科学研究費助成事業（学術研究助成基金助成金）
基盤研究(C)　平成23年度～平成25年度　課題番号 23500086
研 究 課 題　非再帰型擬似乱数生成アルゴリズムを応用したハッシュ関数の研究
研究代表者　谷口 礼偉

研究報告資料

新しい非再帰型擬似乱数生成法とハッシュ関数への応用

平成 26 年 3 月

まえがき

本報告は，新しい非再帰的な擬似乱数生成法とそのハッシュ関数への応用を自己包括的にまとめたものである。新しい乱数生成法は，一般教育の講義「数値計算と統計処理」の講義準備中にたまたま遭遇した計算誤差による"不思議な現象"が起源であり，全くゼロから始まった研究である。研究を続けているうちに，乱数生成の数学的背景が少しずつ明らかになってきて，その本質は，本書第２章にあるように，エルゴード理論の分野で扱われているβ変換（さらに詳しくいえば linear mod one 変換）であることが分かった。乱数生成の数学的背景が明らかになったことにより，この研究がさらに発展する可能性が高まったと考えている。本年度は科学研究費による研究の最終年度にあたるので，この機会に研究を総合的に記録しておくべく，研究の発端から現時点までの状況を網羅的にまとめたものが本書である。内容は，新しい乱数生成法の発端からアルゴリズムの構築と乱数性の検証までが第１章となり，β変換に基づく第１章のアルゴリズムの数式的扱いとそれに基づくハッシュ関数の構成および安全性の解析，さらに６４ビット擬似乱数の構成が第２章となっている。第１章は今までにまとめたものがベースになっており，第２章は新しい内容である。この研究記録が今後少しでも役に立つことがあればと願っている。

２０１４年3月　三重大学教育学部　谷口礼偉

目　次

第１章　新しい非再帰型擬似乱数生成法とその応用

第２章　[1,2)上のβ変換を利用した乱数の生成とその応用

Appendix

Construction and security of nonalgebraic hash functions based on β -Transformations on [1,2)

第 1 章

新しい非再帰型擬似乱数生成法とその応用

第１章　新しい非再帰型擬似乱数生成法とその応用

１．新しい非再帰型擬似乱数生成法の起源

右下図は sin x を

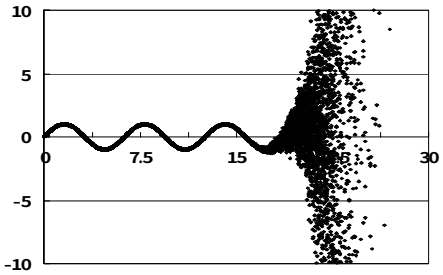
$$\sin x \approx x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\frac{x^9}{9!}\cdots+(-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!} \quad (n=30)$$

と近似し，区間 [0,30] を 19999 等分した各点において

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

$k=2,\cdots,30,$ と単精度計算し，

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



を求めて図示したものである。

x=16.5 前後からグラフの乱れが顕著に観察される。このグラフの乱れについての数理的解釈が新しい乱数生成法の出発点である。

グラフの乱れが, コンピュータにおける数値計算の誤差に起因するものであることは容易に想像されるので，さらに詳しく x=22 での，途中での計算値 F_i と，最後の sin x の計算値をみてみると，

	単精度計算	倍精度計算
F1	22.000000000000000	22.000000000000000
F2	-1774.666625976560000	-1774.666666666660000
F3	42946.933593750000000	42946.933333333300000
F4	-494912.281250000000000	-494912.279365079000000
F5	3326910.250000000000000	3326910.322398580000000
F6	-14638405.000000000000000	-14638405.418553700000000
F7	45416588.000000000000000	45416591.170384800000000
F8	-104674424.000000000000000	-104674429.173648000000000
F9	186258896.000000000000000	186258910.735463000000000
F10	-263594464.000000000000000	-263594481.859544000000000
F11	303761248.000000000000000	303761260.047665000000000
F12	-290554240.000000000000000	-290554248.741245000000000
F13	234380416.000000000000000	234380427.317938000000000
F14	-161595616.000000000000000	-161595622.253393000000000
F15	96320536.000000000000000	96320543.313598900000000
F16	-50128108.000000000000000	-50128110.713743900000000
F17	22975382.000000000000000	22975384.077132600000000
F18	-9344609.000000000000000	-9344609.994396800000000
F19	3395488.500000000000000	3395488.916882920000000
F20	-1108918.000000000000000	-1108918.107807910000000
F21	327266.031250000000000	327266.075718922000000
F22	-87705.843750000000000	-87705.858609057700000
F23	21439.207031250000000	21439.209882214100000
F24	-4799.526367187500000	-4799.527096665880000
F25	987.657653808593000	987.657786898931000
F26	-187.461288452148000	-187.461321121208000
F27	32.921360015869100	32.921364086598300
F28	-5.364962577819820	-5.364963036334550
F29	0.813484311103820	0.813484370171028
F30	-0.115057393908500	-0.115057403612734
sin x	-16.180646896362300	-0.022378806339483

となっている。上表の数値は，C 言語で計算した値を Excel で表示したものである。途中項 F_i の値は，単精度・倍精度とも最初の６～７桁は同じであるが，F_i の総和をとった $\sin x$ の値は大きく異なっている。

単精度での有効桁数は 6～7 桁程度，倍精度の有効桁数は 15～16 桁程度，Excel での有効桁数は 15 桁であるとされているので，単精度，倍精度の数値が表す意味（有効性）から考えなくてはならない。

一方，

$$\sin x \approx x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\frac{x^9}{9!}\cdots+(-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!} \qquad (n=30)$$

$$= x\left(1-\frac{x^2}{2\cdot 3}\left(\cdots\left(1-\frac{x^2}{(2n-4)(2n-3)}\left(1-\frac{x^2}{(2n-2)(2n-1)}\right)\right)\cdots\right)\right)$$

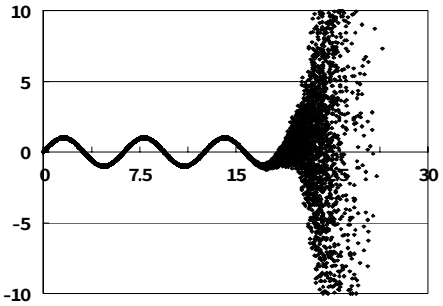
G_1 ⟶

⟵ G_2 ⟶

と変形し，

$$G_0=1, \qquad G_k=1-\frac{x^2}{(2n-2k)(2n-2k+1)}\times G_{k-1}, \qquad G_{30}=x\cdot G_{29}$$

と計算する方法（ホーナー法）は，計算誤差が生じにくい方法とされているが，実際に区間 [0,30] を 19999 等分した各点において単精度計算し図示してみると，



	単精度計算	倍精度計算
G1	0.858562231063842000	0.858562244301578000
G2	0.869817018508911000	0.869817003057028000
G3	0.858252048492431000	0.858252043946261000
G4	0.849276483058929000	0.849276491556607000
G5	0.838804006576538000	0.838803991406510000
G6	0.827388942241668000	0.827388974557503000
G7	0.814775109291076000	0.814775086176765000
G8	0.800832748413085000	0.800832756712346000
G9	0.785380363464355000	0.785380368633014000
G10	0.768217027187347000	0.768217013159524000
G11	0.749111294746398000	0.749111312841288000
G12	0.727800428867340000	0.727800393832444000
G13	0.703987061977386000	0.703987066710165000
G14	0.677339255809783000	0.677339261091174000
G15	0.647492229938507000	0.647492255518141000
G16	0.614056348800659000	0.614056340306920000
G17	0.576633512973785000	0.576633520358191000
G18	0.534848988056182000	0.534848960244392000
G19	0.488405317068099000	0.488405342374929000
G20	0.437171012163162000	0.437170986406034000
G21	0.381313532590866000	0.381313574793799000
G22	0.321486204862594000	0.321486138969856000
G23	0.259050846099853000	0.259050993993283000
G24	0.196278139948844000	0.196276853028880000
G25	0.136376187205314000	0.136378184667288000
G26	0.083248965442180600	0.083235536403225500
G27	0.040654778480529700	0.040809532877115400
G28	0.016154360026121100	0.012409304373806700
G29	-0.303118377923965000	-0.001017219487076060
sinx	-6.668604373931880000	-0.022378828715673400

となり，やはり正確に計算されない。x=22 での計算値をみてみると（上表），計算の最後の方で単精

度，倍精度の計算値が急激に異なってくることが観察される。

これらの現象を正確に解釈しようと思えば, コンピュータで数値データがどのように扱われているかを理解し，また，数値計算では誤差がどのように発生するかを理解しておく必要がある。

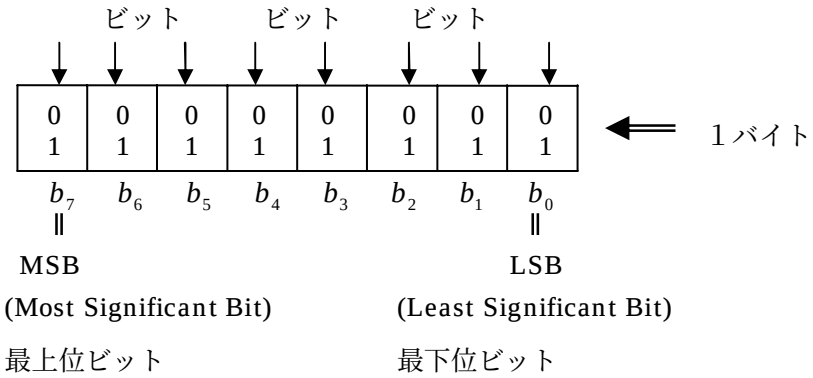
本書は自己完結的であることを目指しており，また, 乱数を使う方々が必ずしも情報科学あるいは数値計算に詳しい方とは限らないので，次節（２節）においてコンピュータ内での数値情報の扱いについて初歩からスタートして少し詳しく説明し, 続く３節で数値計算における誤差について述べることにする。もちろんこれらについて理解されている方は読み飛ばして頂いて結構である。その後，新しい乱数生成法とそのハッシュ関数への応用について述べることにする。

２．コンピュータ内での数値情報の処理形態

コンピュータ内では 0-1 のディジタル信号（電流のパルスなど）で処理が行われている。

1　0　0　1　1　0　1

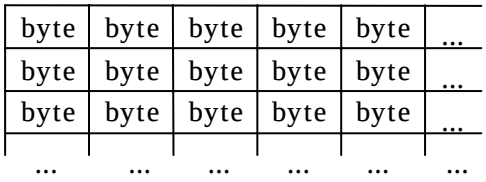
この世界では { 0, 1 } が基本状態の全てであり,「0 ↔ 1 を区別する」 ことに対応する情報単位が最小である。この情報の最小単位を**ビット**(bit)という。実際の情報処理を行うにあたっては，ビット単位では能率が悪いので通常８ビットをまとめて１つの処理単位にしている。これを**バイト**(byte)という。



（註）MSB, LSB は，後に断り無く使われる。

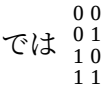
コンピュータの主記憶装置(メモリ)には，このバイト列がぎっしり詰まっていて，そのビット配列は状況に応じて様々な意味に解釈されている。

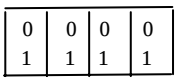
［コンピュータ主記憶装置の概念図］

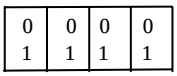


２.１　１バイトの世界

１ビット  では  の２種類の情報を見分けることができる。

２ビット  では  の，$2^2=4$ 種類の情報を見分けることができる。

４ビット  では $2^4=16$ 種類の情報を見分けることができる。

＊ ４ビット  では 16 種類の情報を見分けることができるが，16 を法として数をかぞえる方法を**１６進法**という。情報処理の世界では 16 種類の文字

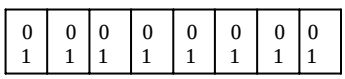
0 1 2 3 5 6 7 8 9 A B C D E F

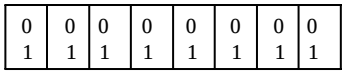
を

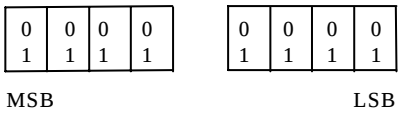
0000 →0　0001 →1　0010 →2　0011 →3
0100 →4　0101 →5　0110 →6　0111 →7
1000 →8　1001 →9　1010 →A　1011 →B
1100 →C　1101 →D　1110 →E　1111 →F

の規則で割り当て，これら 16 種類の文字を使って(16進で)数を数えることが多い。

（註）この 16 進表記も，後に断り無く使われる。

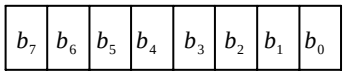
そして，１バイト  では $2^8=256$ 種類の情報を見分けることができる。

１バイト 
MSB　　LSB
の状態を記述するのに，これを



と２つに分割し，それぞれを１６進表記して並べて書くことが多い。

（例） 00001110 → 0000 1110 → 0E

最小単位である１バイトデータ  をどのように解釈するかに当たっては以下の４つの代表的な立場

（１）１バイトが数値を表すとみなす立場
（２）１バイトが文字を表すとみなす立場
（３）１バイトを音声，画像などの情報記述型データとみなす立場
（４）１バイトを行動命令（計算機の機械語）とみなす立場

がある。このうち，（１）が数値計算で扱われるので，少し詳しく説明する。

２.２ 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 , $i = 0,1,\cdots,7$，であるから，

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

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，

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

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「$b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$ の **2 の補数**」の

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

である数を表す。

（例） 10110001 を 1 バイトが 0～255 の整数値を表すとみる立場に立つと

$$2^7 + 2^5 + 2^4 + 1 = 128 + 32 + 16 + 1 = 177$$

を表す。1 バイトが -128～127 の整数値を表すとみる立場に立つと,
負の数で絶対値が $2^8 - 177 = 256 - 177 = 79$ となる数，すなわち，-79
を表す。

現在使われているパソコンの多くは,通常 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，表せる正の最大値は 2147483647 ，負の最小値は -2147483648 である。

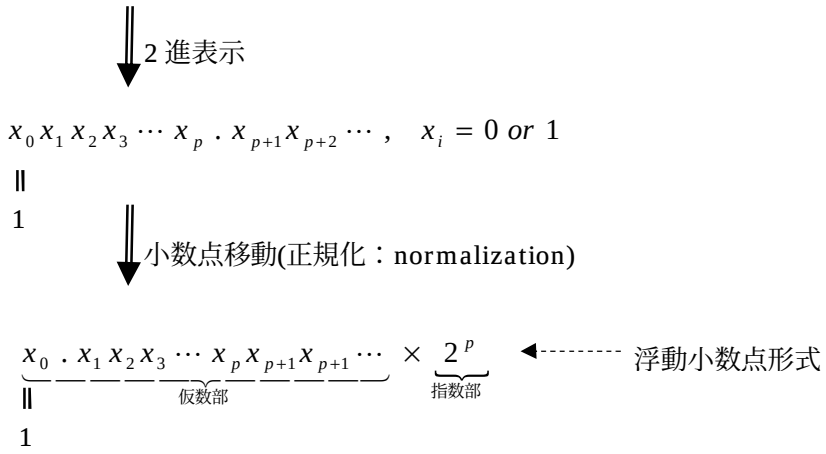
２.３ コンピュータにおける実数の扱い

コンピュータでの小数点を含む実数値の扱いは,整数の場合とかなり異なる。コンピュータでは,与えられた実数に対して，例えば

$$1234.567 = 1.234567 \times 10^3$$
$$0.01234567 = 1.234567 \times 10^{-2}$$
$$1234567000 = 1.234567 \times 10^9$$

などのように小数点を定位置に移動した後の値(仮数) 1234567(小数点は省略する)と，その際移動した小数点の桁数(指数) 3, -2, 9 をそれぞれ別に記憶するのである（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

$$\text{実数} \quad x = x_0 \cdot 2^p + x_1 \cdot 2^{p-1} + \cdots + x_p \cdot 2^0 + x_{p+1} \cdot 2^{-1} + x_{p+2} \cdot 2^{-2} + \cdots$$



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 $x_1 x_2 x_3 \ldots$ を仮数として記憶する（ケチ表現，IEEE 754 規格)。

（例）　$x = 33.75$ を 2 進表示すると，

$$x = 33.75 = 32 + 1 + 0.5 + 0.25$$
$$= 2^5 + 2^0 + \left(\frac{1}{2}\right)^1 + \left(\frac{1}{2}\right)^2 = 2^5 + 2^0 + 2^{-1} + 2^{-2}$$
$$= 1\cdot 2^5 + 0\cdot 2^4 + 0\cdot 2^3 + 0\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} + 1\cdot 2^{-2}$$

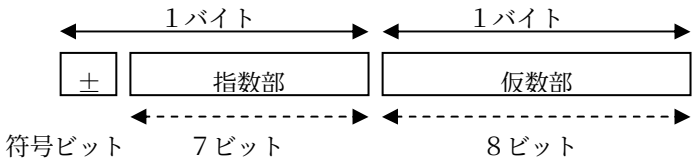
より，100001.11 となる。浮動小数点形式に直せば

$$100001\ .11 = 1.0000111\ \times 2^5$$

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(ケチ表現)，指数として 5 を(あらかじめ定められた規則により)
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。)

２.４　２バイトによる実数の世界

ここでは, 実数が実際にどのようにメモリ上に格納され, どのような数が表わされるかを考察する。簡単のため，まず，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

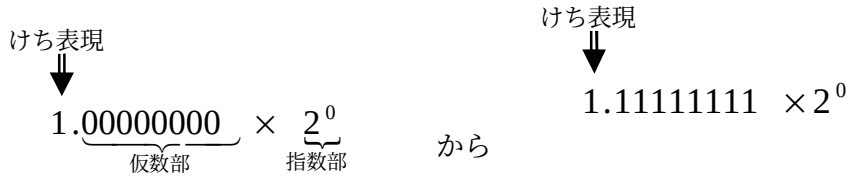


考察する２バイトによる浮動小数点形式

を,「2バイトによる実数の世界」という視点で考察する。

[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



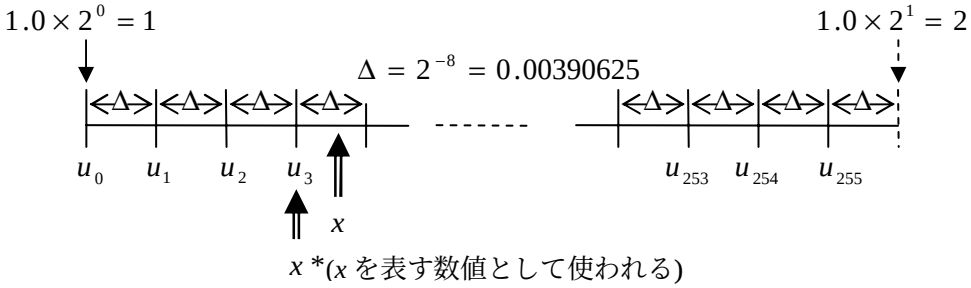
までの２５６種類となる。2 進数の 0.00000001 は，10 進数では 2^{-8}=0.00390625 である。よって

$$\Delta = 0.00000001 \quad (2\text{進}) = 2^{-8}(10\text{進}) = 0.00390625$$

とおけば，これら２５６種類の数値は,

	2進		10進
u_0	$=1.00000000$	$\longrightarrow$	1
u_1	$=1.00000001$	$\longrightarrow$	$1+\Delta \quad =1.00390625$
u_2	$=1.00000010$	$\longrightarrow$	$1+2\cdot\Delta \quad =1.0078125$
u_3	$=1.00000011$	$\longrightarrow$	$1+3\cdot\Delta \quad =1.01171875$
	------	------	-------
u_{254}	$=1.11111110$	$\longrightarrow$	$1+254\cdot\Delta \quad =1.9921875$
u_{255}	$=1.11111111$	$\longrightarrow$	$1+255\cdot\Delta \quad =1.99609375$

のようにとびとびの値を表すことになる。



したがって，この 2 バイトで実数を表わそうとすれば，各 $x \in [1,2)$ について x に最も近い値 $x^* = 1 + k \cdot \Delta$ 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。

一般に，x を真値，x^* をその近似値とするとき，

$$|x - x^*| \leq \varepsilon_\alpha$$

を保証する最小の $\varepsilon_\alpha = \varepsilon_\alpha(x^*)$ 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：

誤差限界の桁

$$x^* = d_1^* d_2^* \cdots d_p^* \,.\, d_{p+1}^* d_{p+2}^* \cdots$$

←― 有効桁 ―→

すると，今扱っている浮動小数点形式では，$x \in [1,2)$ の近似値 x^* については，誤差の限界は，

$$\Delta = 0.00000001(2\text{進}) = 0.000000001111\cdots\cdots(2\text{進})$$

b_8 b_9

であるから

誤差限界の桁

$$x^* = 1\,.\,b_1^* b_2^* \cdots\cdots b_8^* b_9^* \cdots \quad (2\text{進})$$

←―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

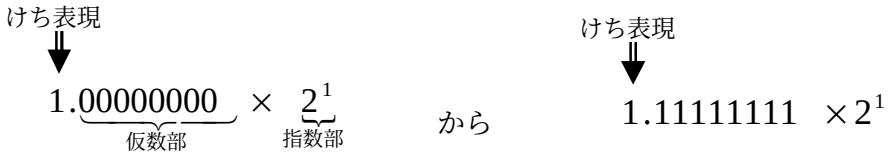
誤差限界の桁

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

となり，有効桁数は３桁となる。

（註：コンピュータ（２進法）の世界で話を閉じていれば有効桁数の記述は明快であるのに，その話を人間の世界（10 進法）に持ち込むから，ややこしくなってきてしまう。）

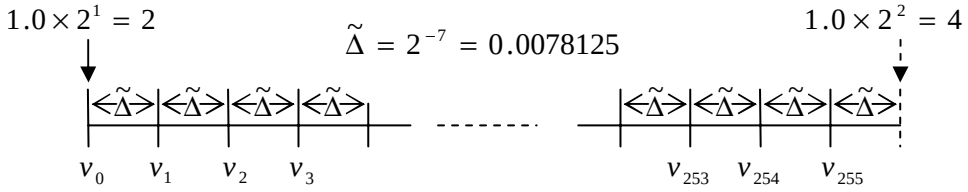
（２） 指数部 2^p が $2^1 = 2$ のときは，表せる数値は



までの２５６種類の数値を表せる。2 進数の 0.0000001 は，10 進数では

$$\tilde{\Delta} = 0.0000001 \ \ (2\text{進}) = 2^{-7}(10\text{進}) = 0.0078125$$

である。したがって，これら２５６種類の数値は，１０進で以下の図のようなとびとびの値になる：



[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しよう。

指数部のビット	対応する指数	
0000000	数 0(仮数=0 の時)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	$\times 2^{0}$	⟸ **このビット配列を指数値の起点とする**
1000000	$\times 2^{1}$	
1000001	$\times 2^{2}$	
1000010	$\times 2^{3}$	
……	…	
1111110	$\times 2^{63}$	
1111111	無限大(仮数=0 の時) および 非数を表すのに使用	

(*)denormalized 数...けち表現を使わない表現

このとき，正数で，表すことのできる最大値は,

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

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

けち表現

$$1.00000000\ \times 2^{-62} = 2.168 \times 10^{-19}$$

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

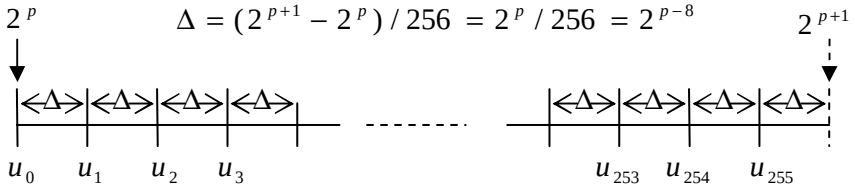
けち表現なし

$$0.00000000\ \times 2^{-62} = 2^{-70} = 8.470 \times 10^{-22}$$

となる。以下に例をあげておく。

[例]　指数部 2^p が一般のときの有効桁数はどうなるか。

（答）2^p 以上 2^{p+1} 未満の 256 個の値が表現できる。



$2^p = 0.d_1d_2d_3\cdots \times 10^q$ （10 進）とすると，誤差の限界Δは，2^pと 2^{p+1}の区間を 256 等分したときの小区間の幅になるから,

$$\begin{aligned}\Delta &= (2^{p+1} - 2^p) / 256 = 2^p / 256 \\ &= (0.d_1d_2d_3\cdots \times 10^q) / 256 \\ &= (0.d_1d_2d_3\cdots / 256) \times 10^q\end{aligned}$$

となる。したがって,

$d_1d_2d_3$. … > 256 のとき，$(0.d_1d_2d_3\cdots / 256) > 0.001$ となり，d_3

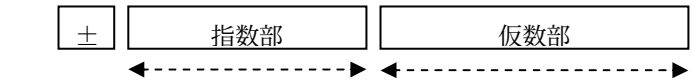
の桁が誤差限界の桁に入り，有効桁数は２桁になるが，
$2^{p+1} = 1.d_1^* d_2^* d_3^* \cdots \times 10^q$ であるときは，$2^p + k\Delta$ が 2^{p+1} に近いときは，1. が有効桁に寄与するので，有効桁数は３桁になる。

$d_1 d_2 d_3$ ．… ＜ 256 のとき，$(0.d_1 d_2 d_3 \cdots / 256) < 0.001$ となり，d_4 の桁が誤差限界の桁に入り，有効桁数は３桁となる。この場合
$2^{p+1} = 1.d_1^* d_2^* d_3^* \cdots \times 10^q$ の形（1. がある形）になることはないので，有効数字が３桁から４桁に上がることはない。

$d_1 d_2 d_3$ ．… ＝ 256 のときは，256=255.9999... と考えれば，
$d_1 d_2 d_3$ ．… ＜ 256 に準じた扱いができる。

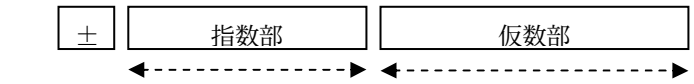
現在使われているパソコンの多くは，IEEE 754 規格に基づいて，

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。

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。
また，$\cdots \times 2^0$ を表す指数部のビット配列は 7F(16 進)である。

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。
また，$\cdots\times2^{0}$ を表す指数部のビット配列は 3FF(16 進)
である。

[註] {単精度}

仮数部が 23 ビットであり $2^{23}=8{,}388{,}608$ (7桁) であるので，指数部が $2^{p}=0.d_1d_2d_3\cdots\times10^{q}$（10 進）の場合，誤差の限界Δは，$2^{p}$ と 2^{p+1} の区間を等分したときの小区間の幅になるから，

$$\begin{aligned}\Delta &= (2^{p+1}-2^{p})/8{,}388{,}608 = 2^{p}/8388608\\ &= (0.d_1d_2d_3\cdots\times10^{q})/8388608\\ &= (0.d_1d_2d_3\cdots/8388608)\times10^{q}\end{aligned}$$

となる。したがって，

$d_1d_2\cdots d_7.\cdots > 8388608$ のとき，$(0.d_1d_2\cdots d_7\cdots/8388608) > 0.0000001$ となり，d_7 の桁が誤差限界の桁に入り，有効桁数は６桁になるが，$2^{p+1}=1.d_1^{*}d_2^{*}d_3^{*}\cdots\times10^{q}$ であるときは，$2^{p}+k\Delta$ が 2^{p+1} に近いときは，1. が有効桁に寄与するので，有効桁数は７桁になる。

$d_1d_2\cdots d_7.\cdots < 8388608$ のとき，$(0.d_1d_2\cdots d_7\cdots/8388608) < 0.0000001$ となり，d_8 の桁が誤差限界の桁に入り，有効桁数は７桁になるが，$2^{p+1}=1.d_1^{*}d_2^{*}d_3^{*}\cdots\times10^{q}$ であるときは，$2^{p}+k\Delta$ が 2^{p+1} に近いときは，1. が有効桁に寄与するので，有効桁数は８桁になる。

{倍精度}

仮数部が 52 ビットであり $2^{52}=0.45036\cdots\times10^{16}$ (16 桁整数)であるので，指数部が $2^{p}=0.d_1d_2d_3\cdots d_{16}\cdots\times10^{q}$（10 進）の場合，誤差の限界Δは，$2^{p}$ と 2^{p+1} の区間を等分したときの小区間の幅になるから，

$$\begin{aligned}\Delta &= (2^{p+1}-2^{p})/(0.45036\cdots\times10^{16}\text{(16 桁整数)})\\ &= 2^{p}/(0.45036\cdots\times10^{16}\text{(16 桁整数)})\\ &= (0.d_1d_2d_3\cdots d_{16}\cdots\times10^{q})/(0.45036\cdots\times10^{16}\text{(16 桁整数)})\end{aligned}$$

となる。したがって，$= (0.d_1d_2d_3\cdots d_{16}\cdots/(0.45036\cdots\times10^{16}\text{(16 桁整数)}))\times10^{q}$

$d_1 d_2 d_3 \cdots d_{16}$.$\cdots$ > 45036 $\cdots \times 10^{16}$ (16 桁整数) のとき

$0.d_1 d_2 d_3 \cdots d_{16} \cdots / (0.45036 \cdots \times 10^{16}$ (16 桁整数)) $> 1 \times 10^{-16}$ となり，d_{16} の桁が誤差限界の桁に入り，有効桁数は１５桁になるが，$2^{p+1} = 1.d_1^* d_2^* d_3^* \cdots \times 10^q$ であるときは，$2^p + k\Delta$ が 2^{p+1} に近いときは，1. が有効桁に寄与するので，有効桁数は１６桁になる。

$d_1 d_2 d_3 \cdots d_{16}$.$\cdots$ < 45036 $\cdots \times 10^{16}$ (16 桁整数) のとき

$0.d_1 d_2 d_3 \cdots d_{16} \cdots / (0.45036 \cdots \times 10^{16}$ (16 桁整数)) $< 1 \times 10^{-16}$ となり，d_{17} の桁が誤差限界の桁に入り，有効桁数は１６桁になる。この場合

$2^{p+1} = 1.d_1^* d_2^* d_3^* \cdots \times 10^q$ の形（1. がある形）になることはないので，有効数字が１６桁から１７桁に上がることはない。

３．数値計算における誤差

コンピュータにおける実数値の計算は，本質的に"近似の世界"で行われるため誤差が生じる。例えば，無理数である π の正確な値

π＝3.14159265358979323846264338327950288...(無限長)

をコンピュータでは記憶できないので，適当な長さ（有限長）でやめた近似数を用いることになる（→誤差の発生）。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

数値を有限の桁数で記述する（切り捨て，切り上げ，四捨五入）ために生ずる誤差。

例：$1.234567^2 \Rightarrow 1.524155$ (切り捨て)

（例）ある C 言語で

```
float  f=1.0/3;
```

として表示したところ

```
f=0.333333343267440796
```

となった。float は単精度であり，有効桁数ほぼ 7 桁が保証されている。

```
double  f=1.0/3;
```

としたところ

```
f=0.333333333333333315
```

となった。double は倍精度であり，有効桁数 15 桁が保証されている。

（２）情報落ち誤差

有効 7 桁での演算のとき

$$123456.7 \ \pm \ 0.001234567 \quad \Rightarrow \quad 123456.7$$

となるように，$|x|>>|y|$（$|y|$が$|x|$に比べて非ｔ常に小さい）のとき，加減算 $x \pm y$ で，y の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）。

（例）

$$e=1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\frac{1}{4!}+\cdots+\frac{1}{n!}+\cdots$$

を正確に計算しようと思って，やたらに n を大きくして計算しても情報落ち誤差が発生して意味がない。情報落ち誤差を考慮しながら計算することが賢いといえる。

（３）桁落ち誤差

有効 7 桁での演算のとき

$$\underbrace{100000\ .2}_{\text{有効 7 桁}} - \underbrace{100000\ .1}_{\text{有効 7 桁}} \quad \Rightarrow \quad \overbrace{000000\ .1}^{\text{脱落する有効桁}}$$

（.1 は有効桁で残った部分）

となるように，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非常に恐ろしい誤差。英語では cancellation error などと言われる。この誤差が，本報告で主要な役割を演じる。

（例）有名な例であるが，$ax^2+bx+c=0$ の解の公式 $x=\dfrac{-b\pm\sqrt{b^2-4ac}}{2a}$ で

$b^2>>|4ac|$ であると，b と $\sqrt{b^2-4ac}$ が非常に近くなり $-b+\sqrt{b^2-4ac}$ で桁落ち誤差が発生する。$\dfrac{-b+\sqrt{b^2-4ac}}{2a}$ の分子・分母に $b+\sqrt{b^2-4ac}$ を掛けて $x=-\dfrac{2c}{b+\sqrt{b^2-4ac}}$ と変形すると，引き算が足し算になって桁落ちが防げる。

（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。例えば

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

であるが，コンピュータで無限に計算するわけにはいかないので計算を途中で打ち切ることになる。打ち切られた部分に対応する値が打ち切り誤差になる：

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} \cdots + (-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!} \quad \left[+ (-1)^n \frac{x^{2n+1}}{(2n+1)!} + \cdots \right.$$

以降の省略が
打ち切り誤差になる

４．$\sin x$ のグラフの乱れの解析

ここに至ってやっと sin x のグラフの乱れを説明する環境が整った。単精度計算では有効桁数が 10 進で 7 桁程度，倍精度計算では有効桁数が 10 進で 15 桁であることに留意して，改めて x=22 での計算の様子を見てみよう。

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

による計算

	単精度計算	倍精度計算
F1	22.000000000000000	22.000000000000000
F2	-1774.666625976560000	-1774.666666666660000
F3	42946.933593750000000	42946.933333333300000
F4	-494912.281250000000000	-494912.279365079000000
F5	3326910.250000000000000	3326910.322398580000000
F6	-14638405.000000000000000	-14638405.418553700000000
F7	45416588.000000000000000	45416591.170384800000000
F8	-104674424.000000000000000	-104674429.173648000000000
F9	186258896.000000000000000	186258910.735463000000000
F10	-263594464.000000000000000	-263594481.859544000000000
F11	303761248.000000000000000	303761260.047665000000000
F12	-290554240.000000000000000	-290554248.741245000000000
F13	234380416.000000000000000	234380427.317938000000000
F14	-161595616.000000000000000	-161595622.253393000000000
F15	96320536.000000000000000	96320543.313598900000000
F16	-50128108.000000000000000	-50128110.713743900000000
F17	22975382.000000000000000	22975384.077132600000000
F18	-9344609.000000000000000	-9344609.994396800000000
F19	3395488.500000000000000	3395488.916882920000000
F20	-1108918.000000000000000	-1108918.107807910000000
F21	327266.031250000000000	327266.075718922000000
F22	-87705.843750000000000	-87705.858609057700000
F23	21439.207031250000000	21439.209882214100000
F24	-4799.526367187500000	-4799.527096665880000
F25	987.657653808593000	987.657786898931000
F26	-187.461288452148000	-187.461321121208000
F27	32.921360015869100	32.921364086598300
F28	-5.364962577819820	-5.364963036334550
F29	0.813484311103820	0.813484370171028
F30	-0.115057393908500	-0.115057403612734
sin*x*	-16.180646896362300	-0.022378806339483

では，

単精度計算で生じた　sin22 の値　-16.180... における 16. の位置する桁

は，F_8 〜F_{14}（単精度） においては，有効桁の外にある。すなわち単精度計算では，大きな絶対値をもつ F_i の有効桁は，総和をとる際の＋と－によって互いに打ち消されて有効桁が残らなくなり，その残骸の桁（有効桁を外れた桁）にあたるところに，総和の値(sin22)が出てくるので，正確な sin22 の値が得られないことが分る。よって，$\sin x$ のグラフの乱れの原因は桁落ち誤差による，大きな値を持つ F_i の上位の有効桁の脱落であったことが分かる。

倍精度計算では，sin22 の値 -0.02237... の最初の何桁までは正しく計算されている（有効桁から外れていない）筈であるが，sin22 の正確な値は

sin22 = -0.008851309290403875921690...

である。この値は上表の倍精度計算の値とは明らかに異なる。実は，倍精度計算でも，F_{34} あたりまで計算しないと sin22 の近似値は得られない。すなわち，F_{30} より後を打ち切った sin の公式を使って sin22 を計算すると，「打ち切り誤差(公式誤差)」が表（おもて）に出てきて，正確な値が得られないのである。

Horner 法による計算

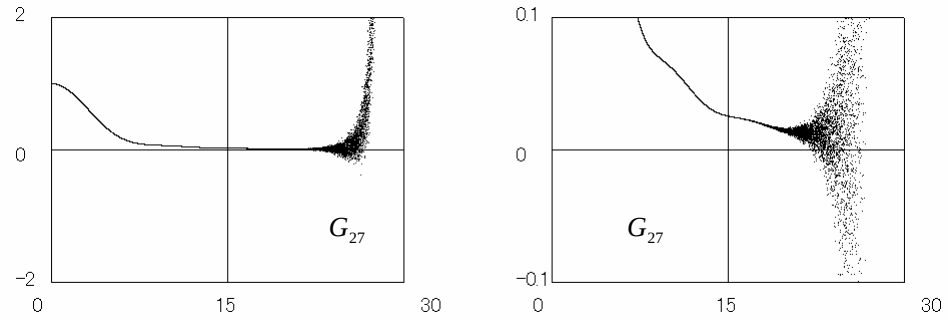
$$\sin x \approx x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\frac{x^9}{9!}\cdots+(-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!}$$

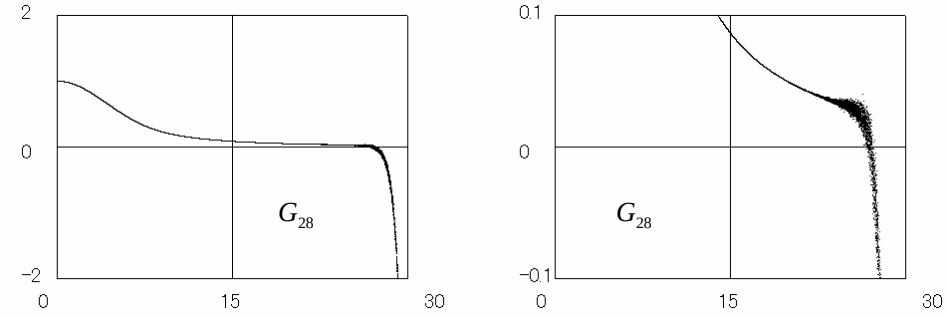
$$= x\left(1-\frac{x^2}{2\cdot 3}\left(\cdots\left(1-\frac{x^2}{(2n-4)(2n-3)}\left(1-\frac{x^2}{(2n-2)(2n-1)}\right)\right)\cdots\right)\right)$$

G_1 ⟶　　G_2 ⟶　⟵ G_2

$$G_0=1,\qquad G_k=1-\frac{x^2}{(2n-2k)(2n-2k+1)}\times G_{k-1},\qquad G_{30}=x\cdot G_{29}$$

では，桁落ちを起こすような項のキャンセルを引き起こす加減算が一見ではみられない。計算途中の項を見てみると，G_{27} あたりから顕著な乱れが観察される（下図参照（右図は，左図の Y 軸拡大））。





	単精度計算	倍精度計算
G1	0.858562231063842000	0.858562244301578000
G2	0.869817018508911000	0.869817003057028000
G3	0.858252048492431000	0.858252043946261000
G4	0.849276483058929000	0.849276491556607000
G5	0.838804006576538000	0.838803991406510000
G6	0.827388942241668000	0.827388974557503000
G7	0.814775109291076000	0.814775086176765000
G8	0.800832748413085000	0.800832756712346000
G9	0.785380363464355000	0.785380368633014000
G10	0.768217027187347000	0.768217013159524000
G11	0.749111294746398000	0.749111312841288000
G12	0.727800428867340000	0.727800393832444000
G13	0.703987061977386000	0.703987066710165000
G14	0.677339255809783000	0.677339261091174000
G15	0.647492229938507000	0.647492255518141000
G16	0.614056348800659000	0.614056340306920000
G17	0.576633512973785000	0.576633520358191000
G18	0.534848988056182000	0.534848960244392000
G19	0.488405317068099000	0.488405342374929000
G20	0.437171012163162000	0.437170986406034000
G21	0.381313532590866000	0.381313574793799000
G22	0.321486204862594000	0.321486138969856000
G23	0.259050846099853000	0.259050993993283000
G24	0.196278139948844000	0.196277685302888000
G25	0.136376187205314000	0.136378184667288000
G26	0.083248965442180600	0.083235536403225500
G27	0.040654778480529700	0.040809532877115400
G28	0.016154360026121100	0.012409304373806700
G29	-0.303118377923965000	-0.001017219487076060
sin$x$	-6.668604373931880000	-0.022378828715673400

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**********

単精度の仮数として実際に記憶される桁

桁落ち

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。G_{26} ,G_{27} ,G_{28} の各計算で，１０進２桁 0.0 の桁落ちが発生すると，これだけで計１０進６桁の桁落ちが発生する。G_{26} ,G_{27} ,G_{28} の前後でも少なからず桁落ちが発生していると考えられるので，結局単精度計算の全ての有効桁が桁落ちで失われてしまい, Horner 法による計算でも $\sin x$ のグラフが乱れるのである。

このさみだれ的な計算中の桁落ちの自然発生に対して, 人工的に桁落ちをさみだれ的に発生させることができれば同様な乱れ（→乱雑性→乱数）を作ることができると考えられる。

５．Horner 法計算を用いた乱数の生成と検定

この節では，桁落ち誤差領域における sin x の Horner 法計算により，乱数が生成できることを示す。最初に Horner 法計算から２万個の４桁整数を生成する方法を示し，続いて生成された整数列が乱数列と見なされること統計的検定により示す。この検定にパスすることは, 新しい乱数生成法を導出する際の精神的支えを提供することになる。

統計的検定法については, 馴染みの無い方が多いと思われるので, 統計学に関する知識が全くなくても理解できるように, また本報告が自己完結的であるようにするため, 検定の基礎的な内容から丁寧に述べることにする。

５．１ Horner 法を用いた４桁整数列の生成

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 の近似式を，Horner 法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**86123657E+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, 8188

5659, 9127, 9128, 7107, 7425, 5777, 7124, 6652, 1248, 4866

… …　　… …

1865, 3989, 1062, 3191, 5297, 7684, 4450, 0978, 9402, 5752.

[使用したプログラム]

```
function  Ho(x:single) : single;
  var
    i : integer;
    G : single;
  begin
    G:=1;
    for i:=29 downto 1 do
        G:=1-(G*x*x/((2*i)*(2*i+1)));
    Ho:=x*G;
  end;
```

```
procedure gen20000sglHo;
  var
    x, y, xstep : single;
    i : integer;
    YStr, RanStr : string[50];
    F : TextFile;
  begin
    XStep:=(26.0-22.0)/19999;
    AssignFile(F,'ran20000.txt');
    Rewrite(F);
    for i:=0 to 19999 do
      begin
        x:=22+xstep*i;
        y:=abs(Ho(x));
        YStr:=FloatToStrF(y, ffExponent, 15, 3);
        if YStr[2]='.' then
            RanStr:=Copy(YStr,5,4)
          else
            RanStr:='error';
        WriteLn(F,RanStr);
      end;
    CloseFile(F);
  end;
```

次に，これらの整数列が乱数列であることを検証する。

5.2 Horner 法による数値列の乱数性の検定

5.2.1　乱数とは何か？

現実的には，正しく作られた「0～9の目を持つさいころ」(正 20 面体を使うとよい)を何回も投げるときに, 出てくる目を並べて作られる数をイメージしてもらえれば良いが, 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,

$$\Omega = \{0, 1, 2, \cdots, n-1\}$$

を n 種類の値をとる乱数値の空間とし,

[例]　Ω = さいころを投げたときに，出る可能性のある目全体

また,

X_i , $i = 0, 1, 2, \cdots$

を 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〕　$\Pr\{ X_1 = k_1 , X_2 = k_2 , \cdots, X_l = k_l \} = \prod_{i=1}^{l} \Pr\{X_i = k_i\}$

(Pr : 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　$\Pr\{X_i = k \} = \frac{1}{n}$

が要求されることが通常である。(一様乱数と言われる。)

5.2.2 乱数性の検定とは何か

我々は Horner 法で４桁の整数値を 20000 個発生した。ここで,

これらの数は均等な確率で(=一様に)，でたらめに(=独立に)出現している
と判断してよいか？

という疑問が生じてくる。

統計学では，以下のプロセスで「仮説の検定」を行っている。

（1）出現データに関する正しいと思われる仮説を設定して(例えば，一様かつ独立に出現する)，その仮説のもとで，何らかの計算式で出現データより得られる値（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。

＊正規分布，χ^2分布，F 分布などがよく使われる。

（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）で仮説が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）の仮説は否定されないが，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！

5.2.3 Horner 法による数値列の一様乱数性の検証

ここでは，Horner 法で生成した 20000 個の数値について乱数性の基本的な検証を行う。

(I) 最初に，予備評価として，2 数の対による平面上の点の散布状態の図示，
ならびに Monte Calro 法によるπの近似値の算出，
(II) 数字 0～9 の出現頻度のχ^2検定，
(III) 数字 0 の出現間隔のχ^2検定，
(IV)一様分布に対する Kolmogorov-Smirnov 検定，

を行う。検定結果は必要に応じてコンピュータソフトウェア Delphi 2.0 が内蔵している乱数関数 random による結果と比較することにする。

検定(I) 予備評価

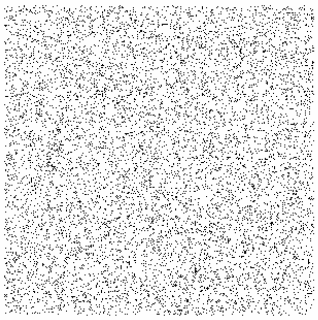
2 数の対による平面上の点の散布状態，および，Monte Calro 法によるπの近似値をみる。

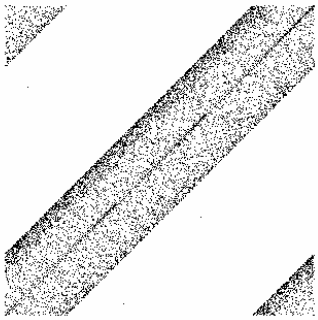
下の左図は発生した数値を２つずつ組み合わせ，10000 個の XY 平面上の点

(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。まんべんなく散らばっており一様乱数の性格を持っていることがうかがえる。

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り，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。





上の左図の 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。得られた数値データが，もし，一様に分布する乱数であれば，これらの点の総数の比 10000:7852 は，

「一辺の長さ１の正方形の面積」：「半径１の四分円の面積」

に近いと思われるから，

$$10000 : 7852 \quad = \quad 1 : \pi/4$$

となる。これより，

$$\pi \qquad 0.7852\times 4 = 3.1408$$

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（この値はシミュレーションとしてはちょっと良すぎる。たまたまであろう。）。同様の計算を，ソフトウェア付属の乱数関数 random(10000) で行うと(RandSeed=1)，$x^2+y^2<1$ を満たすものの数は 7871 個であり，π　3.1484 となり，相対誤差は 2.16684×10^{-3} と計算される。

検定(II) 文字 0～9 の出現頻度の χ^2 検定 (一様性の検定)

我々は Horner 法で４桁の整数値を 20000 個発生した。ここで，

発生した 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。

「５．２．２ 乱数性の検定とは何か」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。）

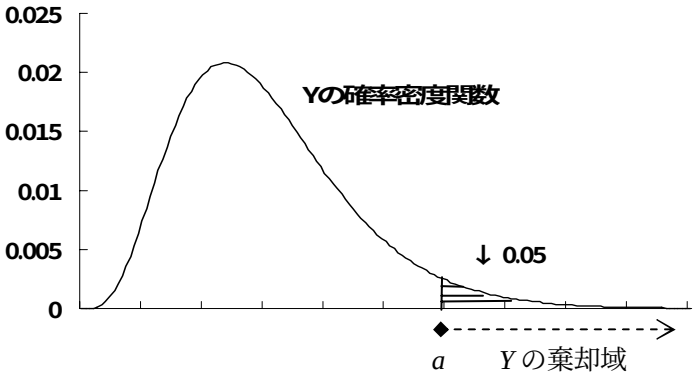
（１-２）現実には，０～９の数字は理想通りには現れなく，バラツキがあるので，0 の発生数は X_0，1 の発生数は X_1，… ，9 の発生数は X_9 となる。（$X_0,\cdots,X_9$ はある分布にしたがう確率変数になっている。）

（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，$\Pr(a \le Y)=0.05$ となるように a を定める。
（[a, ∞) 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 $X_0=x_0,\cdots,X_9=x_9$

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
8022	8021	7897	7992	8067	8107	7953	8016	8022	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，$x_0,\cdots,x_9$ と各 X_i の期待値 8000 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，$\Pr(y \le Y)$ を答として返してくる。したがって，

(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，

(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。

【参考】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($np_i \ge 5$)，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

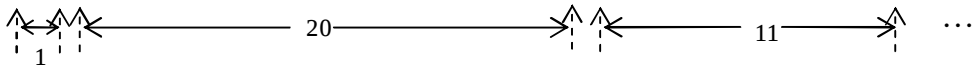
$$Y=\frac{(x_1-np_1)^2}{np_1}+\frac{(x_2-np_2)^2}{np_2}+\cdots+\frac{(x_k-np_k)^2}{np_k}$$

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 $\Pr(y \le Y)$ 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 …



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
出現回数	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	71.1	64	57.6	51.8	46.6	42

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	30.6	27.5	24.8	22.3	20.1	18.1	16.3	146.4

数字 0 が確率 0.1 で独立に発生すると仮定すれば， $p = 0.1$ とおくと，0 が続けて発生する確率は p ，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（この分布は幾何分布と呼ばれる）。これから，全部で 8021 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，以下の

Kolmogorov-Smirnov 検定(KS 検定と略)

を行う。

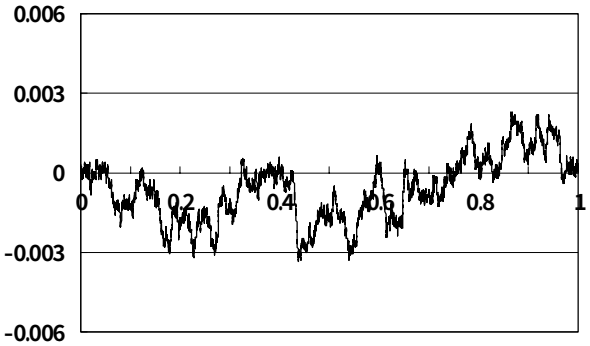
[0,1] 上の一様分布から導かれる分布関数 F は, $x \in [0,1]$ では $F(x) = x$ で与えられる。発生した乱数値を順に $u_1, u_2, u_3, \cdots, u_n, n = 20000$ ，とおき，数値 $0.u_1, 0.u_2, 0.u_3, \cdots$ の分布関数を

$$F_n(x) = \frac{1}{n} \#\{\ i \mid 0.u_i \le x\ \} \quad (\#=\text{集合に含まれる要素の個数})$$

で定める。このとき,

$$\{F_n(x) - F\ (x)\} \times \sqrt{n} \quad (\Leftarrow F_n \text{の一様分布からのずれ})$$

のグラフは以下のようになる。



Kolmogorov-Smirnov 検定は,

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

を評価するものである。まず,

$$\#\{\ i \mid\ 0.u_i \le k\ \} - 2(k+1), \qquad k = 0,1,,9999,$$

を求めると，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

$$K_n^+ = 43/\sqrt{20000} = 0.3041 \qquad K_n^- = 69/\sqrt{20000} = 0.4879$$

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　-0.5ln0.05 =1.2239 であるので，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，我々が発生した数値は一様乱数と呼んでよいであろう。この結果に自信を得て，次節以降，人工的な桁落ち誤差の発生による新しい乱数生成法を展開することにする。

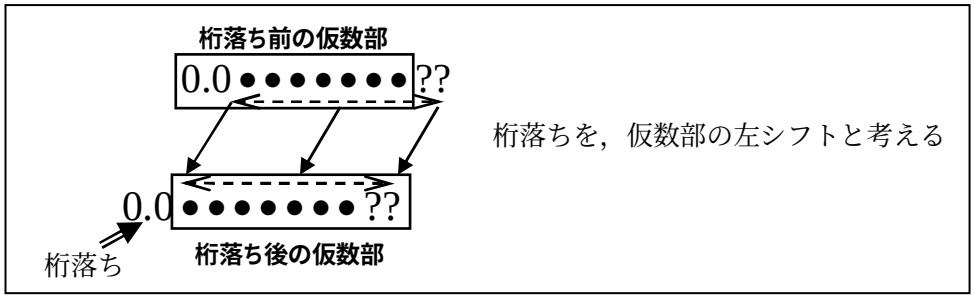
６．新しい乱数生成法(単純実数シフト法(SSR 法))の構成

Horner 法による乱数発生の主要因は

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

単精度(桁落ち前の仮数)　単精度(桁落ち後の仮数)　桁落ち

による桁落ちであった。この桁落ちは，仮数部（数値データ）の上位桁が落ちて下位桁が持ち上がる左シフトとして捉えることができる。



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。この節では，仮数部の左シフトを人工的に発生させることにより，新たな乱数発生法(実数シフト法)を導入する。

６．１ 単純実数シフト計算(SSR 計算)

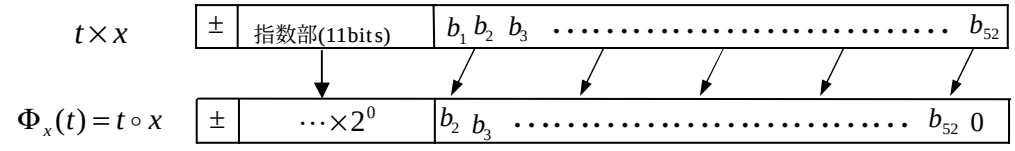
以下の計算を $t, x \in [1,2)$ の**単純実数シフト計算**（the Simplified Shift-Real computation）といい，$\Phi_x(t)$ あるいは $t \circ x$ で表すことにする。

単純実数シフト計算 (SSR 計算) $\Phi_x(t)$

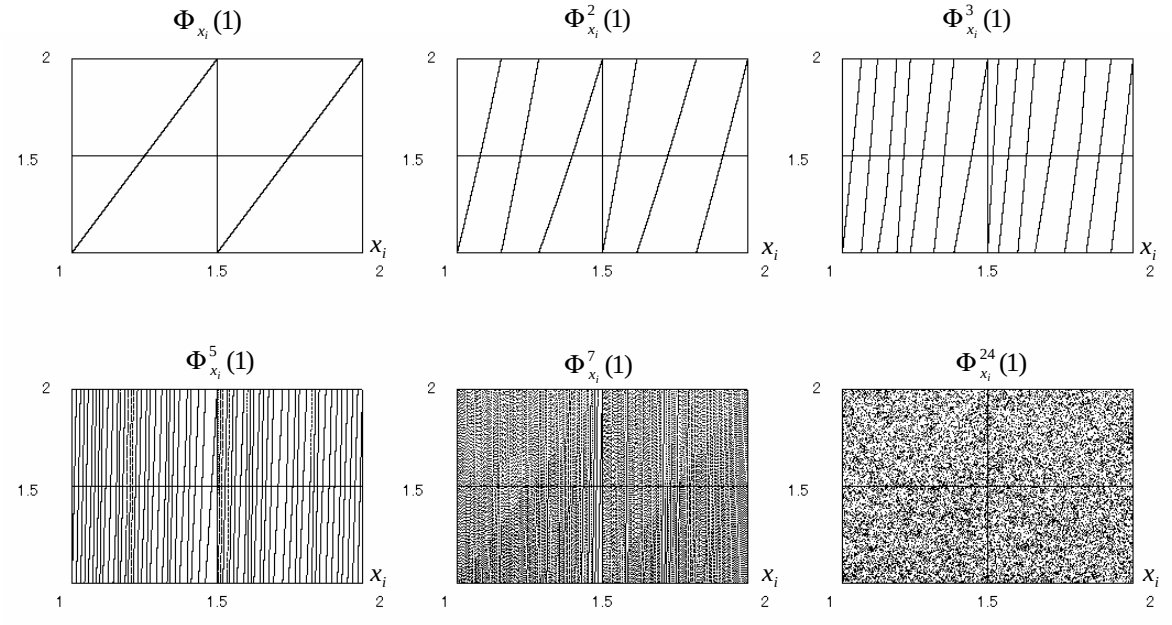
i)　$t, x \in [1,2)$ に対し $t \times x$ を倍精度計算し，

ii)　$t \times x$ の仮数部の全ビットを１ビット左にシフトし（⇒最左端ビットは捨てる），

iii) $t \times x$ の指数部を $\cdots \times 2^0$ となるように設定する（⇒ $1 \le t \times x < 2$ となる）。



単純実数シフト計算の ii) における仮数部の左シフトが，1ビット分の桁落ちに相当すると言うわけである。桁落ちをさみだれ的に発生するには，$\Phi_x(\Phi_x(t))$ のように，Φ_x を繰り返し適用すればよい。区間 $[1,2)$ を 19999 等分した各点 x_i における $\Phi_{x_i}^k(1)$，$i=0,1,\cdots,19999$，のグラフは，次のようになる：



$$\Phi_x^{24}(1)=1\circ\underbrace{x\circ x\circ x\circ\cdots\cdots\circ x\circ x}_{24},\quad x\in[1,2)$$

６．２ 実数シフト乱数生成法(SSR method, SSR 乱数生成法)

実数シフト乱数生成法は，乱数値の生成関数として，上述の $\Phi_x^{24}(1)$ を使うものである。

実数シフト乱数生成法(SSR 乱数生成法)

(1) $x\in[1,2)$ に対し，$\Phi_x^{24}(1)=1\circ\underbrace{x\circ x\circ x\circ\cdots\cdots\circ x\circ x}_{24}$ を計算する。

(2) $\Phi_x^{24}(1)$ の上位３桁を棄て，続く４桁を**乱数値**とする。

(3) x を $x_1, x_2, x_3, \cdots$ と $[1,2)$ 内で一様に変化させて，多くの乱数値を得る。

次ページに，$\Phi_x^{24}(1)$ を計算するプログラムコードを載せておく。

単純実数シフト計算のプログラムコード

[PASCAL(Delphi)]

```
    type
      DblI2 = record
        case I2 : Boolean of
          True  : (L, H : integer);
          False : (Dbl : double);
      end;
    function  SSR(x:double) : double;
      var
        i : integer;
        FLLH : DblI2;
      begin
        FLLH.Dbl:=1;
        for i:=1 to 24 do
          begin
            FLLH.Dbl:=FLLH.Dbl*x;
            FLLH.H:=(FLLH.H and $0007ffff) shl 1;   {shift mantissa H}
            if (FLLH.L and  $80000000)<>0 then inc(FLLH.H);
            FLLH.L:=FLLH.L shl 1;                   {shift mantissa L}
            FLLH.H:=FLLH.H or $3ff00000;            {set exponent 2^0}
          end;
        SSR:=FLLH.Dbl;
      end;
```

[FORTRAN]

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

[C]

```
/*=======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;
    double SSR(double x)
        {
        int    i;
        DblI2  FLLH;
        FLLH.Dbl=1;
        for (i=1; i<=24; i++)
            {
            FLLH.Dbl=FLLH.Dbl*x;
            FLLH.LH.H=(FLLH.LH.H & 0x0007ffff) << 1;   /*mantissa H*/
            if (FLLH.LH.L & 0x80000000) {FLLH.LH.H++;}
            FLLH.LH.L <<= 1;                          /*mantissa L*/
            FLLH.LH.H |= 0x3ff00000;             /*set exponent 2^0*/
            }
        return(FLLH.Dbl);
        }
```

実数シフト乱数生成法で x を $[1,2)$ 内で一様に変化させ方法は色々あるが，次節では，生成される数値の特性を調べるため，ホーナー法で乱数値を得た方法と同様の方法をとる。そのあとで長い周期を持つように x_i を変化させる方法を考える。この実数シフト乱数生成法のアルゴリズムは，最終的には完全整数演算化され，SSI32乱数生成法となる。

６.３ 単純実数シフト法による乱数の検定

実数シフト乱数生成法は，新しい乱数生成法であるので，まず手始めに，Horner 法を用いた４桁乱数列の生成法（４.１）と同様に 20000 個の乱数列を生成して，その乱数性を調べてみる。

1) $x_0 = 1$ と $x_{20001} = 2$ の間を 40001 等分して，

$$x_i = \frac{1}{40001} \times i, \quad i = 10001, 10002, \cdots, 30000, \text{ (端の分点を避ける)}$$

を作り，

2) 各 x_i について，$\Phi_x^{24}(1)$ を，単純実数シフト計算により倍精度計算し，得られた数値の上位３桁の数字を捨て，残った桁から続けて 4 桁の数を取り出し乱数値とする。

得られる 20000 個の数値は次のようになる：

9379 8539 5121 9581 9154 3286 9624 1871 5016 9400
3072 5611 5019 0157 0644 2609 6927 2225 8865 3027
・・・・・・・・・・・・・・・・・・・・・・・・・・・・
0892 1251 1111 9845 4444 4836 8051 9987 1203 2805

この 20000 個の 4 桁整数値について，既に説明した４つの方法で一様乱数性を調べてみる。

検定(Ⅰ) 実数シフト法による 2 数の対の平面上の点 10000 個の散布状態

$x^2 + y^2 < 1$ を満たすものの数は 7838 個であり，π 3.1352 となり，相対誤差は -0.002035 と計算される。

検定(II) 実数シフト法による数字 0～9 の出現頻度

数字 0	1	2	3	4	5	6	7	8	9
7818	7984	7814	8004	7960	8184	8146	8025	8066	7999

CHITEST: 0.0625

検定(III) 単純実数シフト法による数字 0 の出現間隔

間隔	0	1	2	3	4	5	6	7	8
出現回数	766	728	586	544	525	452	414	398	326
理論回数	781.69	703.531	633.168	569.822	512.908	461.549	415.453	373.839	336.514

9	10	11	12	13	14	15	16	17	18
301	266	231	235	195	157	173	156	131	126
302.794	272.583	245.295	220.737	198.712	178.831	160.899	144.819	130.395	117.336

19	20	21	22	23	24	25	16	27	28
104	109	88	79	68	71	62	64	54	46
105.642	95.0191	85.5659	76.9898	69.2908	62.3715	56.1343	50.4819	45.4142	40.9313

29	30	31	32	33	34	35	36	37	38以上
23	35	30	21	19	21	15	17	24	157
36.8382	33.1349	29.8214	26.8003	24.169	21.7326	19.5885	17.6394	15.8852	142.675

CHITEST : 0.3991

検定(IV) 単純実数シフト法による一様分布に対する Kolmogorov-Smirnov 検定

$$K^{+}_{20000} = 0.6576 \qquad K^{-}_{20000} = 0.8485$$

危険率 0.05 に対応する K^{+}_{20000}, K^{-}_{20000} の値の近似値は 1.2239 であるので, 一様に分布しているという仮定は否定されない。

◎ 実数シフト法に関する以上の結果をまとめると,

発生法	π の相対誤差	0-9 文字 χ^2	0 間隔 χ^2	K^{+}_{20000}	K^{-}_{20000}
Delphi	0.002167	0.8393	0.8841	1.0677	0.2758
Horner	-0.000252	0.8355	0.9262	0.3041	0.4879
実数シフト	-0.002035	0.0625	0.3991	0.6576	0.8485

(註：Delphi は Delphi 内蔵の乱数)

となる。この結果から,「単純実数シフト法」は一様乱数を生成するものと考えられる。

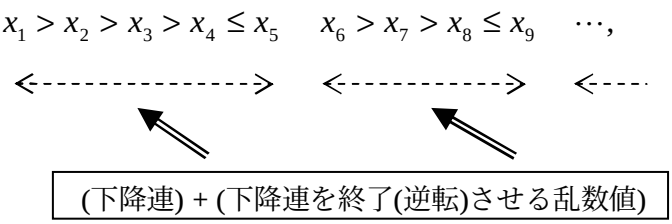
７．単純実数シフト法(SSR 法) 乱数をさらに詳しく検定する

我々は, ホーナー法, 単純実数シフト法(SSR 法)によってそれぞれ 20000 個の 4 桁整数を発生し, その乱数性を検定（I）～（IV）により行った。ここでは, この 20000 個の 4 桁整数に対してさらに検定（V）～（VIII）を追加する。各検定では, 最初に検定法の説明を行い, 続いて SSR 乱数についての検定結果を示す。

検定(V) 単純下降連[上昇連]テスト

i 番目の乱数値を x_i , $i=1,2,3,\cdots$, 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,



までを１つの下降連ブロックとみなし(上図では $x_1 > x_2 > x_3 > x_4 \le x_5$,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き(上図では $x_6 > x_7 > x_8 \le x_9$, 連の長さは 3), これを繰り返したときに現れる下降連の長さを χ^2 検定する。この単純下降連テストでは, 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 $\frac{1}{t!}$ 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。実際, a=0, b=1 として,

$$\Pr\{\text{長さ 2 以上の連が現れる}\}$$

$$=\Pr\{(x_1,x_2)\in[0,1]^2 \mid x_1>x_2\}$$

$$=\left|\{(x_1,x_2)\in[0,1]^2 \mid x_1>x_2\}\right| \quad (|\ |\text{は集合の測度を表す; 面積})$$

$$=\int_0^1 dx_1\int_0^{x_1} dx_2 = \int_0^1 x_1 dx_1 = \left[\frac{1}{2}x_1^2\right]_0^1 = \frac{1}{2}$$

Pr{長さ 3 以上の連が現れる}

$$= \Pr\{(x_1, x_2, x_3) \in [0,1]^3 \mid x_1 > x_2 > x_3\}$$

$$= \left| \{(x_1, x_2, x_3) \in [0,1]^3 \mid x_1 > x_2 > x_3\} \right| \quad (|\ | \text{は集合の測度を表す; 体積})$$

$$= \int_0^1 dx_1 \int_0^{x_1} dx_2 \int_0^{x_2} dx_3 = \int_0^1 x_1 dx_1 \left(\frac{1}{2} x_1^2\right) = \left[\frac{1}{3!} x_1^3\right]_0^1 = \frac{1}{3!}$$

一般に，{長さ t 以上の連が現れる} 確率は

Pr{長さ t 以上の連が現れる}

$$= \Pr\{(x_1, x_2, \cdots, x_t) \in [0,1]^t \mid x_1 > x_2 > \cdots > x_t\}$$

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

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

である。よって，

Pr{長さ r の連が現れる}

$$= \Pr\{\text{長さ } r \text{ 以上の連が現れる}\} - \Pr\{\text{長さ } r+1 \text{ 以上の連が現れる}\}$$

$$= \frac{1}{r!} - \frac{1}{(r+1)!} .$$

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。

上昇連長	発生回数	理論回数
1	3709	3687.0
2	2461	2458.0
3	898	921.75
4	242	245.80
5 以上	64	61.45

CHITEST : 0.9229

下降連長	発生回数	理論回数
1	3739	3702.50
2	2479	2468.33
3	912	925.63
4	224	246.83
5 以上	51	61.71

CHITEST : 0.3335

検定(VI) 4 枚の 0〜9 カードによる古典ポーカーテスト

まず，通常の５枚の 0〜9 カードによる古典ポーカーテストを説明する。数字 $0, 1, \cdots, 9$ のうちの何れか１つが書いてある $5n$ 枚のカードを並べる。カード上にある各数字を

$$Y_0, Y_1, Y_2, Y_3, Y_4, \cdots\cdots, Y_{5n-5}, Y_{5n-4}, Y_{5n-3}, Y_{5n-2}, Y_{5n-1}$$

とする。このとき，5 枚のカードの組

$$\{ Y_{5j}, Y_{5j+1}, Y_{5j+2}, Y_{5j+3}, Y_{5j+4} \}, \quad j = 0,1,\cdots,n-1,$$

について，以下の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)。

```
All different  : abcde
     One pair  : aabcd
    Two pairs  : aabbc
Three of a kind  : aaabc
    Full house  : aaabb
 Four of a kind  : aaaab
 Five of a kind  : aaaaa
```

各パターンの出現確率は計算できるので，実際の現れ方と比較して，χ^2 検定を行うことができる。

我々の発生した数値は４桁の乱数であった。４桁を構成する各数字を，4 枚のカード上の各数字とみなして古典ポーカーテストを行うことができる。このとき，現れるパターンとその組み合わせの数は以下のようになる：

	組み合わせパターンの数
All different : abcd	$10 \cdot 9 \cdot 8 \cdot 7 = 5040$
One pair : aabc	$10 \cdot 9 \cdot 8 \times 3 \times 2 = 4320$
Three of a kind : aaab	$10 \cdot 9 \times 4 = 360$
Two pairs : aabb	$10 \cdot 9 \times 3 = 270$
Four of a kind : aaaa	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 検定を行えばよい。

	出現回数	理論回数
All different	10001	10080
One pair	8730	8640
Three of a ki	718	720
Two pairs	522	540
Four of akind	29	20

CHITEST : 0.1839

検定(VII) 遅れ k の系列相関テスト

発生乱数を $U_0, U_1, U_2, \cdots\cdots, U_{n-1}$ 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

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

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 $\sigma_{XY}, \sigma_X, \sigma_Y$ は,

$$X = (X_1, X_2, \cdots, X_n), \qquad Y = (Y_1, Y_2, \cdots, Y_n)$$

であるとき

$\sigma_{XY} : X, Y$ の共分散

$$\sigma_{XY} = \frac{1}{n}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)\left(Y_i - \overline{Y}\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(X_iY_i - \overline{X}Y_i - X_i\overline{Y} + \overline{X}\overline{Y}\right) = \left(\frac{1}{n}\sum_{i=1}^{n}X_iY_i\right) - \overline{X}\overline{Y}$$

$\sigma_X : X$ の標準偏差，　$\sigma_Y : X$ の標準偏差

$$\sigma_X^2 = \frac{1}{n}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\left(X_i^2 - \overline{X}X_i - X_i\overline{X} + \overline{X}^2\right)$$

$$= \left(\frac{1}{n}\sum_{i=1}^{n}X_i^2\right) - \overline{X}^2$$

$$\sigma_Y^2 = \left(\frac{1}{n}\sum_{i=1}^{n}Y_i^2\right) - \overline{Y}^2$$

である。

註： $n\sigma_X^2$，$n\sigma_Y^2$ は，それぞれベクトル

$$(X_1-\overline{X},X_2-\overline{X},\cdots,X_n-\overline{X}),\quad (Y_1-\overline{Y},Y_2-\overline{Y},\cdots,Y_n-\overline{Y})$$

のノルムの２乗であり，$n\sigma_{XY}$ は，上の２つのベクトルの内積であることを注意しておく。

よって，

$$r=\frac{\sigma_{XY}}{\sigma_X\sigma_Y}=\frac{n^2\sigma_{XY}}{\sqrt{n^2\sigma_X^2}\sqrt{n^2\sigma_Y^2}}$$

$$=\left\{\left(\frac{1}{n}\sum_{i=1}^{n}X_iY_i\right)-(n\overline{X})(n\overline{Y})\right\}\Big/\left[\left\{\left(n\sum_{i=1}^{n}X_i^2\right)-(n\overline{X})^2\right\}\left\{\left(n\sum_{i=1}^{n}Y_i^2\right)-(n\overline{Y})^2\right\}\right]^{1/2}$$

となる。

[註]

＊ Schwarz の不等式

$$\left(\sum_{i=1}^{n}a_ib_i\right)^2\le\left(\sum_{i=1}^{n}a_i^2\right)\left(\sum_{i=1}^{n}b_i^2\right)$$

により $r\le1$ である。

＊ Schwarz の不等式で等号が成り立つのは，すべての i について「$a_i:b_i=$一定」が成り立つときであるから，$r=1$ は，すべての i について

$$(X_i-\overline{X}):(Y_i-\overline{Y})=\text{一定}$$

すなわち，平面上の点 $((X_i-\overline{X}),(Y_i-\overline{Y}))$ が，原点を通るある直線上に並ぶ（ばらつきがない）ことを意味している。

＊ また，$X_i,i=1,\cdots,n,$ と $Y_i,i=1,\cdots,n,$ が独立のとき

$$\sum_{i=1}^{n}(X_i-\overline{X})(Y_i-\overline{Y})=\left\{\sum_{i=1}^{n}(X_i-\overline{X})\right\}\left\{\sum_{i=1}^{n}(Y_i-\overline{Y})\right\}=0$$

となるので，$r=0$ となる。

ここで $X_i=U_{i-1}$，$Y_i=U_i$ とおけば，遅れ 1 の系列相関係数 C が得られる。（$U_{n+i}=U_i$ とする。）

また，$X_i = U_{i-1}$ ，$Y_i = U_{i+k-1}$ とおけば，遅れ k の系列相関係数が得られる。

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(Knuth の本[Ku])。よって n=20000 のとき，$[\mu_n - 2\sigma_n, \mu_n + 2\sigma_n]$ は [-0.01419, 0.01409] となる。我々の SSR 乱数については,

$$C = -0.008182 \in [-0.01419, 0.01409]$$

であった。ちなみに，遅れ２，３，４の系列相関係数は,

遅れ２：0.01005　　遅れ３：-0.00457　　遅れ４：-0.002421

であり，これらの値も [-0.01419, 0.01409] に入っている。

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,

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

となる。$m(m-1)\cdots(m-n+c+1)$ 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	0.011	0.049	0.184	0.491	0.89	0.948	0.961	0.992

よって，衝突回数が 61 回以下ならテスト OK とする。

我々の SSR 乱数の衝突回数は 47 回であった。

衝突の確率は Knuth の本[Ku] 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。

```
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;
```

◎以上の追加検定による結果を表にしてまとめておく。

発生法	上昇連	下昇連	poker	相関(遅延1)	相関(遅延2)	衝突
Delphi	0.1172	0.3377	0.5742	0.005928	-0.001413	49
Horner	0.9360	00288	01371	-0.000238	0.004387	51
実数シフト	0.9229	0.3335	0.1839	-0.008182	0.001005	47

以上の検定結果から，生成した２万個の４桁整数列は乱数列であるということが出来よう。しかしながら，このＳＳＲアルゴリズムが本当に有効であることを確認するには，ＳＳＲアルゴリズムに基づいて多量の数を発生させ（→生成効率の改良と長周期化が必要），その乱数性をＮＩＳＴによる検定[Www1]などで本格的に検定する必要がある。

８．非再帰的な乱数生成法

以上の議論で，検定に供された２万個の４桁整数は，$x_k \in [0,1)$ から単純実数シフト計算 $\Phi_{x_k}^{24}(1)$ を行い，得られた数値の上位３桁の数字を捨て，残った桁から続けて 4 桁の数をとったものであった。このように，k 番目の乱数がその場で直接計算される乱数生成法は，「**非再帰的な乱数生成法**」と言われ，従来からの乱数生成法が，既に作成した乱数値を利用して次の乱数値を計算する方法（再帰的な乱数生成法）とは対照的な方法である。非再帰的な乱数生成法は，乱数の生成を分割（幾つかの計算機で分担生成）することが容易なので，並列計算に適していると言えよう。

９．ＳＳＲ法乱数の生成効率の改良と長周期化(SSRex)

単純実数シフト計算（ＳＳＲ計算）は，

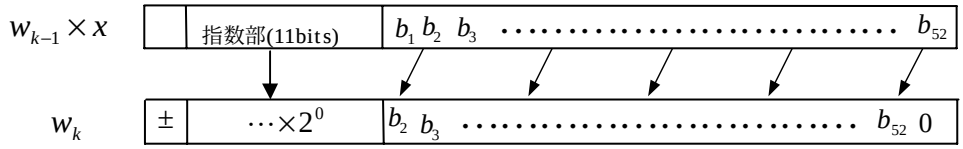
$$\Phi_x^{24}(w_0) = w_0 \circ \underbrace{x \circ x \circ x \circ \cdots\cdots \circ x \circ x}_{24}, \qquad x \in [1,2)$$

を

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

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) 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 $\Phi_x^{24}(w_0)$ 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。

上位３桁を棄て続く４桁を取り出すと言う操作は，倍精度変数 $\Phi_x^{24}(w_0)$ の仮数部のほぼ半分を乱数生成に利用するものである。（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。）この節では，この点を改良して，仮数部全体の情報を利用することにより，

１回のＳＳＲ計算で生成する乱数の桁数を 10 進８桁

として，生成効率のアップを図ることにする。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)］

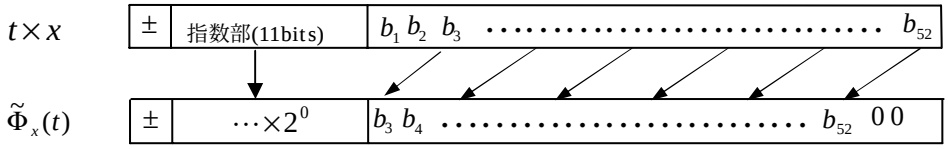
以下の計算を，$t, x \in [1,2)$ の拡張**単純実数シフト計算**といい，$\tilde{\Phi}_x(t)$ で表すことにする。

拡張ＳＳＲ計算(SSRex 計算) $\tilde{\Phi}_x(t)$

i） $t, x \in [1,2)$ に対し $t \times x$ を倍精度計算し，

ii）$t \times x$ の仮数部の全ビットを２ビット左にシフトし，

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



拡張ＳＳＲ計算では，ii) における仮数部の左シフトが，２ビット分の桁落ちになっている。
この $\tilde{\Phi}_x$ を２６回繰り返し使って，１度に１０進８桁の乱数を生成する。

拡張実数シフト(SSRex)乱数生成法

(1) $x \in [1,2)$ に対し，$\tilde{\Phi}_x^{26}(1)$ を計算する。

(2) $\tilde{\Phi}_x^{26}(1)$ の上位５桁を棄て，続く８桁を**乱数値**とする。

(3) x を $x_1, x_2, x_3, \cdots$ と $[1,2)$ 内で一様に変化させて，多くの乱数列を得る。

$\tilde{\Phi}_x$ では，２ビットの仮数部シフトをおこなうので，$\tilde{\Phi}_x^{26}$ では，計５２ビットのシフトがなされる。この５２ビットの決定にあたっては，倍精度変数の仮数部の大きさが５２ビットであることを考慮している。なお，$\tilde{\Phi}_x^{26}$ 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} を使えばよい。

［SSRex 乱数生成法の長周期化］

拡張実数シフト(SSRex)乱数生成法の (3) で，x を $x_1, x_2, x_3, \cdots$ と $[1,2)$ 内で一様に変化させるとしたが，具体的には以下のようにすれば良い。

まず，$1.e$ で数 1.27181... を表すことにし（$e = 2.7181\cdots$；ネイピアの数），その２進表現を $1.\hat{r}_1\hat{r}_2\hat{r}_3\cdots\hat{r}_{31}\cdots$ とする。次に $p=$, $r=$とする（p, r はいずれも素数である）。そして $k = 0,1,2,\cdots$, に対して $rk \mod p$ を計算し，その２進表現を $\nu_1\nu_2\cdots\nu_{31}$ とする。このとき x_k を

$$x_k = 1.(\hat{r}_1 \oplus \nu_1)(\hat{r}_2 \oplus \nu_2)\cdots(\hat{r}_{31} \oplus \nu_{31})\hat{r}_{32}\hat{r}_{33}\cdots$$（⊕は，排他的論理和 XOR）

で定める。この x_k に対して $\tilde{\Phi}^{26}_{x_k}(1)$ を計算し，k 番目の乱数を取り出せばよい。この方法での乱数周期は，もちろん p になる。

拡張実数シフト乱数生成法は, 乗算後の仮数部のシフトの大きさを変化させてもＯＫであることを示しており，この事実はＳＳＲアルゴリズムを完全整数演算化する際に使われる。

１０.単純実数シフト法（ＳＳＲ法）の理論的解析

単純実数シフト乱数生成法（Simplified Shift-Real method, ＳＳＲ法）のアルゴリズムはかなり単純であるため，生成する乱数値の分布に関する数理的な解析が可能である。以下に紹介する。

１０.１ 単純実数シフト計算（ＳＳＲ計算）の数式表現

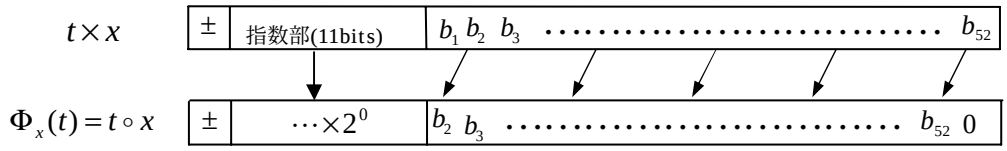
単純実数シフト擬似乱数生成法（ＳＳＲ法）は，以下の単純実数シフト計算(SSR 計算)の繰り返しであった。

単純実数シフト計算（ＳＳＲ計算）$\Phi_x(t)$

i)　$t, x \in [1,2)$ に対し $t \times x$ を倍精度計算し，

ii) $t \times x$ の仮数部の全ビットを１ビット左にシフトし，

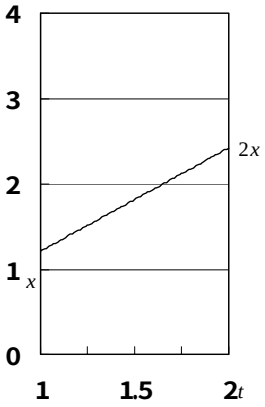
iii) $t \times x$ の指数部を $\cdots \times 2^0$ となるように設定する。



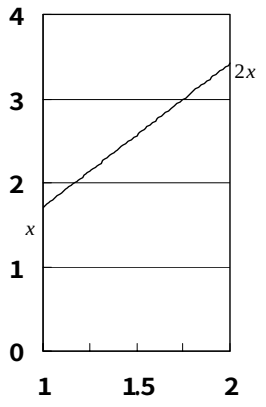
ＳＳＲ計算中の ii) と iii) は順序を問わないので，以下の理論的解析では iii) を先に行い，続いて ii) を行うこととして考察する。

写像 $\varphi_x, \varphi_e, \varphi_s$ を以下のように定める：

$$\varphi_x : [1,2) \to [1,4), \quad \varphi_x(t) = tx, \quad (x \in [1,2))$$

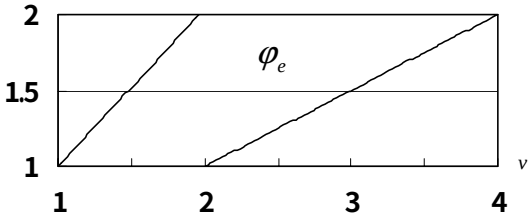


φ_x　$(1 \le x < 1.5)$

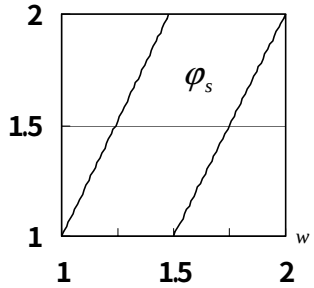


φ_x　$(1.5 \le x < 2)$

$$\varphi_e : [1,4) \to [1,2), \quad \varphi_e(v) = \begin{cases} v & if \quad 1 \le v < 2 \\ v/2 & if \quad 2 \le v < 4 \end{cases}$$



$$\varphi_s : [1,2) \to [1,2), \quad \varphi_s(w) = \begin{cases} 2w-1 & if \quad 1 \le w < 1.5 \\ 2w-2 & if \quad 1.5 \le v < 2 \end{cases}$$



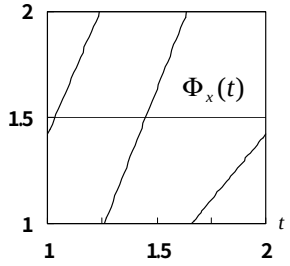
φ_x はＳＳＲ計算 i）の x による掛け算に相当し，φ_e は iii）の指数部を …$\times 2^0$ に設定する操作，φ_s は ii）の仮数部の左シフトに相当する写像である。このとき，ＳＳＲ計算で t から $\Phi_x(t)$ を得る一連のプロセスは,

$$\Phi_x(t) \ = \ (\varphi_s \circ \varphi_e \circ \varphi_x)(t)$$

と表される。したがって，$\Phi_x(t)$ は具体的に

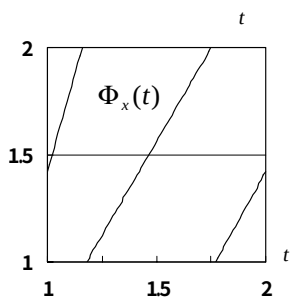
$1 \le x < 1.5$ のとき

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

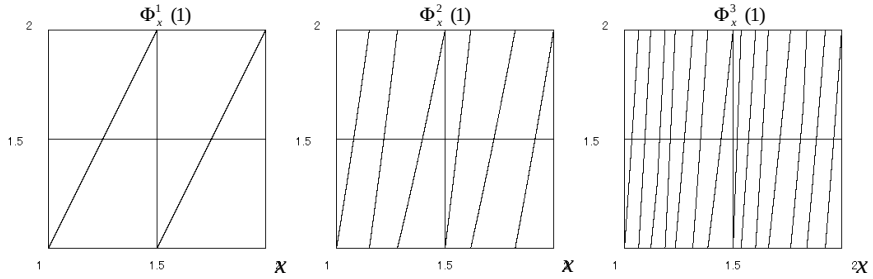


$1.5 \le x < 2$ のとき

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



と表される。SSR 乱数は，$\Phi_x^{24}(1)$ で，x を変化させて多くの乱数値を得ている。$\Phi_x^n(1)$ を x の関数とみて，$\Phi_x^1(1), \Phi_x^2(1), \Phi_x^3(1)$ のグラフを描いてみると以下の様になる。



図および Φ_x の数式表現から，$\Phi_x^{n-1}(1)$ から $\Phi_x^n(1)$ に至る過程 $\varphi_s \circ \varphi_e \circ \varphi_x (= \Phi_x)$ 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 $\Phi_x^n(1)$ における分岐数を γ_n とすると

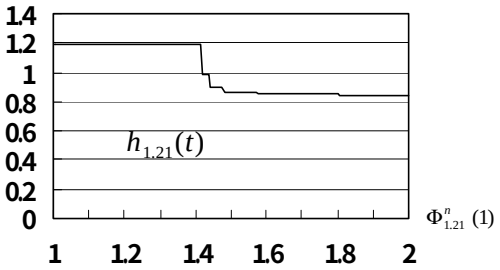
$$\begin{cases} \gamma_0 = 0 \\ \gamma_n = 2(\gamma_{n-1} + 1), \quad n = 1,2,\cdots,24 \end{cases}$$

の関係がある。よって，$\Phi_x^{24}(1)$ における分岐本数は，若干の計算をすれば

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

であることが分かる。SSR 乱数の作成過程 $\Phi_{x_k}^{24}(1), k = 1,2,\cdots,$ では，毎回この 33,554,430 本の分岐の中の１つの分岐について $\Phi_{x_k}^{24}(1)$ の値を得ていることになる。

ともあれ，SSR 乱数の基本となる計算は，各 $x_k \in [1,2)$ ごとの $\Phi_{x_k}^{24}(1)$ の計算であるから，写像 Φ_x が繰り返されたときの性質, すなわち Φ_x のエルゴード的性質の理論的な解明が必要となる。例えば，x=1.21 における，値 $\Phi_{1.21}^n(1), n = 1,2,\cdots,10000000$ の理論分布は，以下で



のようになると解析される。

［**註**］ $\frac{d}{dt}\Phi_x(t) = x$ または $2x$ であり，特に $\frac{d}{dt}\Phi_x(t) > 1$， $x \in [1,2)$, 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 $\Phi_x^n(1)$，$n = 1,2,\cdots,$ がカオス的様相 [Bo][De][Ro] を呈する。Φ_x のカオス性については，１２節で扱う。

１０.２ 写像 Φ_x に対する Perron-Frobenius 作用素と不変測度

X=[1,2) とし，X 上の Borel 集合体を B， (X,B) 上の Lebesgue 測度をλ, (X,B,λ) 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。

$\Phi_x(t) = (\varphi_s \circ \varphi_e \circ \varphi_x)(t)$ は,

$$\Phi_x : X \to X, \quad \Phi_x \in L^1, \quad \frac{d}{dt}\Phi_x(t) = x \text{ または } 2x$$

であるが， Φ_x の不変測度 μ_x の密度関数 h_x は，Perron-Frobenius 作用素 $L_x : L^1 \to L^1$,

$L_x : h \to (L_x h)$, ただし,

$1 \le x < 1.5$ のとき
$$(L_x h)(t) = \begin{cases} \frac{1}{2x} h(\frac{t+2}{2x}) + \frac{1}{x} h(\frac{t+1}{x}) & (1 \le t < 2x-1) \\ \frac{1}{2x} h(\frac{t+1}{2x}) + \frac{1}{2x} h(\frac{t+2}{2x}) & (2x-1 \le t < 2) \end{cases}$$

$1.5 \le x < 2$ のとき
$$(L_x h)(t) = \begin{cases} \frac{1}{x} h(\frac{t+1}{x}) + \frac{1}{x} h(\frac{t+2}{x}) & (1 \le t < 2x-2) \\ \frac{1}{2x} h(\frac{t+2}{2x}) + \frac{1}{x} h(\frac{t+1}{x}) & (2x-2 \le t < 2) \end{cases}$$

の不動点（$L_x h_x = h_x$ なる関数 h_x ）で与えられる [Po]。

【Φ の不変測度μ 】

$\forall A \in (X,B)$ ； $\mu(\Phi^{-1}(A)) = \mu(A)$

を満たす (X,B) 上の測度μを，Φの不変測度という。

【Φ に対応する Perron-Frobenius 作用素 L 】

$g \in L^1$ に対し $Lg \in L^1$ を

$$\int f(\Phi(x))g(x)dx = \int f(t)(Lg)(t)dt, \quad \forall f \in L^\infty$$

を満たす関数として定義する。適当な条件のもとで

$$(Lg)(t) = \sum_{y \in \Phi^{-1}(t)} \frac{g(y)}{|\Phi'(y)|}$$

と表される。

h_x が存在することは Φ_x の性質($\frac{d}{dt}\Phi_x(t) = x$ or $2x > 1$， expansive) から分るが，h_x を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），（註：つい最近 [Go] に式があるらしいことを知った）

$$h_x(t) = \lim_{n\to\infty}\left\{\frac{1}{n}\sum_{k=0}^{n-1} L_x^k \mathbf{1}\right\}(t)$$

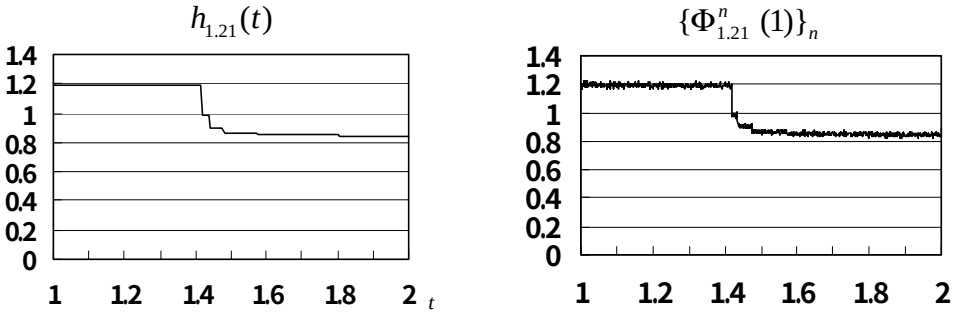
なる関係式を使って数値計算で求める([Bo][LY])。ここで，$\mathbf{1}$ は恒等的に１をとる関数である。実際の計算に当たっては，n=200 とし，区間 [1,2) を 集合 $\left\{c_i = 1+\frac{i}{10000} \mid i = 0,1,2,\cdots,9999\right\}$ で離散近似して，$h_x(c_i)$, $i = 0,1,\cdots,9999$, を求める。以下にx=1.21について h_x のグラフを示しておく（左図）。また数値計算で得られた h_x のグラフの妥当性を確認するために，同じ x の値について，

$$\Phi_x^n(1),\ n = 1,2,\cdots,10000000,$$

が，

区間 [1+0.001×l, 1+0.001×(l+1)), l =0,1, … , 999,

に入る回数を 10000 で割った値のグラフも示しておく（右図，1+0.001×l を X 軸にとっている）。



１０.３ ＳＳＲ計算値 $\{\Phi_{x_k}^{24}(1)\}_k$ の分布

x_k ， $k = 0,1,2,\cdots$， が [1,2) 内を一様に分布するように変化するとき，ＳＳＲ計算値 $\Phi_{x_k}^{24}(1)$ の分布がどのようになるか考えてみる。

まず

p=49933453, q=22801201, r=491377, s=47513,

a=1920000, b=48060000 （註：p, q, r, s はいずれも素数）

とし，

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

を計算し，

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

とする。この x_k は，周期

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

で，厳密ではないが，[1,2) を一様に分布するようになっている。久保氏のアイデアは，

> **[主張]** $\{\Phi^q_{x_k}(1)\}_{k=0,1,2,\cdots}$ の分布は，x_k が [1,2) 上を一様に分布する時，q が十分大きければ，前出の h_x , $x\in[1,2)$, を x について平均した密度関数
>
> $$H(t)=\int_1^2 h_x(t)dx\,,\qquad t\in[1,2)\,,$$
>
> で記述される，

というものである。この主張は，$H(t)$ の満たすべき性質に関する次のような推論に基づく。

q が十分大きければ，$\{\Phi^q_{x_k}(1)\}_{k=0,1,2,\cdots}$ の分布は，x_k が [1,2) 上を一様に分布する時，q の値にかかわらず安定状態に達し，安定状態に達した $\{\Phi^q_{x_k}(1)\}_{k=0,1,2,\cdots}$ 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，

$$\frac{1}{N}\sum_{k=0}^{N-1} f(\Phi^q_{x_k}(1)) \approx \int_1^2 f(t)H(t)dt$$

と近似されると考えられる。この式の両辺を，$q=Q, Q+1, \ldots, Q+R-1$ について和をとり平均すれば，

$$\frac{1}{N}\sum_{k=0}^{N-1}\frac{f(\Phi^Q_{x_k}(1))+f(\Phi^{Q+1}_{x_k}(1))+\cdots+f(\Phi^{Q+R-1}_{x_k}(1))}{R} \;\approx\; \int_1^2 f(t)H(t)dt$$

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

$$\frac{1}{N}\sum_{k=0}^{N-1}\int_1^2 f(t)h_{x_k}(t)dt$$

に近づくので，

$$\int_1^2 f(t)\left(\frac{1}{N}\sum_{k=0}^{N-1}h_{x_k}(t)\right)dt \;\approx\; \int_1^2 f(t)H(t)dt$$

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，

$$\int_1^2 f(t)\left(\frac{1}{N}\sum_{k=0}^{N-1}h_{x_k}(t)\right)dt \quad\to\quad \int_1^2 f(t)\left(\int_1^2 h_x(t)dx\right)dt$$

となり

$$\int_1^2 f(t)(\int_1^2 h_x(t)dx)dt \;=\; \int_1^2 f(t)H(t)dt$$

を得る。f は任意であるので，結局，

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

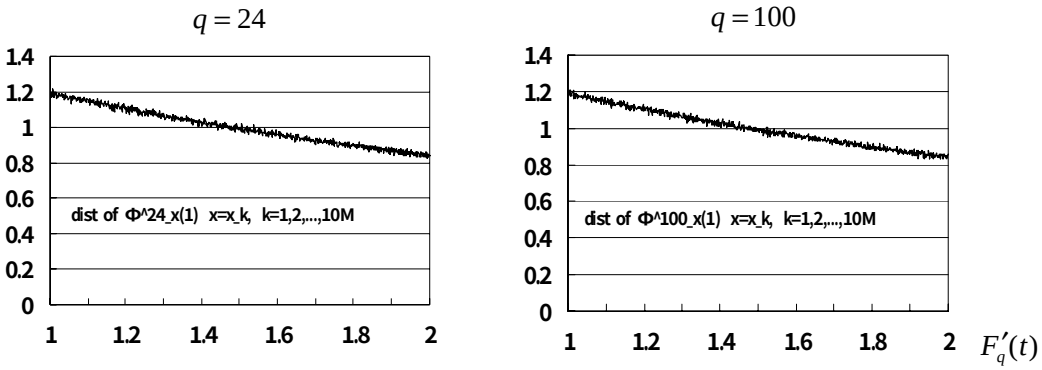
となる。

もしこの［主張］に証明を与えるとすれば，以上の推論に，極限操作に関する数学的厳密さを付加することになろう。

次に，この［主張］を数値実験で確かめてみる。まず q が大きければ，$\{\Phi^q_{x_k}(1)\}_{k=0,1,2,\cdots}$ の分布が安定状態に達することを見てみる。以下は，q=24 および q=100 に対する

$$\Phi^q_{x_k}(1),\quad k=1,2,\cdots,10000000$$

の分布のグラフである。これから，q = 24, 25, 26, ... に対して，$\{\Phi^q_{x_k}(1)\}_{k=0,1,2,\cdots}$ の分布は，十分安定していると考えられる。



x_k が $[1,2)$ を一様に変化するとき，$\{\Phi^q_{x_k}(1)\}_{k=0,1,2,\cdots}$ の確率分布は，x が $[1,2)$ を動いたときの $\Phi^q_x(1)$ の確率分布，すなわち $F_q(t)=\left|\{x\in[1,2)\,|\,\Phi^q_x(1)<t\}\right|$ になると考えられる（$|\cdot|$ はルベーグ測度）。q が大きいとき，$\Phi^q_x(1)$, $x\in[1,2)$ のグラフの各分枝（非常にたくさんある）はほぼ垂直になるので，$F_q(t)$ の t に関する増加率（密度関数）はほぼ一様で１に近くなる（上のグラフから 0.8~ 1.2）と考えられる。$F'_q(t)$ は，１０．１節の $\Phi^q_x(1)$ のグラフにおいて，各分枝上で $\Phi^q_x(1)=t$ となる x における $\Phi^q_x(1)$ の微分係数の逆数を，すべての分枝について和をとったものである。q が大きくなったとき，$\{\Phi^q_{x_k}(1)\}_{k=0,1,2,\cdots}$ の分布が安定状態に達するとは，q が大きくなったとき $F_q(t)$，$F'_q(t)$ が収束することを意味している。

続いて，$H(t)$ 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，

$$l_s = 1.0 + \frac{s}{997}\ ,\quad s = 1,2,\cdots,996,$$

を選ぶ。次に，前節と同様にして

$$h_{l_s}(c_i),\ i = 0,1,\cdots,9999,$$

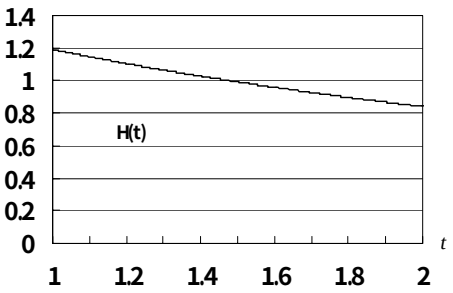
を求める。そして，$h_{l_s}(c_i),\ s = 1,2,\cdots,996,$ を平均して $H(c_i)$ を求める：

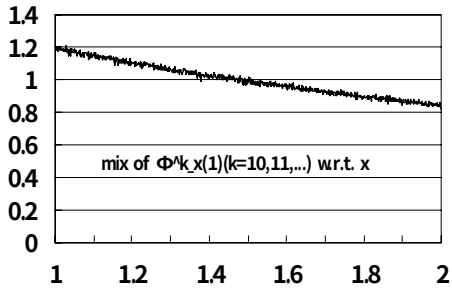
$$H(c_i) = \frac{1}{996}\sum_{s=1}^{996} h_{l_s}(c_i),\qquad i = 0,1,\cdots,9999.$$

$H(c_i),\ i = 0,1,\cdots,9999,$ のグラフは下図左のようになる。このグラフの妥当性確認するため，実際に，

$$\Phi_{l_s}^{q}(1),\qquad s = 1,2,\cdots,996,\qquad q = 10,11,\cdots,10009$$

を計算し，得られた 996×10000 個の値　$\Phi_{l_s}^{q}(1) = 1.d_1d_2d_3\cdots$　の分布を下図右に示しておく（縦軸は $1.d_1d_2d_3\cdots$ 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，［主張］どおりである。念のため，次節で，$\{\Phi_{x_k}^{q}(1)\}_{k=0,1,2,\cdots}$ の分布が $H(t)$ で既述されることについて，統計的検定を行う。

１０．４　ＳＳＲ計算値 $\{\Phi_{x_k}^{24}(1)\}_k$ の分布の検定

前節で，$\{\Phi_{x_k}^{24}(1)\}_k$ の分布が，密度関数 $H(t)$ で記述される，という主張を得たので，これを統計的に検定してみる。検定には，検定(IV)で使われた Kolmogorov-Smirnov 統計量を大量に発生させて，その分布を調べる。

検定(IX) Kolmogorov-Smirnov 統計量の検定

20000 個の $z_k = \Phi_{x_k}^{24}(1)$ からなる数値列

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

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

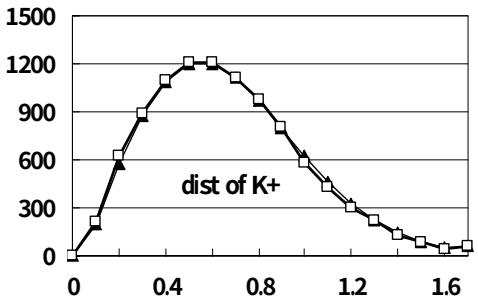
$$F(t)=\left\{\sum_{i:t_i<t}H(t_i)\right\}\Big/10000\,,\quad t_i=1+\frac{1}{10000},\quad i=0,1,2,\cdots,9999$$

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^-] , $j=0,1,\cdots,9999,$ の分布は，確率分布関数

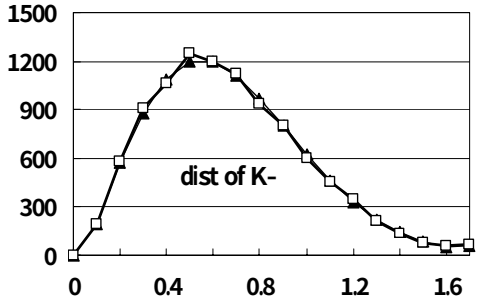
$$P(x)=1-\exp\left(-2x^2\right)$$

にほぼしたがうことが知られているので[Ku] ，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，

$\{\Phi_{x_k}^{24}(1)\}_k$ 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。

１０.５ ＳＳＲ計算値 $\{\Phi_{x_k}^{24}(1)\}_k$ の分布 $H(t)$ を近似する関数 $\tilde{H}(t)$

ＳＳＲ計算値 $\{\Phi_{x_k}^{24}(1)\}_{k=0,1,2,\cdots}$ 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。ここでは， $H(t)$ を近似する関数 $\tilde{H}(t)$，

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

を求めてみる。

ＳＳＲ計算の基礎となる写像 $\Phi_x:[1,2)\to[1,2)$ は $\Phi_x=\varphi_s\circ\varphi_e\circ\varphi_x$ と表されるが，φ_s を

を除いた写像

$$\Psi_x(t) = (\varphi_e \circ \varphi_x)(t), \quad t \in [1,2)$$

は,

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

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

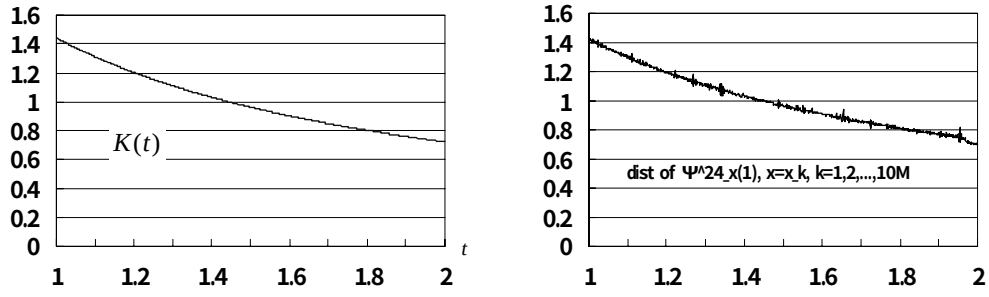
$\tilde{L}_x : k \to (\tilde{L}_x k)$, ただし,

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

となる。久保氏は, $\tilde{L}_x$ 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（この事実の確認は容易である）。すると，今までの議論を再現すれば, $\{\Psi_{x_k}^{24}(1)\}_{k=0,1,2,\cdots}$ の分布は, $\Psi_x(t)$ の不変測度の密度関数 $k_x(t) = K(t)$, $t \in [1,2)$, 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 10.3 節と同様にして $K(t)$ のグラフと, $\Psi_{x_k}^{24}(1), \quad k = 1,2,\cdots,10000000$ の分布のグラフを作ってみると，そのことが理解される。



なお，上のグラフの曲線の傾きは, $\Phi_x = \varphi_s \circ \varphi_e \circ \varphi_x$ の場合に比してきつい。これは $\Psi_x = \varphi_e \circ \varphi_x$ では，仮数部のシフト φ_s が無いためである。

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

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

と書けることに注意する。ここで

シフト φ_s に対応する Perron-Frobenius 作用素を L_{φ_s}

とするとき，$\Phi_x = \varphi_s \circ \varphi_e \circ \varphi_x$ ，$\Psi_x = \varphi_e \circ \varphi_x$ であるので，Φ_x , Ψ_x に対応する Perron-Frobenius 作用素 L_x , $\tilde{L}_x$ の関係は

$$L_x = L_{\varphi_s} \circ \tilde{L}_x$$

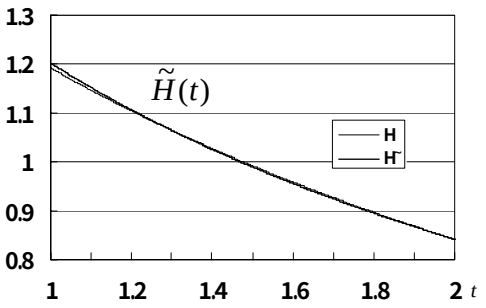
である。そこで，10.3 節 の

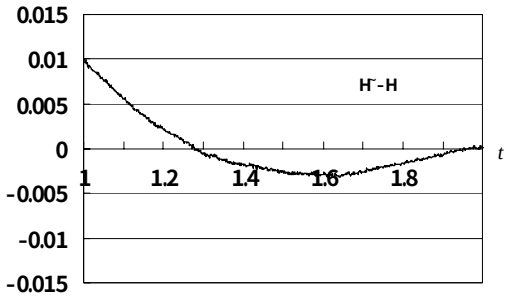
$$H(t) = \int_1^2 h_x(t)dx = \int_1^2 (L_x h_x)(t)dx \quad (\because\ L_x h_x = h_x)$$

において, 少々乱暴であるが $h_x(t) \cong K(t) = \dfrac{1}{t\log 2}$ と考えると（これは, 集合 $\{\Phi_{x_k}^{23}(1)\}_{k=0,1,2,\cdots}$ 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），

$$\begin{aligned} H(t) &\cong \int_1^2 (L_x K)(t)dx = \int_1^2 ((L_{\varphi_s} \circ \tilde{L}_x)K)(t)dx = \int_1^2 (L_{\varphi_s} K)(t)dx \\ &= \int_1^2 \frac{1}{2}\left\{K\left(\frac{1}{2}+\frac{t}{2}\right)+K\left(1+\frac{t}{2}\right)\right\}dx \\ &= \left\{\frac{1}{1+t}+\frac{1}{2+t}\right\}\frac{1}{\log 2} \\ &= \tilde{H}(t) \end{aligned}$$

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





１０.６ 分布を近似する関数 $\tilde{H}(t)$ 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 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 $L_x : L^1 \to L^1$ は,

$L_x : h \to (L_x h)$, ただし,

$1 \le x < 1.5$ のとき

$$(L_x h)(t) = \begin{cases} \frac{1}{2x} h(\frac{t+2}{2x}) + \frac{1}{x} h(\frac{t+1}{x}) & (1 \le t < 2x-1) \\ \frac{1}{2x} h(\frac{t+1}{2x}) + \frac{1}{2x} h(\frac{t+2}{2x}) & (2x-1 \le t < 2) \end{cases}$$

$1.5 \le x < 2$ のとき

$$(L_x h)(t) = \begin{cases} \frac{1}{x} h(\frac{t+1}{x}) + \frac{1}{x} h(\frac{t+2}{x}) & (1 \le t < 2x-2) \\ \frac{1}{2x} h(\frac{t+2}{2x}) + \frac{1}{x} h(\frac{t+1}{x}) & (2x-2 \le t < 2) \end{cases}$$

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,

$$(L_x h)(t) = \begin{cases} \frac{1}{2x} h(\frac{t+1}{2x}) + \frac{1}{2x} h(\frac{t+2}{2x}) & (1 \le x \le \frac{t+1}{2}) \\ \frac{1}{2x} h(\frac{t+2}{2x}) + \frac{1}{x} h(\frac{t+1}{x}) & (\frac{t+1}{2} < x \le \frac{t+2}{2}) \\ \frac{1}{x} h(\frac{t+1}{x}) + \frac{1}{x} h(\frac{t+2}{x}) & (\frac{t+2}{2} < x < 2) \end{cases}$$

と表されるので，$(L_x h)(t)$ の x についての平均 $\int_1^2 (L_x h)(t)dx$ は,

$$\int_1^{(t+1)/2} \left\{ \frac{1}{2x} h(\frac{t+1}{2x}) + \frac{1}{2x} h(\frac{t+2}{2x}) \right\} dx + \int_{(t+1)/2}^{(t+2)/2} \left\{ \frac{1}{2x} h(\frac{t+2}{2x}) + \frac{1}{x} h(\frac{t+1}{x}) \right\} dx + \int_{(t+2)/2}^{2} \left\{ \frac{1}{x} h(\frac{t+1}{x}) + \frac{1}{x} h(\frac{t+2}{x}) \right\} dx$$

$$= \int_1^{(t+1)/2} \frac{1}{2x} h(\frac{t+1}{2x}) dx + \int_1^{(t+2)/2} \frac{1}{2x} h(\frac{t+2}{2x}) dx + \int_{(t+1)/2}^{2} \frac{1}{x} h(\frac{t+1}{x}) dx + \int_{(t+2)/2}^{2} \frac{1}{x} h(\frac{t+2}{x}) dx$$

$$= \frac{1}{2} \int_{2/(t+1)}^{1} \frac{1}{u} h(\frac{1}{u}) du + \frac{1}{2} \int_{2/(t+2)}^{1} \frac{1}{u} h(\frac{1}{u}) du + \int_{1/2}^{2/(t+1)} \frac{1}{u} h(\frac{1}{u}) du + \int_{1/2}^{2/(t+2)} \frac{1}{u} h(\frac{1}{u}) du$$

$$= \int_{1/2}^{1} \frac{1}{u} h(\frac{1}{u}) du + \frac{1}{2} \int_{1/2}^{2/(t+1)} \frac{1}{u} h(\frac{1}{u}) du + \frac{1}{2} \int_{1/2}^{2/(t+2)} \frac{1}{u} h(\frac{1}{u}) du$$

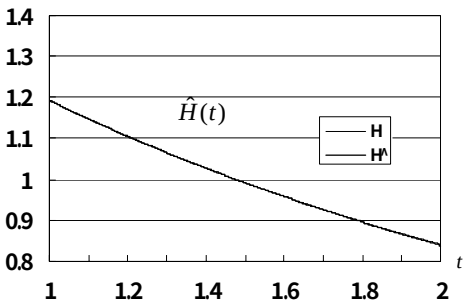
と計算される（この計算はやや面倒)。ここで，h として特に前出の

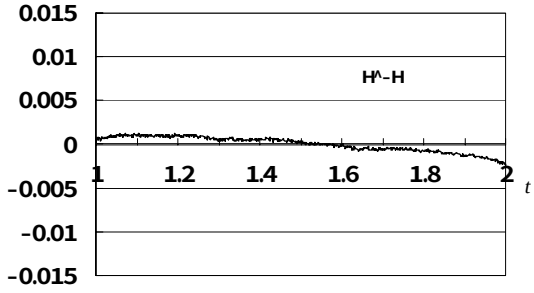
$$\tilde{H}(t) = \left\{ \frac{1}{1+t} + \frac{1}{2+t} \right\} \frac{1}{\log 2}$$

をとると,

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

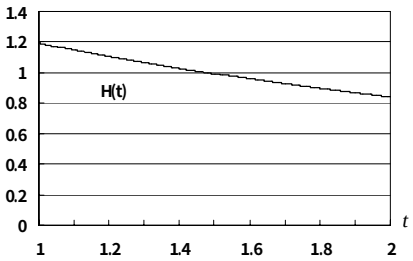
を得る。以下に，$\hat{H}(t)$ および $\hat{H}(t)-H(t)$ のグラフを示しておく。期待通り $\hat{H}(t)$ ではさらに $H(t)$ に近づいていることが見てとれる。



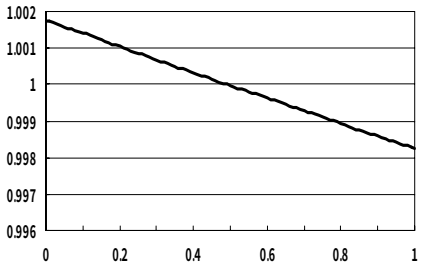


１１．ＳＳＲ乱数の分布特性の改良

ＳＳＲ法では，$\Phi_{x_k}^{24}(1)$，$k=0,1,\cdots$，から乱数列を作る際，$\Phi_{x_k}^{24}(1)$ の最初の３桁を棄てているが，ＳＳＲ計算の理論的な考察で明らかになったように，もともとの数値列 $\{\Phi_{x_k}^{24}(1)\}_{k=0,1,\cdots}$ の密度分布 $H(t)$ は，下図左のように下に凸な単調減少関数になっている。したがって，上位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（下図右）。



$\{\Phi_{x_k}^{24}(1)\}_{k=0,1,\cdots}$の分布



SSR乱SR乱数値（y軸拡大）

この節では，久保泉氏のアイデアによる以下のようなＳＳＲ乱数値の改善について述べる。

K改良ＳＳＲ乱数生成法(SSRK)

$\Phi_{x_k}^{24}(w_0)$ 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

$$\Phi_{x_k}^{24}(1.e)\ ,\quad \Phi_{x_k}^{24}(1.\tilde{\pi})$$

を得る。そして $\Phi_{x_k}^{24}(1.e)-\Phi_{x_k}^{24}(1.\tilde{\pi})\mod[1,2)$ を $\Phi_{x_k}^{24}(1)$ の代りとする。

（註）$z \bmod [1,2)$ は，z に整数 m を加えて $z+m\in[1,2)$ を得ることを表す。

この方法は，$\Phi_{x_k}^{24}(w_0)$ が初期値 w_0 に非常に鋭敏であるので（これは，後の理論的考察で示すように，Φ_x がカオス写像であることに由来している），２つの初期値 $1.e$ と $1.\tilde{\pi}$ に対する $\Phi_{x_k}^{24}(1.e)$ と $\Phi_{x_k}^{24}(1.\tilde{\pi})$ について，k を変化させた時,

$\{\Phi_{x_k}^{24}(1.e)\}_{k=0,1,\cdots}$ と $\{\Phi_{x_k}^{24}(1.\tilde{\pi})\}_{k=0,1,\cdots}$ は，同じ分布を持つ，ほぼ独立な数値列

とみなせることを巧みに利用している。$\{\Phi_{x_k}^{24}(1.e)\}_{k=0,1,\cdots}$ と $\{\Phi_{x_k}^{24}(1.\tilde{\pi})\}_{k=0,1,\cdots}$ の振舞いが互いに独立であると仮定すると，$\Phi_{x_k}^{24}(1.e)-\Phi_{x_k}^{24}(1.\tilde{\pi}) \mod [1,2)$ の分布の密度関数 $I(y)$, $y\in[1,2)$，は $H(t)$ からの畳み込み

$$I(y)=\int_1^y H(t)H(t-y+2)dt \ + \ \int_y^2 H(t)H(t-y+1)dt$$

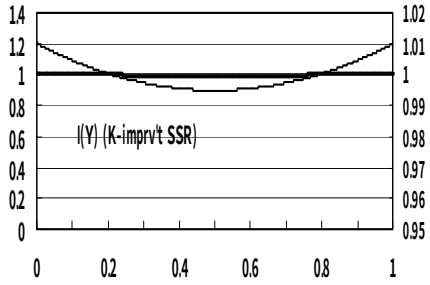
で得られる。この $I(y)$ は,

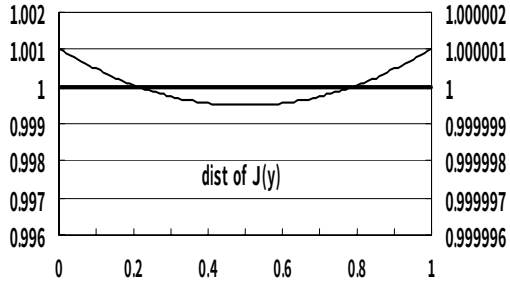
$$y_i=1+i/10000,\ i=0,1,\cdots,9999, \qquad c_j=1+j/10000,\ j=0,1,\cdots,9999,$$

について

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

と数値計算される。$I(y)$ のグラフを下図左に示す。（註：同じグラフをスケールを変えて二重表示している。右側の Y 軸が，拡大したスケール。）下図右は，$\Phi_{x_k}^{24}(1.e)-\Phi_{x_k}^{24}(1.\tilde{\pi}) \mod [1,2)$ の最初の３桁を棄てたあとの続く４桁を乱数として採用した時の分布である。特性が格段に改良されることが分かる。





１２．ＳＳＲ計算 $\Phi_x(t)$ のカオス性とマルコフ変換

一般に，距離空間 $\boldsymbol{X}$ 上の写像 $f:\boldsymbol{X}\to\boldsymbol{X}$ が以下の性質を持つとき，f は**カオス写像**であると言われる[Bo][De][Ro]：

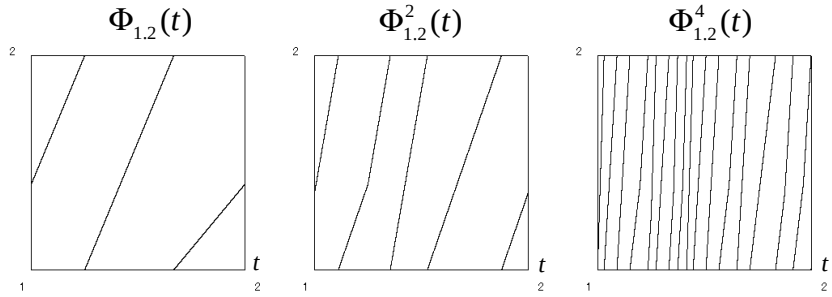
i) f は不変集合 $\boldsymbol{Y}$ 上で初期値に対する敏感な依存性をもつ，すなわち，

$$\exists r>0,\ \forall x\in\boldsymbol{Y},\ \forall\varepsilon>0,\ \exists y\in\boldsymbol{Y}(d(x,y)<\varepsilon),\ \exists k\geq 0;\ d(f^k(x),\ f^k(y))>r$$

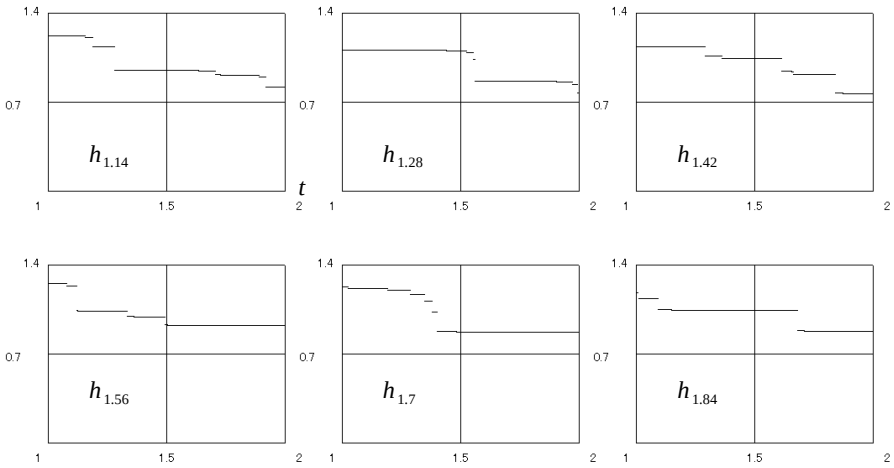
ii) f は，$\boldsymbol{Y}$ で推移的である，すなわち，点 p の正の向きの軌道が Y で稠密になる。

iii) 周期点が $\boldsymbol{Y}$ で稠密である。

このことを $f(t)=\Phi_x(t)$ について見てみよう。まず，$\frac{d}{dt}\Phi_x(t)>1$，$x\in[1,2)$，であることから，Φ_x は拡張的(expansive)な写像であり，i) であることが分かる。次に，Φ_x^n の周期点は，$y=\Phi_x^n(t)$ と直線 $y=t$ の交点で与えられるが，例えば以下の Φ_x，Φ_x^2，Φ_x^4，$x=1.2$，のグラフを見ても分かるとおり，



Φ_x^n の周期点は n が増加すると共に，[1,2) 内でほぼ一様に増えていく。これより，i) であることが分かる。また，Φ_x の不変測度の確率密度関数 h_x のグラフを見てみると，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) で一様に分布していることが分かる。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。（註： Φ_x の不変測度については [Go] の最近の結果を参照）

区間 $\boldsymbol{X}$=[1,2) 上の写像 $f:\boldsymbol{X}\to\boldsymbol{X}$ が以下の性質を持つとき，f は**マルコフ変換**であると言われる [Bo][Po]：

区間 $\boldsymbol{X}$=[1,2) のある分割 Δ

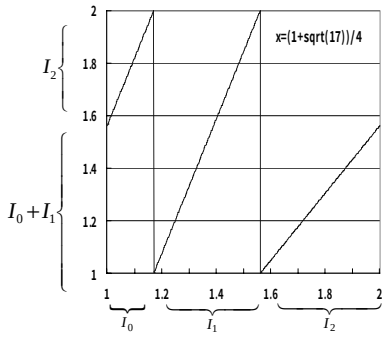
$$\Delta : a_0 = 1 < a_1 < a_2 < \cdots < a_n = 2$$

があって，f は，各区間 $I_i = (a_{i-1}, a_i)$ を，ある区間 $(a_{j(i)}, a_{k(i)})$ の上に１対１かつ連続に写像し，またその逆写像も連続である。

我々の $\Phi_x : [1,2) \to [1,2)$ は, x が特別な値をとるときマルコフ写像になるので, そのことを示す。まず,

<u>1 ≦ x < 1.5 のとき</u>

$$\Phi_x(t) = \begin{cases} 2xt-1 & (1 \le t < \dfrac{3}{2x}) \\ 2xt-2 & (\dfrac{3}{2x} \le t < \dfrac{2}{x}) \\ xt-1 & (\dfrac{2}{x} \le t < 2) \end{cases}$$



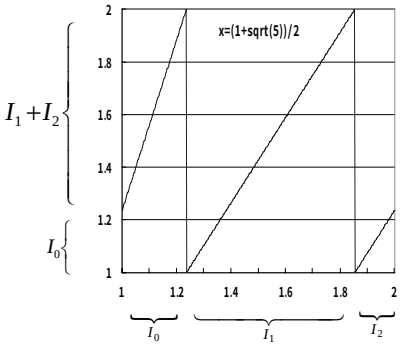
であるので，$\Phi_x(2) = 2x-1 = \dfrac{2}{x}$ であれば，$\begin{cases} \Phi_x(I_0) = I_2 \\ \Phi_x(I_1) = I_0 \cup I_1 \cup I_2 \\ \Phi_x(I_2) = I_0 \cup I_1 \end{cases}$ が

成り立つ。方程式を解けば $x = \dfrac{1+\sqrt{17}}{4} = 1.280\cdots$ であることが分る。

同様に,

<u>1.5 ≦ x < 2 のときは</u>

$$\Phi_x(t) = \begin{cases} 2xt-2 & (1 \le t < \dfrac{2}{x}) \\ xt-1 & (\dfrac{2}{x} \le t < \dfrac{3}{x}) \\ xt-2 & (\dfrac{3}{x} \le t < 2) \end{cases}$$



であるので，$\Phi_x(2) = 2x - 2 = \dfrac{2}{x}$ であれば，$\begin{cases} \Phi_x(I_0) = I_1 \cup I_2 \\ \Phi_x(I_1) = I_0 \cup I_1 \cup I_2 \\ \Phi_x(I_2) = I_0 \end{cases}$ が

成り立つ。方程式を解けば $x = \dfrac{1+\sqrt{5}}{2} = 1.6180\cdots$ であることが分る。

マルコフ変換のときは，Φ_x の不変測度を与える密度関数は非常に見やすい関数になる [Ym]。

$X = [0,1)$ とし，

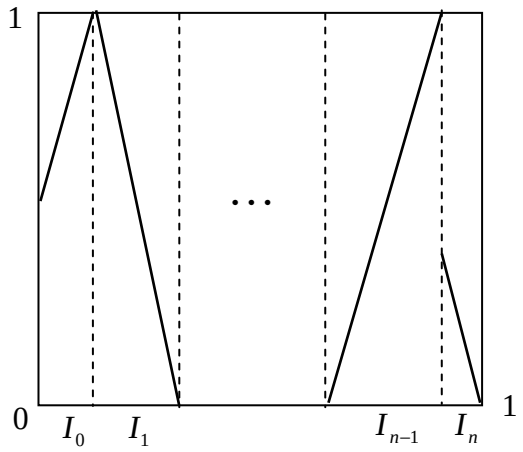
$$\Delta : a_0 = 0 < a_1 < a_2 < \cdots < a_n < a_{n+1} = 1$$

を X の分割とする。小区間を

$$I_i = [a_i, a_{i+1}),\ i = 0, \cdots, n,$$

とおき，写像 $T : [0,1) \to [0,1)$ を

$$T(X) = \begin{cases} \alpha x + (1 - \alpha a_1) & (x \in I_0) \\ \beta_1 (x - a_1) & (x \in I_1) \\ \quad\vdots & \\ \beta_i (x - a_i) & (x \in I_i) \\ \quad\vdots & \\ \beta_{n-1} (x - a_{n-1}) & (x \in I_{n-1}) \\ \gamma (x - a_n) & (x \in I_n) \end{cases}$$



で定める。ただし，

$$|\alpha| < \frac{1}{|I_0|}, \qquad |\beta_i| = \frac{1}{|I_i|} \quad (i = 1,2,\cdots,n-1), \qquad |\gamma| < \frac{1}{|I_n|}$$

であり，$|I_i| = a_{i+1} - a_i$ である。

命題 T が

$$T(I_0) = I_k \cup I_{k+1} \cup \cdots \cup I_n, \qquad T(I_n) = I_0 \cup I_1 \cup \cdots \cup I_{j-1}$$

を満たすマルコフ変換であるとき以下が成り立つ。

(i) $j < k$ のとき

$$h(y) = \frac{|T(I_0)|(|T(I_n)| + |I_n|)}{|T(I_0)||T(I_n)| - |I_0||I_n|} \chi_{I_0 \cup I_1 \cup \cdots \cup I_{j-1}}(y) + \chi_{I_j \cup I_{j+1} \cup \cdots \cup I_{k-1}}(y) + \frac{|T(I_n)|(|T(I_0)| + |I_0|)}{|T(I_0)||T(I_n)| - |I_0||I_n|} \chi_{I_k \cup I_{k+1} \cup \cdots \cup I_n}(y)$$

は，T の不変測度の密度を与える（χ は特性関数）。

(ii) $j > k$ のとき

$$h(y) = \frac{|T(I_0)|}{|T(I_0)| + |I_0|} \chi_{I_0 \cup I_1 \cup \cdots \cup I_{k-1}}(y) + \chi_{I_k \cup I_{k+1} \cup \cdots \cup I_{j-1}}(y) + \frac{|T(I_n)|}{|T(I_n)| + |I_n|} \chi_{I_j \cup I_{j+1} \cup \cdots \cup I_n}(y)$$

は，T の不変測度の密度を与える。

(iii) $j=k$ のとき

$$h(y)=\frac{|T(I_n)|+|I_n|}{|T(I_n)|}\chi_{I_0\cup I_1\cup\cdots\cup I_{j-1}}(y)+\frac{|T(I_0)|+|I_0|}{|T(I_0)|}\chi_{I_j\cup I_{j+1}\cup\cdots\cup I_n}(y)$$

は，T の不変測度の密度を与える。

１３．ＳＳＲ計算の整数化（ＳＳＩ計算）と [1,2) 上のβ変換

１３.１ ＳＳＲ計算の整数化

非再帰的に擬似乱数を生成する実数シフト法（ＳＳＲ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。本節では，実数シフト乱数生成アルゴリズムを，整数の乗算とシフトによるアルゴリズム（ＳＳＩアルゴリズム，simplified shift integer algorithm，単純整数シフト）で実現し，実用に耐えうる乱数生成法 SSI32 として構成する。

実数シフト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。最近は６４ビットの整数演算が可能な計算機が殆どであるので，６４ビット×６４ビットの整数乗算（→実行結果は通常６４ビット）を行ったとき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（桁落ち量）に相当するように調整することで，実行速度をおとさずに整数演算化することが可能である。６４ビット×６４ビットの整数乗算を行ったとき，（６４ビットを超える）オーバーフローの桁数は一定ではないが（後で示されるように，我々の整数化アルゴリズム SSI32 ではオーバーフローの桁数の違いが１ビットである），SSRex アルゴリズムの導入箇所において言及したように，仮数部のシフトサイズは必ずしも固定したものでなくても良いので，演算の高速化を図るため，SSR アルゴリズムの整数演算化では，６４ビット×６４ビットの整数乗算後のシフト量の調整を行わない方法を採用する。

以下の計算を，64 ビット整数 $t,\ x$ の**単純整数シフト計算**（the Simplified Shift-Integer computation, SSI 計算）といい，$\Psi_x(t) = t \bullet x$ で表すことにする。

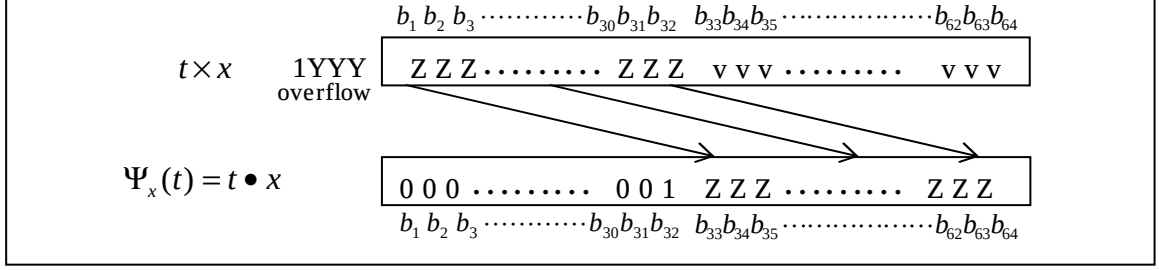
単純整数シフト計算(SSI 計算) $\Psi_x(t)$

t, x を６４ビット整数とする

i) $t \times x$ 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，

iii) 空いた $b_1 b_2 b_3 \cdots\cdots b_{30} b_{31} b_{32}$ を 000……001 とする。



SSI 計算の i)におけるオーバーフロー 1YYY の YYY の部分が $\Psi_x(t)$ では棄てられるので，この部分が，SSR 計算の仮数部の左シフトにより棄てられる部分に相当する。このように，SSI 計算では桁落ちに相当する左シフトをわざわざ実行する必要がないので，SSR 計算より計算時間が短くなる。実際の SSI 計算では，YYY が，３または４ビットになるように t, x が調整されている。

SSI 計算のプログラム例

```
typedef struct
   {unsigned int L;  unsigned int H;} uiHL;    //Intel's MPU only
typedef union
   {unsigned long long int I;  uiHL HL;} uiLLHL;
unsigned long long int SSIComputation(unsigned long long int T,
                                      unsigned long long int X)
   {
   uiLLHL  TX;
   TX.I = T*X;    TX.HL.L = TX.HL.H;    TX.HL.H =1;
   return(TX.I);
   }
```

単純整数シフト(SSI)乱数生成法は，乱数値の生成関数として，$\Psi_x^{22}(w_0)\times x$ を使うものである。以下の SSI 乱数生成法では，乱数周期の長期化と特性の改善のために，既述の K 改良を最初から適用しておく。

K 改良 SSI 乱数生成法(SSI32K 乱数生成法)

(1) p，q，$r(<p)$，$s(<q)$ を素数とし，

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

とおく。

(2) ６４ビット整数 x，y に対し，x_k，y_k を次のように定める：

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

(註) XOR は，ビット毎の排他的論理和(exclusive or) を表す。

(3) $\Psi_{x_k}{}^{22}(w_0)\times x_k - \Psi_{y_k}{}^{22}(v_0)\times y_k$ の上位１６ビットを棄て，続く３２ビットを k 番目の**乱数値**とする。

$$\Psi_{x_k}{}^{22}(w_0)\times x_k - \Psi_{y_k}{}^{22}(v_0)\times y_k \quad \Rightarrow \quad \underbrace{b_1 \cdots\cdots\ b_{17}\, b_{18}\, b_{19} \cdots\cdots b_{46} b_{47} b_{48}}_{\text{random number}} \cdots\cdots b_{64}$$

具体的な数値を当てはめて，もう少し詳しく SSI32K 乱数生成法を説明する。まず

p = 0x7ffffffe1	r = 0x39f750241
q = 0x7fffffcf	s = 0x32f50fee9
x = 0x88237449a	y = 0xbdda73ad3
w_0 = 0x18237449a	v_0 = 0x1dda73ad3

である（註：0x は１６進表記であることを示す)。このとき $\Psi^i_{x_k}(w_0)$ ， x_k は

$$0x100000000 \le \Psi^i_{x_k}(w_0) \le 0x1fffffffff \ , \quad 0x800000000 \le x_k \le 0xfffffffff$$

であるので（ $\Psi^i_{y_k}(v_0)$, y_k についても同様)，整数乗算 $\Psi^i_{x_k}(w_0)\times x_k$ を行うと，

$$0x80000000000000000 \le \Psi^i_{x_k}(w_0)\times x_k < 0x1fffffffffffffffff$$

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，オーバーフローして棄てられる 1YYY のビット数は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 $b_1\,b_2\,b_3\cdots\cdots\cdots b_{30}b_{31}b_{32}$ が $b_{33}b_{34}b_{35}\cdots\cdots\cdots\cdots\cdots b_{62}b_{63}b_{64}$ として保存され，元の $b_1\,b_2\,b_3\cdots\cdots\cdots b_{30}b_{31}b_{32}$ の所には 000・・・001 がセットされるので，ＳＳＲ計算の仮数部の左シフトで棄てられるビットに対応するＳＳＩ計算のビットは，YYY の３～４ビットになる。このプロセスが $\Psi_{x_k}{}^{22}(w_0)\times x_k$ により，２３回繰り返されることになるので，平均 3.5 ×23 =80.5 ビット相当のシフトがＳＳＩ乱数生成法でなされていると考えられる。

SSI32K 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 $\Psi_{x_k}{}^{22}(w_0)\times x_k - \Psi_{y_k}{}^{22}(v_0)\times y_k$ の周期は，

$$pq = 34359738337\times 34359738319 \approx 1.18\times 10^{21}$$

となる。

１３.２ ＳＳＩ計算と [1,2) 上のβ変換 M_β

上述のＳＳＩ計算はアルゴリズム先行で数式的背景が分かりにくいが，以下のように [1,2) 上のβ変換 $M_\beta(t) = \beta t - \lfloor \beta t \rfloor + 1$，$\beta > 1$，$t \in [1,2)$，としてとらえることができる。

まず

$$0x100000000 \le \Psi^i_{x_k}(w_0) \le 0x1fffffffff \ , \quad 0x800000000 \le x_k \le 0xfffffffff$$

であるので，$\Psi^{i}_{x_k}(w_0)$ を $1.t_1t_2\cdots t_{32}$（33 ビット） と同一視し，x_k を $1.x_1x_2\cdots x_{35}$（36 ビット）と同一視する。すると $\Psi^{i}_{x_k}(w_0)\times x_k$ は，$\breve{b}_0\breve{b}_1.\breve{b}_2\breve{b}_3\cdots\breve{b}_{68}$ （69 ビット）と同一視される。（$\breve{b}_0\breve{b}_1$ は，乗算結果が [1,2) 内にあるときは $\breve{b}_0\breve{b}_1=01$ に，[2,4) 内にあるときは $\breve{b}_0=1$ となる。）このときＳＳＩ計算のオーバーフロー 1YYY に相当する部分は，$\breve{b}_0\breve{b}_1\breve{b}_2\breve{b}_3$ であり，1ZZZ...ZZZ に相当する部分は $1\breve{b}_4\breve{b}_5\breve{b}_6\cdots\breve{b}_{68}$ となる。この $1\breve{b}_4\breve{b}_5\breve{b}_6\cdots\breve{b}_{68}$ は，最終的に３３ビットに丸められて $1\breve{b}_4\breve{b}_5\breve{b}_6\cdots\breve{b}_{35}$ になり，ＳＳＩ計算を終えるが，最後の３３ビット $1\breve{b}_4\breve{b}_5\breve{b}_6\cdots\breve{b}_{35}$ は，$1.\breve{b}_4\breve{b}_5\breve{b}_6\cdots\breve{b}_{35}$ と同一視され，次の $1.t_1t_2\cdots t_{32}$ となる。$1.x_1x_2\cdots x_{35}$，$1.t_1t_2\cdots t_{32}$ から，ＳＳＩ計算により $1.\breve{b}_4\breve{b}_5\breve{b}_6\cdots\breve{b}_{35}$ を得るには以下の図のような操作で行うことが出来るが，

$$\begin{matrix}1.x_1x_2\cdots x_{35}\\ 1.t_1t_2\cdots t_{32}\end{matrix}\xrightarrow{\text{mul}}\breve{b}_0\breve{b}_1.\breve{b}_2\breve{b}_3\breve{b}_4\breve{b}_5\cdots\cdots\breve{b}_{68}\xrightarrow{\times 2^2}\breve{b}_0\breve{b}_1\breve{b}_2\breve{b}_3.\breve{b}_4\breve{b}_5\cdots\cdots\breve{b}_{68}$$

$$\xrightarrow{\bmod[1,2)}1.\breve{b}_4\breve{b}_5\cdots\breve{b}_{68}\xrightarrow{\text{cut}}1.\breve{b}_4\breve{b}_5\cdots\breve{b}_{35}$$

最後の丸め(=cut)を除いた部分までの計算は，$\beta=2^2\times 1.x_1x_2\cdots x_{33}$ とおくと

$$\begin{aligned}M_\beta(1.t_1t_2\cdots t_{32})&=\beta\times 1.t_1t_2\cdots t_{32}\ \bmod[1,2)\\&=\beta\times 1.t_1t_2\cdots t_{32}-\lfloor\beta\times 1.t_1t_2\cdots t_{32}\rfloor+1\end{aligned}$$

と数式化される。この関数 M_β は，よく知られているβ変換（[Re]）

$$T_\beta(t)=\beta t-\lfloor\beta t\rfloor\ \equiv\ \beta t\ \bmod 1$$

の概念を，[1,2) 上の変換として発展させたものとしてとらえることができる。以降，我々はこの変換 M_β を「[1,2) 上のβ変換」あるいは「変形β変換」(modified beta-transformation) と呼ぶことにする。ここにおいて，我々は，ＳＳＩ計算を数式的に扱う手段 M_β を手に入れたので，次章では M_β を中心に据えて議論を進めていくことになる。

１４．ＳＳＩ３２K乱数に対する統計的検定

節の最後に，SSI32K に対する検定結果について以下にまとめておく。他の乱数との比較でも分るように， SSI32K 乱数の特性について特に問題点はない。

【 14.1 ＳＳI32K乱数に対するNISTの検定 】

NISTの検定

National Insitute of Standards and Technology（ http://csrc.nist.gov/rng/ ）**が提供する乱数検定プログラム。**

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１５種類１８８テストで構成されている（() 内はテスト数（1は省略））。

[検定のための乱数データ]	**1ギガ ビットからなるファイル**
[検定の実施]	**すべてのテストを適用**
[検定時のパラメータ]	**block-frequency テストの block length = 20000 を除いて、既定値のまま**

【14.2 NISTの検定の実行 】

● **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-VALUE）

◆ $p \geq 0.01$ 以上であるものの割合（PROPORTION）

が、finalAnalysisReport **として出力される。**

[finalAnalysisReport **の例**]

```
------------------------------------------------------------------------------
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
...............
```

NIST **の検定を１６回繰り返し、**finalAnalysisReport **の各値を平均する。**

⇒ **平均された** P-VALUE 、PROPORTION **の最大値･最小値を求める。**

【14.3 NISTの検定の結果】

[16回の検定結果の平均値] の最大･最小

SSI32K

```
[(p値 χ² 検定値)  平均max] = 0.661006  at  nonperiodic-templates [i=38]
[(p値 χ² 検定値)  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]
```

(i = the # of test)

他乱数との比較

		MT	FSR	SSR	SSIK	物理乱数
[χ^2 test of P-VALUE]	AvMax	0.721409	0.685129	0.666747	0.661006	0.691940
[χ^2 test of P-VALUE]	AvMin	0.304160	0.292520	0.338575	0.320636	0.310535
[PROPORTION(pass seq)]	AvMax	0.992250	0.991750	0.992112	0.991750	0.992013
[PROPORTION(pass seq)]	AvMin	0.987500	0.986788	0.987188	0.986687	0.986894

[MT: Mersenne Twister , FSR : Feedback Shift Register]

［註］FSR 乱数は，M 系列と言われる

$$Y_n \;=\; Y_{n-32} \;\text{XOR}\; Y_{n-521}$$

により得られる 32 ビット整数である。

MT 乱数は [Ma] に述べられている著名な乱数生成法である。

物理乱数は「エルイーテック」の「真性乱数発生器」による。

１４．４　TestU01 による検定

TestU01 [LE] は，乱数検定プログラムの現時点での集大成という感じの乱数検定ソフトウェアである。複数の検定の組み合わせのレベルにより SmallCrush, Crash, BigCrush の組(suite)が提供されている。ここでは検定数が最も多い BigCrush を使用する。BigCrush は 160 の統計的検定（テスト）からなり，各テストは倍精度の [0,1] 間の一様乱数あるいは 32 ビットの整数乱数を要求する。32 ビットの整数乱数が要求されたときは，SSI32K 乱数をそのまま渡す。倍精度の浮動小数点乱数が要求されたときは，倍精度変数の仮数部の MSB を含む最初の 32 ビットに SSI32K 乱数をセットし，残りの 20 ビットには，今使った SSI32K 乱数を計算するときに使用した $\Psi_{x_k}{}^{22}(w_0)\times x_k$ の最初の 16 ビットを棄てたあとの 20 ビットをセットする。さらに指数部が …×2^0 となるようにセットすれば [1,2) 内の一様乱数が得られるので，これから 1 を引いて [0,1) 一様乱数とする。

BigCrush の各テストは，そのテスト結果をＮＩＳＴの検定と同様に p 値として報告し，さらに全テスト終了後 Summary として p 値が [0.001,0.999] に収まらなかったテストの名称を一覧表示して BigCrush を終了する。

TestU01 による SSI32K 乱数の検定結果は以下の通りである。

```
========= Summary results of BigCrush =========

 Version:         TestU01 1.2
 Generator:       My SSI32 implementation
 Number of statistics:  160
 Total CPU time:   71:24:46.76

 All tests were passed
```

第 2 章

[1,2)上のβ変換を利用した乱数の生成とその応用

第 2 章 [1,2)上のβ変換を利用した乱数の生成とその応用

１．はじめに

前章１３．２節で，ＳＳＩ計算は [1, 2) 上のβ変換により数式表現されることについて述べたが，本章では，記述が何となくもやもやしたＳＳＩアルゴリズムに触れることなく，[1, 2) 上のβ変換 M_β を用いてＳＳＩ乱数を数式的に作る方法を改めて述べる。次いで，その数式的記述に基づいて情報セキュリティで重要な役割を果たしているハッシュ関数を構成し，その安全性を議論することにする。記述に当たっては，理解を容易にするため前章までの議論を“ちゃら”にして，改めて自己完結的な内容で記述することにした。また，議論を単純にするため，ＳＳＩ計算で用いた乗算時に発生するオーバーフローを利用する方法をやめて，32 ビット×32 ビット＝64 ビットの乗算の範囲で収まるアルゴリズムを採用した。（そのためここで作成する 32 ビット乱数を MB32rand と呼んで SSI32rand と区別した。）それでは，[1,2) 上のβ変換の定義から始めることにしよう。

２．[1,2) 上のβ変換を利用した 32 ビット小型乱数生成器の構成

$\beta > 1$ に対し，[1,2) 上のβ変換 $M_\beta : [1,2) \to [1,2)$ を

$$M_\beta(t) = \beta t \mod [1,2) = \beta t - \lfloor \beta t \rfloor + 1 \qquad (2.1)$$

で定める。ここで $\lfloor \beta t \rfloor$ は βt を超えない最大の整数を表す（ $\lfloor\ \rfloor$ はガウス記号に相当する）。次のグラフは $M_{2^3\times 1.2781\cdots}(t),\ t \in [1,2)$，および $y_n = M^{16}_{2^3 x_n}(1),\ x_n = 1 + \dfrac{n}{20000},\ n = 0,1,\cdots,19999,$ を倍精度計算して表示したものである。

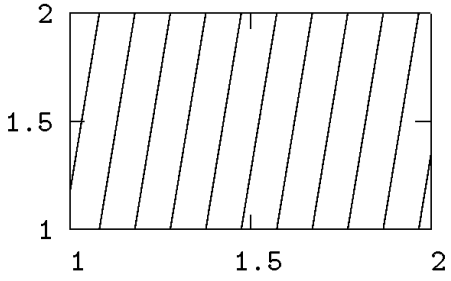


図１．$M_{2^3\times 1.2781}(t)$ のグラフ

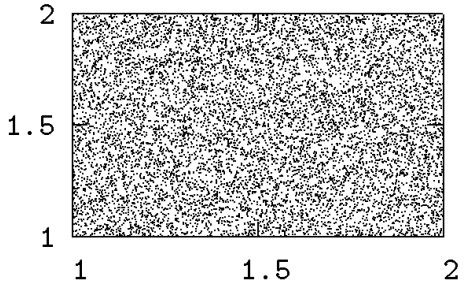


図２．$y_n = M^{16}_{2^3 x_n}(1)$ のグラフ

上図右側のグラフは，$\{y_n\}$ の最初の何桁かを棄てれば，$\{y_n\}$ から良い乱数系列が得られることを暗

示している。実際，次の２．１節のように [1,2) 上のβ変換 $M_{2^3x_n}$ を利用して 32 ビットの小型乱数器　MB32rand を構成することができる。

２．１　MB32rand のアルゴリズム

$1.e$ で数 1.27181... を表すことにする。ここで，ネイピアの数 $e=2.7181\cdots$ は，超越数（有理数を係数とする代数方程式の解とならないような数）であることに注意しておく。$1.e$ の２進表現を $1.\hat{r}_1\hat{r}_2\hat{r}_3\cdots\hat{r}_{31}\cdots$ とするとき，３１ビットの非負整数 $n=\nu_1\nu_2\cdots\nu_{31}$ に対し，

$$x_n=1.(\hat{r}_1\oplus\nu_1)(\hat{r}_2\oplus\nu_2)\cdots(\hat{r}_{31}\oplus\nu_{31})\hat{r}_{32}\hat{r}_{33}\cdots$$

と定める。ここで $\oplus$ は，排他的論理和 XOR を表す。この x_n に対し，

$$z_n\equiv M^{16}_{2^3x_n}(x_n)=1.\breve{b}_5\breve{b}_6\cdots\breve{b}_{15}\breve{b}_{16}\cdots\breve{b}_{47}\cdots\quad（２進）$$

を計算し，z_n の最初の１２ビットを棄て，続く３２ビット $\breve{b}_{16}\cdots\breve{b}_{47}$ を n 番目の乱数 ζ_n として取り出す。すなわち ζ_n は，

$$\zeta_n=\lfloor 2^{32}(2^{11}z_n-\lfloor 2^{11}z_n\rfloor)\rfloor \tag{2.2}$$

で計算される数値である。この $\{\zeta_n\}_{n=0,1,\cdots}$ が乱数列となることの統計的検定は後ほど行われるが，この３２ビット乱数生成器を **MB32rand**（the modified beta 32-bit random number generator）と呼ぶことにする。

２．２　MB32rand のアルゴリズムの計算機への実装

上述のアルゴリズムを計算機に実装するにあたっては，各数が使えるビット数を決めなければならない。我々は以下の３２ビット実数系で MB32rand のアルゴリズムをコンピュータ上に実装する。

- [1,2) 内のすべての実数は，（乗算結果を除いて）３２ビット $1.b_1b_2b_3\cdots b_{31}$ で表される（３２ビットを超す部分は切り捨てる），
- ２数 $x=1.x_1x_2\cdots x_{31}$ と $t=1.t_1t_2\cdots t_{31}$ の乗算結果は６４ビット $\breve{b}_0\breve{b}_1.\breve{b}_2\breve{b}_3\cdots\breve{b}_{63}$ で表され（$\breve{b}_0\breve{b}_1$ は，乗算結果が [1,2) 内にあるときは $\breve{b}_0\breve{b}_1=01$ に，[2,4) 内にあるときは $\breve{b}_0=1$ となる），その後，左シフト等を経て３２ビットに切りつめられる。

上述の３２ビット実数系の数 $1.b_1b_2b_3\cdots b_{31}$ は，３２ビットの整数 $1b_1b_2b_3\cdots b_{31}$ と同一視すると何かと具合が良いので，そうすることが多い。もちろん，乗算結果の $\breve{b}_0\breve{b}_1.\breve{b}_2\breve{b}_3\cdots\breve{b}_{63}$ は６４ビット整数 $\breve{b}_0\breve{b}_1\breve{b}_2\breve{b}_3\cdots\breve{b}_{63}$ と同一視する。この同一視を行うと，例えば $1.e$ は３２ビット整数で 0xa2cb4411 と表される。ここで 0x は１６進表現であることを示す。関数 $M_{2^3x}(t)$ は，

$$
\begin{aligned}
M_{2^3x}(t) &= (2^3x)t \mod [1,2) = (2^3)xt \mod [1,2) \\
&= 2^3 \times \breve{b}_0\breve{b}_1.\breve{b}_2\breve{b}_3\breve{b}_4\breve{b}_5\breve{b}_6\cdots \mod [1,2) \\
&= \breve{b}_0\breve{b}_1\breve{b}_2\breve{b}_3\breve{b}_4.\breve{b}_5\breve{b}_6\cdots \mod [1,2) \\
&= 1.\breve{b}_5\breve{b}_6\cdots
\end{aligned}
$$

と計算されるので，３２ビット整数系での $M_{2^3x}(t)$ は６４ビット整数 $\breve{b}_0\breve{b}_1\breve{b}_2\breve{b}_3\cdots\breve{b}_{63}$ から抽出される３２ビット整数 $1\breve{b}_5\breve{b}_6\cdots\breve{b}_{35}$ と同一視される。この時，６４ビット整数 $\breve{b}_0\breve{b}_1\breve{b}_2\breve{b}_3\cdots\breve{b}_{63}$ から３２ビット整数 $1\breve{b}_5\breve{b}_6\cdots\breve{b}_{35}$ の抽出は，

$$
\breve{b}_0\cdots\breve{b}_4\breve{b}_5\cdots\breve{b}_{35}\cdots\breve{b}_{63} \xrightarrow{\text{(i) shift 4}} \breve{b}_4\breve{b}_5\cdots\breve{b}_{35}\cdots\breve{b}_{63} \xrightarrow{\text{(ii) OR}} 1\breve{b}_5\cdots\breve{b}_{35}\cdots\breve{b}_{63} \xrightarrow{\text{(iii) cut}} 1\breve{b}_5\cdots\breve{b}_{35}
$$

のように，３つの整数演算

（ｉ）４ビットの左シフト（$\breve{b}_0\breve{b}_1\breve{b}_2\breve{b}_3$ は棄てられる），
（ii）$\breve{b}_4$ との論理和 OR，
（iii）$\breve{b}_{36}\cdots\breve{b}_{63}$ の廃棄（カットオフ）

で行えることに注意する。n 番目の乱数 ζ_n は，２．１節の x_n に対して計算される $M^{16}_{2^3x_n} = 1.\breve{b}_5\cdots\breve{b}_{16}\cdots\breve{b}_{47}\cdots\breve{b}_{63}$ から得られる３２ビット整数 $\breve{b}_{16}\cdots\breve{b}_{47}$ であり，$\zeta_0 = 0\text{x}6\text{f}890520$, $\zeta_1 = 0\text{xb}16\text{d}7669$ などと計算される。

２．３　MB32rand の乱数性の統計的検定

この節では $\zeta_0, \zeta_1, \zeta_2, \zeta_3, \cdots$が乱数列になっていることを，ＮＩＳＴ [Www1] が提供する統計的検定 sts-2.1.1 を用いて調べる。この検定に使ったデータは，ζ_n，$n = 0,1,2,\cdots$，から作られる各ファイル $1024^2 \times 1000$ ビットからなる１０個のファイルである。各ファイルは，sts-2.1.1 に 1024^2 ビットのビット列を１０００本供給する。sts-2.1.1 を構成する各テスト（１８８ある）は，この供給された１０００本の各ビット列に対して検定を行い，p-値を計算し報告する。また同時に，危険率 0.01 でテストをパスした割合も報告する。（sts-2.1.1 のパラメータは， block frequency test の block length=20000 を除いて sts-2.1.1 の既定値を使用した。）p-値は [0,1] 内に一様に分布する確率変数とされ，０または１に非常に近い値が棄却域になる（両側検定）。また，危険率 0.01 で各テストをパスする割合は，1.00－0.01＝0.99 に近い値が正常である。われわれは，sts-2.1.1 を構成するテストを１０個の各ファイルについて行い（＝１０回繰り返し），１８８個の各テスト毎に得られた１０個の p-値，および，「危険率 0.01 でテストをパスした割合」について，平均値を計算した。最後に，結果の記述を簡単にするため，テスト毎に得られた１８８個の平均値について，その最大値および最小値を求めた。結果は以下の通りである。（比較のため，よく知られている擬似乱数生成法である

Mersenne Twister ar [Ma] についての結果も記した。）

```
[MB32rand]
  P-VALUE AvMax = 0.737001 at NonOverlappingTemplate [i=83]
  P-VALUE AvMin = 0.285480 at RandomExcursionsVariant [i=181]
  PROPORTION AvMax = 0.992503 at RandomExcursionsVariant [i=170]
  PROPORTION AvMin = 0.986800 at FFT [i=7]

[Mersenne Twister ar]
  P-VALUE AvMax = 0.713578 at NonOverlappingTemplate [i=105]
  P-VALUE AvMin = 0.188745 at NonOverlappingTemplate [i=126]
  PROPORTION AvMax = 0.992879 at RandomExcursionsVariant [i=168]
  PROPORTION AvMin = 0.986809 at RandomExcursions [i=159]
```

ここで，[i=**] の ** は，sts-2.1.1 を構成するテストの通し番号である。これらの結果から $\{\zeta_n\}$ は乱数列であるということができよう。

３．[1,2）上のβ変換を利用した 32 ビット小型ハッシュ関数の構成

ハッシュ関数は, その値が任意の入力バイト列 $B = B_1 B_2 \cdots B_N$ に従って決まる擬似乱数生成器と考えることができる。上述の MB32rand では，M_β が繰り返し使われるが，その繰り返しの合間にバイト情報 B_k を埋め込むことは容易であるので，MB32rand のアルゴリズムを利用してハッシュ関数を構成することができる。ここでは，３２ビットのハッシュ値をもつ小型ハッシュ関数 MB32hash を構成する。この MB32hash は，次節で扱う n=160, 192, 256, ..., 2048, 4096 ビットのハッシュ値をもつハッシュ関数 MB$n$hash の基礎となるものである。

３．１　MB32hash のアルゴリズム

ハッシュ関数への入力バイト列を $B = B_1 B_2 \cdots B_N$ とする。MB32hash は,

圧縮(compression),　　攪乱(scrambling)

と呼ばれる２つの過程から構成されている。

３．１．１　圧縮過程のアルゴリズム（MB32hash）

$w_0 = 1.e$ とし，w_{k-1} から w_k を

$$w_k = M_{2^3 1.e}(w_{k-1} \oplus B_k), \quad k = 1,2,\cdots,N \tag{3.1}$$

により計算する。ここで，バイト $B_k = c_1 c_2 \cdots c_8$ に対して，

$$w_{k-1} \oplus B_k = 1.b_1 \cdots b_7 (b_8 \oplus c_1) \cdots (b_{15} \oplus c_8) b_{16} b_{17} \cdots b_{31} \cdots \qquad (3.2)$$

である。これを繰り返して B から最終的に w_N を得るプロセスを，我々は**圧縮**と呼ぶことにする。w_N は B の全ての情報を使って作られたと考えられる。この過程の最後に，B の長さ N に関する情報を w_N に埋め込むために，$w_N = 1.b_1 b_2 b_3 \cdots$，$N = \nu_1 \nu_2 \cdots \nu_{31}$（$N < 2^{31}$ を仮定する）として，

$$y \equiv w_N \oplus N = 1.(b_1 \oplus \nu_1)(b_2 \oplus \nu_2) \cdots (b_{31} \oplus \nu_{31}) b_{32} b_{33} \cdots$$

を作っておく。

３．１．２　攪乱過程のアルゴリズム（MB32hash）

入力 B に対応する３２ビットの乱数を得るには，即ち，入力 B に対するハッシュ値 ς を得るには，MB32rand のアルゴリズムを使う（２．１節参照）：すなわち，上述の y に対して

$z = M^{16}_{2^3 y}(y)$　を計算し，　$\zeta = \lfloor 2^{32}(2^{11} z - \lfloor 2^{11} z \rfloor) \rfloor$　とする（(2.2)参照）。　(3.3)

我々は，この y から z を得るプロセスを，y の**攪乱**と言うことにする。また，入力バイト列 B に対する３２ビットハッシュ値 ς を得る圧縮と攪乱の両プロセスを併せて **MB32hash** と呼ぶことにする。

３．２　MB32hash のアルゴリズムの計算機への実装(implementation)

３．１節で記述された MB32hash アルゴリズムの計算機への実装は，２．２節（MB32rand の実装）で導入された３２ビット実数系を用いて行う。

３．３　MB32hash が生成するハッシュ値の乱数性

ここで，MB32hash が生成するハッシュ値の乱数性を調べておく。乱数性は２．３節と同様 NIST の検定を用いる。多量のハッシュ値（乱数）$\varsigma_k, k = 0,1,2,\cdots$を発生させるため，入力バイト列 B として，８バイトからなる文字列 B_k を使う。ここで B_k の値は，B_k を８バイトの整数としてみたとき値が k となるものである（たとえば，１バイトを１６進表記の２文字で表すとき，

B_0＝(00) (00) (00) (00) (00) (00) (00) (00)，　B_1＝(00) (00) (00) (00) (00) (00) (00) (01)

などとなる）。検定結果は以下の通りであり，特に問題となる点は見られない：

```
[MB32hash]
  P-VALUE AvMax = 0.713263 at NonOverlappingTemplate [i = 20]
  P-VALUE AvMin = 0.223232 at NonOverlappingTemplate [i = 116]
  PROPORTION AvMax = 0.992982 at RandomExcursionsVariant [i = 168]
  PROPORTION AvMin = 0.986110 at RandomExcursions [i = 159]
```

３．４　MB32hash の乱数性の向上

TestU01 [LE] は P. L'Ecuyer らによって作られた，乱数性を検定する統計的検定の一連のセットである。TestU01 は，SmallCrush，Crush，BigCrush と呼ばれる３つの統計的検定の組からなり，Crush，BigCrush は多量の乱数を使って乱数性をテストする。そのため，検定は NIST の検定よりも詳細であり，上述のハッシュ値の列 $\{\varsigma_k\}_k$ に，Crush，BigCrush を適用すると，それらを構成する幾つかの検定にパスしないという現象が生じる。しかしながら，MB32hash のアルゴリズムは非常に柔軟であるので, この問題は容易に解決される。１つの簡単な方法は, 圧縮過程で写像 $M_{2^{31}.e}$ の代わりに写像 $M_{2^{23}1.e}$を使い，攪乱過程で写像 $M^{16}_{2^2 y}$ の代わりに写像 $z = M^{16}_{2^{23} y}$ を使う方法である。ハッシュ値としては $\zeta = \lfloor 2^{32}(2^3 z - \lfloor 2^3 z \rfloor) \rfloor$ で与えられる３２ビットを採用する。またこのアルゴリズムのコンピュータへの実装にあたっては，MSB（most significant bit，最上位ビット）および LSB（least significant bit，最下位ビット）が常に１に保たれる４４ビットの実数体系 $1.b_1 b_2 \cdots b_{42} 1$ を用いる。このようにして生成されたハッシュ値 $\{\varsigma_k\}_k$ は，Crush，BigCrush のすべての検定をパスする。（LSB を１にすることは，$\{\varsigma_k\}_k$ の乱数性を増す効果がある。（４．２節，６．１．２節参照））

４.ハッシュ関数 MB*n*hash, *n*=192, 256, ..., 2048, 4096, の構成

MB32hash のアルゴリズムは単純で柔軟性に富んでいるので，関数 M_β の β をいろいろ変えることにより，様々なハッシュ値長をもつハッシュ関数 MB*n*hash を構成することが出来る。ここでは，ハッシュ値長 n = 192, 256, 384, 512, 1024, 2048, 4096 ビットの場合を扱うが，ハッシュ関数の構成の仕方は各 n について同じであるので，以下 n=1024 と仮定して議論を進める。

４．１　MB1024hash のアルゴリズム

４．１．１　圧縮過程のアルゴリズム（MB1024hash）

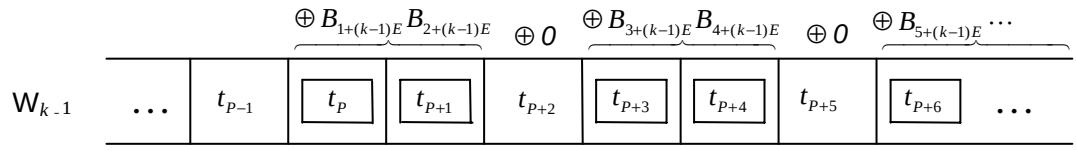
ハッシュ関数に対する入力バイト列を $B = B_1 B_2 \cdots B_N$ とする。 $W_0 = 1.e$ とし，$W_{k-1} \in [1,2)$ から $W_k \in [1,2)$ を次のように計算する：

$$W_k = M_{2^{P-1}1.e}(W_{k-1} \oplus \underline{B_{1+(k-1)*E} B_{2+(k-1)*E} \cdots B_{E+(k-1)*E}}) \quad (P = 128, E = 32). \qquad (4.1)$$

ここで，$\oplus$(=XOR) は，$W_{k-1} = 1.t_1 t_2 t_3 \cdots t_p \cdots$ のビット t_p から取り始める。また

$$\underline{B_{1+(k-1)*E} B_{2+(k-1)*E} \cdots B_{E+(k-1)*E}} = B_{1+(k-1)*E} B_{2+(k-1)*E} 0 B_{3+(k-1)*E} B_{4+(k-1)*E} 0 \cdots 0 B_{E-1+(k-1)*E} B_{E+(k-1)*E})$$

であり，0 は値０(=0x00) を持つ１バイト文字で，バイト列 $B_{1+(k-1)*E} B_{2+(k-1)*E} \cdots B_{E+(k-1)*E}$ に対して２バイトおきに挿入される（最後の 0 はつけない）。



B の中にヌルバイト(null byte) 0 を埋め込むのは，W_{k-1} の中に，B が触れることが出来ない場所を確保するためである（→ セキュリティの向上；６．１．１節参照)。最後の W_k ，すなわち $W_{\lfloor (N-1)/E \rfloor + 1} = 1.t_1 t_2 t_3 \cdots$ を計算した後，B の長さ $N = \nu_1 \nu_2 \cdots \nu_{31} \nu_{32}$（２進）を $W_{\lfloor (N-1)/E \rfloor + 1}$ に

$$\tilde{W} = 1.\, t_1 \cdots t_{P-1} (t_P \oplus \nu_1) \cdots (t_{P+31} \oplus \nu_{32}) t_{P+32} \cdots \qquad (4.2)$$

のように埋め込んで $\tilde{W}$ を得ておく。(註: N は特に３２ビットである必要はないことに注意する。)

４．１．２　攪乱過程のアルゴリズム（MB1024hash）

$S = 96,\ Q = 224$ とする。(S は $16S$ が n (=1024) の１.５倍程度になるように定められている。また $16S$ の 16 は $(M_\beta)^{16}$ の 16 に由来している。）攪乱過程ではまず最初に $Z = (M_{2^{S-1}\tilde{W}})^{16}(\tilde{W})$ を計算する。続いて Z の最初の $Q/2 - S$ ビットを棄て，続く $n = 1024$ ビットをハッシュ値 ζ として採用する：

$$\zeta = \lfloor 2^n (2^{(Q/2)-S-1} Z - \lfloor 2^{(Q/2)-S-1} Z \rfloor) \rfloor .$$

４．２　MB1024hash のアルゴリズムの計算機への実装

４．２．１　圧縮過程アルゴリズムの計算機への実装（MB1024hash）

MB1024hash の圧縮過程のアルゴリズムの実装にあたっては，(4.1) 中の定数 $2^{P-1}1.e$ および変数 W_{k-1} に与えるビット数は，それぞれ $M = 160,\ L = 1024\ (= n)$ ビットとする。M を n より小

さい値に取った理由は，(4.1) 中の乗算にかかる計算時間を短くしたいためである。また，実装にあたっては，$1.e$ および W_{k-1} の MSB, LSB は常に１にしておく（３．４節，６．１．２節を参照)。このとき，$1.e = 1.x_1x_2\cdots x_{M-2}1,\ t = 1.t_1t_2\cdots t_{n-2}1$ とすれば $M_{2^{P-1}1.e}(t)$ の計算は次のように行われる：

$$\begin{matrix} 1.e = 1.x_1x_2\cdots x_{M-2}1 \\ t = 1.t_1t_2\cdots t_{n-2}1 \end{matrix} \xrightarrow{\text{mul}} \breve{b}_0\breve{b}_1.\breve{b}_2\cdots\breve{b}_{P-1}\breve{b}_P\breve{b}_{P+1}\cdots\cdots\breve{b}_{n+M-2}1 \xrightarrow{\times 2^{P-1}}$$

$$\breve{b}_0\breve{b}_1\breve{b}_2\cdots\breve{b}_{P-1}\breve{b}_P.\breve{b}_{P+1}\cdots\cdots\breve{b}_{n+M-2}1 \xrightarrow{\text{mod}[1,2)} 1.\breve{b}_{P+1}\cdots\breve{b}_{P+n-1}\cdots\breve{b}_{n+M-2}1$$

$$\xrightarrow{\text{cut}} 1.\breve{b}_{P+1}\cdots\breve{b}_{P+n-2}\breve{b}_{P+n-1} \xrightarrow{\text{OR}} 1.\breve{b}_{P+1}\cdots\breve{b}_{P+n-2}1\ .$$

もちろん，実際の計算にあたっては，小数点付き数 $1.x_1x_2\cdots x_{M-2}1$ および $1.t_1t_2\cdots t_{n-2}1$ は，整数 $1x_1x_2\cdots x_{M-2}1$ および $1t_1t_2\cdots t_{n-2}1$ に同一視した上で計算を行う。（このとき，MB*n*hash の圧縮過程のプログラム（コード）は [Ya8] の SSI*n*hash の圧縮過程のプログラムと同じものになる。）

４．２．２　攪乱過程アルゴリズムの計算機への実装（MB1024hash）

MB1024hash の攪乱過程のアルゴリズムの実装にあたっては，$(M_{2^{S-1}\tilde{W}})^{16}(\tilde{W})$ 中にある２つの $\tilde{W}$ の役割を $\hat{W}$ と $\tilde{W}$ に分離して $(M_{2^{S-1}\hat{W}})^{16}(\tilde{W})$ を計算する。$M_{2^{S-1}\hat{W}}(U)$ 中の定数 $\hat{W}$ および変数 U に与えるビット数は，それぞれ $Q = 224,\ L = 1024\ (= n)$ ビットである。圧縮過程と同様 $\hat{W}$ および U の MSB, LSB は常に１にしておく。このとき $\hat{W}$ および U を整数 $1x_1x_2\cdots x_{Q-2}1$ および $1u_1u_2\cdots u_{n-2}1$ と同一視すれば，$M_{2^{S-1}\hat{W}}(U)$ は以下の整数演算で計算される：

$$\begin{matrix} \hat{W}:\ 1x_1x_2\cdots x_{Q-2}1 \\ U:\ 1u_1u_2\cdots u_{n-2}1 \end{matrix} \xrightarrow{\text{mul}} \breve{b}_0\breve{b}_1\breve{b}_2\cdots\breve{b}_S\breve{b}_{S+1}\cdots\cdots\breve{b}_{n+Q-2}1 \xrightarrow{\text{shift } S}$$

$$\breve{b}_S\breve{b}_{S+1}\cdots\breve{b}_{S+n-2}\breve{b}_{S+n-1}\cdots\breve{b}_{n+Q-2}1 \xrightarrow{\text{cut}} \breve{b}_S\breve{b}_{S+1}\cdots\breve{b}_{S+n-2}\breve{b}_{S+n-1} \xrightarrow{\text{OR}} 1\breve{b}_{S+1}\cdots\breve{b}_{S+n-2}1\ .$$

注意．　(4.2) の $\tilde{W}$ を整数 $1w_1w_2\cdots w_{n-2}w_{n-1}$ と同一視し，

$$\tilde{x}_0\tilde{x}_1\tilde{x}_2\cdots\tilde{x}_{Q-2}\tilde{x}_{Q-1} \ =\ (1w_1\cdots w_{Q-1})\oplus(w_Q\ w_{Q+1}\cdots w_{2Q-1})\oplus\cdots\oplus(w_{kQ}\ w_{kQ+1}\cdots w_{n-1})$$

とおく（$\oplus$ (XOR) はビット毎にとり，k は $kQ < n \le (k+1)Q$ を満たす整数である)。このとき４．２．２節の $1x_1x_2\cdots x_{Q-2}1$ を $1\tilde{x}_1\tilde{x}_2\cdots\tilde{x}_{Q-2}1$ で置き換えれば [Ya8] にある SSI*n*hash の攪乱過程と同じアルゴリズムを得る。（$1\tilde{x}_1\tilde{x}_2\cdots\tilde{x}_{Q-2}1$ は $1w_1w_2\cdots w_{n-2}w_{n-1}$ の全ての情報を使って定められているのに対して，$1x_1x_2\cdots x_{Q-2}1 \ = 1w_1w_2\cdots w_{Q-2}1$ はそうではないことに注意せよ。MB*n*hash のアルゴリズムにおいて情報量の少ない $1x_1x_2\cdots x_{Q-2}1$ を敢えて使った理由は，MB32hash のアルゴリズムの単純性をそのまま踏襲して，安全性の解析を単純かつ容易にするためである。）

４．３　MB*n*hash, *n*=192, 256, ..., 2048, 4096，のパラメータ値とハッシュ値生成速度

４．３．１　MB*n*hash, *n*=192, 256, ..., 2048, 4096, のパラメータ値

MB*n*hash のアルゴリズムに基づいていろいろな長さのハッシュ値を生成するには，ハッシュ値長に対応してパラメータ L, M, E, P, Q, S の値を適宜決めればよい。我々が，ハッシュ値長 n （ビット）の MB*n*hash に対して使用したパラメータ値（バイト）は以下の通りである：

MB*n*hash	L (size of $W_k, \tilde{W}$)	M (size of $1.e$)	E (bytes Embeded.)	P	Q (size of $\hat{W}$)	S
160	20	8	8	5	12(8)	2
192	24	8	8	5	12(8)	2
256	32	8	8	5	12(8)	3
384	48	12	16	9	12	4
512	64	12	16	9	16	6
1024	128	20	32	17	28	12
2048	256	36	64	33	52	24
4096	512	68	128	65	96	48

Table 1．MB*n*hash に対するパラメータ値

ここで, 表中の括弧内の数値(=8)は SSI*n*hash のものである(SSI*n*hash では MB*n*hash より短くて済む; ４．２．２の「注意」参照)。

４．３．２　MB*n*hash, *n*=192, 256, ..., 2048, 4096, のハッシュ値生成速度

MB*n*hash がハッシュ値を生成する速度を，以下の２つの方法（(A)および(B)）で測定する。(A)は入力データ B が非常に長いときに，対応するハッシュ値を生成するのに要する時間を調べ，(B)は，入力データ B そのものは非常に短いが, 多数の入力データが存在する場合に, 対応する多数のハッシュ値を生成するのに要する時間を調べるものである。具体的には，(A)での入力データ B は，メモリ上にある全ての値が０(=0x00)である 1024^3 バイトの連続データである。また，(B)での入力データは，８バイトで構成され，８バイト整数としての値 k を持つ B_k，$k = 0,1,2,\cdots,10^7$，である。これらの入力データに対して，ハッシュ値を生成する時間の計測を３回行い，平均値を求めた。以下の表２および表３が結果である。なお，使用した計算機の環境は

OS: Windows 7 Professional 64-bit,　CPU: Xeon 3.1GHz,

メモリー：　8GB,　C コンパイラー：　Intel icl v.12

である。

Mbnhash →	160	192	256	384	512	1024	2048	4096	(SHA512)(1)	(SHA512)(2)
time(32-bit)	6.3	7	8.3	7.7	9.7	14.7	22.3	37	16.7	5.7
time(64-bit)	-	5.3	6	5.7	6.3	8.3	11.3	17.7	-	-

Table 2．メモリー上にある連続1024^3 バイトデータに対するハッシュ値の生成時間（秒）

MBnhash →	160	192	256	384	512	1024	2048	4096	(SHA512)(1)	(SHA512)(2)
time(32-bit)	6.7	8	9.3	13	22.7	80.3	274.7	965.7	26	6.7
time(64-bit)	-	5.3	7	10.3	11.3	40.7	128.7	400	-	-

Table 3. 8バイトからなる10^7個の入力データに対するハッシュ値の生成時間（秒）

表中で，(SHA512)(1), (SHA512)(2) は SHA512 のソースコードがそれぞれ NIST sts-1.8 および Gifford [Gi] であることを示す。また，(32-bit), (64-bit) は，MB*n*hash の計算中に現れる多倍長の乗算において，それぞれ (32-bit)×(32-bit)＝(64-bit) および (64-bit)×(64-bit)＝(128-bit) の整数乗算が使われていることを示している。

４．４　MB*n*hash が生成するハッシュ値の乱数性の検定

MB*n*hash, n =192, ... , 4096, が生成するハッシュ値の乱数性は TestU01 V.1.2.3 中にある Crush suite を用いて検証された（TestU01 は NIST の検定より強力である）。$\zeta_k, k = 0,1,2,\cdots$ を３．３節で導入された８バイトの入力データ B_k に対する MB*n*hash のハッシュ値とする。乱数性の検定は，この ζ_k の最初の３２ビットと全ビット（＝n ビット）についてそれぞれ行われた。検定の組 Crush は，検定の過程において，約 0x800000000 $\approx 3\times 10^{10}$ 個の３２ビット整数もしくは [0,1] 内に値を持つ倍精度実数を要求してくるので，倍精度実数が要求された場合には，３２ビットを倍精度変数の仮数部にセットし，指数部には $\cdots\times 2^0$ となるようにビットをセットして [1,2) 内の数値を作り，それから１を引いて[0,1] 内の倍精度実数を得るようにした。Crush による検定の結果は， "All tests were passed" であった。最も時間がかかった Crush テストは勿論 MB4096hash が生成するハッシュ値の最初の３２ビットを検定する場合であり，延べ時間で約 6600 時間であった（テストの実行は，Crush を構成する検定を幾つかのグループに分け，並列的に行った）。

５．アルゴリズムの観点から見た MB32hash の安全性

MB32hash, MB*n*hash, n=192, 256, ..., 4096, のアルゴリズムは本質的に同じなので，この節および次節で MB32hash の安全性について考察する。この節では，MB32hash のアルゴリズムの安全性を考察し，次節で，アルゴリズムを計算機に実装する際に発生する危険性について考察する。

本節では，アルゴリズムそのものの安全性を扱い，アルゴリズムがどのように実装されるかは考慮しないので，扱われる [1,2) 内の数は，十分なビット数を持って表されている（≈無限のビット長で表されている）と仮定する。

以下では，２つの異なる入力データ B および $\hat{B}$ に対して，どのような時にハッシュ値が同じに

なるかを考察する。ハッシュ値は，圧縮過程を経て撹乱過程により生成されるので，議論もそれぞれの過程にわけて行う。５．１節では，$B = B_1 B_2 \cdots B_N$ および $\hat{B} = \hat{B}_1 \hat{B}_2 \cdots \hat{B}_{\hat{N}}$ を圧縮したとき，圧縮後の値 w_N と $\hat{w}_{\hat{N}}$ がどのようなときに等しくなるかを調べる。５．２節では，異なる y と $\hat{y}$ に対して，どのようなときに撹乱後の値（ハッシュ値）ζ と $\hat{\zeta}$ が等しくなるかを調べる。

５．１　アルゴリズムの観点から見た圧縮過程の安全性

５．１．１　議論がどのように行われるかを示すため，まず $B = B_1$ および $\hat{B} = \hat{B}_1$ であるような簡単な場合を考察する。

$$M_{2^3 x}(t) \;=\; (2^3 x)t \mod [1,2) \;=\; (2^3)(xt) - \lfloor (2^3)(xt) \rfloor + 1 \tag{5.1}$$

であるので，

$$w_1 = M_{2^3 1.e}(w_0 \oplus B_1) \;=\; 2^3(1.e)(1.e \oplus B_1) - \lfloor 2^3(1.e)(1.e \oplus B_1) \rfloor + 1 \tag{5.2}$$

と表される。簡単のため $M_{2^3 1.e}$ の $2^3 1.e$ を γ で表す，すなわち，

$$\gamma \equiv 2^3(1.e) = 10.17462\cdots$$

とする。このとき (5.2) は

$$w_1 = M_\gamma(w_0 \oplus B_1) \;=\; \gamma(1.e \oplus B_1) - \lfloor \gamma(1.e \oplus B_1) \rfloor + 1 \tag{5.3}$$

となるので $w_1 = \hat{w}_1$ は，

$$\gamma(1.e \oplus B_1) - \lfloor \gamma(1.e \oplus B_1) \rfloor \;=\; \gamma(1.e \oplus \hat{B}_1) - \lfloor \gamma(1.e \oplus \hat{B}_1) \rfloor \tag{5.4}$$

となる。(5.4) を満たす異なる B_1 および $\hat{B}_1$ が存在するかどうかが問題となるが，我々はこの問題を次のように分解して考える。まず，

$$\gamma(1.e + t_1) - \lfloor \gamma(1.e + t_1) \rfloor \;=\; \gamma(1.e + \hat{t}_1) - \lfloor \gamma(1.e + \hat{t}_1) \rfloor \tag{5.5}$$

を満たす t_1 および $\hat{t}_1$ が区間 $(-1,1)$ の中にあるかどうか調べる。次に，もしそのような t_1 および $\hat{t}_1$ があれば，$1.e + t_1 = 1.e \oplus B_1$ および $1.e + \hat{t}_1 = 1.e \oplus \hat{B}_1$ を満たす８ビットの B_1，$\hat{B}_1$ があるかどうかを調べる。ここで注意すべきことは，B_1，$\hat{B}_1$ が８ビットであるので，t_1 および $\hat{t}_1$ が取り得る値の数は，それぞれ 2^8 個であるということである。

ここで，関数 $M_\gamma : [1,2) \to [1,2)$ の性質を調べておく。２節の図１は M_γ のグラフである。このグラフから，$M_\gamma(s) = 1$ となるような s は，$s_i = i/\gamma,\ i = 11, 12, \cdots, 20,$ であることが分かる。具体的には

$$s_{11} = 1.08112\cdots,\quad s_{12} = 1.17940\cdots,\quad s_{13} = 1.27768\cdots,\quad \cdots\cdots,\quad s_{20} = 1.96567\cdots$$

などとなる。便宜上，$s_{10} = 1$，$s_{21} = 2$ とする。さらに，区間 I_i を $I_i = [s_i, s_{i+1}),\ i = 10, \cdots, 20,$ で

定める。この時，以下のことが容易に分かる：

$$[1,2) = I_{10} \cup I_{11} \cup \cdots \cup I_{20}, \tag{5.6a}$$

$$M_\gamma(1) = 1.17462\cdots, \qquad \lim_{s\to 2-0} M_\gamma(s) = 1.34925\cdots\ , \tag{5.6b}$$

$$s \in I_i \text{ に対し，} \quad \lfloor \gamma s \rfloor = i, \quad i = 10, \cdots, 20, \tag{5.6c}$$

$$\lim_{s\to s_i - 0} M_\gamma(s) = 2, \quad i = 11, \cdots, 20, \tag{5.6d}$$

$$M_\gamma(s) \text{ のグラフは } I_\gamma,\ i = 10, \cdots, 20, \text{ 上で直線で，増加関数である，} \tag{5.6e}$$

$$s \in (s_i, s_{i+1}), \quad i = 10, \cdots, 20\,, \text{ に対し，} \ \frac{d}{ds}\{M_\gamma(s)\} = \gamma \text{ である。} \tag{5.6f}$$

話を元に戻す。$1.e + t_1$ と $1.e + \hat{t}_1$ が $1.e \oplus B_1$ および $1.e \oplus \hat{B}_1$ の形で実現されるためには，$1.e + t_1$ と $1.e + \hat{t}_1$ は $[1,2)$ の中になければならない。よって $1.e + t_1 \in I_p$ と $1.e + \hat{t}_1 \in I_q$ となる $p,\ q$ が存在する必要がある。このとき，(5.6c) より (5.5) は

$$\gamma(1.e + t_1) - \gamma(1.e + \hat{t}_1) \ = \ p - q \tag{5.7}$$

と表される。ここで (5.7) を，変数 $\tilde{\gamma}$ に関する一次方程式

$$\tilde{\gamma}(t_1 - \hat{t}_1) + (-p + q) \ = \ 0 \tag{5.8}$$

という観点で見る。すると，方程式 (5.8) で，t_1 と $\hat{t}_1$ が 区間 $(-1, 1)$ の範囲内を動き，$p,\ q$ が $\{10, \cdots, 20\}$ 内の値を動くとき，$\gamma \equiv 2^3(1.e)$ は (5.8) の解 $\tilde{\gamma}(t_1, \hat{t}_1, p, q)$ の一つとして含まれていなくてはならない。もし $t_1 = \hat{t}_1$ であれば（これは $B_1 = \hat{B}_1$ の場合に相当する），$p = q$ となるので，どのような $\tilde{\gamma}$ も (5.8) の解となりうるので，もちろん $\gamma \equiv 2^3(1.e)$ も (5.8) の解に含まれ，何の問題も起こらない。次に $t_1 \neq \hat{t}_1$，とし，$1.e + t_1$ と $1.e + \hat{t}_1$ が $1.e \oplus B_1$ および $1.e \oplus \hat{B}_1$ の形で実現されたとする。この場合 t_1 と $\hat{t}_1$ が取り得る形は

$$(+ \text{ or } -)\, 0.0000000\, b_8 b_9 \cdots b_{15} 000\cdots \quad （2進）$$

の形になる。したがってt_1 と $\hat{t}_1$ が (5.8) の中で取り得る値の数は，高々 $2^8 \times 2 = 2^9$ である。また，$p,\ q$ が取り得る値は，10 から 20 までの整数であるので，(5.8) の形で現れる一次方程式の総数は高々 $2^9 \times 2^9 \times 11 \times 11$ である。このような状況において，$t_1 \neq \hat{t}_1$ のとき，

$$\gamma \equiv 2^3(1.e) = 2^3\left\{1 + \frac{1}{10}\left(1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots\right)\right\}$$

が，高々 $2^9 \times 2^9 \times 11 \times 11$ 個現れる (5.8) の一次方程式の解になり得ないことは容易に分かる。以上のことから，$B_1 \neq \hat{B}_1$ であって，$w_1 = \hat{w}_1$ となるような $B = B_1$ と $\hat{B} = \hat{B}_1$ は存在しないことが分かる。

５．１．２ 次に少し複雑な $B = B_1 B_2$ および $\hat{B} = \hat{B}_1$ であるような場合を考える。この場合 B および $\hat{B}$ の圧縮は次のように行われる：

$$w_1 = M_\gamma(w_0 \oplus B_1) \;=\; \gamma(1.e \oplus B_1) - \lfloor \gamma(1.e \oplus B_1) \rfloor + 1 \quad \text{((5.3) より)},$$
$$w_2 = M_\gamma(w_1 \oplus B_2) \;=\; \gamma(w_1 \oplus B_2) - \lfloor \gamma(w_1 \oplus B_2) \rfloor + 1,$$
$$\hat{w}_1 = M_\gamma(w_0 \oplus \hat{B}_1) \;=\; \gamma(1.e \oplus \hat{B}_1) - \lfloor \gamma(1.e \oplus \hat{B}_1) \rfloor + 1.$$

したがって，もし $w_2 = \hat{w}_1$ が成り立ったとすると，

$$\gamma(w_1 \oplus B_2) - \lfloor \gamma(w_1 \oplus B_2) \rfloor + 1 \;=\; \gamma(1.e \oplus \hat{B}_1) - \lfloor \gamma(1.e \oplus \hat{B}_1) \rfloor$$

すなわち

$$\gamma\{(\gamma(1.e \oplus B_1) - \lfloor \gamma(1.e \oplus B_1) \rfloor + 1) \oplus B_2\} - \lfloor \gamma\{(\gamma(1.e \oplus B_1) - \lfloor \gamma(1.e \oplus B_1) \rfloor + 1) \oplus B_2\} \rfloor$$
$$= \gamma(1.e \oplus \hat{B}_1) - \lfloor \gamma(1.e \oplus \hat{B}_1) \rfloor \qquad (5.9)$$

が成り立つことになる。ここで５．１．１節と同様に, (5.9) を満たす B_1，B_2 と $\hat{B}_1$ を直接探さずに，まず

$$\gamma\{(\gamma(1.e + t_1) - \lfloor \gamma(1.e + t_1) \rfloor + 1) + t_2\} - \lfloor \gamma\{(\gamma(1.e + t_1) - \lfloor \gamma(1.e + t_1) \rfloor + 1) + t_2\} \rfloor$$
$$= \gamma(1.e + \hat{t}_1) - \lfloor \gamma(1.e + \hat{t}_1) \rfloor \qquad (5.10)$$

を満たす t_1，t_2 および $\hat{t}_1$ が 区間 $(-1,1)$ の中にあるかどうかを調べ，次に，もしそのような t_1 ，t_2 および $\hat{t}_1$ があれば $1.e + t_1$，$(\gamma(1.e + t_1) - \lfloor \gamma(1.e + t_1) \rfloor + 1) + t_2$ および $1.e + \hat{t}_1$ が が ，$1.e \oplus B_1$ ，$(\gamma(1.e \oplus B_1) - \lfloor \gamma(1.e \oplus B_1) \rfloor + 1) \oplus B_2$ および $1.e \oplus \hat{B}_1$ により実現できるかどうかを調べる。そこで t_1，t_2，$\hat{t}_1$ について

$$1.e + t_1 \in I_{p_1}, \quad (\gamma(1.e + t_1) - \lfloor \gamma(1.e + t_1) \rfloor + 1) + t_2 \in I_{p_2} \quad \text{および} \quad 1.e + \hat{t}_1 \in I_q$$

であったとする。このとき (5.10) は,

$$\gamma\{(\gamma(1.e + t_1) - p_1 + 1) + t_2\} - p_2 = \gamma(1.e + \hat{t}_1) - q \qquad (5.11)$$

となる。ここで (5.11) を (5.8) と同様，変数 $\tilde{\gamma}$ に関する二次方程式

$$\tilde{\gamma}^2(1.e + t_1) + \tilde{\gamma}(-p_1 + 1 - 1.e + t_2 - \hat{t}_1) - p_2 + q = 0 \qquad (5.12)$$

と見る。すると，方程式 (5.12) の解 $\tilde{\gamma}(t_1, t_2, \hat{t}_1, p_1, p_2, q)$ は $t_1, t_2, \hat{t}_1$ ，が 区間 $(-1,1)$ の範囲を動き p_1，p_2，q が $\{10, \cdots, 20\}$ の値を動くとき，$\gamma \equiv 2^3(1.e)$ を解の一つとして含まなければならない。他方で，(5.12) の解 $\tilde{\gamma}(t_1, t_2, \hat{t}_1, p_1, p_2, q)$ に対して

$$1.e + t_1, \qquad \{(\tilde{\gamma}(1.e + t_1) - \lfloor \tilde{\gamma}(1.e + t_1) \rfloor + 1) + t_2\}, \qquad 1.e + \hat{t}_1$$

が

$$1.e \oplus B_1, \qquad \{(\tilde{\gamma}(1.e \oplus B_1) - \lfloor \tilde{\gamma}(1.e \oplus B_1) \rfloor + 1) \oplus B_2\}, \qquad 1.e \oplus \hat{B}_1$$

により実現されるとすれば，t_1，t_2，$\hat{t}_1$ は $\pm 0.0000000 b_8 b_9 \cdots b_{15} 000 \cdots$ の形でなければならないので，方程式 (5.12) として現れる数式の数は，高々 $(2^9)^3 \times 11^3$ である。このような数の二次方程式の解の中に，$\gamma \equiv 2^3(1.e) = 2^3\left\{1 + \frac{1}{10}\left(1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots\right)\right\}$ を見つけることは不可能である。

よって，もし $w_2 = \hat{w}_1$ とすると，このような不可能が可能となる必要性が出てきて，矛盾が生じる。これから，$w_2 = \hat{w}_1$ となるような $B = B_1 B_2$ と $\hat{B} = \hat{B}_1$ は存在しないことが分かる。

５．１．３ 一般に $B = B_1 B_2 \cdots B_N$ に対しては，上と同様に

$$w_k = \gamma(w_{k-1} + t_k) - p_k + 1 \equiv \gamma(w_{k-1} + t_k) + \tilde{p}_k \ , \qquad k = 1, 2, \cdots, N \ , \tag{5.13}$$

となる。 ここで

$$p_k = \lfloor \gamma(w_{k-1} + t_k) \rfloor \ , \qquad \tilde{p}_k = -p_k + 1$$

である。よって w_N は,

$$\begin{aligned} w_N &= \gamma\{(\gamma \cdots \gamma\{(\gamma\{(\gamma(w_0 + t_1) + \tilde{p}_1) + t_2\} + \tilde{p}_2) + t_3\} + \tilde{p}_3 \cdots + \tilde{p}_{N-1}) + t_N\} + \tilde{p}_N \\ &= \gamma^N (w_0 + t_1) + \gamma^{N-1}(\tilde{p}_1 + t_2) + \gamma^{N-2}(\tilde{p}_2 + t_3) + \cdots + \gamma(\tilde{p}_{N-1} + t_N) + \tilde{p}_N \end{aligned} \tag{5.14}$$

と表される。ただし $w_0 = 1.e$, $\tilde{p}_k \in \{-19, -18, \cdots, -9\}$である。したがって $B = B_1 B_2 \cdots B_N$ と $\hat{B} = \hat{B}_1 \hat{B}_2 \cdots \hat{B}_{\hat{N}}$, $N \geq \hat{N}$, に対し $w_N = \hat{w}_{\hat{N}}$ が成り立つとすれば, γ が満たすべき方程式は,

$$\begin{aligned} &\tilde{\gamma}^N (1.e + t_1) + \tilde{\gamma}^{N-1}(\tilde{p}_1 + t_2) + \tilde{\gamma}^{N-2}(\tilde{p}_2 + t_3) + \cdots + \tilde{\gamma}(\tilde{p}_{N-1} + t_N) + \tilde{p}_N \\ &= \tilde{\gamma}^{\hat{N}} (1.e + \hat{t}_1) + \tilde{\gamma}^{\hat{N}-1}(\tilde{q}_1 + \hat{t}_2) + \tilde{\gamma}^{\hat{N}-2}(\tilde{q}_2 + \hat{t}_3) + \cdots + \tilde{\gamma}(\tilde{q}_{\hat{N}-1} + \hat{t}_{\hat{N}}) + \tilde{q}_{\hat{N}} \end{aligned}$$

である；すなわち, $N = \hat{N}$ のときは

$$\begin{aligned} &\tilde{\gamma}^N (t_1 - \hat{t}_1) + \tilde{\gamma}^{N-1}(\tilde{p}_1 - \tilde{q}_1 + t_2 - \hat{t}_2) + \tilde{\gamma}^{N-2}(\tilde{p}_2 - \tilde{q}_2 + t_3 - \hat{t}_3) + \cdots \\ &\qquad + \tilde{\gamma}(\tilde{p}_{N-1} - \tilde{q}_{N-1} + t_N - \hat{t}_N) + (\tilde{p}_N - \tilde{q}_N) = 0 \end{aligned} \tag{5.15}$$

であり, $N > \hat{N}$ のときは

$$\begin{aligned} &\tilde{\gamma}^N (1.e + t_1) + \tilde{\gamma}^{N-1}(\tilde{p}_1 + t_2) + \cdots + \tilde{\gamma}^{\hat{N}+1}(\tilde{p}_{N-\hat{N}-1} + t_{N-\hat{N}}) \\ &\qquad + \tilde{\gamma}^{\hat{N}}(\tilde{p}_{N-\hat{N}} - 1.e + t_{N-\hat{N}+1} - \hat{t}_1) + \tilde{\gamma}^{\hat{N}-1}(\tilde{p}_{N-\hat{N}+1} - \tilde{q}_1 + t_{N-\hat{N}+2} - \hat{t}_2) + \cdots \\ &\qquad + \tilde{\gamma}(\tilde{p}_{N-1} - \tilde{q}_{\hat{N}-1} + t_N - \hat{t}_{\hat{N}}) + (\tilde{p}_N - \tilde{q}_{\hat{N}}) = 0 \end{aligned} \tag{5.16}$$

である。そこで $\tilde{\gamma} = \tilde{\gamma}(t_1, \cdots, t_N, \hat{t}_1, \cdots \hat{t}_{\hat{N}}, \tilde{p}_1, \cdots, \tilde{p}_N, \tilde{q}_1, \cdots, \tilde{q}_{\hat{N}})$ を (5.15) あるいは (5.16) の解とする。もし $B = \hat{B}$であれば, 任意の k に対して $t_k = \hat{t}_k$ となるので $\gamma \equiv 2^3(1.e)$ を含む任意の $\tilde{\gamma}$ が (5.15) の解となりうるので特に問題は起こらない。次に $B \neq \hat{B}$ の場合を考えてみる。すると５．１．２節と同様な議論により, (5.15) あるいは (5.16) の形で現れることの出来る方程式の数は，高々 $\left(2^9\right)^{N+\hat{N}} \times 11^{N+\hat{N}}$ である。この様な制約のもとで $\gamma \equiv 2^3(1.e) = 2^3\left\{1 + \frac{1}{10}\left(1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots\right)\right\}$ の形を見れば, γ をそのような数の代数方程式(5.15) あるいは (5.16) の解の中に見つけることは不可能であることが分かる。かくして, $1.e$ を含む [1,2) の各数を表すビット数が無限に近ければ, 圧縮値 w_N と $\hat{w}_{\hat{N}}$ が等しくなるような，有限長の異なる $B = B_1 B_2 \cdots B_N$ および $\hat{B} = \hat{B}_1 \hat{B}_2 \cdots \hat{B}_{\hat{N}}$ を見つけることは大変困難（不可能）であることが分かった。

５．１．４ (5.14) より

$$\frac{\partial W_N}{\partial t_k} = \gamma^{N-k+1} = \left(2^3(1.e)\right)^{N-k+1} \tag{5.17}$$

であるので，ほとんどすべての $(t_1, t_2, \cdots, t_N)$ において，$\Delta_1, \Delta_2, \cdots, \Delta_N$ が十分に小さいとき

$$W_N(t_1+\Delta_1, t_2+\Delta_2, \cdots, t_N+\Delta_N) - W_N(t_1, t_2, \cdots, t_N)$$

$$\approx \sum_{k=1}^{N} \frac{\partial W_N}{\partial t_k}\Delta_k = \sum_{k=1}^{N} (2^3(1.e))^{N-k+1}\Delta_k$$

となる。このことから $t_k+\Delta_k$，$k=1,\cdots,N$，の Δ_k をうまく調整して，W_N の値をコントロールできると考えるかも知れないが, 如何せん $t_k+\Delta_k$ は $t_k \oplus B_k$ により実現されなくてはならないので，Δ_kの取り得る値は離散的な 2^8 個のみである。また (5.17) は，$N-k$ が大きくなるにつれて $\frac{\partial W_N}{\partial t_k}$ の値が指数的に大きくなり，$N-h$ が小さいような $\frac{\partial W_N}{\partial t_h}$ に比して極端に大きな値となることを表している。このことから，$\Delta_1, \cdots, \Delta_N$ をうまく調節して W_N の値を精密にコントロールしようとすることは困難であることが分かる。

５．２　アルゴリズムの観点から見た攪乱過程の安全性

５．２．１　攪乱は $z = M^{16}_{2^3 y}(y)$ により行われるので値 y の攪乱過程は，全バイトが値０を持つ１６バイトの入力バイト列 $B = 00\cdots00$ の 写像 $M_{2^3 y}$ による圧縮過程とみなすことが出来る;すなわち $M^{16}_{2^3 y}(y)$ は次のように計算される（(3.1) 参照）:（写像が $M_{2^3 1.e}$ でなく $M_{2^3 y}$ であることに注意する）

$$u_0 = y, \quad u_k = M_{2^3 y}(u_{k-1} \oplus 0), \quad k = 1,2,\cdots,16, \qquad z = u_{16}\ . \tag{5.18}$$

したがって MB32hash の圧縮過程の解析で行われた議論をここでも利用することができる。以下に, $z \equiv u_{16}(y) = M^{16}_{2^3 y}(y)$ のグラフをイメージできるよう，$u_1(y) = M_{2^3 y}(y)$ のグラフと, $u_2(y) = M^{2}_{2^3 y}(y)$ のグラフ（一部分）を載せておく。

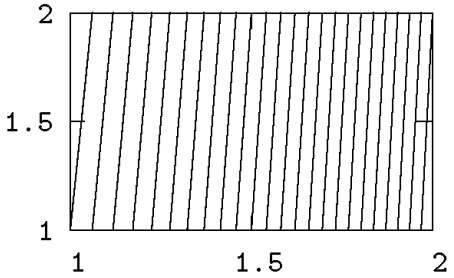


図３．$u_1(y) = M_{2^3 y}(y)$ のグラフ

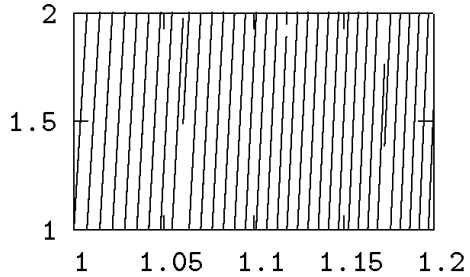


図４．$u_2(y) = M^{2}_{2^3 y}(y)$ のグラフ(一部分)

関数 $u_{16}(y)$ が $[1,2)$ 上で右連続であること，ほとんど至る所で微分可能である（→したがって連

続である）ことは容易に分かる。また (5.18) と(5.1) から

$$u_0 = y, \quad u_k(y) \;=\; 2^3 y u_{k-1}(y) - \lfloor 2^3 y u_{k-1}(y) \rfloor + 1 \;=\; 2^3 y u_{k-1}(y) + \breve{p}_k \tag{5.19}$$

となる；ここで $\breve{p}_k = -\lfloor 2^3 y u_{k-1}(y) \rfloor + 1$ であるが，$y, u_{k-1} \in [1,2)$ であるので，

$$\breve{p}_k \in \{-30, -29, \cdots, -7\}$$

である。この $\breve{p}_k$ は $u_{16}(y)$ が微分可能であるような y の近傍で値が変化しないので，(5.19) より $k = 1, 2, \cdots, 16$ について

$$u_k'(y) = 2^3 u_{k-1}(y) + 2^3 y u_{k-1}'(y),$$
$$u_k''(y) = 2^4 u_{k-1}'(y) + 2^3 y u_{k-1}''(y)$$

が成り立つ。（u_k'，u_k'' は正の値であることに注意する。）よって $u_{16}'(y)$ は，ほとんど全ての y について

$$u_{16}'(y) \;=\; \left\{ \sum_{k=0}^{15} 2^3 (2^3 y)^k u_{15-k}(y) \right\} + 2^3 (2^3 y)^{15} y$$

と表される。$y, u_k \in [1,2)$ であるので，上式から

$$\sum_{k=0}^{15} 2^3 (2^3 y)^k \;\le\; u_{16}'(y) \;<\; \sum_{k=0}^{15} 2^4 (2^4)^k + (2^4)^{16}$$

という不等式が得られる。この不等式から，さらに

$$\frac{2^{48} - 1}{1 - 2^{-3}} \;\le\; u_{16}'(y) \;<\; \frac{2^{68} - 1}{1 - 2^{-4}} \tag{5.20}$$

という評価が得られる。この不等式は後程５．２．３節で使われる。

５．２．２ 以下では，同じハッシュ値 ς と $\hat{\varsigma}$ を持つ，異なる y と $\hat{y}$ を見つけることが大変困難であることを示す。（ς と $\hat{\varsigma}$ は，それぞれ $z = M^{16}_{2^3 y}(y)$ と $\hat{z} = M^{16}_{2^3 \hat{y}}(\hat{y})$ から取り出されることに注意する。）攪乱過程は (5.14) で，$w_0 = u_0 (= y)$ および $t_k = 0$（$\leftarrow B_k = 0$ に対応）とおいたものに相当するので，$\breve{p}_k$ を(5.19) で定義したものとすれば

$$z \equiv z(y) \;=\; (2^3 y)^{16} y + (2^3 y)^{15} \breve{p}_1 + (2^3 y)^{14} \breve{p}_2 + \cdots + (2^3 y) \breve{p}_{15} + \breve{p}_{16} \tag{5.21}$$

となる。同様に $\hat{z}$ についても

$$\hat{z} \equiv \hat{z}(\hat{y}) \;=\; (2^3 \hat{y})^{16} \hat{y} + (2^3 \hat{y})^{15} \breve{q}_1 + (2^3 \hat{y})^{14} \breve{q}_2 + \cdots + (2^3 \hat{y}) \breve{q}_{15} + \breve{q}_{16} \tag{5.22}$$

となる。ここで $\breve{q}_k \in \{-30, -29, \cdots, -7\}$ は $\breve{p}_k$ と同様に式

$$\hat{u}_0 = \hat{y}, \quad \hat{u}_k = 2^3 \hat{y} \hat{u}_{k-1} - \lfloor 2^3 \hat{y} \hat{u}_{k-1} \rfloor + 1 \equiv 2^3 \hat{y} \hat{u}_{k-1} + \breve{q}_k$$

を通じて定義される。（$\hat{z}(\hat{y})$, $\hat{u}_k(\hat{y})$ は，正確には $z(\hat{y})$, $u_k(\hat{y})$ と表されるべきものであるが，短縮して書かれたときに $\hat{y}$ による依存性が分かり易いように，あえて $\hat{z}$, $\hat{u}_k$ という表記を使用した。）ハッシュ値 ς と $\hat{\varsigma}$ は z と $\hat{z}$ から

$$\varsigma = \left\lfloor 2^{32} \left(2^{11} z - \lfloor 2^{11} z \rfloor \right) \right\rfloor \quad \text{および} \quad \hat{\varsigma} = \left\lfloor 2^{32} \left(2^{11} \hat{z} - \lfloor 2^{11} \hat{z} \rfloor \right) \right\rfloor$$

により取り出されるので，$\varsigma = \hat{\varsigma}$ は

$$\left\lfloor 2^{32}\left(2^{11}z-\left\lfloor 2^{11}z\right\rfloor\right)\right\rfloor = \left\lfloor 2^{32}\left(2^{11}\hat{z}-\left\lfloor 2^{11}\hat{z}\right\rfloor\right)\right\rfloor \tag{5.23}$$

と表される。この式から，$\zeta=\hat{\zeta}$ であるためには,

$$\left| 2^{32}\left(2^{11}z-\left\lfloor 2^{11}z\right\rfloor\right) - 2^{32}\left(2^{11}\hat{z}-\left\lfloor 2^{11}\hat{z}\right\rfloor\right)\right| < 1$$

であることが**必要な条件**である。よって $\zeta=\hat{\zeta}$ であるためには

$$\left| \left(z-2^{-11}\left\lfloor 2^{11}z\right\rfloor\right) - \left(\hat{z}-2^{-11}\left\lfloor 2^{11}\hat{z}\right\rfloor\right)\right| < 2^{-43} \tag{5.24}$$

が満たされている必要がある。

5.2.3 $z(y)$ の y に関する局所的な連続性から, y と $\hat{y}$ がお互いに非常に近ければ $\varsigma=\hat{\varsigma}$ となる（(5.23) が満たされる）ことは明らかであるが, 非常に近いとは数値的にどの程度のことなのかをここで考察してみる。y と $\hat{y}=y+\eta,\ \eta>0,$ が非常に近いと言うからには，y と $\hat{y}$ は少なくとも

$$\breve{p}_k=\breve{q}_k,\ k=1,\cdots,16, \tag{5.25}$$

と

$$\left\lfloor 2^{11}z\right\rfloor = \left\lfloor 2^{11}\hat{z}\right\rfloor \tag{5.26}$$

を満たしていると考えられる。この場合は $\eta<2^{-91}$ でなければならない。実際, (5.21), (5.22), (5.25) と u_{16} の局所単調性より，$u_{16}(y)$ と $\hat{u}_{16}(\hat{y})$ は u_{16} のグラフ上で同じ分枝(brach)にある。すると，u'_{16} の局所単調性から

$$\hat{z}(\hat{y})-z(y) = \hat{u}_{16}(\hat{y})-u_{16}(y) = u_{16}(\hat{y})-u_{16}(y) \geq u'_{16}(y)\eta$$

となり，これと (5.20) から

$$\frac{2^{48}-1}{1-2^{-3}}\eta \leq \hat{z}-z \tag{5.27}$$

を得る。さらに (5.26) により， 条件 (5.24) は

$$\left| z-\hat{z}\right| < 2^{-43} \tag{5.28}$$

と単純化される。ここで，もし $\eta\geq 2^{-91}$ とすると,

$$\frac{2^{48}-1}{1-2^{-3}}\eta \geq \frac{2^{48}-1}{1-2^{-3}}\cdot 2^{-91} = \frac{8}{7}\left(2^{-43}-2^{-91}\right) > 2^{-43} \tag{5.29}$$

となり, (5.27) より $z-\hat{z} > 2^{-43}$ となるが, これは $\zeta=\hat{\zeta}$ であるための条件 (5.28) に反する。したがって, (5.25) と (5.26) が成り立つとすれば，η は $\eta<2^{-91}$ でなければならない。しかしながら $\eta<2^{-91}$ とすると問題が生じる。すなわち y と $\hat{y}=y+\eta$ をコンピュータに実装する際に使うことができるビット数（MB32hash では３２ビット）は，$\eta<2^{-91}$ のとき y と $\hat{y}$ に違いを表現するには不十分であると言うことである。よって，$\zeta=\hat{\zeta}$ であって，$|y-\hat{y}|\geq 2^{-91}$ であるような y と $\hat{y}$ を求めなくてはならないが，これをしようとすると，（$|y-\hat{y}|\geq 2^{-91}$ のとき (5.25) and/or (5.26) が成り立たないため,） (5.25) and/or (5.26) が満たされないという条件のもとで

(5.23) を解かなくてはならない。

５．２．４　ここでは，(5.23) を解くことが大変困難であることを示す。まず，３２ビット整数 $\hat{\zeta}$ に対して $\lfloor A \rfloor = \hat{\zeta}$ は $\hat{\zeta} \le A < \hat{\zeta}+1$ と同値であることに注意する。すると (5.23) は

$$\hat{\zeta} \;\le\; 2^{32}\left(2^{11}z - \left\lfloor 2^{11}z \right\rfloor\right) \;<\; \hat{\zeta}+1$$

すなわち，

$$2^{-11}\left\lfloor 2^{11}z \right\rfloor + 2^{-43}\hat{\zeta} \;\le\; z \;<\; 2^{-11}\left\lfloor 2^{11}z \right\rfloor + 2^{-43}(\hat{\zeta}+1)$$

と書かれ，これは (5.21) より

$$\begin{aligned} & 2^{-11}\left\lfloor 2^{11}z \right\rfloor + 2^{-43}\hat{\zeta} \\ \le\; & 2^{-11}\left\lfloor 2^{11}z \right\rfloor + 2^{-43}\hat{\zeta} \;\le\; (2^3)^{16}y^{17} + (2^3)^{15}\breve{p}_1 y^{15} + (2^3)^{14}\breve{p}_2 y^{14} + \cdots + (2^3)\breve{p}_{15}y + \breve{p}_{16} \\ <\; & 2^{-11}\left\lfloor 2^{11}z \right\rfloor + 2^{-43}(\hat{\zeta}+1) \end{aligned} \tag{5.30}$$

となる。（(5.30) を満たしている y は，$\hat{y}$ から生成されるハッシュ値 $\hat{\zeta}$ と同じハッシュ値を生成することに注意する。）ここで，(5.30) を満たすような y と $\hat{\zeta}$ を見つけることを考える。まず任意に $\hat{\zeta}$ を固定しておく。(5.30) において，y に関して連続な所では y の微少な変化は

$$\breve{p}_k,\ k = 1, \cdots, 16, \quad \text{と} \quad \left\lfloor 2^{11}z \right\rfloor$$

の値を変えないので，もし y が (5.30) を満たすとすると，y は 次の $\breve{y}$ に関する不等式

$$\begin{aligned} & 2^{-11}Z + 2^{-43}\hat{\zeta} \\ \le\; & (2^3)^{16}\breve{y}^{17} + (2^3)^{15}P_1\breve{y}^{15} + (2^3)^{14}P_2\breve{y}^{14} + \cdots + (2^3)P_{15}\breve{y} + P_{16} \\ <\; & 2^{-11}Z + 2^{-43}(\hat{\zeta}+1) \end{aligned} \tag{5.31}$$

の，$Z = \left\lfloor 2^{11}z \right\rfloor$ および $P_k = \breve{p}_k,\ k = 1, \cdots, 16,$ としたときの解でなければならない。このことより，ハッシュ値 ζ が $\hat{\zeta}$ であるような y を見つけようとすると，$P_k \in \{-30, -29, \cdots, -7\}$ であり，かつ MSB =1 であるような１２ビットの整数変数 Z から構成される不等式 (5.31) を解かなければならない。しかしながら，５次以上の代数方程式を解く一般的なアルゴリズムは無いので，(5.31) を解くことは非常に難しい。$\hat{\zeta}$ は任意に固定されたものであるので，どのような $\hat{\zeta}$ についても (5.31) を解くことが非常に困難となり，従って (5.23) を理論的に解くと言うことは大変困難であることが分かる。（(5.31) の数値解析については６．２節で扱う。）

最後に２つほど注意を与えておく。

（１）ある $P_k,\ k = 1, \cdots, 16,$ と Z に対して (5.31) が解けたとして，その解を $\breve{y}$ とするとき，$\breve{y}$ から (5.19) により $\breve{p}_k$ が定まる。$\breve{y}$ がハッシュ値 $\hat{\zeta}$ を生成するためには，$P_k = \breve{p}_k$ でなければならない。$\breve{y}$ から (5.21) により定まる z に対して，$Z = \left\lfloor 2^{11}z \right\rfloor$ でなければならないことも同様である。これらが満たされないときは，$\breve{y}$ を (5.30) の解として使うことが出来ない。しかしなが

ら， P_k，$k=1,\cdots,16,$ と Z を互いに独立に選んで (5.31) の解 $\breve{y}$ を得たとしても，$\breve{y}$ が上述の条件（ $P_k=\breve{p}_k$ ， $Z=\lfloor 2^{11}z \rfloor$ ）をパスする確率は大変低い。

（2）仮に同じハッシュ値 ζ と $\hat{\zeta}$ を生じる y と $\hat{y}$ を見つけたとしても，５．１節の議論から分かるように, 圧縮により y と $\hat{y}$ を生成する入力データ B および $\hat{B}$ を見つけるのは大変困難である。

６．アルゴリズムの実装という観点から見た MB32hash の安全性

５節で行われた議論は，[1,2) 内の各数を表すビット数は十分に長い，という仮定のもとで行われてきた。しかしながら，数を計算機に実装する際にはビット数が制限される。実際，MB32hash は２．２節で述べた３２ビットの数体系で実装されている。そこで，ここでは MB32hash アルゴリズムを有限ビット数による数体系で実装した場合に発生する特有の問題について議論する。議論は **[Ya8]** で行われた SSRhash の安全性の議論に基づくが, MB32hash を構成する写像 $M_\beta:[1,2)\to[1,2)$ が単純であることから，SSRhash の場合に比してより単純で明快な議論が可能になる。

６．１　アルゴリズムの実装という観点から見た圧縮過程の安全性

この節では [1,2) 内の各数は ２．２節で述べた３２ビット系で $1b_1b_2b_3\cdots b_{31}$ のように表されるものとする。

６．１．１ 最初に $w_{k-1}\oplus B_k$ において $\oplus$(XOR) をとる w_{k-1} での位置が，LSB(least significant bit，最下位ビット) を含む $b_{24}\cdots b_{31}$ である場合には，異なる入力バイト列 B ，$\hat{B}$ で圧縮後の値が等しくなる（$w_N=\hat{w}_{\hat{N}}$ となる）ものが存在することを示す。

B ＝(00)(00)（１６進表現，２バイト）とすると，(3.1) の圧縮過程は

$$w_0=\text{a2cb4411}\xrightarrow{\text{XOR}}\text{a2cb44}(1\oplus 0)(1\oplus 0)=\text{a2cb4411}\xrightarrow{\text{mul}}\text{6785e38a890f0921}$$

$$\xrightarrow{\text{shift4}}\text{785e38a890f0921}\xrightarrow{\text{OR}}\text{f85e38a890f0921}\xrightarrow{\text{cut}}\text{f85e38a8}=w_1\xrightarrow{\text{XOR}}$$

$$\text{f85e38}(\text{a}\oplus 0)(8\oplus 0)=\mathbf{f85e38a8}\xrightarrow{\text{mul}}\text{9df0d49ac2866328}\xrightarrow{\text{shift 4}}$$

$$\text{df0d49ac2866328}\xrightarrow{\text{OR}}\text{df0d49ac2866328}\xrightarrow{\text{cut}}\text{df0d49ac}=w_2$$

となる。一方，$\hat{B}$ ＝(01)(36) に対しては,

$$w_0 = \text{a2cb4411} \xrightarrow{\text{XOR}} \text{a2cb44}(1\oplus 0)(1\oplus 1) = \text{a2cb4410} \xrightarrow{\text{mul}} \text{6785e389e643c510}$$

$$\xrightarrow{\text{shift4}} \text{785e389e643c510} \xrightarrow{\text{OR}} \text{f85e389e643c510} \xrightarrow{\text{cut}} \text{f85e389e} = \hat{w}_1 \xrightarrow{\text{XOR}}$$

$$\text{f85e38}(9\oplus 3)(\text{e}\oplus 6) = \mathbf{f85e38a8}\,(B \text{ と同じ}) \xrightarrow{\text{mul}} \cdots \xrightarrow{\text{cut}} \text{df0d49ac} = \hat{w}_2$$

となり，$w_2 = \hat{w}_2$ となる。これは，w_1 と $\hat{w}_1$ とで異なるビットの部分が，次の $\oplus$ で修正可能な位置にあることが原因になっている。この問題を避けるには，w_{k-1} において $\oplus$ をとる位置を，LSB を含む下位ビットを避けるようにすれば良い。例えば (3.2) のように w_{k-1} の $b_8\cdots b_{15}$ の位置で $\oplus$ をとれば，b_{16} 以降は $\oplus$ の操作で触られることがないので，w_{k-1} の $b_{16}\cdots b_{31}$ と $\hat{w}_{k-1}$ の $\hat{b}_{16}\cdots\hat{b}_{31}$ の違いはそのまま残されることになる。

６．１．２ 次に w_0 と x がともに 0x80013000 で，乗算後の左シフトのサイズが１６ビットである場合を考える。また，$\oplus$ は w_{k-1} の $b_8\cdots b_{15}$ と取るものとする。この時，対応する圧縮過程の写像は

$$w_k = M_{2^{15}x}(w_{k-1}\oplus B_k) \;=\; (2^{15}x)(w_{k-1}\oplus B_k) - \lfloor (2^{15}x)(w_{k-1}\oplus B_k)\rfloor + 1$$

となるので（註：w_0 と x を [1,2) 内の数と同一視する；２．２節参照），$B = (00)(00)$ の圧縮は，

$$w_0 = \text{80130000} \xrightarrow{\text{XOR}} \text{801}(1\oplus 0)(3\oplus 0)\text{0000} = \text{80130000} \xrightarrow{\text{mul}} \text{4013016900000000}$$

$$\xrightarrow{\text{shift16}} \text{016900000000} \xrightarrow{\text{OR}} \text{816900000000} \xrightarrow{\text{cut}} \text{81690000} = w_1 \xrightarrow{\text{XOR}}$$

$$\text{81}(6\oplus 0)(9\oplus 0)\text{0000} = \mathbf{81690000} \xrightarrow{\text{mul}} \text{40be1acb00000000} \xrightarrow{\text{shift 16}}$$

$$\text{1acb00000000} \xrightarrow{\text{OR}} \text{9acb00000000} \xrightarrow{\text{cut}} \text{9acb0000} = w_2$$

と行われる。一方，$\hat{B} = (01)(3\text{f})$ を圧縮すると，

$$w_0 = \text{80130000} \xrightarrow{\text{XOR}} \text{801}(1\oplus 0)(3\oplus 1) = \text{80120000} \xrightarrow{\text{mul}} \text{4012815600000000}$$

$$\xrightarrow{\text{shift16}} \text{815600000000} \xrightarrow{\text{OR}} \text{815600000000} \xrightarrow{\text{cut}} \text{81560000} = \hat{w}_1 \xrightarrow{\text{XOR}}$$

$$\text{81}(5\oplus 3)(6\oplus \text{f})\text{0000} = \mathbf{81690000}\;(B \text{ と同じ}) \xrightarrow{\text{mul}} \cdots \xrightarrow{\text{cut}} \text{9acb0000} = \hat{w}_2$$

となり，B と同じ圧縮値を得る。これは，乗算後に LSB を含むあまりにも多くの０ビットが下位に続くためである。これを避けるためには，３．４節にあるように w_0 と x の LSB を１にしておけば良い；すなわち，w_0 ，x について 0x80013000 の代わりに 0x80013001 を使えばよい。この改良により，B_1 と $\hat{B}_1$ が異なっていれば，乗算，左シフト後に得られる w_1 と $\hat{w}_1$ において，LSB を含む後半の１６ビットは全てが０となることはなく，異なったものになる。例えば，$B = (00)B_2\cdots$ と $\hat{B} = (01)\hat{B}_2\cdots$ に対しては，それぞれ

$$w_0 = 80130001 \xrightarrow{\text{XOR}} 801(1\oplus 0)(3\oplus 0)0001 = 80130001 \xrightarrow{\text{mul}} 4013016a00260001$$
$$\xrightarrow{\text{shift16}} 016a00260001 \xrightarrow{\text{OR}} 816a00260001 \xrightarrow{\text{cut}} 816a\,\mathbf{0026} \quad = w_1$$
$$\xrightarrow{\text{XOR}} 81(6a\oplus B_2)\,\mathbf{0026} \quad \rightarrow \cdots$$

および,

$$w_0 = 80130001 \xrightarrow{\text{XOR}} 801(1\oplus 0)(3\oplus 1)0001 = 80120001 \xrightarrow{\text{mul}} 4012815700250001$$
$$\xrightarrow{\text{shift16}} 815700250001 \xrightarrow{\text{OR}} 815700250001 \xrightarrow{\text{cut}} 8157\,\mathbf{0025} = \hat{w}_1$$
$$\xrightarrow{\text{XOR}} 81(57\oplus \hat{B}_2)\,\mathbf{0025} \quad \rightarrow \cdots$$

と圧縮が行われ，w_1 と $\hat{w}_1$ の LSB を含む後半の１６ビットは異なっている。さらに, $(6a\oplus B_2)$ と $(57\oplus \hat{B}_2)$ の中にある $\oplus B_2$ と $\oplus \hat{B}_2$ は, w_1 と $\hat{w}_1$ の LSB を含む後半の１６ビットに触ることが出来ないので, $B=(00)(00)$ と $\hat{B}=(01)(3f)$ については勿論のこと, どのような $B=(00)B_2$ と $\hat{B}=(01)\hat{B}_2$ に対しても $w_1 \neq \hat{w}_1$ となる。このことは，４．２節の MB*n*hash で w_k の LSB を常に１にしている理由の１つになっている。

６．２　アルゴリズムの実装という観点から見た撹乱過程の安全性

５．２節では代数不等式 (5.31)を導き，５次以上の代数方程式を解く一般的な理論が存在しないということを撹乱過程のアルゴリズムの安全性の根拠とした。しかしながら, 代数方程式を解くもう一つの方法がある；すなわち，数値計算による方法である。その手法としては，数値計算で見つけた (5.31) の解を参考にして，同じハッシュ値 ζ と $\hat{\zeta}$ を生じる異なる３２ビットの y と $\hat{y}$ を見つけようと言うものである。しかし，この方法も以下の理由でうまくいかない。仮に，ある $\hat{\zeta}$ に対して，(5.31) の数値解 Y が見つかり，$y=Y$ に対して，(5.19), (5.21) により自動的に決まる $\breve{p}_1,\cdots,\breve{p}_{16}$，$\lfloor 2^{11}z \rfloor$ について，$P_k=\breve{p}_k$，$Z=\lfloor 2^{11}z \rfloor$ を満たしたとしよう（$\Rightarrow$ このとき，$\breve{p}_k=P_k$，$\lfloor 2^{11}z \rfloor = Z$ で (5.30) が満たされることになる)。$Y \equiv 1.Y_1Y_2\cdots Y_{31}\cdots$ であったとする。この Y から，３２ビットの数 Y^* で, 不等式 (5.30) を $y=Y^*$ として満たすものを見つけたいと考える。以下の議論の考え方は Y^* の取り方によらず同じであるので，$Y^* \equiv 1.Y_1Y_2\cdots Y_{31}$ としてみる。するとこの Y^* から (5.19) により $\breve{q}_1^*,\cdots,\breve{q}_{16}^*$ と $\lfloor 2^{11}z^* \rfloor$ が計算される。このとき,

$$\breve{p}_k = \breve{q}_k^*,\ k=1,\cdots,16 \quad , \quad \lfloor 2^{11}z \rfloor = \lfloor 2^{11}z^* \rfloor \qquad ((5.25)'(5.26)')\ (6.1)$$

を仮定してよい。（なぜなら (6.1) が成り立たなくなった時点で（すなわち，$\breve{p}_k \neq \breve{q}_k^*$ または $\lfloor 2^{11}z \rfloor \neq \lfloor 2^{11}z^* \rfloor$ が起こった時点で)，Y に関する不等式 (5.30) は，Y^* に関する不等式 (5.30) を近似する不等式として意味をなさなくなるからである。）すると５．２．３節の最初の部分で行った議論により $Y-Y^* < 2^{-91}$ である。これは，Y が３２ビット数 Y^* に丸められたとき，その

誤差は 2^{-91} 以下でなければならないことを示している。このことは，今の場合，Y のビット $Y_{32}Y_{33}\cdots Y_{90}$ がすべて０であることが要求される。しかしながら，現実的には，このようなことはほとんど不可能である。かくしてほとんどの場合，(5.30) の数値解は，３２ビットに丸められたとき，役に立たないことが分かる。

最後に，３２ビット数 u_{k-1}^{*} から $M_{2^3Y*}(u_{k-1}^{*})$，$k=1,\cdots,16,$ を計算して（→計算中の乗算で６４ビット数が現れる），３２ビット数 u_k^{*} を得る際に生ずる丸め誤差について考えてみる。$M_{2^3Y*}(u_{k-1}^{*})$ から u_k^{*} を得るときに生じた丸め誤差は，その後に続く写像 M_{2^3Y*} による 2^3Y^{*} 倍が $16-k$ 回適用されるので，k が小さいときには，その誤差は急速に拡大されていく。特に初期段階で発生した丸め誤差は，上述した Y の Y^{*} への切り捨てと同様の効果を引き起こす；すなわち, 期待される不等式 (5.30) を無意味にしてしまう。さらに, 丸め誤差の大きさは不確定であり，また，値を制御することも難しい。このようなことから，丸め誤差は，MB32hash への数値解析に基づく攻撃に対する耐性を増加させていることが分かる。

７．[1,2) 上のβ変換を利用した非再帰型 64 ビット乱数生成器の構成

最近のパソコンは６４ビットのＯＳを搭載するものが多い。この節では 64 ビット×64 ビット＝128 ビットの乗算を用いた本格的な非再帰型６４ビット乱数生成器 SSI64rand を作成する[Ya11]。現時点の普通のＣコンパイラには，64 ビット×64 ビット＝128 ビットの乗算を行う機能がないのでインラインアセンブラを使うことにする。アルゴリズムは第１章の SSI32Krand と同じであり，２系列の MB64rand を構成して，それぞれの計算値の差をとって１つの乱数値を作成する。

７．１　SSI64rand のアルゴリズム

M_β を [1,2) 上のβ変換　$M_\beta(t) = \beta t - \lfloor \beta t \rfloor + 1$　とする。$1.e$ で数 1.27181... を，$1.\pi$ で数 1.31415... を表すことにする。$1.e$ の２進表現を $1.\hat{r}_1\hat{r}_2\hat{r}_3\cdots\hat{r}_{63}\cdots$ とするとき，６３ビットの非負整数 $\nu_1\nu_2\cdots\nu_{63}$ に対し，

$$x = 1.(\hat{r}_1 \oplus \nu_1)(\hat{r}_2 \oplus \nu_2)\cdots(\hat{r}_{63} \oplus \nu_{63})\hat{r}_{64}\hat{r}_{65}\cdots$$

と定める。この x を $(1.e)\oplus(\nu_1\nu_2\cdots\nu_{63})$ と記すことにする。$w_0 \in [1,2)$ を x に依存しない数とし，

$$u \equiv M^{16}_{2^5 x}(w_0) = 1.\breve{b}_7\breve{b}_8\cdots\breve{b}_{127}\breve{b}_{128}\cdots$$

を計算する。同様に，$\nu_1\nu_2\cdots\nu_{63}$，w_0 と異なる $\tilde{\nu}_1\tilde{\nu}_2\cdots\tilde{\nu}_{63}$，$\tilde{w}_0$ について

$$y = (1.\pi)\oplus(\tilde{\nu}_1\tilde{\nu}_2\cdots\tilde{\nu}_{63}),\quad v \equiv M^{16}_{2^5 y}(\tilde{w}_0) = 1.\tilde{b}_7\tilde{b}_8\cdots\tilde{b}_{127}\tilde{b}_{128}\cdots$$

を計算する。そして６４ビット乱数値 ζ として

$$\hat{b}_0\,\hat{b}_7\hat{b}_8\cdots\hat{b}_{127}\hat{b}_{128} \;=\; 1\breve{b}_7\breve{b}_8\cdots\breve{b}_{127}\breve{b}_{128} - 1\tilde{b}_7\tilde{b}_8\cdots\tilde{b}_{127}\tilde{b}_{128}$$

の中から $\hat{b}_{32}\,\hat{b}_{33}\cdots\hat{b}_{95}$ を取り出す。多量の乱数 $\{\zeta_n\}_{n=0,1,\cdots}$ を得るには，$k-$番目の $\nu_1\nu_2\cdots\nu_{63}$，$\tilde{\nu}_1\tilde{\nu}_2\cdots\tilde{\nu}_{63}$ として

$$\nu_1\nu_2\cdots\nu_{63} \;=\; (\texttt{0x39f750241c2d5d33})\times k \bmod \texttt{0x7fffffffffffffe7},$$
$$\tilde{\nu}_1\tilde{\nu}_2\cdots\tilde{\nu}_{63} \;=\; (\texttt{0x32f50fee9b2a32bb})\times k \bmod \texttt{0x7fffffffffffff5b}$$

を使う。このとき，$M^{16}_{2^5 x}(w_0)$，$M^{16}_{2^5 y}(\tilde{w}_0)$ 中の x および y はそれぞれ

$$x_k = (1.e)\oplus(\nu_1\nu_2\cdots\nu_{63}) \quad \text{および} \quad y_k = (1.\pi)\oplus(\tilde{\nu}_1\tilde{\nu}_2\cdots\tilde{\nu}_{63})$$

となる。$\nu_1\nu_2\cdots\nu_{63}$，$\tilde{\nu}_1\tilde{\nu}_2\cdots\tilde{\nu}_{63}$ の定義に使われた４つの数はいずれも素数なので，SSI64rand の周期は $\{(x_k, y_k)\}_{k=0,1,2,\cdots}$ の周期に等しく，約 $2^{126} \approx 10^{38}$ である。w_0，$\tilde{w}_0$ は乱数の系列を変えるときに使われるが，既定値として $w_0 = 1.e$，$\tilde{w}_0 = 1.\pi$ を充てている。

７．２　SSI64rand のアルゴリズムの計算機への実装

SSI64rand のアルゴリズムの実装は，MB32rand アルゴリズムの実装を６４ビットに拡張して行う：

・[1,2) 内のすべての実数は，（乗算結果を除いて）６４ビット $1.b_1b_2b_3\cdots b_{63}$ で表される（６４ビットを超す部分は切り捨てる），
・２数　$x=1.x_1x_2\cdots x_{63}$ と　$t=1.t_1t_2\cdots t_{63}$ の乗算結果は１２８ビット $\breve{b}_0\breve{b}_1.\breve{b}_2\breve{b}_3\cdots\breve{b}_{127}$ で表され（$\breve{b}_0\breve{b}_1$ は，乗算結果が [1,2) 内にあるときは $\breve{b}_0\breve{b}_1=01$ に，[2,4) 内にあるときは $\breve{b}_0=1$ となる），その後，左シフト等を経て６４ビットに切りつめられる。

上述の $1.b_1b_2b_3\cdots b_{63}$ は，６４ビットの整数 $1b_1b_2b_3\cdots b_{63}$ と同一視され，乗算結果の $\breve{b}_0\breve{b}_1.\breve{b}_2\breve{b}_3\cdots\breve{b}_{127}$ は１２８ビット整数 $\breve{b}_0\breve{b}_1\breve{b}_2\breve{b}_3\cdots\breve{b}_{127}$ と同一視される。この同一視では，$1.e$，$1.\pi$ はそれぞれ

```
0xa2cb4411ba257552,   0xa8365eed39e1c070
```

となる。関数 $M_{2^5x}(t)$ は，

$$\begin{aligned}M_{2^5x}(t) &= (2^5x)t \mod [1,2) = (2^5)xt \mod [1,2)\\ &= 2^5\times\breve{b}_0\breve{b}_1.\breve{b}_2\breve{b}_3\breve{b}_4\breve{b}_5\breve{b}_6\breve{b}_7\breve{b}_8\cdots \mod [1,2)\\ &= \breve{b}_0\breve{b}_1\breve{b}_2\breve{b}_3\breve{b}_4\breve{b}_5\breve{b}_6.\breve{b}_7\breve{b}_8\cdots \mod [1,2)\\ &= 1.\breve{b}_7\breve{b}_8\cdots\end{aligned}$$

と計算されるので，６４ビット整数系では

$$\begin{matrix}1x_1x_2\cdots x_{63}\\ 1t_1t_2\cdots t_{63}\end{matrix}\xrightarrow{\text{mul}}\breve{b}_0\breve{b}_1\breve{b}_2\cdots\breve{b}_5\breve{b}_6\breve{b}_7\cdots\cdots\breve{b}_{127}\xrightarrow{\text{shift 6}}$$

$$\breve{b}_6\breve{b}_7\cdots\breve{b}_{69}\cdots\breve{b}_{127}\xrightarrow{\text{OR}}1\breve{b}_7\cdots\breve{b}_{69}\cdots\breve{b}_{127}\xrightarrow{\text{cut}}1\breve{b}_7\cdots\breve{b}_{69}$$

と計算すればよい。n 番目の SSI64rand 乱数は，$M^{16}_{2^5x_n}(1.e)$, $M^{16}_{2^5y_n}(1.\pi)$ がそれぞれ $1\breve{b}_7\breve{b}_8\cdots\breve{b}_{127}\breve{b}_{128}$，$1\tilde{b}_7\tilde{b}_8\cdots\tilde{b}_{127}\tilde{b}_{128}$ に同一視されるとき，この２数の２進差をとった

$$\hat{b}_0\,\hat{b}_7\hat{b}_8\cdots\hat{b}_{127}\hat{b}_{128} = 1\breve{b}_7\breve{b}_8\cdots\breve{b}_{127}\breve{b}_{128} - 1\tilde{b}_7\tilde{b}_8\cdots\tilde{b}_{127}\tilde{b}_{128}$$

中の $\hat{b}_{32}\,\hat{b}_{33}\cdots\hat{b}_{95}$ である。例えば，１番目，２番目の SSI64rand 乱数は `0x8eaafb19f73587f8`，`0x4bb2533b46fb5cf1` になる。少し長いが，以下に SSI64rand のプログラムコードを載せてお

く。（使用コンパイラは，インテルの icl（64 ビット） および gcc（64 ビット）。）

```
//*******************************
#define MB_W64_X64_16(out, in1, in2) \
     __asm__("mov %3, %%rcx;    \
              mov %2, %%rax;    \
              mov  $0x8000000000000000, %%r10; \
              or  %%r10, %%rax; \
              mul %%rcx;        \
              shl $6, %%rdx;\
              shr $58, %%rax; \
              or  %%rdx, %%rax; \

      //上の or から or まで(5行)を,14回繰返す

              or  %%r10, %%rax; \
              mul %%rcx;     \
              mov %%rdx, %0; \
              mov %%rax, %1; \
              "              \
             : "=r"(out.hi64), "=r"(out.lo64) \
             : "r"(in1), "r"(in2)     \
             : "%rax", "%rcx", "%rdx", "%r10" \
             );
//*******************************
#define SUB_128_128_SHL32(out, in1, in2) \
     __asm__("mov %2, %%rax; \
              mov %3, %%rdx; \
              mov %5, %%rcx; \
              sub %%rcx, %%rdx; \
              mov %4, %%rcx; \
              sbb %%rcx, %%rax; \
              mov %%rdx, %%rcx; \
              shl $32, %%rax; \
              shr $32, %%rcx; \
              or  %%rcx, %%rax; \
              shl $32, %%rdx; \
              mov %%rax, %0; \
              mov %%rdx, %1; \
              "              \
             : "=r"(out.hi64), "=r"(out.lo64) \
    :"r"(in1.hi64),"r"(in1.lo64),"r"(in2.hi64),"r"(in2.lo64) \
             : "%rax", "%rcx", "%rdx" \
             );
//*******************************
typedef struct
    {
    unsigned long long int lo64;
    unsigned long long int hi64;
    } my_i128;
//*******************************//
unsigned long long int genSSI64rand(unsigned long long int x,
                                    unsigned long long int t,
                                    unsigned long long int y,
                                    unsigned long long int s)
    {
    my_i128 u, v, umv;

    MB_W64_X64_16(u, t, x);
    MB_W64_X64_16(v, s, y);
    SUB_128_128_SHL32(umv, u, v);

    return(umv.hi64);
    }
//*******************************
```

７．３　SSI64rand の乱数性の統計的検定

SSI64rand が生成する乱数の乱数性を，NIST の検定および TestU01 の BigCrush を使って行う。NIST の検定の方法は２．３節の MB32rand と同様であるので以下に結果だけを記す：

```
 [SSI64rand]
P-VALUE AvMax  =  0.742517 at NonOverlappingTemplate [i=131]
P-VALUE AvMin  =  0.185703 at NonOverlappingTemplate [i=45]
PROPORTION AvMax = 0.992719 at RandomExcursionsVariant [i=179]
PROPORTION AvMin = 0.986145 at RandomExcursions [i=159].
```

TestU01 は３２ビットのコードで作成されているため，６４ビットの SSI64rand は使えない。そのため SSI64rand に相当するコードを３２ビットで作成して検定を行った。３２ビット版の SSI64rand では，６４ビットの乗算を３２ビットの乗算を使ってエミュレートするため，６４ビット版より７倍ほど遅くなる（インテルの icl の場合）。BigCrush による検定結果は

```
"All tests were passed"
```

である。

８．[1,2) 上のβ変換のエルゴード的性質

８．１　[1,2) 上のβ変換の性質

この節では，$[1,2)$ 上の β 変換 M_β の性質について手短にまとめておく（M_β の定義は (2.1) を参照）。元々 β 変換（$\beta>1$）は，$[0,1)$ 上の変換として

$$T_\beta(s) = \beta s \mod 1 = \beta s - \lfloor \beta s \rfloor$$

と定義されたものであり，これは，linear mod one 変換

$$T_{\beta,\alpha}(s) = \beta s + \alpha \mod 1 = \beta s + \alpha - \lfloor \beta s + \alpha \rfloor \quad (0 \le \alpha < 1)$$

として拡張された。変換 T_β，$T_{\beta,\alpha}$ の性質については，長い研究の歴史があり多くの結果が得られている。一方で，我々の $[1,2)$ 上の β 変換 M_β は $T_{\beta,\alpha}$ と

$$M_\beta(t) = T_{\beta,\hat{\beta}}(t-1)+1, \quad t \in [1,2) \tag{8.1}$$

のような関係を持っている。ただし，$\hat{\beta}$ は β の小数部分である（すなわち，$\hat{\beta}$ は $\beta = \lfloor \beta \rfloor + \hat{\beta}$ により定義される $[0,1)$ 内の数である）。等式 (8.1) は以下のように確かめられる：

$$\begin{aligned} T_{\beta,\hat{\beta}}(t-1) &= \beta(t-1)+\hat{\beta}-\lfloor \beta(t-1)+\hat{\beta} \rfloor \\ &= \beta t-\lfloor \beta \rfloor-\lfloor \beta t-\lfloor \beta \rfloor \rfloor = \beta t-\lfloor \beta t \rfloor \\ &= M_\beta(t)-1. \end{aligned}$$

関係式 (8.1) より，M_β のグラフは $T_{\beta,\hat{\beta}}$ のグラフを右方向に1，上方向に1ずらして得られることが分かる。よって $T_{\beta,\alpha}$ に関する性質は M_β についても成り立つ。以下に $T_{\beta,\alpha}$ の性質を少しまとめておく。

$X=[0,1)$ とし，(X,B,λ) を X 上の確率空間とする；ただし，B はボレル σ 加法族，λ はルベーグ測度である。

Parry [Pa] は，

$$h_{\beta,\alpha}(x) = \sum_{x<T^n_{\beta,\alpha}(1),\, n\geq 0} \frac{1}{\beta^n} - \sum_{x<T^n_{\beta,\alpha}(0),\, n\geq 1} \frac{1}{\beta^n} \tag{8.2}$$

とすると，

$$\nu_{\beta,\alpha}(E) = \int_E h_{\beta,\alpha}(x)d\lambda(x)$$

は，$T_{\beta,\alpha}$ 不変(invariant)な有限符号付測度(finite signed measure) であることを示した。（測度 $\nu_{\beta,\alpha}$ は，すべての $E\in\mathsf{B}$ に対して $\nu_{\beta,\alpha}\left(\left(T_{\beta,\alpha}\right)^{-1}(E)\right)=\nu_{\beta,\alpha}(E)$ が成り立つとき，$T_{\beta,\alpha}$ 不変と言われる。）さらに，$\nu_{\beta,\alpha}$ が強エルゴード的(strongly ergodic)ならば，ほとんど至るところで $h_{\beta,\alpha}(x)\geq 0$ であり，$\nu_{\beta,\alpha}$ は，$T_{\beta,\alpha}$ 不変な有限正測度であることを示した。（写像 $T_{\beta,\alpha}$ は，『$\left(T_{\beta,\alpha}\right)^{-1}(E)\subset E$ ならば，$\lambda(E)=0$ または $\lambda(E)=1$』を満たすとき，強エルゴード的と言われる。）（註：[Ha] によれば，一般に $h_{\beta,\alpha}(x)\geq 0$）

Wilkinson [Wi] は，$\beta>2$ ならば $T_{\beta,\alpha}$ は強エルゴード的であることを示した。

これらの結果を結合させると，$T_{\beta,\alpha}$ ，$\beta>2$，について以下が成り立つことが分かる：

(i) a.e. $x\in X$ に対して $h_{\beta,\alpha}(x)\geq 0$，　(a.e. = almost everywhere, ほとんど至るところ)

(ii) 確率測度 $\mu_{\beta,\alpha}(\cdot)=\nu_{\beta,\alpha}(\cdot)/\nu_{\beta,\alpha}(X)$ は $T_{\beta,\alpha}$ 不変である，

(iii) $f\in L^1(X,\mathsf{B},\mu_{\beta,\alpha})$ に対し，a.e. $x\in X$ について

$$\lim_{N\to\infty}\frac{1}{N}\sum_{n=0}^{N-1} f(T^n_{\beta,\alpha}(x)) = \int_X f(y)d\mu_{\beta,\alpha}(y) \quad \text{([Po]).} \tag{8.3}$$

さらに (8.2) から，簡単な計算により

$$\frac{\beta-2}{\beta-1} \leq h_{\beta,\alpha}(x) \leq \frac{\beta}{\beta-1} \quad \text{すなわち} \quad 1-\frac{1}{\beta-1} \leq h_{\beta,\alpha}(x) \leq 1+\frac{1}{\beta-1} \tag{8.4}$$

であることも分かる。

８．２　写像 M_β の性質

ここで，$[1,2)$ 上の変換 M_β の話に戻る。M_β に対応する $T_{\beta,\alpha}$ は $T_{\beta,\hat\beta}$ であるので，以下 $T_{\beta,\hat\beta}$ とその不変測度 $h_{\beta,\hat\beta}$ について議論を進める。$Z=\left(M_{2^{s-1}\tilde W}\right)^{16}(\tilde W)$ 中にある $\beta_{\tilde W}\equiv 2^{s-1}\tilde W$ は2より大きいので，上述の (i), (ii), (iii) が $T_{\beta_{\tilde W},\hat\beta_{\tilde W}}$ に対して成り立つ。さらに (8.4) と $2^{s-1}\le\beta_{\tilde W}\le 2^s$ であることにより，

$$1-\frac{1}{2^{S-1}-1}\ \le\ h_{\beta_{\tilde W},\hat\beta_{\tilde W}}(x)\ \le\ 1+\frac{1}{2^{S-1}-1} \tag{8.5}$$

を得る。MB1024hash のとき $S=96$ なので，(8.5) は

$$1-\frac{1}{2^{95}-1}\ \le\ h_{\beta_{\tilde W},\hat\beta_{\tilde W}}(x)\ \le\ 1+\frac{1}{2^{95}-1}\qquad (2^{95}\approx 3.96\times 10^{28}) \tag{8.6}$$

となり，$h_{\beta_{\tilde W},\hat\beta_{\tilde W}}(x)$ は $\tilde W$，x によらずほぼ1であることが分かる。次に，$\tilde W$ の値が w であったとする（$\tilde W$ は入力データ B に依存して値が変化することに注意せよ）。このとき $\beta_w=2^{s-1}w$ である。すると (X,B,λ) 上の有界可測関数 f に対して，(8.3) より，ほとんど全ての x に対して

$$\lim_{N\to\infty}\frac{1}{N}\sum_{n=0}^{N-1} f(T^n_{\beta_w,\hat\beta_w}(x))=\int_X f(y)d\mu_{\beta_w,\hat\beta_w}(y)$$

が成り立つ。これは，N が大きくなるにつれ，$\{T^n_{\beta_w,\hat\beta_w}(x)\}_{n=0,1,\cdots,N-1}$ の分布が，x に依存せず，確率密度関数が $\hat h_w(\cdot)\equiv\dfrac{1}{\nu_{\beta_w,\hat\beta_w}(X)}h_{\beta_w,\hat\beta_w}(\cdot)$ であるような確率測度 $\mu_{\beta_w,\hat\beta_w}$ に近づくことを表している。$\tilde W$ の値 w は B に依存するので，B が色々変化するとき，$Z=\left(M_{2^{s-1}\tilde W}\right)^{16}(\tilde W)$ の分布は，多くの $\hat h_w$，$w\in[1,2)$ ，を用いて記述されるはずである。$\tilde W$ が $[1,2)$ 内を一様に分布するとすれば，Z の分布 $H(y)$ は $\hat h_w(y-1)$ の一様な混ぜ合わせ，すなわち，
$H(y)=\int_1^2\hat h_w(y-1)\,dw$ になると考えられるが（詳細は [Ya8] の5節を参照），(8.5) より我々の使う S の値においては $\hat h_w(\cdot)$ は w に関する依存性が非常に少ない関数で，ほとんど1に近い値であるので，$H(y)$ はほとんど1に近い値であることが分かる。

8．3　$h_{\beta,\alpha}$ が $T_{\beta,\alpha}$ の不変測度を定める関数であることの証明

$h_{\beta,\alpha}$ が $T_{\beta,\alpha}$ 変換の不変測度を定める関数であることの証明は [Pa] にあるが，西村は [Ni] で直截的な証明を付けているので紹介しておく。簡単のため $h_{\beta,\alpha}$，$T_{\beta,\alpha}$ を，それぞれ h，T と書くことにすると，(8.2) は

$$\begin{aligned} h(y) &= \sum_{y<T^n(1),\,n\ge 0}\frac{1}{\beta^n}-\sum_{y<T^n(0),\,n\ge 1}\frac{1}{\beta^n}\\ &=\sum_{n=0}^{\infty}\frac{1}{\beta^n}1_{[0,T^n(1))}(y)-\sum_{n=1}^{\infty}\frac{1}{\beta^n}1_{[0,T^n(0))}(y) \end{aligned}\qquad (8.2)'\quad (8.7)$$

と表される。ここで $1_{[a,b)}(y) = \begin{cases} 1 & if \ \ y \in [a,b) \\ 0 & if \ \ y \notin [a,b) \end{cases}$ である。 $T(s) = \beta s + \alpha - \lfloor \beta s + \alpha \rfloor$ であるので，

T に対する Perron-Frobenius 作用素を L とすると，L は $f \in L^1$ に対して

$$\begin{aligned}(Lf)(y) = \frac{1}{\beta} f\left(\frac{y-\alpha}{\beta}\right) 1_{[T(0),1)}(y) + \frac{1}{\beta} f\left(\frac{y-\alpha}{\beta} + \frac{1}{\beta}\right) + \cdots \\ + \frac{1}{\beta} f\left(\frac{y-\alpha}{\beta} + \frac{\lfloor \beta+\alpha \rfloor - 1}{\beta}\right) + \frac{1}{\beta} f\left(\frac{y-\alpha}{\beta} + \frac{\lfloor \beta+\alpha \rfloor}{\beta}\right) 1_{[0,T(1))}(y)\end{aligned} \tag{8.8}$$

となる。h が T の不変測度を定める関数であることを示すには，h が L の不動点である，すなわち

$$Lh = h \quad \text{あるいは} \quad \beta Lh = \beta h \tag{8.9}$$

を満たすことを示せば良い（１０．２節参照)。以下では，$\beta Lh = \beta h$ を証明する。(8.8) より

$$\begin{aligned}(\beta Lh)(y) &= h\left(\frac{y-\alpha}{\beta}\right) 1_{[T(0),1)}(y) + h\left(\frac{y-\alpha}{\beta} + \frac{1}{\beta}\right) + \cdots \\ &\quad + h\left(\frac{y-\alpha}{\beta} + \frac{\lfloor \beta+\alpha \rfloor - 1}{\beta}\right) + h\left(\frac{y-\alpha}{\beta} + \frac{\lfloor \beta+\alpha \rfloor}{\beta}\right) 1_{[0,T(1))}(y) \\ &= \left(\sum_{n=0}^{\infty} \frac{1}{\beta^n} 1_{[0,T^n(1))}\left(\frac{y-\alpha}{\beta}\right) - \sum_{n=1}^{\infty} \frac{1}{\beta^n} 1_{[0,T^n(0))}\left(\frac{y-\alpha}{\beta}\right)\right) 1_{[T(0),1)}(y) \\ &\quad + \left(\sum_{n=0}^{\infty} \frac{1}{\beta^n} 1_{[0,T^n(1))}\left(\frac{y-\alpha+1}{\beta}\right) - \sum_{n=1}^{\infty} \frac{1}{\beta^n} 1_{[0,T^n(0))}\left(\frac{y-\alpha+1}{\beta}\right)\right) \\ &\quad + \cdots \\ &\quad + \left(\sum_{n=0}^{\infty} \frac{1}{\beta^n} 1_{[0,T^n(1))}\left(\frac{y-\alpha+\lfloor \beta+\alpha \rfloor - 1}{\beta}\right) - \sum_{n=1}^{\infty} \frac{1}{\beta^n} 1_{[0,T^n(0))}\left(\frac{y-\alpha+\lfloor \beta+\alpha \rfloor - 1}{\beta}\right)\right) \\ &\quad + \left(\sum_{n=0}^{\infty} \frac{1}{\beta^n} 1_{[0,T^n(1))}\left(\frac{y-\alpha+\lfloor \beta+\alpha \rfloor}{\beta}\right) - \sum_{n=1}^{\infty} \frac{1}{\beta^n} 1_{[0,T^n(0))}\left(\frac{y-\alpha+\lfloor \beta+\alpha \rfloor}{\beta}\right)\right) 1_{[0,T(1))}(y) \\ &= \left(\sum_{n=0}^{\infty} \frac{1}{\beta^n} 1_{[\alpha,\beta T^n(1)+\alpha)}(y) - \sum_{n=1}^{\infty} \frac{1}{\beta^n} 1_{[\alpha,\beta T^n(0)+\alpha))}(y)\right) 1_{[T(0),1)}(y) \\ &\quad + \left(\sum_{n=0}^{\infty} \frac{1}{\beta^n} 1_{[\alpha,\beta T^n(1)+\alpha)}(y+1) - \sum_{n=1}^{\infty} \frac{1}{\beta^n} 1_{[0,\beta T^n(0)+\alpha)}(y+1)\right) \\ &\quad + \cdots \\ &\quad + \left(\sum_{n=0}^{\infty} \frac{1}{\beta^n} 1_{[\alpha,\beta T^n(1)+\alpha)}(y+\lfloor \beta+\alpha \rfloor - 1) - \sum_{n=1}^{\infty} \frac{1}{\beta^n} 1_{[0,\beta T^n(0)+\alpha)}(y+\lfloor \beta+\alpha \rfloor - 1)\right) \\ &\quad + \left(\sum_{n=0}^{\infty} \frac{1}{\beta^n} 1_{[\alpha,\beta T^n(1)+\alpha)}(y+\lfloor \beta+\alpha \rfloor) - \sum_{n=1}^{\infty} \frac{1}{\beta^n} 1_{[\alpha,\beta T^n(0)+\alpha)}(y+\lfloor \beta+\alpha \rfloor)\right) 1_{[0,T(1))}(y)\end{aligned} \tag{8.10}$$

ここで，(8.10) において

$$1_{[\alpha,\beta T^n(1)+\alpha)}(y)=1 \quad \Rightarrow \quad 1_{[T(0),1)}(y)=1 \tag{8.11a}$$

$$1_{[\alpha,\beta T^n(0)+\alpha)}(y)=1 \quad \Rightarrow \quad 1_{[T(0),1)}(y)=1 \tag{8.11b}$$

$$1_{[\alpha,\beta T^n(1)+\alpha)}(y+\lfloor\beta+\alpha\rfloor)=1 \quad \Rightarrow \quad 1_{[0,T(1))}(y)=1 \tag{8.11c}$$

$$1_{[\alpha,\beta T^n(0)+\alpha)}(y+\lfloor\beta+\alpha\rfloor)=1 \quad \Rightarrow \quad 1_{[0,T(1))}(y)=1 \tag{8.11d}$$

が成り立つ。

実際 (8.11c) は,

$$\begin{aligned} 1_{[\alpha,\beta T^n(1)+\alpha)}(y+\lfloor\beta+\alpha\rfloor)=1 \quad &\Rightarrow \quad \lfloor\beta+\alpha\rfloor \le y+\lfloor\beta+\alpha\rfloor < \beta T^n(1)+\alpha \\ &\Rightarrow \quad \lfloor\beta+\alpha\rfloor \le y+\lfloor\beta+\alpha\rfloor < \beta+\alpha \equiv \lfloor\beta+\alpha\rfloor+T(1) \\ &\Rightarrow \quad 0 \le y < T(1) \quad \Rightarrow \quad 1_{[0,T(1))}(y)=1 \end{aligned} \tag{8.12c}$$

と示され, (8.11d) は (8.12c) で $T^n(1)$ を $T^n(0)$ で置き換えれば示される。

また (8.11a) は, $y<1$ であるので,

$$1_{[\alpha,\beta T^n(1)+\alpha)}(y)=1 \quad \Rightarrow \quad T(0)\equiv\alpha \le y < 1 \quad \Rightarrow \quad 1_{[T(0),1)}(y)=1 \tag{8.12a}$$

と示され, (8.12a) で $T^n(1)$ を $T^n(0)$ で置き換えれば (8.11b) が得られる。

よって, (8.10) に (8.11a)-(8.11d) を適用すれば,

$$\begin{aligned} (\beta Lh)(y) \;=\; & \left(\sum_{n=0}^{\infty}\frac{1}{\beta^n}1_{[\alpha,\beta T^n(1)+\alpha)}(y)-\sum_{n=1}^{\infty}\frac{1}{\beta^n}1_{[\alpha,\beta T^n(0)+\alpha))}(y)\right) \\ &+\left(\sum_{n=0}^{\infty}\frac{1}{\beta^n}1_{[\alpha,\beta T^n(1)+\alpha)}(y+1)-\sum_{n=1}^{\infty}\frac{1}{\beta^n}1_{[0,\beta T^n(0)+\alpha)}(y+1)\right) \\ &+\cdots \\ &+\left(\sum_{n=0}^{\infty}\frac{1}{\beta^n}1_{[\alpha,\beta T^n(1)+\alpha)}(y+\lfloor\beta+\alpha\rfloor-1)-\sum_{n=1}^{\infty}\frac{1}{\beta^n}1_{[0,\beta T^n(0)+\alpha)}(y+\lfloor\beta+\alpha\rfloor-1)\right) \\ &+\left(\sum_{n=0}^{\infty}\frac{1}{\beta^n}1_{[\alpha,\beta T^n(1)+\alpha)}(y+\lfloor\beta+\alpha\rfloor)-\sum_{n=1}^{\infty}\frac{1}{\beta^n}1_{[\alpha,\beta T^n(0)+\alpha)}(y+\lfloor\beta+\alpha\rfloor)\right) \\ =\; & \left\{\sum_{k=0}^{\lfloor\beta+\alpha\rfloor}1_{[\alpha,\beta+\alpha)}(y+k)\right\}+\left\{\frac{1}{\beta}\left(\sum_{k=0}^{\lfloor\beta+\alpha\rfloor}1_{[\alpha,\beta T(1)+\alpha)}(y+k)\right)-\frac{1}{\beta}\left(\sum_{k=0}^{\lfloor\beta+\alpha\rfloor}1_{[\alpha,\beta T(0)+\alpha)}(y+k)\right)\right\} \\ &+\left\{\frac{1}{\beta^2}\left(\sum_{k=0}^{\lfloor\beta+\alpha\rfloor}1_{[\alpha,\beta T^2(1)+\alpha)}(y+k)\right)-\frac{1}{\beta^2}\left(\sum_{k=0}^{\lfloor\beta+\alpha\rfloor}1_{[\alpha,\beta T^2(0)+\alpha)}(y+k)\right)\right\} \\ &+\cdots \\ &+\left\{\frac{1}{\beta^n}\left(\sum_{k=0}^{\lfloor\beta+\alpha\rfloor}1_{[\alpha,\beta T^n(1)+\alpha)}(y+k)\right)-\frac{1}{\beta^n}\left(\sum_{k=0}^{\lfloor\beta+\alpha\rfloor}1_{[\alpha,\beta T^n(0)+\alpha)}(y+k)\right)\right\} \\ &+\cdots \end{aligned} \tag{8.13}$$

となる。ここで (8.13) 中の $\sum$ の部分は,

$$\sum_{k=0}^{\lfloor\beta+\alpha\rfloor} 1_{[\alpha,\beta T^n(1)+\alpha)}(y+k) \;=\; \#\left\{ k\in Z \mid \alpha \le y+k < \beta T^n(1)+\alpha,\quad k \le \lfloor\beta+\alpha\rfloor \right\}$$

$$= \#\left\{ k\in Z \mid \alpha \le y+k < \beta T^n(1)+\alpha \right\}$$

$$(\because k \le \lfloor y\rfloor + k = \lfloor y+k\rfloor \le \lfloor \beta T^n(1)+\alpha\rfloor \le \lfloor\beta+\alpha\rfloor)$$

$$= \#\left\{ k\in Z \mid \alpha - y \le k < \beta T^n(1)+\alpha - y \right\}$$

$$= \#\left\{ k\in Z \mid \alpha - y \le k < \lfloor \beta T^n(1)+\alpha\rfloor + T^{n+1}(1) - y \right\}$$

$$(\because \beta T^n(1)+\alpha - \lfloor \beta T^n(1)+\alpha\rfloor = T^{n+1}(1))$$

$$= \#\left\{ k\in Z \mid 1_{[0,\alpha)}(y) \le k \le \lfloor \beta T^n(1)+\alpha\rfloor - 1_{[T^{n+1}(1),1)}(y) \right\}$$

$$(\because -1 < \alpha - y < 1,\quad -1 < T^{n+1}(1) - y < 1)$$

$$= \lfloor \beta T^n(1)+\alpha\rfloor - 1_{[T^{n+1}(1),1)}(y) - 1_{[0,\alpha)}(y) + 1$$

$$= \lfloor \beta T^n(1)+\alpha\rfloor + 1_{[0,T^{n+1}(1))}(y) - 1_{[0,\alpha)}(y)$$

および，同様に

$$\sum_{k=0}^{\lfloor\beta+\alpha\rfloor} 1_{[\alpha,\beta T^n(0)+\alpha)}(y+k) \;=\; \lfloor \beta T^n(0)+\alpha\rfloor + 1_{[0,T^{n+1}(0))}(y) - 1_{[0,\alpha)}(y)$$

と変形される。これを使うと (8.13) は，

$$(\beta Lh)(y) \;=\; \left\{ \lfloor\beta+\alpha\rfloor + 1_{[0,T^1(1))}(y) - 1_{[0,\alpha)}(y) \right\}$$

$$+ \sum_{n=1}^{\infty} \frac{1}{\beta^n} \left\{ \left(\lfloor \beta T^n(1)+\alpha\rfloor + 1_{[0,T^{n+1}(1))}(y) - 1_{[0,\alpha)}(y) \right) - \left(\lfloor \beta T^n(0)+\alpha\rfloor + 1_{[0,T^{n+1}(0))}(y) - 1_{[0,\alpha)}(y) \right) \right\}$$

$$= \left\{ \lfloor\beta+\alpha\rfloor + 1_{[0,T^1(1))}(y) - 1_{[0,\alpha)}(y) \right\}$$

$$+ \sum_{n=1}^{\infty} \frac{1}{\beta^n} \left(\lfloor \beta T^n(1)+\alpha\rfloor - \lfloor \beta T^n(0)+\alpha\rfloor \right) + \sum_{n=1}^{\infty} \frac{1}{\beta^n} \left(1_{[0,T^{n+1}(1))}(y) - 1_{[0,T^{n+1}(0))}(y) \right) \quad (8.14)$$

となる。上式において，

$$\sum_{n=1}^{\infty} \frac{1}{\beta^n} \left(\lfloor \beta T^n(1)+\alpha\rfloor - \lfloor \beta T^n(0)+\alpha\rfloor \right)$$

$$= \sum_{n=1}^{\infty} \frac{1}{\beta^n} \left((\beta T^n(1)+\alpha - T^{n+1}(1)) - (\beta T^n(0)+\alpha - T^{n+1}(0)) \right) \quad (\because \beta T^n(1)+\alpha - \lfloor \beta T^n(1)+\alpha\rfloor = T^{n+1}(1))$$

$$= \left(\sum_{n=1}^{\infty} \frac{1}{\beta^{n-1}} T^n(1) - \sum_{n=1}^{\infty} \frac{1}{\beta^n} T^{n+1}(1) \right) - \left(\sum_{n=1}^{\infty} \frac{1}{\beta^{n-1}} T^n(0) - \sum_{n=1}^{\infty} \frac{1}{\beta^n} T^{n+1}(0) \right) \quad \text{（各級数は絶対収束）}$$

$$= T(1) - T(0) \;=\; (\beta+\alpha) - \lfloor\beta+\alpha\rfloor - \alpha \;=\; \beta - \lfloor\beta+\alpha\rfloor$$

であるので，(8.14) は

$$(\beta Lh)(y) \;=\; \beta + 1_{[0,T^1(1))}(y) - 1_{[0,\alpha)}(y) \;+\; \beta \sum_{n=2}^{\infty} \frac{1}{\beta^n} \left(1_{[0,T^n(1))}(y) - 1_{[0,T^n(0))}(y) \right)$$

$$= \beta \sum_{n=0}^{\infty} \frac{1}{\beta^n} 1_{[0,T^n(1))}(y) - \beta \sum_{n=0}^{\infty} \frac{1}{\beta^n} 1_{[0,T^n(0))}(y)$$

$= (\beta h)(y)$　　（(8.7) より)

となり，(8.9) が示された。

（註：Gora は [Go] で piecewise linear map の不変測度を定める密度関数を決めている）

参考文献

[Bo] Boyarsky, A., Gora, P.: Laws of Chaos. Birkhauser (1997)

[De] Devaney, R. L.: An introduction to chaotic dynamical systems, Second edition. Westview Press, 2003.

[Gi] Gifford, A. D.: http://www.aarongifford.com/

[Go] Gora, P.: Invariant densities for piecewise linear maps of the unit interval. Ergo. Th. & Dynam. Sys., 29, 1549-1583 (2009)

[Ha] Halfin, S.: Explicit construction of invariant measures for a class of continuous state Markov processes. Ann. Prob., 3, 859-864 (1975)

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

[LE] L'Ecuyer, P., Simard, R.: TestU01: A C Library for empirical testing of random number generators. ACM Transactions on Mathematical Software, 33, 4, Article 22 (2007) (http://www.iro.umontreal.ca/~simardr/testu01/tu01.html)

[LY] Lasota, A., Yorke, J. A.: On the existence of invariant measures for piecewise momotone transformations. Trans. Amer. Math. Soc., 186, 481-488 (1973)

[Ma] 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) (http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/mt19937ar)

[Ni] 西村美穂 ： [1,2) 上のベータ変換により生成される非再帰型擬似乱数の分布の研究. 修士論文 (2014)

[Pa] Parry, W.: Representations for real numbers. Acta Math. Acad. Sci. Hungar., 15, 95-105 (1964)

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

[Re] R'enyi, A.: Representations for real numbers and their ergodic properties. Acta Math. Acad. Sci. Hungar., 8, 477-493 (1957)

[Ro] Robinson, C.: Dynamical Systems. CRC Pr (1998)

[Ya1] Yaguchi, H.: Randomness of Horner's rule and a new method of generating random number. Monte Carlo Methods and Appl ., 6, 61-76 (2000)

[Ya2] 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)

[Ya3] Yaguchi, H.: Construction of a long-period nonalgebraic and nonrecursive pseudorandom

number generator. Monte Carlo Methods and Appl., 8, 203-213 (2002)

[Ya4] Yaguchi, H.: A new nonrecursive pseudorandom number generator based on chaos mappings. 7th International Conference on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing (MCQMC 2006).

[Ya5] 谷口礼偉、上田澄江、高嶋恵三: 新しいハッシュ関数 SSI160 の構築. ISM Research Memorandum No. 986 (2006)

[Ya6] 谷口礼偉: 非代数的な手法による擬似乱数生成法の研究. 平成 17～18 年度科学研究費補助金(基盤研究(C))研究成果報告書 (2007)

[Ya7] Yaguchi, H., Kubo, I.: A new nonrecursive pseudorandom number generator based on chaos mappings. Monte Carlo Methods Appl., 14, 87-102 (2008)

[Ya8] Yaguchi, H., Ueda, S.: Construction and randomness of new hash functions derived from chaos mappings. Interdisciplinary Information Sciences, 18, 1-11 (2012) DOI 10.4036/iis.2012.1

[Ya9] Yaguchi, H.: http://math1.edu.mie-u.ac.jp/yaguchi/

[Ya10] Yaguchi, H.: Construction and security of nonalgebraic hash functions based on β-transformations on [1,2).

[Ya11] Yaguchi, H.: Construction of a nonrecursive 64-bit pseudorandom number generator based on beta transformations on [1,2). Bulletin of the Faculty of Education, Mie University (2014)

[Ym] 山口知也：乗算と仮数部のシフトから導かれる[1,2)上の変換の不変測度の研究. 修士論文 (2014)

[Wi] Wilkinson, K. M.: Ergodic properties of certain linear mod one transformations. Advances in Math., 14, 64-72 (1974)

[Www1] http://csrc.nist.gov/groups/ST/toolkit/rng/index.html

[Www2] http://www.ism.ac.jp/ismlib/jpn/ismlib/soft.html#ssi32

Appendix

Construction and security of nonalgebraic hash functions based on β -Transformations on [1,2)

Construction and Security of Nonalgebraic Hash Functions Based on β-Transformations on [1,2)

Hirotake Yaguchi*
Department of Mathematics, Mie University, Tsu City, 514-8507, Japan
yaguchi@edu.mie-u.ac.jp

Abstract

We construct nonalgebraic hash functions MB$n$hash ($n = 32$, 160, 192, 256, $\cdots$, 2048, 4096) by repeating β-transformation M_β on [1,2), $\beta > 1$, which is defined by $M_\beta(t) = \beta t$ mod [1,2). The algorithm of MB$n$hash is simple and flexible, and the randomness of hash values is sufficient. We show that the security of the algorithm of MB$n$hash is reduced to the problem of solving algebraic equations whose degree is higher than 4. We also consider the problem of security which arises when the algorithm of MB$n$hash is implemented in a finite size of bits.

key words. hash function, security, β-transformation, linear mod one transformation, algebraic equation.

§1. Introduction

It is important and sometimes useful to have many algorithm of hash functions whose securities are well-analyzed. In [7] they constructed hash functions based on chaos mappings $\Phi_x : [1,2) \to [1,2)$, $x \in [1,2)$, which is defined by $\Phi_x(t) = 2^S xt$ mod [1,2) if $xt \in [1,2)$, and $= 2^{S-1}xt$ mod [1,2) if $xt \in [2,4)$. ($S = 2, 4$ for example.) Here "a mod [1,2)" is $a - \lfloor a \rfloor + 1$. The analysis of its security, which is rather complicated, was proceeded using Φ_x by deriving algebraic equations of high degree. In this paper we introduce hash function MB$n$hash based on β-transformation M_β on [1,2), $\beta > 1$, which is defined simply by $M_\beta(t) = \beta t$ mod [1,2), and then show that the security of MB$n$hash is sufficient. The map M_β is an extension of the well-known β-transformation T_β on [0,1) defined by $T_\beta(t) = \beta t$ mod 1 [4]. Also M_β is equivalent to a special version of the so-called

* This work was supported by JSPS KAKENHI Grant Number 23500086.

linear mod one transformation $T_{\beta,\alpha}$ on [0,1) defined by $T_{\beta,\alpha}(t) = \beta t + \alpha \bmod 1$ [2] (see §7). Thus M_β has its root on classical ergodic theory. This means that we can make use of various results in that field [2][5]. Moreover the simplicity of M_β makes very clear the mathematical description of the algorithm of MB$n$hash. Resultantly it becomes possible to give more strict and precise (and readable) argument concerning the security of MB$n$hash.

This paper consists of seven sections. In the next section we introduce a tiny random number generator MB32rand which generates 32-bit random numbers by computing $M^{16}_{2^3x_n}(x_n)$, $n = 0, 1, \cdots$. In §3 based on MB32rand we construct a tiny hash function MB32hash. The algorithm of MB32hash consists of two stages, namely, "compression" of an input stream $\mathbf{B}$ of bytes and "scrambling" of the compressed value. The compression of bytes $\mathbf{B} = B_1B_2\cdots B_N$ is proceeded such as $w_k = M_{2^31.e}(w_{k-1} \oplus B_k)$ ($\oplus$ =XOR) and the scrambling of $y \equiv w_N \oplus N$ is proceeded such as $z = M^{16}_{2^3y}(y)$. The hash value of $\mathbf{B}$ is the 32 bits given by $\zeta \equiv \lfloor 2^{32}(2^{11}z - \lfloor 2^{11}z \rfloor)\rfloor$, that is, 32 bits after dropping the first 12 bits of z. Since the algorithm of MB32hash is very simple and flexible, we extend in §4 the algorithm of MB32hash to MB$n$hash, n = 160, 192, 256, 384, 512, 1024, 2048 and 4096. In a real computation, $M_\beta(t)$ is computed using multiple-integer operations instead of real number operations. The relation between SSI$n$hash in [7] and MB$n$hash will be stated in §4.2 (SSI=*S*implified *S*hift *I*nteger, MB=*M*odified *B*eta, say). Randomness of hash values by MB$n$hash and speed of generating hash values are treated at the end of §4. Because the security of MB$n$hash is the same as MB32hash, we discuss in this paper the security of MB32hash. The argument here is more strict and precise than the one given in [7]. We divide the argument into two sections: the security of *algorithm* of MB32hash is discussed in §5 and the security of *implementation* of the algorithm is discussed in §6. We will know that the security of the algorithm of MB32hash is reduced to solving algebraic equations whose degrees are higher than 4. Since we have no general algorithm of solving such algebraic equations, we can say that the security of algorithm of MB$n$hash is very high. The analysis of β-transformation and linear mod one transformation has a long history [2][4][5]. In the last section we summarize some of the results which concern with our M_β. We finally remark that a summary of sections 2, 3, 5 and 6 was reported at the rump session of ASIACRYPT 2011.

§2. A tiny pseudorandom number generator based on β-transformations on [1,2)

For $\beta > 1$, the β-transformation M_β on [1,2) is the function defined by

$$M_\beta(t) \;=\; \beta t \bmod [1,2) \;=\; \beta t - \lfloor \beta t \rfloor + 1, \tag{2.1}$$

where $\lfloor \beta t \rfloor$ is the largest integer not exceeding βt. The following is the graph of $M_{2^3\times 1.2781\cdots}(t)$, $t \in [1,2)$, and $y_n = M^{16}_{2^3x_n}(1)$, $x_n = 1 + \frac{n}{20000}$, $n = 0, 1, \cdots, 19999$, which are computed under the double precision floating point number system.

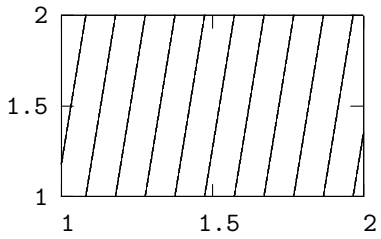


Figure 1. Graph of $M_{2^3\times 1.2781\cdots}(t)$

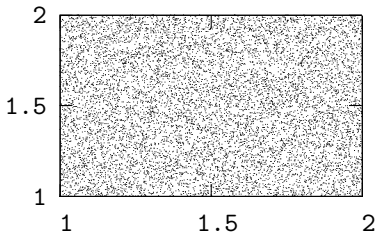


Figure 2. Graph of $y_n = M^{16}_{2^3 x_n}(1)$

Figure 2 suggests us that if we discard the first several digits of y_n and take out succeeding several digits, then we could obtain good random numbers from $\{y_n\}$. In fact we can construct a random number generator based on M_β's as follows.

2.1. Algorithm of MB32rand. Let $1.e$ denote the number $1.27181\cdots$ (e stands for Euler's number $2.7181\cdots$, a transcendental number), and let $1.\hat{r}_1\hat{r}_2\cdots\hat{r}_{31}\cdots$ be the binary representation of $1.e$. For a nonnegative integer $n = \nu_1\nu_2\cdots\nu_{31}$ (binary) we define

$$x_n = 1.(\hat{r}_1\oplus\nu_1)(\hat{r}_2\oplus\nu_2)\cdots(\hat{r}_{31}\oplus\nu_{31})\hat{r}_{32}\hat{r}_{33}\cdots,$$

where the symbol $\oplus$ denotes the logical operation XOR. Let $z_n \equiv M^{16}_{2^3 x_n}(x_n) = 1.\check{b}_5\check{b}_6\cdots\check{b}_{15}\check{b}_{16}\cdots$ (binary). For z_n we drop the first 12 bits and take out the succeeding 32 bits $\check{b}_{16}\check{b}_{17}\cdots\check{b}_{47}$ as the n-th random number ζ_n, that is, ζ_n is the number given by

$$\zeta_n = \lfloor 2^{32}(2^{11}z_n - \lfloor 2^{11}z_n\rfloor)\rfloor. \tag{2.2}$$

As we shall see in §2.3 $\{\zeta_n\}_{n=0,1,\cdots}$ becomes a sequence of random numbers. Hence we call this tiny random number generator the modified-beta 32-bit random number generator (*MB32rand*).

2.2. Implementation of MB32rand. We implement the algorithm of MB32-rand on a computer using 32-bit numbers $1.b_1\cdots b_{31}$ except that a result of multiplication of two numbers $x = 1.x_1\cdots x_{31}$ and $t = 1.t_1\cdots t_{31}$ is represented by 64-bit number $\check{b}_0\check{b}_1.\check{b}_2\check{b}_3\cdots\check{b}_{63}$ with $\check{b}_0\check{b}_1 = 01$ or $\check{b}_0 = 1$ depending on whether the result is in [1,2) or [2,4) respectively. It is convenient for computation to identify a number $1.b_1\cdots b_{31}$ with a 32-bit integer $1b_1\cdots b_{31}$. For example, the first 32 bits of $1.e$ is identified with the 32-bit integer 0xa2cb4411, here "0x" means "in hexa-decimal notation". So the result xt of multiplication of two numbers x and t is identified with a 64-bit integer $\check{b}_0\check{b}_1\check{b}_2\check{b}_3\cdots\check{b}_{63}$ with $\check{b}_0\check{b}_1 = 01$ or $\check{b}_0 = 1$. Since

$$\begin{aligned} M_{2^3x}(t) &= (2^3x)t \bmod [1,2) \ = (2^3)(xt) \bmod [1,2) \\ &= 2^3\times\check{b}_0\check{b}_1\ .\ \check{b}_2\check{b}_3\check{b}_4\check{b}_5\check{b}_6\cdots \bmod [1,2) \\ &= \check{b}_0\check{b}_1\check{b}_2\check{b}_3\check{b}_4\ .\ \check{b}_5\check{b}_6\cdots \bmod [1,2) \\ &= 1\ .\ \check{b}_5\check{b}_6\cdots, \end{aligned}$$

the result $M_{2^3x}(t)$ is identified with the 32-bit integer $1\check{b}_5\check{b}_6\cdots\check{b}_{35}$ extracted from $\check{b}_0\check{b}_1\check{b}_2\cdots\check{b}_{63}$ which corresponds to xt. Notice that the bits $1\check{b}_5\check{b}_6\cdots\check{b}_{35}$ is obtained simply from $\check{b}_0\check{b}_1\check{b}_2\cdots\check{b}_{63}$ by (i) a shift of 4 bits to the left ($\rightarrow \check{b}_0\check{b}_1\check{b}_2\check{b}_3$ is dropped), (ii) a bit-OR with $\check{b}_4$, and (iii) a cut-off of $\check{b}_{36}\cdots\check{b}_{63}$ as follows:

$$\check{b}_0\cdots\check{b}_4\check{b}_5\cdots\check{b}_{35}\cdots\check{b}_{63} \quad \stackrel{shift4}{\longrightarrow} \quad \check{b}_4\check{b}_5\cdots\check{b}_{35}\cdots\check{b}_{63}$$

$$\stackrel{OR}{\longrightarrow} \quad 1\check{b}_5\cdots\check{b}_{35}\cdots\check{b}_{63} \quad \stackrel{cut}{\longrightarrow} \quad 1\check{b}_5\check{b}_6\cdots\check{b}_{35}.$$

The n-th random number ζ_n is the 32-bit integer $\check{b}_{16}\check{b}_{17}\cdots\check{b}_{47}$ obtained from $M^{16}_{2^3x_n} = 1.\check{b}_5\cdots\check{b}_{16}\cdots\check{b}_{47}\cdots\check{b}_{63}$. Under these circumstances we have
ζ_0 =0x6f890520, ζ_1 =0xb16d7669 and so on.

2.3. Statistical test of randomness of MB32rand. Let us show that $\zeta_0, \zeta_1, \cdots$ is a sequence of random numbers. To see the randomness of $\{\zeta_n\}$ we apply the NIST statistical test suite sts-2.1.1 [8]. For NIST's suite we prepare 10 files which consist of $1024^2 \times 1000$ bits of ζ_n, $n = 0, 1, \cdots$. Each file supplies 1000 sequences of 1024^2 random bits to the suite. A result of each test of the suite is given us by a p-value and a fraction(proportion) of sequences passing the test at the level of 0.01. (In the block frequency test we set "block length"=20000.) We repeated NIST's suite ten times using ten files prepared above and averaged ten values of each test. And finally we took the maximum and minimum over 188 tests in the suite to describe the results shortly. The result of NIST's test is the following: (for comparison's sake we also give the result of Mersenne Twister random number generator [10].)

[MB32rand]
P-VALUE AvMax = 0.737001 at NonOverlappingTemplate [$i = 83$]
P-VALUE AvMin = 0.285480 at RandomExcursionsVariant [$i = 181$]
PROPORTION AvMax = 0.992503 at RandomExcursionsVariant [$i = 170$]
PROPORTION AvMin = 0.986800 at FFT [$i = 7$]
[Mersenne Twister ar]
P-VALUE AvMax = 0.713578 at NonOverlappingTemplate [$i = 105$]
P-VALUE AvMin = 0.188745 at NonOverlappingTemplate [$i = 126$]
PROPORTION AvMax = 0.992879 at RandomExcursionsVariant [$i = 168$]
PROPORTION AvMin = 0.986809 at RandomExcursions [$i = 159$]

Here $**$ in [$i = **$] is the serial number of the test in the suite. From these results we can say that ζ_n's are random numbers. Theoretical consideration of ergodic properties of M_β will be given in §7.

§3. A tiny 32-bit hash function based on $\beta-$transformations on [1,2)

We can consider that a hash function is a pseudorandom number generator whose values are determined by input streams of bytes **B** of arbitrary size. Because it is easy to insert byte-data between a repetition of M_β, we can construct

a hash function by making use of the algorithm of MB32rand. In the following we first state the algorithm of a tiny hash function MB32hash, and next give the implementation of the algorithm. The MB32hash is the base of our hash function MB$n$hash ($n = 160, 192, 256, \cdots, 2048, 4096$) which is stated in the next section.

3.1. Algorithm of MB32hash. Let $\mathbf{B} = B_1 B_2 \cdots B_N$ be an input streams of bytes. The algorithm of MB32hash consists of two stages called compression and scrambling.

3.1.1. Algorithm of the compression. Define $w_0 = 1.e$, and for w_{k-1} compute w_k by

$$w_k = M_{2^3 1.e}(w_{k-1} \oplus B_k), \quad k = 1, 2, \cdots, N, \tag{3.1}$$

where

$$w_{k-1} \oplus B_k = 1.b_1 \cdots b_7 (b_8 \oplus c_1) \cdots (b_{15} \oplus c_8) b_{16} b_{17} \cdots b_{31} \cdots \tag{3.2}$$

for $B_k = c_1 \cdots c_8$ in binary. We call the process of obtaining the final w_N from $\mathbf{B}$ the *compression* of $\mathbf{B}$. It is reasonable to think that w_N is generated by using all information of $\mathbf{B}$. To impose in w_N the size N of $\mathbf{B}$ (we assume $N < 2^{31}$), we put

$$y \equiv w_N \oplus N = 1.(b_1 \oplus \nu_1)(b_2 \oplus \nu_2) \cdots (b_{31} \oplus \nu_{31}) b_{32} b_{33} \cdots,$$

where $w_N = 1.b_1 b_2 \cdots$ and $N = \nu_1 \nu_2 \cdots \nu_{31}$ in binary.

3.1.2. Algorithm of the scrambling. To get a 32-bit random number corresponding to $\mathbf{B}$, that is, to get a hash value ζ corresponding to $\mathbf{B}$, we use the algorithm of MB32rand as follows (cf. (2.2)):

$$z = M^{16}_{2^3 y}(y) \quad \text{and} \quad \zeta = \lfloor 2^{32}(2^{11} z - \lfloor 2^{11} z \rfloor) \rfloor. \tag{3.3}$$

We call the process of obtaining z from y the *scrambling* of y.

The process of obtaining a hash value ζ from an input stream of bytes $\mathbf{B}$ is called *MB32hash*.

3.2. Implementation of MB32hash. We use the same 32-bit number system which was introduced in §2.2 when MB32rand was implemented.

3.3. Randomness of MB32hash. In the same way as §2.3 we applied NIST's suite to $\{\zeta_k\}$ where ζ_k is the hash value corresponding to the input stream $\mathbf{B}_k$ of 8 bytes which is identified with an 8-byte integer having the value k. The result is as follows:

[MB32hash]
P-VALUE AvMax = 0.713263 at NonOverlappingTemplate [$i = 20$]
P-VALUE AvMin = 0.223232 at NonOverlappingTemplate [$i = 116$]
PROPORTION AvMax = 0.992982 at RandomExcursionsVariant [$i = 168$]

PROPORTION AvMin = 0.986110 at RandomExcursions [$i = 159$]

3.4. **Increasing the randomness of MB32hash**. Testu01 [1] is a suite of statistical tests of random numbers and consists of three sub-suites named SmallCrush, Crush and BigCrush. Crush and BigCrush require enormous amount of random numbers, and if we apply Crush and BigCrush to the above $\{\zeta_k\}$, then $\{\zeta_k\}$ fails to pass some of tests in them. However the algorithm and implementation of MB32hash are very flexible, and so we can consider various ways of increasing the randomness of MB32hash. One simple way is to use $M_{2^{23}1.e}$ instead of $M_{2^3 1.e}$ in the compression and to use $M^{16}_{2^{23}y}$ instead of $M^{16}_{2^3 y}$ in the scrambling. A hash value ζ is 32 bits given by $\zeta = \lfloor 2^{32}(2^3 z - \lfloor 2^3 z \rfloor) \rfloor$. We implement the above algorithm using the 44-bit number system where MSB(most significant bit) and LSB(least significant bit) are always kept to be 1 such as $1.b_1 b_2 \cdots b_{42}1$. Then new $\{\zeta_k\}$ passes all the tests in Crush and BigCrush. (To set the LSB to be 1 has an effect of increasing the randomness of $\{\zeta_k\}$. (See also §4.2 and §6.1.2.))

§4. Hash function MB*n*hash, n= 192, 256, $\cdots$, 2048, 4096

Because the algorithm of MB32hash is simple and flexible, by changing the map M_β we can construct a series of hash functions MB$n$hash whose hash value is n bits (n = 192, 256, 384, 512, 1024, 2048 and 4096). Since the story is the same for all n, we assume $n = 1024$ in the following.

4.1. Algorithm of MB1024hash.

4.1.1. Algorithm of the compression. Suppose an input streams of bytes is $\mathbf{B} = B_1 B_2 \cdots B_N$. Let $\mathbf{W}_0 = 1.e$ and $P = 128$. For $\mathbf{W}_{k-1} \in [1, 2)$ we compute $\mathbf{W}_k$ by

$$\mathbf{W}_k = M_{2^{P-1}1.e}(\mathbf{W}_{k-1} \oplus \underline{B_{1+(k-1)*E} B_{2+(k-1)*E} \cdots B_{E+(k-1)*E}}) \quad (E = 32) \quad (4.1)$$

where

$$\underline{B_{1+(k-1)*E} B_{2+(k-1)*E} \cdots B_{E+(k-1)*E}}$$
$$= B_{1+(k-1)*E} B_{2+(k-1)*E} \mathbf{0} B_{3+(k-1)*E} B_{4+(k-1)*E} \mathbf{0} \cdots\cdots \mathbf{0} B_{E-1+(k-1)*E} B_{E+(k-1)*E}$$

and $\oplus$(= XOR) is taken from the bit t_P of $\mathbf{W}_{k-1} = 1.t_1 t_2 t_3 \cdots t_P \cdots$. Notice that a null byte $\mathbf{0}$(=0x00) is inserted after every two bytes of $B_{1+(k-1)*E} B_{2+(k-1)*E} \cdots B_{E+(k-1)*E}$ except the last one. (There may be a case that $1 + (k-1) * E \leq N < E + (k-1) * E$ at the last step of the compression.) The meaning of inserting $\mathbf{0}$'s in $\mathbf{B}$ is to keep places in $\mathbf{W}_{k-1}$ where $\mathbf{B}$ can not touch (see §6.1.1). After computing the the last $\mathbf{W}_k$, namely, $\mathbf{W}_{\lfloor (N-1)/E \rfloor + 1} = 1.t_1 t_2 \cdots$, we embed $N = \nu_1 \nu_2 \cdots \nu_{31} \nu_{32}$ (binary) in $\mathbf{W}_{\lfloor (N-1)/E \rfloor + 1}$ such as

$$\tilde{\mathbf{W}} = 1.t_1 \cdots t_{P-1}(t_P \oplus \nu_1) \cdots (t_{P+31} \oplus \nu_{32}) t_{P+32} \cdots , \quad (4.2)$$

and obtain $\tilde{\mathbf{W}}$. (Notice here that there is no need that N should be 32-bit.)

4.1.2. Algorithm of the scrambling. Let $S = 96$ and $Q = 224$. (The number S is chosen so that $16S$ is about one and a half of $n(= 1024)$.) We first compute $\mathbf{Z} = (M_{2^{S-1}\tilde{\mathbf{W}}})^{16}(\tilde{\mathbf{W}})$. Then we drop the first $Q/2 - S$ bits of $\mathbf{Z}$ and take out the succeeding $n = 1024$ bits by

$$\zeta = \lfloor 2^n (2^{(Q/2)-S-1}\mathbf{Z} - \lfloor 2^{(Q/2)-S-1}\mathbf{Z} \rfloor) \rfloor$$

as a hash value.

4.2. Implementation of MB1024hash.

4.2.1. Implementation of the compression. In implementing the compression of MB1024hash, the sizes of bits which we give to $2^{P-1}1.e$ and $\mathbf{W}_{k-1}$ in (4.1) are $M = 160$ and $L = 1024(= n)$ respectively. The reason why we use the short size M instead of n is to reduce the time of multiplication in (4.1). We always keep the MSB and LSB of $1.e$ and $\mathbf{W}_{k-1}$ to be one (cf. §3.4 and §6.1.2). Then a computation $M_{2^{P-1}1.e}(t)$ with $1.e = 1.x_1x_2\cdots x_{M-2}1$ and $t = 1.t_1t_2\cdots t_{n-2}1$ is proceeded as follows:

$$\begin{matrix} 1.e = 1.x_1x_2\cdots x_{M-2}1 \\ t = 1.t_1t_2\cdots t_{n-2}1 \end{matrix} \quad \overset{mul}{\longrightarrow} \quad \check{b}_0\check{b}_1 \,.\, \check{b}_2\cdots\check{b}_{P-1}\check{b}_P\check{b}_{P+1}\cdots\cdots\check{b}_{n+M-2}1 \quad \overset{\times 2^{P-1}}{\longrightarrow}$$

$$\check{b}_0\check{b}_1\check{b}_2\cdots\check{b}_{P-1}\check{b}_P \,.\, \check{b}_{P+1}\cdots\cdots\check{b}_{n+M-2}1 \quad \overset{mod\ [1,2)}{\longrightarrow} \quad 1 \,.\, \check{b}_{P+1}\cdots\check{b}_{P+n-1}\cdots\check{b}_{n+M-2}1$$
$$\overset{cut}{\longrightarrow} \quad 1 \,.\, \check{b}_{P+1}\cdots\cdots\check{b}_{P+n-2}\check{b}_{P+n-1} \quad \overset{OR}{\longrightarrow} \quad 1 \,.\, \check{b}_{P+1}\cdots\cdots\check{b}_{P+n-2}1.$$

Of course, in a real computing, numbers $1.x_1x_2\cdots x_{M-2}1$ and $t = 1.t_1t_2\cdots t_{n-2}1$ are identified with integers $1x_1x_2\cdots x_{M-2}1$ and $t = 1t_1t_2\cdots t_{n-2}1$ respectively. In this case it is not so hard so see that the code of compression of MB$n$hash becomes the same as that of SSI$n$hash in [7].

4.2.2. Implementation of the scrambling. In implementing the scrambling of MB1024hash, we separate the role of two $\tilde{\mathbf{W}}$'s in $(M_{2^{S-1}\tilde{\mathbf{W}}})^{16}(\tilde{\mathbf{W}})$ into $\hat{\mathbf{W}}$ and $\tilde{\mathbf{W}}$ such as $(M_{2^{S-1}\hat{\mathbf{W}}})^{16}(\tilde{\mathbf{W}})$. The sizes of bits which we give to $\hat{\mathbf{W}}$ and $\mathbf{U}$ in $M_{2^{S-1}\hat{\mathbf{W}}}(\mathbf{U})$ are $224(= Q)$ and $1024(= L(= n))$ respectively. We keep the MSB and LSB of $\hat{\mathbf{W}}$ and $\mathbf{U}$ to be one as well. Then if we identify $\hat{\mathbf{W}}$ and $\mathbf{U}$ with integers $1x_1x_2\cdots x_{Q-2}1$ and $1u_1u_2\cdots u_{n-2}1$ respectively, the computation $M_{2^{S-1}\hat{\mathbf{W}}}(\mathbf{U})$ is equivalent to the following integer operations:

$$\begin{matrix} \hat{\mathbf{W}} & : & 1x_1x_2\cdots x_{Q-2}1 \\ \mathbf{U} & : & 1u_1u_2\cdots u_{n-2}1 \end{matrix} \quad \overset{mul}{\longrightarrow} \quad \check{b}_0\check{b}_1\check{b}_2\cdots\check{b}_S\check{b}_{S+1}\cdots\cdots\check{b}_{n+Q-2}1 \quad \overset{shift\ S}{\longrightarrow}$$

$$\check{b}_S\check{b}_{S+1}\cdots\check{b}_{S+n-2}\check{b}_{S+n-1}\cdots\check{b}_{n+Q-2}1 \quad \overset{cut}{\longrightarrow} \quad \check{b}_S\check{b}_{S+1}\cdots\check{b}_{S+n-2}\check{b}_{S+n-1}$$
$$\overset{OR}{\longrightarrow} \quad 1\check{b}_{S+1}\cdots\check{b}_{S+n-2}1.$$

Remark. Let $\tilde{\mathbf{W}}$ in (4.2) be identified with $1w_1w_2\cdots w_{n-2}w_{n-1}$ and put

$$\tilde{x}_0\tilde{x}_1\tilde{x}_2\cdots\tilde{x}_{Q-2}\tilde{x}_{Q-1}$$
$$= (1w_1\cdots w_{Q-1}) \oplus (w_Q w_{Q+1}\cdots w_{2Q-1}) \oplus \cdots \oplus (w_{kQ}w_{kQ+1}\cdots w_{n-1}),$$

here $\oplus$(XOR) is taken bitwise and k is the one satisfying $kQ < n \leq (k+1)Q$. We note that if we replace $1x_1x_2\cdots x_{Q-2}1$ with $1\tilde{x}_1\cdots\tilde{x}_{Q-2}1$ in the above, we have the same algorithm of scrambling as SSI$n$hash in [7]. (Notice that $1\tilde{x}_1\tilde{x}_2\cdots\tilde{x}_{Q-2}\tilde{x}_{Q-2}1$ is determined by using all information of $1w_1w_2\cdots w_{n-2}w_{n-1}$, while $1x_1x_2\cdots x_{Q-2}1 = 1w_1\cdots w_{Q-1}1$ is not. The reason why we employ $1x_1x_2\cdots x_{Q-2}1$ here is to take over the simplicity of the algorithm of MB32hash.)

4.3. Parameters and speed of MB$n$hash.

MB$n$hash	L (size of $\mathbf{W}_k$, $\tilde{\mathbf{W}}$)	M (size of 1.e)	E (bytes Embed.)	P	Q (size of $\hat{\mathbf{W}}$)	S
160	20	8	8	5	12(8)	2
192	24	8	8	5	12(8)	2
256	32	8	8	5	12(8)	3
384	48	12	16	9	12	4
512	64	12	16	9	16	6
1024	128	20	32	17	28	12
2048	256	36	64	33	52	24
4096	512	68	128	65	96	48

Table 1: Values (bytes) of parameters for MB$n$hash

4.3.1. Parameters of MB$n$hash. In Table 1 we give the values of parameters which we used for MB$n$hash, $n = 160, \cdots, 4096$. A value in ( ) is that of SSI$n$hash in [7] which is different from MB$n$hash.

4.3.2. Speed of MB$n$hash. We measured the speed of MB$n$hash in two ways (A) and (B). The (A) is to measure the speed of generating a hash value for long input stream of bytes, while (B) is to measure the speed of generating hash values for short but enormously many input streams. More precisely in (A) the input stream of bytes $\mathbf{B}$ is the 1024^3 null(=0x00) bytes on memory, and in (B) the number of input streams $\mathbf{B}$ is 10^7 and each $\mathbf{B}$ is an 8-byte integer which is given a value $0, 1, 2, \cdots$. Each time in Tables 2 and 3 is an average of three trials measured in seconds. The environment of our computer is the following : OS: Windows 7 Professional 64-bit, CPU: Xeon 3.1 GHz, Memory: 8 GB and Compiler: Intel icl v.12. In the table, $(\text{SHA512})_1$ and $(\text{SHA512})_2$ mean that the source codes are of NIST sts-1.8 and of Gifford [9] respectively. The (32-bit) and (64-bit) indicate that multiplications of multiple-integers were performed by using (32-bit)×(32-bit)=(64-bit) multiplication and (64-bit)×(64-bit)=(128-bit) multiplication respectively.

MB$n$hash →	160	192	256	384	512	1024	2048	4096	$(\text{SHA512})_1$	$(\text{SHA512})_2$
time(32-bit)	6.3	7.0	8.3	7.7	9.7	14.7	22.3	37.0	16.7	5.7
time(64-bit)	-	5.3	6.0	5.7	6.3	8.3	11.3	17.7	-	-

Table 2: Time (A) of hashing 1024^3 bytes on memory

MB$n$hash →	160	192	256	384	512	1024	2048	4096	$(\text{SHA512})_1$	$(\text{SHA512})_2$
time(32-bit)	6.7	8.0	9.3	13.0	22.7	80.3	274.7	965.7	26.0	6.7
time(64-bit)	-	5.3	7.0	10.3	11.3	40.7	128.7	400.0	-	-

Table 3: Time (B) of hashing 10^7 input streams of 8 bytes

4.4. Statistical test of randomness of MB*n*hash. We tested the randomness of hash values $\{\zeta_k\}$ by MB$n$hash, $n = 160, \cdots, 4096$, using the Crush suite in TestU01 V.1.2.3 [1]. Here ζ_k is the hash value corresponding to the input stream $\mathbf{B}_k$ of 8 bytes which is introduced in §3.3. The bits tested are the first 32 bits of each ζ_k and full bits of each ζ_k, respectively. The Crush requires approximately 0x800000000($\approx 3 \times 10^{10}$) 32-bit integers or double precision numbers in [0,1]. When a double precision number is requested, we made a number by setting 32 bits from ζ_k in the mantissa of double precision variable. The results of Crush are that all tests were passed. The maximum time taken by Crush was, of course, for testing the first 32 bits of ζ_k by MB4096hash and was 6600 hours. (We splitted the Crush suite into many parts and executed them in parallel.)

§5. Security of MB32hash from a point of view of algorithm

Since the security of MB$n$hash, $n = 160, \cdots, 4096$ is the same as MB32hash, we consider here and in the next section the security of MB32hash. We argue in this section the security of MB32hash from a point of view of algorithm. The security of MB32hash from a point of view of implementation will be discussed in the next section.

Throughout this section we assume that the size of bits under which each number in [1,2) is represented is sufficiently long (and can be regarded as infinite) because we argue the security from a point of view of *algorithm* and do not care how MB32hash would be implemented.

In the following we consider when two different input streams of bytes $\mathbf{B}$ and $\hat{\mathbf{B}}$ yield the same hash values. We divide the argument into two parts. The first part(§5.1) is to consider when equality $w_N = \hat{w}_{\hat{N}}$ holds after the compression of $\mathbf{B} = B_1 B_2 \cdots B_N$ and $\hat{\mathbf{B}} = \hat{B}_1 \hat{B}_2 \cdots \hat{B}_{\hat{N}}$. The second part(§5.2) is to consider when different y and $\hat{y}$ yield the same 32-bit hash values after the scrambling of

y and $\hat{y}$. The argument below is based on [7] but is more strict and precise.

5.1. Security of the compression from a point of view of algorithm.

5.1.1. We start with the simple case $\mathbf{B} = B_1$ and $\hat{\mathbf{B}} = \hat{B}_1$ in order to show how the argument is proceeded. Since

$$M_{2^3x}(t) \;=\; (2^3x)t \bmod [1,2) \;=\; (2^3)(xt) - \lfloor (2^3)(xt) \rfloor + 1, \tag{5.1}$$

we have

$$w_1 \;=\; M_{2^3 1.e}(w_0 \oplus B_1) \;=\; 2^3(1.e)(1.e \oplus B_1) - \lfloor 2^3(1.e)(1.e \oplus B_1) \rfloor + 1. \tag{5.2}$$

For the time being let γ denote the number $2^3 1.e$ of $M_{2^3 1.e}$, that is,

$$\gamma \equiv 2^3(1.e) = 10.17462\cdots .$$

Then (5.2) is rewritten such as

$$w_1 = M_\gamma(w_0 \oplus B_1) = \gamma(1.e \oplus B_1) - \lfloor \gamma(1.e \oplus B_1) \rfloor + 1, \tag{5.3}$$

and hence $w_1 = \hat{w}_1$ means

$$\gamma(1.e \oplus B_1) - \lfloor \gamma(1.e \oplus B_1) \rfloor = \gamma(1.e \oplus \hat{B}_1) - \lfloor \gamma(1.e \oplus \hat{B}_1) \rfloor. \tag{5.4}$$

Instead of finding B_1 and $\hat{B}_1$ satisfying (5.4) directly, we first search t_1 and $\hat{t}_1$ which are in the interval $(-1, 1)$ and satisfy

$$\gamma(1.e + t_1) - \lfloor \gamma(1.e + t_1) \rfloor = \gamma(1.e + \hat{t}_1) - \lfloor \gamma(1.e + \hat{t}_1) \rfloor, \tag{5.5}$$

and then check that $1.e + t_1$ and $1.e + \hat{t}_1$ are realized by $1.e \oplus B_1$ and $1.e \oplus \hat{B}_1$ respectively. (Notice that the values which t_1 and $\hat{t}_1$ can take are only 2^8 values because B_1 and $\hat{B}_1$ are 8-bit.) Let $s_{11}, \cdots, s_{20}$ be the numbers satisfying $M_\gamma(s) = 1$. (See the graph of M_γ given in Figure 1.) It is clear that $s_i = i/\gamma$. Below we give some values of s_i for reference sake:

$$s_{11} = 1.08112\cdot\cdot, \quad s_{12} = 1.17940\cdot\cdot, \quad s_{13} = 1.27768\cdot\cdot, \quad \cdots, \quad s_{20} = 1.96567\cdot\cdot.$$

We further put $s_{10} = 1$ and $s_{21} = 2$. Now define $I_i = [s_i,\, s_{i+1})$, $i = 10, \cdots, 20$. Then it is easy to see that

$$[1,2) = I_{10} \cup I_{11} \cup \cdots \cup I_{20}, \tag{5.6a}$$

$$M_\gamma(1) = 1.17462\cdots, \quad \lim_{s \to 2-0} M_\gamma(s) = 1.34925\cdots, \tag{5.6b}$$

$$\text{if } \; s \in I_i, \;\; \text{then } \; \lfloor \gamma s \rfloor = i, \;\; i = 10, \cdots, 20, \tag{5.6c}$$

$$\lim_{s \to s_i - 0} M_\gamma(s) = 2, \quad i = 11, \cdots, 20, \tag{5.6d}$$

$$M_\gamma(s) \text{ is linear and increasing on } I_i,\ i = 10, \cdots, 20, \tag{5.6e}$$

$$\frac{d}{ds}\{M_\gamma(s)\} = \gamma \text{ for } s \in (s_i, s_{i+1}),\quad i = 10, \cdots, 20. \tag{5.6f}$$

In order that $1.e+t_1$ and $1.e+\hat{t}_1$ are realized by $1.e\oplus B_1$ and $1.e\oplus \hat{B}_1$ respectively, it is necessary that they are in [1,2). Hence suppose $(1.e+t_1) \in I_p$ and $(1.e+\hat{t}_1) \in I_q$ for some p and q. Then (5.5) becomes

$$\gamma(1.e + t_1) - \gamma(1.e + \hat{t}_1) = p - q \tag{5.7}$$

by (5.6c). Now we regard (5.7) as a linear equation of $\tilde{\gamma}$:

$$\tilde{\gamma}(t_1 - \hat{t}_1) + (-p + q) = 0, \tag{5.8}$$

whose solution $\tilde{\gamma}(t_1, \hat{t}_1, p, q)$ must include $\gamma \equiv 2^3(1.e)$ when t_1, $\hat{t}_1$ vary over the interval $(-1, 1)$ and p, q vary over $\{10, \cdots, 20\}$. If t_1 is equal to $\hat{t}_1$, which is the case of $B_1 = \hat{B}_1$, then any $\tilde{\gamma}$ including $\gamma \equiv 2^3(1.e)$ can be a solution of (5.8) because $t_1 = \hat{t}_1$ means $p = q$. Next assume that $t_1 \neq \hat{t}_1$ and $1.e + t_1$ and $1.e + \hat{t}_1$ are realized by $1.e \oplus B_1$ and $1.e \oplus \hat{B}_1$ respectively. Then t_1 and $\hat{t}_1$ can have the form

$$(+ \text{ or } -)0.0000000b_8b_9 \cdots b_{15}000 \cdots \quad \text{in binary.}$$

Hence the number of values which t_1 and $\hat{t}_1$ can take in (5.8) is at most $2^8 \times 2 = 2^9$. Since p and q are integers which vary from 10 to 20, the number of linear equations which can appear as (5.8) is at most $2^9 \times 2^9 \times 11 \times 11$. Then it is not hard to see that when $t_1 \neq \hat{t}_1$,

$$\gamma \equiv 2^3(1.e) = 2^3\{1 + \tfrac{1}{10}(1 + \tfrac{1}{1!} + \tfrac{1}{2!} + \tfrac{1}{3!} + \cdots)\}$$

is not included in the solutions of such number of linear equations. This means that there are no $\mathbf{B} = B_1$ and $\hat{\mathbf{B}} = \hat{B}_1$ satisfying $B_1 \neq \hat{B}_1$ and $w_1 = \hat{w}_1$.

5.1.2. We next consider the case $\mathbf{B} = B_1B_2$ and $\hat{\mathbf{B}} = \hat{B}_1$. Then the compression of $\mathbf{B}$ and $\hat{\mathbf{B}}$ is proceeded as follows:

$$w_1 = M_\gamma(w_0 \oplus B_1) = \gamma(1.e \oplus B_1) - \lfloor \gamma(1.e \oplus B_1) \rfloor + 1 \quad \text{(by (5.3))},$$

$$w_2 = M_\gamma(w_1 \oplus B_2) = \gamma(w_1 \oplus B_2) - \lfloor \gamma(w_1 \oplus B_2) \rfloor + 1,$$

$$\hat{w}_1 = M_\gamma(w_0 \oplus \hat{B}_1) = \gamma(1.e \oplus \hat{B}_1) - \lfloor \gamma(1.e \oplus \hat{B}_1) \rfloor + 1.$$

Therefore if equality $w_2 = \hat{w}_1$ holds, we have

$$\gamma(w_1 \oplus B_2) - \lfloor \gamma(w_1 \oplus B_2) \rfloor = \gamma(1.e \oplus \hat{B}_1) - \lfloor \gamma(1.e \oplus \hat{B}_1) \rfloor,$$

that is,

$$\gamma\{(\gamma(1.e\oplus B_1) - \lfloor \gamma(1.e\oplus B_1) \rfloor + 1)\oplus B_2\} - \lfloor \gamma\{(\gamma(1.e\oplus B_1) - \lfloor \gamma(1.e\oplus B_1) \rfloor + 1)\oplus B_2\} \rfloor$$
$$= \gamma(1.e \oplus \hat{B}_1) - \lfloor \gamma(1.e \oplus \hat{B}_1) \rfloor. \tag{5.9}$$

In the same way as §5.1.1, instead of finding B_1, B_2 and $\hat{B}_1$ satisfying (5.9) directly, we first search t_1, t_2 and $\hat{t}_1$ in $(-1,1)$ satisfying

$$\gamma\{(\gamma(1.e+t_1)-\lfloor\gamma(1.e+t_1)\rfloor+1)+t_2\}-\lfloor\gamma\{(\gamma(1.e+t_1)-\lfloor\gamma(1.e+t_1)\rfloor+1)+t_2\}\rfloor$$
$$=\gamma(1.e+\hat{t}_1)-\lfloor\gamma(1.e+\hat{t}_1)\rfloor, \tag{5.10}$$

and then check that $1.e+t_1$, $\gamma(1.e+t_1)-\lfloor\gamma(1.e+t_1)\rfloor+1)+t_2$ and $1.e+\hat{t}_1$ are realized by $1.e\oplus B_1$, $(\gamma(1.e\oplus B_1)-\lfloor\gamma(1.e\oplus B_1)\rfloor+1)\oplus B_2$ and $1.e\oplus\hat{B}_1$ respectively. Suppose that

$$(1.e+t_1)\in I_{p_1},\ \{(\gamma(1.e+t_1)-\lfloor\gamma(1.e+t_1)\rfloor+1)+t_2\}\in I_{p_2} \text{ and } (1.e+\hat{t}_1)\in I_q.$$

Then (5.10) becomes

$$\gamma\{(\gamma(1.e+t_1)-p_1+1)+t_2\}-p_2=\gamma(1.e+\hat{t}_1)-q. \tag{5.11}$$

We regard (5.11) as a quadratic equation of $\tilde{\gamma}$:

$$\tilde{\gamma}^2(1.e+t_1)+\tilde{\gamma}(-p_1+1-1.e+t_2-\hat{t}_1)-p_2+q=0, \tag{5.12}$$

whose solution $\tilde{\gamma}(t_1,t_2,\hat{t}_1,p_1,p_2,q)$ must include $\gamma\equiv 2^3(1.e)$ when $t_1,t_2,\hat{t}_1$ and p_1,p_2,q vary over $(-1,1)$ and $\{10,\cdots,20\}$ respectively. Now suppose

$$1.e+t_1,\ \{(\tilde{\gamma}(1.e+t_1)-\lfloor\tilde{\gamma}(1.e+t_1)\rfloor+1)+t_2\} \text{ and } 1.e+\hat{t}_1$$

are realized by

$$1.e\oplus B_1,\ \ \{(\tilde{\gamma}(1.e\oplus B_1)-\lfloor\tilde{\gamma}(1.e\oplus B_1)\rfloor+1)\oplus B_2\} \text{ and } 1.e\oplus\hat{B}_1,$$

respectively. Then t_1,t_2 and $\hat{t}_1$ can be of the form $\pm 0.0000000b_8b_9\cdots b_{15}000\cdots$, and so, by the same reason as §5.1.1, the number of equations which can appear as (5.12) is at most $(2^9)^3\times(11)^3$. Then it would be impossible to show that $\gamma\equiv 2^3(1.e)=2^3\{1+\frac{1}{10}(1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots)\}$ is included in the solutions of such number of quadratic equations. Thus we know that there are no $\mathbf{B}=B_1B_2$ and $\hat{\mathbf{B}}=\hat{B}_1$ satisfying $w_2=\hat{w}_1$.

5.1.3. In general for $\mathbf{B}=B_1B_2\cdots B_N$, in the same way as above, we have

$$w_k=\gamma(w_{k-1}+t_k)-p_k+1\equiv\gamma(w_{k-1}+t_k)+\tilde{p}_k,\quad k=1,2,\cdots,N, \tag{5.13}$$
$$\text{where}\qquad p_k=\lfloor\gamma(w_{k-1}+t_k)\rfloor\qquad\text{and}\qquad\tilde{p}_k=-p_k+1,$$

and hence

$$w_N=\quad\gamma\{(\gamma\cdots\gamma\{(\gamma\{(\gamma(w_0+t_1)+\tilde{p}_1)+t_2\}+\tilde{p}_2)+t_3\}+\tilde{p}_3\cdots+\tilde{p}_{N-1})+t_N\}+\tilde{p}_N$$
$$=\gamma^N(w_0+t_1)+\gamma^{N-1}(\tilde{p}_1+t_2)+\gamma^{N-2}(\tilde{p}_2+t_3)+\cdots+\gamma(\tilde{p}_{N-1}+t_N)+\tilde{p}_N \tag{5.14}$$

with $w_0 = 1.e$. (Notice that $\tilde{p}_k \in \{-19, -18, \cdots, -9\}$.) Therefore if $w_N = \hat{w}_{\hat{N}}$ for $\mathbf{B} = B_1 B_2 \cdots B_N$ and $\hat{\mathbf{B}} = \hat{B}_1 \cdots \hat{B}_{\hat{N}}$, $N \geq \hat{N}$, the equation which γ should satisfy is

$$\tilde{\gamma}^N(1.e + t_1) + \tilde{\gamma}^{N-1}(\tilde{p}_1 + t_2) + \tilde{\gamma}^{N-2}(\tilde{p}_2 + t_3) + \cdots + \tilde{\gamma}(\tilde{p}_{N-1} + t_N) + \tilde{p}_N$$

$$= \tilde{\gamma}^{\hat{N}}(1.e + \hat{t}_1) + \tilde{\gamma}^{\hat{N}-1}(\tilde{q}_1 + \hat{t}_2) + \tilde{\gamma}^{\hat{N}-2}(\tilde{q}_2 + \hat{t}_3) + \cdots + \tilde{\gamma}(\tilde{q}_{\hat{N}-1} + \hat{t}_{\hat{N}}) + \tilde{q}_{\hat{N}},$$

that is, if $N = \hat{N}$,

$$\tilde{\gamma}^N(t_1 - \hat{t}_1) + \tilde{\gamma}^{N-1}(\tilde{p}_1 - \tilde{q}_1 + t_2 - \hat{t}_2) + \tilde{\gamma}^{N-2}(\tilde{p}_2 - \tilde{q}_2 + t_3 - \hat{t}_3) + \cdots$$

$$+\tilde{\gamma}(\tilde{p}_{N-1} - \tilde{q}_{N-1} + t_N - \hat{t}_N) + (\tilde{p}_N - \tilde{q}_N) = 0, \tag{5.15}$$

and if $N > \hat{N}$,

$$\tilde{\gamma}^N(1.e + t_1) + \tilde{\gamma}^{N-1}(\tilde{p}_1 + t_2) + \cdots + \tilde{\gamma}^{\hat{N}+1}(\tilde{p}_{N-\hat{N}-1} + t_{N-\hat{N}})$$

$$+\tilde{\gamma}^{\hat{N}}(\tilde{p}_{N-\hat{N}} - 1.e + t_{N-\hat{N}+1} - \hat{t}_1) + \tilde{\gamma}^{\hat{N}-1}(\tilde{p}_{N-\hat{N}+1} - \tilde{q}_1 + t_{N-\hat{N}+2} - \hat{t}_2) + \cdots$$

$$+\tilde{\gamma}(\tilde{p}_{N-1} - \tilde{q}_{\hat{N}-1} + t_N - \hat{t}_{\hat{N}}) + (\tilde{p}_N - \tilde{q}_{\hat{N}}) = 0. \tag{5.16}$$

Let $\tilde{\gamma} = \tilde{\gamma}(t_1, \cdots, t_N, \hat{t}_1, \cdots, \hat{t}_{\hat{N}}, \tilde{p}_1, \cdots, \tilde{p}_N, \tilde{q}_1, \cdots, \tilde{q}_{\hat{N}})$ be a solution of (5.15) or (5.16). If $\mathbf{B} = \hat{\mathbf{B}}$, then $t_k = \hat{t}_k$ for any k, and so any $\tilde{\gamma}$ including $\gamma \equiv 2^3(1.e)$ can be a solution of (5.15) ($\rightarrow$ no problems occur). Suppose $\mathbf{B} \neq \hat{\mathbf{B}}$. Then by the same argument as §5.1.1 and §5.1.2, the number of algebraic equations which can appear as (5.15) or (5.16) is at most $(2^9)^{N+\hat{N}} \times (11)^{N+\hat{N}}$. Under these restrictions if we look at the form of $\gamma \equiv 2^3(1.e) = 2^3\{1 + \frac{1}{10}(1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots)\}$, we know that it would be impossible to find γ in the solutions of such finite number of algebraic equations. Thus, if the size of bits where each number in [1,2) is represented is nearly infinite, it is very hard (impossible) to find different $\mathbf{B}$ and $\hat{\mathbf{B}}$ whose length are finite and compressed values w_N and $\hat{w}_{\hat{N}}$ are the same.

5.1.4. From (5.14) we have

$$\frac{\partial W_N}{\partial t_k} = \gamma^{N-k+1} = (2^3(1.e))^{N-k+1}. \tag{5.17}$$

Hence for almost every $(t_1, \cdots, t_N)$, if $\triangle_1, \cdots, \triangle_N$ are sufficiently small, then

$$W_N(t_1 + \triangle_1, \cdots, t_N + \triangle_N) - W_N(t_1, \cdots, t_N)$$

$$\approx \sum_{k=1}^{N} \frac{\partial W_N}{\partial t_k} \triangle_k = \sum_{k=1}^{N} (2^3(1.e))^{N-k+1} \triangle_k.$$

From this one might think that one can control the value of W_N by changing $\triangle_k$ of $t_k + \triangle_k$, $k = 1, \cdots, N$. However the values which $\triangle_k$ can take are only discrete 2^8 values because each $t_k + \triangle_k$ should be realized by $t_k \oplus B_k$. Further, (5.17)

tells us that as $N-k$ becomes large, the value of $\frac{\partial W_N}{\partial t_k}$ grows exponentially, and is extremely large compared to $\frac{\partial W_N}{\partial t_h}$ which has small $N-h$. This implies that it is very difficult to adjust the value W_N accurately by changing $\triangle_1, \cdots, \triangle_N$.

5.2. Security of the scrambling from a point of view of algorithm.

5.2.1. Since the scrambling of y is done by $z = M^{16}_{2^3y}(y)$, we can regard the scrambling of y as the compression of $\mathbf{B} = 00\cdots00$ (zeros of 16 bytes) by the map M_{2^3y} instead of $M_{2^31.e}$, that is, $M^{16}_{2^3y}(y)$ is given by

$$u_0 = y, \quad u_k = M_{2^3y}(u_{k-1} \oplus 0), \ k = 1, 2, \cdots, 16, \quad \text{and} \quad z = u_{16} \tag{5.18}$$

(cf. (3.1)). So in the following we can make use of the argument developed in the analysis of the compression of MB32hash. Below in order that the reader can image the graph of $z \equiv u_{16}(y) = M^{16}_{2^3y}(y)$, we give the graphs of $u_1(y) = M_{2^3y}(y)$ and $u_2(y) = M^2_{2^3y}(y)$ (partly) as Figure 3 and 4.

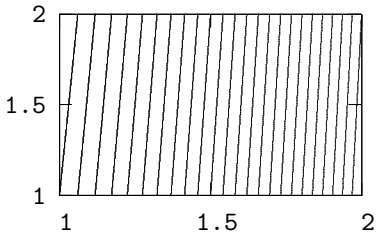


Figure 3. Graph of $u_1(y) = M_{2^3y}(y)$

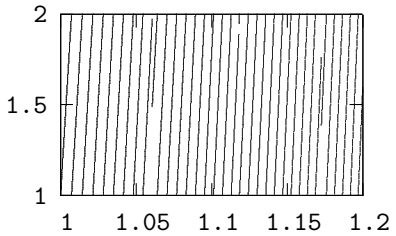


Figure 4. Graph of the initial part of $M^2_{2^3y}(y)$

It is not so hard to see that $u_{16}(y)$ is right continuous and is differentiable (therefore continuous) almost everywhere on [1,2). From (5.18) and (5.1) we have

$$u_0 = y, \quad u_k(y) = 2^3 y u_{k-1}(y) - \lfloor 2^3 y u_{k-1}(y) \rfloor + 1 = 2^3 y u_{k-1}(y) + \check{p}_k, \tag{5.19}$$

where $\check{p}_k = -\lfloor 2^3 y u_{k-1}(y) \rfloor + 1$. Note that

$$\check{p}_k \in \{-30, -29, \cdots, -7\}$$

because y and u_{k-1} are in [1,2). Since each $\check{p}_k$ does not change its value near a point y where $u_{16}(y)$ is differentiable, it holds from (5.19) that

$$\begin{aligned} u'_k(y) &= 2^3 u_{k-1}(y) + 2^3 y u'_{k-1}(y), \\ u''_k(y) &= 2^4 u'_{k-1}(y) + 2^3 y u''_{k-1}(y) \end{aligned}$$

for $k = 1, 2, \cdots, 16$. (Note that u'_k and u''_k are positive.) Hence $u'_{16}(y)$ is expressed such as

$$u'_{16}(y) = \left\{ \sum_{k=0}^{15} 2^3 (2^3 y)^k u_{15-k}(y) \right\} + 2^3 (2^3 y)^{15} y,$$

for almost every y. Because $1 \le u_k < 2$ and $1 \le y < 2$, the above expression yields the following inequality:

$$\sum_{k=0}^{15} 2^3(2^3)^k \le u'_{16}(y) < \sum_{k=0}^{15} 2^4(2^4)^k + (2^4)^{16},$$

which gives us

$$\frac{2^{48}-1}{1-2^{-3}} \le u'_{16}(y) < \frac{2^{68}-1}{1-2^{-4}}. \tag{5.20}$$

This inequality will be used in §5.2.3.

5.2.2. What we want to show below is that it is very difficult to find different y and $\hat{y}$ which yield the same hash values ζ and $\hat{\zeta}$. (Recall that ζ and $\hat{\zeta}$ are extracted from $z = M^{16}_{2^3y}(y)$ and $\hat{z} = M^{16}_{2^3\hat{y}}(\hat{y})$ respectively.) By referring to (5.14) with $w_0 = u_0(= y)$ and $t_k = 0$ (since$B_k = 0$), we have

$$z \equiv z(y) = (2^3y)^{16}y + (2^3y)^{15}\check{p}_1 + (2^3y)^{14}\check{p}_2 + \cdots + (2^3y)\check{p}_{15} + \check{p}_{16}, \tag{5.21}$$

here $\check{p}_k$ is the one defined in (5.19). In the same manner, for $\hat{z}$ we have

$$\hat{z}(\hat{y}) = (2^3\hat{y})^{16}\hat{y} + (2^3\hat{y})^{15}\check{q}_1 + (2^3\hat{y})^{14}\check{q}_2 + \cdots + (2^3\hat{y})\check{q}_{15} + \check{q}_{16}, \tag{5.22}$$

where $\check{q}_k \in \{-30, -29, \cdots, -7\}$ is defined by

$$\hat{u}_0 = \hat{y}, \quad \hat{u}_k = 2^3\hat{y}\hat{u}_{k-1} - \lfloor 2^3\hat{y}\hat{u}_{k-1} \rfloor + 1 \equiv 2^3\hat{y}\hat{u}_{k-1} + \check{q}_k, \quad k = 1, \cdots, 16.$$

(More exactly, $\hat{z}(\hat{y})$ and $\hat{u}_k(\hat{y})$ should be written such as $z(\hat{y})$ and $u_k(\hat{y})$ respectively. Nevertheless we use $\hat{z}$ and $\hat{u}_k$ in order to show their dependence on $\hat{y}$ explicitly when they are abbreviated.) Since the hash values ζ and are $\hat{\zeta}$ extracted from z and $\hat{z}$ by

$$\zeta = \lfloor 2^{32}(2^{11}z - \lfloor 2^{11}z \rfloor) \rfloor \quad \text{and} \quad \hat{\zeta} = \lfloor 2^{32}(2^{11}\hat{z} - \lfloor 2^{11}\hat{z} \rfloor) \rfloor,$$

$\zeta = \hat{\zeta}$ is written as

$$\lfloor 2^{32}(2^{11}z - \lfloor 2^{11}z \rfloor) \rfloor = \lfloor 2^{32}(2^{11}\hat{z} - \lfloor 2^{11}\hat{z} \rfloor) \rfloor, \tag{5.23}$$

which *requires*

$$|2^{32}(2^{11}z - \lfloor 2^{11}z \rfloor) - 2^{32}(2^{11}\hat{z} - \lfloor 2^{11}\hat{z} \rfloor)| < 1.$$

Therefore in order that $\zeta = \hat{\zeta}$ it is necessary that

$$|(z - 2^{-11}\lfloor 2^{11}z \rfloor) - (\hat{z} - 2^{-11}\lfloor 2^{11}\hat{z} \rfloor)| < 2^{-43}. \tag{5.24}$$

5.2.3. It is obvious from the local continuity of $z(y)$ that if y and $\hat{y}$ are very close to each other then we have $\zeta = \hat{\zeta}$. Then how much is the degree of "very

close"? To this question it is natural to think that y and $\hat{y} = y + \eta$, $\eta > 0$, at least satisfy

$$\check{p}_k = \check{q}_k, \ k = 1, \cdots, 16, \tag{5.25}$$

and

$$\lfloor 2^{11} z \rfloor = \lfloor 2^{11} \hat{z} \rfloor. \tag{5.26}$$

In this case η should satisfy $\eta < 2^{-91}$. In fact (5.21), (5.22), (5.25) and the local monotonicity of u_{16} tells us that $u_{16}(y)$ and $\hat{u}_{16}(\hat{y})$ are on the same branch in the graph of u_{16}. Then from the local monotonicity of u'_{16} we have

$$\hat{z}(\hat{y}) - z(y) \ = \ \hat{u}_{16}(\hat{y}) - u_{16}(y) \ = \ u_{16}(\hat{y}) - u_{16}(y) \ \geq \ u'_{16}(y)\eta,$$

which, combined with (5.20), yields

$$\frac{2^{48} - 1}{1 - 2^{-3}} \eta \ \leq \ \hat{z} - z. \tag{5.27}$$

Further (5.26) reduces the condition (5.24) to

$$|z - \hat{z}| < 2^{-43}. \tag{5.28}$$

Therefore if $\eta \geq 2^{-91}$, since

$$\frac{2^{48} - 1}{1 - 2^{-3}} \eta \geq \frac{2^{48} - 1}{1 - 2^{-3}} \cdot 2^{-91} = \frac{8}{7}(2^{-43} - 2^{-91}) > 2^{-43}, \tag{5.29}$$

it holds that $\hat{z} - z > 2^{-43}$ from (5.27), and hence (5.28) is not satisfied. Thus η must satisfy $\eta < 2^{-91}$ if (5.25) and (5.26) hold. However if $\eta < 2^{-91}$, there is a problem in implementing y and $\hat{y} = y + \eta$ because the size of bits which y and $\hat{y}$ can use (currently 32 bits) is insufficient to distinguish the difference of y and $\hat{y}$. So if we want to find y and $\hat{y}$ satisfying $\zeta = \hat{\zeta}$ and $|y - \hat{y}| \geq 2^{-91}$ (in this case (5.25) and/or (5.26) are not satisfied) we have to solve (5.23).

5.2.4. Let us show that to solve (5.23) is very difficult. We first remark that $\lfloor A \rfloor = \hat{\zeta}$ ($\hat{\zeta}$:integer) is equivalent to $\hat{\zeta} \leq A < \hat{\zeta} + 1$. Then (5.23) is rewritten such as

$$\hat{\zeta} \leq 2^{32}(2^{11} z - \lfloor 2^{11} z \rfloor) < \hat{\zeta} + 1, \quad \text{that is,}$$
$$2^{-11} \lfloor 2^{11} z \rfloor + 2^{-43} \hat{\zeta} \leq z < 2^{-11} \lfloor 2^{11} z \rfloor + 2^{-43}(\hat{\zeta} + 1),$$

and so from (5.21)

$$\begin{aligned} & 2^{-11} \lfloor 2^{11} z \rfloor + 2^{-43} \hat{\zeta} \\ \leq \ & (2^3)^{16} y^{17} + (2^3)^{15} \check{p}_1 y^{15} + (2^3)^{14} \check{p}_2 y^{14} + \cdots\cdots + (2^3) \check{p}_{15} y + \check{p}_{16} \\ & < 2^{-11} \lfloor 2^{11} z \rfloor + 2^{-43}(\hat{\zeta} + 1). \end{aligned} \tag{5.30}$$

Remember that a y satisfying (5.30) generates the hash value $\hat{\zeta}$ which is determined by $\hat{y}$. Now we want to find y satisfying (5.30) for some $\hat{\zeta}$. Let $\hat{\zeta}$ be fixed arbitrarily. Since a small perturbation of y does not change the values

$$\check{p}_k,\ k = 1, \cdots, 16, \quad \text{and} \quad \lfloor 2^{11} z \rfloor$$

in (5.30) at a continuous point of y, if y satisfies (5.30) then y must be a solution of the following algebraic inequality of $\check{y}$

$$\begin{aligned} & 2^{-11} Z + 2^{-43} \hat{\zeta} \\ \leq\ & (2^3)^{16} \check{y}^{17} + (2^3)^{15} P_1 \check{y}^{15} + (2^3)^{14} P_2 \check{y}^{14} + \cdots + (2^3) P_{15} \check{y} + P_{16} \\ & < 2^{-11} Z + 2^{-43} (\hat{\zeta} + 1) \end{aligned} \tag{5.31}$$

with $Z = \lfloor 2^{11} z \rfloor$ and $P_k = \check{p}_k$, $k = 1, \cdots, 16$. Therefore if we want to find y whose hash value is equal to $\hat{\zeta}$, then we have to solve an inequality (5.31) whose P_k's are in $\{-30, -29, \cdots, -7\}$ and Z is a 12-bit integer with MSB = 1. However it will be very difficult to solve inequality (5.31) because we have no general algorithms of solving algebraic equations whose degree is higher than 4. Since $\hat{\zeta}$ is arbitrarily fixed, we conclude that it is very hard to solve (5.23) theoretically. (Numerical analysis of (5.31) will be treated in §6.2.)

Finally we give two remarks. (1) For a solution $\check{y}$ of (5.31) P_k's must satisfy $P_k = \check{p}_k$ where $\check{p}_k$'s are integers determined from $\check{y}$ by (5.19). This situation is the same for $Z = \lfloor 2^{11} z \rfloor$. If these restrictions are not satisfied, we can not use $\check{y}$ as a solution satisfying (5.30). However the probability that P_k's and Z pass these restrictions is very low if if they are chosen independently. (2) Even if one may find y and $\hat{y}$ which yield the same hash values ζ and $\hat{\zeta}$, it would be very difficult by the argument given in §5.1 to find input streams $\mathbf{B}$ and $\hat{\mathbf{B}}$ which generate y and $\hat{y}$ respectively.

§6. Security of MB32hash from a point of view of implementation

The argument given in §5 was based on the assumption that the size of bits which each number in [1,2) can use is sufficiently long. However our MB32hash is implemented on the 32-bit system given in §2.2. So we have to discuss particular problems which arise when MB32hash is implemented on a finite-bit number system. The argument here is based on the security of SSRhash in [7] but is rather clear because of the simplicity of M_β.

6.1. Security of the compression from a point of view of implementation. We assume in this subsection that each number in [1,2) is represented using 32 bits $1b_1 \cdots b_{31}$ which was introduced in §2.2. Hence we use 0xa2cb4411 as $1.e$ and w_0.

6.1.1. We first show that if XOR($\oplus$) is taken at the places which include LSB(the least significant bit) such as $b_{24} \cdots b_{31}$, then we can find $\mathbf{B}$ and $\hat{\mathbf{B}}$ whose

compressed values w_N and $\hat{w}_{\hat{N}}$ are the same. Let $\mathbf{B} = (00)(00)$ in hexa-decimal. For $\mathbf{B}$ the compression (3.1) is proceeded as follows:

$$w_0 = a2cb4411 \overset{XOR}{\longrightarrow} a2cb44(1 \oplus 0)(1 \oplus 0) = a2cb4411 \overset{mul}{\longrightarrow}$$

$$6785e38a890f0921 \overset{shift4}{\longrightarrow} 785e38a890f0921 \overset{OR}{\longrightarrow} f85e38a890f0921$$

$$\overset{cut}{\longrightarrow} f85e38a8 = w_1 \overset{XOR}{\longrightarrow} f85e38(a \oplus 0)(8 \oplus 0) = \mathbf{f85e38a8}$$

$$\overset{mul}{\longrightarrow} 9df0d49ac2866328 \overset{shift4}{\longrightarrow} df0d49ac2866328 \overset{OR}{\longrightarrow} df0d49ac2866328$$

$$\overset{cut}{\longrightarrow} df0d49ac = w_2.$$

On the other hand, for $\hat{\mathbf{B}} = (01)(36)$ $\hat{w}_2$ is computed as follows:

$$w_0 = a2cb4411 \overset{XOR}{\longrightarrow} a2cb44(1 \oplus 0)(1 \oplus 1) = a2cb4410 \overset{mul}{\longrightarrow}$$

$$6785e389e643c510 \overset{shift4}{\longrightarrow} 785e389e643c510 \overset{OR}{\longrightarrow} f85e389e643c510$$

$$\overset{cut}{\longrightarrow} f85e389e = \hat{w}_1 \overset{XOR}{\longrightarrow} f85e38(9 \oplus 3)(e \oplus 6) = \mathbf{f85e38a8}(\text{the same as } \mathbf{B})$$

$$\overset{mul}{\longrightarrow} \cdots \overset{cut}{\longrightarrow} df0d49ac = \hat{w}_2.$$

Thus we have $w_2 = \hat{w}_2$. To avoid this problem it is sufficient for us to avoid taking $\oplus$ with lower bits including LSB of w_{k-1}. For example we have only to take $\oplus$ with $b_8 \cdots b_{15}$ in (3.2). By this improvement bits after b_{15} are not touched by the operation $\oplus$, and hence the difference between bits $b_{16} \cdots b_{31}$ of w_{k-1} and $\hat{b}_{16} \cdots \hat{b}_{31}$ of $\hat{w}_{k-1}$ remains as it was.

6.1.2. Next suppose that w_0 and x are equal to 0x80013000, and the size of shift to the left after multiplication is 16 bits. Let $\oplus$ be taken with $b_8 \cdots b_{15}$ of w_{k-1}. Then the compression of $\mathbf{B} = (00)(00)$ by the corresponding map

$$w_k = M_{2^{15}x}(w_{k-1} \oplus B_k) = (2^{15}x)(w_{k-1} \oplus B_k) - \lfloor (2^{15}x)(w_{k-1} \oplus B_k) \rfloor + 1$$

is proceeded as follows:

$$w_0 = 80130000 \overset{XOR}{\longrightarrow} 80(1 \oplus 0)(3 \oplus 0)0000 = 80130000$$

$$\overset{mul}{\longrightarrow} 4013016900000000 \overset{shift16}{\longrightarrow} 016900000000 \overset{OR}{\longrightarrow} 816900000000$$

$$\overset{cut}{\longrightarrow} 81690000 = w_1 \overset{XOR}{\longrightarrow} 81(6 \oplus 0)(9 \oplus 0)0000 = \mathbf{81690000}$$

$$\overset{mul}{\longrightarrow} 40be1acb00000000 \overset{shift16}{\longrightarrow} 1acb00000000 \overset{OR}{\longrightarrow} 9acb00000000$$

$$\overset{cut}{\longrightarrow} 9acb0000 = w_2.$$

On the other hand, the compression of $\hat{\mathbf{B}} = (01)(3f)$ is proceeded as follows:

$$w_0 = 80130000 \overset{XOR}{\longrightarrow} 80(1 \oplus 0)(3 \oplus 1)0000 = 80012000$$

$$\stackrel{mul}{\longrightarrow} 4012815600000000 \stackrel{shift16}{\longrightarrow} 815600000000 \stackrel{OR}{\longrightarrow} 815600000000$$
$$\stackrel{cut}{\longrightarrow} 81560000 = \hat{w}_1 \stackrel{XOR}{\longrightarrow} 81(5 \oplus 3)(6 \oplus f)0000 = \mathbf{81690000}(\text{the same as } \mathbf{B})$$
$$\stackrel{mul}{\longrightarrow} \cdots \stackrel{cut}{\longrightarrow} 9acb0000 = \hat{w}_2.$$

Thus we have $w_2 = \hat{w}_2$. This problem was caused due to too many zero-bits after the multiplication. We can avoid this problem by setting the LSB of x and w_0 to be one as in §3.4, that is, by employing 0x80013001 as x and w_0 instead of 0x80013000. By this improvement, if B_1 and $\hat{B}_1$ are different, then after multiplication and shift the last 16 bits including LSB of w_1 and $\hat{w}_1$ are not all zeros and not the same. For example, for $\mathbf{B} = (00)B_2 \cdots$ and $\hat{\mathbf{B}} = (01)\hat{B}_2 \cdots$, w_1 and $\hat{w}_1$ are computed as follows:

$$w_0 = 80130001 \stackrel{XOR}{\longrightarrow} 80(13 \oplus 00)0001 = 80130001 \stackrel{mul}{\longrightarrow}$$
$$4013016a00260001 \stackrel{shift16}{\longrightarrow} 016a00260001 \stackrel{OR}{\longrightarrow} 816a00260001$$
$$\stackrel{cut}{\longrightarrow} 816a\mathbf{0026} = w_1 \stackrel{XOR}{\longrightarrow} 81(6a \oplus B_2)\mathbf{0026} \longrightarrow \cdots$$

and

$$w_0 = 80130001 \stackrel{XOR}{\longrightarrow} 80(13 \oplus 01)0001 = 80120001 \stackrel{mul}{\longrightarrow}$$
$$4012815700250001 \stackrel{shift16}{\longrightarrow} 815700250001 \stackrel{OR}{\longrightarrow} 815700250001$$
$$\stackrel{cut}{\longrightarrow} 8157\mathbf{0025} = \hat{w}_1 \stackrel{XOR}{\longrightarrow} 81(57 \oplus \hat{B}_2)\mathbf{0025} \longrightarrow \cdots.$$

Since $\oplus B_2$ and $\oplus\hat{B}_2$ in $(6a \oplus B_2)$ and $(57 \oplus \hat{B}_2)$ above can not touch lower 16 bits including LSB of w_1 and $\hat{w}_1$, we have $w_2 \neq \hat{w}_2$ for any B_2 and $\hat{B}_2$. This is one of the reason why we keep the LSB of w_k to be one in MB$n$hash in §4.2.

6.2. Security of the scrambling from a point of view of implementation. In §5.2 we derived an algebraic inequality (5.31). There we said that we have no general theoretical ways of solving algebraic equations whose degrees are higher than 4. However we have another way of solving algebraic equations, that is, we can use numerical analysis. Hence one may think that by referring to numerical solutions of (5.31) one can find different 32-bit numbers y and $\hat{y}$ which yield the same hash values ζ and $\hat{\zeta}$. But this is not affirmative. Suppose for some $\hat{\zeta}$ we can find a numerical solution $\mathbf{Y}$ of (5.31) which satisfies $P_k = \check{p}_k$ and $Z = \lfloor 2^{11}z \rfloor$, and hence satisfies (5.30) with $\check{p}_k = P_k$ and $\lfloor 2^{11}z \rfloor = Z$. (Recall that $\check{p}_1, \cdots, \check{p}_{16}$ and $\lfloor 2^{11}z \rfloor$ are determined automatically from $y = \mathbf{Y}$ by (5.19)). Let $\mathbf{Y} \equiv 1.Y_1Y_2 \cdots Y_{31} \cdots$. From $\mathbf{Y}$ we want to find a 32-bit number $\mathbf{Y}^*$ which keeps the inequality (5.30) with $y = \mathbf{Y}^*$. Let $\mathbf{Y}^* = 1.Y_1Y_2 \cdots Y_{31}$ for example, and compute $\check{q}_1^*, \cdots, \check{q}_{16}^*$ and $\lfloor 2^{11}z^* \rfloor$ for $\mathbf{Y}^*$ using (5.19). Here we assume that the equalities

$$\check{p}_k = \check{q}_k^*, \ k = 1, \cdots, 16, \quad \text{and} \quad \lfloor 2^{11}z \rfloor = \lfloor 2^{11}z^* \rfloor \qquad ((5.25)'(5.26)')(6.1)$$

hold, because at the point of time when (6.1) does not hold (that is, when $\check{p}_k \neq \check{q}_k^*$ or $\lfloor 2^{11}z \rfloor \neq \lfloor 2^{11}z^* \rfloor$ occurs) the inequality (5.30) for $\mathbf{Y}$ becomes meaningless as the inequality which approximates (5.30) for $\mathbf{Y}^*$. Then by the argument at the beginning of §5.2.3 we have $\mathbf{Y} - \mathbf{Y}^* < 2^{-91}$. This requires that when $\mathbf{Y}$ is truncated to a 32-bit number $\mathbf{Y}^*$ the error should be less than 2^{-91}, that is, it is required that at least bits $Y_{32}Y_{33}\cdots Y_{90}$ of $\mathbf{Y}$ should be all zeros. However in reality this is almost impossible. In this way we know that in most cases a numerical solution of (5.30) becomes useless when it is rounded to 32-bit.

Finally let us consider the effect of rounding errors which arise when a 64-bit number $M_{2^3\mathbf{Y}^*}(u_{k-1}^*)$, $k = 1, \cdots, 16$, is truncated to a 32-bit number u_k^*. Since a rounding error of $M_{2^3\mathbf{Y}^*}(u_{k-1}^*)$ is multiplied $16-k$ times by $2^3\mathbf{Y}^*$ of subsequent $M_{2^3\mathbf{Y}^*}$'s, the effect of rounding error grows up very quickly if k is small. Especially rounding errors in early stages causes the same effect as the cut-off of $\mathbf{Y}$ to $\mathbf{Y}^*$ stated above, that is, makes an expected inequality meaningless. Further an amount of rounding error is uncertain and difficult to control. Therefore rounding error increases the resistance of MB32hash against attacks based on numerical analysis.

§7. Ergodic properties of $\beta-$transformations on [1,2)

In this section we discuss properties of β-transformation M_β on [1,2) shortly (for the definition of M_β, see (2.1)). Originally a β-transformation, $\beta > 1$, is defined on the interval [0,1) such as

$$T_\beta(s) \quad = \quad \beta s \bmod 1 \quad = \quad \beta s - \lfloor \beta s \rfloor$$

and is generalized to a linear mod one transformation

$$T_{\beta,\alpha}(x) \quad = \quad \beta x + \alpha \bmod 1 \quad = \quad \beta x + \alpha - \lfloor \beta x + \alpha \rfloor$$

here $0 \leq \alpha < 1$. Investigation of properties of T_β and $T_{\beta,\alpha}$ has a long history and many results are obtained. On the other hand, our β-transformation M_β on [1,2) is related to $T_{\beta,\alpha}$ by

$$M_\beta(t) = T_{\beta,\hat{\beta}}(t-1) + 1, \quad t \in [1,2), \tag{7.1}$$

where $\hat{\beta}$ is the fractional part of β, that is, $\hat{\beta}$ is the number in [0,1) defined by $\beta = \lfloor \beta \rfloor + \hat{\beta}$. The equality (7.1) is verified as follows:

$$\begin{aligned} T_{\beta,\hat{\beta}}(t-1) &= \beta(t-1) + \hat{\beta} - \lfloor \beta(t-1) + \hat{\beta} \rfloor \\ &= \beta t - \lfloor \beta \rfloor - \lfloor \beta t - \lfloor \beta \rfloor \rfloor = \beta t - \lfloor \beta t \rfloor \\ &= M_\beta(t) - 1. \end{aligned}$$

The relation (7.1) says that the graph of M_β is obtained by translating the graph of $T_{\beta,\hat{\beta}}$ to the right by one and then upward by one. In the following we summarize properties of $T_{\beta,\alpha}$ briefly.

Let $X = [0, 1)$ and let $(X, \mathcal{B}, \lambda)$ be the probability space on X where $\mathcal{B}$ is the Borel σ-field and λ is Lebesgue measure.

In [2] Parry showed that $\nu_{\beta,\alpha}(E) = \int_E h_{\beta,\alpha}(x)d\lambda(x)$ where

$$h_{\beta,\alpha}(x) = \sum_{x<T^n_{\beta,\alpha}(1),\ n\geq 0} \frac{1}{\beta^n} - \sum_{x<T^n_{\beta,\alpha}(0),\ n\geq 1} \frac{1}{\beta^n} \tag{7.2}$$

is a finite signed measure which is invariant under $T_{\beta,\alpha}$. (A measure $\nu_{\beta,\alpha}$ is said to be invariant under $T_{\beta,\alpha}$ if $\nu_{\beta,\alpha}((T_{\beta,\alpha})^{-1}(E)) = \nu_{\beta,\alpha}(E)$ for every $E \in \mathcal{B}$.) Moreover if $T_{\beta,\alpha}$ is strongly ergodic, then $h_{\beta,\alpha}(x) \geq 0$ almost everywhere and $\nu_{\beta,\alpha}$ is a finite positive measure invariant under $T_{\beta,\alpha}$. (A map $T_{\beta,\alpha}$ is said to be strongly ergodic if $(T_{\beta,\alpha})^{-1}(E) \subset E$ implies $\lambda(E) = 0$ or $\lambda(E) = 1$.)

In [5] Wilkinson showed that if $\beta > 2$, then $T_{\beta,\alpha}$ is strongly ergodic.

Combining these results we have for $T_{\beta,\alpha}$, $\beta > 2$,

(i) $h_{\beta,\alpha}(x) \geq 0$ for a.e. $x \in X$,
(ii) the probability measure $\mu_{\beta,\alpha}(\cdot) = \nu_{\beta,\alpha}(\cdot)/\nu_{\beta,\alpha}(X)$ is invariant under $T_{\beta,\alpha}$, and
(iii) for $f \in L^1(X, \mathcal{B}, \mu_{\beta,\alpha})$

$$\lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} f(T^n_{\beta,\alpha}(x)) = \int_X f(y)d\mu_{\beta,\alpha}(y) \ \text{ for a.e. } x \in X \ \ ([3]). \tag{7.3}$$

Here a.e. stands for "almost everywhere". We remark that

$$\frac{\beta-2}{\beta-1} \leq h_{\beta,\alpha}(x) \leq \frac{\beta}{\beta-1}, \ \text{ that is, } \ 1 - \frac{1}{\beta-1} \leq h_{\beta,\alpha}(x) \leq 1 + \frac{1}{\beta-1} \tag{7.4}$$

from (7.2) after a simple computation ([5]).

Now let us return to our β-transformation M_β on [1,2). Recall that $T_{\beta,\alpha}$ corresponding to M_β is $T_{\beta,\hat{\beta}}$. Since $\beta_{\tilde{\mathbf{W}}} \equiv 2^{S-1}\tilde{\mathbf{W}}$ in $\mathbf{Z} = (M_{2^{S-1}\tilde{\mathbf{W}}})^{16}(\tilde{\mathbf{W}})$ is larger than 2, the above (i), (ii) and (iii) hold for $T_{\beta_{\tilde{\mathbf{W}}},\hat{\beta}_{\tilde{\mathbf{W}}}}$. From (7.4) and the inequality $2^{S-1} \leq \beta_{\tilde{\mathbf{W}}} \leq 2^S$ we know that

$$1 - \frac{1}{2^{S-1}-1} \leq h_{\beta_{\tilde{\mathbf{W}}},\hat{\beta}_{\tilde{\mathbf{W}}}}(x) \leq 1 + \frac{1}{2^{S-1}-1}. \tag{7.5}$$

If S is 96, which is the case of MB1024hash, (7.5) becomes

$$1 - \frac{1}{2^{95}-1} \quad \leq h_{\beta_{\tilde{\mathbf{W}}},\hat{\beta}_{\tilde{\mathbf{W}}}}(x) \leq 1 + \frac{1}{2^{95}-1} \quad (2^{95} \approx 3.96 \times 10^{28}),$$

which tells us $h_{\beta_{\tilde{\mathbf{W}}},\hat{\beta}_{\tilde{\mathbf{W}}}}(x)$ is nearly 1 independently of $\tilde{\mathbf{W}}$ and x.

Next, suppose the value of $\tilde{\mathbf{W}}$ is w and so $\beta_w = 2^{S-1}w$. (Recall that $\tilde{\mathbf{W}}$ changes its value depending on an input stream $\mathbf{B}$.) Then for a bounded measurable function f on $(X, \mathcal{B}, \lambda)$, we have from (7.3)

$$\lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} f(T^n_{\beta_w,\hat{\beta}_w}(x)) = \int_X f(y) d\mu_{\beta_w,\hat{\beta}_w}(y) \tag{7.6}$$

for almost every x. This means that when N becomes large, then independently of x the empirical measure of $\{T^n_{\beta_w,\hat{\beta}_w}(x)\}_{n=0,1,\cdots,N-1}$ approaches to $\mu_{\beta_w,\hat{\beta}_w}$ whose probability density function is given by $\hat{h}_w(\cdot) \equiv \frac{1}{\nu_{\beta_w,\hat{\beta}_w}(X)} h_{\beta_w,\hat{\beta}_w}(\cdot)$. Because the value w of $\tilde{\mathbf{W}}$ depends on $\mathbf{B}$, the distribution of $\mathbf{Z} = (M_{2^{S-1}\tilde{\mathbf{W}}})^{16}(\tilde{\mathbf{W}})$ for various $\mathbf{B}$ should be described by many $\hat{h}_w$, $w \in [1, 2)$. If $\tilde{\mathbf{W}}$ spreads uniformly in [1,2), then the distribution $H(y)$ of $\mathbf{Z}$ would be given by a uniform mixture of $\hat{h}_w(y-1)$, namely, by $H(y) = \int_1^2 \hat{h}_w(y-1)dw$ (for details, see §5 of [6]).

Acknowledgment. The author would like to express his gratitude to Professors I. Kubo, N. Fukagai and H. Ishitani for their useful comments and informations.

References

[1] P. L'Ecuyer and R. Simard, TestU01: A C Library for empirical testing of random number generators, *ACM Transactions on Mathematical Software*, vol. 33, no. 4 (2007), Article 22. (http://www.iro.umontreal.ca/~simardr/testu01/tu01.html)

[2] W. Parry, Representations for real numbers, *Acta Math. Acad. Sci. Hungar.*, vol. 15 (1964), pp. 95-105.

[3] M. Pollicott and M. Yuri, *Dynamical systems and ergodic theory*, Cambridge Univ Press, Cambridge, 1998.

[4] A. Rényi, Representations for real numbers and their ergodic properties, *Acta Math. Acad. Sci. Hungar.*, vol. 8 (1957), pp. 477-493.

[5] K. M. Wilkinson, Ergodic properties of certain linear mod one transformations, *Advances in Math.*, vol. 14 (1974), pp. 64-72.

[6] H. Yaguchi and I. Kubo, A new nonrecursive pseudorandom number generator based on chaos mappings, *Monte Carlo Methods Appl.*, vol. 14 (2008), pp. 85-98. DOI 10.1515/MCMA.2008.005

[7] H. Yaguchi and S. Ueda, Construction, randomness and security of new hash functions derived from chaos mappings, *Interdisciplinary Information Sciences*, vol. 18 no. 1 (2012), pp. 1-11. DOI 10.4036/iis.2012.1

[8] http://csrc.nist.gov/groups/ST/toolkit/rng/index.html

[9] A. D. Gifford, http://www.aarongifford.com/computers/sha.html

[10] http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/mt19937ar.html

Document for Research Report
(JSPS KAKENHI 2300086)

New nonrecursive pseudo-random number generator and its application to hash function

March, 2014