-value	:	カイ 2 乗統計量からの p-value（10 個）
	p-value	:	KS 検定による p-value
処理内容	ステップ 1	:	X_i を列の先頭から順に 32 ビットずつ取り出し、32 ビットの整数からなる新しい列 $\{Y_i\}$ を作り、2 個ずつ組にする。
	ステップ 2	:	ステップ 1 の 2 つ組を 12,000 個作る。2 つ組は、$(Y_1, Y_2), (Y_3, Y_4)$ のように重ね合わせずに作っていく。
	ステップ 3	:	ステップ 2 で得られた 12,000 個の 2 つ組 (Y_{2i-1}, Y_{2i}) に対し、$(v_{2i-1}, v_{2i}) = (100Y_{2i-1}/2^{32}, 100Y_{2i}/2^{32})$ とおく。
	ステップ 4	:	成功回数 $s = 0$ とする。
	ステップ 5	:	ステップ 3 で得られた 2 つ組を 2 次元の点の座標とみなし、$(v_1, v_2), (v_3, v_4)$ と順に 2 次元座標空間に点をプロットしていく。その回にプロットする点と、今までにプロットされた点すべての組み合わせの距離を計算し、どの距離も 1 より真に大きいとき成功として成功回数 s に 1 を加えていく。s は平均 3523、標準偏差 21.9 の正規分布に従うので、この分布から p-value を計算する。
	ステップ 6	:	ステップ 2〜5 を 10 回くり返し、得られた 10 個の p-value に対して、$[0, 1)$ に一様に分布しているかどうか KS 検定を行い、その p-value を計算する。

3.2.4 最小距離検定

0 と 1 からなる乱数列を 32 ビット単位で分割し、各 32 ビット Y を $u = 10000Y/2^{32}$ と変換する。変換した乱数列の 2 文字の組み合わせを 2 次元座標上の点とみなし、8000 個の点をプロットしていく。すべての点の組み合わせの中から距離が最小となる 2 点の距離を計算し、最小距離の偏りを調べる。

入力	$\{X_i\}$	:	0 と 1 の中から値をとる乱数列
出力	p-value	:	カイ 2 乗統計量からの p-value（20 個）
	p-value	:	KS 検定による p-value
処理内容	ステップ 1	:	X_i を列の先頭から順に 32 ビットずつ取り出し、32 ビットの整数からなる新しい列 $\{Y_i\}$ を作り、2 個ずつ組にする。
	ステップ 2	:	ステップ 1 の 2 つ組を 8,000 個作る。2 つ組は、$(Y_1, Y_2), (Y_3, Y_4)$ のように重ね合わせずに作っていく。
	ステップ 3	:	ステップ 2 で得られた 8,000 個の 2 つ組 (Y_{2i-1}, Y_{2i}) に対し、$(v_{2i-1}, v_{2i}) = (10000Y_{2i-1}/2^{32}, 10000Y_{2i}/2^{32})$ とおく。
	ステップ 4	:	ステップ 3 で得られた 2 つ組を 2 次元の点の座標とみなし、すべての点の組み合わせの中から距離が最小となる 2 点の距離 d を求める。d は平均 0.995 の指数分布に従うので、この分布から p-value を計算する。
	ステップ 5	:	ステップ 2〜4 を 100 回くり返し、得られた 100 個の p-value に対して、$[0, 1)$ に一様に分布しているかどうか KS 検定を行い、その p-value を計算する。

3.2.5 3DSPHERES 検定

0 と 1 からなる乱数列を 32 ビット単位で分割し、各 32 ビット Y を $u = 1000Y/2^{32}$ と変換する。変換した乱数列の 3 文字の組み合わせを 3 次元座標上の点とみなし、4000 個の点をプロットしていく。すべての点の組み合わせの中から距離が最小となる 2 点の距離を計算し、最小距離の偏りを調べる。

入力　$\{X_i\}$　：　0 と 1 の中から値をとる乱数列

出力　p-value　：　カイ 2 乗統計量からの p-value（20 個）
　　p-value　：　KS 検定による p-value

処理内容　ステップ 1　：　X_i を列の先頭から順に 32 ビットずつ取り出し、32 ビットの整数からなる新しい列 $\{Y_i\}$ を作り、3 個ずつ組にする。
　　ステップ 2　：　ステップ 1 の 3 つ組を 4,000 個作る。3 つ組は、$(Y_1, Y_2, Y_3), (Y_4, Y_5, Y_6)$ のように重ね合わせずに作っていく。
　　ステップ 3　：　ステップ 2 で得られた 4,000 個の 3 つ組 $(Y_{3i-2}, Y_{3i-1}, Y_{3i})$ に対し、$(v_{3i-2}, v_{3i-1}, v_{3i}) = (1000Y_{3i-2}/2^{32}, 1000Y_{3i-1}/2^{32}, 10000Y_{3i}/2^{32})$ とおく。
　　ステップ 4　：　ステップ 3 で得られた 3 つ組を 3 次元の点の座標とみなし、すべての点の組み合わせの中から距離が最小となる 2 点の距離 d を求める。半径 d の球の体積は平均 $120\pi/3$ の指数分布に従うので、この分布から p-value を計算する。
　　ステップ 5　：　ステップ 2〜4 を 20 回くり返し、得られた 20 個の p-value に対して、$[0, 1)$ に一様に分布しているかどうか KS 検定を行い、その p-value を計算する。

3.2.6　ビット列検定

0 と 1 からなる乱数列から 20 ビットの数値を 2^{21} 個作り、一度も出現しなかった数値の個数の偏りを調べる。なお、20 ビットの数値は、1 つ目が $(X_1, X_2, \cdots, X_{20})$、2 つ目が $(X_2, X_3, \cdots, X_{21})$ のように 1 ビットずつずらして重ねながら作っていく。

入力　$\{X_i\}$　：　0 と 1 の中から値をとる乱数列

出力　p-value　：　正規分布統計量からの p-value (20 個)

処理内容　ステップ 1　：　X_i を列の先頭から順に 20 ビットずつ取り出し、20 ビットの整数からなる新しい列 $\{Y_i\}$ を作る。
　　ステップ 2　：　ステップ 1 の列 $\{Y_i\}$ を $i = 2^{21}$ まで作る。Y_1 は $(X_1, X_2, \cdots, X_{20})$、$Y_2$ は $(X_2, X_3, \cdots, X_{21})$ のように 1 ビットずつずらして重ねながら作っていく。
　　ステップ 3　：　ステップ 2 で作った 2^{21} 個の中に、20 ビットの数値として現れていない数値の個数 s を数える。s は平均 141,909、標準偏差 428 の正規分布に従うので、この分布から p-value を計算する。

3.2.7 OPSO 検定

0 と 1 からなる乱数列を 32 ビット単位で分割し、各 32 ビットから 10 ビット Y を取り出した新たな列 $Y_1, Y_2, \cdots$ を作る。$(Y_1, Y_2), (Y_2, Y_3), \cdots$ と連続する 2 つの 10 ビットの組を 2^{21} 個作り、一度も出現しなかった 2 つの 10 ビットの組の個数の偏りを調べる。

入力	$\{X_i\}$	:	0 と 1 の中から値をとる乱数列
出力	p-value	:	正規分布統計量からの p-value（23 個）
処理内容	ステップ 1	:	X_i を列の先頭から順に 32 ビットずつに区切る。
	ステップ 2	:	各 32 ビットから 10 ビットを取り出し、10 ビットの整数からなる新しい列 $\{Y_i\}$ を作り、Y_i を 2 個ずつ組にする。
	ステップ 3	:	ステップ 2 の組を 2^{21} 組作る。2 つ組は、$(Y_1, Y_2), (Y_2, Y_3)$ のように 10 ビットずつずらして重ねながら作っていく。
	ステップ 4	:	ステップ 3 で作った 2^{21} 組の中に、10 ビットの 2 つ組として現れていない組の個数 s を数える。s は平均 141,909、標準偏差 290 の正規分布に従うので、この分布から p-value を計算する。
	ステップ 5	:	ステップ 1〜4 を 23 回くり返す。各回の違いは、32 ビットから取り出す 10 ビットの位置で、1 回目は 23 ビット目から 10 ビット、2 回目は 22 ビット目から 10 ビット、⋯、23 回目は 1 ビット目から 10 ビットと行う。

3.2.8 OQSO 検定

0 と 1 からなる乱数列を 32 ビット単位で分割し、各 32 ビットから 5 ビット Y を取り出した新たな列 $Y_1, Y_2, \cdots$ を作る。$(Y_1, Y_2, Y_3, Y_4), (Y_2, Y_3, Y_4, Y_5), \cdots$ と連続する 4 つの 5 ビットの組を 2^{21} 個作り、一度も出現しなかった 4 つの 5 ビットの組の個数の偏りを調べる。

入力　$\{X_i\}$　：　0 と 1 の中から値をとる乱数列

出力　p-value　：　正規分布統計量からの p-value（28 個）

処理内容　ステップ 1　：　X_i を列の先頭から順に 32 ビットずつに区切る。
ステップ 2　：　各 32 ビットから 5 ビット取り出し、5 ビットの整数からなる新しい列 $\{Y_i\}$ を作り、Y_i を 4 個ずつ組にする。
ステップ 3　：　ステップ 2 の 4 つ組を 2^{21} 組作る。$(Y_1, Y_2, Y_3, Y_4), (Y_2, Y_3, Y_4, Y_5)$ のように 5 ビットずつずらして重ねながら作っていく。
ステップ 4　：　ステップ 3 で作った 2^{21} 組の中に、5 ビットの 4 つ組として現れていない組の個数 s を数える。s は平均 141,909、標準偏差 295 の正規分布に従うので、この分布から p-value を計算する。
ステップ 5　：　ステップ 1〜4 を 28 回くり返す。各回の違いは、32 ビットから取り出す 5 ビットの位置で、1 回目は 28 ビット目から 5 ビット、2 回目は 27 ビット目から 5 ビット、$\cdots$、28 回目は 1 ビット目から 5 ビットと行う。

3.2.9 DNA 検定

0 と 1 からなる乱数列を 32 ビット単位で分割し、各 32 ビットから 2 ビット Y を取り出した新たな列 $Y_1, Y_2, \cdots$ を作る。$(Y_1, Y_2, \cdots, Y_{10}), (Y_2, Y_3, \cdots, Y_{11}), \cdots$ と連続する 10 個の 2 ビットの組を 2^{21} 個作り、一度も出現しなかった 10 個の 2 ビットの組の個数の偏りを調べる。

入力　$\{X_i\}$　：　0 と 1 の中から値をとる乱数列

出力　p-value　：　正規分布統計量からの p-value（31 個）

処理内容　ステップ 1　：　X_i を列の先頭から順に 32 ビットずつに区切る。
ステップ 2　：　各 32 ビットから 2 ビット取り出し、2 ビットの整数からなる新しい列 $\{Y_i\}$ を作り、Y_i を 10 個ずつ組にする。
ステップ 3　：　ステップ 2 の 10 個組を 2^{21} 組作る。10 個組は、$(Y_1, Y_2, \cdots, Y_{10}), (Y_2, Y_3, \cdots, Y_{11})$ のように 2 ビットずつずらして重ねながら作っていく。
ステップ 4　：　ステップ 3 で作った 2^{21} 組の中に、2 ビットの 10 個組として現れていない組の個数 s を数える。s は平均 141,909、標準偏差 339 の正規分布に従うので、この分布から p-value を計算する。
ステップ 5　：　ステップ 1〜4 を 31 回くり返す。各回の違いは、32 ビットから取り出す 2 ビットの位置で、1 回目は 31 ビット目から 2 ビット、2 回目は 30 ビット目から 2 ビット、$\cdots$、31 回目は 1 ビット目から 2 ビットと行う。

3.2.10 札集め検定

0 から $d-1$ の値を取る乱数列において、すべての文字がそろったところで部分列に分割するという処理を n 回行い、部分列の長さの頻度を求め、文字の出現の偏りを調べる。

入力	$\{U_i\}$	:	乱数列
	n	:	観測する部分列の数
	t	:	部分列の長さの上限
出力	χ_0^2	:	カイ 2 乗統計量
処理内容	ステップ 1	:	部分列の長さを $d, d+1, \cdots, , t-2, t-1$ および t 以上の $t-d+1$ 個のクラスに分ける。
	ステップ 2	:	各部分列に 0 から $d-1$ のすべての文字が含まれるように $\{U_i\}$ を分割する。
	ステップ 3	:	ステップ 2 で得られた n 個の部分列の長さに応じてクラスの度数を加えていく。各クラスの度数を $f_d, f_{d+1}, \cdots, f_t$ とする。
	ステップ 4	:	$p_i = \frac{d!}{d^i} \left\{ \begin{array}{c} i-1 \\ d-1 \end{array} \right\} (d \leq i < t)$, $p_t = 1 - \frac{d!}{d^{t-1}} \left\{ \begin{array}{c} t-1 \\ d \end{array} \right\}$ として、カイ 2 乗統計量 χ_0^2 を計算する。χ_0^2 は、自由度 $t-d$ の χ^2 分布に従う。

以下のような 0 と 1 からなる乱数列を考える。この場合、

$$1,0,1,1,0,0,0,0,0,1,0,0,1,1,1,0,0,0,\cdots$$

$$\Downarrow \text{(全部がそろうところで区切る)}$$

$$1,0\Big|1,1,0\Big|0,0,0,1\Big|0,0,1\Big|1,1,0\Big|0,0,\cdots$$

$$\Downarrow \text{(区切りをひとつ手前に移動)}$$

$$1\Big|0,1,1\Big|0,0,0,0\Big|1,0,0\Big|1,1,1\Big|0,0,0,\cdots$$

$$\Downarrow \text{(部分列がすべて同じ文字になるようにさらに区切る)}$$

$$1\Big|0\Big|1,1\Big|0,0,0,0\Big|1\Big|0,0\Big|1,1,1\Big|0,0,0,\cdots$$

と分割できるので、これは連の検定と同じである。なら、区切りを 1 つ手前にシフトさせると、長さ i の部分列は、

- 半分は、長さ i の連
- 半分は、長さ $i-1$ の連と長さ 1 の連

と考えることができる。この検定は、0 と $d-1$ の中から値を取る乱数列に対しては、すべての文字が集まるまでのブロックの長さの偏りを調べるという点で有効であるが、0 と 1 の中から値を取る乱数列の場合は、すべての文字が集まるまでのブロックの長さは連長に一致するため、0 と 1 の

中から値を取る乱数列に対しては札集め検定を行う意味がない。また、0 と 1 の中から値を取る乱数列を 0 と $d-1$ の中から値を取る乱数列に変換し札集め検定を行うことを考えた場合、検定結果は変換方法に大きく依存するため今回の調査では 4 章のミニマムセットの対象からは除外する。なお、DIEHARD ではビット列検定や OPSO 検定、OQSO 検定などのように 0 と 1 の中から値を取る乱数列を変換し、一度も出現しない文字を調べる検定もある。

3.2.11 間隔検定

0 から $d-1$ の値を取る乱数列において、ある文字に注目して、その文字の出現する間隔の偏りを調べる。

入力	$\{U_i\}$	:	乱数列
	n	:	乱数列の長さ（乱数の個数）
出力	χ_0^2	:	カイ 2 乗統計量
処理内容	ステップ 1	:	注目する 1 つの数値を選ぶ。ここでは 0 とする。
	ステップ 2	:	間隔の値を ν 個のクラスに分ける。例えば $0, 1, 2, \cdots, 9$ および 10 以上の 11 個に分ける。
	ステップ 3	:	注目した数値の間隔を数え、各クラスの度数に加えていく。各クラスの合計の値をそれぞれ $n_0, n_1, \cdots, n_{\nu-1}$ とおく。ここで間隔を数える際には U_i のインデックス i は $\bmod n$ でサイクリックになっているとみなす。つまり U_{n-1} の次は U_0 が続いているものとする。
	ステップ 4	:	各 i に対し、$p_i = (1-1/d)^i(1/d)$ とおく。ただしこれはクラスが間隔の値が $i1$ つだけの場合で、いくつかの間隔の値を 1 まとめにしたクラスに対しては、確率を変える必要がある。例えば間隔の長さが j 以上のものを 1 つのクラスにした場合、対応する確率は $(1-1/d)^j$ となる。
	ステップ 5	:	カイ 2 乗統計量 χ_0^2 を計算する。χ_0^2 は ν 個の和で、自由度 $\nu-1$ の χ^2 分布に従う。

以下のような 0 と 1 からなる乱数列を考える。この場合、

$$1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, \cdots$$

$$\Downarrow \text{(0 の間隔を調べるため 0 の前後を区切る)}$$

$$1\Big|0\Big|1, 1\Big|0\Big|0\Big|0\Big|0\Big|1\Big|0\Big|0\Big|1, 1, 1\Big|0\Big|0\Big|0\Big|\cdots$$

$$\Downarrow \text{(間隔が 0 に対応する区切りを除く (0 の後が 0 になっている部分))}$$

$$1\Big|0\Big|1, 1\Big|0, 0\Big|0, 0\Big|1\Big|0, 0\Big|1, 1, 1\Big|0, 0, 0\Big|\cdots$$

と分割できるので、文字 1 に対しても同様の処理を行えば、連の出現度数を調べていることと同じである。この検定は、0 と $d-1$ の中から値を取る乱数列に対する検定であり、0 と 1 の中から値を取る乱数列の場合は、文字が 2 種類のため間隔を調べることは上記のように連長を調べることと等しいので、0 と 1 の中から値を取る乱数列に対しては間隔検定を行う意味がない。また、0 と 1 の中から値を取る乱数列を 0 と $d-1$ の中から値を取る乱数列に変換し間隔検定を行うことを考えた場合、検定結果は変換方法に大きく依存するため今回の調査では 4 章のミニマムセットの対象からは除外する。なお、間隔を調べる検定としては、間隔検定以外には DIEHARD のバースデイ空間検定などがある。

3.2.12 バースデイ空間検定

0 と 1 からなる乱数列を 32 ビット単位で分割し、各 32 ビットから 24 ビットの Z を取り出した新たな列 $Z_1, Z_2, \cdots, Z_{1024}$ を作る。Z_i を小さい順に並べてその 1,023 個の間隔の値 $\{Y_i\}$ を求める。Y_i を数直線にプロットしていき、それまでにプロットした値と重なった回数 s を数え、その偏りを調べる。

入力	$\{X_i\}$	:	0 と 1 の中から値をとる乱数列
出力	p-value	:	ポアソン分布統計量からの p-value（9 個）
	p-value	:	KS 検定による p-value
処理内容	ステップ 1	:	X_i を列の先頭から順に 32 ビットずつに区切る。
	ステップ 2	:	各 32 ビットから 24 ビットを取り出す。
	ステップ 3	:	ステップ 2 を 1,024 回くり返し、小さい順に並べてその 1,023 個の間隔の値 $\{Y_i\}$ を求める。
	ステップ 4	:	1,023 個の間隔の値 $\{Y_i\}$ を $Y_1, Y_2, \cdots$ と順に数直線にプロットしていき、それまでにプロットした値と重なった回数 s を数える。
	ステップ 5	:	ステップ 2〜4 を 500 回くり返し、500 個の s を求める。s は $\lambda = (2^{10})^3/4 \times 2^{24} = 16$ のポアソン分布に従うので、この分布から p-value を計算する。
	ステップ 6	:	ステップ 2〜5 を 9 回くり返す。各回の違いは、32 ビットから取り出す 24 ビットの位置で、1 回目は 1 ビット目から 24 ビット、2 回目は 2 ビット目から 24 ビット、$\cdots$、9 回目は 9 ビット目から 24 ビットと行う。
	ステップ 7	:	ステップ 6 で得られた 9 個の p-value に対して、$[0,1)$ に一様に分布しているかどうか KS 検定を行い、その p-value を計算する。

3.2.13 t 個の数の最大値検定

区間 $[0,1)$ 上を分布する乱数列を長さ t の部分列に分割し、各部分列の最大値を計算し、求めた最大値の列が一様分布になっているかを調べる。

入力	$\{u_i\}$	:	区間 $[0,1)$ 上を分布する乱数列
	t	:	部分列の長さ
出力	p-value	:	KS 検定による p-value
処理内容	ステップ 1	:	$\{u_i\}$ を長さ t の部分列に分割する。
	ステップ 2	:	各部分列の最大を取り出し、新しい乱数列を作る。
	ステップ 3	:	ステップ 2 で作った乱数列が 一様に分布しているかどうか KS 検定を行い、その p-value を計算する。

この検定は、区間 $[0,1)$ 上を分布する乱数列に対する検定であり、0 と 1 から値を取る乱数列の場合は乱数列を変換しない限り適用することができず、また、検定結果は変換方法に大きく依存するため今回の調査では 4 章のミニマムセットの対象からは除外する。なお、t 個の数の最大値検定では、長さ t のブロックから最大値を取り出して新しい乱数列を作り、その一様性を調べているのに対し、重なりのある和検定ではブロックの和から新しい乱数列を作り、その一様性を調べているので、各ブロックごとに求めた１つのある値から新たな乱数列を作り、その一様性を調べるという点で２つの検定は類似の検定であるといえる。

3.2.14 重なりのある和検定

0 と 1 からなる乱数列を 32 ビット単位で分割し、各 32 ビットから新たな列 $Y_1, Y_2, \cdots$ を作る。$(Y_1, Y_2, \cdots, Y_{100}), (Y_2, Y_3, \cdots, Y_{101}), \cdots$ と連続する 100 個の組を作り、それぞれの組の和を計算し、求めた和が、一様に分布しているかを調べる。

入力　$\{X_i\}$　：　0 と 1 の中から値をとる乱数列

出力　p-value　：　KS 検定による p-value（10 個）
　　p-value　：　(上記の 10 個の p-value への)KS 検定による p-value

処理内容　ステップ 1　：　X_i を列の先頭から順に 32 ビットずつ取り出し，その 32 ビットの整数を 2^{32} で割った $[0,1)$ 内の小数からなる新しい列 $\{Y_i\}$ を作り，100 個ずつ組にする．

ステップ 2　：　ステップ 1 の 100 個組を 100 個作る．100 個組は，$(Y_1, Y_2, \cdots, Y_{100}), (Y_2, Y_3, \cdots, Y_{101})$ のように 32 ビットずつずらして重ねながら作っていく．

ステップ 3　：　ステップ 2 で得られた 100 個の 100 個組 $(Y_{100i-99}, Y_{100i-98}, \cdots, Y_{100i})$ に対し，その和 $s_i = Y_{100i-99} + Y_{100i-98} + \cdots + Y_{100i}$ を計算する．

ステップ 4　：　ステップ 3 で得られた 100 個の s_i に対し，$y_i = (s_i - 50) \times \sqrt{12}$ を計算する．

ステップ 5　：　ステップ 4 で得られた 100 個の y_i に対し，次のようにして x_i に線形変換する．各 x_i は標準正規分布に従うので、この分布からそれぞれの p-value を計算する．

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{100} \end{pmatrix} = C \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{100} \end{pmatrix}$$

ここで行列 $C = (c_{i,j})$ は次の値である．

$$c_{1,1} = \frac{1}{\sqrt{100}}$$
$$c_{2,1} = -\frac{1}{\sqrt{100}} \times \frac{99}{\sqrt{199}}, \qquad c_{2,2} = \sqrt{\frac{100}{199}}$$
$i \geq 3$ に対しては
$$c_{i,1} = \frac{1}{\sqrt{(202-i)(402-2i)}}, \quad c_{i,i-1} = -\sqrt{\frac{201-i}{404-2i}}, \quad c_{i,i} = \sqrt{\frac{202-i}{402-2i}}$$
上記以外の (i,j) については
$c_{i,j} = 0$.

ステップ 6　：　ステップ 5 で得られた 100 個の p-value に対して，$[0,1)$ に一様に分布しているかどうか KS 検定を行い，その p-value を計算する．

ステップ 7　：　ステップ 2〜6 を 10 回くり返し，得られた 10 個の p-value に対して，$[0,1)$ に一様に分布しているかどうか KS 検定を行い，その p-value を計算する．

3.3 状態遷移・ランダムウォークに関する検定

3.3.1 累積和検定

0 と 1 からなる乱数列 $X_1, X_2, \cdots X_n$ に対し、$S_i = \sum_{j=1}^{i}(2X_j - 1)$ および $S'_i = \sum_{j=n-i+1}^{n}(2X_j - 1)$ $(1 \leq i \leq n)$ の絶対値の最大値を求め、その偏りを調べる。

入力	$\{X_i\}$	:	0 と 1 の中から値をとる乱数列
	n	:	乱数列の長さ（乱数の個数）
出力	p-value	:	正規分布統計量からの p-value(2 個)
処理内容	ステップ 1	:	$S_1 = 2X_1 - 1$ とし、$S_k = S_{k-1} + 2X_k - 1$ を求める $(2 \leq k \leq n)$。さらに、$z = \max_{1 \leq k \leq n} \lvert S_k \rvert$ を求める。[Mode 0]
	ステップ 2	:	$p - \text{value} = 1 - \sum_{k=(-n/z+1)/4}^{(n/z-1)/4} \left[\Phi\left(\frac{(4k+1)z}{\sqrt{n}}\right) - \Phi\left(\frac{(4k-1)z}{\sqrt{n}}\right) \right] + \sum_{k=(-n/z-3)/4}^{(n/z-1)/4} \left[\Phi\left(\frac{(4k+3)z}{\sqrt{n}}\right) - \Phi\left(\frac{(4k+1)z}{\sqrt{n}}\right) \right]$ を求める。ただし、Φ は標準正規分布の累積分布関数である。
	ステップ 3	:	$S_1 = 2X_n - 1$、$S_k = S_{k-1} + 2X_{n-k} - 1$ とし、ステップ 1、2 を行う。[Mode 1]

3.3.2 ランダム偏差検定

0 と 1 からなる乱数列 $X_1, X_2, \cdots X_n$ に対し、$S_i = \sum_{j=1}^{i}(2X_j - 1)$ $(1 \leq i \leq n)$ を求め、$S_i = 0$ から次に 0 になるまでを 1 つのサイクルとみなし、-4〜-1、および、1〜4 の 8 種類の状態ごとにサイクルの出現度数の偏りを調べる。

入力　$\{X_i\}$　：　0 と 1 の中から値をとる乱数列
　　n　：　乱数列の長さ（乱数の個数）

出力　p-value　：　カイ 2 乗統計量からの p-value(8 個)

処理内容　ステップ 1　：　$S_1 = 2X_1 - 1$ とし、$S_k = S_{k-1} + 2X_k - 1$ を求める $(2 \leq k \leq n)$。さらに、$0, S_1, S_2, \cdots, S_n, 0$ という新しい数列 S' を定める。

ステップ 2　：　0 から始まり次に現れる 0 までを 1 サイクルとし、 数列 S' からサイクルを求める。なお、数列 S' のサイクル数 J が $J < 500$ ならば、検定を中止する。

ステップ 3　：　8 種類の状態値を-4〜-1、および、1〜4 とし、各サイクルごとに状態値の出現度数を求める。

ステップ 4　：　状態値 x の出現度数が k となるサイクルの数を $\nu_k(x)$ とし、8 種類のすべての状態に対し $\nu_k(x)$ を求める $(1 \leq k \leq 5)$。ただし、$\nu_5(x)$ は状態値 x の出現度数が 5 回以上のサイクル数とする。

ステップ 5　：　8 種類のすべての状態値に対し、カイ 2 乗統計量 $\chi^2(obs) = \sum_{k=0}^{5} \dfrac{(\nu_k(x) - J\pi_k(x))^2}{J\pi_k(x)}$ を求める。ただし、

	$\pi_0(x)$	$\pi_1(x)$	$\pi_2(x)$	$\pi_3(x)$	$\pi_4(x)$	$\pi_5(x)$
$\lvert x\rvert = 1$	0.5000	0.2500	0.1250	0.0625	0.0312	0.0312
$\lvert x\rvert = 2$	0.7500	0.0625	0.0469	0.0352	0.0264	0.0791
$\lvert x\rvert = 3$	0.8333	0.0278	0.0231	0.0193	0.0161	0.0804
$\lvert x\rvert = 4$	0.0875	0.0156	0.0137	0.0120	0.0105	0.-733

である。

ステップ 6　：　$\chi^2(obs)$ は自由度 5 の χ^2 分布に従うので、この分布から 8 種類のすべての状態値に対する p-value を計算する。

3.3.3　種々のランダム偏差検定

0 と 1 からなる乱数列 $X_1, X_2, \cdots X_n$ に対し、$S_i = \sum_{j=1}^{i}(2X_j - 1)$ $(1 \leq i \leq n)$ を求め、-9〜-1、および、1〜9 の 18 種類の状態の出現度数の偏りを調べる。

入力	$\{X_i\}$	:	0 と 1 の中から値をとる乱数列
	n	:	乱数列の長さ（乱数の個数）
出力	p-value	:	正規分布統計量からの p-value(18 個)
処理内容	ステップ 1	:	$S_1 = 2X_1 - 1$ とし、$S_k = S_{k-1} + 2X_k - 1$ を求める $(2 \leq k \leq n)$。さらに、$0, S_1, S_2, \cdots, S_n, 0$ という新しい数列 S' を定める。
	ステップ 2	:	0 から始まり次に現れる 0 までを 1 サイクルとし、 数列 S' からサイクルを求める。なお、数列 S' のサイクル数を J とする。
	ステップ 3	:	18 種類の状態値を-9〜-1、および、1〜9 とし、状態値 x の出現度数 $\xi(x)$ を求める。
	ステップ 4	:	18 種類のすべての状態値に対する p-value を $p - \text{value} = erfc\left(\dfrac{\lvert\xi(x) - J\rvert}{\sqrt{2J(4\lvert x\rvert - 2)}}\right)$ により求める。ただし、$erfc$ は標準正規分布の誤差関数である。

3.4 一様性・圧縮可能性に関する検定

3.4.1 Maurer のユニバーサル統計検定

0 と 1 からなる乱数列における長さ L ビットのパターンの間隔を調べることにより乱数列の一様性・圧縮可能性を調べる。

入力　$\{X_i\}$　：　0 と 1 の中から値をとる乱数列
　　n　：　乱数列の長さ（乱数の個数）
　　L　：　ブロックの長さ
　　Q　：　初期系列のブロック数

出力　p-value　：　正規分布統計量からの p-value

処理内容　ステップ 1　：　乱数列を長さ L のブロックに分割し、先頭から Q ブロック分を初期セグメントとし、残りの K ブロック分をテストセグメントとする。ただし、$K = \lfloor (n - QL)/L \rfloor$ とする。

ステップ 2　：　T_j $(0 \leq j \leq 2^L)$ の初期値を $T_j = 0$ とし、「i 番目の L ビットブロックを 2 進数とみなしたときの値が j のときに、$T_j = i$ とする」という処理を初期セグメントの先頭ブロックから初期セグメントの最終ブロックまで順に行う $(1 \leq i \leq Q)$。

ステップ 3　：　sum の初期値を 0 とし、「i 番目の L ビットブロックを 2 進数とみなしたときの値が j のときに、$sum = sum + \log_2(i - T_j)$ とし、さらに $T_j = i$ とする」という処理をテストセグメントの先頭ブロックからテストセグメントの最終ブロックまで順に行う $(Q+1 \leq i \leq Q+K)$。また、$f_n = sum/K$ を求める。

ステップ 4　：　f_n は下記の表にある平均、分散の正規分布に従うので、この分布から p-value を計算する 。

L	平均	分散
6	5.2177052	2.954
7	6.1962507	3.125
8	7.1836656	3.238
9	8.1764248	3.311
10	9.1723243	3.356
11	10.170032	3.384
12	11.168765	3.401
13	12.168070	3.410
14	13.167693	3.416
15	14.167488	3.419
16	15.167379	3.421

3.4.2 Lempel-Ziv 圧縮検定

0 と 1 からなる乱数列において、異なる部分列の総数を Lemple-Ziv アルゴリズムで調べることにより乱数列の一様性・圧縮可能性を調べる。

入力	$\{X_i\}$	:	0 と 1 の中から値をとる乱数列
	n	:	乱数列の長さ（乱数の個数）
出力	p-value	:	正規分布統計量からの p-value
処理内容	ステップ 1	:	増分分解法により乱数列を部分列に分割し、このときの部分列の個数を W_{obs} とする。なお、増分分解法では、最後の部分列を除いたすべての部分列は異なっており、さらに、部分列はそれまでに出現した部分列に 1 文字を付加したものとなっている。
	ステップ 2	:	$n = 1,000,000$ のとき W_{obs} は平均 $\mu = 69586.25$、分散 $\sigma^2 = 70.448718$ の正規分布に従うので、この分布から p-value を計算する 。

長さ n の系列から得られる異なる部分列の数を $W(n)$ とすると、NIST のドキュメントでは、$W(n)$ の平均を

$$\lim_{n\to\infty} \frac{E[W(n)]}{n/\log n} = 1$$

が成り立つことから、

$$E[W(n)] \approx \frac{n}{\log n}$$

によって近似している。しかし、$W(n)$ の平均について、Louchard と Szpankowski は以下のようなより詳しい漸近式を示している [11]。

$$E[W(n)] = \frac{n}{\log n}\left(1 + \frac{\log\log n}{\log n} + \frac{A}{\log n} + O\left(\frac{(\log\log n)^2}{\log^2 n}\right)\right) \times \left(1 + O\left(\sqrt{\frac{\log n}{n}}\right)\right) + O(1)$$

ただし、

$$A \approx 1.5297$$

である。ここで、$n = 1,000,000$ とすると、

$$\begin{aligned} E[W(1000000)] &\approx \frac{1000000}{\log 1000000}\left(1 + \frac{\log\log 1000000}{\log 1000000} + \frac{1.5297}{\log 1000000}\right) \\ &= \frac{1000000}{\log 1000000} \times 1.29334 \end{aligned}$$

となり、実際の平均値は NIST のツールで採用している値よりも 1.29 倍だけ大きいことが分かる。これは、より正確な $W(n)$ の分布は、NIST の検定で用いている分布を右側に 29%だけ移動したものになることを意味しており、得られる p-value は正確な分布に対しては、小さすぎることになる。したがって、NIST のツールでは、パスしない乱数列の出現頻度が多いことが分かる。

一方、NIST のツールのソースコードを見ると、平均値は理論値 $\frac{n}{\log n}$ で近似しているのではなく、(上記の近似式からの値とはまったく異なる) SHA-1 からの出力が検定をパスするような値をヒューリスティックに計算し、(乱数列の長さの推奨値として 100 万以上とあるが）n が 10 万、20 万、40 万、60 万、80 万、100 万の場合のみ定めている。このため、乱数の検定としては有効性が疑問視される。

3.4.3 系列検定

0 と 1 からなる乱数列における長さ m ビットのパターン長さ $m-1$ ビットのパターン、長さ $m-2$ ビットのパターンが一様に出現しているかを調べることにより乱数列の一様性・圧縮可能性を調べる。

入力	$\{X_i\}$	:	0 と 1 の中から値をとる乱数列
	n	:	乱数列の長さ（乱数の個数）
	m	:	ブロックの長さ
出力	p-value	:	カイ 2 乗統計量からの p-value(2 個)
処理内容	ステップ 1	:	重なりのある m ビットブロックの列を $(X_1, X_2, \cdots, X_m), (X_2, X_3, \cdots, X_{m+1}) \cdots$ のように定める。同様に、重なりのある $m-1$ ビットブロックの列、重なりのある $m-2$ ビットブロックの列を定める。
	ステップ 2	:	m ビットパターン $i_1 i_2 \cdots i_m$ の出現頻度 $\nu_{i_1 i_2 \cdots i_m}$、$m-1$ ビットパターン $i_1 i_2 \cdots i_{m-1}$ の出現頻度 $\nu_{i_1 i_2 \cdots i_{m-1}}$、$m-2$ ビットパターン $i_1 i_2 \cdots i_{m-2}$ の出現頻度 $\nu_{i_1 i_2 \cdots i_{m-2}}$ を求める。
	ステップ 3	:	$\Psi_m^2 = \frac{2^m}{n} \sum_{i_1 i_2 \cdots i_m} \left(\nu_{i_1 i_2 \cdots i_m} - \frac{n}{2^m}\right)^2 = \frac{2^m}{n} \sum_{i_1 i_2 \cdots i_m} \nu_{i_1 i_2 \cdots i_m}^2 - n$、$\Psi_{m-1}^2 = \frac{2^{m-1}}{n} \sum_{i_1 i_2 \cdots i_{m-1}} \left(\nu_{i_1 i_2 \cdots i_{m-1}} - \frac{n}{2^{m-1}}\right)^2 = \frac{2^{m-1}}{n} \sum_{i_1 i_2 \cdots i_{m-1}} \nu_{i_1 i_2 \cdots i_{m-1}}^2 - n$、$\Psi_{m-2}^2 = \frac{2^{m-2}}{n} \sum_{i_1 i_2 \cdots i_{m-2}} \left(\nu_{i_1 i_2 \cdots i_{m-2}} - \frac{n}{2^{m-2}}\right)^2 = \frac{2^{m-2}}{n} \sum_{i_1 i_2 \cdots i_{m-2}} \nu_{i_1 i_2 \cdots i_{m-2}}^2 - n$ を求める。
	ステップ 4	:	$\nabla \Psi_m^2 = \Psi_m^2 - \Psi_{m-1}^2$ および $\nabla^2 \Psi_m^2 = \Psi_m^2 - 2\Psi_{m-1}^2 + \Psi_{m-2}^2$ を求める。
	ステップ 5	:	$\nabla \Psi_m^2$ は自由度 2^{m-1} の χ^2 分布に従うので、この分布から p-value を計算する。同様に、$\nabla^2 \Psi_m^2$ は自由度 2^{m-2} の χ^2 分布に従うので、この分布から p-value を計算する。

3.4.4 近似エントロピー検定

0 と 1 からなる乱数列における長さ m ビットのパターン長さ $m+1$ ビットのパターンが一様に出現しているかを調べることにより乱数列の一様性・圧縮可能性を調べる。

入力	$\{X_i\}$	:	0 と 1 の中から値をとる乱数列
	n	:	乱数列の長さ（乱数の個数）
	m	:	ブロックの長さ
出力	p-value	:	カイ 2 乗統計量からの p-value
処理内容	ステップ 1	:	重なりのある m ビットブロックの列を $(X_1, X_2, \cdots, X_m), (X_2, X_3, \cdots, X_{m+1}) \cdots$ のように定める。同様に、重なりのある $m+1$ ビットブロックの列を定める。
	ステップ 2	:	すべての m ビットパターンに対して、出現頻度 $\#i$ および $C_i^m = \#i/n$ を求める。ただし、i は m ビットブロックを 2 進数とみなしたときの値とする。さらに、$\phi^{(m)} = \sum_{i=0}^{2^m-1} \pi_i \log \pi_i$ を求める。ただし、$\pi_i = C_j^m$, $j = \log_2 i$ とする。
	ステップ 3	:	すべての $m+1$ ビットパターンに対して、出現頻度 $\#i$ および $C_i^{m+1} = \#i/n$ を求める。ただし、i は $m+1$ ビットブロックの値を 2 進数とみなしたときの値とする。さらに、$\phi^{(m+1)} = \sum_{i=0}^{2^{m+1}-1} \pi_i \log \pi_i$ を求める。ただし、$\pi_i = C_j^{m+1}$, $j = \log_2 i$ とする。
	ステップ 4	:	カイ 2 乗統計量 $\chi_0^2 = 2n[\log 2 - (\phi^{(m)} - \phi^{(m+1)})]$ を求める。
	ステップ 5	:	χ_0^2 は自由度 2^m の χ^2 分布に従うので、この分布から p-value を計算する。

3.5 周期性に関する検定

3.5.1 離散フーリエ変換検定

0 と 1 からなる乱数列を離散フーリエ変換により周波数成分に分解し、各周波数におけるピークが閾値を超える割合を求めることにより乱数列の周期性を調べる。

入力	$\{X_i\}$	:	0 と 1 の中から値をとる乱数列
	n	:	乱数列の長さ（乱数の個数）
出力	p-value	:	正規分布統計量からの p-value
処理内容	ステップ 1	:	乱数列を離散フーリエ変換し、複素数の列 S を得る。ただし、$f_j = \sum_{k=1}^{n}(2X_k - 1)\exp\{2\pi i(k-1)j/n\}$ である $(0 \le j \le n-1)$。
	ステップ 2	:	S' を S の最初の $n/2$ 個からなる部分列とし、$M = Modulus(S') \equiv \|S'\|$ を求める。
	ステップ 3	:	ピーク点の 95%の値である $T = \sqrt{3n}$ を求め、$N_0 = .95n/2$ を求める。
	ステップ 4	:	M の中で T を超えないものの数 N_1 を求め、$d = \dfrac{(N_1 - N_0)}{\sqrt{n(.95)(.05)/2}}$ を求める。
	ステップ 5	:	$\|d\|$ は標準正規分布に従うので、この分布から p-value を計算する 。

3.5.2 線形複雑度検定

0 と 1 からなる乱数列を長さ M のブロックに分割し、ブロックごとの線形複雑度を求めることにより乱数列の周期性を調べる。

入力　$\{X_i\}$　：　0 と 1 の中から値をとる乱数列
　　n　：　乱数列の長さ（乱数の個数）
　　M　：　ブロックの長さ

出力　p-value　：　カイ 2 乗統計量からの p-value

処理内容　ステップ 1　：　乱数列を長さ M のブロックに分割し、$N = \lfloor n/m \rfloor$ とする。
　　ステップ 2　：　Berlekamp-Massey アルゴリズムを用いて、ブロック i の線形複雑度 L_i を求める $(i = 1, \cdots, N)$。
　　ステップ 3　：　$\mu = \dfrac{M}{2} + \dfrac{(9 + (-1)^{M+1})}{36} + \dfrac{(M/3 + 2/9)}{2^M}$ を求め、$T_i = (-1)^M \times (L_i - \mu) + 2/9$ を求める $(i = 1, \cdots, N)$。
　　ステップ 4　：　T_i の値により 7 つのクラス $\nu_0, \nu_1, \cdots, \nu_6$ を定め、各クラスの度数を求める。

クラス	T_i の範囲
ν_0	$T_i \leq -2.5$
ν_1	$-2.5 < T_i \leq -1.5$
ν_2	$-1.5 < T_i \leq -0.5$
ν_3	$-0.5 < T_i \leq 0.5$
ν_4	$0.5 < T_i \leq 1.5$
ν_5	$1.5 < T_i \leq 2.5$
ν_6	$2.5 < T_i$

　　ステップ 5　：　カイ 2 乗統計量 χ_0^2 を計算する。クラス i の確率の値 π_i は、次の表の値を用いる。χ_0^2 は自由度 6 の χ^2 分布に従うので、この分布から p-value を計算する。

クラス	確率
π_0	0.01047
π_1	0.03125
π_2	0.125
π_3	0.5
π_4	0.25
π_5	0.0625
π_6	0.02078

3.5.3　部分数列に関する検定

乱数列全体に対して各種検定を適用するのではなく、乱数列から一定の間隔で取り出した列に対して各種検定を適用し、乱数列全体の周期性を調べる検定であり、それ自体が新しい検定方法ではない。文献 [1] にもあるように部分列が全体よりも規則的になることはまれなので、4 章のミニマムセットの対象からは除外する。

3.6 その他の検定

3.6.1 系列相関検定

系列相関検定は、U_{j+1} が U_j に依存する程度を表す系列相関係数

$$C = \frac{n(U_0U_1 + U_1U_2 + \cdots + U_{n-2}U_{n-1} + U_{n-1}U_n) - (U_0 + U_1 + \cdots + U_{n-1})^2}{n(U_0^2 + U_1^2 + \cdots + U_{n-1}^2) - (U_0 + U_1 + \cdots + U_{n-1})^2} \tag{1}$$

を求める。$-1 \leq C \leq 1$ であり、$C = 0$ のとき独立であるといえる。$< U_n >$ が一様分布のときの C の正確な分布は求められていないので、4 章のミニマムセットの対象からは除外する。

3.6.2 スペクトル検定

スペクトル検定は、線形合同法のパラメータが適切かどうかを検定する手法であり、乱数列が与えられたときに、その乱数列の性質を調べると言ったものではない。したがって、4 章のミニマムセットの対象からは除外する。

4　ミニマムセット

4.1　ミニマムセットの導出

ここでは、3 章で分類した検定の中から有効な検定を選びミニマムセットを導出する。3 章で述べた各検定には、類似の検定が多数含まれるため、乱数列を検定する上で必要最小限のものを選び出すのが目的である。根拠が明確でない検定や 0 と 1 からなる乱数列に対しては意味がない検定等は、ミニマムセットから除外する。また、乱数列の長さが十分大きいという仮定の下で理論的に正しいような検定でも、NIST のツールや DIEHARD で実際に用いている乱数列の長さで期待する結果を得ることができるかは不明なため、既存の乱数生成アルゴリズムからの出力を NIST のツールや DIEHARD で実際に検定することにより各検定の有効性を検証する。なお、本調査開発では、3 章の検定以外に高次元度数検定を新たに導入し、ミニマムセットをさらに絞り込む。

4.1.1　ブロック単位の頻度に関する検定

本調査開発では、高次元度数検定を新たに実装する。4.1 の高次元度数検定では、次元 m をパラメータとし m 次元以下の検定をすべて実行する。このため、$m \geq 5$ のときは、1 次元度数検定、2 次元度数検定、ポーカー検定を含んでいる。

表 1 は、100 万ビット ×250 本の乱数列を NIST のツールを使って検定した際の結果である。この場合、高次元度数検定では、12 次元以下のすべての次元の検定を行っている。×は、検定にパスしなかったことを意味する。なお、検定対象の乱数列は、NIST のツール付属の 16 種の乱数生成アルゴリズムを使って生成したものを使用している。

表 1:ブロック単位の頻度に関する検定の NIST ツールでの検定結果 [100 万ビット ×250 本]

生成アルゴリズム	G Using SHA-1		G Using DES		ANSI X9.17(3-DES)		Linear Congruential	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
1 次元度数検定								
連の検定 (SP800-22)								
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								
(32×32) の 2 値行列ランク検定								
ブロック単位の最長連検定								
ブロック単位の度数検定								
高次元度数検定								

生成アルゴリズム	Blum-Blum-Shub		Micali-Schonorr		Modular Exponentiation		Quadratic Congruential I	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
1 次元度数検定			×	×	×	×	×	×
連の検定 (SP800-22)								×
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								
(32×32) の 2 値行列ランク検定								
ブロック単位の最長連検定								
ブロック単位の度数検定			×	×	×	×	×	×
高次元度数検定			×	×	×	×	×	×

生成アルゴリズム	Quadratic Congruential II		Cubic Congruential		XOR	
検定名	一様性	比率	一様性	比率	一様性	比率
1 次元度数検定			×	×		
連の検定 (SP800-22)			×	×		
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					×	×
(32×32) の 2 値行列ランク検定					×	×
ブロック単位の最長連検定						
ブロック単位の度数検定			×	×		
高次元度数検定	×		×	×	×	×

表 2 は、10 万ビット ×500 本の乱数列を NIST のツールを使って検定した際の結果である。この場合、高次元度数検定では、10 次元以下のすべての次元の検定を行っている。乱数列の長さが、10 万ビットの場合には、重なりのないテンプレート適合検定、および、重なりのあるテンプレート適合検定は乱数列の長さが推奨値に満たないため実行されない。×は検定にパスしなかったことを意味する。また、E は桁あふれや桁落ち等の理由で正しい結果が得られなかったことを意味する。なお、検定対象の乱数列は、NIST のツールおよび DIEHARD 付属の 26 種の乱数生成アルゴリズムを使って生成したものを使用している。

表 2:ブロック単位の頻度に関する検定の NIST ツールでの検定結果 [10 万ビット ×500 本]

生成アルゴリズム	G Using SHA-1		G Using DES		ANSI X9.17(3-DES)		Linear Congruential	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
1 次元度数検定								
連の検定 (SP800-22)								
(32×32) の 2 値行列ランク検定								
ブロック単位の最長連検定								
ブロック単位の度数検定								
高次元度数検定								

生成アルゴリズム	Blum-Blum-Shub		Micali-Schonorr		Modular Exponentiation		Quadratic Congruential I	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
1 次元度数検定					×	×	×	
連の検定 (SP800-22)					×	×		
(32×32) の 2 値行列ランク検定								
ブロック単位の最長連検定								
ブロック単位の度数検定					×	×		
高次元度数検定					×	×	×	

生成アルゴリズム	Quadratic Congruential II		Cubic Congruential		XOR		MWC Generator	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
1 次元度数検定			×					
連の検定 (SP800-22)			×					
(32×32) の 2 値行列ランク検定					×	×		
ブロック単位の最長連検定			×					
ブロック単位の度数検定								
高次元度数検定			×		×	×		

生成アルゴリズム	MWC1616 Generator		MOTHER Generator		KISS Generator		COMBO Generator	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
1 次元度数検定								
連の検定 (SP800-22)								
(32×32) の 2 値行列ランク検定								
ブロック単位の最長連検定								
ブロック単位の度数検定								
高次元度数検定								
生成アルゴリズム	ULTRA Generator		MWC/SWB Generator		EXCONG Generator		SUPERDUPER Generator	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
1 次元度数検定								
連の検定 (SP800-22)								
(32×32) の 2 値行列ランク検定								
ブロック単位の最長連検定								
ブロック単位の度数検定								
高次元度数検定								
生成アルゴリズム	SWB Generator		RAN2 Generator		Specified shift register Generator		MSRAN Generator	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
1 次元度数検定			E	E			E	E
連の検定 (SP800-22)			E	E			E	E
(32×32) の 2 値行列ランク検定			E	E	E	E	E	E
ブロック単位の最長連検定			×				×	
ブロック単位の度数検定			E	E			E	E
高次元度数検定			E	E			E	E

生成アルゴリズム	Lagged-Fibonacci Generator		INVCONG Generator					
検定名	一様性	比率	一様性	比率				
1 次元度数検定			E	E				
連の検定 (SP800-22)			E	E				
(32×32) の 2 値行列ランク検定			E	E				
ブロック単位の最長連検定			E	E				
ブロック単位の度数検定			E	E				
高次元度数検定			E	E				

表 1,2 より、高次元度数検定は、連の検定 (SP800-22)、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等の 10 ビット以下のブロックに対する検定をカバーしていることが分かる。また、ブロック単位の最長連検定、ブロック単位の度数検定が有効であることが分かる。

表 3 は、ブロック単位の頻度に関する検定の DIEHARD での検定結果をまとめたものである。DIEHARD では、検定にパスしたかどうかは出力されず、対応する p-value のみが出力される。したがって、表 3 では、各検定ごとの p-value の中で、0.025 未満、または、0.975 を超えるもの (両側 5%) の数を記している。なお、各検定に対応する p-value の数を検定名の後のカッコ内で示している。検定対象の乱数列は、NIST のツールおよび DIEHARD 付属の 25 種の乱数生成アルゴリズムを使って生成したものを使用している。表 3 の最後に、両側 5%に含まれる p-value の合計を記している。さらに、検定ごとに対応する p-value の数が異なるため、合計を対応する p-value の数で割った値も同時に示している。

表 3:ブロック単位の頻度に関する検定の DIEHARD での検定結果 (1)

生成アルゴリズム 検定名	G Using SHA-1	G Using DES	ANSI X9.17(3-DES)	Linear Congruential
スクイーズ検定 (1)	1			
クラップス検定 (2)				
8 ビット中の文字数検定 (2)				
特定位置の 8 ビット中の文字数検定 (25)				1
OPERM5 検定 (2)			1	
(6×8) の 2 値行列ランク検定 (25)	2	1	2	2
(31×31) の 2 値行列ランク検定 (1)				
(32×32) の 2 値行列ランク検定 (1)				

生成アルゴリズム 検定名	Blum-Blum-Shub	Micali-Schonorr	Modular Exponentiation	Quadratic Congruential I
スクイーズ検定 (1)				
クラップス検定 (2)	1	1		
8 ビット中の文字数検定 (2)		2	2	1
特定位置の 8 ビット中の文字数検定 (25)	2	25	9	4
OPERM5 検定 (2)		2	1	1
(6×8) の 2 値行列ランク検定 (25)	1	3	7	2
(31×31) の 2 値行列ランク検定 (1)				
(32×32) の 2 値行列ランク検定 (1)	1			

生成アルゴリズム 検定名	Cubic Congruential	XOR	MWC Generator	MWC1616 Generator
スクイーズ検定 (1)		E		
クラップス検定 (2)		E		
8 ビット中の文字数検定 (2)		2		
特定位置の 8 ビット中の文字数検定 (25)	4	25		1
OPERM5 検定 (2)		2	1	
(6×8) の 2 値行列ランク検定 (25)		25	1	1
(31×31) の 2 値行列ランク検定 (1)		1		
(32×32) の 2 値行列ランク検定 (1)		1		

生成アルゴリズム 検定名	MOTHER Generator	KISS Generator	COMBO Generator	ULTRA Generator
スクイーズ検定 (1)				
クラップス検定 (2)				
8 ビット中の文字数検定 (2)				
特定位置の 8 ビット中の文字数検定 (25)		2		
OPERM5 検定 (2)				1
(6×8) の 2 値行列ランク検定 (25)	1		1	
(31×31) の 2 値行列ランク検定 (1)				
(32×32) の 2 値行列ランク検定 (1)				

生成アルゴリズム 検定名	MWC/SWB Generator	EXCONG Generator	SUPERDUPER Generator	SWB Generator
スクイーズ検定 (1)				
クラップス検定 (2)				1
8 ビット中の文字数検定 (2)				2
特定位置の 8 ビット中の文字数検定 (25)	2	1		
OPERM5 検定 (2)			1	
(6×8) の 2 値行列ランク検定 (25)	2			
(31×31) の 2 値行列ランク検定 (1)				1
(32×32) の 2 値行列ランク検定 (1)				

生成アルゴリズム 検定名	RAN2 Generator	Specified shift register Generator	MSRAN Generator	Lagged-Fibonacci Generator
スクイーズ検定 (1)		1		
クラップス検定 (2)		2		
8 ビット中の文字数検定 (2)	2	2	2	1
特定位置の 8 ビット中の文字数検定 (25)	3	25	1	25
OPERM5 検定 (2)		2	1	1
(6×8) の 2 値行列ランク検定 (25)	1	25	1	6
(31×31) の 2 値行列ランク検定 (1)		1		
(32×32) の 2 値行列ランク検定 (1)	1	1	1	

生成アルゴリズム 検定名	INVCONG Generator	合計	(合計)/(対応する p-value の数)	
スクイーズ検定 (1)	1	3	3	
クラップス検定 (2)	2	7	3.5	
8 ビット中の文字数検定 (2)	2	18	9	
特定位置の 8 ビット中の文字数検定 (25)	25	155	6.2	
OPERM5　検定 (2)	2	16	8	
(6×8) の 2 値行列ランク検定 (25)	25	109	4.36	
(31×31) の 2 値行列ランク検定 (1)	1	4	4	
(32×32) の 2 値行列ランク検定 (1)	1	6	6	

表 3 より、DIEHARD の検定では、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、OPERM5 検定、(32×32) の 2 値行列ランク検定が有効であることが分かる。しかし、(32×32) の 2 値行列ランク検定だけが有効である乱数生成アルゴリズムがないため、(32×32) の 2 値行列ランク検定はミニマムセットに採用しないこととする。

以上の結果より、ブロック単位の頻度に関する検定では、

- 高次元度数検定
- ブロック単位の度数検定
- ブロック単位の最長連検定
- 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
- OPERM5 検定

をミニマムセットに加える。

4.1.2　パターンの出現に関する検定

表 4 は、パターンの出現に関する検定の DIEHARD での検定結果をまとめたものである。表 4 では、各検定ごとの p-value の中で、0.025 未満、または、0.975 を超えるものの数を記している。なお、各検定に対応する p-value の数を検定名の後のカッコ内で示している。検定対象の乱数列は、NIST のツールおよび DIEHARD 付属の 25 種の乱数生成アルゴリズムを使って生成したものを使用している。表 4 の最後に、両側 5%に含まれる p-value の合計を記している。さらに、検定ごとに対応する p-value の数が異なるため、合計を対応する p-value の数で割った値も同時に示している。

表 4:パターンの出現性に関する検定の DIEHARD での検定結果

生成アルゴリズム 検定名	G Using SHA-1	G Using DES	ANSI X9.17(3-DES)	Linear Congruential
連の検定 (4)				
駐車場検定 (10)		1		
最小距離検定 (20)	1			1
3DSPHERES 検定 (20)	1	2	1	2
ビット列検定 (20)	2	3		1
OPSO 検定 (23)	2	4		1
OQSO 検定 (28)	1	1		1
DNA 検定 (31)	1		3	
バースデイ空間検定 (9)				
重なりのある和検定 (10)	1			

生成アルゴリズム 検定名	Blum-Blum-Shub	Micali-Schonorr	Modular Exponentiation	Quadratic Congruential I
連の検定 (4)				
駐車場検定 (10)		4	4	
最小距離検定 (20)	2	1		
3DSPHERES 検定 (20)	1		1	3
ビット列検定 (20)	4	20	2	1
OPSO 検定 (23)	1	23	13	12
OQSO 検定 (28)	2	28	28	6
DNA 検定 (31)	2	32	5	3
バースデイ空間検定 (9)				
重なりのある和検定 (10)				2

生成アルゴリズム 検定名	Cubic Congruential	XOR	MWC Generator	MWC1616 Generator
連の検定 (4)		E		
駐車場検定 (10)	1	10	1	1
最小距離検定 (20)	2	20	2	
3DSPHERES 検定 (20)	2	20		2
ビット列検定 (20)		20	2	
OPSO 検定 (23)	14	23	2	2
OQSO 検定 (28)	9	28	1	5
DNA 検定 (31)	5	32	2	1
バースデイ空間検定 (9)		9		
重なりのある和検定 (10)		E	1	

生成アルゴリズム 検定名	MOTHER Generator	KISS Generator	COMBO Generator	ULTRA Generator
連の検定 (4)			1	
駐車場検定 (10)	1			
最小距離検定 (20)	1	3	1	2
3DSPHERES 検定 (20)		2	2	1
ビット列検定 (20)	1		2	1
OPSO 検定 (23)	1	1		2
OQSO 検定 (28)	3			
DNA 検定 (31)	3	3	3	4
バースデイ空間検定 (9)			1	1
重なりのある和検定 (10)				

生成アルゴリズム 検定名	MWC/SWB Generator	EXCONG Generator	SUPERDUPER Generator	SWB Generator
連の検定 (4)				
駐車場検定 (10)	1			1
最小距離検定 (20)	2			2
3DSPHERES 検定 (20)	1	1		
ビット列検定 (20)	1	1		
OPSO 検定 (23)	2	1	1	2
OQSO 検定 (28)	2		1	1
DNA 検定 (31)	2	1	2	2
バースデイ空間検定 (9)	1			9
重なりのある和検定 (10)	1		1	

生成アルゴリズム 検定名	RAN2 Generator	Specified shift register Generator	MSRAN Generator	Lagged-Fibonacci Generator
連の検定 (4)		4		
駐車場検定 (10)	2	1	1	1
最小距離検定 (20)	1	1		
3DSPHERES 検定 (20)				3
ビット列検定 (20)	20	1	20	4
OPSO 検定 (23)	2	5	3	
OQSO 検定 (28)	4	28	2	3
DNA 検定 (31)	3	11	3	1
バースデイ空間検定 (9)	3	9	1	9
重なりのある和検定 (10)	1			

生成アルゴリズム 検定名	INVCONG Generator	合計	(合計)/(対応する p-value の数)	
連の検定 (4)	4	9	2.25	
駐車場検定 (10)	10	40	4	
最小距離検定 (20)	20	62	3.1	
3DSPHERES 検定 (20)	20	65	3.25	
ビット列検定 (20)	20	126	6.3	
OPSO 検定 (23)	23	140	6.09	
OQSO 検定 (28)	28	182	6.5	
DNA 検定 (31)	31	156	5.03	
バースデイ空間検定 (9)	9	52	5.78	
重なりのある和検定 (10)	10	17	1.7	

表 4 より、ビット列検定、OPSO 検定、OQSO 検定が有効であることが分かる。なお、SWB Generator や Lagged Fibonacci Generator からの出力のようにバースデイ空間検定のみがパスしない乱数列が存在するため、パターンの出現に関する検定では、

- ビット列検定
- OPSO 検定
- OQSO 検定
- バースデイ空間検定

をミニマムセットに加える。

4.1.3 状態遷移・ランダムウォークに関する検定

表 5 は、100 万ビット ×250 本の乱数列を NIST のツールを使って検定した際の結果である。× は検定にパスしなかったことを意味する。なお、検定対象の乱数列は、NIST のツール付属の 11 種の乱数生成アルゴリズムを使って生成したものを使用している。

表 5:状態遷移・ランダムウォークに関する検定の NIST ツールでの検定結果
[100 万ビット ×250 本]

生成アルゴリズム	G Using SHA-1		G Using DES		ANSI X9.17(3-DES)		Linear Congruential	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
累積和検定								
ランダム偏差検定								
種々のランダム偏差検定								
	Blum-Blum-Shub		Micali-Schonorr		Modular Exponentiation		Quadratic Congruential I	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
累積和検定			×	×	×	×	×	×
ランダム偏差検定								
種々のランダム偏差検定								
生成アルゴリズム	Quadratic Congruential II		Cubic Congruential		XOR			
検定名	一様性	比率	一様性	比率	一様性	比率		
累積和検定			×	×				
ランダム偏差検定								
種々のランダム偏差検定								

表 6 は、10 万ビット ×500 本の乱数列を NIST のツールを使って検定した際の結果である。乱数列の長さが、10 万ビットの場合には、ランダム偏差検定、および、種々のランダム偏差検定は、乱数列の長さが推奨値に満たないため実行されない。×は検定にパスしなかったことを意味する。また、E は桁あふれや桁落ち等の理由で正しい結果が得られなかったことを意味する。なお、検定対象の乱数列は、NIST のツールおよび DIEHARD 付属の 26 種の乱数生成アルゴリズムを使って生成したものを使用している。

表 6:状態遷移・ランダムウォークに関する検定の NIST ツールでの検定結果
[10 万ビット ×500 本]

生成アルゴリズム	G Using SHA-1		G Using DES		ANSI X9.17(3-DES)		Linear Congruential	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
累積和検定								
生成アルゴリズム	Blum-Blum-Shub		Micali-Schonorr		Modular Exponentiation		Quadratic Congruential I	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
累積和検定					×	×	×	
生成アルゴリズム	Quadratic Congruential II		Cubic Congruential		XOR		MWC Generator	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
累積和検定								
生成アルゴリズム	MWC1616 Generator		MOTHER Generator		KISS Generator		COMBO Generator	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
累積和検定								
生成アルゴリズム	ULTRA Generator		MWC/SWB Generator		EXCONG Generator		SUPERDUPER Generator	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
累積和検定								
生成アルゴリズム	SWB Generator		RAN2 Generator		Specified shift register Generator		MSRAN Generator	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
累積和検定			E	E	×		E	E
生成アルゴリズム	Lagged-Fibonacci Generator		INVCONG Generator					
検定名	一様性	比率	一様性	比率				
累積和検定			E	E				

表 5,6 より、状態遷移・ランダムウォークに関する検定では、乱数列の長さが短い場合でも有効な

- 累積和検定

をミニマムセットに加える。

4.1.4 一様性・圧縮可能性に関する検定

表 7 は、100 万ビット ×250 本の乱数列を NIST のツールを使って検定した際の結果である。× は検定にパスしなかったことを意味する。なお、検定対象の乱数列は、NIST のツール付属の 11 種の乱数生成アルゴリズムを使って生成したものを使用している。

表 7:一様性・圧縮可能性に関する検定の NIST ツールでの検定結果 [100 万ビット ×250 本]

生成アルゴリズム	G Using SHA-1		G Using DES		ANSI X9.17(3-DES)		Linear Congruential	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
Maurer のユニバーサル統計検定								
Lempel-Ziv 圧縮検定								
系列検定								
近似エントロピー検定								
	Blum-Blum-Shub		Micali-Schonorr		Modular Exponentiation		Quadratic Congruential I	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
Maurer のユニバーサル統計検定								
Lempel-Ziv 圧縮検定								
系列検定			×	×	×	×	×	×
近似エントロピー検定			×	×	×	×	×	
生成アルゴリズム	Quadratic Congruential II		Cubic Congruential		XOR			
検定名	一様性	比率	一様性	比率	一様性	比率		
Maurer のユニバーサル統計検定								
Lempel-Ziv 圧縮検定			×					
系列検定			×	×	×	×		
近似エントロピー検定			×	×	×	×		

表 8 は、10 万ビット ×500 本の乱数列を NIST のツールを使って検定した際の結果である。乱数列の長さが、10 万ビットの場合には、Maurer のユニバーサル統計検定、および、Lempel-Ziv 圧縮検定は、乱数列の長さが推奨値に満たないため実行されない。×は検定にパスしなかったことを意味する。また、E は桁あふれや桁落ち等の理由で正しい結果が得られなかったことを意味する。なお、検定対象の乱数列は、NIST のツールおよび DIEHARD 付属の 26 種の乱数生成アルゴリズムを使って生成したものを使用している。

表 8:一様性・圧縮可能性に関する検定の NIST ツールでの検定結果 [10 万ビット ×500 本]

生成アルゴリズム	G Using SHA-1		G Using DES		ANSI X9.17(3-DES)		Linear Congruential	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
系列検定								
近似エントロピー検定								
生成アルゴリズム	Blum-Blum-Shub		Micali-Schonorr		Modular Exponentiation		Quadratic Congruential I	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
系列検定					×	×		
近似エントロピー検定					×	×		
生成アルゴリズム	Quadratic Congruential II		Cubic Congruential		XOR		MWC Generator	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
系列検定			×	×	×	×		
近似エントロピー検定			×	×	×	×		
生成アルゴリズム	MWC1616 Generator		MOTHER Generator		KISS Generator		COMBO Generator	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
系列検定								
近似エントロピー検定								
生成アルゴリズム	ULTRA Generator		MWC/SWB Generator		EXCONG Generator		SUPERDUPER Generator	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
系列検定								
近似エントロピー検定								

生成アルゴリズム	SWB Generator		RAN2 Generator		Specified shift register Generator		MSRAN Generator	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
系列検定			E	E			E	E
近似エントロピー検定			E	E			E	E
生成アルゴリズム	Lagged-Fibonacci Generator		INVCONG Generator					
検定名	一様性	比率	一様性	比率				
系列検定			E	E				
近似エントロピー検定			E	E				

表 7,8 より、系列検定と近似エントロピー検定が有効であることが分かる。しかし、、系列検定と近似エントロピー検定の処理は類似しており、また、表 7,8 の結果を見ると、系列検定は近似エントロピー検定を完全にカバーしているため、一様性・圧縮可能性に関する検定では、

- 系列検定

をミニマムセットに加える。

4.1.5 周期性に関する検定

表 9 は、100 万ビット ×250 本の乱数列を NIST のツールを使って検定した際の結果である。× は検定にパスしなかったことを意味する。なお、検定対象の乱数列は、NIST のツール付属の 11 種の乱数生成アルゴリズムを使って生成したものを使用している。

表 9:周期性に関する検定の NIST ツールでの検定結果 [100 万ビット ×250 本]

生成アルゴリズム	G Using SHA-1		G Using DES		ANSI X9.17(3-DES)		Linear Congruential	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
離散フーリエ変換検定								
線形複雑度検定								
	Blum-Blum-Shub		Micali-Schonorr		Modular Exponentiation		Quadratic Congruential I	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
離散フーリエ変換検定								
線形複雑度検定								
生成アルゴリズム	Quadratic Congruential II		Cubic Congruential		XOR			
検定名	一様性	比率	一様性	比率	一様性	比率		
離散フーリエ変換検定	×							
線形複雑度検定					E	E		

表 10 は、10 万ビット ×500 本の乱数列を NIST のツールを使って検定した際の結果である。乱数列の長さが、10 万ビットの場合には、線形複雑度検定は乱数列の長さが推奨値に満たないため実行されない。×は検定にパスしなかったことを意味する。また、E は桁あふれや桁落ち等の理由で正しい結果が得られなかったことを意味する。なお、検定対象の乱数列は、NIST のツールおよび DIEHARD 付属の 26 種の乱数生成アルゴリズムを使って生成したものを使用している。

表 10:周期性に関する検定の NIST ツールでの検定結果 [10 万ビット ×500 本]

生成アルゴリズム	G Using SHA-1		G Using DES		ANSI X9.17(3-DES)		Linear Congruential	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
離散フーリエ変換検定	×		×		×		×	
生成アルゴリズム	Blum-Blum-Shub		Micali-Schonorr		Modular Exponentiation		Quadratic Congruential I	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
離散フーリエ変換検定	×		×		×		×	
生成アルゴリズム	Quadratic Congruential II		Cubic Congruential		XOR		MWC Generator	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
離散フーリエ変換検定	×		×		×		×	
生成アルゴリズム	MWC1616 Generator		MOTHER Generator		KISS Generator		COMBO Generator	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
離散フーリエ変換検定	×		×		×		×	
生成アルゴリズム	ULTRA Generator		MWC/SWB Generator		EXCONG Generator		SUPERDUPER Generator	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
離散フーリエ変換検定	×		×		×		×	
生成アルゴリズム	SWB Generator		RAN2 Generator		Specified shift register Generator		MSRAN Generator	
検定名	一様性	比率	一様性	比率	一様性	比率	一様性	比率
離散フーリエ変換検定	×		E	E			E	E
生成アルゴリズム	Lagged-Fibonacci Generator		INVCONG Generator					
検定名	一様性	比率	一様性	比率				
離散フーリエ変換検定			E	E				

表 10 より、離散フーリエ変換検定は、乱数列の長さが 10 万ビットの場合、すべての乱数生成アルゴリズムでパスしていないことが分かる。これは、乱数列の長さが 10 万ビットでは、乱数列の長さが短すぎ正しく検定が行われていないと考えられる。表 9,10 を見る限りでは、線形複雑度検定が有効であるとは言えないが、LFSR からの出力を検出するという意味合いから周期性に関する検定では、

- 線形複雑度検定

をミニマムセットに加える。

4.2 ミニマムセットの妥当性検証

ここでは、4.1 で導出したミニマムセットの妥当性を各種検定結果により検証する。
ミニマムセットをまとめると、以下の通りになる。

(1) ブロック単位の頻度に関する検定

- 高次元度数検定
- ブロック単位の度数検定
- ブロック単位の最長連検定
- 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
- OPERM5 検定

(2) パターンの出現に関する検定

- ビット列検定
- OPSO 検定
- OQSO 検定
- バースデイ空間検定

(3) 状態遷移・ランダムウォークに関する検定

- 累積和検定

(4) 一様性・圧縮可能性に関する検定

- 系列検定

(5) 周期性に関する検定

- 線形複雑度検定

表 11 では、4.1 で用いた各検定結果をミニマムセットとそれ以外の検定の 2 つのカテゴリーに分類している。NIST のツールで採用されている検定に対しては、一様性、比率のどちらか一方でも検定をパスしないものは FAIL とする。また、DIEHARD の検定に対しては、対応する p-value のうち、0.025 未満、あるいは、0.975 より大きなものの数が 1/4 より多いものには (FAIL) とし、半数より多い場合は FAIL としている。なお、表 11 では、エラーとなる検定も FAIL としている。さらに、離散フーリエ変換検定の 10 万ビットの乱数列に対する結果は検討の対象外とし、推奨パラメータや実装上の理由で検定を行っていないものは – としている。

表 11:ミニマムセットの妥当性の検証

	生成アルゴリズム検定名	G Using SHA-1	G Using DES	ANSI X9.17(3-DES)	Linear Congruential
ミニマムセット	高次元度数検定				
	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				
	OPERM5 検定			(FAIL)	
	ブロック単位の最長連検定				
	ブロック単位の度数検定				
	ビット列検定				
	OPSO 検定				
	OQSO 検定				
	バースデイ空間検定				
	累積和検定				
	系列検定				
	線形複雑度検定				
その他の検定	1 次元度数検定				
	連の検定 (SP800-22)				
	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				
	スクイーズ検定				
	クラップス検定				
	(6×8) の 2 値行列ランク検定				
	(31×31) の 2 値行列ランク検定				
	(32×32) の 2 値行列ランク検定				
	DNA 検定				
	連の検定				
	駐車場検定				
	最小距離検定				
	3DSPHERES 検定				
	重なりのある和検定				
	ランダム偏差検定				
	種々のランダム偏差検定				
	Maurer のユニバーサル統計検定				
	Lempel-Ziv 圧縮検定				
	近似エントロピー検定				
	離散フーリエ変換検定				

	生成アルゴリズム 検定名	Blum-Blum-Shub	Micali-Schonorr	Modular Exponentiation	Quadratic Congruential I
ミニマムセット	高次元度数検定		FAIL	FAIL	FAIL
	8 ビット中の文字数検定		FAIL	FAIL	(FAIL)
	特定位置の 8 ビット中の文字数検定		FAIL	(FAIL)	
	OPERM5 検定		FAIL	(FAIL)	(FAIL)
	ブロック単位の最長連検定				
	ブロック単位の度数検定		FAIL	FAIL	FAIL
	ビット列検定		FAIL		
	OPSO 検定		FAIL	FAIL	FAIL
	OQSO 検定		FAIL	FAIL	
	バースデイ空間検定				
	累積和検定		FAIL	FAIL	FAIL
	系列検定		FAIL	FAIL	FAIL
	線形複雑度検定				
その他の検定	1 次元度数検定		FAIL	FAIL	FAIL
	連の検定 (SP800-22)			FAIL	FAIL
	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				
	スクイーズ検定				
	クラップス検定				
	(6×8) の 2 値行列ランク検定			(FAIL)	
	(31×31) の 2 値行列ランク検定				
	(32×32) の 2 値行列ランク検定				
	DNA 検定		FAIL		
	連の検定				
	駐車場検定				
	最小距離検定		(FAIL)	(FAIL)	
	3DSPHERES 検定				
	重なりのある和検定				
	ランダム偏差検定				
	種々のランダム偏差検定				
	Maurer のユニバーサル統計検定				
	Lempel-Ziv 圧縮検定				
	近似エントロピー検定		FAIL	FAIL	FAIL
	離散フーリエ変換検定				

	生成アルゴリズム 検定名	Quadratic Congruential II	Cubic Congruential	XOR	MWC Generator
ミニマムセット	高次元度数検定	FAIL	FAIL	FAIL	
	8 ビット中の文字数検定	–		FAIL	
	特定位置の 8 ビット中の文字数検定	–		FAIL	
	OPERM5 検定	–		FAIL	(FAIL)
	ブロック単位の最長連検定		FAIL		
	ブロック単位の度数検定		FAIL		
	ビット列検定	–		FAIL	
	OPSO 検定	–	FAIL	FAIL	
	OQSO 検定	–	(FAIL)	FAIL	
	バースデイ空間検定	–		FAIL	
	累積和検定		FAIL		
	系列検定		FAIL	FAIL	
	線形複雑度検定			FAIL	–
その他の検定	1 次元度数検定		FAIL		
	連の検定 (SP800-22)		FAIL		
	重なりのないテンプレート適合検定		FAIL	FAIL	–
	重なりのあるテンプレート適合検定			FAIL	–
	スクイーズ検定	–		FAIL	
	クラップス検定	–		FAIL	
	(6×8) の 2 値行列ランク検定	–		FAIL	
	(31×31) の 2 値行列ランク検定	–		FAIL	
	(32×32) の 2 値行列ランク検定			FAIL	
	DNA 検定	–		FAIL	
	連の検定	–		FAIL	
	駐車場検定	–		FAIL	
	最小距離検定	–		FAIL	
	3DSPHERES 検定	–		FAIL	
	重なりのある和検定	–		FAIL	
	ランダム偏差検定				–
	種々のランダム偏差検定				–
	Maurer のユニバーサル統計検定			FAIL	–
	Lempel-Ziv 圧縮検定		FAIL		–
	近似エントロピー検定		FAIL	FAIL	
	離散フーリエ変換検定	FAIL	FAIL		

	生成アルゴリズム 検定名	MWC1616 Generator	MOTHER Generator	KISS Generator	COMBO Generator
ミニマムセット	高次元度数検定				
	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				
	OPERM5 検定				
	ブロック単位の最長連検定				
	ブロック単位の度数検定				
	ビット列検定				
	OPSO 検定				
	OQSO 検定				
	バースデイ空間検定				
	累積和検定				
	系列検定				
	線形複雑度検定	–	–	–	–
その他の検定	1 次元度数検定				
	連の検定 (SP800-22)				
	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	–	–	–	–
	スクイーズ検定				
	クラップス検定				
	(6×8) の 2 値行列ランク検定				
	(31×31) の 2 値行列ランク検定				
	(32×32) の 2 値行列ランク検定				
	DNA 検定				
	連の検定				
	駐車場検定				
	最小距離検定				
	3DSPHERES 検定				
	重なりのある和検定				
	ランダム偏差検定	–	–	–	–
	種々のランダム偏差検定	–	–	–	–
	Maurer のユニバーサル統計検定	–	–	–	–
	Lempel-Ziv 圧縮検定	–	–	–	–
	近似エントロピー検定				
	離散フーリエ変換検定				

	生成アルゴリズム 検定名	ULTRA Generator	MWC/SWB Generator	EXCONG Generator	SUPERDUPER Generator
ミニマムセット	高次元度数検定				
	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				
	OPERM5 検定				(FAIL)
	ブロック単位の最長連検定				
	ブロック単位の度数検定				
	ビット列検定				
	OPSO 検定				
	OQSO 検定				
	バースデイ空間検定				
	累積和検定				
	系列検定				
	線形複雑度検定	–	–	–	–
その他の検定	1 次元度数検定				
	連の検定 (SP800-22)				
	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	–	–	–	–
	スクイーズ検定				
	クラップス検定				
	(6×8) の 2 値行列ランク検定				
	(31×31) の 2 値行列ランク検定				
	(32×32) の 2 値行列ランク検定				
	DNA 検定				
	連の検定				
	駐車場検定				
	最小距離検定				
	3DSPHERES 検定				
	重なりのある和検定				
	ランダム偏差検定	–	–	–	–
	種々のランダム偏差検定	–	–	–	–
	Maurer のユニバーサル統計検定	–	–	–	–
	Lempel-Ziv 圧縮検定	–	–	–	–
	近似エントロピー検定				
	離散フーリエ変換検定				

	生成アルゴリズム検定名	SWB Generator	RAN2 Generator	Specified shift register Generator	MSRAN Generator
ミニマムセット	高次元度数検定		FAIL		FAIL
	8 ビット中の文字数検定	FAIL	FAIL	FAIL	FAIL
	特定位置の 8 ビット中の文字数検定			FAIL	
	OPERM5 検定			FAIL	(FAIL)
	ブロック単位の最長連検定		FAIL		FAIL
	ブロック単位の度数検定		FAIL		FAIL
	ビット列検定		FAIL		FAIL
	OPSO 検定				
	OQSO 検定			FAIL	
	バースデイ空間検定	FAIL	(FAIL)	FAIL	
	累積和検定			FAIL	
	系列検定		FAIL		FAIL
	線形複雑度検定	–	–	–	–
その他の検定	1 次元度数検定		FAIL		FAIL
	連の検定 (SP800-22)		FAIL		FAIL
	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	–	–	–	–
	スクイーズ検定			FAIL	
	クラップス検定	(FAIL)		FAIL	
	(6×8) の 2 値行列ランク検定			FAIL	
	(31×31) の 2 値行列ランク検定	FAIL		FAIL	
	(32×32) の 2 値行列ランク検定		FAIL	FAIL	FAIL
	DNA 検定			(FAIL)	
	連の検定			FAIL	
	駐車場検定				
	最小距離検定				
	3DSPHERES 検定				
	重なりのある和検定				
	ランダム偏差検定	–	–	–	–
	種々のランダム偏差検定	–	–	–	–
	Maurer のユニバーサル統計検定	–	–	–	–
	Lempel-Ziv 圧縮検定	–	–	–	–
	近似エントロピー検定		FAIL		FAIL
	離散フーリエ変換検定				

	生成アルゴリズム 検定名	Lagged- Fibonacci Generator	INVCONG Generator		
ミニマムセット	高次元度数検定		FAIL		
	8 ビット中の文字数検定	(FAIL)	FAIL		
	特定位置の 8 ビット中の文字数検定	FAIL	FAIL		
	OPERM5 検定	(FAIL)	FAIL		
	ブロック単位の最長連検定		FAIL		
	ブロック単位の度数検定		FAIL		
	ビット列検定		FAIL		
	OPSO 検定		FAIL		
	OQSO 検定		FAIL		
	バースデイ空間検定	FAIL	FAIL		
	累積和検定		FAIL		
	系列検定		FAIL		
	線形複雑度検定	–	–		
その他の検定	1 次元度数検定		FAIL		
	連の検定 (SP800-22)		FAIL		
	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	–	–		
	スクイーズ検定		FAIL		
	クラップス検定		FAIL		
	(6×8) の 2 値行列ランク検定		FAIL		
	(31×31) の 2 値行列ランク検定		FAIL		
	(32×32) の 2 値行列ランク検定		FAIL		
	DNA 検定		FAIL		
	連の検定		FAIL		
	駐車場検定		FAIL		
	最小距離検定		FAIL		
	3DSPHERES 検定		FAIL		
	重なりのある和検定		FAIL		
	ランダム偏差検定	–	–		
	種々のランダム偏差検定	–	–		
	Maurer のユニバーサル統計検定	–	–		
	Lempel-Ziv 圧縮検定	–	–		
	近似エントロピー検定		FAIL		
	離散フーリエ変換検定				

表 11 から分かるように、評価したすべての乱数生成アルゴリズムに対して、ミニマムセットに含まれない検定で FAIL ときは、必ずミニマムセットの中に結果が FAIL となる検定が存在している。また、漸化式

$$X_i = aX_{i-1} + c \pmod{M}$$

で表現される線形合同法 (Linear Congruential) でもよいパラメータを選べば、すべての検定をパスすることが分かる。一般に、シミュレーションなどに使う場合、線形合同法からの出力は乱数として十分な性質を持っているとみなすことができるが、暗号に利用する場合は、安全でないことが

知られている [12-18]。したがって、統計的な乱数検定だけでは暗号で用いる乱数の評価として不十分なことが分かる。この結果から、暗号で用いる乱数列を評価する場合は、生成アルゴリズムに出力が一様でない、あるいは、周期が短い等の欠点があるものは初めに除外し、欠点が見つかっていないアルゴリズムからの出力に対し統計的な検定を行いその性質を調べるといった、理論的な評価と統計的な評価を両方行う必要があることが分かる。

5　まとめ

本調査・開発では、既存の乱数検定法や乱数検定ツールの調査によりリストアップされた各種乱数検定法に対し、数学的な考察や実際に乱数生成アルゴリズムからの出力に対して検定を行うことにより、有効な検定とそうでないものとを分類し、乱数列を検定する上で必要最小限のもので構成されるミニマムセットを導出した。本調査・開発では、ミニマムセットの導出に用いる実データとして、NIST のツール [5] および DIEHARD [7] に付属している乱数生成アルゴリズムからの出力を使用した。

なお、4 章の結果からも分かるように、統計的な乱数検定は、真にランダムな系列の持つべき種々の性質の一部を保障するものであり、各種検定に合格したからといって、検定対象の乱数列の性質の十分性を示しているわけではないことに注意する必要がある。したがって、暗号で用いる乱数列を評価する場合は、統計的な評価だけでは不十分であり、乱数生成アルゴリズムの理論的な評価も同時に行う必要がある。

参考文献

[1] D. Knuth, The art of computer programming, seminumerical algorithms, Vol.3.

[2] NIST, FIPS PUB 140-2, “Security requirements for cryptographic modules,”

[3] NIST, FIPS PUB 140-2, “Change Notice 2,” 2002.

[4] 伏見 正則, “乱数, ” 東京大学出版.

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

[6] NIST, Special Publication 800-22, “NIST Statistical Suite,”

[7] G. Marsaglia, “DIEHARD,” (`http://stat.fsu.edu/ geo/diehard.html`, `http://stat.fsu.edu/pub/diehard/`)

[8] P. Hellekalek, “Tests for random numbers,” (`http://random.mat.sbg.ac.at/tests/`)

[9] J. Walker, “A Pseudorandom Number Sequence Test Program,” (`http://www.fourmilab.ch/random/`)

[10] The Information Security Research Centre at Queensland University of Technology, “Crypt-X,” (`http://www.isrc.qut.edu.au/resource/cryptx/`)

[11] G. Louchard and W. Szpankowski, “On the average redundancy rate of the Lempel-Ziv code,” *IEEE Trans. on Inform. Theory*, Vol. 43, No. 1, 1997.

[12] J. A. Reeds, “Cracking multiplocative congruential encryption algorithm,” in Information Linkage Between Applied Mathematics and Industry, P.C.C. Wang, ed., Academic Press, pp.467-472, 1979.

[13] J. A. Reeds, “Solution of challenge cipher,” Cryptologia, Vol. 3, No. 2, pp.83-95, 1979.

[14] J. B. Plumstread, “Inferring a sequence generated by a linear congruence,” Proceedings of the 23rd *IEEE* Symposium on the Foundations of Computer Science, pp.153-159, 1982.

[15] J.C. Lagarias and J. Reeds, “Unique extrapolation of polynomial recurrences,” SIAM Journal on Computing, Vol. 17, No. 2, pp.342-362, 1988.

[16] H. Krawczyk, “How to predict congruential generators,”, Advances in Cryptology–CRYPTO ’89, pp.138-153, 1990.

[17] A.M. Frieze, J. Hastad, R. Kannan, J. C. Lagarias and A. Shamir, “Reconstracting truncated integer variables satisfying linear congruences,” SIAM Journal on Computing, Vol. 17, No. 2, pp.262-280, 1988.

[18] J. Stern, “Secret linear congruential generators are not cryptographically Secure,” Proceedings of the 28th Symposium on the Foundations of Computer Science, pp.421-426, 1987.