Rokko Lectures in Mathematics 8

数値計算誤差と乱数生成

谷口 礼偉
(三重大学教育学部)

2001 年 3 月
神戸大学理学部数学教室

Rokko Lectures in Mathematics 8

Generation of random numbers

By

Hirotake Yaguchi (Mie University)

ISBN 4-907719-08-6

March, 2001

Edited and published by

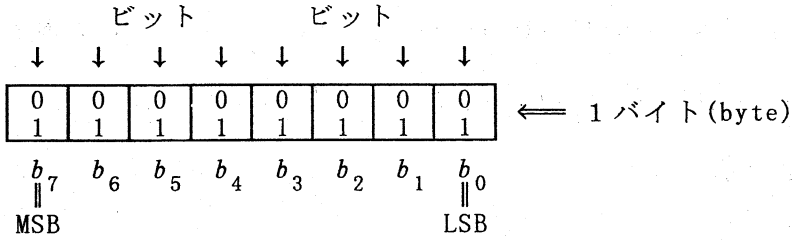
Department of Mathematics
Faculty of Science
Kobe University
Rokko, Kobe, 657-8501 Japan

数値計算誤差と乱数生成

目　次

1．コンピュータ内での数値情報の処理形態

まず，以後必要となる情報科学的な概念をまとめておく。コンピュータ内では 0-1 のディジタル信号で処理が行われている。この世界では {0, 1} が基本状態の全てであり，「0⟷1 を区別する」 ことに対応する情報単位が最小である。この情報の最小単位を**ビット**という。実際の情報処理を行うにあたっては，通常８ビットをまとめて１つの処理単位にしている。これを**バイト**という。



MSB : most significant bit　　**LSB** : least significant bit
（最上位ビット）　　（最下位ビット）

［主記憶(メインメモリ)装置の概念］

byte	byte	byte	byte	byte	···
byte	byte	byte	byte	byte	···
byte	byte	byte	byte	byte	···
···	···	···	···	···	···

各 byte には通し番号(アドレス) が付けられている。

以下しばらく，コンピュータ内での数値の取り扱い方に対するイメージをつかむため，１バイトで整数，２バイトで実数を表そうとする立場に立って少し詳しく考えてみる。

1．1　1バイトで整数値を表す

１バイトでは 2^8=256 種類の情報を見分けることができるが，これらで 256 種の整数値を表そうという立場である。

（イ）0～255の整数値を表すとみる立場

b_i = 0 または 1 ，i=0, ···, 7, であるから，

$$b_7\times2^7 + b_6\times2^6 + b_5\times2^5 + b_4\times2^4 + b_3\times2^3 + b_2\times2^2 + b_1\times2^1 + b_0\times2^0$$

とすれば，１バイトで整数値 0～255 が表される。

（ロ）-128～0～127の整数値を表すとみる立場

b_7=0 の時，

$$b_6\times2^6 + b_5\times2^5 + b_4\times2^4 + b_3\times2^3 + b_2\times2^2 + b_1\times2^1 + b_0\times2^0$$

で計算される**非負**(0または正)の数を表し，

b_7=1 の時，**負**の数で絶対値が $b_7b_6b_5b_4b_3b_2b_1b_0$ の**2の補数**である

$$2^8 - \{b_7\times2^7 + b_6\times2^6 + b_5\times2^5 + b_4\times2^4 + b_3\times2^3 + b_2\times2^2 + b_1\times2^1 + b_0\times2^0\}$$

である数を表す。

[問] 10110001 を上述（イ），（ロ）の立場に立って考察せよ。[177, -79]

[問] 現在使われているパソコンの多くは，32ビット(4バイト)を使って正負の整数値を表している。この場合，表せる整数の範囲を求めよ。
[-2^{31}=-2147483648 ～ 0 ～ 2147483647=2^{31}-1 （10桁程度の整数）]

1．2 コンピュータにおける実数の扱い

コンピュータでは，与えられた実数に対して，例えば

$1234.567 = 1.234567\times10^3$　　$0.01234567 = 1.234567\times10^{-2}$

などのように，小数点を定位置に移動した後の値(仮数)1234567(小数点は省略)と，その際移動した小数点の桁数(指数) 3 あるいは -2 をそれぞれ別々に記憶する（浮動小数点形式）。現実のコンピュータでは2進法でこの操作が行われている。すなわち，実数値 x を以下のように2進数で表して

$$x = x_0\cdot2^p + x_1\cdot2^{p-1} + \cdots + x_p\cdot2^0 + x_{p+1}\cdot2^{-1} + x_{p+2}\cdot2^{-2} + \cdots$$

⇓ 2進表示

$$x_0x_1x_2x_3\cdots x_p \cdot x_{p+1}x_{p+2}\cdots \qquad , x_i=0 \text{ or } 1$$

x_0 = 1

⇓ 小数点移動(正規化:**normalization**)

$$\underbrace{x_0 \cdot x_1x_2x_3\cdots x_px_{p+1}x_{p+2}\cdots}_{\text{仮数部}} \times \underbrace{2^p}_{\text{指数部}}$$

←・・・浮動小数点形式

とし，仮数部，指数部をそれぞれ独立に２進数形式で記憶する。このとき，仮数部の x_0 はいつも１となるので，x_0 を除いた $x_1x_2x_3\cdots$ を仮数として記憶する（**けち表現**，IEEE 754 規格）。

[**例**] $x = 33.75 = 2^5 + 2^0 + \frac{1}{2} + \left(\frac{1}{2}\right)^2$ を2進表示すると，

$$x = \mathbf{1}\cdot 2^5 + \mathbf{0}\cdot 2^4 + \mathbf{0}\cdot 2^3 + \mathbf{0}\cdot 2^2 + \mathbf{0}\cdot 2^1 + \mathbf{1}\cdot 2^0 + \mathbf{1}\cdot 2^{-1} + \mathbf{1}\cdot 2^{-2}$$

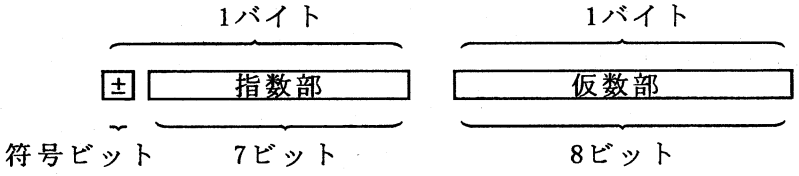
より　100001.11　となる。浮動小数点形式に直せば

$$100001.11 = 1.0000111 \times 2^5$$

となり，仮数は 10000111 ，指数は 10進で 5 となる。よって，コンピュータは，仮数として 0000111(けち表現)，指数として 5 を(あらかじめ定められた方法で)2進で記憶すればよい。(指数部については後で詳しく説明する。)

１．３　２バイトによる実数の世界

簡単のため，仮数部に８ビット（１バイト），指数部に７ビット，数値の正負符号に１ビットを割り当てている浮動小数点形式の場合についてもう少し詳しく考えてみる。



考察する２バイトによる浮動小数点形式

[仮数部の考察]

（１）　指数部 2^p が $1(=2^0)$ のときは，表せる数値は

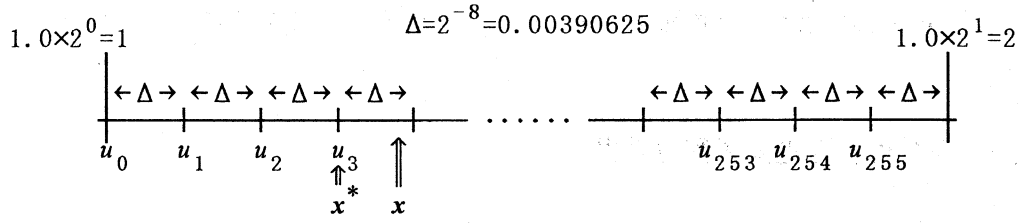
けち表現 ⇓ 1.00000000×2^0 から けち表現 ⇓ 1.11111111×2^0

（1.00000000：仮数部，2^0：指数部）

までの２５６種類となる。すなわち，

	2進		10進	
u_0	= 1.00000000	⟶	$1 + 0\cdot 2^{-8}$	= 1
u_1	= 1.00000001	⟶	$1 + 1\cdot 2^{-8}$	= 1.00390625
u_2	= 1.00000010	⟶	$1 + 2\cdot 2^{-8}$	= 1.0078125
u_3	= 1.00000011	⟶	$1 + 3\cdot 2^{-8}$	= 1.01171875
	·····			
u_{254}	=1.11111110	⟶	$1 + 254\cdot 2^{-8}$	= 1.9921875
u_{255}	=1.11111111	⟶	$1 + 255\cdot 2^{-8}$	= 1.99609375
			(註： $1 + 256\cdot 2^{-8}$	= 2)

のようにとびとびの値を表す：



したがって，この2バイトで実数 $x\in[1,2)$ を表わそうとすれば，x に近い値 $x^* = 1 + k\cdot\Delta$ をあらかじめ決められた方法で選ぶことになる。ここでは x を超えない最大の $1 + k\cdot\Delta$ を x^* とする（⟹切り捨てる）ことにする。

【誤差の限界と有効桁数】

一般に，x を真値，x^* をその近似値とするとき，

$$|x-x^*| \le \varepsilon_a$$

を保証する最小の $\varepsilon_a=\varepsilon_a(x^*)$ を（絶対）**誤差の限界**という。よって，今の場合 $x\in[1,2)$ に対する近似値 x^* に対する誤差の限界は Δ となる。また，x の近似値 $x^*=d_1^*d_2^*\cdots d_p^*.\,d_{p+1}^*d_{p+2}^*\cdots$ に対して，誤差の限界の桁より上にある桁数を**有効桁数**という：

$$x^*=d_1^*d_2^*\cdots\cdots d_p^*.\,d_{p+1}^*\overset{\text{誤差限界の桁}}{\overset{\Downarrow}{d_{p+2}^*}}\cdots$$
$$\longleftarrow \text{有効桁} \longrightarrow$$

したがって，今扱っている浮動小数点形式では，$x\in[1,2)$ にの近似値 x^* について，その有効桁数は，

$$\Delta = 0.00000001\ (2進) = 0.0000000011111.....\ (2進)$$
$$\Downarrow \qquad\qquad \Downarrow$$
$$b_8 \qquad\qquad b_9$$

であるから

$$x^* = 1.b_1^*b_2^*b_3^*\cdots\cdots b_8^*b_9^*\cdots \quad (2進)$$

誤差限界の桁 ⇓

←― 有効桁 ―→

より，2進で9桁となる。10進の場合には，$\Delta = 2^{-8} = 0.00390625$ であるから

$$x^* = 1 + k\cdot\Delta = 1.d_1^*d_2^*d_3^*\cdots\cdots \quad (10進)$$

誤差限界の桁 ⇓

となり，有効桁数は３桁となる。

（２） 指数部 2^p が $2^1=2$ のとき，仮数部は

けち表現 ⇓

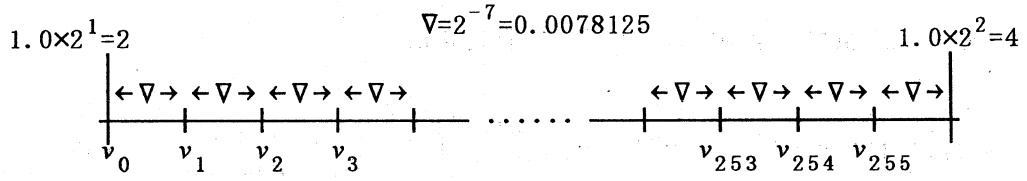
$1.\underbrace{00000000} \times \underbrace{2^1}$ から 1.11111111×2^1

けち表現 ⇓

まで，すなわち，2進で

10.0000000 から 11.1111111

までの256種類の数値 $v_0, v_1, \cdots, v_{255}$ をとびとびで表せる：



この場合も，有効桁数は３桁である。

[演習] 指数部 2^p が一般のときの有効桁数を考察せよ。

（答）仮数は1バイトであるから，2^p と 2^{p+1} の間で256種類の数を区別できる。

$$2^p = 0.d_1d_2d_3\cdots \times 10^q ,$$

$$2^{p+1} = 0.d_1^*d_2^*d_3^*\cdots \times 10^q \quad \text{または} \quad 2^{p+1} = 1.d_1^*d_2^*d_3^*\cdots \times 10^q$$

とするとき，$_1^0d_1^*d_2^*d_3^*\cdots$ と $d_1d_2d_3\cdots$ の差(=$d_1d_2d_3\cdots$)が 256 以下なら，有効桁数は 3 桁，そうでないときは 2 桁になる。ただし $1.d_1^*d_2^*d_3^*$ となったときは，1桁上がる。

[指数部の考察]

指数部は7ビットあるので，2^7=128 種類のビット配列がある。各ビット配列に対して，ここでは，以下のように指数を対応させることにする。

指数部のビット	対応する指数
0000000	〔数0(仮数=0)およびdenormalized数を表すのに使用〕
0000001	$\times 2^{-62}$
·····	···
0111101	$\times 2^{-2}$
0111110	$\times 2^{-1}$
0111111	$\times 2^{0}$
1000000	$\times 2^{1}$
1000001	$\times 2^{2}$
1000010	$\times 2^{3}$
·····	···
1111110	$\times 2^{63}$
1111111	〔無限大(仮数=0) および 非数を表すのに使用〕

このとき，正数で，表すことのできる最大値は，

$$1.11111111 \times 2^{63} \fallingdotseq 2^{64} = \left(10^{\log_{10}2}\right)^{64} = (10^{0.3010})^{64} = 1.845 \times 10^{19}$$

となる。また，正数での最小値は，denormalizedを考慮しないときは
(=normalized, けち表現が有効なとき)，

けち表現
⇓
$1.00000000 \times 2^{-62} = 2.17 \times 10^{-19}$

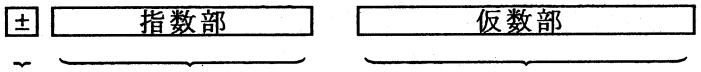
denormalizedを考慮するときは(=normalizeしない, けち表現を使わない)，

$$0.00000001 \times 2^{-62} = 2^{-70} = 8.47 \times 10^{-22}$$

となる。

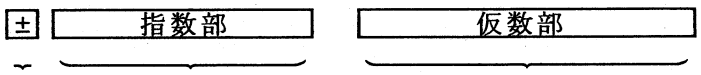
[演習]　現在使われているパソコンの多くは，(IEEE 754 に従い)

単精度と呼ばれる形式では，



符号ビット　+　8ビット　　+　　23ビット　　= 4バイト

倍精度と呼ばれる形式では，



符号ビット　+ 11ビット　　+　　52ビット　　= 8バイト

となっており，記憶方式は2バイトによる浮動少数点形式と同様である。以下について考察せよ：

単精度では、有効桁数は10進で約７桁であり(一部で6桁,8桁)，

1.4×10^{-45} ～ 3.4×10^{38} の範囲の数が扱える。

倍精度では、有効桁数は10進で 15～16 桁であり，

5.3×10^{-324} ～ 1.7×10^{308} の範囲の数が扱える。

[解説]｛単精度｝ 2^{23}=8,388,608 (7桁) であるので，${}^{0}_{1}d^{*}_{1}d^{*}_{2}d^{*}_{3}d^{*}_{4}d^{*}_{5}d^{*}_{6}d^{*}_{7}.\cdots$ と $d_1d_2d_3d_4d_5d_6d_7.\cdots$ の差(=$d_1d_2\cdots d_7.\cdots$)が 8,388,608 以下なら，有効桁数は7桁，そうでないとき（例えば，$x=2^{53}\approx 0.90072\times 10^{16}$）は6桁になる。ただし $1d^{*}_{1}d^{*}_{2}d^{*}_{3}\ldots$ となった数については，1 桁上がる（例えば 7桁⟶8桁）。
｛倍精度｝ $2^{52}\approx 4.5036\times 10^{15}$ (16桁) なので，${}^{0}_{1}d^{*}_{1}d^{*}_{2}d^{*}_{3}d^{*}_{4}\cdots d^{*}_{14}d^{*}_{15}d^{*}_{16}.\cdots$ と $d_1d_2d_3d_4\cdots d_{14}d_{15}d_{16}.\cdots$ との差(=$d_1d_{24}\cdots d_{16}.\cdots$)が $2^{52}\approx 4.5\times 10^{15}$ 以下なら，有効桁数は16桁，そうでないとき(例えば，$x=2^{53}\approx 0.90072\times 10^{16}$)は15桁になる。ただし $1d^{*}_{1}d^{*}_{2}d^{*}_{3}\ldots$ となった数については 1 桁上がるが，16桁⟶17桁となることはない。

2．数値計算における誤差

コンピュータにおける(実)数値計算は，本質的に"近似の世界"であるため誤差が生じる。例えば，無理数である π の正確な値

$$\pi = 3.14159265358979323846264338327950288...\text{(無限長)}$$

をコンピュータでは記憶できない。以下に，コンピュータによる数値計算時に現れる主な誤差についてまとめておく。

（1）丸め誤差 ･･･ 数値をあらかじめ定められた有限の桁数内で記述する（切り捨て，切り上げ，四捨五入など）ために生ずる誤差。

（2）情報落ち誤差 ･･･ $|x|>>|y|$ （$|y|$が$|x|$に比べて非常に小さい）のとき，加減算 $x\pm y$ で，y 情報の一部または全部が失われて生じる誤差（丸め誤差の一種）。

（3）桁落ち誤差 ･･･ x と y の値が非常に近いとき， $x-y$ を行うと有効桁数が激減してしまうという恐ろしい誤差。例えば，有効7桁で演算中

100000.2 － 100000.1 ⟹ 000000.1 となるように。
↑ ↑ ↑
有効7桁 有効7桁 有効1桁

（4）打ち切り誤差(公式誤差) ･･･ 無限項からなる関数の展開式を，有限項で近似するために生じる誤差。

(*) オーバーフロー・アンダーフロー ･･･ 整数型演算ではオーバーフローを起こしてもエラーにはならないので注意が必要である。

3．桁落ち誤差領域におけるHorner法の乱雑性

3.1 sin の数値計算と桁落ち誤差

下図は $\sin x$ を

$$(1)\quad \sin x \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots + (-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!}$$

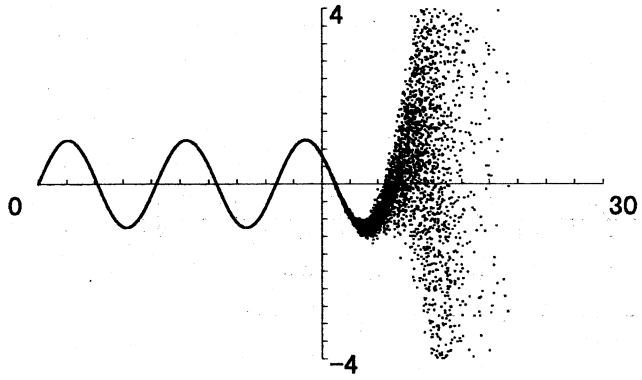
と近似し，n=30 の場合において，

(2) $F_1(x) = x$, $F_k(x) = -F_{k-1}(x) \cdot \frac{x^2}{(2k-2)(2k-1)}$, $k=2, \cdots, 30,$

と単精度計算し,

(3) $\sin x \approx \{\{\{\{F_1(x)+F_2(x)\} +F_3(x)\} +F_4(x)\} +F_5(x)\} +\cdots+F_{30}(x)$

を求めた結果である。



単精度計算では有効桁数が10進で7桁程度であることに留意して，例えば，x=22では，以下のような計算が行われている。

	単精度計算	倍精度計算
F_1	22.0000000000000	22.0000000000000
F_2	-1774.66662597656	-1774.66666666667
F_3	42946.9335937500	42946.9333333333
F_4	-494912.281250000	-494912.279365079
F_5	3326910.25000000	3326910.32239859
F_6	-14638405.0000000	-14638405.4185538
F_7	45416588.0000000	45416591.1703848
F_8	-104674424.000000	-104674429.173649
F_9	186258896.000000	186258910.735463
F_{10}	-263594464.000000	-263594481.859545
F_{11}	303761248.000000	303761260.047666
F_{12}	-290554240.000000	-290554248.741246
F_{13}	234380416.000000	234380427.317938
F_{14}	-161595616.000000	-161595622.253393
F_{15}	96320536.0000000	96320543.3135989
F_{16}	-50128108.0000000	-50128110.7137439
F_{17}	22975382.0000000	22975384.0771326
F_{18}	-9344609.00000000	-9344609.99439681
F_{19}	3395488.50000000	3395488.91688292
F_{20}	-1108918.00000000	-1108918.10780792
F_{21}	327266.031250000	327266.075718922
F_{22}	-87705.8437500000	-87705.8586090578
F_{23}	21439.2070312500	21439.2098822141
F_{24}	-4799.52636718750	-4799.52709666588
F_{25}	987.657653808594	987.657786898931
F_{26}	-187.461288452148	-187.461321121209

F_{27}	32.9213600158691	32.9213640865984
F_{28}	-5.36496257781982	-5.36496303633455
F_{29}	0.813484311103821	0.813484370171028
F_{30}	-0.115057393908501	-0.115057403612735
$\sum_{i=1}^{30} F_i$	-16.1806468963623	-0.0223788063394826

単精度計算では，桁落ち誤差が発生している様子が良く分かる。

3.2 Horner法による計算

Horner法は，(1)を

$$(4)\qquad x\left(1-\frac{x^2}{2\cdot 3}\left(\cdots\left(1-\underbrace{\frac{x^2}{(2n-4)(2n-3)}\left(1-\frac{x^2}{(2n-2)(2n-1)}\right)}_{G_1}\right)\cdots\right)\right)$$

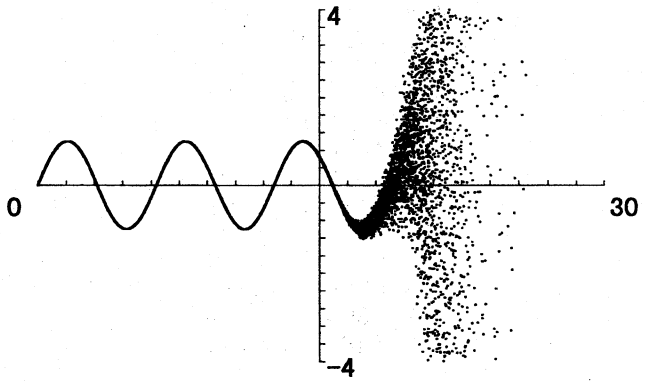
$$\longleftarrow\qquad G_2\qquad\longrightarrow$$

のように変形し，

$$(5)\qquad G_0=1,\quad G_k=1-\frac{x^2}{(2n-2k)(2n-2k+1)}\times G_{k-1},\quad k=1,2,\cdots,n-1,$$

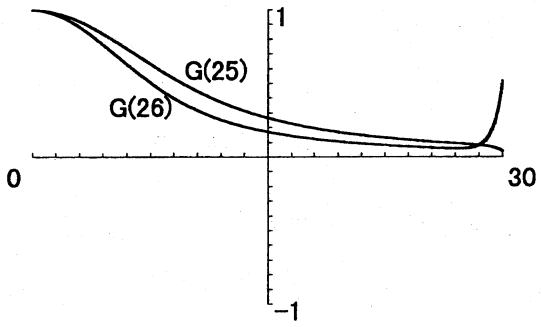
$$G_n=x\times G_{n-1}$$

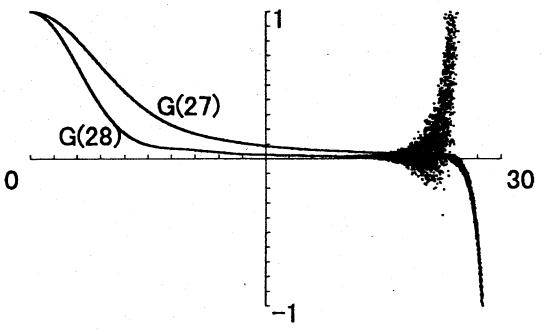
と計算するものである。



桁落ち誤差が起こりにくい計算法のように思われるが，前と同様な条件（n=30）でグラフを描いてみると，やはり突然グラフが乱れる現象が現れる。

計算途中の項 G_i を見てみると，G_{27} あたりから顕著な乱れが観察される。





前と同様，項数 n=30，x=22 で計算を行い，G_0～G_{30} を記しておく：

	単精度計算	倍精度計算
G_0	1.0	1.0
G_1	0.858562231063843	0.858562244301578
G_2	0.869817018508911	0.869817003057029
G_3	0.858252048492432	0.858252043946262
G_4	0.849276483058929	0.849276491556607
G_5	0.838804006576538	0.838803991406511
G_6	0.827388942241669	0.827388974557504
G_7	0.814775109291077	0.814775086176766
G_8	0.800832748413086	0.800832756712346
G_9	0.785380363464355	0.785380368633015
G_{10}	0.768217027187347	0.768217013159525
G_{11}	0.749111294746399	0.749111312841289
G_{12}	0.727800428867340	0.727800393832445
G_{13}	0.703987061977386	0.703987066710165
G_{14}	0.677339255809784	0.677339261091174
G_{15}	0.647492229938507	0.647492255518142
G_{16}	0.614056348800659	0.614056340306921
G_{17}	0.576633512973785	0.576633520358191
G_{18}	0.534848988056183	0.534848960244392
G_{19}	0.488405317068100	0.488405342374929
G_{20}	0.437171012163162	0.437170986406034
G_{21}	0.381313532590866	0.381313574793800
G_{22}	0.321486204862595	0.321486138969856
G_{23}	0.259050846099854	0.259050993993284
G_{24}	0.196278139948845	0.196277685302889
G_{25}	0.136376187205315	0.136378184667289
G_{26}	0.0832489654421806	0.0832355364032255
G_{27}	0.0406547784805298	0.0408095328771154
G_{28}	0.0161543600261211	0.0124093043738067
G_{29}	-0.303118377923965	-0.00101721948707606
G_{30}	-6.66860437393188	-0.0223788287156735

G_{26}，G_{27}，G_{28} あたりで，

$$G_k = 1 - \frac{x^2}{(2n-2k)(2n-2k+1)} \times G_{k-1}$$

⇓計算後の単精度有効桁

$= (-)0.0\text{**********}$

桁落ち⇑　⇑単精度の仮数として実際に記憶される桁

となる（甚だしいものではないが）桁落ち誤差がさみだれ的に発生していることに注目して欲しい。

4．Horner法による乱数の生成

桁落ち誤差領域におけるHorner法計算を用いて，以下のように 10進４桁（0～9999）の数値 20000 個を作成する。

1) $x_0 = 22$ と $x_{19999} = 26$ の間を19999等分して，

$$x_i = x_0 + \frac{x_{19999} - x_0}{19999} \times i \ , \qquad i = 0, 1, \cdots, 19999,$$

を作り，

2) 各 x_i について， sin の近似式 (1) を，Horner法 (5) により単精度計算し，数値 y_i を得る。

3) y_i を浮動小数点表示し，上位３桁の数字を捨て，残った桁から続けて4桁の数をとる。

y_0 ⟹ -6.66**8604**37393188E+000 ⟹ 8604
y_1 ⟹ -7.80**0577**16369629E+000 ⟹ 0577
y_2 ⟹ -7.64**5578**6123657E+000 ⟹ 5578
y_3 ⟹ 4.75**3845**21484375E+000 ⟹ 3845

得られる数値は次のようになる：

8604, 0577, 5578, 3845, 6527, 7537, 5051, 2681, 7372, 4805
8009, 6213, 4359, 2754, 1024, 7043, 3334, 9752, 0997, 3157
3356, 1578, 3988, 6373, 2061, 9528, 7992, 8025, 8380, ・・・.

注意：使用する計算機および言語によって単精度計算の扱いが異なるので，数学的に同じ条件であっても上とは異なった乱数値が生成されることがある(まずないと思われるが)。

これらの数値に対する乱数性の検証は，次章において行う。

5．Horner法による乱数の検定

5.1 乱数とは？

でたらめな数が次から次へと出てくる状況であるが，数学的には以下のようにとらえられる：

・**独立性** ･･･ 出現する数の間には因果関係が無い。すなわち，

$\Omega = \{ 0, 1, 2, \cdots, n-1 \}$　(n種類の値をとる乱数値の空間)

[例] Ω = さいころを投げたときに，出る可能性のある目全体

X_i , $i \in \{ 0, 1, 2, \cdots \}$　(i番目の乱数値を表す確率変数)

[例] i 番目にさいころを投げたときに出た目を X_i とする。

とするとき，

〔独立性〕　$\mathrm{Prob}\{X_1=k_1, X_2=k_2, \cdots, X_\ell=k_\ell\} = \Pi_{i=1}^{\ell} \mathrm{Prob}\{X_i=k_i\}$

(Prob : probability, 確率)

・**一様性** ･･･ 発生可能な数値が同程度に発生されること，すなわち，

〔一様性〕　任意の i, k について　$\mathrm{Prob}\{X_i=k\} = \frac{1}{n}$

が要求されることもある。(一様乱数)

5.2 仮説の検定

統計学では，以下のプロセスで「仮説の検定」を行う。

（1）データに関する正しいと思われる仮説を設定して，得られたデータから作られる値(統計量=確率変数)が満たすべき確率分布をあらじめ導出しておく。

＊ この講義では χ^2 統計量，KS統計量をよく使う。

（2）（1）で得られた確率分布において，余り起こらないと思われるある特定の領域（棄却域）をあらかじめ定めておく。

（3）実際に出てきたデータから求めた値が，棄却域に入ってしまったら，（1）で正しいと予想したことを反省して，仮説を否定する。

注意：棄却域に入らなかったら，（1）の仮説を否定しないが，積極的に正しいと主張しているわけではないことに注意せよ！

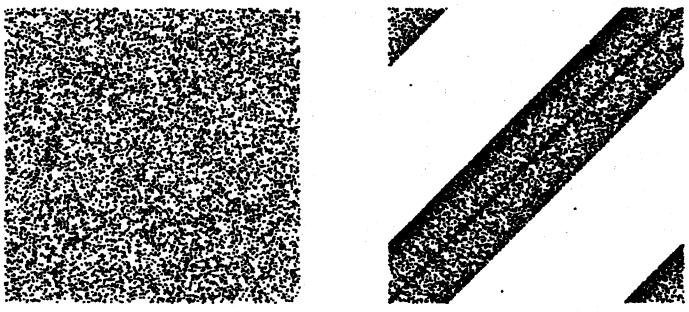
5.3 Horner法乱数の一様乱数性の検証

検定(I) 予備評価

左図は発生した数値を２つずつ組み合わせ，10000個のXY平面上の点

(0.8604, 0.0577), (0.5578, 0.3845), (0.6527, 0.7537), ･･･

を作り，プロットしたものである。**右図**は同様のことを，x=0 と x=4 の間を19999等分して行なったものである。



非桁落ち領域では一様乱数性が期待できないことが分かる。

左図の10000個の点のうち $x^2+y^2<1$ を満たすものの総数は7852個である。これ

より，π　0.7852×4=3.1408 と計算される。このとき，真値π=3.1415926...
との相対誤差は $(3.1408-\pi)/\pi = -2.52309\times10^{-4}$ となる。

検定(II) 文字0～9の出現頻度のχ^2検定（一様性の検定）

発生した４桁の整数値20000個に含まれている文字 0,1,...,9 の分布は以下の通りである：

文字0	1	2	3	4	5	6	7	8	9
8022	8021	7897	7992	8067	8107	7953	8016	8022	7903

ここで，80000個の０～９の数字が均等に出現しているかどうかを調べる。

「５．２ 乱数の検定」に基づいて検定する。

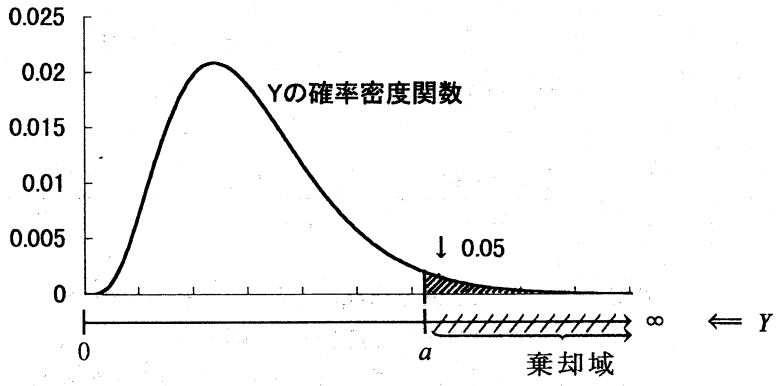
（１_１）０～９の数字が均等な確率（=0.1）でお互いに無関係に出現していると仮定してみる。（⟹各数字は，理想的には 8000 個現れることになる。）

（１_２）現実には，０～９の数字は理想通りには現れなく，バラツキがあるので，0 の発生数は X_0 ，1 の発生数は X_1 ，･･･，9 の発生数は X_9 となる。（$X_0,\cdots,X_9$ はある分布にしたがう確率変数になっている。）

（１_３）新しい確率変数 Y を

$$Y = \frac{(X_0-8000)^2}{8000} + \frac{(X_1-8000)^2}{8000} + \cdots + \frac{(X_9-8000)^2}{8000}$$

で定めれば，Y の分布は，自由度 9 の χ^2分布にほぼしたがうことが知られている。



（２）Y は $[0,\infty)$ の値をとるが，0 から離れるほど起こりにくい。よって，a を適切にとって，Y が $[a,\infty)$ の範囲に起こる確率を 0.05 になるようにする。すなわち，$P(a\leq Y<\infty)=0.05$ となるように a を定める。
（$[a,\infty)$ が棄却域になり，0.05 は有意水準，危険率と言われる。）

（３）確率変数 $X_0,\cdots,X_9$ に対して，実際に得た値 X_0=8022，$\cdots$，X_9=7903 から確率変数 Y の実現値 y を計算して，その値が棄却域 $[a,\infty)$ に入るかどうかをチェックすれば良い。マイクロソフトの表計算ソフト EXCEL の CHITEST 関数では，$x_0,\cdots,x_9$ から y の値を計算してχ^2分布における $[y,\infty)$ の確率，すなわち，$P(y\leq Y<\infty)$ を答として返してくる。したがって，

y = [EXCEL の CHITEST 関数の答] < 0.05

の時は y が棄却域に入ったことになり，仮説を棄却する。
我々の場合，

y = [EXCEL の CHITEST 関数の答] = 0.8355

であるので，0.8355>0.05 となり，数字0～9の出現頻度は一様(8000)であるという仮説は棄却できないことが分かる。

【χ^2検定】一般に，独立同分布のデータを無作為に n 個とったとき，各データは互いに交わらない k 個のクラス $A_1,\cdots,A_k$ の何れかに，確率 $p_1,\cdots,p_k$ で属するものとする。n 個のデータのうち，クラス A_i に入った個数を表す確率変数を X_i とすれば，n が十分大きいとき，確率変数

$$Y=\frac{(X_1-np_1)^2}{np_1}+\frac{(X_2-np_2)^2}{np_2}+\cdots+\frac{(X_k-np_k)^2}{np_k}$$

の分布は自由度 k-1 の χ^2分布にほぼしたがうことが知られている。したがって，$X_1,\cdots,X_k$ の実現値 $x_1,\cdots,x_k$ に対して Y の実現値 y を

$$y=\frac{(x_1-np_1)^2}{np_1}+\frac{(x_2-np_2)^2}{np_2}+\cdots+\frac{(x_k-np_k)^2}{np_k}$$

で計算し，それが棄却域に入るかどうかで検定を行うことができる。EXCEL の CHITEST 関数は，x_i，np_i，i=1,$\cdots$,k，から，y を計算し，自由度 k-1 の χ^2分布をする確率変数 Y に対する $P(y\leq Y<\infty)$ を答としている。

検定(III) 文字0の出現間隔のχ^2検定（独立性の検定）

発生した2万個の数値には 8022 個 の文字 0 がある。

8604, 0577, 5578, 3845, 6527, 7537, 5051, 2681, 7372, 4805 ･･･

↑←1→↑←－－－－－ 20 －－－－－－→↑←－－ 11 －－→↑←･･･

最初に 0 が現れてから，以後順々に 0 が発生するまでの間隔の度数分布は，以下の通りである。

間隔	0	1	2	3	4	5	6	7	8
回数	787	745	655	600	506	471	400	386	369
理論回数	802.1	721.9	649.7	584.7	526.3	473.6	426.3	383.6	345.3

9	10	11	12	13	14	15	16	17	18
349	260	259	218	202	177	163	143	142	116
310.7	279.7	251.7	226.5	203.9	183.5	165.1	148.6	133.8	120.4

19	20	21	22	23	24	25	16	27	28
110	99	93	82	68	54	48	43	48	54
108.4	97.5	87.8	79.0	71.1	64.0	57.6	51.8	46.6	42.0

29	30	31	32	33	34	35	36	37	38以上
32	27	34	32	25	23	18	21	16	146
37.8	34.0	30.6	27.5	24.8	22.3	20.1	18.1	16.3	146.4

数字 0 が独立に発生すると仮定すれば出現間隔の理論分布は幾何分布であるので，p=0.1 とおくと，間隔 k で発生する確率は $(1-p)^k \cdot p$，間隔 k 以上で発生する確率は $(1-p)^k$ となる。これから，全部で 8022個ある0の間隔の理論出現回数を計算して(上表参照)χ^2検定を MS-EXCELで行えば(CHITEST)，値 0.9262 が得られ，0.9262>0.05 となり，危険率0.05 で仮説は棄却できない。すなわち，0 が独立に発生しているという仮定は否定できない。

検定(IV) 一様分布に対する Kolmogorov-Smirnov 検定 (一様性の検定)

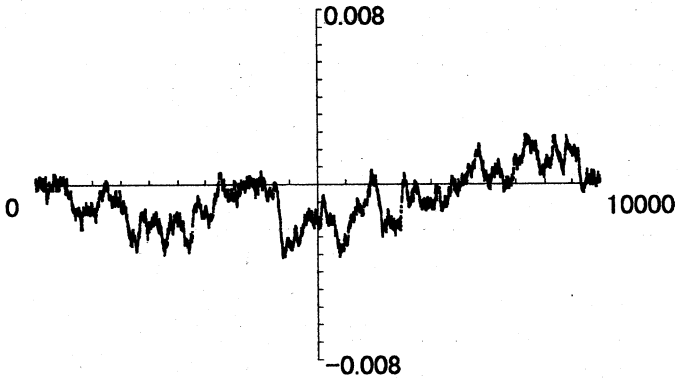
最後に，我々の数値を，区間 [0,1] 上の一様分布にしたがう乱数とみなして，Kolmogorov-Smirnov検定(KS検定と略)を行う。[0,1] 上の一様分布の分布関数 F は，$x\in[0,1]$ では $F(x)=x$ で与えられる。発生した乱数値を順に u_1, u_2, u_3, $\cdots$, u_n, $n=20000$, とおき，数値 $0.u_1$, $0.u_2$, $0.u_3$, $\cdots$, の分布関数を

$$F_n(x) = \frac{1}{n}\,\#\{\, i \mid 0.u_i \le x\},\quad x\in[0,1)$$

で定める(#=集合に含まれる要素の個数)。このとき，

$$F_n(x) - F(x) \qquad (\Longleftarrow F_n \text{ の一様分布からのずれ})$$

のグラフは下図のようになる。



Kolmogorov-Smirnov検定は，

$K_n^+ = \sqrt{n}\,\max_x\{F_n(x)-F(x)\}$,

$K_n^- = \sqrt{n}\,\max_x\{F(x)-F_n(x)\}$

を評価するものである。

まず，$\#\{\, i \mid u_i \le k\,\}-2(k+1)$, $k=0,1,\cdots,9999$, を求めると (#=集合の要素の個数)，その最大値は 43，最小値は -69 である。これから

$$\max_x\{F_n(x)-F(x)\} = 43/20000,\qquad \max_x\{F(x)-F_n(x)\} = 69/20000$$

となり $K_n^+=43/\sqrt{20000}=0.3041$ ， $K_n^-=0.4879$ と計算される。この統計量の危険率 0.05 に対応する値の近似値は $\sqrt{-0.5\ln 0.05}=1.2239$ であるので([4])，分布が一様であるという仮説は棄却できない。

●以上の検定から，Horner法計算により発生した数値は一様乱数と呼んでよいであろう。(この乱数の性質は後半(§9)でさらに詳しく調べる。) 我々はこの乱数現象をヒントにして，以後，新しい乱数生成法について考察していくことにする。

6．新しい乱数生成法(実数シフト法,(SR)法)

Horner法による乱数発生の主要因は

$$G_k = 1 - \frac{x^2}{(2m-2k)(2m-2k+1)} \times G_{k-1} \quad = (-)0.0\text{**********}$$

単精度(桁落ち前の仮数)　単精度(桁落ち後の仮数)

による桁落ちと考えられる。これは桁落ち誤差領域の x に対して発生し，かなり偶然で自然発生的なものである。この桁落ちを，数字データの上位桁が落ちて下位桁が持ち上がる左シフトと考え，左シフトを人工的に発生させて，新たな乱数発生法（実数シフト法）を導く。

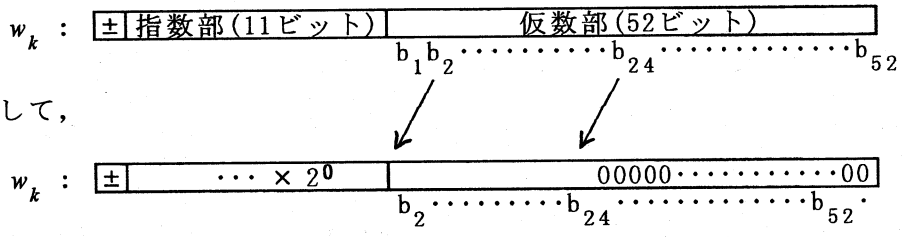
6.1 実数シフト計算(the shift-real computation)

$$f(x) = x \cdot \frac{x}{2} \cdot \frac{x}{3} \cdot \frac{x}{4} \cdots \frac{x}{23} \cdot \frac{x}{24}$$

を

(6)　$w_0 := 1$　　$w_k := w_{k-1} \times \frac{x}{k}$，　$k=1,2,\cdots,24,$

により倍精度計算する。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

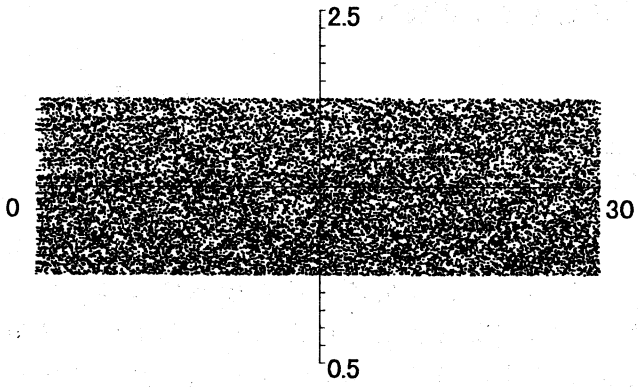


i) 仮数部の全てのビット値を１ビット左にシフトし，
（$\Longrightarrow$ 最左端ビットは捨てる），

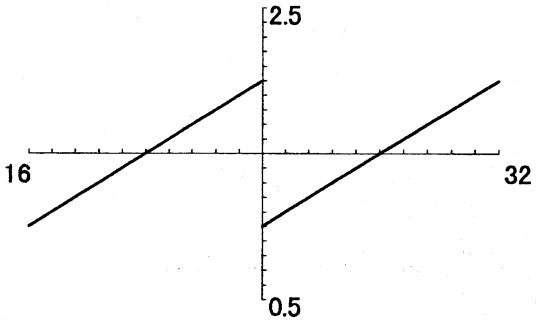
ii) さらに，仮数部上位23ビット以外のビットは０にする，
($\Longrightarrow$ 単精度に相当)。

iii) 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)。

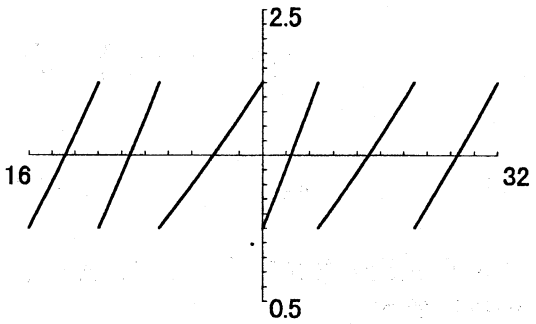
このような計算法を，**実数シフト計算**と呼ぶことにする。実数シフト計算による $f(x)$ のグラフは以下のようになる：



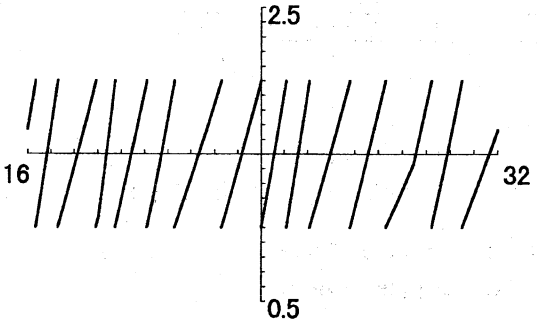
実数シフト計算の途中の計算状況を以下のグラフで示しておく：



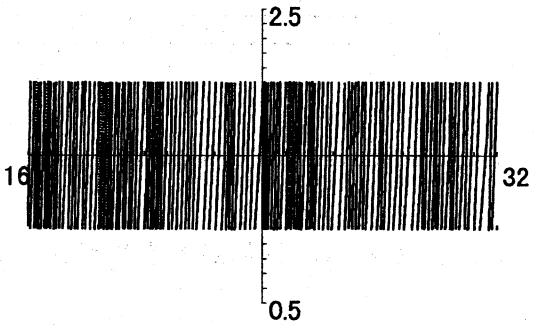
w_1



w_2



w_3



w_6

実数シフト計算のリスト

【PASCAL(Delphi)】

```
type
  Dbl_I2 = record
    case I2 : Boolean of
      True  : (L, H : integer);
      False : (Dbl : double);
  end;
function  SR(x:double) : double;
  var
    i : integer;
    FLLH : Dbl_I2;
  begin
    FLLH.Dbl:=1;
    for i:=1 to 24 do
      begin
        FLLH.Dbl:=FLLH.Dbl*x/i;
        FLLH.H:=(FLLH.H and $0007ffff) shl 1;    {shifted-mantissa H}
        if (FLLH.L and  $80000000)<>0 then inc(FLLH.H);
        FLLH.L:=(FLLH.L and $f0000000) shl 1;    {shifted-mantissa L}
        FLLH.H:=FLLH.H or $3ff00000;               {set exponent 2^0}
      end;
    SR:=FLLH.Dbl;
  end;
```

【FORTRAN】

```
C  **********************
C  IN MS-WINDOWS, DBLHL(0) AND DBLHL(1) BELOW MUST BE EXCHANGED
C  **********************
      FUNCTION SR(X)
      DOUBLE PRECISION SR,X,DBLFL
      INTEGER  I,DBLHL(0:1)
      EQUIVALENCE(DBLFL,DBLHL)
      INTEGER  Z7FFFF,Z807,ZF07,Z3FF05
      DATA      Z7FFFF/524287/,Z807/-2147483648/
      DATA      ZF07/-268435456/,Z3FF05/1072693248/
      DBLFL=1
      DO 200 I=1,24
        DBLFL=DBLFL*X/I
        DBLHL(0)=ISHFT(IAND(DBLHL(0),Z7FFFF),1)
        IF (IAND(DBLHL(1),Z807).NE.0) DBLHL(0)=DBLHL(0)+1
        DBLHL(1)=ISHFT(IAND(DBLHL(1),ZF07),1)
        DBLHL(0)=IOR(DBLHL(0),Z3FF05)
200   CONTINUE
      SR=DBLFL
      RETURN
      END
```

【C については Appendix 参照】

6.2 実数シフト法による乱数の生成 (the shift-real method)

Horner 法計算と同様に,

1) x_0= 22 と x_{19999}=26 の間を19999等分して,

$$x_i = x_0 + \frac{x_{19999}-x_0}{19999} \times i \ , \quad i=0, 1, \cdots, 19999,$$

を作り,

2) 各 x_i について, $f(x_i)$ を, 実数シフト計算により倍精度計算し, 数値 f_i を得る。

3) f_i を浮動小数点表示し, 上位３桁の数字を捨て, 残った桁から続けて4桁の数をとる。

このように実数シフト計算を使って4桁整数を得る方法を, **実数シフト法 ((SR)法)** と呼ぶことにする。得られる20000個の数値は次のようになる:

2225, 7209, 2438, 3418, 9087, 2178, 1622, 6854, 0399, 3480
9106, 7999, 2569, 3375, 3375, 4945, 3534, 7295, 7722, ・・・.

6.3 実数シフト法による乱数の検定

(SR)法について, §5.3 の検定を適用した結果を以下にまとめておく。表中の Delphi は, Borland の Delphi 4.0 に付属している乱数関数 random(10000) によるものである(RandSeed=1)。

	発生法	π	相対誤差	0-9文字χ^2	0間隔χ^2	K^+_{20000}	K^-_{20000}
	Delphi	3.1484	0.002167	0.8393	0.8841	1.0677	0.2758
	Horner	3.1408	-0.0002523	0.8355	0.9262	0.3041	0.4879
⟹	(SR)法	3.1264	-0.004836	0.7390	0.4085	0.4243	0.6152

これらの結果から, 「実数シフト法」は一様乱数を生成すると考えられる。

実数シフト法は次のようなパラメータを持っている：

	現設定
(1) 分割数(n-1)	19999
(2) 分割される x の区間	[22,26]
(3) $f(x)$	$f(x) = x \cdot \frac{x}{2} \cdot \frac{x}{3} \cdot \frac{x}{4} \cdots \frac{x}{23} \cdot \frac{x}{24}$

これらは自由に変えることができ，「実数シフト法は**乱数系列設定の自由度が非常に高い**乱数生成法」ということができよう。

7．実数シフト法の改良I

実数シフト法で分割数 n-1 を多数変化させていくと $K_n^+ > K_n^-$ の傾向がある。これは，

KS検定の $F_n(x) - F(x)$ のグラフが上半面に滞在しやすい
n が多い

ことを意味している。例えば，

$$T_{80000}^+ = \{ n \mid K_n^+ > K_n^- ,\ 80000 \leq n < 89999 \}$$

と置いたとき，(SR)法では $\#T_{80000}^+ = 6224$ となる。今対象になっている分布は一様分布であるから，$K_n^+ > K_n^-$ も $K_n^+ < K_n^-$ も同程度に起こり，$\#T_{80000}^+$ は 5000 に近い値になることが期待される。この期待される値からのずれの原因は，x^k (k>1) が x≥0 において，狭義単調凸関数であるためと考えられる（§6.1 のグラフ参照）。

このことは，良い乱数特性を持つ特定の n を単発的に使う場合には問題ないが，多数の n を使う場合には問題となるので改良を試みる。

7.1 改良実数シフト法I(SR/1)

【実数シフト計算I】 実数シフト法による f の計算結果を $f_i \in [1,2)$ ，f_i より得られる乱数値を $v_1 v_2 v_3 v_4$ とするとき，

$f_i < 1+\alpha$ または $f_i \geq 2-\alpha$

$\Longrightarrow$ $9999 - v_1 v_2 v_3 v_4$ を発生乱数値とする。

この方法を，「実数シフト法I(SR/1)」と呼ぶことにする。

［**改良例**］α=0.15 とすると，(SR/1)法では $\#T^{+}_{80000}$ = 5030 となる。

註１． 単精度，倍精度の扱いはコンピュータシステムにより扱いが異なることが多い。したがって，(SR)法でここに書かれた結果と同一の結果を得たいときは，[7] に記した Delphi 4.0 のプログラムリストをそのまま使い，Windows 95/98 上で計算を行って欲しい。また，実数計算においては，保証された有効桁数より大きい桁における値は，計算途中の結果をメモリにいったん保存するかしないかで，異なることがある。これは，MPU（マイクロプロセッサ）にある実数レジスタのビット長が，メモリに保存されるビット長と異なるためである。

註２． Windows 95/98 で(SR)法計算を行うと，まったく同じプログラムを動かしても計算結果が微妙に異なることがある。例えば

［**改良例**］α=0.15 とすると，(SR/1)法では $\#T^{+}_{80000}$ = 5033 となる

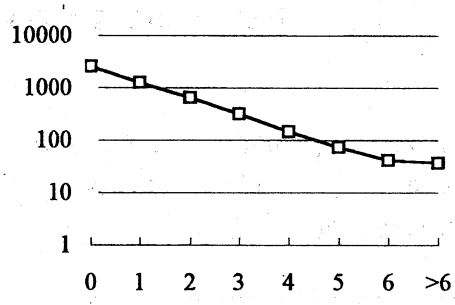
という具合である。これは，MPUが内蔵する数値演算プロセッサにある，コントロールレジスタ"PC"の状態が，Windows 95/98 では一定していないことによる。このレジスタは，数値演算時における浮動少数点形式の仮数部の長さを決めている([3])。

註３． $K^{+}_{n} > K^{-}_{n}$ を調べるとき，数式上 $K^{+}_{n} = K^{-}_{n}$ であっても，コンピュータで計算すると，計算誤差のため $K^{+}_{n} > K^{-}_{n}$ あるいは $K^{+}_{n} < K^{-}_{n}$ となることが多いので注意を要する。これも，計算環境に依存して，$\#T^{+}_{80000}$ に微妙な影響を与える。

7．2 分割数 n-1 に対応して生成される乱数系列間の独立性

上述の(SR/1)法では，区間 [22,26) を 79999～89998 等分して，K^{+}_{n} , K^{-}_{n} , n=80000,...,89999, を算出した。ある n で $K^{+}_{n} > K^{-}_{n}$ となったとき，n+1 で $K^{+}_{n+1} > K^{-}_{n+1}$ となる可能性はどのようなものであるか調べよう。(SR/1)において $T^{+}_{80000} = \{n_1 \le n_2 \le \cdots \le n_{5030}\}$ としたとき， n_i の発生間隔 $n_{i+1}-n_i-1$ の分布および対数度数グラフは以下の通りである。

発生間隔	0	1	2	3	4	5	6	≥7
回数	2533	1239	644	319	143	73	41	37
期待回数	2515	1257	629	314	157	79	39	39



$K_n^+ > K_n^-$ が確率 $\frac{1}{2}$ で独立に発生するとすれば，発生間隔が t である確率は 0.5^{t+1} であるので，CHITEST を行うことができる。結果は 0.9091 であり，仮説は棄却されない。

●各 n に対する乱数系列間に独立性があるとすれば，実数シフト法は
多くの n に対応する乱数系列を，同時並列的に生成する
並列処理に適しているといえよう。

8．実数シフト法の改良II

K_n^+, K_n^- の分布は，n が十分大きいとき，分布 $P(x)=1-\exp(-2x^2)$ に近づくことが知られている([4])。(SR/1) をもう少し精密化して，K_n^+ の分布が $P(x)$ に良く合うようにする。

8.1 改良実数シフト法II(SR/2)

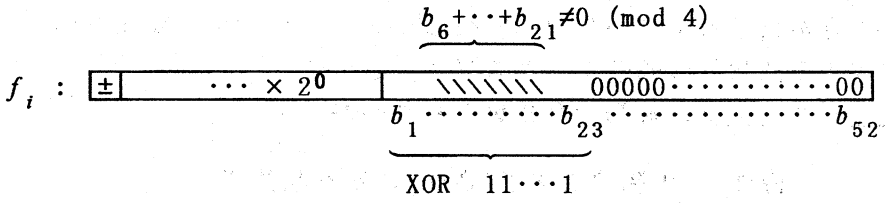
【実数シフト計算II】 実数シフト法による $f(x_i)$ の計算結果 f_i が，

「$f_i < 1+\alpha$　または　$f_i \geq 2-\alpha$」

のとき，さらに，

「f_i の仮数部 6～21 ビット値の和が $\neq 0 \pmod 4$」

ならば，f_i の仮数部 1～23 ビットの全ビット値を反転する。

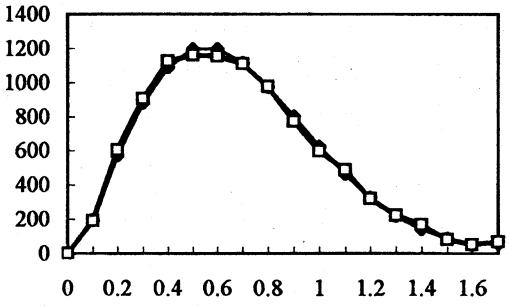


この方法を，「実数シフト法II(SR/2)」と呼ぶことにする。この方法では，f_i がある区間に入ると，確率 約3/4 で，b_1～b_{23} の全ビットが反転するものと期待されている。

計算例： α=0.19 とおくと，(SR/2)では $\#T^+_{80000}$ =5051 となる。また，K^+_n ，n=80000,...,89999, について，区間 I_1=[0,0.1]，I_j=(0.1×(j-1)，0.1×j]，j=2,...16，I_{17}=(1.6,∞) に入るものの個数を調べると，

区間	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9
回数	192	605	905	1125	1162	1155	1108	976	774
期待値	198.0	570.8	878.5	1091.2	1196.2	1197.8	1114.4	972.7	801.4

区間	I_{10}	I_{11}	I_{12}	I_{13}	I_{14}	I_{15}	I_{16}	I_{17}
回数	598	487	321	222	169	81	53	67
期待値	625.6	464.1	327.9	220.9	142.1	87.3	51.3	59.8



となる(**左図参照**)。これらの値について CHITEST を行うと結果 0.4147 を得る。ちなみに，(SR/1) による同様の結果は，CHITEST=0.0741 である。

9．さらに詳しく乱数を検定する

この章では，Knuth[4] に基づいて，さらに４つの検証を追加する。

検定(V) 単純下降連[上昇連]テスト

i 番目の乱数値を x_i ，i=1,2,3,...，とする。ある乱数値から調べ始めて，乱数値の下降が止まるまでの(下降)乱数値を並べたものを下降連という。またその連を構成する乱数値の個数を「連の長さ」という。

単純下降連テストは，最初の乱数値から調べ始めて，

下降連 + 下降連を終了(逆転)させる乱数値

$$\underset{\longleftarrow}{x_1>x_2>x_3>x_4\leq x_5}\ \underset{\longleftarrow}{x_6>x_7>x_8\leq x_9}\ \underset{\leftarrow\cdots}{\cdots\cdots}$$

までを１つの下降連ブロックとみなし(上図では $x_1>x_2>x_3>x_4\leq x_5$, 下降連の長さは 4)，続く下降連ブロックを残る乱数値から同様に構成していく(上図では $x_6>x_7>x_8\leq x_9$, 連の長さは 3)。この単純下降連では，各 x_i がある区間 $[a, b]$ 内の実数値を独立かつ一様にとるとき，長さ t 以上の連が現れる確率は，

$$\left| \{ (x_1, x_2, \ldots, x_t) \in [0,1]^t \mid x_1>x_2>\cdots>x_t \} \right|$$

$$= \int_0^1 dx_1 \int_0^{x_1} dx_2 \cdots \int_0^{x_{t-1}} dx_t = \left[\frac{1}{t!}x_1^t\right]_0^1 = \frac{1}{t!}$$

より $\frac{1}{t!}$ であり，長さ r の連が現れる確率は $\frac{1}{r!} - \frac{1}{(r+1)!}$ である。このように発生確率が計算されるので，実際の連の個数を調べて χ^2 検定を行うことができる。

検定(VI) 4枚の 0～9 カードによる古典ポーカーテスト

数字 0, 1, ... , 9 のうちの何れか１つが書いてある $4n$ 枚のカードを並べる。カード上にある各数字を Y_0, Y_1, Y_2, Y_3, ..., Y_{4n-2}, Y_{4n-1} とする。このとき，4枚のカードの組 $\{Y_{4j}, Y_{4j+1}, Y_{4j+2}, Y_{4j+3}\}$, $j=0, 1, \ldots, n-1$, について，数字の組み合わせパターンを調べる(数字の出現順序は問わない)。

	組み合わせパターンの数		
All different : abcd	$10\cdot9\cdot8\cdot7$	=	5040
One pair : aabc	$10\cdot9\cdot8\times3\times2$	=	4320
Three of a kind : aaab	$10\cdot9\times4$	=	360
Two pairs : aabb	$10\cdot9\times3$	=	270
Four of a kind : aaaa	10	=	10

これから計算される出現個数の期待値と実際の出現個数について χ^2 検定を行う。

[演習] 20000個の4桁乱数に対する，組み合わせパターンの数をカウントするプログラムを考えよ。

検定(VII) 遅れ k の系列相関テスト

発生乱数を $U_0, U_1, U_2, \ldots, U_{n-1}$ とするとき，U_{j+1} がどの程度 U_j に関係しているかを計る尺度として，系列相関係数

$$C = \frac{n(U_0U_1+U_1U_2+\cdots+U_{n-2}U_{n-1}+U_{n-1}U_0)-(U_0+U_1+\cdots+U_{n-1})^2}{n(U_0^2+U_2^2+\cdots+U_{n-1}^2)-(U_0+U_1+\cdots+U_{n-1})^2}$$

を計算することができる。

一般に，統計学における相関係数 r は

$$r = \frac{\sigma_{XY}}{\sigma_X\sigma_Y}$$

である。ここで σ_{XY}，σ_X，σ_Y はそれぞれ X,Y の共分散，X の標準偏差，Y の標準偏差であり，

$$\sigma_{XY} = \frac{1}{n}\sum_{i=1}^{n}(X_i-\overline{X})(Y_i-\overline{Y}) \qquad (\overline{X}: X\text{の平均値}, \overline{Y}: Y\text{の平均値})$$

$$\sigma_X = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i-\overline{X})^2}, \qquad \sigma_Y = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i-\overline{Y})^2},$$

で与えられる。$n\sigma_{XY}$ が，ベクトル（$X_1-\overline{X}, \ldots, X_n-\overline{X}$）と（$Y_1-\overline{Y}, \ldots, Y_n-\overline{Y}$）との内積であることに注意すれば，$-1 \le r \le 1$ であり，また，無相関の時には $r=0$ になることがわかる。$X_i = U_{i-1}$，$Y_i = U_i$ とおけば，遅れ 1 の系列相関係数 C が得られる。この C は，

$$\mu_n = \frac{-1}{n-1}, \qquad \sigma_n^2 = \frac{n^2}{(n-1)^2(n-2)}$$

とおくと，ほぼ 95% の確率で，$\mu_n-2\sigma_n$ と $\mu_n+2\sigma_n$ の間にある。$n=20000$ のとき，$[\mu_n-2\sigma_n, \mu_n+2\sigma_n]$ は $[-0.01419, 0.01409]$ となる。$X_i = U_{i-1}$，$Y_i = U_{i+k-1}$ とすれば，遅れ k の系列相関係数が得られる。

検定(VIII) 衝突テスト

m 個の壷を準備しておき，そこに n 個の玉を 1 個ごとにでたらめに入れていく。ここで m は n に比べて非常に大きいものとする。殆どの玉は空の壷に入っていくと思われるが，なかには，既に玉が入っている壷にまた入ることになる（玉の衝突が起こったと言うことにする）。衝突テストは，この玉の衝突回数 c を数えるものである。ある乱数生成法について，c が多すぎたり，あまりにも少

ないときは，このテストはパスできないことになる。c 回の衝突が起こる確率は $n-c$ 個の壷が n 個の玉によって占められる確率であるから，

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

となる。$m(m-1)\ldots(m-n+c+1)$ は，m 個の壷から $n-c$ 個の壷を順序を付けて選ぶ場合の数，$\left\{ \begin{matrix} n \\ n-c \end{matrix} \right\}$ (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。

ここでは，4桁数2つから 6桁数1つを *8604 0577* → *860577* のように作り，計1万個作る。この場合は，壷の数 m=1000000，玉の数 n=10000 となり，衝突回数の分布は以下のようになる：

衝突回数≤	34	38	43	49	58	61	62	67
確率	.011	.049	.184	.491	.890	.948	.961	.992

よって，衝突回数が61回以下ならテストOKとする。

衝突の確率は [4] の Algorithm S により求めたが，具体的なプログラムが略されているので，以下に載せておく。ちょうど j 個の壷が使われている(=10000$-j$ 回の衝突が起こっている)確率が，ClsDstTbl[j] に入る。

[衝突の確率を求めるプログラム]

```
procedure TForm1.Button1Click(Sender: TObject);
const
  m : longint =1000000;
var
  i, j, j0, j1 : longint;
  ClsDstTbl : array[0..10000] of double;
  ClsDst : double;
begin
  for i:=0 to 10000 do ClsDstTbl[i]:=0;
  ClsDstTbl[1]:=1;
  j0:=1; j1:=1;
  for i:=2 to 10000 do
    begin
      inc(j1);
      for j:=j1 downto j0 do
        begin
          ClsDstTbl[j]:=(j/m)*ClsDstTbl[j]
                        +((1+(1/m))-(j/m))*ClsDstTbl[j-1];
          if ClsDstTbl[j]<1E-20 then
            begin
              ClsDstTbl[j]:=0;
              if j=j1 then dec(j1) else
                if j=j0 then inc(j0);
            end;
        end;
    end;
end;
```

検定結果　　(test(I)-test(IV)の説明は§5)

method	test (I)	(II)	(III)	K^+ (IV)	K^-
(Del)	0.002167	0.8393	0.8841	1.0677	0.2758
(Ho)	-0.0002523	0.8355	0.9262	0.3041	0.4879
(SR)	-0.004836	0.7390	0.4085	0.4243	0.6152
(SR/1)	0.008024	0.9224	0.3136	0.7920	0.0990
(SR/2)	0.002040	0.9239	0.6244	0.6930	0.3182

method	up (V)	down	(VI)	delay 1 (VII)	delay 2	(VIII)
(Del)	0.1172	0.3377	0.5742	0.005928	-0.001413	49
(Ho)	0.9360	0.0288Δ	0.1371	-0.000238	0.004387	51
(SR)	0.3413	0.0142Δ	0.2494	0.01126	0.004205	51
(SR/1)	0.9936	0.9713	0.2494	0.006379	-0.000422	36
(SR/2)	0.2856	0.2853	0.3659	0.007801	-0.007048	44

１０．(SR)法の長周期化と(SR/4)

１０．１ (SR)法の長周期化

今まで，(SR)法の乱数特性を調べてきた。そして，連続発生した乱数の個数は高々 80000×10000=8×10^8 個程度であった。ここでは，周期が 10^{15} 程度の (SR)法乱数を発生する方法について考える。原理は至って単純で

命題　p, q, r, s が素数であるとき，$(rk \bmod p,\ sk \bmod q)$，$k=0, 1, 2, \ldots,$ の周期は pq である。

を利用するものである。

[演習] 命題に証明を与えよ。

以下では，p=49933453，q=22801201，r=491377，s=47513 とするので，

$$p \times q = 1,138,542,698,477,053 \approx 1.1 \times 10^{15}$$

となる。この長周期乱数においては，従来の再帰的乱数発生とは異なり，n番目の乱数値が直接計算されるという特徴がある（その代わり計算に時間がかかる）。

まず，SR$l(n,i)$，$0\le i<n$，で，$x_i=16+\frac{16}{n}\cdot i$ において，実数シフト計算（SR/l）が生成する4桁乱数を表すことにする((SR0)=(SR))。x_i は $[16,32)=[2^4,2^5)$ の範囲を動くが，このように設定したのは以下の状況を避けるためである：浮動小数点演算では x_i と $x_i\cdot 2^e$ は同じ仮数部を生成するため，(SR)計算では $f(x_i)$ と $f(x_i\cdot 2^e)$ は同一計算になってしまう。

次に，a，b を正の整数とする（以下では a=1920000，b=48060000ととる）。そして，関数

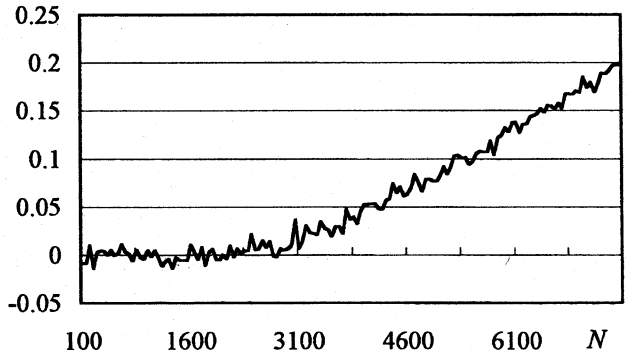
$$(r_k,s_k)\equiv(rk \bmod p,\ sk \bmod q) \longmapsto z(r_k,s_k)$$

を

$$z_k \equiv z(r_k,s_k)=\begin{cases}\mathrm{SR}l(a+s_k,\ r_k) & \text{if}\quad (a+s_k)>r_k\ ,\\ \mathrm{SR}l(b-s_k,\ r_k-(a+s_k)) & \text{if}\quad (a+s_k)\le r_k\ .\end{cases}$$

で定める。a，b として，(i) $a+2q<b$ および (ii) $a+b>p$ を満たすものをとれば，(i)より $a+s_k<b-s_k$ が常に成り立ち，また，(ii)より $b-s_k>r_k-(a+s_k)$ が保証されるので，平面上の点 $(a+s_k,r_k)$ あるいは $(b-s_k,r_k-(a+s_k))$，$k=0,1,\ldots,pq-1$，はすべて異なり，4桁整数列 z_0，z_1，z_2，…，の周期は (r_k,s_k) と同じ pq となる。一方，§7.2 で行われた議論から，一様乱数列としての SR$l(n,i)$，$i=0,1,\ldots,n-1$，は n が異なるとき，乱数列として互いに独立と考えられる。すなわち，SR$l(n,i)$ は i に関して独立，n に関しても独立ということで，数列 z_0，z_1，…，z_{pq-1} は一様乱数列になることが期待される。

乱数の周期 pq を大きくするためには，p（したがって b）を大きくとればよいが，(SR)計算で，単精度の仮数部に相当する23ビット b_2 から b_{24} までのみを使用しているため，あまり大きな b を選ぶと，$x_i=16+\frac{16}{b-s_k}\cdot i$ として，$f(x_i)$ と $f(x_{i+1})$ の違いが現れなくなり，むやみやたらに大きな値をとるわけにはいかない。



上の図は，x 軸の値 N に対して

$$x_{step} = \frac{1}{N\times 1000}$$

としたときの，乱数列

$$SR(x_i), \quad x_i = 22 + x_{step} \times i, \quad i=0,1,\cdots,19999,$$

に対する遅れ１の系列相関係数を計算した結果である。N=3000 あたりから正の相関が目立つことから，x_{step} の値は $1/(3\times10^6)$ 以下が望ましいことが分かる。区間 [16,32) の分割数に換算すると，$(3\times10^6)\times16 = 4.8\times10^7$ 分割となる。上の b の値はほぼこの値になっている。(SR)計算で，使用するビット数をさらに多くすれば x_{step} の間隔を狭めることができるので，b の値を大きくとることが可能となり，その結果 p も大きくとれ，乱数の周期を長くすることができる。また，a, b, p, q, r, s は色々変えることができ，それに対応して，異なる乱数列が生成されることになる。さらにこの方法は，従来の多くの乱数発生法と異なり，再帰的に乱数を計算せず，k番目の乱数を直接 $z(r_k, s_k)$ として計算するという特徴を持っている。ただ，直接計算するため，乱数発生に時間がかかるという欠点が生じる。

（**註**. 実際のプログラムでは，$SRl(n, i)$ のかわりに $SRl(n+3, i+1)$ を使う。これは，$SRl(n, i)$ で $i=0$ となるのを避けるためである。）

１０．２ 長周期化法の(SR/4)への適用

以上の長周期化の議論を，次に述べる「実数シフト計算IV」に適用してその乱数特性を調べてみよう。まず２つの基本計算を定義する。

【実数シフト計算IVa】 実数シフト法(SR)による $f(x_i)$ の計算結果 を f_i とする。

f_i : ［±］［$\cdots \times 2^0$］［\\\\\\\ 00000$\cdots\cdots$00］

$b_1 \cdots\cdots b_{23} \cdots\cdots\cdots\cdots b_{52}$

また，

$$b_e = b_6 + b_8 + \cdots + b_{20} \pmod 2 \qquad b_o = b_7 + b_9 + \cdots + b_{21} \pmod 2$$

とおく。このとき，

(i) 「$f_i < 1+\alpha$ または $f_i \geq 2-\alpha$」であって $b_e \neq b_o$ のとき，

もしくは

(ii) 「$f_i \geq 1+\alpha$ かつ $f_i < 2-\alpha$」 であって $b_e=b_o$ のとき,

f_i の仮数部 1〜23 ビットの全ビット値を反転する。

【実数シフト計算IVb】 「実数シフト計算IVa」における

(i) の ... $b_e \neq b_o$ ... と (ii) の ... $b_e=b_o$ ... を

入れ替えて計算したもの。

実数シフト計算 IVa と IVb では,

- f_i の仮数部の上位23ビットの値が互いに裏返しになってくる,
- $b_e=b_o$ (したがって $b_e \neq b_o$) となる確率は, ほぼ 1/2 であると思われる

ことに注意する。実際の計算では, 乱数の対称性を良くするため, さらにこの2つをミックスする。すなわち,

【実数シフト計算IVc】

$$b_{eo} = b_6+b_7+b_8+\cdots+b_{19}+b_{20}$$

とおく。 $b_{eo}<8$ なら「実数シフト計算IVa」を適用し, $b_{eo}\geq 8$ なら「実数シフト計算IVb」を適用する。

以下特に断らない限り, **【実数シフト計算IV】** は【実数シフト計算IVc】を指すものとし, これに前節の長周期化の手法を適用した乱数生成法を「実数シフト法IV(SR/4)」と呼ぶことにする。この方法により乱数特性が改善される理由は現時点でははっきりしないが, $b_e=b_o$ (もしくは $b_e \neq b_o$) の導入により確率≈1/2 で f_i の b_1 〜 b_{23} の全ビットが反転し特性が改善されるのに加え, さらに α を使って f_i がとる値の周辺で $b_e=b_o$ となる確率が微妙に異なる部分を引き出して改善に利用しているものと思われる。

p=49933453, q=22801201, r=491377, s=47513, a=1920000, b=48060000 のもとで α の最適値を調べるため, ここで §8.1 で使った検定を定式化しておく:

検定(IX) Kolmogorov-Smirnov 統計量のχ^2検定

80000 個の 4 桁整数乱数

$$z_{1+80000\times j},\quad z_{2+80000\times j},\quad \cdots,\quad z_{80000+80000\times j}$$

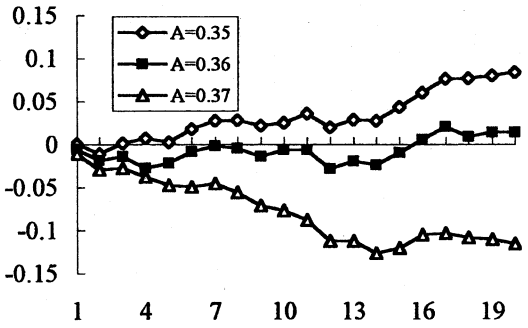
からなる乱数系列を1万系列(j=0, 1, ..., 9999)発生させる。次に各j系列ごとに,

Kolmogorov-Smirnov 検定値 $K^+(j)$ および $K^-(j)$ を計算する。この1万個の $K^+(j)[K^-(j)]$ の分布は，確率密度関数 $P(x)=1-\exp(-2x^2)$ 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。

乱数分布の非対称性を見るために，検定(IX)を1回行うごとに発生する $K^+(j)$, $K^-(j)$, $j=0,1,\ldots,9999$, に対して，

$$(7) \qquad D = \left[\#\{ j \mid K^+(j) \leq K^-(j) \} - \#\{ j \mid K^+(j) > K^-(j) \} \right] \Big/ 10000$$

とおく。検定(IX)を20回繰り返して D の列 D_1, D_2, D_3, ..., D_{20} を作り，これから $S_t = \sum_{l=1}^{t} D_l$, $t=1,\ldots,20$, を作る。 $\alpha=0.35$, 0.36, 0.37 としたときの S_t のグラフは以下のようになる。



S_t は 0 に近いことが望ましいので，以後 α として 0.36 を採用する。

$\alpha=0.36$ のとき，長周期化された (SR/4) が生成する乱数の最初の部分は次のようになる：

7604	5145	9073	0877	0248	7451	6046	5509	7850	2572
9460	7231	6762	2391	9731	2517	3520	2656	0799	6922
4672	6321	0395	6914	6197	7512	4687	2570	5221	····

我々は，「9．さらに詳しく乱数を検定する」において，20000個の乱数値に対して検定(II)から検定(VIII)まで7種類計10の検定を行った。まとめると

[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^+)
[検定IV] 乱数値の Kolmogorov-Smirnov 検定 (K^-)
[検定V] 単純上昇連テスト
[検定V] 単純下降連テスト
[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 の系列相関テスト
[検定VII] 遅れ 2 の系列相関テスト
[検定VIII] 衝突テスト

である。これらの検定を (SR/4) に適用すると,

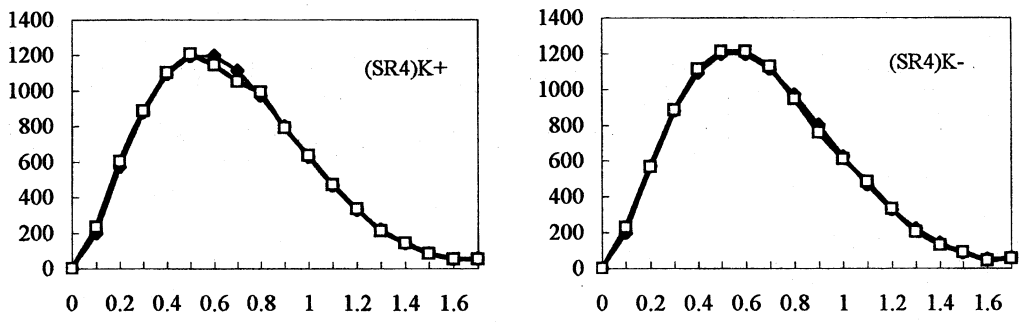
π相対誤差	検定(II)	(III)	K^+ (IV) K^-
0.001148	0.6030	0.5881	0.9900 0.3111

up (V) down	(VI)	delay 1 (VII) delay 2	(VIII)
0.0808 0.1280	0.4333	-0.00192 -0.00525	44

となり，いずれの検定でも一様乱数であるという仮説は棄却されない。また

[検定(IX)] KS統計量 $K^+[K^-]$ に対するχ^2検定

については，検定値 0.3784 [0.6024] を得る（下図参照）。



以上の結果から，長周期化の手法は大変有効に機能していることが分かる。

１１．(SR/4)の特性および他乱数との比較

この章では，[検定(II)]-[検定(IX)] を繰り返し行うことにより，(SR/4) の特性を，よく知られている既存の乱数生成法および物理乱数と比べてみることにする。比較する乱数生成法は，

(LC) 線形合同法,

(Ms) M系列(Maximum-length linearly recurring sequence),

(MT) MT(Mersenne Twister)法,

(Ph) 統計数理研究所の物理乱数

(BC) Borland C++ V.55 に付属する乱数関数

(SR4) (SR/4)法で0番目の乱数値から始めたもの

(SR4+) (SR/4)法で987654321番目の乱数値から始めたもの

である。また，使用した言語ソフトは Borland C++ V.55 である。

11.1 各乱数生成法の概略

まず最初に(SR/4)以外の各乱数生成法の概略を示す。

(LC)線形合同法

$$X_0 = 987654321, \quad X_n = 1664525 * X_{n-1} + 1013904223 \ (\mathrm{mod}\ 2^{32}),$$

により，X_1, X_2, ... を発生し([5])，$X_n/429496.7296$ の整数部分を取り出して10 進 4 桁乱数を得る。

(Ms)M系列

$$Y_n = Y_{n-32} \ \mathrm{XOR}\ Y_{n-521} \quad (\text{32-bit integer}), \quad n \geq 521,$$

により，Y_{521}, Y_{522}, ... を発生し([1])，$Y_n/429496.7296$ の整数部分を取り出して 10 進4桁乱数を得る。初期値 Y_0, Y_1, ..., Y_{520} は，上述(LC)の $X_1,\ldots,X_{521}$ の MSB(最上位ビット) $a_1,\ldots,a_{521}$ から

$$a_t = a_{t-32} + a_{t-521} \ (\mathrm{mod}\ 2), \quad t = 522,\ldots,16672,$$

により定める。

(MT)MT法

Matsumoto, M. and Nishimura, T. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. ACM Trans. on Modeling and Computer Simulation. Vol.8.1(1998), No.1, pp.3-30,

に述べられている乱数生成法である。

http://www.math.keio.ac.jp/~matsumoto/mt.html

よりダウンロードした C プログラムにより，32ビット一様乱数を得，それを 429496.7296 で割り，整数部分を取り出して10進4桁乱数とした(seed=4357)。MT法は，

$$x_{k+n} = x_{k+m} \text{ XOR } (x_k^{upper} \text{ OR } x_{k+1}^{lower})A$$

により x_i を計算し，x_iT を乱数値として出力している。天文学的周期 $2^{19937}-1$ を持っている。

(Ph)**物理乱数**

統計数理研究所のコンピュータ HITACHI SR8000 から，物理乱数を31ビット整数値で取得し，これを 214748.3648 で割ってその整数部分をとり，4桁乱数とした。物理乱数は使い捨てであり，乱数値の再現性はない。

(BC)**BC++ V.55**

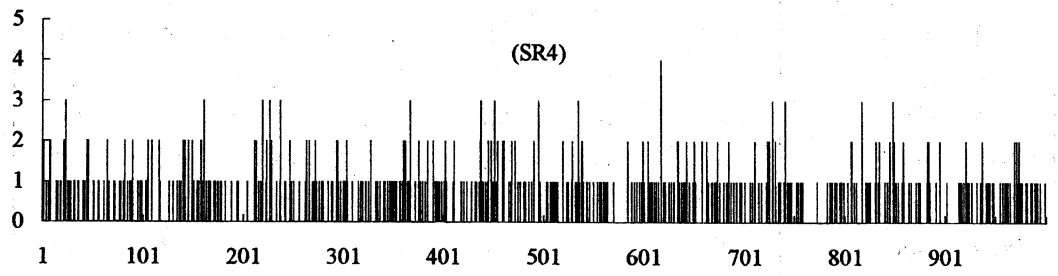
BC++ V.55 は Borland(Inprise社)の提供する32ビット版フリーコンパイラである。random(10000) により，4桁整数乱数を得た(seed=987654321)。乱数生成速度の速さからみて，random() はアセンブラで書かれているようである。同じ random() でも(有料の) BC++ V.50 とは出てくる乱数が違うので注意が必要である。

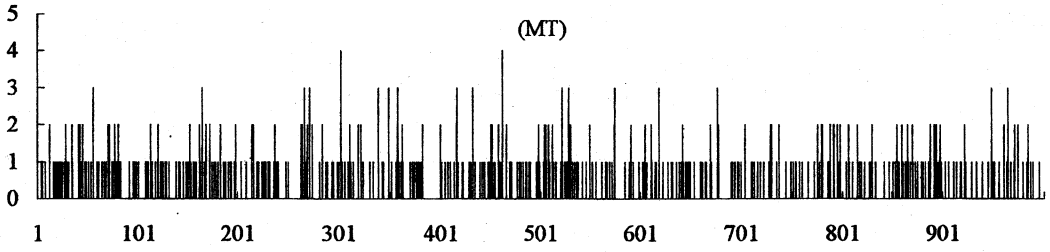
11.2 テスト結果

10.2節でまとめた［検定(II)］から［検定(VIII)］までの7種10検定を，(SR4)に対して適用し，危険率 0.05 で仮説が棄却される回数 $c(0 \le c \le 10)$ を数える。これを 1000 回繰り返せば，棄却回数の列

$$c_1=0, \quad c_2=2, \quad c_3=0, \quad \ldots, \quad c_{1000}=1$$

が得られ，そのグラフは以下のようになる。また，比較のため(MT)についてのグラフも示しておく。





高さ 4 のデータを取り除けば，一目では区別が付かない状況である。各乱数生成法における c の分布は

c の値 ⇒	0	1	2	≥3	CHITEST
(LC)	629	283	69	19	0.0176
(Ms)	606	301	80	13	0.7288
(MT)	597	303	82	18	0.1818
(Ph)	614	310	68	8	0.5460
(BC)	610	305	78	7	0.4840
(SR4)	592	314	79	15	0.7060
(SR4+)	650	256	77	17	0.0004
$Bin(10, 0.05)$	0.5987	0.3151	0.0746	0.0115	

である。各検定が危険率 0.05 で独立に行われると仮定すれば（厳密に言えば，正確な仮定とは言えないが），c の分布は二項分布 $Bin(10, 0.05)$ になる。これについて CHITEST を行なった結果が上表右側である。(SR4+)では，c=1 となる回数が少なすぎて検定ではじかれている。

また，1000回の繰り返しにおける，各検定の棄却回数の内訳は次のようになる：

検定	(LC)	(Ms)	(MT)	(Ph)	(BC)	(SR4)	(SR4+)
(II) 0-9 char	55	48	56	51	53	62	55
(III) 0-intv	40	54	48	40	56	55	44
(IV) K^+	41	62	47	58	46	44	46
(IV) K^-	40	46	50	44	46	47	50
(V) up	48	44	63	44	46	46	57
(V) down	52	42	52	36	46	58	38
(VI) poker	51	55	71	44	57	58	40
(VII) delay 1	48	54	46	47	45	49	50

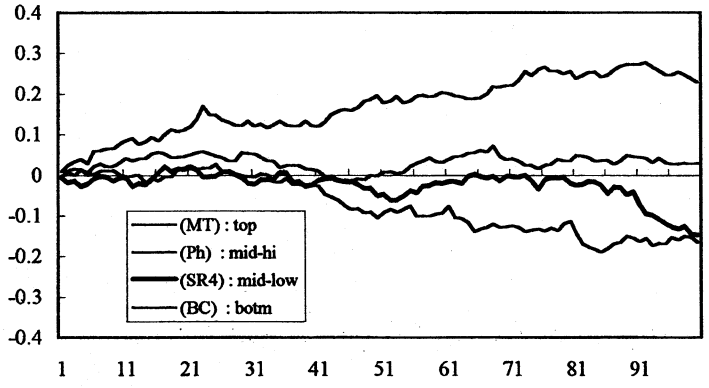
(VII) delay 2	61	47	35	53	40	49	43
(VIII) colli	42	50	55	54	47	50	42
棄却回数計	478	502	523	471	482	518	465

各検定においては危険率を 0.05 としているので，各乱数生成法とも妥当な値を出していると言えよう。

[検定IX]を各乱数生成法について100回繰り返したとき，危険率 0.05 で仮説が棄却された回数は以下の通りである。

CHITEST of ⇓	(Ms)	(MT)	(Ph)	(BC)	(SR4)
K^+	73	40	29	39	36
K^-	62	31	29	35	38

全体的に棄却数が多いのが気になるが，(Ms)を除いてほぼ横並びというところである。また，[検定IX] に付随して発生する 100個の D_1, D_2, ..., D_{100} (D の定義は §10.2 の 式(7)) について， $S_t = \sum_{l=1}^{t} D_l$ のグラフは次のようになり，物理乱数 (Ph) の安定性が目立つ。



11.3 乱数生成速度について

乱数を 8×10^8 個発生させる [検定IX] の所要時間は，インテルの Celeron (466MHz) を内蔵する PC で，

	(LC)	(Ms)	(MT)	(BC)	(SR4)
sec.	366	376	423	136	4170

であった。統計数理研究所の HITACHI SR8000 の物理乱数との比較は，

	(Ph)	(SR4)
sec.	3264	7312

であった。これから，

$$(LC) : (Ms) : (MT) : (Ph) : (BC) : (SR4)$$
$$\fallingdotseq 0.87 : 0.89 : 1.00 : 4.40 : 0.32 : 9.86$$

を得る。物理乱数の発生がかなり高速であることが分かるとともに，(SR4)法の遅さが目立つ。(SR4)法では乱数値を（再帰的ではなく）直接計算していることを考慮すれば，止むを得ないところであろう。

以上のように，(SR/4)に対する結果は，生成速度が少々遅い点を除けば，他の乱数と比して大きく破綻することもなく，(SR/4)は他の擬似乱数と同程度の性質は持っているものと考えられる。

１２．ランダムウォークテストと DIEHARD テスト

１２．１ ランダムウォークテスト

以下で行うランダムウォークテストは，多量の乱数を継続して発生する場合の乱数特性を調べるのに有効である([6])。(SR)法の改良において(SR/3) が省かれているのは，このランダムウォークテストおよび[検定IX]で少々さえないところがあったためである。(SR/4)では，もちろんこのことは改良されている。

最初に，検定に使用する，ランダムウォークに付随する確率変数を簡単にまとめておく。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: $r>0$ に対し

$$FP_r = \min\{\ k\ :\ S_k = r\ \}$$

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，実際のデータと照らし合わせてχ^2検定を行うことができる。

[演習] 上にあげた各確率変数について調べよ。

テストは，パス(1行程の歩数)の長さ 200， パスの本数 50,000 についての分布を調べ，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}^{-} を行い棄却されるかどうか調べる。これを 100 回繰り返して棄却された回数を数える。（このテストにあたっては [6] を参照し，また，岡山理科大・高嶋氏が作成したプログラムのソースコードを使わせていただいた。）

テストの入力データは，

(Ms),(MT)においては，32ビット整数乱数のMSB(最上位ビット)を使用した;

(Ph)においては，31ビット整数乱数のMSB(最上位ビット)を使用した ;

(SR4)においては，(SR/4) の4桁の10進乱数値が5000未満のときは -1,
5000以上のときは 1 とした。

ただし，MSB使用時には，MSB = 0[1] に対応して $X_k = -1[+1]$ とする。以下に結果をまとめる。

	K_{30}^{+} 95~99%	99%~	K_{30}^{-} 95~99%	99%~	total
First passage time test					
(Ms)	1	1	4	0	6
(MT)	7	0	2	1	10
(Ph)	3	1	4	1	9
(SR4)	3	2	4	0	9
Hamming weight test					
(Ms)	4	0	5	1	10
(MT)	5	2	4	1	12

(Ph)	4	3	3	2	12
(SR4)	1	0	3	2	6
Last visit time test					
(Ms)	3	2	4	2	11
(MT)	5	0	6	1	12
(Ph)	7	0	3	1	11
(SR4)	2	1	6	1	10
Maximun test					
(Ms)	3	0	3	1	7
(MT)	6	0	4	1	11
(Ph)	8	1	2	1	12
(SR4)	5	2	4	1	12
Sojourn time test					
(Ms)	5	2	4	2	13
(MT)	6	3	6	0	15
(Ph)	3	1	0	1	5
(SR4)	6	3	4	1	14

また，参考のため棄却回数の総数もまとめておく：

	K_{30}^{+}	K_{30}^{-}	total
(Ms)	21/500	26/500	47/1000
(MT)	34/500	26/500	60/1000
(Ph)	31/500	18/500	49/1000
(SR4)	25/500	26/500	51/1000

total がうまい具合にまとまるものである。((Ms)については [6] の結果も参照のこと。)

12.2 (SR/4) の DIEHARD テスト

乱数検定の一連のプログラムとして，Marsaglia の DIEHARD battery of tests (http://stat.fsu.edu/~geo/diehard.html) が広く知られている(以下 DIEHARD テストと略す)。使用したバージョンは DOS, Jan 7, 1997 であり，テ

ストは DIEHARD battery の全項目について行なった。DIEHARD テストはテスト項目が多く，ある乱数生成法の特定乱数列が一度の試行でその全てのテストにパスすることはまずない。パスするとすれば，何回かの試行を繰り返す時に，偶然通るということになるであろう。(SR/4)は 1325400064(=0x4f000000) 番目の乱数値から始めると，以下のテスト項目で良い成績を出すので，これについて結果をまとめておく(表の(SR4~))。結果は DIEHARDテストの p-value をそのまま載せている。(註. p-value は，通常の検定のように，0 または 1 に極端に近くなければ良いとされている。) また，比較のため (SR4)の 0 番目の乱数値から始めたもの ((SR4) と記す)，および，物理乱数((Ph)と記す)についての結果も示す。なお，DIEHARDテストのための入力データは，

(SR4~),(SR4)においては，(SR/4) のビット b_{13} , ... , b_{20} を使用し，
(Ph)においては，31ビット乱数の下16ビットを使用した。

	(SR4~)	(SR4)	(Ph)
BIRTHDAY SPACINGS TEST	.399888	.039789Δ	.978813Δ
OVERLAPPING 5-PERMUTATION TEST	.726745	.336998	.645826
	.510901	.999756Δ	.787156
BINARY RANK TEST for 31x31 matrices	.673389	.368053	.958216Δ
BINARY RANK TEST for 32x32 matrices	.341111	.397228	.379871
BINARY RANK TEST for 6x8 matrices	.424825	.445135	.119085
COUNT-THE-1's TEST on a stream	.590451	.597031	.709226
	.788174	.175226	.697456
PARKING LOT TEST	.542633	.047857Δ	.860631
MINIMUM DISTANCE TEST	.821627	.219076	.769942
3DSPHERES TEST	.589579	.321632	.675080
SQUEEZE test	.385524	.773681	.921651
OVERLAPPING SUMS test	.832807	.207408	.815915
RUNS test (up)	.196048	.215586	.367097
(down)	.235305	.408405	.524058
(up)	.739171	.613116	.930633
(down)	.374973	.470105	.666390
CRAPS TEST	.311588	.817026	.746197
	.296983	.148324	.320808

各テストの内容については，DIEHARDテストの付属テキスト あるいは [2] を参照のこと。

１２．３ (SR/4) の長所・短所

最後に，(SR/4) の長所・短所を思いつくままにまとめておく。

◎ (SR/4) は n 番目の乱数値を直接生成するので，乱数列の取り扱いが容易である。例えば，あらかじめ１回の計算処理で消費する乱数の数を調べておけば，１つのコンピュータで直列的に何回か繰り返す計算処理を，乱数生成開始位置をずらして指定することにより，いくつかのコンピュータ（あるいはジョブ）で並列的に同時実行することができる。同様に，いったん計算終了後さらに計算処理を追加することも簡単に行える。

◎ (SR/4) は，パラメータとして p=49933453, q=22801201, r=491377, s=47513, a=1920000, b=48060000, α=0.36 を使っているが，前にも触れたように，これらを適当に変えれば，別の乱数系列が得られる。例えば，複数の r, s を準備しておいて，それらを組み合わせて使えば，同じ周期 pq をもつ多くの乱数系列を作ることができる。この意味で，(SR/4)は，複数のCPUを同時に動かす並列処理にも適していると言えよう。

［例］ r=81899, s=7919 と，これまでの r=491377, s=47513 とを組み合わせて，4系列の(SR/4)乱数列を作る。これらについて，［検定(IX)］を100回行ったとき棄却される回数は以下のようになる(§11.2 参照)。

K^+		r	
		491377	81899
s	47513	36	26
	7919	24	28

K^-		r	
		491377	81899
s	47513	38	31
	7919	32	38

また，r=81899, s=7919 としたとき， 6140461056(=0x16e000000) 番目の乱数値から始めると，上記のDIEHARDテスト項目でよい結果が得られる。

◎ (SR/4) は，従来の擬似乱数とはかなり異なる方法で乱数を生成するので，いろいろな性格をもつ乱数についての計算結果を比較したい場合，いま１つの乱数列として使うことができる。

△ (SR/4) は，乱数生成速度が1桁ほど遅い。他の乱数生成法と違って，n 番目の乱数値を直接生成するので止むを得ない面でもある。（口悪くいえば，(SR/4) は巨大な電子式乱数表ということになる。）

△ (SR)法についての理論的検討がなされていない。理論的解析がなされれば，乱数生成法としての性格がはっきりし，また改良点・弱点も見えてくる。

△ (SR)法は浮動小数点演算により乱数を生成するが，浮動小数点演算は計算機のアーキテクチャの違いにより微妙に異なる。このため(SR/4)に関する統計結果等は，計算機によって多少異なることがある（可搬性(ポータビリティ)の問題）。64ビットの整数計算ができる計算機では，(SR)法で使う浮動小数点計算を整数演算のみでエミュレートすることができるので，この問題を避けることができる。64ビット整数演算では，乱数生成が速くなる計算機(≈高級品!)もあり，遅くなる計算機もある。

Appendix

以下は，周期 1,138,542,698,477,053 をもつ (SR/4) 乱数生成プログラムである。*nn* 番目の乱数から生成するには，SRIni((double)*nn*.0) と指定する。指定がない場合は 0 番目の乱数から生成する。SR4Dg4() で10進4桁の乱数を得，SR4Byt() で８ビットの乱数を得る。プログラム作成に使用したソフトウェアは Borland C++ 5.5(フリーソフト) である。なお，このプログラムでは，本文中の SR*l*(*n*, *i*) のかわりに SR*l*(*n*+3, *i*+1) を使っている。これは，SR*l*(*n*, *i*) で *i*=0 となるのを避けるためである。また，実行速度を重視したプログラムでないことをお断りしておく。（浮動小数点演算に関する計算機依存性については，§7.1の註参照。）

```
#include <stdio.h>
#include <math.h>
/*=======Select either of the following typedef ================*/
    typedef struct {int L; int H;} intLH;        /* Windows only */
 /* typedef struct {int H; int L;} intLH; */  /* In some systems */
/*==============================================================*/
    typedef union  {double Dbl; intLH LH;} DblI2;
    typedef struct {int p; int q; int r; int s;
                    int a; int b; int rk; int sk;} SRlpPara;
    SRlpPara SRPara = {49933453, 22801201, 491377, 47513,
                       1920000, 48060000, 0, 0};
    double SR4alpha=0.36;
```

```
double SR(double x)
    {
    int    i;
    Dbl I2  FLLH;
    FLLH.Dbl=1;
    for (i=1; i<=24; i++)
        {
        FLLH.Dbl=FLLH.Dbl*x/i;
        FLLH.LH.H=(FLLH.LH.H & 0x0007ffff) << 1;    /*mantissa H*/
        if (FLLH.LH.L & 0x80000000) {FLLH.LH.H++;}
        FLLH.LH.L=(FLLH.LH.L & 0xf0000000) << 1;    /*mantissa L*/
        FLLH.LH.H |= 0x3ff00000;                    /*set exponent 2^0*/
        }
    return(FLLH.Dbl);
    }
double SR4(double x)
    {
    int    i, B=1, b_e=0, b_o=0, b_eo=0;
    Dbl I2  FLLH;
    FLLH.Dbl=SR(x);
    if (FLLH.LH.L & 0x80000000) {b_o++;}
    for (i=1; i<=15; i++)
        {
        if (FLLH.LH.H & B)
            {
            if (i & 1) {b_e++;} else {b_o++;}
            b_eo++;
            }
        B+=B;
        }
    b_e&=1; b_o&=1;
    if ((FLLH.Dbl<1+SR4alpha) || (FLLH.Dbl>=2-SR4alpha))
        {
        if (b_eo<8)
            {if (b_o != b_e)
                {FLLH.LH.L^=0xe0000000; FLLH.LH.H^=0x000fffff;}}
          else
            {if (b_o == b_e)
                {FLLH.LH.L^=0xe0000000; FLLH.LH.H^=0x000fffff;}}
        }
      else
        {
        if (b_eo<8)
            {if (b_o == b_e)
                {FLLH.LH.L^=0xe0000000; FLLH.LH.H^=0x000fffff;}}
          else
            {if (b_o != b_e)
                {FLLH.LH.L^=0xe0000000; FLLH.LH.H^=0x000fffff;}}
        }
    return(FLLH.Dbl);
    }
double nextX(void)
    {
    int  n, i;
    double xstep, x;
    SRPara.rk+=SRPara.r;
    if (SRPara.rk>=SRPara.p) {SRPara.rk-=SRPara.p;}
    SRPara.sk+=SRPara.s;
```

```
    if (SRPara.sk>=SRPara.q) {SRPara.sk-=SRPara.q;}
    n=SRPara.a+SRPara.sk;
    if (SRPara.rk<n)
        {i=SRPara.rk;}
      else
        {i=SRPara.rk-n; n=SRPara.b-SRPara.sk;}
    n+=3;  i++;
    xstep=16.0/n;          /* store in memory */
    x=16+xstep*i;          /* store in memory */
    return(x);
    }
void SRIni(double n)
    {
    double k, rkD, skD;
    if (n<0) {k=-n;} else {k=n;}
    rkD=SRPara.r*(k-floor(k/SRPara.p)*SRPara.p);
    rkD=rkD-floor(rkD/SRPara.p)*SRPara.p;
    skD=SRPara.s*(k-floor(k/SRPara.q)*SRPara.q);
    skD=skD-floor(skD/SRPara.q)*SRPara.q;
    SRPara.rk=(int)rkD;
    SRPara.sk=(int)skD;
    if (n<0)
        {
        if (SRPara.rk) {SRPara.rk=SRPara.p-SRPara.rk;}
        if (SRPara.sk) {SRPara.sk=SRPara.q-SRPara.sk;}
        }
    }
int SR4Byt(void)
    {
    double x;
    Dbl12  FLLH;
    x=nextX();                      /* store in memory */
    FLLH.Dbl=SR4(x);                /* store in memory */
    return(FLLH.LH.H &= 0xff);
    }
int SR4Dg4(void)
    {
    double x, y;
    x=nextX();                      /* store in memory */
    y=SR4(x);                       /* store in memory */
    return((int)(floor(y*1000000)) % 10000);
    }
/*****************/
int main() /* show 1,138,542,698,477,053 random numbers all! */
    {
    int   i, j, SRrand;
    SRIni((double)0.0);
    /*for (i=0; i<49933453; i++)*/
        /*for (j=0; j<22801201; j++)*/
            {
            SRrand=SR4Dg4();   /* get a 4-digits random number */
            printf("%04d¥n", SRrand);
            }
    return(0);
    }
```

学科　　番号　　氏名　　

乱数プログラム名：**ranrep.exe**　　乱数発生開始番号

検定(I) 予備評価

$x^2+y^2<1$ を満たす(x,y)の数は10000個のうち　　個。

πの近似値 =　　相対誤差 =

(註：π＝3.14159265358979323846264338327950288...)

検定(II) 文字0～9の出現頻度のχ^2検定 (一様性の検定)

文字0	1	2	3	4	5	6	7	8	9

[EXCEL(or相当するもの) の CHITEST 関数の答] =

仮説は棄却 (される　　されない) (以下，有意水準は 0.05とする。)

検定(III) 文字0の出現間隔のχ^2検定 (独立性の検定)

発生した2万個の数値には　　個 の文字 0 がある。

0-0間隔長	0	1	2	3	4	5	6	7	8
回数 理論回数									

9	10	11	12	13	14	15	16	17	18

19	20	21	22	23	24	25	16	27	28

29	30	31	32	33	34	35	36	37	38以上

(註：0 が 間隔k で発生する確率は $(1-p)^k \cdot p$，k以上で発生する確率は $(1-p)^k$)

[EXCEL の CHITEST 関数の答] =

仮説は棄却 (される　　されない)

検定(IV) Kolmogorov-Smirnov 検定 (一様性の検定)

K^+ = 　　仮説は棄却 (される　　されない)

K^- = 　　仮説は棄却 (される　　されない)

(註：n=20000のとき，KS統計量の危険率 0.05 に対応する値の近似値は1.2239)

検定(V) 単純下降連[上昇連]テスト(一様性・独立性の検定)

上昇連長	1	2	3	4	5以上	計
回数 理論回数						

[EXCEL の CHITEST 関数の答] =

仮説は棄却 (される　　されない)

下降連長	1	2	3	4	5以上	計
回数 理論回数						

[EXCEL の CHITEST 関数の答] =

仮説は棄却 (される　　されない)

(註：長さt以上の連が現れる確率は $\frac{1}{t!}$，長さrの連が現れる確率は $\frac{1}{r!} - \frac{1}{(r+1)!}$)

検定(VI) 4枚の 0～9 カードによる古典ポーカーテスト(一様性・独立性の検定)

	回数	理論回数
All different		10080
One pair		8640
Three of a kind		720
Two pairs		540
Four of a kind		20

[EXCEL の CHITEST 関数の答] = [　　]

仮説は棄却 (される　　されない)

検定(VII) 遅れ k の系列相関テスト(独立性の検定)

遅れ1の系列相関係数=[　　]　　仮説は棄却 (される　　されない)

遅れ2の系列相関係数=[　　]　　仮説は棄却 (される　　されない)

(註 : n=20000 のとき, ほぼ 95% の確率で [-0.01419, 0.01409] にある。)

検定(VIII) 衝突テスト(一様性・独立性の検定)

衝突回数 = [　　]　　仮説は棄却 (される　　されない)

【 χ^2検定の行い方 】(MS-EXCELの場合)

実測値, 期待値のデータを作成しておく :

実測値 ⇓	期待値 ⇓
1973	2000
1984	2000
2019	2000
2030	2000
1957	2000
2034	2000
1985	2000
2017	2000
2014	2000
1987	2000

必要な実測値データは, ranrep.exe があるディレクトリに, ファイル

0_9chars.txt
0intvals.txt
runup.txt
rundown.txt
pokertst.txt

としてあるので, EXCEL で読みこむ。

ついで, カーソルを計算結果を保存するセルに移し, そこで

挿入 --> 関数 --> 統計 --> CHITEST

とクリックで進み, 「OK」ボタンを押す。

実測値範囲　　期待値範囲

の問い合わせがあるので, 各範囲を A1:A10 のように指定し「OK」ボタンを押す。計算結果が, 有意水準(危険率)以上(0.05以上, 0.01以上等)であれば, 仮説は棄却されない。上の例では, CHITEST の計算結果は 0.9617 となる。

【終りに】以上の内容は，sin曲線の乱れから始めて，観察・思考・実験の繰り返しとして構成されたものであった。この意味で，自然現象に対する人間[科学]のかかわり方の１つの小さなモデルともなっている。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。≈≈内容についてお気づきの点などありましたら，是非ご教示下さい≈≈

参 考 文 献

[1] 伏見正則 : 乱数. 東京大学出版会. 1989.

[2] Gentle, E. : *Random number generation and Monte Carlo methods.* Springer-Verlag, 1998.

[3] Hummel, R. L. : The processor and coprocessor. Ziff-Davis Press, 1992. (80x86/80x87ファミリー・テクニカルハンドブック. 槌田浩一訳, 技術評論社, 1993.)

[4] Knuth, D.E. : *The art of computer programming.* Vol.2(3rd ed.). Addison-Wesley, 1997.

[5] Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P. : *Numerical recipes in C.* 2nd ed. Cambridge University Press, 1992.

[6] Takashima, K. : Hybrid pseudo-random number generation. Monte Carlo Methods and Appl., **6**(2000), 49-59.

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

[8] Yaguchi, H.: Construction of a long-period nonalgebraic and nonrecursive pseudorandom number generator.

Rokko Lectures in Mathematics

1. Brown 運動とその周辺
   河野 敬雄 著 (1995) ISBN: 4-907719-01-9
2. Eisenstein 級数と概均質ベクトル空間のゼータ関数
   佐藤 文広 著 (1996) ISBN: 4-907719-02-7
3. Lecture Notes on Interacting Particle Systems
   今野 紀雄 著 (1997) ISBN: 4-907719-03-5
4. p 進体上の簡約代数群の admissible 表現論入門
   高橋 哲也 著 (1998) ISBN: 4-907719-04-3
5. Projective Differential Geometry and Linear Differential Equations
   佐々木 武 著 (1999) ISBN: 4-907719-05-1
6. ウィーナー空間
   佐藤 坦 著 (1999) ISBN: 4-907719-06-X
7. パンルヴェ方程式の眺望
   高野 恭一　野海 正俊 編 (2000) ISBN: 4-907719-07-8
8. 数値計算誤差と乱数生成
   谷口 礼偉 著 (2001) ISBN: 4-907719-08-6
9. 計算機代数
   野呂 正行 著 (2001) ISBN: 4-907719-09-4

Available on http://www.math.kobe-u.ac.jp/publications