数理解析研究所講究録 1351 巻 2004 年 80-93

擬似乱数検証ツールの調査開発

丹羽 朗人　　栃窪 孝也
Akito Niwa　　Kouya Tochikubo
東芝ソリューション (株) SI 技術開発センター
Systems Integration Technology Center, Toshiba Solutions Corporation

概 要

情報セキュリティの分野では、様々な場面で乱数を利用する。このため、システムの安全性評価には、システムで利用する乱数の評価が不可欠である。現在、様々な乱数の検定方法が提案されているが、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。

本稿では、既存の乱数検定法の有効性の検証を基に開発した擬似乱数検証ツールの概要を述べる。開発ツールでは、乱数列を検定する上で必要最小限のもので構成される乱数検定法のミニマムセットにより、擬似乱数の検定を行う点が特徴である。

1 はじめに

情報セキュリティの分野では、鍵生成やチャレンジーレスポンス認証など様々な場面で乱数を利用する。このため、システムの安全性評価には、システムで利用する乱数の評価が不可欠である。乱数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り、様々な統計的検定法が提案されている [1-10]。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、本研究では、乱数に関する文献及び評価ツールを調査し、文献及び評価ツールで採用されている検定の目的および数学的根拠を明確にする。さらに、実際に様々な乱数列を検定しその有効性を検証することにより擬似乱数の評価に有効な検定とそうでないものとを分類し、乱数を検定する上で必要な検定法のミニマムセットを求めている。そして、調査・分析結果に基づき定めたミニマムセットを実装した乱数検定ツールを開発している。開発した乱数検定ツールでは、調査で導出したミニマムセットの検定を実行する IPA 推奨乱数検定機能の他に、FIPS 140-2 [2, 3] で定められた乱数検定を実行する FIPS 140-2 乱数検定機能、NIST Special Publication 800-22(SP800-22) [5, 6] と同様の検定を行う NIST SP800-22 乱数検定機能などがある。

2 調査の概要

本研究では以下の 5 つの文献、ツールを中心に調査を行った。

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

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 (frequency test)

K2 系列検定 (2 次元度数検定)(serial test)

K3 間隔検定 (gap test)

K4 ポーカー検定 (poker test)

K5 札集め検定 (coupon collector's test)

K6 順列検定 (permutation test)

K7 連の検定 (run test)

K8 t 個の数の最大値検定 (maximum-of t test)

K9 衝突検定 (collosion test)

K10 系列相関検定 (serial correlation test)

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

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

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

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、

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

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、

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

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。

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

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。

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)

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。本研究では、各検定を以下のように分類している。

3.1 ブロック単位の頻度に関する検定

- 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。

- 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。

- 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。

- 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。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。

- 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。

- 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。

- 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。

- 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。

- 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。

- 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。

- 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。

- (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。

- (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。

- (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。

- 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。なお、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。

- ブロック単位の度数検

  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。

3.2 パターンの出現に関する検定

- 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。

- 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。

- 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。

- 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。

- 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。

- 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。

- 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。

- 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。

- 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。

- 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。なお、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。

- 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。なお、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。

- 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。

- 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。

- 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。

3.3 状態遷移・ランダムウォークに関する検定

- 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。

- 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。

- 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。

3.4 一様性・圧縮可能性に関する検定

- 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。

- 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。

- 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。

- 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。

3.5 周期性に関する検定

- 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。

- 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。

- 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。

3.6 その他の検定

- 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

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

  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。

- 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。

4 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、本研究では、入力データとして、SP800-22 付属の擬似乱数生成プログラムおよび DIEHARD 付属の擬似乱数生成プログラムからの出力を用いている。

本研究で得られたミニマムセットは以下の通りである。

- 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
- 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
- 状態遷移・ランダムウォークに関する検定
  - 累積和検定
- 一様性・圧縮可能性に関する検定
  - 系列検定
- 周期性に関する検定
  - 線形複雑度検定

高次元度数検定は2章の文献、ツールでで採用されている検定法ではなく、本研究で新たに導入した検定法である。

しかしながら、漸化式

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

で表現される線形合同法 (Linear Congruential) からの出力などは、導出したミニマムセットのすべての検定をパスしてしまう。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 [11-17]。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、アルゴリズムの理論的な評価を行い、さらに、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 開発ツールの概要

本章では、開発した擬似乱数検証ツールの概要を述べる。

5.1 開発ツールの機能

開発ツールには、大きく次の4つの検定機能を持っている。

- FIPS140-2 乱数検定機能
  FIPS140-2 に規定されている 4 種類の検定法による乱数検定を行う場合に使用する。
- SP800-22 乱数検定機能
  SP800-22 で採用されている 16 種類の検定法による乱数検定を行う場合に使用する。
- IPA 推奨乱数検定機能
  調査結果から得られたミニマムセットによる乱数検定を行う場合に使用する。本機能のみで十分信頼できる検定結果を得ることができる。なお、ツールでは、「IPA 推奨乱数検定機能」という名称を付けているが、実際にはこの名称は、乱数検定に必要なミニマムセットであることを意味しているのみであることを注意する。
- 個別検定機能
  調査結果から得られたすべての有効な検定法 28 種類の中から個別に選択し、乱数検定を行う場合に使用する。

開発ツールを起動すると図 1 のような初期画面が現れ、4 つの検定機能から実行する機能を選択できる。

なお、開発ツールでは、入力する乱数列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。また、図 2 のような検定結果が出力される。

5.2 開発ツールの構成

本開発ツールは、大きく GUI とライブラリから構成される (図 3)。

- GUI 機能
  グラフィカルユーザインターフェース（GUI）を介して、各検定機能の選択、および各検定機能を実行するために必要なパラメータ（乱数列のファイル名、ビット長など）の入力を行う。乱数検定ライ

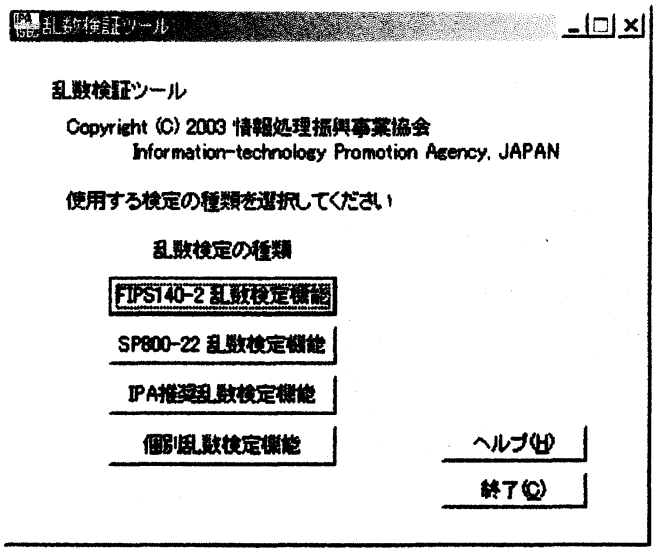

図 1: 初期画面

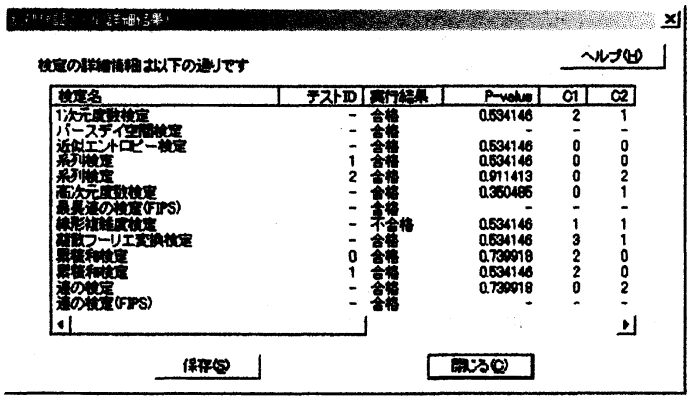

図 2: 詳細結果画面

ブラリの中から、選択された各検定機能に含まれる各検定法を呼び出して実行し、各検定結果及び検定結果サマリを出力する。

- 乱数検定ライブラリ
  各検定機能で用いられる検定法の集合である。各検定機能を実行する際に、本ライブラリ内から対象となる検定法が呼び出される。また、本ライブラリ内には、パブリックドメインソフト（PDS）の数学関数を組み込む。

6 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。さらに、導出されたミニマムセットの検定を実行する乱数検定ツールを開発した。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。

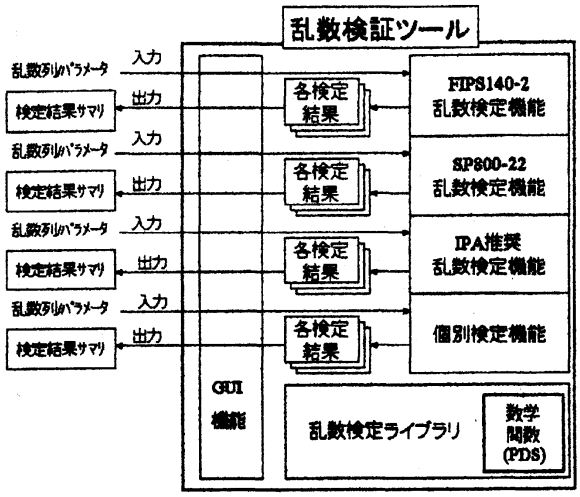

図 3: 乱数検定ツールの機能関連図

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。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。今後の課題としては、統計的な検定法だけでも十分に乱数の評価が可能な乱数検定法の開発などが挙げられる。

謝辞

本研究は、情報処理振興事業協会 (IPA) の電子政府情報セキュリティ技術開発事業の一環として行われたものである。また、本研究にあたり、御教示を頂いた南山大学伏見正則教授ならびに東京工業大学植松友彦教授に深謝する。

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] 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.

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

[13] 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.

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

[15] H. Krawczyk, "How to predict congruential generators,", Advances in Cryptology–CRYPTO '89, pp.138-153, 1990.

[16] 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.

[17] 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.