非代数的な手法による擬似乱数生成法の研究

１７５４０１１１

平成１７年度～平成１８年度科学研究費補助金

（基盤研究（Ｃ））研究成果報告書

平成１９年３月

研究代表者　谷　口　礼　偉

三重大学教育学部教授

＜はしがき＞

本報告書は、平成１７年度～平成１８年度科学研究費補助金(基盤研究（C）)による研究成果の報告書である。本研究は非代数的な手法、とりわけ力学系のエルゴード理論に基礎をおく擬似乱数の生成法の研究である。この研究は、本研究期間中に理論的にもまた実際の乱数生成法としても格段の進展があった。本報告書の「研究成果」をぜひ見ていただきたい。

研 究 組 織

研究代表者　：　谷 口 礼 偉　（三重大学教育学部教授）

交付決定額（配分額）　（金額単位：円）

	直接経費	間接経費	合　計
平成１７年度	1,100,000	0	1,100,000
平成１８年度	500,000	0	500,000
総　計	1,600,000	0	1,600,000

研究発表

（1）学会誌等

谷口礼偉：実数シフト乱数生成法のメッセージダイジェスト性について．三重大学教育学部研究紀要，２００５年３月

谷口礼偉，上田澄江，高島恵三：新しいハッシュ関数ＳＳＩ１６０の構築．　統計数理研究所 Research Memorandum No. 986, March 28, 2006

谷口礼偉：実数シフト乱数生成法の乱数特性の改良について．三重大学教育学部研究紀要，２００６年３月

谷口礼偉：実数演算を用いた非再帰的な擬似乱数生成の整数演算化．三重大学教育学部研究紀要，２００７年３月

Kubo, I., Yaguchi, H. : A new nonrecursive pseudorandom number generator based on chaos mappings. (投稿中)

（2）口頭発表

久保泉，谷口礼偉：カオス写像で生成される擬似乱数の Perron-Frobenius 作用素による解析．日本数学会，一般講演，２００５年３月２７日

谷口礼偉：64 ビット整数の乗算とシフトによる非再帰的な擬似乱数の生成．日本数学会，一般講演，２００７年３月２７日

Kubo, I., Yaguchi, H. : A new nonrecursive pseudorandom number generator based on chaos mappings. MCQMC 2006. 2006.8.15

研究成果

数値計算における桁落ち誤差の発生メカニズムを利用した、非再帰的な擬似乱数生成である実数シフト法（ＳＲ法）は、それまでの研究により乱数性が確認され、さらに長周期化が行われていた。しかしながら、乱数性の数理的なメカニズムは、明らかではなかった。

本研究期間前までには、ＳＲ法アルゴリズムの必須部分を抽出した単純実数シフト法（ＳＳＲ法）について、乱数性が検証され、アルゴリズムも各部分に分解されて、乱数性の背後にはカオス写像が潜んでいることも分かった。しかしながら、なお全体的な見通しがつくまでには至らなかった。

幸いなことに、本研究期間中に、理論的な研究が進み、乱数特性が数理的に記述できる段階にまで至った。その結果、ＳＳＲのメカニズムが非常にクリアになり、ＳＳＲアルゴリズムの発展的な応用も可能となり、ＳＳＲ乱数生成法の全整数演算化（ＳＳＩ法)、ＳＳＩ法のハッシュ関数への応用が可能となった。これらの応用研究はまだ始まったばかりと言えるが、本研究成果報告では、それらを含めて、結果を詳述した。また、論文「A new nonrecursive pseudorandom number generator based on chaos mappings」には、ＳＳＩ法の核心となる乱数生成部分のプログラムコードが記載されているので、これも併せて収録した。

［以下、研究成果本文に続く。］

目　次

1．単純実数シフト法（ＳＳＲ法）の導入とその理論的解析

実数シフト法（Shift-Real method, ＳＲ法）が生成する乱数値の分布に関する理論的な背景が最近明らかになってきたのでここで紹介する。

まず、ＳＲ法アルゴリズムの本質部分を抽出して、新たに単純実数シフト法（Simplified Shift-Real method, ＳＳＲ法）擬似乱数生成法を導入することから始める。

1.1 ＳＲ法アルゴリズムの単純化

非再帰的に擬似乱数を生成するＳＲ法乱数生成法の中心となる計算は、

$$f(x) = x \cdot \frac{x}{2} \cdot \frac{x}{3} \cdot \frac{x}{4} \cdots \frac{x}{23} \cdot \frac{x}{24} \quad , \quad x \in [1, 2)$$

を

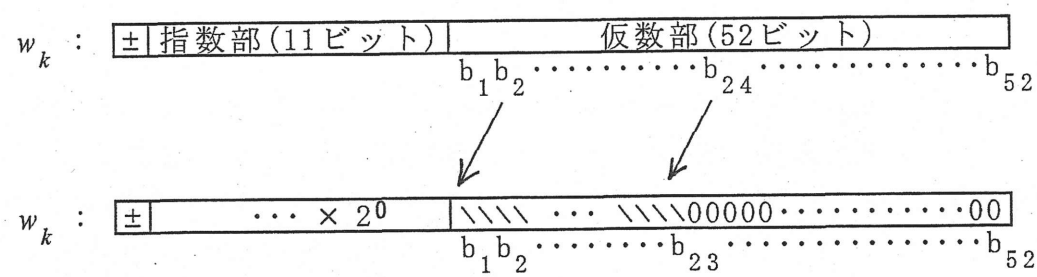
$$w_0 := 1 \qquad w_k := w_{k-1} \times \frac{x}{k} \quad , \quad k=1, 2, \cdots, 24,$$

と倍精度計算し、w_k を1回計算するごとに、w_k を表す倍精度変数の

i) 仮数部の全てのビット値を１ビット左にシフトし、

ii) さらに、仮数部上位23ビット以外のビットは０にする、

iii) 指数部は　$\cdots \times 2^0$　となるように設定する、



というもの（実数シフト計算, Shift-Real computation）であった。また、実際の乱数値としては、x を変化させ、対応する計算結果 w_{24} から、上位３桁を乗て続く４桁を取り出すものであった。

その後、アルゴリズムを分解して詳しく調べてみると、アルゴリズム中の

○ $w_k := w_{k-1} \times \frac{x}{k}$ における $\times \frac{1}{k}$ 、

o (ii) 仮数部上位23ビット以外のビットは0にする、

を取り除いても、目立つような特性の劣化が見られないことが分った。すなわち、上述実数シフト計算を簡略化して、

[単純実数シフト計算(SSR)]

$$g(x) = u_0 \underbrace{\cdot x \cdot x \cdot x \cdot \cdots \cdot x \cdot x}_{24}, \quad x \in [1,2)$$

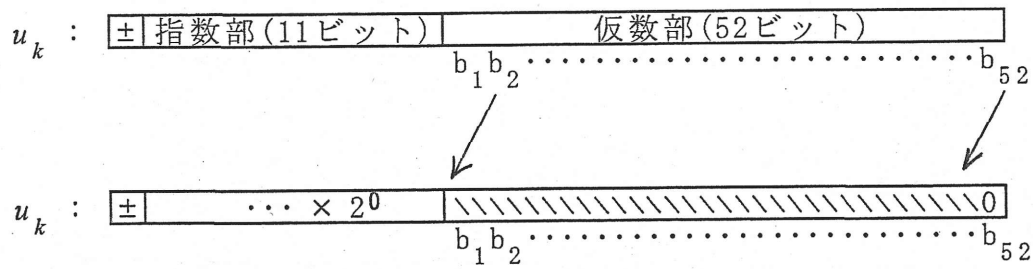
を

$$u_0 := 1 \qquad u_k := u_{k-1} \times x, \quad k=1,2,\cdots,24,$$

と倍精度計算し、u_k を1回計算するごとに、u_k を表す倍精度変数の

i) 仮数部の全てのビット値を1ビット左にシフトし、

ii) 指数部は $\cdots \times 2^0$ となるように設定する、



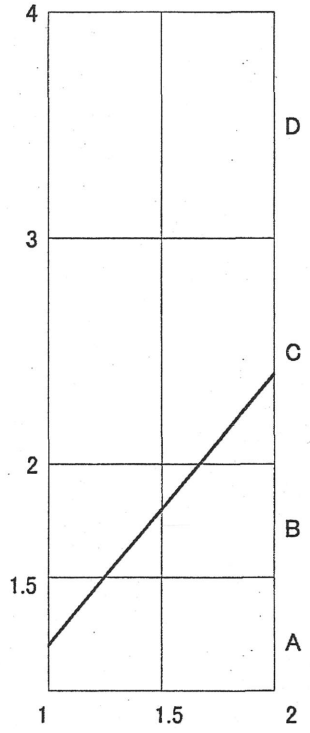
というアルゴリズムで十分な乱数特性をもつ数値列が得られることが分った。このことは、ＳＲ法の乱数生成メカニズムの解析がかなり単純化されることを意味する。この簡略化されたＳＲ法を**単純ＳＲ法**(simplified Shift-Real method, **ＳＳＲ法**) と呼ぶことにする。以下の節でその乱数生成のメカニズムを詳しく調べるが、この簡略化は次のような副次的なメリットも生ずることを指摘しておく：

(a) $f(x) = x \cdot \frac{x}{2} \cdot \frac{x}{3} \cdots \frac{x}{24}$ の代わりに $g(x) = x \cdot x \cdot x \cdots x$ を計算するので、除算 $/i$ が無くなり、計算が高速化される、

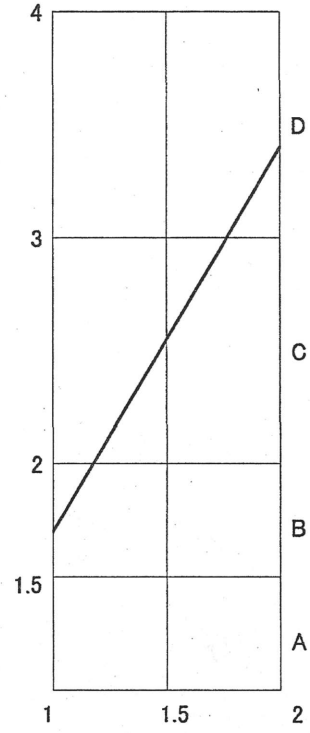
(b) 仮数部シフト後の右端には、通常 0 が自動的に入るので、その分の演算を節約できる。

1．2 ＳＳＲ計算のアルゴリズムの分解

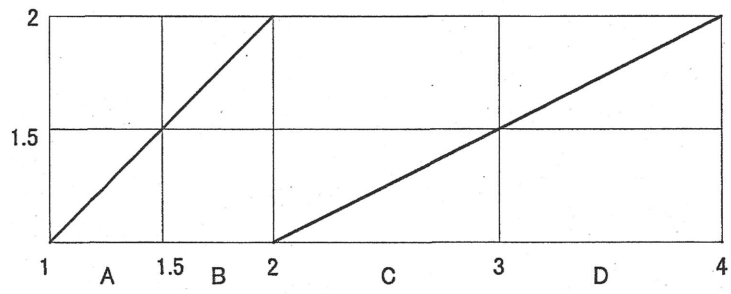
ＳＳＲ計算のアルゴリズム中の u_{k-1} を x 倍する写像を φ_x と書き、$\varphi_x(u_{k-1})$ の指数部を $\cdots\times 2^0$ とする写像を φ_e と書き、さらに $\varphi_e(\varphi_x(u_{k-1}))$ の仮数部をシフトする写像を φ_s と書くと、それらのグラフは次の様になる。



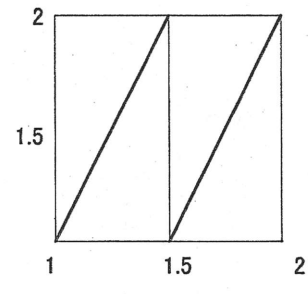
φ_x　$(1 \leq x < 1.5)$



φ_x　$(1.5 \leq x < 2)$



φ_e



φ_s

このとき、ＳＳＲ計算で u_{k-1} から u_k を得る一連のプロセスは、

$$u_k = (\varphi_s \circ \varphi_e \circ \varphi_x)(u_{k-1}), \quad k=1, 2, \cdots, 24,$$

と表され、したがって

$$u_k = (\varphi_s \circ \varphi_e \circ \varphi_x)^k(1) \ , \quad k=1,2,\cdots,24,$$

となる。一般に $\Phi_x \equiv \varphi_s \circ \varphi_e \circ \varphi_x$ とおくと、$\Phi_x : [1,2) \longrightarrow [1,2)$ であり、

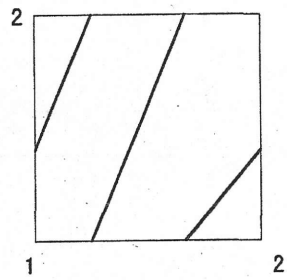
$1 \leq x < 1.5$ のとき

$$\Phi_x(t) = \begin{cases} 2xt-1 & (1 \leq t < \frac{3}{2x} \text{ のとき}) \\ 2xt-2 & (\frac{3}{2x} \leq t < \frac{2}{x} \text{ のとき}) \\ xt-1 & (\frac{2}{x} \leq t < 2 \text{ のとき}) \end{cases}$$

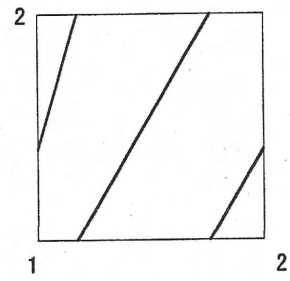
$1.5 \leq x < 2$ のとき

$$\Phi_x(t) = \begin{cases} 2xt-2 & (1 \leq t < \frac{2}{x} \text{ のとき}) \\ xt-1 & (\frac{2}{x} \leq t < \frac{3}{x} \text{ のとき}) \\ xt-2 & (\frac{3}{x} \leq t < 2 \text{ のとき}) \end{cases}$$

となり、それぞれ次の様なグラフになる：



$\Phi_x \equiv \varphi_s \circ \varphi_e \circ \varphi_x \ (1 \leq x < 1.5)$



$\Phi_x \ (1.5 \leq x < 2)$

ＳＳＲ法の基本となる計算は、各 $x \in [1,2)$ 毎の $u_{24} = \Phi_x^{24}(1)$ の計算であるから、ＳＳＲ法による乱数の特性解析には、写像 Φ_x , $1 \leq x < 2$, のエルゴード的性質の理論的な解明が必要となる。

［註］ $\frac{d}{dt}\Phi_x(t) = x$ or $2x$ であり、特に $\frac{d}{dt}\Phi_x > 1$, $x \in [1,2)$, となる。このような性質を持つ区分的に連続な関数 Φ_x は、多くの場合 $\Phi_x^k(t)$, $k=1,2,\cdots$, がカオス的様相[1]を呈する。Φ_x のカオス性については、１．７節を参照のこと。

1．3 Perron-Frobenius 作用素と 写像 Φ_x の不変測度

$\mathcal{X}=[1,2)$ とし、$\mathcal{X}$ 上の Borel 集合体を $\mathcal{B}$，$(\mathcal{X},\mathcal{B})$ 上の Lebesgue 測度を λ，$(\mathcal{X},\mathcal{B},\lambda)$ 上の可積分関数全体のなす空間を L^1 と書くことにする。ＳＳＲ計算値 $\Phi_x^{24}(1)$ の分布を解析する以下のアイデアは久保泉氏による。

$\Phi_x(t)\equiv\varphi_s\circ\varphi_e\circ\varphi_x(t)$ とすると、$\Phi_x:\mathcal{X}\longrightarrow\mathcal{X}$，$\Phi_x\in L^1$，$\frac{d}{dt}\Phi_x(t)=x$ or $2x$ であり、Φ_x の不変測度 μ_x の密度関数 h_x は、Perron-Frobenius 作用素 $\mathcal{L}_x:L^1\longrightarrow L^1$，

$\mathcal{L}_x:h\longmapsto(\mathcal{L}_xh)$，ただし，

$1\le x<1.5$ のとき

$$(\mathcal{L}_xh)(t)=\begin{cases}\frac{1}{2x}h(\frac{t+2}{2x})+\frac{1}{x}h(\frac{t+1}{x}) & 1\le t<2x-1 \text{ のとき}\\ \frac{1}{2x}h(\frac{t+1}{2x})+\frac{1}{2x}h(\frac{t+2}{2x}) & 2x-1\le t<2 \text{ のとき}\end{cases}$$

$1.5\le x<2$ のとき

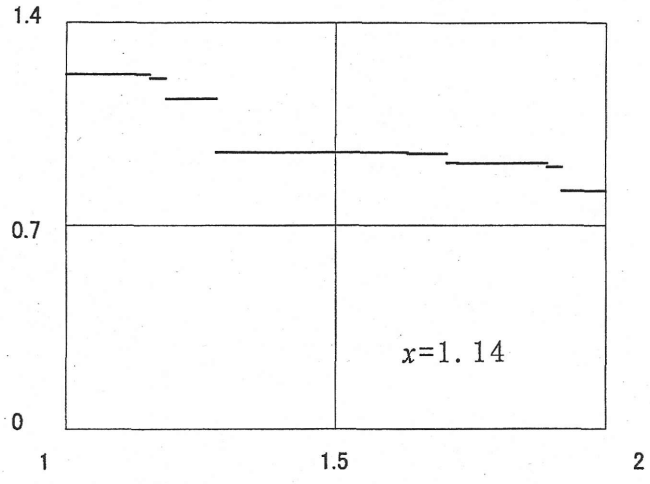
$$(\mathcal{L}_xh)(t)=\begin{cases}\frac{1}{x}h(\frac{t+1}{x})+\frac{1}{x}h(\frac{t+2}{x}) & 1\le t<2x-2 \text{ のとき}\\ \frac{1}{2x}h(\frac{t+2}{2x})+\frac{1}{x}h(\frac{t+1}{x}) & 2x-2\le t<2 \text{ のとき}\end{cases}$$

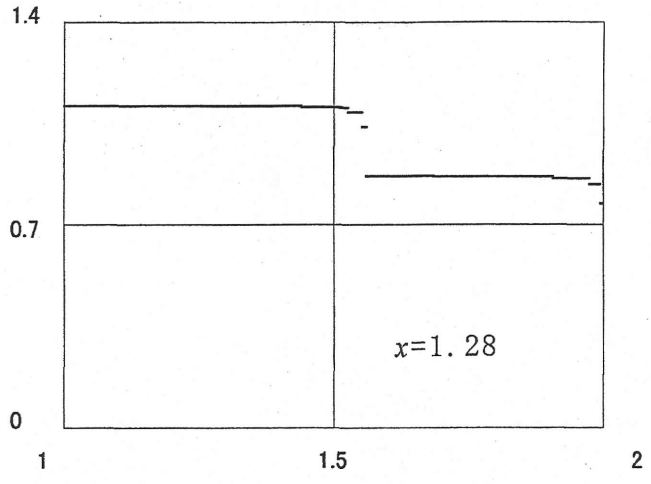
の不動点($\mathcal{L}_xh_x=h_x$ なる関数 h_x）で与えられる。h_x が存在することは Φ_x の性質($\frac{d}{dt}\Phi_x(t)=x$ or $2x$)から分るが、式で表すのは大変なので（特別な x を除き、おそらく不可能）、適当な関数 h を選び

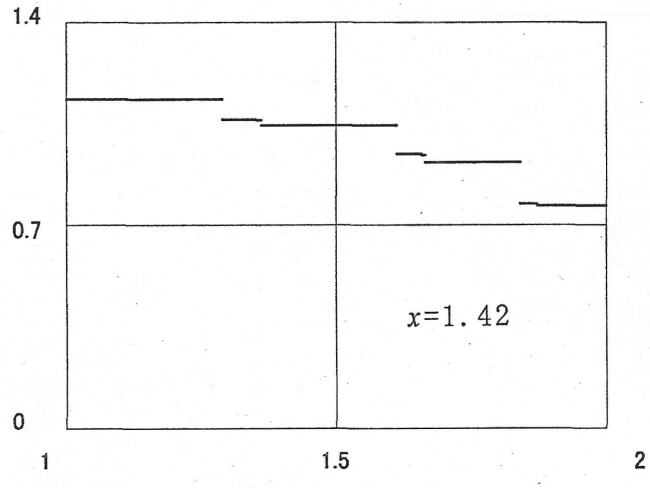
$$h_x(t)=\lim_{n\to\infty}\{\frac{1}{n}\sum_{k=0}^{n-1}\mathcal{L}_x^kh\}(t)$$

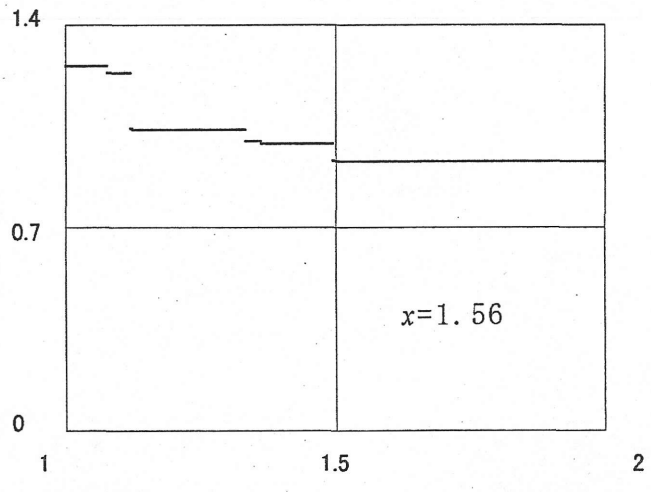
なる関係式を使って数値計算で求める([ ])。実際の計算に当たっては、$n=200$，$h\equiv1$ とし、区間 $[1,2)$ を 集合 $\{c_i=1+\frac{i}{10000}\mid i=0,1,2,\cdots,9999\}$ で離散近似して、$h_x(c_i)$，$i=0,1,2,\cdots,9999$，を求る。次ページにいくつかの x の値について h_x のグラフを示しておく。

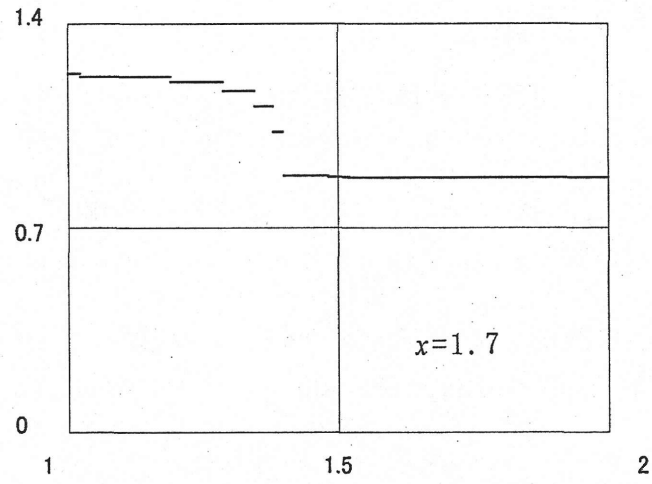
次々ページには、h_x のグラフの妥当性を確認するために、同じ x の値について、$\Phi_x^n(1)$，$n=1,2,\cdots,10000000$，が区間 $[1+0.001\times l,\ 1+0.001\times(l+1))$，$l=0,1,\cdots,999$，に入る回数を 10000 で割った値のグラフを示してある（$1+0.001\times l$ を X 軸にとっている）。両ページの同じ x の値のグラフの類似性に注目して欲しい。

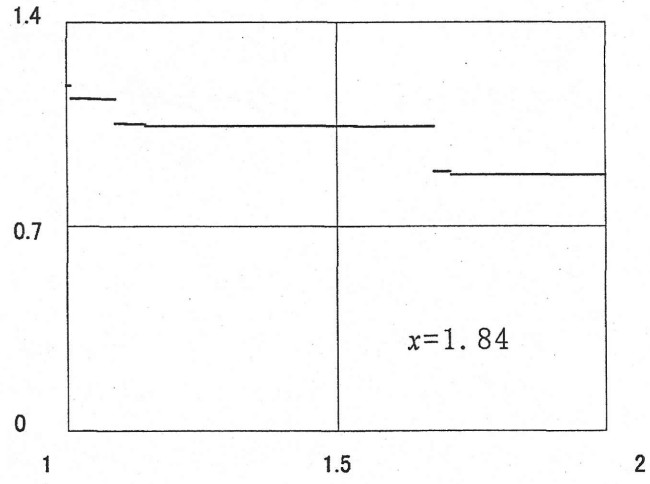

1.4
0.7
0
1
1.5
2
x=1.14

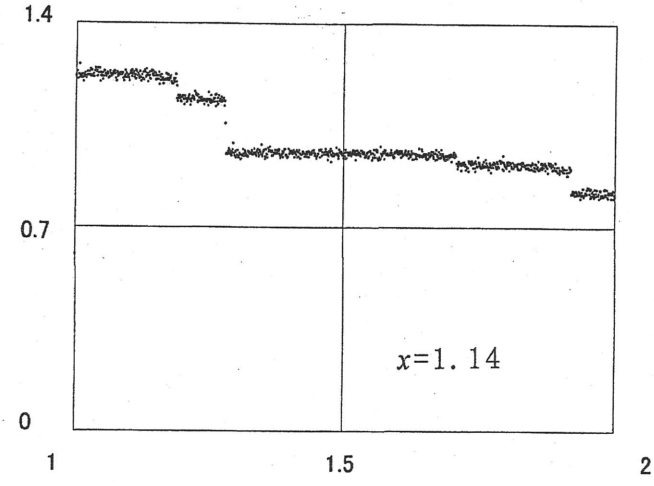

1.4
0.7
0
1
1.5
2
x=1.28

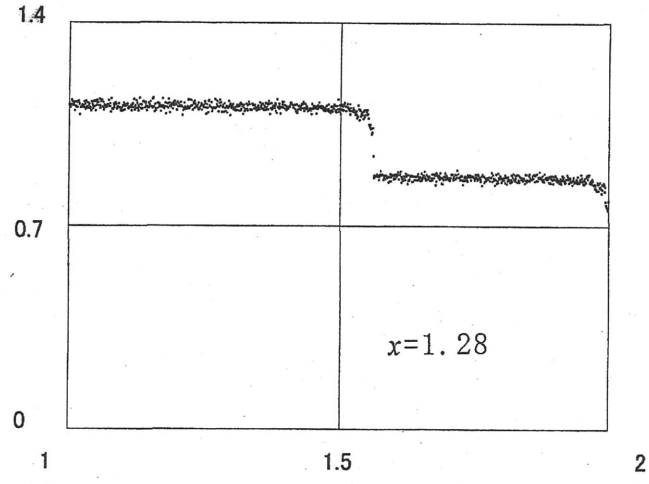

1.4
0.7
0
1
1.5
2
x=1.42

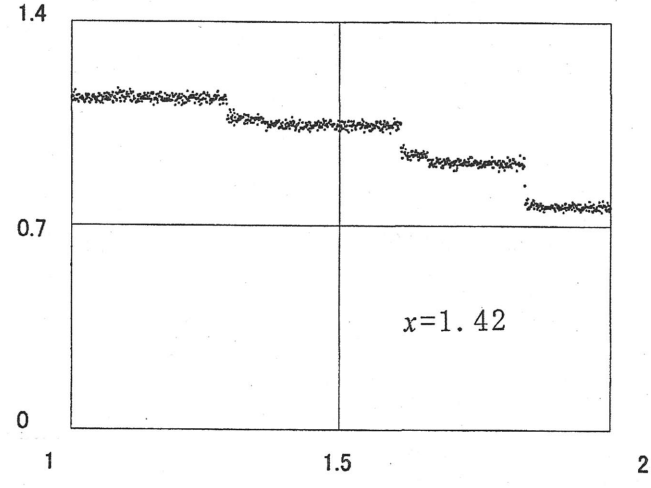

1.4
0.7
0
1
1.5
2
x=1.56

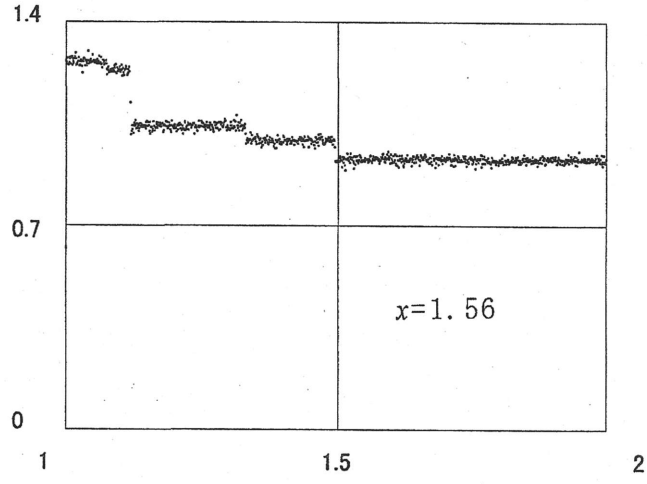

1.4
0.7
0
1
1.5
2
x=1.7

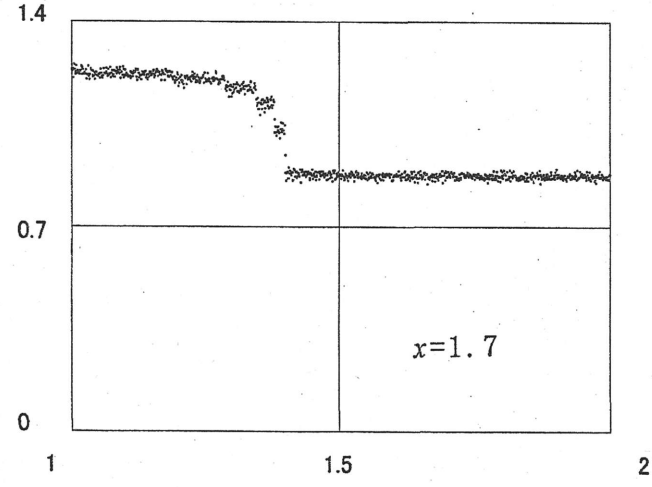

1.4
0.7
0
1
1.5
2
x=1.84

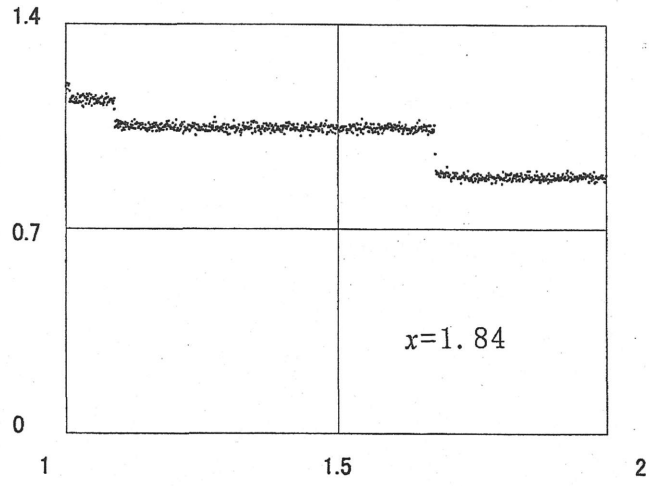

1.4
0.7
0
1
1.5
2
x=1.14


1.4
0.7
0
1
1.5
2
x=1.28


1.4
0.7
0
1
1.5
2
x=1.42


1.4
0.7
0
1
1.5
2
x=1.56


1.4
0.7
0
1
1.5
2
x=1.7


1.4
0.7
0
1
1.5
2
x=1.84

1.4 写像 Φ_x の不変密度関数の平均とＳＳＲ計算値の分布

ＳＳＲ乱数生成法は、以下で定める各 x_k , k =1, 2, 3, ･･･, ごとに $\Phi_{x_k}^{24}(1)$ を計算し、最初の３桁を棄て続く４桁を乱数値として採用するものであった。ここで、x_k は、

$$(r_k, s_k) \equiv (rk \bmod p,\ sk \bmod q) ,$$

として、

$$x_k = \begin{cases} 1.0 + \dfrac{r_k}{a+s_k} & \text{if} \quad (a+s_k) > r_k \\ 1.0 + \dfrac{r_k-(a+s_k)}{b-s_k} & \text{if} \quad (a+s_k) \leq r_k , \end{cases}$$

ただし、

p=49933453, q=22801201, r=491377, s=47513,
a=1920000, b=48060000,

と定めるが（註：実際のプログラムでは、x_k を定める分数の分母に +3, 分子には +1 を行っている）、その定め方は（ここでは）さほど重要ではない。重要なのは $\Phi_{x_k}^{24}(1)$ において x_k を [1, 2) 上で一様に変化させることにより多くの乱数値を得ているということである。久保氏のアイデアは、

［主張］ $\{\Phi_{x_k}^{24}(1)\}_{k=1,2,\ldots}$ の分布は、前出の h_x, $x \in [1,2)$, を x について平均した密度関数 $H(t)$、すなわち、

$$H(t) \equiv \int_1^2 h_x(t)\,dx , \quad t \in [1,2),$$

で記述される、

というものである。この主張は、次のような推論に基づく（註：数学的には必ずしも厳密ではない）。

q が十分大きければ、$\{\Phi_{x_k}^{q}(1)\}_{k=1,2,\ldots}$ の分布は、q の値にかかわらず安定状態に達し、安定状態に達した $\{\Phi_{x_k}^{q}(1)\}_{k=1,2,\ldots}$ の分布は、ある [1, 2) 上の確率密度関数 $H(t)$ により記述される。f を [1, 2) 上の任意の有界可測関数とすると、N が十分大きければ、

$$\frac{1}{N}\sum_{k=1}^{N} f(\Phi_{x_k}^{q}(1)) \approx \int_1^2 f(t)H(t)\,dt$$

と近似されると考えられる。この式の両辺を、$q = Q,\ Q+1,\ \cdots,\ Q+R-1$ について

和をとり平均すれば、

$$\frac{1}{N}\sum_{k=1}^{N}\frac{f(\Phi_{x_k}^{Q}(1))+f(\Phi_{x_k}^{Q+1}(1))+\cdots+f(\Phi_{x_k}^{Q+R-1}(1))}{R}\approx\int_1^2 f(t)H(t)\,dt$$

となる。R を十分大きくして、分数部分にエルゴード定理を適用すると、左辺は、

$$\frac{1}{N}\sum_{k=1}^{N}\int_1^2 f(t)h_{x_k}(t)\,dt$$

に近づき、したがって、

$$\int_1^2 f(t)\left(\frac{1}{N}\sum_{k=1}^{N}h_{x_k}(t)\right)dt\approx\int_1^2 f(t)H(t)\,dt$$

となる。ここで、左辺の N を大きくしていくと、x_k は $[1,2)$ 上を一様に分布するので、

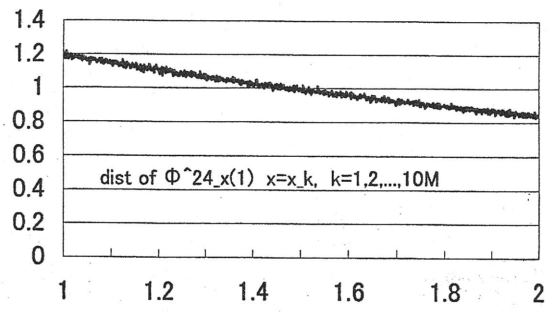
$$\int_1^2 f(t)\left(\frac{1}{N}\sum_{k=1}^{N}h_{x_k}(t)\right)dt\ \longrightarrow\ \int_1^2 f(t)\left(\int_1^2 h_x(t)\,dx\right)dt$$

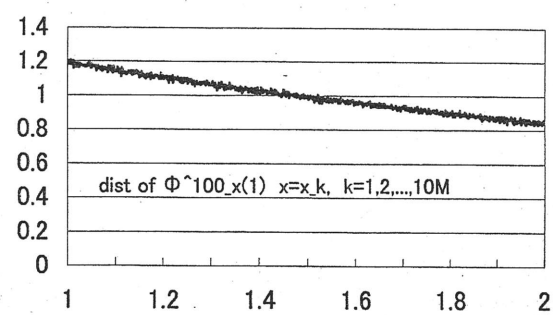
となる。f は任意であるので、結局、

$$H(t)=\int_1^2 h_x(t)\,dx$$

であることが予想される。

早速［主張］を数値実験で確かめてみよう。まず q が大きければ、$\{\Phi_{x_k}^{q}(1)\}_{k=1,2,\ldots}$ の分布が安定状態に達することを見てみる。以下は、$q=24,\ 100$ に対する、$\Phi_{x_k}^{q}(1)=1.d_1d_2d_3\cdots,\ k=1,\cdots,\ 10000000,$ の分布のグラフである。これから、$q=24,\ 25,\ \cdots$ に対して、$\{\Phi_{x_k}^{q}(1)\}$ の分布は、十分安定していると考えられる。



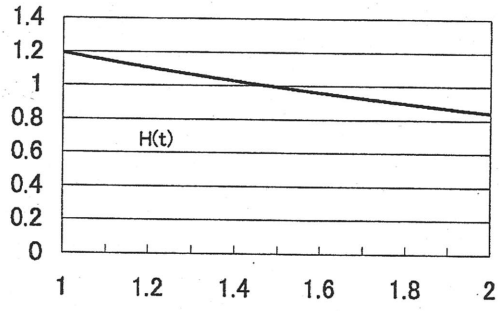


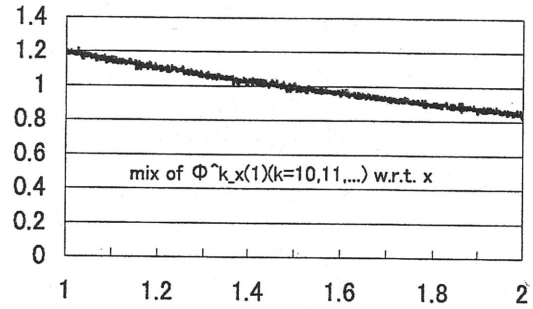
次に、$H(t)$ を求めてみる。$H(t)$ はとても数式では書き表せそうもないので、数値計算を行うことになる。h_x の x として、$l_s=1.0+\frac{s}{997}$，$s=1,2,\cdots,996,$

を選ぶ。次に、前節と同様にして $h_{l_s}(c_i)$, $i = 0, 1, \cdots, 9999$, を求める。そして、$h_{l_s}(c_i)$, $s = 1, 2, \cdots, 996$, を平均して $H(c_i)$ を求める：

$$H(c_i) = \frac{1}{996} \sum_{s=1}^{996} h_{l_s}(c_i) .$$

$H(c_i)$, $i = 0, 1, \cdots, 9999$, のグラフは下図左のようになる。このグラフの妥当性を確認するため、実際に、$\Phi^q_{l_s}(1)$, $s = 1, 2, \cdots, 996$, $q = 10, \cdots, 10009$, を計算し、得られた 996×10000 個の値 $1.d_1d_2d_3\cdots$ の分布を下図右に示しておく（縦軸は $1.d_1d_2d_3$ の出現回数を 9960 で割ったもの）。





以上４つのグラフはほとんど見分けがつかず、久保氏のアイデアには相当な説得力があることが分る。

1.5 ＳＳＲ計算値の分布の検定

前節で、$\{\Phi^{24}_{x_k}(1)\}$ の分布が、密度関数 $H(t)$ で記述される、という予想を得たので、これを統計的に検定してみる。検定は、「検定(IX) Kolmogorov-Smirnov 統計量のχ^2検定」を準用する。具体的には、検定(IX)で基準となる分布を一様分布から $H(t)$ に変え、また１回の検定に使う乱数の個数を 20000 個に変えて、以下の様に行う：

［検定Ｘ］20000 個の $z_k = \Phi^{24}_{x_k}(1)$ からなる数値列

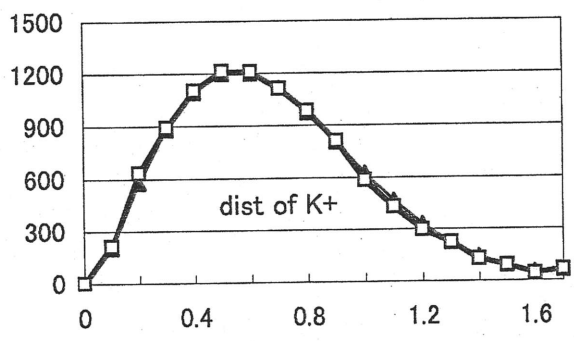
$$z_{1+20000\times j},\quad z_{2+20000\times j},\quad \cdots,\quad z_{20000+20000\times j}$$

を１万系列(j=0, 1, . . . , 9999)発生させる。次に基準となる分布を

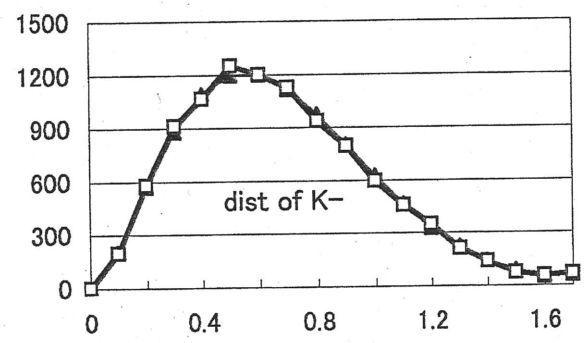
$$F(t) = \left\{\sum_{i:t_i<t} H(t_i)\right\} \Big/ 10000\ ,\quad t_i = 1 + \frac{i}{10000}\ ,\quad i = 0, 1, 2, \cdots, 9999$$

として、各 j 系列ごとに Kolmogorov-Smirnov 統計量 κ_j^+ および κ_j^- を計算する。この 1 万個の κ_j^+ [κ_j^-] の分布は，確率分布関数 $P(x)$ = $1-\exp(-2x^2)$ にほぼしたがうことが知られているので，χ^2検定を行うことができる。

κ_j^+ , κ_j^- の分布のグラフ、およびχ^2検定の結果は次の様になる。



CHITEST 0.20098



CHITEST 0.66702

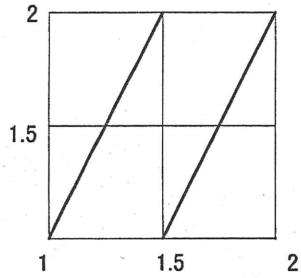
これらの結果から、

$\{\Phi^{24}_{x_k}(1)\}$ の分布が、密度関数 $H(t)$ で記述される、

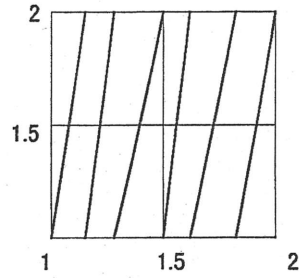
という仮説は棄却されないことが分る。

1.6 ＳＳＲ計算値の条件付き分布

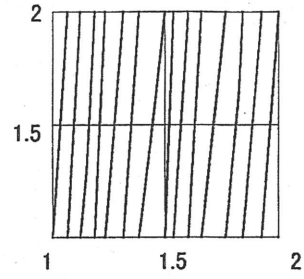
ＳＳＲ計算値は $\Phi_x^{24}(1)$ により得られるが、$w_k = \Phi_x^k(1)$ を x の関数とみて、w_1 , w_2 , w_3 のグラフを描いてみると以下の様になる。



w_1



w_2



w_3

図および §1.2 の考察から、w_{k-1} から w_k に到る過程 $\varphi_s \circ \varphi_e \circ \varphi_x (\equiv \Phi_x)$ 中の φ_x では分岐は増えないが、φ_e では１本増える。この分岐は φ_s でさらに２倍になる。したがって w_k における分岐数を γ_k とすると

$$\begin{cases} \gamma_k = 2(\gamma_{k-1} + 1) , & k=1, 2, \cdots, 24, \\ \gamma_0 = 0 \end{cases}$$

の関係がある。よって、

命題 w_{24} における分岐本数は、

$$\gamma_{24} = 2^{25} - 2 = 33,554,430$$

となる。

［演習］ このことを示せ。

ここで、

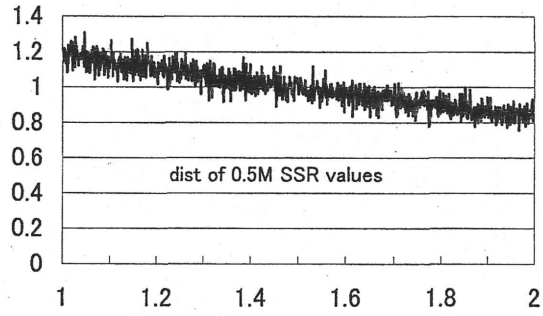
$$\Delta = \frac{1}{500000} ,$$

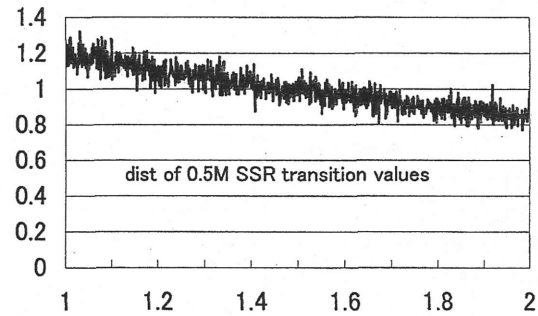
とおいて

$$v_i = 1 + \Delta i , \quad i = 0, 1, 2, \cdots, 499999 ,$$

とおく。（上の命題から $\Delta = v_{i+1} - v_i$ の中には、平均で 67 本程度の w_{24} の分

岐があると考えられる。）このとき、$\Phi_{\nu_i}^{24}(1)$, $i=0,1,\cdots$, の分布は、$H(t)$ で記述される（下図左参照）。我々が興味を持つのは、任意の u を固定したとき、$\Phi_{\nu}^{24}(1)=u$ となるような ν について（このような ν は γ_{24} 個ある）$\Phi_{\nu+\Delta}^{24}(1)$ はどのような値分布をするか、ということである。様子をさぐるため、$u=1.1$ とおき、$\Phi_{\nu}^{24}(1)=1.1$ を満たす ν として、ν_i の右側にあって最初に $\Phi_{\nu}^{24}(1)=1.1$ を満たす ν を ν_i' とおく（i=0, 1, 2, $\cdots$, 499999）。そして、$\Phi_{\nu_i'+\Delta}^{24}(1)$ の分布を調べてみる（下図右）。





図をみると $\Phi_{\nu_i'+\Delta}^{24}(1)$ の分布もやはり $H(t)$ にしたがうようである。よって、

［**主張**］u を固定したとき、$\Phi_{\nu}^{24}(1)=u$ となるような ν について、$\Phi_{\nu+\Delta}^{24}(1)$ の分布は、Δ が極端に小さくなければ、やはり $H(t)$ で記述される、

について本格的に統計的検定を行ってみる。

（註：$\Delta=500000^{-1}=0.2\times10^{-5}$ としたが、我々が乱数生成に使っている $\Phi_{x_k}^{24}(1)$, $k=1,2,\cdots$, においては、大体において、$x_{k+1}-x_k\geq0.009$ である。）

調べ方は、まず、

x_k ($k=1,2,\cdots$)：今までのもの（§1.4 の方法で作る），

$x_k'(u) = \min\{x \mid \Phi_x^{24}(1)=u,\ x\geq x_k\}$，

$= (x_k$ の右側にあって最初に $\Phi_x^{24}(1)=u$ となる $x)$，

$y_k(u) = \Phi_{x_k'(u)+\Delta}^{24}(1)$

とおく（$x_k'(u)$ の実際の計算精度については、本節最後を参照）。次に、10000個の

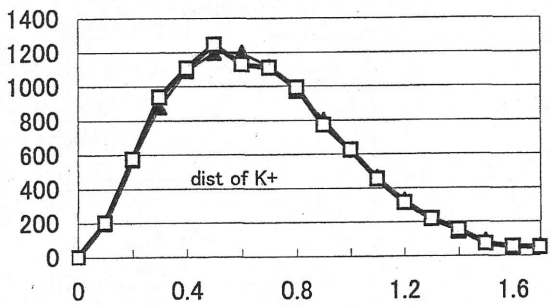
$$u_j = 1.0 + \frac{j}{10001}\ ,\ j = 1, 2, \cdots, 10000,$$

ごとに、20000 個の

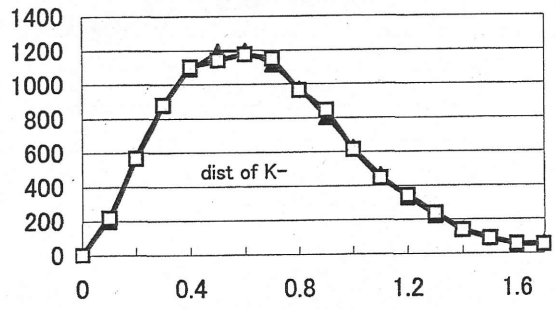
$$y_{1+20000j}(u_j)\ ,\ y_{2+20000j}(u_j)\ ,\ \cdots\ ,\ y_{20000+20000j}(u_j)\ ,$$

を求める。この （20000個／系列） × （10000系列） の数値列に対して、［検定X］を適用する。すなわち、各系列に対し $H(t)$ から求めた $F(t)$ を基準分布にして Kolmogorov-Smirnov 検定を行い、その検定値 κ_j^+ $[\kappa_j^-]$, $j = 1, \cdots, 10000$, の分布を χ^2 検定する。

結果は次のとおりである。



CHITEST = 0.28893



CHITEST = 0.65665

よって［主張］は棄却されないことが分かる。この結果から、我々の $\Phi_{x_k}^{24}(1)$ について、

> $\Phi_{x_k}^{24}(1)$ がある値をとった時，次の $\Phi_{x_{k+1}}^{24}(1)$ の値は、deterministic に計算されるにもかかわらず、あたかも
>
> $H(t)$ で分布するの値の中からランダムに選ばれた
>
> ように見える、

と考えることができる。このことは、

> 数値列 $\Phi_{x_k}^{24}(1)$, $k = 1, 2, \cdots$, は、あたかも、出現値の密度分布が $H(t)$ であるような試行の，独立な繰り返しにおける，１つのパスのように見える、

と解釈出来ることをも意味している。後節では、このパスを色々変化させて多くの数値（乱数）系列を作り、それらの特性を調べている。

最後に、$\Phi_x^{24}(1)=u$ を満たす x の精度であるが、数値計算で求めた x の近似値を x' とするとき、u と $\Phi_{x'}^{24}(1)=u'$ の１０進表記($u'=1.u'_1u'_2u'_3\cdots$)が何桁目から不一致になったかを、上の検定で現れた 20000×10000 個の $\Phi_{x'}^{24}(1)$ について調べて見ると次の様になる。（下表で、不一致位置が l であるとは、

$$1.u_1u_2u_3\cdots u_{l-1}=1.u'_1u'_2u'_3\cdots u'_{l-1}\ ,\quad u_l\neq u'_l\ ,$$

を意味する。）

位置 l	1	2	3	4	5	6	7	8	9	10	11
回数	0	0	0	0	0	0	0	48	30	4027	597240

12	13	14	15	16	17	18	19	20	21
2559911	744805	84479	8453	929	70	6	2	0	0

$\Phi_{x_k}^{24}(1)$ から乱数を作るときは、最初の３桁を棄て、続く４桁を乱数として採用するので、上表で７桁以上が同じならば、$\Phi_{x'}^{24}(1)$ は u と同じ乱数値 $u_3u_4u_5u_6$ を生成することになる。

1.7 写像 Φ_x のカオス性

一般に、距離空間 X 上の写像 $f:X\longrightarrow X$ がカオス写像であることは、以下の i), ii), iii) で特徴付けられる [ ]。

i) f が不変集合 Y 上で初期値に対する敏感な依存性をもつ、すなわち、
$$\forall r>0,\ \forall x\in Y,\ \forall \varepsilon>0,\ \exists y\in Y,\ \exists k\geq 0;\ d(f^k(x),f^k(y))>r$$
ii) f は、Y で推移的である、すなわち、点 p の正の向きの軌道が Y で稠密になる、
iii) 周期点が Y で稠密である。

このことを Φ_x について簡単に（≈数学的には厳密ではないが）状況を見てみよう。まず、$\frac{d}{dt}\Phi_x>1$, $x\in[1,2)$, であることから、Φ_x は拡張的(expansive)な写像であり、i) であることが理解される。次に、§１．６の w_k の写像と $y=x$ のグラフの交点が周期点であることから iii) が分かる。また、Φ_x の不変測度の

確率密度関数 h_x のグラフを §1．3 に見ると、値が 0.7 以上であり、このことから $\{\Phi_x^n(1)\}_n$ は、多少の偏りはあるにしても [1,2) で一様に分布する。これは、f が ii) を満たすことを意味する。以上のことより、Φ_x は、i)-iii) の意味でカオス写像であることが分かる。

2．ＳＳＲ計算値の分布特性の改良

ＳＳＲ計算値 $\Phi_{x_k}^{24}(1)$, $k=1,2,\cdots$, から乱数列を作る際は、$\Phi_{x_k}^{24}(1)$ の最初の３桁を棄てている。これにより一様性は大分改善されるが、もともとの数値列 $\{\Phi_{x_k}^{24}(1)\}$ の分布 $H(t)$ が下に凸な単調減少関数であることから、３桁を棄てたとしてもやはりこの性質、特に分布の非対称性、が僅かながら乱数列に残ってしまう。この節では、乱数値の非対称性を改善する方法について考察する。

まず、久保氏の改良法を紹介する。

【Ｋ改良ＳＳＲ計算】(SSRK)

$$g(x) = w_0 \underbrace{\cdot x \cdot x \cdot x \cdot \cdots \cdot x \cdot x}_{24}, \quad x \in [1,2)$$

を２つの初期値

$$w_0 = 1.2718281828459 \ (=1.e)$$
$$w_0 = 1.8141592653589 \ (=1.\tilde{\pi})$$

のそれぞれの場合について、今までどおりＳＳＲ計算し

$$g_e = \Phi_x^{24}(1.e), \qquad \tilde{g}_p = \Phi_x^{24}(1.\tilde{\pi})$$

を得る。そして $g_e - \tilde{g}_p \pmod 1$ を $\Phi_x^{24}(1)$ の代りとする。

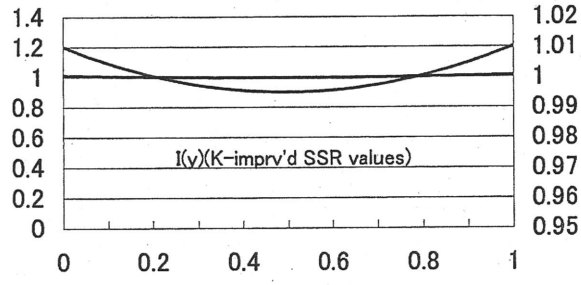
この改良法は、カオス写像の性質から予想される、x が変化について g_e と $\tilde{g}_p$ はほぼ独立であるということを巧みに利用している。この場合、$g_e - \tilde{g}_p$ の分布の密度関数 $I(y)$, $y \in [0,1)$, は $H(t)$ からの畳み込み

$$I(y) = \int_1^{1+y} H(t)H(t-y+1)\,dt + \int_{1+y}^{2} H(t)H(t-y)\,dt$$

で得られるので、$y_i = i/10000$, $i=0,1,\cdots,9999$, $c_j = 1+j/10000$, $j=0,1,\cdots,9999$, について

$$I(y_i) = \frac{1}{10000} \sum_{j=0}^{9999} H(c_j)H(c_j - y_i \bmod [1,2))$$

と計算される。$I(y_i)$ のグラフを以下に示す。（註：同じグラフをスケールを変えて二重表示している；以下同様。）



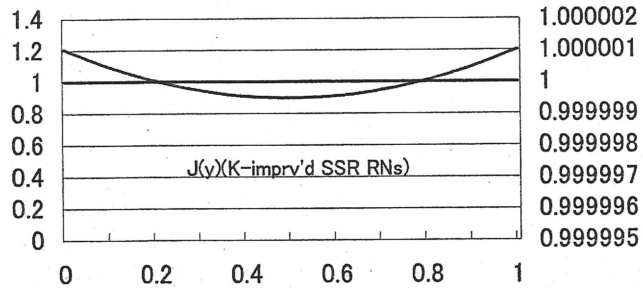
ＳＳＲ計算値に「Ｋ改良」を施して得た値 $0.d_1d_2d_3d_4\cdots$ から d_1d_2 を乗てて得られる乱数値 $0.d_3d_4d_5d_6$ の分布 $J(y)$ について考えてみる。「Ｋ改良」を施して得た計算値の密度分布関数 $I(y)$, $y\in[0,1)$, は、数値計算により $I(c_i)$, $c_i=\frac{i}{10000}$, $i=0,1,\cdots,9999$, が求められていた。一般に、ランダムに現れる値 $0.d_1d_2d_3d_4\cdots$ から d_1d_2 の２桁を乗てたとき $0.d_3d_4d_5d_6$ となる数は $0.{**}d_3d_4d_5d_6$ (** は任意数) である。したがって $0.d_3d_4d_5d_6\cdots$ の分布の密度関数 $J(y)$, $y\in[0,1)$, は

$$J(0.d_3d_4d_5d_6\cdots)=\frac{1}{100}\sum_{d_1=0}^{9}\sum_{d_2=0}^{9}I(0.d_1d_2d_3d_4d_5d_6\cdots)$$

となる。特に、$I(c_i)$, $c_i=0.d_1d_2d_3d_4$, から $J(y)$ を $y=0.00$, 0.01, $\cdots$, 0.99, において求めるには、

$$J(0.d_3d_4)=\frac{1}{100}\sum_{d_1=0}^{9}\sum_{d_2=0}^{9}I(0.d_1d_2d_3d_4)$$

とすればよい。$J(y)$ のグラフは以下の通りであり、特性が $I(y)$ に比べて相当改良されていることが分る。



次に、K改良をヒントにして、谷口の従来からの改良法（Y改良、仮数部のビット反転による改良）を発展させた方法を紹介する。

新たな改良法「X改良」は、「K改良」と「Y改良」の中間的な性格を持つ。すなわち、「Y改良」では特定のビットを調べ、ある条件のもとで仮数部の全てのビットを XOR で反転していたが、この「ある条件のもとで...全てのビット」が曲者で、以下で述べるＮＩＳＴの検定の approximate entropy test で棄却されてしまうことが分かった。「X改良」では、仮数部のビット反転を、別のSSRex計算で得られた仮数部と XOR をとることにより、ランダムに行おうというものである。

【X改良ＳＳＲ計算】(SSRX)

$$g(x) = w_0 \underbrace{\cdot x \cdot x \cdot x \cdot \cdots \cdot x \cdot x}_{24}, \quad x \in [1,2)$$

を初期値　$w_0 = 1.2718281828459\ (=1.e)$　について、今までどおりＳＳＲ計算し g_e を得る。また、

$$\hat{g}(x) = w_0 \underbrace{\cdot x \cdot x \cdot \cdots \cdot x}_{10}, \quad x \in [1,2)$$

を初期値　$w_0 = 1.8141592653589\ (=1.\tilde{\pi})$　についてＳＳＲ計算し、$\hat{g}_p$ を得る。ただし、仮数部の全ビットのシフトのところは、

i) 仮数部の全てのビット値を８ビット左にシフトし，

と変更する。そして（g_e xor $\hat{g}_p$）をＳＳＲ計算値の代りとする。

したがって、「X改良」では、ＳＳＲ計算値 $g_e(x)$ の仮数部のビット反転を、$\hat{g}_p$ のビット値を使ってランダムに行っている。（g_e xor $\hat{g}_p$）の分布特性は、数式では記述しがたいが、$\hat{g}_p$ が g_e と独立に乱数を発生するとすれば、かなりの特性改良が期待される。（ここでは、$\hat{g}_p$ のＳＳＲ計算で、x を掛ける回数を10回、１回あたりのシフトを８ビットとしたが、乱数特性が良く、計算回数が少なくて済む値があればそれを使えばよい。）

乱数の生成速度は、[検定IX]の実行時間で比べると、

「改良なし」　：　「K改良」　：　「X改良」

＝　1　：　1.073　：　1.042　(MPU は Itanium)

である。

3．ＳＳＲ計算値の分布を近似する関数

3.1 分布 $H(t)$ を近似する関数 $\tilde{H}(t)$

ＳＳＲ計算値 $\Phi_{x_k}^{24}(1)$, $k=1,2,\cdots$, の分布が、密度関数 $H(t)$ で記述されることは既に述べた。この $H(t)$ を数式で表すことは難しく、数値計算でその値を求めてきたが、これでは再現性、移動性が乏しく不便である。うまい具合に $H(t)$ を近似する関数 $\tilde{H}(t)$,

$$\tilde{H}(t) \equiv \left\{\frac{1}{1+t} + \frac{1}{2+t}\right\}\frac{1}{\log 2} ,$$

が見つかったので紹介する。

ＳＳＲ計算の基礎となる写像 $\Phi_x:[1,2)\to[1,2)$ は $\Phi_x \equiv \varphi_s \circ \varphi_e \circ \varphi_x$ と表されるが、φ_s を除いた写像 $\Psi_x(t) \equiv \varphi_e \circ \varphi_x(t)$, $t\in[1,2)$, は、

$$\Psi_x(t) = \begin{cases} xt & (1 \le t < \frac{2}{x} \text{ のとき}) \\ \\ \frac{1}{2}xt & (\frac{2}{x} \le t < 2 \text{ のとき}) \end{cases}$$

である。写像 $\Psi_x(t)$ が導く Perron-Frobenius 作用素 $\tilde{\mathcal{L}}_x : L^1 \longrightarrow L^1$, は、

$\tilde{\mathcal{L}}_x : k \longmapsto (\tilde{\mathcal{L}}_x k)$, ただし,

$$(\tilde{\mathcal{L}}_x k)(t) = \begin{cases} \frac{2}{x}k(\frac{2t}{x}) & (1\le t<x \text{ のとき}) \\ \\ \frac{1}{x}k(\frac{t}{x}) & (x\le t<2 \text{ のとき}) \end{cases}$$

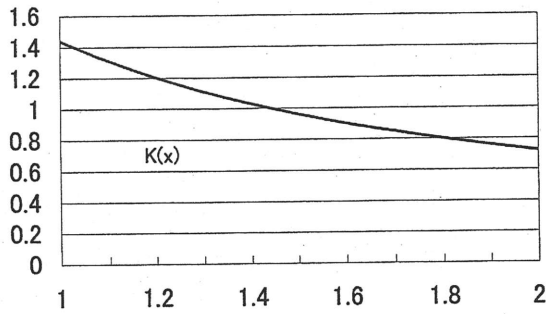
となる。久保氏は、$\tilde{\mathcal{L}}_x$ の不動点関数 $k_x(t)$ が x によらず

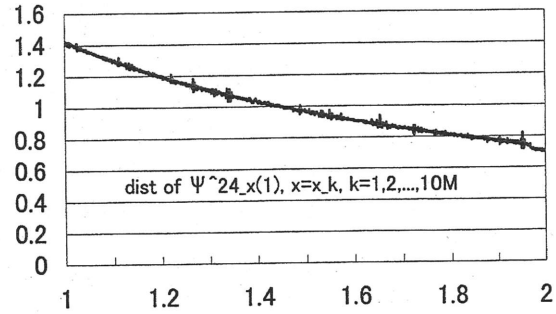
$$K(t) = \frac{1}{t\log 2} , \quad t\in[1,2)$$

であることを見つけた。

［演習］ $K(t) = \frac{1}{t\log 2}$ が、$\tilde{\mathcal{L}}_x$ の不動点関数であることを示せ。

すると、今までの議論を再現すれば、$\Psi_{x_k}^{24}(1)$, $k=1,2,3,\cdots$, の分布は、$\Psi_x(t)$ の不変測度の密度関数 $k_x(t)=K(t)$, $x\in[1,2)$, を x について平均した密度関数 $K(t)=\frac{1}{t\log 2}$ で記述されることになる。実際 §1.4 と同様に $K(t)$ のグラフと、$\Psi_{x_k}^{24}(1)=1.d_1d_2d_3\cdots$, $k=1,\cdots,10000000$, の分布のグラフを作ってみると、そのことが理解される。





この $K(t)$ を用いて $H(t)$ を近似することを考えよう。最初に

$$\Phi_x^{24}(1)=\Phi_x(\Phi_x^{23}(1))=(\varphi_s\circ\varphi_e\circ\varphi_x)(\Phi_x^{23}(1))=(\varphi_s\circ\Psi_x)(\Phi_x^{23}(1))$$

と書けることに注意する。

[演習] シフト φ_s に対応する Perron-Frobenius 作用素を $\mathcal{L}_{\varphi_s}$ とするとき、

$\mathcal{L}_x=\mathcal{L}_{\varphi_s}\circ\mathcal{L}_x^{\sim}$ を示せ。($\mathcal{L}_x$ は Φ_x に対応する Perron-Frobenius 作用素。§1.3 参照。)

そこで、§1.4 の

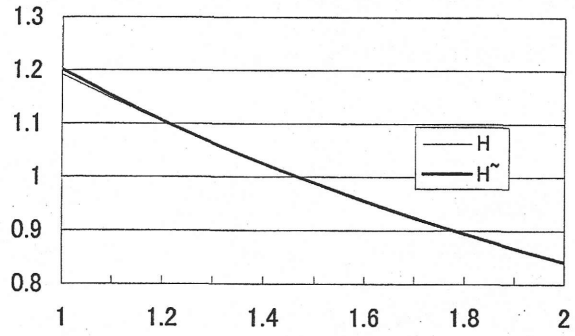
$$H(t)\equiv\int_1^2 h_x(t)\,dx=\int_1^2(\mathcal{L}_x h_x)(t)\,dx\quad(\because\ \mathcal{L}_x h_x=h_x)$$

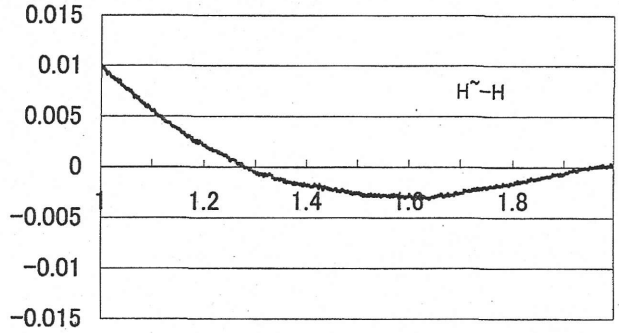
において、少々乱暴であるが $h_x(t)\simeq K(t)$ と考えると(これは、集合 $\{\Phi_{x_k}^{23}(1)\}$ の分布が、Ψ_x の定常分布 $K(t)$ からひどくはずれていないと思うことに相当する)、

$$H(t)\simeq\int_1^2(\mathcal{L}_x K)(t)\,dx=\int_1^2\left((\mathcal{L}_{\varphi_s}\circ\mathcal{L}_x^{\sim})K\right)(t)\,dx=\int_1^2(\mathcal{L}_{\varphi_s}K)(t)\,dx$$

$$=\int_1^2\frac{1}{2}\left\{K\left(\frac{1}{2}+\frac{t}{2}\right)+K\left(1+\frac{t}{2}\right)\right\}dx=\left\{\frac{1}{1+t}+\frac{1}{2+t}\right\}\frac{1}{\log 2}=\tilde{H}(t)$$

を得る。以下に、$\tilde{H}(t)$, $\tilde{H}(t)-H(t)$ のグラフを示しておく。





$\tilde{H}(t)$ から「K改良」後の密度関数 $I(y)$ に相当する関数 $\tilde{I}(y)$, $y\in[0,1)$, を求めてみると

$$\tilde{I}(y) = \int_1^{1+y} \tilde{H}(t)\tilde{H}(t-y+1)\,dt + \int_{1+y}^{2} \tilde{H}(t)\tilde{H}(t-y)\,dt$$

$$= \left(\frac{1}{\log 2}\right)^2 \int_1^{1+y} \left\{\frac{1}{(1+t)} + \frac{1}{(2+t)}\right\}\left\{\frac{1}{(2+t-y)} + \frac{1}{(3+t-y)}\right\} dt$$

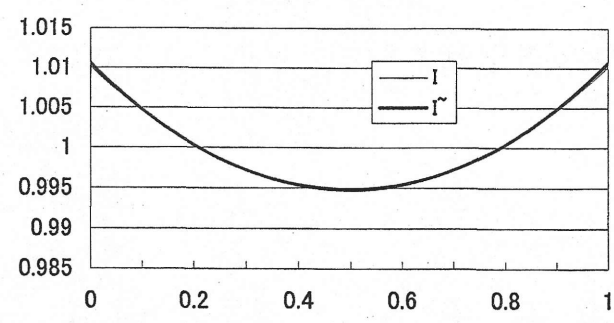
$$+ \left(\frac{1}{\log 2}\right)^2 \int_{1+y}^{2} \left\{\frac{1}{(1+t)} + \frac{1}{(2+t)}\right\}\left\{\frac{1}{(1+t-y)} + \frac{1}{(2+t-y)}\right\} dt$$

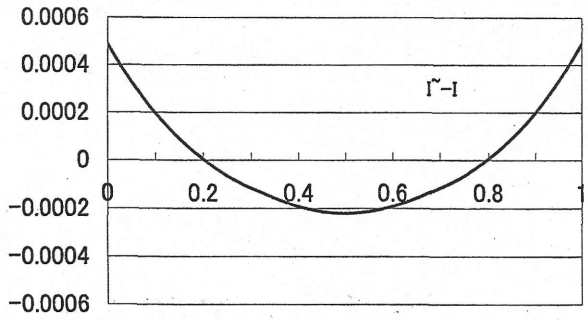
$$= I_1 + I_2$$

$$I_1 = \left(\frac{1}{\log 2}\right)^2 \left\{\frac{1}{(1-y)} \log \frac{(3+y)\cdot(3-y)}{4\cdot 2} + \frac{1}{y} \log \frac{(2+y)\cdot(4-y)}{4\cdot 2}\right\}$$

$$I_2 = \left(\frac{1}{\log 2}\right)^2 \left\{\frac{1}{(1+y)} \log \frac{(3+y)\cdot(3-y)}{4\cdot 2} + \frac{1}{(2-y)} \log \frac{(2+y)\cdot(4-y)}{4\cdot 2}\right\}$$

となる。$\tilde{I}(y)$ および $\tilde{I}(y)-I(y)$ のグラフを以下に示しておく。$\tilde{I}(y)$ が $I(y)$ を非常に良く近似していることが分る。

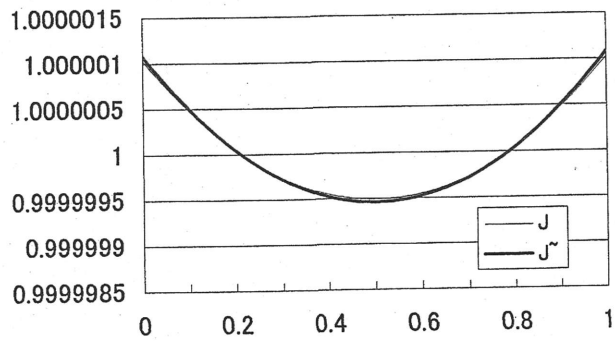


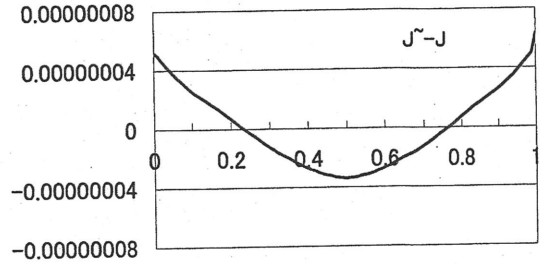


ことのついでに、K改良ＳＳＲ計算値 $0.d_1d_2d_3d_4\cdots$ から d_1d_2 を棄てて得られる乱数値 $0.d_3d_4d_5d_6$ の分布 $J(y)$ を近似する関数 $\tilde{J}(y)$ を $\tilde{I}(y)$ から求めておくと、

$$\tilde{J}(y) = \frac{1}{100}\sum_{l=0}^{99}\tilde{I}\left(\frac{l}{100}+\frac{y}{100}\right), \quad y\in[0,1)$$

となる。$\tilde{J}(y)$, $\tilde{J}(y)-J(y)$ のグラフは次のとおりである。





3.2 関数 $\tilde{H}(t)$ の精密化

分布 $H(t)$ を近似する関数 $\tilde{H}(t)$ はかなり良い精度をもっているので、実用的にはこれで十分と思われるが、分布 $\tilde{H}(t)$ にもう一度 Perron-Frobenius 作用素 $\mathcal{L}_x$ を作用させて、x について平均をとると、もう少し精密な近似関数 $\hat{H}(t)$ が得られるので、以下丁寧に計算してみよう。

写像 Φ_x に対応する Perron-Frobenius 作用素 $\mathcal{L}_x : L^1 \longrightarrow L^1$, は

$\mathcal{L}_x : h \longmapsto (\mathcal{L}_x h)$, ただし,

$1\le x<1.5$ のとき

$$(\mathcal{L}_x h)(t) = \begin{cases} \frac{1}{2x}h\left(\frac{t+2}{2x}\right) + \frac{1}{x}h\left(\frac{t+1}{x}\right) & 1\le t<2x-1 \text{ のとき} \\ \frac{1}{2x}h\left(\frac{t+1}{2x}\right) + \frac{1}{2x}h\left(\frac{t+2}{2x}\right) & 2x-1\le t<2 \text{ のとき} \end{cases}$$

$1.5\le x<2$ のとき

$$(\mathcal{L}_x h)(t) = \begin{cases} \frac{1}{x}h\left(\frac{t+1}{x}\right) + \frac{1}{x}h\left(\frac{t+2}{x}\right) & 1\le t<2x-2 \text{ のとき} \\ \frac{1}{2x}h\left(\frac{t+2}{2x}\right) + \frac{1}{x}h\left(\frac{t+1}{x}\right) & 2x-2\le t<2 \text{ のとき} \end{cases}$$

であることは既に述べた。これを書き直すと、$t\in[1,2)$ を決めたとき、

$$(\mathscr{L}_x h)(t) = \begin{cases} \frac{1}{2x}h(\frac{t+1}{2x}) + \frac{1}{2x}h(\frac{t+2}{2x}) & 1 \le x \le \frac{t+1}{2} \text{ のとき} \\ \frac{1}{2x}h(\frac{t+2}{2x}) + \frac{1}{x}h(\frac{t+1}{x}) & \frac{t+1}{2} < x \le \frac{t+2}{2} \text{ のとき} \\ \frac{1}{x}h(\frac{t+1}{x}) + \frac{1}{x}h(\frac{t+2}{x}) & \frac{t+2}{2} < x < 2 \text{ のとき} \end{cases}$$

と表されるので、$(\mathscr{L}_x h)(t)$ の x についての平均は、

$$\int_1^{(t+1)/2} \left\{\frac{1}{2x}h(\frac{t+1}{2x}) + \frac{1}{2x}h(\frac{t+2}{2x})\right\}dx + \int_{(t+1)/2}^{(t+2)/2} \left\{\frac{1}{2x}h(\frac{t+2}{2x}) + \frac{1}{x}h(\frac{t+1}{x})\right\}dx$$
$$+ \int_{(t+2)/2}^{2} \left\{\frac{1}{x}h(\frac{t+1}{x}) + \frac{1}{x}h(\frac{t+2}{x})\right\}dx$$
$$= \int_1^{(t+1)/2} \frac{1}{2x}h(\frac{t+1}{2x})\,dx + \int_1^{(t+2)/2} \frac{1}{2x}h(\frac{t+2}{2x})\,dx$$
$$+ \int_{(t+1)/2}^{2} \frac{1}{x}h(\frac{t+1}{x})\,dx + \int_{(t+2)/2}^{2} \frac{1}{x}h(\frac{t+2}{x})\,dx$$
$$= \frac{1}{2}\int_{2/(t+1)}^{1} \frac{1}{u}h(\frac{1}{u})\,du + \frac{1}{2}\int_{2/(t+2)}^{1} \frac{1}{u}h(\frac{1}{u})\,du$$
$$+ \int_{1/2}^{2/(t+1)} \frac{1}{u}h(\frac{1}{u})\,\mathrm{du} + \int_{1/2}^{2/(t+2)} \frac{1}{u}h(\frac{1}{u})\,\mathrm{du}$$
$$= \int_{1/2}^{1} \frac{1}{u}h(\frac{1}{u})\,du + \frac{1}{2}\int_{1/2}^{2/(t+1)} \frac{1}{u}h(\frac{1}{u})\,du + \frac{1}{2}\int_{1/2}^{2/(t+2)} \frac{1}{u}h(\frac{1}{u})\,du \quad \cdots(\#)$$

と計算される。ここで、h として特に

$$\tilde{H}(t) = \left\{\frac{1}{1+t} + \frac{1}{2+t}\right\}\frac{1}{\log 2}$$

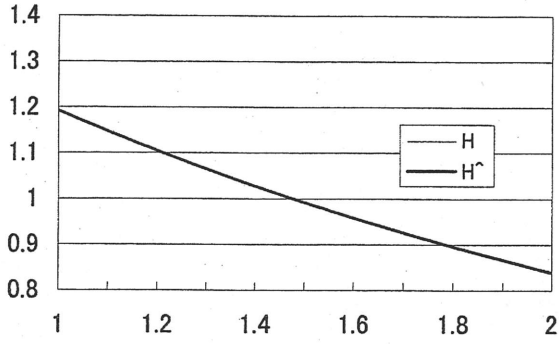
をとると、

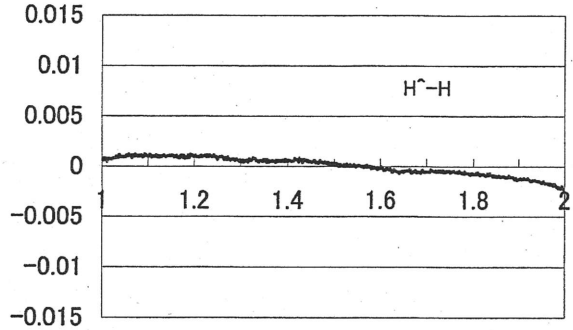
$$\hat{H}(t) \equiv \left(2 - \frac{3}{2}\frac{\log 3}{\log 2}\right) + \frac{1}{2\log 2}\Big[-\frac{3}{2}\log(1+t) - \frac{3}{2}\log(2+t) + \log(3+t)$$
$$+ \log(4+t) + \frac{1}{2}\log(5+t) + \frac{1}{2}\log(6+t)\Big]$$

を得る。

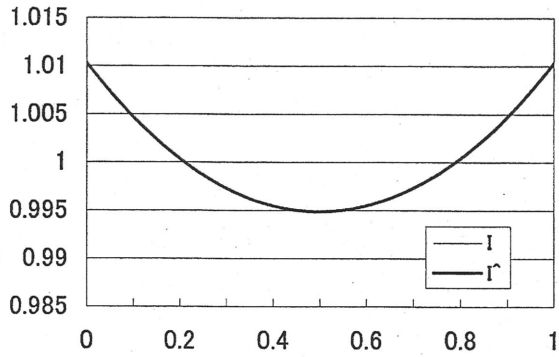
［演習］$\hat{H}(t)$ を導出せよ。

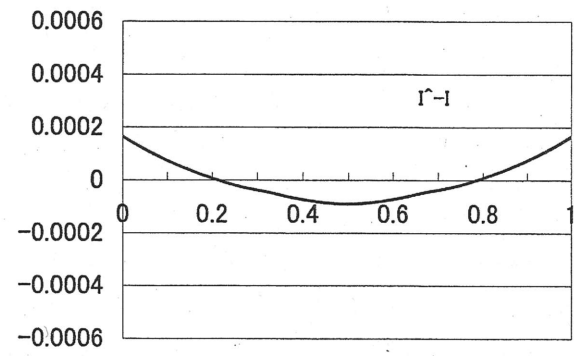
以下に、$\hat{H}(t)$ および $\tilde{H}(t)$-$H(t)$ のグラフを示しておく。期待通り $\hat{H}(t)$ ではさらに $H(t)$ に近づいていることが見てとれる。





$\hat{H}(t)$ にK改良を施した関数 $\tilde{I}(y)$, $y \in [0,1)$, を式で表すことは出来そうもないので、数値計算した結果のグラフを $\tilde{I}(y)$, $\tilde{I}(y)-I(y)$, $\tilde{I}(y)-I(y)$ とともに示しておくことにする。

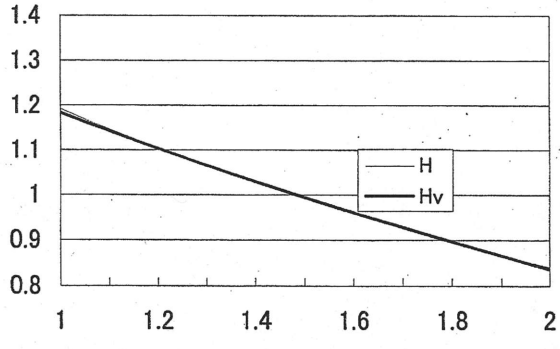


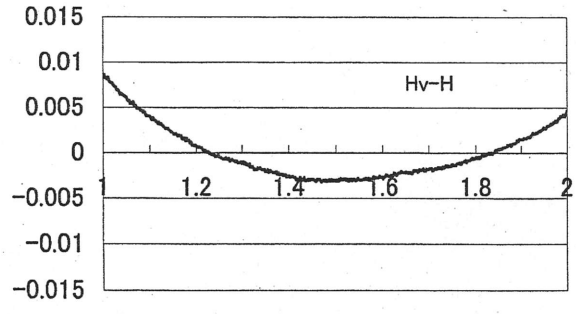


［註］上述の積分（#）において、$h \equiv 1$ とすると、

$$\tilde{H}(t) \equiv 3\log 2 - \frac{1}{2}\log(t+1) - \frac{1}{2}\log(t+2)$$

を得る。$\tilde{H}(t)$ のグラフを描いてみると、$H(t)$ とほぼ重なり、これも $H(t)$ の近似関数として十分使えそうである。





4．ＳＳＲ法乱数の生成効率の改良(SSRex)

単純実数シフト計算（ＳＳＲ計算）は、

$$g(x) = u_0 \underbrace{\cdot x \cdot x \cdot x \cdot \cdots \cdot x \cdot x}_{24}, \quad x \in [1, 2)$$

を

$$u_0 := 1 \qquad u_k := u_{k-1} \times x, \quad k=1, 2, \cdots, 24,$$

と倍精度計算し、u_k を1回計算するごとに、u_k を表す倍精度変数の

i) 仮数部の全てのビット値を１ビット左にシフトし、

ii) 指数部は　$\cdots \times 2^0$　となるように設定する、

というものであった。また、乱数値としては、上記 $g(x)$ の計算結果から、上位３桁を棄て続く４桁を取り出していた。この方法は、k 番目の乱数をその場で直接生成できるという利点があるが、またその故に乱数生成速度が遅くなるという欠点もある。

これを改良するため、今まで倍精度変数 u_k の仮数部のほぼ半分を乱数生成に利用するのみであったものを（これは、実数シフト法が、単精度計算の桁落ち誤差を摸して考案されたという経緯による）、仮数部全体の情報を利用して、

１回のＳＳＲ計算で生成する乱数の桁数を10進８桁

とすることにより、生成効率のアップを図る。具体的には、１回ごとの左シフトを２ビットにし、以下のような計算を行う。

［拡張ＳＳＲ計算(SSRex)］

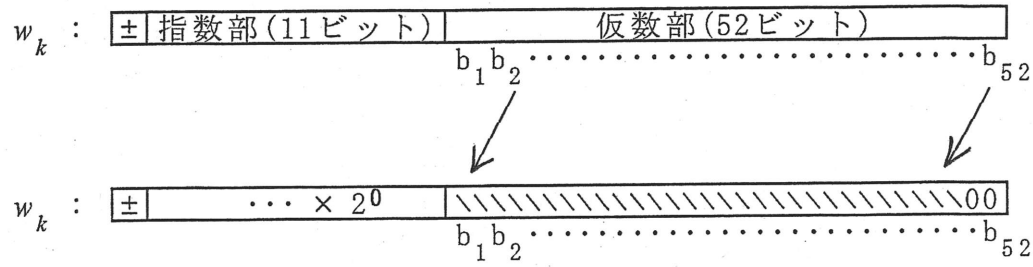
$$h(x) = w_0 \underbrace{\cdot x \cdot x \cdot x \cdot \cdots \cdot x \cdot x}_{26}, \quad x \in [1, 2)$$

を

$$w_k := w_{k-1} \times x, \quad k=1, 2, \cdots, 26,$$

と倍精度計算する。ただし、w_k を1回計算するごとに、w_k を表す倍精度変数に対し

i) 仮数部の全てのビット値を２ビット左にシフトし
（⟹右端には０を２ビット分入れる ）、



ii) 指数部は　$\cdots \times 2^0$　となるように設定する（$\Longrightarrow 1 \leq w_k < 2$ となる）。

この拡張ＳＳＲ計算（extended SSR computation, SSRex）について、ＳＳＲ計算の写像 Φ_x に対応する写像の記号として Θ_x を使うことにする。ＳＳＲex計算から実際に８桁乱数を作るには、§2 で述べたような方法と同様に、

Ｋ改良ＳＳＲex計算

$$h_e = \Theta_x^{26}(1.e) \ , \quad h_p = \Theta_x^{26}(1.\pi)$$

を計算し、

$$h_e - h_p \mod [0,1)$$

を作り、得られた値から 0.　を含めた

最初の５桁を捨て、続く８桁を乱数として採用

すればよい。ここで、

$$1.\pi = 1.3141592653589$$

である。あるいは、

Ｘ改良ＳＳＲex計算

$$\widehat{g}(x) = w_0 \cdot \underbrace{x \cdot x \cdot \cdots \cdot x}_{10} \ , \quad x \in [1,2)$$

を、初期値　$w_0 = 1.\pi$　について、Ｘ改良ＳＳＲ計算の $\widehat{g}_p$ と同様にＳＳＲ計算を行い（＝仮数部の１回でのビットシフトが８ビット）、得られた $\widetilde{g}_p$ に対して $(h_e \text{ xor } \widetilde{g}_p)$ をとる。そして、$(h_e \text{ xor } \widetilde{g}_p)$ から最初の５桁を捨て、続く８桁を乱数として採用する。

ＫおよびＸ改良ＳＳＲex計算による乱数列の特性については、次節の「ＳＳＲex法乱数列の並列列」の結果の一部（r=491377, s=47513 の場合）として述べることにする。

［註１］ＳＳＲex計算値から、最初の４桁を捨て、続く１０桁を乱数として採用しても、乱数特性は十分に保たれると考えられるが、現時点では特に検証を行っていない。

［註２］ＳＳＲex計算値から、乱数値として３２ビットを取り出すときは、b_{10} から b_{41} を使うとよい。

5．ＳＳＲex法乱数列の並列列

5．1 ＳＳＲex法乱数列の並列列の構成

ＳＳＲex法で長周期化乱数を得るには、§1.4 で述べたように、 p=49933453, q=22801201, r=491377, s=47513 （いずれも素数）として、

$$(r_k, s_k) \equiv (rk \bmod p,\ sk \bmod q)\ ,$$

から定まる $x_k = x(r_k, s_k)$ ， $k = 1, 2, 3, \cdots$, について、ＳＳＲex計算を行えばよい。この方法においては、

$$\{ x(r_k, s_k) \mid k=0, 1, \cdots, pq-1 \} \ = \ \{ x(i, j) \mid 0 \le i < p,\ 0 \le j < q \}$$

であることを考えると、$x(r_k, s_k)$, k=1, 2, $\cdots$, は、$\{ x(i, j) \mid 0 \le i < p,\ 0 \le j < q \}$ の要素を、(r_k, s_k) ， $k = 1, 2, \cdots$, にしたがって並び換えたものと考えることができる。よって、前節まで r=491377, s=47513 であった r, s について別の素数を考え、新たに $(\tilde{r}_k, \tilde{s}_k)$ ， $k = 1, 2, \cdots$, を作り直せば、周期 pq の異なる乱数系列が得られることになる。そこで、r として 491377 から連続する 199 個の素数をとり、また s として 47513 から連続する 53 個の素数をとって組み合わせれば計 10547 個の異なる乱数系列を並列的に得ることができる。このようにして得られる擬似乱数生成器は、最近のＰＣクラスター計算機のように、多数のＣＰＵを有する計算機に乱数生成器を実装するのが困難な場合などには、手軽な多系列乱数生成器として特に有効である。また、ＳＳＲ法では k 番目の乱数を直接生成できるので、並列計算が可能である。これらの観点から、ＳＳＲ乱数は

並列型スーパーコンピュータに適した乱数生成法

であるといえよう。

ＳＳＲ乱数の特性を調べるにあたっては、上述の全ての r, s の組み合わせについて特性を調べることは、現時点の計算機では時間がかかるので、r, s をほぼ等間隔に

$$r = 491377,\quad 492299,\quad 493177,\quad 404041,$$

$$s = 47513,\quad 47699,\quad 47869,\quad 48049$$

と取り出し、それらを組み合わせた16個の改良ＳＳＲex乱数系列について調べることにする。

5．2 K改良ＳＳＲex乱数の４桁乱数列としての特性

K改良ＳＳＲex乱数生成法では、$\Theta_x^{26}(1.e) - \Theta_x^{26}(1.\pi) \mod [0,1)$ を計算し、得られた値から 0. を含めた最初の５桁を捨て、続く８桁を乱数として採用するが、既存の４桁乱数の検定プログラムを利用してかなりの特性を調べることができる。具体的には、８桁乱数を上位４桁・下位４桁に分割し、それぞれについて [9] の10進４桁乱数の検定プログラムを適用する、すなわち、

（１） 各 (r,s) の組み合わせに対し、4桁乱数を20000個発生し、次の7種類計10の検定を危険率 0.05 で行う：

[検定II] 文字0～9の出現頻度のχ^2検定、
[検定III] 文字0の出現間隔のχ^2検定、
[検定IV] 乱数値の Kolmogorov-Smirnov 検定(K^+およびK^-)、
[検定V] 単純上昇連および下降連テスト、
[検定VI] 4枚の 0～9 カードによる古典ポーカーテスト、
[検定VII] 遅れ 1 および 2 の系列相関テスト、
[検定VIII] 衝突テスト。

そして仮説が棄却された回数 $c(0 \le c \le 10)$ を数える。この操作を 1000 回繰り返す。各検定が危険率 0.05 で独立に行われると仮定すれば（註：この仮定は厳密には正しくない），c の分布は二項分布 $Bin(10, 0.05)$ になるので、実際に得られた c の分布と対照してχ^2検定値（CHITEST）を求める。

結果は、以下のようになる（95%の確率で 0.05 以上が望まれる）。

(K改良SSRex)上位４桁の検定総棄却回数に関するχ^2検定値

		r			
		491377	492299	493177	404041
s	47513	0.572	0.472	0.621	0.667
	47699	0.111	0.049Δ	0.843	0.420
	47869	0.184	0.156	0.674	0.263
	48049	0.548	0.0002Δ	0.025Δ	0.066

上表 16 個のデータの {最小値, 平均値, 最大値; 標準偏差} は {0.0002, *0.354*, 0.843; 0.278} となる。

(K改良SSRex)下位４桁の検定総棄却回数に関するχ^2検定値

		r			
		491377	492299	493177	404041
s	47513	0.407	0.573	0.037Δ	0.713
	47699	0.731	0.663	0.646	0.589
	47869	0.686	0.218	0.406	0.286
	48049	0.602	0.327	0.795	0.683

上表 16 個のデータの｛最小値, 平均値, 最大値; 標準偏差｝は
{0.037, *0.523*, 0.795; 0.217} となる。

参考までに、他乱数生成法の４桁整数乱数値に関するχ^2検定値は次の通りである。

(FSR) 0.729, (MT) 0.182, (BC) 0.484,

(Ph) (上位４桁) {0.0077, *0.311*, 0.996; 0.312}

(下位４桁) {0.037, *0.295*, 0.873; 0.297}

ここで、

(FSR):フィードバックシフトレジスター法(M系列)、
(MT):メルセンヌツイスター法、
(BC):Borland C++ V5.5 の乱数、
(Ph):統計数理研究所の Altix 3700 に付置された物理乱数生成装置)

である（各乱数生成法の詳細については、[9] 参照）。

（２）次に、

[検定IX] 4桁乱数を80000個発生し、一様に分布に関する Kolmogorov-Smirnov 統計量 $K^+[K^-]$ を求める。これを10000回繰り返した $K^+(j)$ $[K^-(j)]$, j=0,･･･,9999, の分布に関してχ^2検定を危険率 0.05 で行い棄却されるかどうか、

を調べる。

結果は、各 (r, s) ごとに検定を1000回繰り返し棄却される比率を求めると、

(K改良SSRex)上位４桁の K^+/K^- の分布の棄却率

		r			
		491377	492299	493177	404041
s	47513	.345/.345	.360/.357	.342/.348	.333/.330
	47699	.340/.344	.342/.345	.361/.353	.353/.336
	47869	.348/.344	.346/.340	.374/.352	.336/.350
	48049	.350/.351	.325/.358	.340/.341	.360/.322

上表 K^+/K^- のデータの｛最小値，<u>平均値</u>，最大値；標準偏差｝は

K^+（上位４桁）｛ 0.325，　*0.347*，　0.374；　0.0123 ｝

K^-（上位４桁）｛ 0.322，　*0.345*，　0.358；　0.0096 ｝

となる。

(K改良SSRex)下位４桁の K^+/K^- の分布の棄却率

		r			
		491377	492299	493177	404041
s	47513	.341/.353	.349/.368	.346/.357	.321/.324
	47699	.360/.377	.365/.327	.345/.352	.319/.359
	47869	.349/.326	.331/.355	.366/.331	.357/.351
	48049	.365/.328	.382/.325	.354/.359	.363/.339

上表 K^+/K^- のデータの｛最小値，<u>平均値</u>，最大値；標準偏差｝は

K^+（下位４桁）｛ 0.319，　*0.351*，　0.382；　0.0170 ｝

K^-（下位４桁）｛ 0.324，　*0.346*，　0.377；　0.0171 ｝

となる。

有意水準を 0.05 に設定した割には棄却数が多いが、他の乱数生成法および物理乱数においても以下のように同様の傾向を示しており、特に悪い結果ではない：

K^+　(FSR) 0.780　(MT) 0.349　(BC) 0.332

K^-　(FSR) 0.593　(MT) 0.344　(BC) 0.344

K^+　(Ph)（上位４桁）{0.273, *0.299*, 0.318; 0.0156}
（下位４桁）{0.247, *0.288*, 0.313; 0.0166}

K^-　(Ph)（上位４桁）{0.256, *0.284*, 0.31 ; 0.0154}
（下位４桁）{0.257, *0.284*, 0.314; 0.0155}

また、1回の[検定IX]に対して

$$\tilde{D} = \left[\#\{ j \mid K^+(j) < K^-(j) \} - \#\{ j \mid K^+(j) > K^-(j) \} \right]$$

とおくと、分布の非対称性に関する情報が得られる。1000 個の $\tilde{D}$ の平均値について (r, s) を変化させた時の値は

(K改良SSRex)上位４桁に関する $\tilde{D}$ の平均

		r			
		491377	492299	493177	404041
s	47513	4.68	8.61	-3.75	-3.34
	47699	5.96	-2.97	0.76	-1.75
	47869	1.31	1.00	-1.23	-0.20
	48049	-5.40	-1.94	-1.19	0.86

となる。{最小値, 平均値, 最大値; 標準偏差} は

{-5.40, *0.0881*, 8.61; 2.23}

である。

(K改良SSRex)下位４桁に関する $\tilde{D}$ の平均

		r			
		491377	492299	493177	404041
s	47513	0.93	-1.10	-0.55	0.54
	47699	3.24	4.81	-0.09	-5.18
	47869	3.71	-1.15	-1.03	1.44
	48049	2.29	-0.72	1.37	-0.58

となる。｛最小値，平均値，最大値；標準偏差｝は

｛-5.18，*0.4956*，4.81；2.38｝

である。ＳＳＲ値の分布からも予想されることではあるが、かなり良い対称性をもっている。以下に他乱数生成法についての結果も記しておく。

(FSR) 130.879　　(MT) 1.789　　(BC) -6.4

(Ph)（上位４桁）｛ 0.57，*6.3056*，13.43；3.61 ｝

（下位４桁）｛ -6.4 ，*-1.3*，3.65；3.08 ｝

５．３　Ｋ改良ＳＳＲex乱数の８桁乱数列としての特性

前節では、Ｋ改良ＳＳＲex計算が生成する８桁乱数の特定部分を４桁取り出し、その特性を調べた。本節では８桁全体を扱ってその特性を調べる。調べる項目は、

(1) 乱数値を構成する８つの数字に対する古典ポーカーテスト、
(2) 乱数値を構成する８つの数字に対する単純上昇連および下降連テスト、
(3) ８桁の数としての衝突テスト

である。

８つの数字に対する古典ポーカーテスト

８桁乱数を構成する８つの数字について、同じ数字のものが何枚あるかの組み合わせを調べる。 00000000 ～ 99999999 の数に対して、組み合わせパターンとその理論発生回数は以下のとおりである。ここで、組み合わせパターンが 221111 とあるのは、同じ数字のものが、２枚、２枚、１枚、１枚、１枚、１枚、いう意味である（組合せ枚数の順序は問わない）。

パターン ⇒	221111	32111	2111111	22211	311111
回数 ⇒	31752000	16934400	16934400	12700800	8467200

3221	4211	41111	11111111	3311	2222
4233600	2116800	2116800	1814400	1411200	529200

5111	431	332	422	521	611	53
282240	201600	201600	151200	120960	20160	5040

44	62	71	8
3150	2520	720	10

ただし、発生パターンについて検定を行う際には、上表中のパターン 2222 を含む以降のパターンは、１まとめにして扱うことにする。

［演習］８桁の数字の組み合わせパターンの出現確率を導出せよ。

検定は、20000 個の８桁乱数を発生させ、上表から計算される発生パターンの期待回数との間で χ^2 統計量を求める。これを 1000 回繰り返す。得られた 1000 個の χ^2 統計量 について、まず CHITEST を危険率 0.05 で行い、棄却される回数を調べる（⇒ 50 前後の値がでればよい）。次に、同じ 1000 個の χ^2 統計量について、その値を以下の 14 のクラスに分け、

クラス ⇒	[0,4)	[4,5)	[5,6)	[6,7)	[7,8)	[8,9)	[9,10)
発生確率(%) ⇒	5.265	5.617	7.591	8.982	9.661	9.673	9.161

[10,11)	[11,12)	[12,13)	[13,14)	[14,15)	[15,16)	[16,∞)
8.298	7.246	6.138	5.068	4.093	3.243	9.963

各クラスの発生回数と期待回数を比べて χ^2 検定 CHITEST を行う（⇒ 95% の確率で 0.05 以上の値が望まれる）。これらを 16 個の各 (r, s) の組み合わせについて行う。結果は以下のとおりである。

(K改良SSRex) 8桁乱数に対する古典ポーカーテスト（棄却数／CHITEST値）

		r			
		491377	492299	493177	404041
s	47513	53／0.5439	52／0.6106	47／0.3774	46／0.7612
	47699	51／0.8690	62／0.2782	56／0.1504	49／0.2069
	47869	38／0.9864	44／0.3649	37／0.3008	42／0.4229
	48049	49／0.5558	38／0.5536	46／0.5716	40／0.3897

棄却回数，χ^2検定値 の平均は、それぞれ 46.875 ， 0.4965 である。

8つの数字に対する単純上昇連および下降連テスト

上昇連、下降連について考え方は同じなので、単純上昇連テストについて説明する。8桁の数 $d_1d_2d_3d_4d_5d_6d_7d_8$ を左から右に1桁ずつ見ていく。ある桁の数字 d_i の右側の数字 d_{i+1} が d_i に等しいか小さいとき、上昇連が終了したとし、調べ始めた数字から d_{i+1} までの数字の数を連の長さとする（⇒ 連の長さの最小は2となる）。次の連は d_{i+2} から調べ始める。連が終了する前に d_8 に達してしまった場合は、連として扱わないことにする（⇒ 長さを持つ連がない8桁数もある）。 00000000 ～ 99999999 の数に対して、連の長さのパターン、および、その理論発生回数は以下のとおりである。ここで、パターン 322 とあるのは、長さ 3, 2, 2 の連があったということである（連の発生順序は問わない）。また、長さを持つ連が1つもない場合は、パターン 0 とする。

パターン ⇒	322	332	2222	422	222
回数 ⇒	29947500	17968500	9150625	8984250	7486875

43	33	42	32	52	53
6534000	4900500	4900500	4356000	2032800	1219680

44	22	62	5	4	6	3
980100	635250	254100	221760	207900	103950	83160

7	2	8	0
19800	11550	1155	45

ただし、発生パターンについて検定を行う際は、上表中のパターン 52 を含む以降のパターンは、１まとめにして扱う。

［演習］８桁の数字に対する連の長さの各パターンについて発生確率を導出せよ。

検定は、20000 個の８桁乱数を発生させ、上表から計算される発生パターンの期待回数との間で χ^2 統計量を求める。これを 1000 回繰り返す。得られた 1000 個の χ^2 統計量 について、まず CHITEST を危険率 0.05 で行い、棄却される回数を調べる（⇒ 50 前後の値がでればよい）。次に、同じ 1000 個の χ^2 統計量について、その値を 14 のクラスに分け、

クラス ⇒	[0,3)	[3,4)	[4,5)	[5,6)	[6,7)	[7,8)
発生確率(%) ⇒	3.571	5.288	7.710	9.439	10.280	10.297

[8,9)	[9,10)	[10,11)	[11,12)	[12,13)	[13,14)
9.687	8.679	7.478	6.240	5.070	4.028

[14,15)	[15,∞)
3.139	9.094

各クラスの発生回数と期待回数を比べて χ^2 検定 CHITEST を行う（⇒ 95% の確率で 0.05 以上の値が望まれる）。これらを 16 個の各 (r, s) の組み合わせについて行う。結果は以下のとおりである。

(K改良SSRex) ８桁乱数に対する単純上昇連テスト（棄却数／CHITEST値）

		r			
		491377	492299	493177	404041
s	47513	52／0.0164Δ	59／0.7314	40／0.2220	66／0.0761
	47699	58／0.0288Δ	59／0.4435	57／0.3827	44／0.9094
	47869	53／0.7563	55／0.6414	64／0.8364	46／0.7276
	48049	46／0.3173	45／0.7409	46／0.5495	53／0.6310

棄却回数，χ^2検定値 の平均は、それぞれ 52.688 ， 0.5007 である。

(K改良SSRex) 8桁乱数に対する単純下降連テスト（棄却数／CHITEST値）

		r			
		491377	492299	493177	404041
s	47513	52／0.5780	45／0.6887	54／0.8657	66／0.7847
	47699	51／0.8826	54／0.8161	44／0.6709	51／0.4119
	47869	57／0.8816	50／0.6216	67／0.1892	57／0.4361
	48049	50／0.4077	60／0.0504	54／0.0551	52／0.2666

棄却回数，χ^2検定値 の平均は、それぞれ 54.000 , 0.5379 である。

8桁数としての衝突テスト

20000 個のＫ改良ＳＳＲex乱数を発生していくときに、既に発生した乱数値と同じ値を発生する回数（衝突回数）を調べる。

m = 100000000 個の壷に、 n = 20000 個の玉をランダムに投げるとき、 c 回の衝突が起こる確率は

$$\frac{m(m-1)\ldots(m-n+c+1)}{m^n}\left\{ {n \atop n-c} \right\}$$

で与えられる。ここで $\left\{ {n \atop n-c} \right\}$ (Stirling number)は、一連番号の付いている n 個の玉を（番号の順序を無視して）$n-c$ 組に分ける場合の数である。したがって衝突回数の確率分布は次のようになる。

衝突回数(≤) ⇒	0	1	2	3	4	5
確率分布 ⇒	0.1353	0.4060	0.6767	0.8572	0.9474	0.9835

［演習］上記の確率分布を導出せよ。

これから、衝突回数が 4 以下であれば、乱数が独立かつ一様であるという仮説は棄却されない。よって、この検定を (r, s) の各組み合わせごとに 1000 回行い、棄却される回数を調べ、さらに衝突回数の分布について、上記分布に対応する期待回数との間で χ^2 検定 CHITEST を行うことにする。結果は以下のとおりである。

(K改良SSRex) 8桁乱数に対する衝突テスト（棄却数／CHITEST値）

		r			
		491377	492299	493177	404041
s	47513	49／0.5771	47／0.4677	35／0.1773	42／0.2612
	47699	52／0.5723	49／0.3401	61／0.7604	45／0.5019
	47869	60／0.5722	54／0.7831	54／0.7379	60／0.5538
	48049	59／0.5232	68／0.0665	50／0.2697	57／0.5509

棄却回数，χ^2検定値 の平均は、それぞれ　52.625 ，0.4822 である。

5.4 X改良SSRex乱数列の特性

X改良SSRex計算については、§2で既に述べたので、ここでは検定結果のみを記すことにする。検定方法は、K改良SSRex乱数列の場合と全く同様である。

(X改良SSRex)上位4桁の検定棄却回数に関するχ^2検定値

		r			
		491377	492299	493177	404041
s	47513	.285	.046Δ	.639	.114
	47699	.876	.060	.200	.319
	47869	.295	.123	.173	.995
	48049	.051	.0007Δ	.255	.401

上表16個のデータの｛最小値，平均値，最大値；標準偏差｝は
{0.0007，　*0.302*，　0.995；　0.286} となる。

(X改良SSRex)下位４桁の検定棄却回数に関するχ^2検定値

		r			
		491377	492299	493177	404041
s	47513	.006Δ	.973	.760	.596
	47699	.888	.212	.775	.172
	47869	.203	.002Δ	.802	.679
	48049	.685	.768	.017Δ	.044Δ

上表16個のデータの｛最小値，<u>平均値</u>，最大値；標準偏差｝は
{0.002，　*0.474*，　0.973；　0.350} となる。

(X改良SSRex)上位４桁の K^+/K^- の分布の棄却率

		r			
		491377	492299	493177	404041
s	47513	.337/.343	.319/.349	.356/.350	.320/346
	47699	.316/.354	.352/.352	.339/.347	.340/.346
	47869	.349/.325	.339/.336	.337/.310	.365/.347
	48049	.327/.343	.361/.370	.352/.357	.346/.330

上表　K^+/K^- のデータの｛最小値，<u>平均値</u>，最大値；標準偏差｝は

K^+（上位４桁）｛ 0.316，　*0.341*，　0.365；　0.0149 ｝

K^-（上位４桁）｛ 0.310，　*0.344*，　0.370；　0.0138 ｝

となる。

(X改良SSRex)下位４桁の K^+/K^- の分布の棄却率

		r			
		491377	492299	493177	404041
s	47513	.337/.361	.328/.315	.351/.351	.324/.374
	47699	.345/.343	.353/.346	.320/.356	.335/.333
	47869	.337/.333	.348/.336	.351/.356	.356/.353
	48049	.356/.342	.357/.331	.355/.359	.309/.338

上表 K^+/K^- のデータの｛最小値，<u>平均値</u>，　最大値；標準偏差｝は

K^+（下位４桁）｛ 0.309，　*0.341*，　0.357；　0.0148｝

K^-（下位４桁）｛ 0.315，　*0.345*，　0.374；　0.0145｝

となる。

(X改良SSRex)上位４桁に関する $\tilde{D}$ の平均

		r			
		491377	492299	493177	404041
s	47513	7.94	0.59	-1.78	-3.09
	47699	-0.48	-2.85	-2.38	-1.36
	47869	-1.90	-4.20	-0.38	2.20
	48049	0.08	-1.68	3.11	-0.40

｛最小値，<u>平均値</u>，最大値；標準偏差｝は

｛ -4.2，　*-0.4113*，　7.94；　2.92 ｝

である。

(X改良SSRex)下位４桁に関する $\tilde{D}$ の平均

		r			
		491377	492299	493177	404041
s	47513	-3.58	-0.45	1.00	-2.42
	47699	0.60	2.38	-6.42	1.71
	47869	0.66	-1.22	-2.08	-0.31
	48049	-2.18	4.61	2.61	-1.54

{最小値, 平均値, 最大値; 標準偏差} は

{ -6.42, *-0.4144*, 4.61; 2.69 }

である。

(X改良SSRex)８桁乱数に対する古典ポーカーテスト（棄却数／CHITEST値）

		r			
		491377	492299	493177	404041
s	47513	48/0.7905	43/0.9114	45/0.6048	51/0.5167
	47699	51/0.7865	64/0.4598	49/0.1217	47/0.1283
	47869	47/0.2550	54/0.8958	54/0.9326	41/0.0594
	48049	39/0.9526	58/0.2732	37/0.7549	49/0.3493

棄却回数, χ^2検定値 の平均は、それぞれ 48.563 , 0.5495 である。

(X改良SSRex) 8桁乱数に対する単純上昇連テスト（棄却数／CHITEST値）

		r			
		491377	492299	493177	404041
s	47513	55/0.8500	54/0.6376	53/0.5328	42/0.3053
	47699	55/0.2767	48/0.4826	57/0.3578	41/0.4004
	47869	47/0.4418	48/0.2256	56/0.3260	69/0.3840
	48049	56/0.9105	50/0.8908	65/0.3438	49/0.1798

棄却回数，χ^2検定値 の平均は、それぞれ 52.813 ， 0.4716 である。

(X改良SSRex) 8桁乱数に対する単純下降連テスト（棄却数／CHITEST値）

		r			
		491377	492299	493177	404041
s	47513	61/0.8933	50/0.5269	53/0.1920	50/0.3068
	47699	53/0.2535	37/0.8600	41/0.6031	47/0.3692
	47869	48/0.6733	49/0.9377	45/0.5092	42/0.2264
	48049	57/0.6136	51/0.5984	47/0.6430	51/0.2676

棄却回数，χ^2検定値 の平均は、それぞれ 48.875 ， 0.5296 である。

(X改良SSRex) 8桁乱数に対する衝突テスト（棄却数／CHITEST値）

		r			
		491377	492299	493177	404041
s	47513	42/0.1463	42/0.5503	55/0.2378	44/0.7287
	47699	49/0.5671	47/0.6986	51/0.3801	49/0.9943
	47869	63/0.3060	61/0.1751	62/0.4731	55/0.6567
	48049	58/0.1857	55/0.2568	51/0.6086	46/0.7785

棄却回数，χ^2検定値 の平均は、それぞれ 51.875 ， 0.4840 である。

６．ＳＳRex乱数列に対するＮＩＳＴの検定

米国の National Insitute of Standards and Technology (NIST) より提供されている乱数検定プログラムは、広く知られており、各種の統計的検定から構成されている。その内容については、ホームページ http://csrc.nist.gov/rng/ に詳細があるので、ここでは深くは立ち入らない。ＮＩＳＴの検定プログラム Version 1.8 は、

frequency,	block-frequency,
cumulative-sums(2),	runs,
longest-run,	rank,
fft,	nonperiodic-templates(148),
overlapping-templates,	universal,
approximate entropy,	random-excursions(8),
random-excursions-variant(18),	serial(2),
linear-complexity.	

の各テストで構成されている（テスト名のあとの括弧はテスト数、ただしテスト数１は省略）。

SSRexの検定は、各 (r, s) ごとに、1000K×1000 ビットからなるファイルを作り（ただし、１K=1024 とした）、各ファイルに対して、バージョン Sts-1.8 にあるすべてのテストを適用した。各テストは 1000 回実行され、得られた 1000 個の p 値の一様分布性に関する χ^2 検定値、および、p 値が 0.01 以上であるものの割合が、 finalAnalysisReport として出されるので、それらを編集して記した。

（註：Nonper, RndExc, RndExcV の各検定については、データ数が多量のため省略した。）検定時のパラメータは次の通りである：

block-frequency	block length = 20000
nonperiodic-templates	template length = 9
overlapping-templates	template length = 9
universal	block length = 7
	number of initialization = 1280
approximate entropy	block length = 10
serial	block length = 16
linear-complexity	block length = 500
number of bit streams	1000
length of bit	1000000

６．１　Ｋ改良ＳＳＲex乱数列に対するＮＩＳＴの検定

K改良SSRexに、ＮＩＳＴの検定を実行した結果は以下の通りである。

SSRexK に対するＮＩＳＴ乱数検定プログラムの結果(p値分布のχ^2検定値)

rs	Freq	BFreq	Cusum	Cusum	Runs	LRuns	Rank	DFFT
00	.434	.641	.095	.236	.691	.679	.462	.392
01	.306	.752	.970	.811	.225	.664	.618	.294
02	.924	.906	.954	.098	.230	.130	.473	.494
03	.571	.877	.891	.623	.079	.155	.789	.287
10	.434	.641	.095	.236	.691	.679	.462	.392
11	.140	.161	.498	.244	.817	.078	.247	.864
12	.689	.724	.126	.290	.546	.901	.398	.040Δ
13	.450	.155	.981	.359	.338	.616	.065	.943
20	.184	.623	.251	.209	.802	.097	.807	.434
21	.583	.804	.336	.587	.023Δ	.917	.822	.026Δ
22	.577	.430	.824	.256	.408	.297	.822	.197
23	.829	.625	.919	.585	.367	.230	.222	.124
30	.718	.198	.911	.953	.764	.164	.750	.907
31	.100	.257	.341	.500	.379	.370	.004Δ	.883
32	.836	.370	.025Δ	.426	.201	.131	.232	.941
33	.133	.192	.929	.985	.154	.228	.108	.777
Ave.	*.494*	*.522*	*.572*	*.462*	*.420*	*.396*	*.455*	*.500*
RejProp Ave.	*.990*	*.990*	*.990*	*.990*	*.990*	*.989*	*.990*	*.989*

rs	Ovlap	Univ	Apen	Ser	Ser	LinComp
00	.806	.214	.349	.948	.070	.291
01	.620	.933	.064	.675	.254	.732
02	.312	.914	.924	.534	.182	.976
03	.085	.244	.274	.850	.315	.826
10	.806	.214	.349	.948	.070	.291
11	.979	.396	.827	.848	.406	.015Δ
12	.090	.231	.206	.027Δ	.779	.252
13	.010Δ	.132	.610	.423	.467	.724
20	.136	.598	.623	.494	.871	.327
21	.949	.516	.145	.075	.379	.331
22	.006Δ	.338	.110	.633	.687	.635

23	.203	.836	.783	.417	.919	.054
30	.794	.285	.718	.585	.043Δ	.728
31	.272	.941	.485	.441	.238	.581
32	.261	.898	.252	.891	.598	.226
33	.113	.530	.504	.762	.009Δ	.924
Ave.	*.403*	*.514*	*.451*	*.597*	*.393*	*.495*
$\underset{Ave.}{RejProp}$	*.988*	*.987*	*.989*	*.989*	*.989*	*.990*

［註１］表中の　$r = 0, 1, 2, 3$,　$s = 0, 1, 2, 3$　はそれぞれ、

$r = 491377,\quad 492299,\quad 493177,\quad 404041$

$s = 47513,\quad 47699,\quad 47869,\quad 48049$

表す。

［註２］*Ave.*　は、１６個の χ^2 検定値の平均値、

$\underset{Ave.}{RejProp}$ は、１６個の　［p 値が0.01 以上の割合］の平均値。

表中、検定値が、0.05 以下の所には Δ を付しておいた。 Δ は、 0.05 の確率で発生すると考えられるから、計１１個の Δ は、総データ数 $14 \times 16 = 224$ に 0.05 を掛けた値 11.2 に近く、妥当なものである。以上の結果から、SSRexK は、十分な乱数性を有する乱数列を生成することが分かる。

６．２　Ｘ改良ＳＳＲex乱数列に対するＮＩＳＴの検定

Ｘ改良SSRexに、ＮＩＳＴの検定を実行した結果は以下の通りである。

SSRexX に対するＮＩＳＴ乱数検定プログラムの結果（p値分布のχ^2検定値）

rs	Freq	BFreq	Cusum	Cusum	Runs	LRuns	Rank	DFFT
00	.372	.441	.500	.720	.471	.693	.369	.689
01	.941	.462	.705	.036Δ	.869	.583	.076	.432
02	.005Δ	.974	.477	.087	.845	.811	.288	.266
03	.270	.818	.154	.335	.633	.082	.681	.013Δ
10	.319	.777	.843	.269	.506	.135	.339	.635
11	.593	.325	.437	.955	.758	.994	.147	.111
12	.668	.909	.162	.979	.589	.804	.766	.226
13	.023Δ	.236	.084	.040Δ	.542	.252	.055	.691
20	.991	.641	.261	.945	.944	.424	.256	.217
21	.038Δ	.186	.313	.277	.858	.006Δ	.689	.349

22	.573	.976	.768	.266	.175	.530	.290	.369
23	.512	.426	.777	.448	.643	.242	.856	.490
30	.750	.133	.937	.789	.542	.164	.260	.479
31	.758	.120	.184	.581	.986	.336	.981	.750
32	.520	.977	.726	.274	.168	.168	.736	.412
33	.280	.426	.879	.911	.722	.036Δ	.032Δ	.807
ave.	*.476*	*.552*	*.513*	*.495*	*.641*	*.391*	*.426*	*.434*
RejProp Ave.	*.991*	*.989*	*.990*	*.990*	*.990*	*.990*	*.990*	*.989*

rs	**Ovlap**	**Univ**	**Apen**	**Ser**	**Ser**	**LinComp**
00	.534	.893	.851	.184	.766	.266
01	.756	.065	.239	.454	.019Δ	.016Δ
02	.193	.084	.207	.882	.007Δ	.754
03	.342	.241	.084	.124	.785	.858
10	.822	.408	.387	.272	.283	.101
11	.536	.149	.101	.561	.522	.191
12	.135	.932	.997	.398	.100	.502
13	.221	.885	.915	.589	.997	.274
20	.177	.186	.648	.970	.841	.462
21	.322	.073	.487	.650	.687	.288
22	.677	.175	.082	.133	.522	.309
23	.892	.068	.552	.124	.526	.071
30	.315	.948	.265	.526	.102	.879
31	.198	.073	.538	.966	.083	.877
32	.860	.372	.322	.604	.338	.419
33	.883	.066	.083	.639	.868	.190
Ave.	*.491*	*.351*	*.422*	*.505*	*.465*	*.404*
RejProp Ave.	*.989*	*.988*	*.989*	*.989*	*.991*	*.989*

「Ｘ改良」では、Δ印のついた箇所は計１２カ所（比率 0.0536）であり、これも妥当な線である。「Ｘ改良」も「Ｋ改良」と同様、十分な乱数性を有する乱数列を生成している。

ちなみに、ＮＩＳＴの検定を行うためのプログラムで 1000M ビットからなるファイルを作製するのに要する時間は、 SSRexK で 96 秒、SSRexX で 78 秒であった(Celeron)。SSRexX では乱数生成速度が２割弱程速くなっていることが分かる。

［資料］　「Ｙ改良」がＮＩＳＴの検定で棄却される事例

【Ya改良SSRex計算】$0<\alpha<0.5$ とし、$h(x)$ の SSRex計算値 $\Phi_x^{26}(1)$ を $\tilde{h}$ とする。

このとき、

(i)　「$\tilde{h} < 1+\alpha$　または　$\tilde{h} \geq 2-\alpha$」 であって $b_{19}=b_{25}$ のとき、

もしくは

(ii)　「$\tilde{h} \geq 1+\alpha$　かつ　$\tilde{h} < 2-\alpha$」 であって $b_{19}\neq b_{25}$ のとき、

$\tilde{h}$ の仮数部の全ビット値を反転する。

【Yb改良SSRex計算】

「Ya改良SSRex計算」における、

(i) の　... $b_{19}=b_{25}$ ...　と　　(ii) の　... $b_{19}\neq b_{25}$ ...

を入れ替えて計算する。

そして、これらを１つにまとめて、

【Ｙ改良SSRex計算】 $b_{10}=0$ なら「SSRex計算改良Ya」を適用し、$b_6=1$ なら「 SSRex計算改良Yb」を適用する。

SSRexY に対するＮＩＳＴ乱数検定プログラムの結果(p値分布のχ^2検定値)

#	Freq	BFreq	Cusum	Cusum	Runs	LRuns	Rank
1	.679	.787	.434	.746	.956	.423	.216
2	.448	.951	.775	.504	.591	.394	.552
3	.734	.208	.538	.469	.298	.081	.504
4	.362	.469	.045Δ	.342	.483	.471	.573

#	DFFT	Ovlap	Univ	Apen	Ser	Ser	LinComp
1	.108	.200	.192	.000Δ	.056	.656	.618
2	.793	.349	.880	.000Δ	.082	.691	.232
3	.405	.012Δ	.032Δ	.000Δ	.072	.620	.770
4	.180	.864	.313	.000Δ	.498	.864	.681

「Ｙ改良」では、approximate entropy 検定で棄却されることが分かる。

6.3 Mersenne Twister法乱数列に対するＮＩＳＴの検定

以下は、Mersenne Twister法に対する、ＮＩＳＴの検定結果である。

MT に対するＮＩＳＴ乱数検定プログラムの結果(p値分布のχ^2検定値)

#	Freq	BFreq	Cusum	Cusum	Runs	LRuns	Rank	DFFT
1	.220	.025Δ	.069	.067	.011Δ	.107	.565	.724
2	.489	.974	.059	.426	.554	.689	.258	.324
3	.701	.508	.957	.068	.359	.238	.794	.820
4	.238	.697	.303	.826	.789	.796	.939	.199
5	.984	.926	.441	.281	.589	.534	.952	.114
6	.349	.815	.853	.251	.209	.604	.585	.280
7	.635	.362	.746	.777	.225	.165	.824	.506
8	.242	.565	.126	.252	.339	.929	.972	.301
9	.452	.414	.720	.342	.722	.508	.234	.544
10	.791	.084	.891	.233	.269	.752	.616	.720
11	.516	.699	.777	.988	.559	.100	.931	.514
12	.295	.608	.098	.579	.452	.218	.957	.042Δ
13	.600	.244	.639	.158	.726	.534	.736	.214
14	.722	.106	.245	.238	.771	.490	.479	.364
15	.168	.879	.270	.004Δ	.209	.952	.936	.846
16	.210	.796	.518	.174	.785	.364	.764	.052
ave.	*.476*	*.544*	*.482*	*.354*	*.473*	*.499*	*.721*	*.410*
RejProp Ave.	*.990*	*.989*	*.990*	*.990*	*.989*	*.989*	*.990*	*.988*

#	Ovlap	Univ	Apen	Ser	Ser	LinComp
1	.248	.650	.928	.750	.593	.977
2	.365	.909	.518	.896	.074	.941
3	.159	.155	.414	.301	.906	.321
4	.039Δ	.234	.738	.336	.203	.165
5	.038Δ	.064	.278	.785	.313	.055
6	.475	.734	.734	.210	.406	.321
7	.789	.007Δ	.962	.913	.359	.781
8	.298	.811	.119	.139	.958	.405
9	.840	.891	.822	.330	.156	.953
10	.321	.228	.845	.534	.645	.882
11	.111	.811	.556	.831	.720	.766

12	.752	.520	.414	.502	.394	.415
13	.307	.631	.639	.169	.005Δ	.625
14	.082	.111	.357	.066	.074	.391
15	.899	.300	.197	.937	.742	.689
16	.469	.635	.419	.184	.734	.130
Ave.	*.387*	*.481*	*.559*	*.493*	*.455*	*.551*
RejProp Ave.	*.989*	*.988*	*.989*	*.989*	*.989*	*.991*

Δ の箇所は 8 箇所（比率 0.0357）である。

6.4 フィードバックシフトレジスター法乱数列に対するＮＩＳＴの検定

以下は、フィードバックシフトレジスター法(M系列)に対する、ＮＩＳＴの検定結果である。

FSR に対するＮＩＳＴ乱数検定プログラムの結果(*p*値分布のχ^2検定値)

#	Freq	BFreq	Cusum	Cusum	Runs	LRuns	Rank	DFFT
1	.423	.346	.344	.349	.483	.899	.637	.779
2	.693	.666	.608	.848	.833	.689	.285	.034Δ
3	.695	.614	.925	.022Δ	.760	.746	.937	.034Δ
4	.372	.789	.206	.428	.877	.915	.303	.372
5	.273	.276	.683	.028Δ	.672	.602	.338	.689
6	.504	.771	.911	.447	.375	.028Δ	.139	.032Δ
7	.714	.064	.610	.699	.866	.593	.766	.708
8	.914	.158	.426	.943	.153	.920	.475	.548
9	.598	.399	.748	.907	.608	.430	.165	.172
10	.167	.596	.506	.133	.313	.369	.298	.074
11	.623	.434	.794	.251	.610	.284	.227	.249
12	.616	.220	.280	.349	.538	.125	.405	.871
13	.469	.614	.374	.471	.054	.193	.516	.829
14	.303	.600	.389	.212	.328	.679	.933	.176
15	.681	.027Δ	.017Δ	.473	.473	.992	.524	.384
16	.059	.053	.968	.548	.136	.195	.435	.012Δ
ave.	*.506*	*.414*	*.549*	*.444*	*.505*	*.541*	*.461*	*.373*
RejProp Ave.	*.991*	*.987*	*.991*	*.991*	*.990*	*.990*	*.987*	*.988*

#	Ovlap	Univ	Apen	Ser	Ser	LinComp
1	.171	.473	.430	.492	.940	.675
2	.044Δ	.097	.770	.746	.401	.783
3	.270	.336	.738	.987	.608	.581
4	.415	.129	.487	.764	.408	.434
5	.019Δ	.001Δ	.668	.364	.048Δ	.098
6	.144	.627	.960	.354	.764	.009Δ
7	.225	.593	.705	.430	.024Δ	.236
8	.744	.179	.748	.953	.398	.542
9	.354	.006Δ	.277	.925	.428	.151
10	.414	.920	.957	.608	.593	.133
11	.035Δ	.872	.435	.561	.787	.915
12	.437	.905	.077	.695	.489	.641
13	.251	.166	.419	.336	.239	.602
14	.500	.006Δ	.108	.891	.145	.052
15	.382	.520	.479	.069	.591	.660
16	.483	.288	.306	.728	.532	.065
Ave.	*.305*	*.382*	*.535*	*.619*	*.462*	*.411*
RejProp Ave.	*.988*	*.990*	*.990*	*.990*	*.989*	*.989*

Δ の箇所は 18 箇所（比率 0.0803）である。

6.5 SSRexK, SSRexX, MT, FSR に対するＮＩＳＴの検定結果のまとめ

前節までのＮＩＳＴの検定結果一覧は、詳細なものではあるが、１つのテスト名でパラメータを多く変化させる検定については、結果の記述を割愛した。この節では、すべてのテスト結果が生かされるように、finalAnalysisReport のデータの取り扱いを変えて結果をまとめておく。すなわち、

> 各テスト項目（188ある）ごとに出された１６個の「1000 個の p 値の一様分布性に関する χ^2 検定値」を、平均し、その最大値、最小値を求める、

ことにする。また、

> 各テスト項目ごとに出された１６個の「1000 個の p 値のうち、値が 0.01 以上であるものの割合」についても同様

のことを行う。結果は以下の通りである。（[i=*nn*] は該当テストの通し番号）

【SSRexK】

[(*p*値検定)平均 max] = 0.666470 at nonperiodic-templates [i=89]
[(*p*値検定)平均 min] = 0.286432 at nonperiodic-templates [i=81]
[(*p*≥0.01割合)平均 max] = 0.991875 at nonperiodic-template [i=57]
[(*p*≥0.01割合)平均 min] = 0.987187 at universal [i=157]

【SSRexX】

[(*p*値検定)平均 max] = 0.709880 at random-excursions-variant [i=177]
[(*p*値検定)平均 min] = 0.302638 at random-excursions-variant [i=169]
[(*p*≥0.01割合)平均 max] = 0.992650 at random-excursions-variant [i=180]
[(*p*≥0.01割合)平均 min] = 0.986856 at random-excursions [i=159]

【MT】

[(*p*値検定)平均 max] = 0.721409 at rank [i=6]
[(*p*値検定)平均 min] = 0.304160 at random-excursions-variant [i=179]
[(*p*≥0.01割合)平均 max] = 0.992250 at nonperiodic-templates [i=138]
[(*p*≥0.01割合)平均 min] = 0.987500 at fft [i=7]

【FSR】

[(*p*値検定)平均 max] = 0.685129 at nonperiodic-templates [i=58]
[(*p*値検定)平均 min] = 0.292520 at nonperiodic-templates [i=34]
[(*p*≥0.01割合)平均 max] = 0.991750 at nonperiodic-templates [i=44]
[(*p*≥0.01割合)平均 min] = 0.986788 at random-excursions [i=166]

各乱数生成法とも同様の傾向を示しており、ＮＩＳＴの検定についていえば、SSRexK, SSRexX, MT, FSR の乱数特性は一蓮托生の性格をもっていると言えよう。
また、テスト数が多い

nonperiodic-templates(148), random-excursions(8),
random-excursions-variant(18)

に、max, min が集中していることは、納得できることである。

7．実数シフト法の完全整数演算化

カオス写像の繰り返しにより、非再帰的に擬似乱数を生成する実数シフト法は、本質的に浮動小数点演算を用いているため、計算機システムに依存して微妙に異なる乱数値を生成することがある。本章では、実数シフト乱数生成メカニズムの核心部分を整数の乗算とシフトによるアルゴリズムで実現し、実数シフト法の完全整数演算化を行う。

7．1 単純実数シフト法（ＳＳＲ法）と浮動小数点演算

ここで少し、ＳＳＲ法について復習しておく。非再帰的に擬似乱数を生成する「単純実数シフト法」（ Simplified Shift-Real method, ＳＳＲ法）は、以下の部分的アフィン関数 $\Phi_x : [1,2) \rightarrow [1,2)$

$1 \leq x < 1.5$ のとき

$$\Phi_x(t) = \begin{cases} 2xt - 1 & (1 \leq t < \frac{3}{2x} \text{ のとき}) \\ 2xt - 2 & (\frac{3}{2x} \leq t < \frac{2}{x} \text{ のとき}) \\ xt - 1 & (\frac{2}{x} \leq t < 2 \text{ のとき}) \end{cases}$$

$1.5 \leq x < 2$ のとき

$$\Phi_x(t) = \begin{cases} 2xt - 2 & (1 \leq t < \frac{2}{x} \text{ のとき}) \\ xt - 1 & (\frac{2}{x} \leq t < \frac{3}{x} \text{ のとき}) \\ xt - 2 & (\frac{3}{x} \leq t < 2 \text{ のとき}) \end{cases}$$

を用いて、 $u_{24} = \Phi_x^{24}(1)$ を計算し、u_{24} の最初の３桁を棄て続く４桁を乱数値として取り出すものである。また、 x を x_k , k =1, 2, 3, ･･･, と変化させて、多くの乱数値を得ている。ここで x_k は、

p=49933453, q=22801201, r=491377, s=47513,
a=1920000, b=48060000,

とおいて（p, q, r, s は素数）、

$$x_k = \begin{cases} 1.0 + \dfrac{r_k}{a+s_k} & \text{if} \quad (a+s_k) > r_k \\ 1.0 + \dfrac{r_k-(a+s_k)}{b-s_k} & \text{if} \quad (a+s_k) \leq r_k \end{cases},$$

ただし、

$$(r_k, s_k) \equiv (rk \bmod p,\ sk \bmod q),$$

と定められるが、その定め方はさほど重要ではない。重要なことは x_k が $[1,2)$ 上を一様に分布するということである。

ＳＳＲ法に現れる関数 $\Phi_x(t)$ は、コンピュータによる浮動小数点計算で、倍精度変数 u_0 := 1 から始めて、

ⅰ）u_{k-1} を x 倍する写像を φ_x と書き、

ⅱ）変数 $\varphi_x(u_{k-1})$ の指数部を $\cdots\times 2^0$ とする写像を φ_e と書き、

ⅲ）さらに $\varphi_e(\varphi_x(u_{k-1}))$ の仮数部を１ビットシフトする写像を φ_s と書いたとき、

$u_k = \varphi_s(\varphi_e(\varphi_x(u_{k-1}))) = (\varphi_s \circ \varphi_e \circ \varphi_x)(u_{k-1}) \equiv \Phi_x(u_{k-1})$ として現れる。ＳＳＲ法での φ_s の仮数部のビットシフトは１であるが、拡張ＳＳＲ法(SSRex)では、φ_s のシフトサイズを２にして８桁の乱数を発生させている。このことから、乱数を発生させるためには、ビットシフトのサイズには特にこだわる必要がないことが分かる。

一方、ＳＳＲ法での乱数生成には浮動少数演算が必須であるが、浮動少数演算は計算機の性能を左右するため、各ＭＰＵメーカーが高速化のために工夫をこらしている部分であり、メーカーごとに差違がある。数値計算の計算精度については一応 IEEE の標準があるが、浮動小数点数の計算機内部での具体的な計算方法については特に規定があるわけではない。このため、例えば２つの浮動小数点数を乗算する場合、ＭＰＵ内部で処理できるビット長には制限があるので、途中の演算結果を止むを得ず丸めて処理することになり、最終的に結果を倍精度変数の仮数部５２ビットに収める際、仮数部の下位ビットが計算機システムにより微妙に異なってしまうことがある。このことは、ＳＳＲ法では、計算機システムが異なると、微妙に異なる乱数列が生成される可能性があることを意味している。

乱数を使ったモンテカルロシミュレーション等においては、乱数列の生成法によらない統計的な性質が解析の対象になっているので、生成される乱数列が計算機システムによって微妙に違っていてもそれほど問題にならないと思われるが、例えば乱数を暗号処理の目的で使う場合には致命的な欠陥になる。いずれにしても、乱数生成法の可搬性は重要な問題であり、可搬であることに越したことはな

く、実数シフト法ではそれが可能であるので、以下に現時点までの研究成果を述べる。

7.2 単純実数シフト法の整数演算化

実数シフト法（ＳＳＲ法）の整数演算化を考える際、単に浮動小数点演算を整数演算でソフト的にエミュレートするだけでは、実行速度が極端に下がってしまい意味はない。幸い最近は６４ビットの整数演算が可能な計算機が普及してきているので、６４ビット×６４ビットの整数乗算を行ったときに自然に発生するオーバーフローのビットサイズが、実数シフト法の仮数部のシフト量に相当するように調整することで、整数演算化を実現できる。この場合、オーバーフローの桁数を敢て固定化しないようにすると、演算の高速化がはかれる。

以下に述べる単純整数シフト計算（Simplified Shift-Integer computation, ＳＳＩ計算）は、ＳＳＲ法のベースをなす実数シフト計算（ＳＳＲ計算）に対応するものである。

［ＳＳＩ計算］

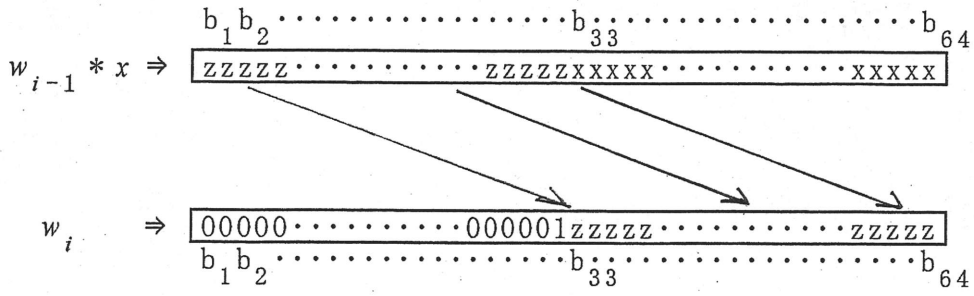
w_0, x を６４ビット整数変数とする。

$$G_{w_0}(x) = w_0 \underbrace{\circ x \circ x \circ x \circ \cdots \circ x * x}_{23}$$

を次のように計算する（$\circ$ の意味）。

(i) $w_{i-1} * x$ を通常の６４ビット整数乗算で行う（オーバーフローは無視する）、

(ii) (i) の結果を３２ビット右にシフトし、空いた $b_1 b_2 \cdots\cdots b_{32}$ を $000\cdots\cdots001$ とおき、結果を w_i とする、



(iii) (i), (ii) を $i = 1, 2, \cdots, 22$ について繰り返す、

(iv) 最後に、$w_{22} * x$ を通常の６４ビット整数乗算で行い（オーバーフローは無視）、$G_{w_0}(x)$ とする。

このＳＳＩ計算をもとに、Ｋ改良ＳＳＲ乱数生成法に相当するＫ改良ＳＳＩ乱数生成法（ＳＳＩＫ）を以下のように構成する。

［ＳＳＩＫ乱数生成法］

p = 0x7ffffffe1　　q = 0x7fffffcf　　r = 0x39f750241

s = 0x32f50fee9　　w_0 = 0x18237449a　　v_0 = 0x1dda73ad3

とする。

$$(r_k, s_k) = (rk \bmod p,\ sk \bmod q)\ ,\ k = 1, 2, 3, \cdots,$$

x = 0x88237449a ,　　y = 0xbdda73ad3

とし、

$$x_k = (\ x \text{ xor } r_k)\ ,\qquad y_k = (\ y \text{ xor } s_k)$$

とおく。そして、２つのＳＳＩ計算 $G_{w_0}(x_k)$, $G_{v_0}(y_k)$ の差

$$G_{w_0}(x_k) - G_{v_0}(y_k)$$

をとり、得られた６４ビット整数の最初の１６ビットを棄て、続く３２ビットを k 番目の乱数値とする。

もう少し詳しくＳＳＩＫ乱数生成法を見てみる。w_i , x_k は、

0x100000000 ≤ w_i ≤ 0x1ffffffff ,　0x800000000 ≤ x_k ≤ 0xffffffff

であるので、整数乗算 $w_i * x_k$ を行うと、

0x80000000000000000 ≤ $w_i * x_k$ ＜ 0x1fffffffffffffff

となるが、このうち下位６４ビットがメモリーに保存されるので、オーバーフロ

ーするビット数は４～５ビットである。 ＳＳＩ計算の(ii)の右シフトでは、メモリーに保存された６４ビットのうちの上位３２ビットが $b_{33}b_{34}\cdots\cdots b_{64}$ として残り、$b_1b_2\cdots\cdots b_{32}$ には 000……001 がセットされるので、ＳＳＲ計算の仮数部の左シフトに対応するＳＳＩ計算のビットシフト数は、３～４ビットになる。このプロセスがＳＳＩ計算の(iii)-(iv)により、２３回繰り返されることになるので、平均 $3.5\times23=80.5$ ビット相当のシフトがＳＳＩ計算でなされていると考えられる。

また、ＳＳＩＫ乱数の周期であるが、p, q, r, s はいずれも素数であるので、r_k, s_k したがって x_k, y_k の周期はそれぞれ p, q となり、結局 $G_{w_0}(x_k)$ - $G_{v_0}(y_k)$ の周期は、

$$pq = 34359738337 \times 34359738319 \approx 1.18\times10^{21}$$

となる。

このＳＳＩＫ乱数の乱数特性は、以下に述べるＳＳＩＸ乱数の特性とともに次節で述べる。

Ｘ改良ＳＳＩ乱数生成法（ＳＳＩＸ）は、Ｘ改良ＳＳＲ乱数生成法に相当するＳＳＩ乱数生成法である。

［ＳＳＩＸ乱数生成法］

p = 0x7ffffffe1　　q = 0x7fffffffc7　　r = 0x39f750241

s = 0x32f50fef7e7　　w_0 = 0x18237449a　　v_0 = 0x1dda73ad3

$$(r_k, s_k) = (rk \bmod p,\ sk \bmod q)\ ,\ k = 1, 2, 3, \cdots,$$

x = 0x88237449a ,　　y = 0xecbdda73ad3

$$x_k = (\ x \text{ xor } r_k)\ ,\qquad y_k = (\ y \text{ xor } s_k)$$

とする。

$$\widehat{G}_{v_0}(y) = v_0 \underbrace{\circ y \circ y \circ y \circ \cdots \circ y * y}_{10}$$

をＳＳＩ計算の $G_{w_0}(x)$ と同様に計算する（掛け算の回数が１０回であることに留意）。そして、６４ビット整数

$$G_{w_0}(x_k) \quad \text{xor} \quad \widehat{G}_{v_0}(y_k)$$

の最初の１６ビットを棄て、続く３２ビットを k 番目の乱数値とする。

$\widehat{G}_{v_0}(y)$ でのオーバーフローの発生状況を見ると、

$$0x100000000 \le w_i \le 0x1fffffffff, \quad 0x80000000000 \le y_k \le 0xfffffffffff$$

であるので、整数乗算 $w_i * y_k$ を行うと、

$$0x8000000000000000000 \le w_i * y_k < 0x1fffffffffffffffffff$$

となり、このうち下位６４ビットがメモリーに保存されるので、オーバーフローするビット数は 12～13 ビットである。 したがって、ＳＳＲ計算の仮数部の左シフトに対応するＳＳＩのビットシフト数は、11～12ビットになる。このプロセスが１０回繰り返されることになるので、平均 11.5×10＝115 ビット相当のシフトが $\hat{G}_{v_0}(y)$ の計算で行われていると考えられる。ＳＳＩＸ計算では $\hat{G}_{v_0}(y)$ でのシフトの繰り返し回数が $G_{w_0}(x_k)$ より少ないので、ＳＳＩＫ計算より計算速度が若干速くなる。また、乱数の周期は、

$$pq = 34359738337 \times 8796093022151 \approx 3.02 \times 10^{23}$$

で、ＳＳＩＫより約２５０倍長い。

７．３ ＮＩＳＴのプログラムによるＳＳＩ乱数の検定

米国の NIST (http://csrc.nist.gov/rng/) が提供している乱数検定プログラムを使って、ＳＳＩＫ、ＳＳＩＸの乱数特性を調べてみる。検定法は今までとまったく同じであるが、念のため要点を記しておく。

ＮＩＳＴのプログラム Version 1.8 は、以下の１５種類１８８テストから構成されている（テスト名のあとの括弧はテスト数、ただしテスト数１は省略）：

frequency,	block-frequency,
cumulative-sums(2),	runs,
longest-run,	rank,
fft,	nonperiodic-templates(148),
overlapping-templates,	universal,
approximate entropy,	random-excursions(8),
random-excursions-variant(18),	serial(2),
linear-complexity.	

検定実行時のパラメータは、block-frequency テストのブロック長を 20000 に変更した以外は既定値のままである。また、各テストに供されたデータは１ギガビットであり、各テストは、１メガビットを使う検定を１０００回繰り返し、p 値を 1000 個出力する。この p 値は一様分布をするとされるので、一様分布性に関する χ^2 検定を適用してその検定値を、また、p 値が 0.01 以上であるもの

の割合を、以下のような finalAnalysisReport として出力する：

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
108 110  78 106 104 108 109 106  77  94  0.106877   0.9890    frequency
 94  84  82 102 110 114 124 115  92  83  0.014051   0.9910    block-frequency
107  91 113  85 100 109 106 101 101  87  0.522100   0.9890    cumulative-sums
105 116 101  81 105  92  93 101 100 106  0.516113   0.9900    cumulative-sums
   ... ... ... ...
```

（表中の、P-VALUE が χ^2 検定値、PROPORTION が $p \geq 0.01$ であるものの割合。）

このＮＩＳＴのプログラムを１６回繰り返し、テスト数が多い nonperiodic-templates, random-excursions, random-excursions-variant の各テストを除いたテストの P-VALUE 値を一覧表にして示す。また、テストごとに１６個の P-VALUE の平均値も表示する。結果は以下の通りである（小数点以下６桁の数値を、小数点以下３桁で切り捨てて表示）。

SSIK に対するＮＩＳＴ乱数検定プログラムの結果（p値のχ^2検定値）

#	Freq	BFreq	Cusum	Cusum	Runs	LRuns	Rank
1	.107	.014Δ	.522	.516	.070	.410	.078
2	.377	.313	.958	.764	.925	.914	.391
3	.107	.637	.008Δ	.359	.473	.052	.456
4	.354	.853	.963	.957	.746	.991	.064
5	.831	.035Δ	.697	.327	.276	.639	.295
6	.132	.740	.155	.800	.254	.083	.254
7	.627	.473	.074	.187	.123	.108	.354
8	.217	.059	.272	.085	.853	.154	.011Δ
9	.187	.697	.841	.681	.077	.367	.248
10	.260	.838	.813	.508	.855	.987	.861
11	.987	.980	.939	.520	.829	.469	.670
12	.437	.188	.871	.777	.589	.303	.021Δ
13	.957	.880	.394	.699	.851	.836	.464
14	.959	.331	.899	.204	.471	.500	.502
15	.119	.550	.804	.483	.377	.248	.119

16	.201	.610	.986	.336	.610	.817	.528
Ave.	*.429*	*.512*	*.637*	*.513*	*.524*	*.492*	*.332*

#	DFFT	Ovlap	Univ	Apen	Ser	Ser	LinComp
1	.011Δ	.120	.283	.952	.492	.014Δ	.579
2	.078	.257	.947	.303	.481	.818	.985
3	.589	.978	.030Δ	.209	.643	.273	.341
4	.677	.036Δ	.083	.333	.891	.144	.214
5	.115	.785	.760	.437	.421	.052	.708
6	.579	.216	.641	.625	.760	.073	.581
7	.864	.057	.730	.052	.423	.398	.919
8	.206	.892	.103	.023Δ	.261	.069	.156
9	.668	.826	.278	.481	.596	.218	.577
10	.001Δ	.039Δ	.041Δ	.011Δ	.344	.764	.914
11	.864	.528	.014Δ	.445	.861	.183	.734
12	.303	.160	.687	.789	.620	.194	.245
13	.561	.133	.406	.447	.088	.372	.817
14	.185	.208	.979	.826	.625	.858	.357
15	.331	.005Δ	.720	.720	.350	.065	.687
16	.656	.972	.448	.591	.693	.637	.183
Ave.	*.418*	*.388*	*.447*	*.453*	*.534*	*.321*	*.562*

検定値が 0.05 以下の所には Δ を付してある。 Δ の出現比率は、 14/224 = 0.0625 であり、0.05 と比して特に問題となる値ではない。また、平均値 Ave にも特に問題点がなく、ＳＳＩＫは十分な乱数性を有しているものと考えられる。

SSIX に対するＮＩＳＴ乱数検定プログラムの結果(*p*値のχ^2検定値)

#	Freq	BFreq	Cusum	Cusum	Runs	LRuns	Rank
1	.936	.369	.978	.868	.375	.675	.591
2	.011Δ	.886	.452	.291	.417	.421	.877
3	.101	.058	.650	.408	.430	.664	.528
4	.126	.538	.516	.811	.290	.127	.654
5	.561	.722	.464	.015Δ	.178	.973	.718
6	.355	.585	.093	.285	.410	.298	.195
7	.094	.049Δ	.045Δ	.127	.942	.239	.113
8	.306	.447	.335	.233	.939	.880	.313
9	.705	.699	.465	.412	.136	.983	.421
10	.374	.137	.853	.606	.770	.316	.230

11	.285	.589	.224	.876	.394	.853	.010Δ
12	.043Δ	.260	.377	.325	.188	.309	.851
13	.648	.804	.858	.544	.538	.986	.123
14	.901	.038Δ	.880	.983	.591	.274	.981
15	.253	.941	.257	.561	.450	.771	.559
16	.979	.508	.573	.498	.072	.331	.687
Ave.	*.417*	*.477*	*.501*	*.490*	*.445*	*.569*	*.491*

#	DFFT	Ovlap	Univ	Apen	Ser	Ser	LinComp
1	.677	.047Δ	.414	.840	.283	.813	.044Δ
2	.522	.157	.752	.306	.670	.730	.043Δ
3	.206	.012Δ	.203	.602	.044Δ	.260	.746
4	.546	.233	.565	.885	.974	.080	.548
5	.109	.672	.035Δ	.956	.899	.710	.606
6	.001Δ	.754	.679	.233	.118	.117	.977
7	.833	.297	.315	.836	.125	.269	.226
8	.325	.462	.015Δ	.369	.899	.225	.600
9	.076	.414	.026Δ	.951	.724	.798	.891
10	.120	.689	.403	.951	.695	.811	.746
11	.024Δ	.339	.157	.049Δ	.110	.217	.866
12	.181	.212	.039Δ	.462	.403	.074	.608
13	.181	.389	.126	.650	.432	.773	.489
14	.540	.596	.548	.357	.234	.159	.335
15	.691	.155	.947	.042Δ	.559	.251	.585
16	.045Δ	.176	.710	.679	.981	.254	.936
Ave.	*.317*	*.350*	*.371*	*.573*	*.509*	*.409*	*.578*

Δ が付いている検定値の箇所の出現比率は 21/224 = 0.0938 で、ＳＳＩＫと比して若干悪いようであるが、特に問題とする値でもない。

7.4 SSIK, SSIX に対するＮＩＳＴの検定結果のまとめ

ＳＳＩＫ、ＳＳＩＸについても、

- 各テスト項目ごとに出された１６個の「1000 個の p 値の一様分布性に関する χ^2 検定値」を平均し、その最大値、最小値を求める、
- 各テスト項目ごとに出された１６個の「1000 個の p 値のうち、値が

0.01 以上であるものの割合」を平均し、その最大値、最小値を求めることをしておく。結果は以下の通りである。

【SSRIK】

[(*p*値検定)平均 max] = 0.661006 at nonperiodic-templates [i=38]
[(*p*値検定)平均 min] = 0.320636 at serial [i=186]
[(*p*≥0.01割合)平均 max] = 0.991750 at nonperiodic-templates [i=33]
[(*p*≥0.01割合)平均 min] = 0.986687 at fft [i=7]

【SSRIX】

[(*p*値検定)平均 max] = 0.667880 at nonperiodic-templates [i=61]
[(*p*値検定)平均 min] = 0.289715 at nonperiodic-templates [i=94]
[(*p*≥0.01割合)平均 max] = 0.992781 at random-excursions-variant [i=179]
[(*p*≥0.01割合)平均 min] = 0.987375 at nonperiodic-templates [i=94]

これらの結果から、平均値についても特に問題となる値はなく、ともに十分な乱数性を有しているものと考えられる。

7.5 整数演算化のまとめ

ＮＩＳＴの検定で、ＳＳＩＫ、ＳＳＩＸ乱数について特に問題となる点はないようである。一方、ＮＩＳＴの検定のための１ギガビットの乱数生成に要する時間の比は、

SSRexK : SSRexX : SSIK : SSIX = 1 : 0.81 : 0.65 : 0.56

であり、整数演算化により生成スピードがかなり速くなっていることが分かる。

乱数特性とスピードを総合すると、特性の安定性ではＳＳＩＫが若干良く、スピードではＳＳＲＸが若干良いということになる。

8．ＳＳＲアルゴリズムのハッシュ関数への応用

近年，情報セキュリティの観点から，ハッシュ関数の重要性は広く認識されるようになってきた。ハッシュ関数は，メッセージダイジェスト関数ともいわれ，任意長のバイト列あるいはビット列を入力データとして，固定長の乱数値を出力する関数とみなすことができる。乱数値には，通常，一様乱数性が要求され，また出力桁数は，１２８ビットあるいは１６０ビット以上であり，最近はさらに長くなる傾向がある。よく知られているハッシュ関数として，ＭＤ５，ＳＨＡ－１，ＲＩＰＥＭＤなどがあるが，我々は k 番目の乱数値を非再帰的に生成する新しい擬似乱数生成法『実数シフト法』のアルゴリズムを利用した，新しいハッシュ関数ＳＳＩを考案したので紹介する。このアルゴリズムは，生成するハッシュ値の長さに依存しないが，以降，統計的性質を詳しく調べた１６０ビットのハッシュ値を生成するＳＳＩ１６０について，その構築法，乱数性のＮＩＳＴおよびランダムウォークによる検定，ハッシュ値の生成速度などについて述べる。

8.1 ＳＳＩ１６０構築の概要

新しいハッシュ関数ＳＳＩは，実数シフト擬似乱数生成法（SSR法）のアルゴリズムを利用しているが，SSR法は，次の実数シフト計算（SSR計算）を基本としている(１．１ 参照)。

［単純実数シフト計算(SSR計算)］

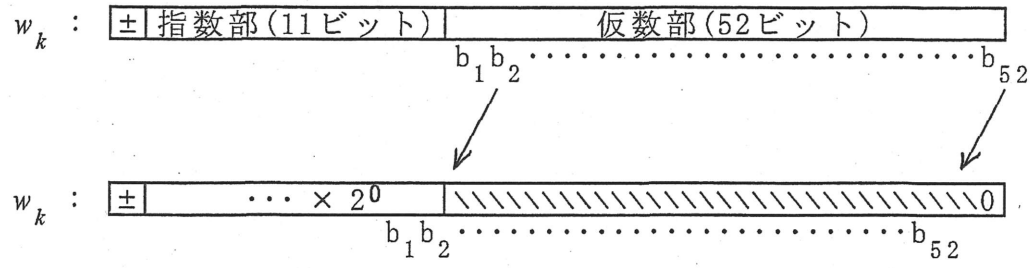
$$g(x) = w_0 \underbrace{\cdot x \cdot x \cdot x \cdot \cdots \cdot x \cdot x}_{N}, \quad x \in [1,2),$$

を

i) $w_k := w_{k-1} \times x$ と倍精度計算し，

さらに，w_k を表す倍精度変数の

ii) 仮数部の全てのビット値を１ビット左にシフトし，



iii) 指数部は　$\cdots \times 2^0$　となるように設定する。

この i) ～ iii) を，$k = 1, 2, \cdots, N$ について繰り返し行い，ＳＳＲ計算値 $g(x)$ を得る。

このＳＳＲ計算のうち，i) の左シフトでは，シフトで空いた最右端ビットには，通常０が無条件に挿入される。しかしながら，我々は，この空いたビットに何を入れても乱数特性があまり変わらないことを知っている。よって左シフトで空いた最右端ビットに，あらかじめ与えられているバイト列のビットを順次埋め込む形で，上述の i) ～ iii) を繰り返せば，バイト列の情報に依存した数値 f が得られる（入力データの圧縮）。最後に，この f を $g(x)$ の x に入れて，ＳＳＲ計算

$$g(f) = w_0 \cdot \underbrace{f \cdot f \cdot f \cdot \cdots \cdot f \cdot f}_{N}$$

を行えば，入力バイト列の情報を含んだ乱数値，すなわちハッシュ値を得ることができる（ハッシュ値の生成）。ここで，ＳＳＲ計算のアルゴリズム中で w_{k-1} から w_k を得る写像を $\Phi_x : [1, 2) \longrightarrow [1, 2)$ とすると，$g(x) = w_N = \Phi_x^N(w_0)$ であることに注意しておく。

現実的には，入力データを１ビットずつ扱うのでは能率が悪いので，ここで紹介するハッシュ関数ＳＳＩ１６０では，不動小数点演算の仮数部に相当する部分を 160 ビットにして，同時に多くのデータを埋め込めるようにしている。具体的な生成法は以下のとおりである。

８．２ 入力データの圧縮

変数の準備

まず，Φ_x の x に相当する変数 X として，160ビット（20バイト）を準備し，

$$X = X_1 \; X_2 \; \cdots\cdots \; X_{20}$$

とおく。ここで，X_i は８ビット（１バイト）である。同様に w_k の仮数部に相

当する１６０ビットの変数を F とし，

$$F = F_1\ F_2\ \cdots\cdots\ F_{20}$$

とする。また，ハッシュ値が生成されるべき入力データ（バイト列）を

$$B_1\ B_2\ \cdots\cdots\ B_N\quad (N \geq 0)$$

とする。ただし，N=0 のときは，入力データがない（空，ϕ）と解釈する。

初期値の設定

(1) F を，$1.e$ = 1.2718281828459... を２進表示した 160 ビット

a2 cb 44 11 ba 25 75 52 23 2c 11 39 a1 72 a5 c9 f8 25 27 80

で初期化する。

（**注意**．以降，SSR計算の浮動小数点演算と対応させるため，F および X の MSB は 常に 1 とする。）

(2) X の $X_1\ X_2\ \cdots\ X_6$ を，線形合同法

C_0 = 987654321 ，

C_{n+1} = 1664525 × C_n + 1013904223　　（32ビット整数）

より得られる，C_1，C_2，C_3 の各上位１６ビット（２バイト）で埋める。また，X_7，X_8 には，F_3，F_4 をコピーする。（F_3，F_4 をコピーする理由は，F に保存されている（今までの）計算結果の一部を，X に保存し，将来の計算 X*F に反映させるためである。）X_9～X_{20} には常に０を入れておく。（したがって，X の 160 ビットのうち，実質６４ビット（８バイト）が使われることになる。これは 160ビット(X)×160ビット(F) の掛け算を，64ビット×160ビットの掛け算に抑えるためである。）もちろん X_1 の MSB は１とし，さらに X_8 の LSB は１とする。（X_8 の LSB を１とする理由は，乗算 X*F の結果（実質２８バイト）から，最初の３２ビット（４バイト）を棄て，続く１６０ビット（２０バイト）を取り出す計算を繰り返したとき（後述），有効ビット数が減少していく現象を避けるためである。）

入力データの埋め込み

(3) ８バイトの入力データ（B_{1+8k}，B_{2+8k}，･･･，B_{8+8k}）を，以下のように X と F に一度に埋め込む：

(B_{1+8k} XOR 0xff)　を　X_6　に XOR する，

(B_{2+8k} XOR 0xa5)　を　F_{20}　に XOR する，

$(B_{3+8k}$ XOR 0xc3) を F_{18} に XOR する，

$(B_{4+8k}$ XOR 0x69) を F_{16} に XOR する，

B_{5+8k} を X_5 に XOR する，

$(B_{6+8k}$ XOR 0x5a) を F_{19} に XOR する，

$(B_{7+8k}$ XOR 0x3c) を F_{17} に XOR する，

$(B_{8+8k}$ XOR 0x96) を F_{15} に XOR する。

(4) $X*F$ を行い，得られた結果の３２０ビットのうち，最初の３２ビット（４バイト）を棄て，続く１６０ビット（２０バイト）を F に保存する。（⇒ $X*F$ の結果を３２ビットシフトしたことになる。また，$X*F$ は，８バイト×２０バイトの乗算なので，４バイトを棄て，続く２０バイトを採用しても，十分に有効桁数の範囲内に納まっている。）

(5) F の MSB を１にする。

(6) 上述 (3)～(5) を，入力データが無くなるまで繰り返す。ただし，X を８回使うごとに（＝入力データの 64 バイトの処理が終わるごとに），X の初期化の方法(2)と同様にして，F_3 F_4 および合同法で作成される乱数 $C_{3i+1}, C_{3i+2}, C_{3i+3}$ を使って，X を作り直す（X を固定したままでは特性が片寄るので，定期的に X の値を変える）。(3) の途中でデータが無くなったら，そこで (3) の処理を止め，ただちに (4)，(5) を実行し，入力データの埋め込み処理を終了する。（注意）$N=0 \pmod 8$ のときは，(3) の途中でデータが無くなることはない。特に，$N=0$ の時は，F は初期値のままである。

８．３ ハッシュ値の生成

上述の方法で入力データから最終的に得た160ビットの F は，入力データの終わり頃がほんの少し違うようなデータに対しては，非常に近い値となってしまう。これではハッシュ関数の意味をなさないので，最後にＳＳＲ計算 $g(f) = w_0 \cdot f \cdot f \cdot f \cdot \cdots \cdot f \cdot f$ に対応する整数計算（**ＳＳＩ計算**ということにする）を行い，乱数特性を有する１６０ビットのハッシュ値を生成する。具体的な計算は以下の通りである。

変数の初期化

(7) 160 ビット変数 F は上で使用したものを，結果を含めてそのまま使う。ただし，データ撹乱のため F_9 F_{10} F_{19} F_{20} については，XOR をそれぞれ 0x69, 0x96, 0xa5, 0x5a でとっておく。（F の MSB は 1 であることに注意する。）

(8) SSR計算中の w_k に相当する 160 ビット（20 バイト）変数 を W とし，それを 1.e で初期化する（初期化の方法は（1）と同じ）。さらに，最後の４バイト W_{17} W_{18} W_{19} W_{20} については

（入力データ長を表す 32 ビット整数） XOR 0x55aaaa55

と XOR をとっておく。

ＳＳＩ計算

(9) W*F を行い，結果の最初の８ビット（１バイト）を棄て，続く１６０ビット（２０バイト）を W に保存する。（W*F を８ビット左シフトしたことになる。）

(10) W_1 の MSB を１にする。

(11) (9),(10) を25回繰り返す。

(12) (9)で最後（25回目）に行った W*F の結果について，最初の３２ビット（４バイト）を棄て，続く１６０ビット（２０バイト）をハッシュ値とする。

以上で，ＳＳＩ１６０のアルゴリズムが確立されたので，その生成されるハッシュ値が一様乱数性を有するかどうかを，ＮＩＳＴによる検定およびランダムウォーク検定で調べる。

８．４ ＳＳＩ１６０に対するNISTの検定

ＮＩＳＴの検定は，米国の National Insitute of Standards and Technology が提供する乱数検定プログラムであり，Version 1.8 は，

frequency,
block-frequency,
cumulative-sums,
runs,
longest-run,
rank,
fft,
nonperiodic-templates,

overlapping-templates, universal,
approximate entropy, random-excursions(8),
random-excursions-variant, serial,
linear-complexity.

の１５種類のテストで構成されている（詳しくは http://csrc.nist.gov/rng/ 参照）。ＳＳＩ１６０に対する検定の方法は，整数値 N （$N=0,1,2,\cdots$）を表す６４ビット（８バイト）からなるデータを入力メッセージとして，１６０ビットのＳＳＩハッシュ値の列 $\{\gamma_N\}$ を生成し，それらがビット列としての一様乱数性を有するかどうかを調べる。また， SHA-1, RIPEMD-160 についても，同様のテストを行い，ＳＳＩ１６０の結果と比較する。

検定は，バージョン Sts-1.8 にあるすべてのテスト（全部で１８８ある）について行うこととし，入力データとして，$\{\gamma_N\}$ から作られる１ギガビットからなるファイルを使用する。検定時のパラメータは以下の通りである。

テスト名	パラメータ
block-frequency	block length = 20000
nonperiodic-templates	template length = 9
overlapping-templates	template length = 9
universal	block length = 7
	number of initialization = 1280
approximate entropy	block length = 10
serial	block length = 16
linear-complexity	block length = 500

number of bit streams	1000	← 各テストの実行回数
length of bit	1000000	← テスト１回あたりの使用ビット数

以上の設定で，ＮＩＳＴの検定を行うと，各テストは 1000 回実行され，「得られた 1000 個の p 値」 について，

- 一様分布性に関する χ^2 検定値，
- p 値が 0.01 以上であるものの割合

が， finalAnalysisReport として出力される。統計的検定は，時たま例外的な値を発生するので，ＮＩＳＴの検定を１６回繰り返して，１６個の finalAnalysisReport の平均値を見ることにする。すなわち，

（Ａ）各テストごとに， 1000 個の p 値の分布に対する χ^2 検定値が，１６個得られることになるので，その平均を求め，平均が最大

になるテスト名と，平均が最小になるテスト名を記す。

また，

（Ｂ）各テストごとに，p 値が 0.01 以上になる割合が１６データ得られるので，その平均値を求め，平均が最大になるテスト名と，平均が最小になるテスト名を記す。

さらに，

（Ｃ）１６回のＮＩＳＴの検定の繰り返しにより，$187 \times 16 = 2992$ 個の，p 値の一様分布性に関する χ^2 検定値 が得られるので，それらが 0.05 および 0.01 未満になる比率も併せて記すことにする。

なお，下表中のＳＳＩおよびＲＭＤは，それぞれ SSI160， RIPEMD-160 を表している。（表データに対応する詳細データは表の後に記述する。）

各テストごとの、１６個の［p値のχ^2検定値］の平均値

	SSI	SHA-1	RMD
最大	0.717037	0.688394	0.663230
最小	0.290542	0.298951	0.330543

各テストごとの、１６個の［p値≥0.01の割合］の平均値

	SSI	SHA-1	RMD
最大	0.991769	0.991750	0.992575
最小	0.986719	0.987875	0.987563

p 値が 0.05 未満になるテストの割合

SSI	SHA-1	RMD
0.0515	0.0575	0.0572

p 値が 0.01 未満になるテストの割合

SSI	SHA-1	RMD
0.0104	0.0080	0.0114

上表に対応する詳しいデータは以下の通りである。（テスト名のあとの［i=nnn］は，該当テストの finalAnalysisReport 中の通しテスト番号を表す。）

［ＳＳＩ１６０ハッシュ関数］

各テストごとの，１６個の［p 値分布のχ^2検定値］の平均値

AvMax= 0.717037 at nonperiodic-templates [i=130]

AvMin= 0.290542 at overlapping-templates [i=156]

各テストごとの，［p 値が 0.01 以上になる割合］の平均値

AvMax= 0.991769 at random-excursions-variant [i=179]
AvMin= 0.986719 at random-excursions [i=159]

p 値が 0.05 未満になるテストの割合　　0.0515　(154/2992)

p 値が 0.01 未満になるテストの割合　　0.0104　(31/2992)

［ＳＨＡ１ハッシュ関数］

各テストごとの，１６個の［p 値分布のχ^2検定値］の平均値
AvMax= 0.688394 at nonperiodic-templates [i=72]
AvMin= 0.298951 at fft [i=7]

各テストごとの，［p 値が 0.01 以上になる割合］の平均値
AvMax= 0.991750 at nonperiodic-templates [i=52]
AvMin= 0.987875 at overlapping-templates [i=156]

p 値が 0.05 未満になるテストの割合　　0.0575　(172/2992)

p 値が 0.01 未満になるテストの割合　　0.0080　(24/2992)

［ＲＩＰＥＭＤ－１６０ハッシュ関数］

各テストごとの，１６個の［p 値分布のχ^2検定値］の平均値
AvMax= 0.663230 at nonperiodic-templates [i=125]
AvMin= 0.330543 at nonperiodic-templates [i=89]

各テストごとの，［p 値が 0.01 以上になる割合］の平均値
AvMax= 0.992575 at random-excursions-variant [i=184]
AvMin= 0.987563 at universal [i=157]

p 値が 0.05 未満になるテストの割合　　0.0572　(171/2992)

p 値が 0.01 未満になるテストの割合　　0.0114　(34/2992)

以上において結果に多少の違いが見られるが，各ハッシュ関数について特に問題となる数値はないと思われる。

８．５ ＳＳＩ１６０に対するランダムウォーク検定

ランダムウォーク検定は，通常の乱数検定では検出が難しい長周期の乱数特性を調べるのに有効である。以下のランダムウォーク検定で使用する確率変数は次の通りである。 X_k, k=1,2,...,を値 +1 または -1 を確率 1/2 で発生する独立な確率変数列とし，$S_n = \sum_{k=1}^{n} X_k$ としたとき，

first passage time: $FP_r = \min\{ k : S_k = r \}$ $(r>0)$

Hamming weight: $HW(n) = \#\{ k : X_k = 1,\ k = 1, \ldots, n \}$

last visit time: $LV(2n) = \max\{ 2k : S_{2k} = 0,\ k = 0, \ldots, n\}$

maximum position: $MX(n) = \max\{S_k : k = 0, \ldots, n\}$

sojourn time(滞在時間): $SJ(2n) = 2\sum_{k=1}^{n} 1_{(0,\infty)}(S_{2k-1})$.

これら確率変数の分布は分かっているので，発生したデータの分布と照らし合わせて χ^2 検定を行うことができる。

テストの方法は，パス(1行程の歩数)の長さ 160，パスの本数 50,000 について，上述の確率変数の値分布を調べ，χ^2 検定値を求める。それを 30 回繰り返し，その分布に対する Kolmogorov-Smirnov 検定 K_{30}^{+}, K_{30}^{-} を行い，危険率 0.05 で棄却されるかどうか調べる。これを 1000 回繰り返し，棄却された回数を見る。

検定のための乱数データの作成はＮＩＳＴの検定と同じであるが，ランダムウォーク検定は，ＮＩＳＴの検定ほどには時間がかからないので，ＳＳＩ１６０が生成する１６０ビットのハッシュ値を全て使用する場合と，１６０ビットのうち最初の３２ビットのみを使用した場合の２通りについて検定を行った。結果は以下の通りである。（註．危険率 0.05 の検定であるので，下表中の棄却数 (A), (B) は，$1000 \times 0.05 = 50$ に近いことが望まれる。）

生成される１６０ビットのハッシュ値を全て使用した場合

	K_{30}^{+} 棄却数(A)	K_{30}^{-} 棄却数(B)	棄却数差 (A)-(B)	棄却数計 (A)+(B)
First passage time test				
(SSI)	46	56	-10	102
(SHA1)	59	43	16	102
(RMD)	52	55	-3	107
Hamming weight test				
(SSI)	41	54	-13	95
(SHA1)	63	65	-2	128
(RMD)	49	49	0	98
Last visit time test				
(SSI)	52	49	3	101

(SHA1)	38	47	-9	85
(RMD)	51	50	1	101
Maximun test				
(SSI)	49	53	-4	102
(SHA1)	43	50	-7	93
(RMD)	58	53	5	111
Sojourn time test				
(SSI)	49	56	-7	105
(SHA1)	52	38	14	90
(RMD)	64	63	1	127

	|棄却数差|の総計	棄却数の総計
(SSI)	37	505
(SHA1)	48	498
(RMD)	10	544

１６０ビットのハッシュ値の最初の３２ビットを使用した場合

	K^{+}_{30} 棄却数(A)	K^{-}_{30} 棄却数(B)	棄却数差 (A)-(B)	棄却数計 (A)+(B)
First passage time test				
(SSI)	39	61	-22	100
(SHA1)	40	55	-15	95
(RMD)	48	52	-4	100
Hamming weight test				
(SSI)	54	42	12	96
(SHA1)	55	44	11	99
(RMD)	49	45	4	94
Last visit time test				
(SSI)	50	42	8	92
(SHA1)	56	46	10	102
(RMD)	40	65	-25	105
Maximun test				
(SSI)	43	52	-9	95

(SHA1)	57	51	6	108
(RMD)	37	53	-16	90
Sojourn time test				
(SSI)	53	48	5	101
(SHA1)	50	46	4	96
(RMD)	75	48	27	123

	|棄却数差|の総計	棄却数の総計
(SSI)	56	484
(SHA1)	46	500
(RMD)	76	512

数値を見ると，どれがどうとは言いにくい状況であるが，強いて言えば，RIPEMD-160 において，ハッシュ値の160ビットを全部使った場合と，最初の32ビットを使った場合の，特性の落差が目立つ。

8.6 実行速度

SSI160 が，ファイルデータを読んでハッシュ値を生成する速度を，SHA1，RIPEMD-160 と比べて見る。

使用しているパソコン（Pentium 4 2.8GHz, Windows XP Professional x64 edition ）内にある全ファイル（約 16400 ファイル，約１３ギガバイト）のハッシュ値を生成するのに要した時間は，

	SHA1	:	RMD	:	SSI(B)	:	SSI(A)
[秒]	352	:	477	:	647	:	3530
[比]	1.0	:	1.36	:	1.84	:	10.0

であった。ここで，SSI(B) は，整数計算に long long int（64ビット）を使った場合であり，SSI(A) はそうでない場合（32ビット整数計算のみ）である。通常のパソコンでも使用可能な６４ビット演算を使った場合，SSI160 はかなりの速さを確保できることが分かる。（註）使用したコンパイラは Borland C++ 5.6 である。Borland C++ 5.5（フリーソフト）では， long long int が扱えないので注意が必要である。

ＮＩＳＴ検定用データの作成では，メモリ上の６４ビットの短い入力メッセージに対して１６０ビットのハッシュ値を生成することを多数回繰り返して，１Ｇビットの大きなファイルを作る。このときかかった時間は，

	SHA1	:	RMD	:	SSI(B)
[秒]	11	:	16	:	120
[比]	1.0	:	1.45	:	10.9

であり，SSI160 にかなり分が悪い。同じことを，インテルのＣ＋＋コンパイラを使って行うと，

	SHA1	:	RMD	:	SSI(B)
[秒]	7	:	5	:	25
[比]	1.0	:	0.71	:	3.57

と，SSI160 が持ち直す。これは，SSI160 が，入力データの圧縮よりも，その後のハッシュ値の生成に時間を喰っていることと，インテルのコンパイラがＢＣＣよりも良い６４ビットのコードを生成していることによる，と考えられる。いずれにせよ，短い入力データに対しては，ハッシュ値の生成時間が短いので，相当な負荷のもとで SSI160 が使われない限り，十分に使用に耐えられるものと思われる。

計算機の高性能化に伴い、より長いハッシュ値長を持つハッシュ関数が常に求められている。ＳＳＩのアルゴリズムは、ハッシュ値長に依存しないと考えられるので、この点で非常に有利である。実際、655360ビットのハッシュ値長を持つハッシュ関数を作成してＮＩＳＴの検定を行ってみたが、特に問題となる結果は出なかった。ハッシュ関数を、入力データに依存した乱数を生成する関数とみなす立場からＳＳＩを見れば、特に問題はないと思われるが、セキュリティに対する攻撃についての耐性などは、今後の研究課題である。

【終りに】

2001年に、

「結果を見る限り(SR)法は、確かに乱数を生成しているようである。
乱数性に関する数学的な考察が自然な科学として残されている。」

と、私の講義録[9]の『終りに』に記した。その続きとして、今回、数理的な考察を加えることが出来たことは、私にとってまたＳＲ法にとって非常に幸運なことであった。ＳＲ法を最初に面白いと言ってくれた高嶋恵三氏、理論的な展開の突破口を開いてくれた久保泉先生に感謝する次第である。今後、ＳＳＲ・ＳＳＩ法がいささかでもこの分野の発展に関われることを願っている。

≈≈ 内容などについてお気付きの点がありましたら、是非ご教示下さい≈≈

参 考 文 献

[1] Boyarsky, A., Góra, P.: Laws of Chaos. Birkhäuser, Boston Basel Berlin (1997)

[2] Knuth, D.E.: The Art of Computer Programming, Volume 2. Addison Wesley, Reading, Massachusetts (1997)

[3] Matsumoto, M., Nishimura, T. :Mersenne Twister: a 623-dimensionally equidistributed uniform pseudorandom number generator. ACM Trans. on Modeling and Computer Simulation, **8-1**, 3-30 (1998)

[4] De Melo, W., Van Strien, S.: One-Dimensional Dynamics. Springer, Berlin Heidelberg New York (1993)

[5] Pollicott, M., Yuri, M.: Dynamical Systems and Ergodic Theory. Cambridge Univ Pr, Cambridge (1998)

[6] Robinson, C.: Dynamical Systems. CRC Pr, Boca Raton (1998)

[7] Yaguchi, H.: Randomness of Horner's rule and a new method of generating random number., Monte Carlo Methods and Appl ., **6**, 61-76 (2000)

[8] Yaguchi, H.: Construction of a long-period nonalgebraic and nonrecursive pseudorandom number generator. Monte Carlo Methods and Appl., **8**, 203-213 (2002)

[9] 谷口礼偉 : 数値計算誤差と乱数生成. Rokko Lectures in Mathematics 8, 神戸大学理学部数学教室, 2001.