数理解析研究所講究録
第 85[illegible] 巻 1993 年 86-95

Twisted GFSR: 新しい乱数発生法

京大数理解析研究所　松本 眞　matumoto@kurims.kyoto-u.ac.jp
工業技術院計量研究所　栗田 良春　kuri@nrlm.go.jp

August 13, 1993

Abstract

一様乱数生成のアルゴリズム twisted GFSR 法 [12][13] を紹介する。これは従来の GFSR 法の改良版であり、初期化の容易さ、メモリ効率、および Weight Distribution において優れている。他の生成法との優劣を比較した統計的検定結果も報告する。従来良いと信じられて使われてきた GFSR 乱数が Weight Distribution 検定により棄却できる事実の報告は、確率数値解析の方面にも意味のあるものだと思われる。

1 TGFSR 法

[0,1] 区間に一様に分布する擬似乱数の生成法として、従来の GFSR 法を改良した Twisted GFSR 法 (TGFSR) を紹介する。

w をワードの bit 数、n, m を適当な自然数とする。GFSR 系列 $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ とは、適当な非零初期値 $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n-1}$ に対し GF(2) 上の線形漸化式

$$\mathbf{x}_{n+l} = \mathbf{x}_{m+l} \oplus \mathbf{x}_l \ (l \geq 0) \tag{1}$$

で得られる w-bit 整数の系列であった。ここで $\mathbf{x}_l$ は 0–1 成分からなる GF(2) 行ベクトルとみなされ、$\oplus$ はビットごとの exclusive-or を表す。([0,1] の一様乱数を得るには、$\mathbf{x}_l$ を符号無し整数とみなし $2^w - 1$ で割って正規化する。)

この方法の基本的性質や問題点は、

G0 Implement が容易で生成が高速。

G1 $t^n + t^m + 1$ が primitive*のとき、かつそのときに限り最大周期 $2^n - 1$ を達成する。

G2 乱数の良し悪しは初期値に大きく依存する [3]。

G3 v-bit 精度での均等分布の次数 $k(v)$ の上限は $k(v) \leq \lfloor n/v \rfloor$ である。すべての v に対し同時に上限を満たす初期値を探すことは困難でない [15]。

G4 各ビットは M 系列をなす。これらは三項の漸化式に従うため、0–1 の現れ方に非対称性がある [9][11][12][14]。

*GF(2) 係数のある多項式 $\varphi(t)$ が primitive であるとは、$t^r - 1$ $(r \geq 1)$ が $\varphi(t)$ を約数と持つような自然数 r の最小値が $2^{\deg \varphi} - 1$ であること。

ここで、$\{\mathbf{x}_i\}_{i\in\mathbf{N}}$ が v-bit 精度で k 次均等分布しているとは、擬似乱数の周期を P としたとき、P 個の k-tuple $(\mathbf{x}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_{i+k})_{i=1,\ldots,P}$ の上位 v-bit に着目した場合、2^{kv} 個の可能なパターンがどれも一周期に同じ回数だけ現れる（ただしすべて零は一個少ない）ことであり、v-bit 精度での均等分布の次数とはそのような最大の k である。（ここでは、$k(v)$ で表す。）

われわれの提案する TGFSR 法 [12][13] は、単に漸化式 (1) を (2) に取り替えることによって得られる：

$$\mathbf{x}_{n+l} = \mathbf{x}_{m+l} \oplus \mathbf{x}_l A \ (l \geq 0). \tag{2}$$

ここに A は GF(2) 係数の $w \times w$ 正則行列であり、$\mathbf{x}_l A$ は行ベクトルと行列の GF(2) 係数のかけ算である。この系列をパラメータ (w, n, m, A) で与えられる TGFSR 系列と呼ぶことにしよう。

この系列の基本的性質は

T0 Implementation は容易で生成も比較的高速。

T1 A の特性多項式を $\varphi_A(t)$ とする。$\varphi_A(t^n + t^m)$ が primitive のとき、かつそのときに限り最大周期を達成する。

T2 一周期の「乱数性」は初期値に依存しない。

T3 v-bit 精度での均等分布の次数 $k(v)$ は $k(v) \leq n\lfloor w/v \rfloor$ をみたす。この上限を達成するか否かは A のみで決まり、すべての v に対し同時に上限を達成する A を見つけることはむずかしくない [13]。

T4 各ビットはある M 系列となる。その特性多項式は多くの項を持ち、三項式に見られる偏りは見られない。

[12] では、計算のしやすさから、A として次のような形（古典的に有理標準形とよばれる）の行列を選んだ。（実は、この選択は高次均等分布に欠陥があり、次の章に述べる改良が必要である。[13] 参照。）

$$\begin{pmatrix} & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ a_0 & a_1 & \cdots & \cdots & a_{w-1} \end{pmatrix}.$$

ここで、行ベクトル $(a_0, a_1, \cdots, a_{w-1})$ を $\mathbf{a}$ で表すことにすると、$\mathbf{x}_l A$ のかけ算は次の Step 4 に示されるようにビット演算で容易に実現でき、この場合の TGFSR 法はつぎのようにインプリメントできる。

$x[n]$ をワード長整数のサイズ n の配列とし、j を整数変数とする。

Step 1. $j \leftarrow 0$

Step 2. Set $x[0]$, $x[1]$, ..., $x[n-1]$ to some suitable initial values.

Step 3. Output $x[j]$.

Step 4. $x[j] \leftarrow x[(j+m) \bmod n] \oplus \mathrm{shiftright}(x[j]) \oplus \begin{cases} 0 & \text{if LSB of } x[j]=0 \\ \mathbf{a} & \text{if LSB of } x[j]=1, \end{cases}$

Step 5. $j \leftarrow (j+1) \bmod n$

Step 6. Goto Step 3.

ここで LSB は最も右のビット、shiftright は 1 ビット右シフトを表す。

統計的検定　最大周期性を満たす5種のTGFSR乱数発生プログラムをインプリメントし、種々の統計的検定を行なった。検定結果をTable 1. に示す。まず、11種類のgeneratorsについて説明する。最初の5種、IDがT400, T403, T775, T800, T1600となっているGeneratorsがTGFSR法によるものである。T400は16ビットマシン、T403,T775,T800は32ビットマシン、T1600は64ビットマシンを想定している。例えば、T400のパラメータ$(w,n,m)=(16,25,11)$は先に述べたインプリメントにおけるパラメーター、a =A875は16進表示でStep 4の16ビット整数aを表している。例として、T775のインプリメント例をprogram 1にあげておく。genrand()は最初に呼ばれた時に配列を初期化し最初の乱数を生成、以後呼ばれるたびに新しい乱数を生成して返す。乱数は[0,1]に一様分布する多倍長実数(double)として返されるが、有効なビット数は31ビットである。初期値を表す25個の31ビット整数が配列の初期値として埋め込まれているが、これは任意の値(「すべて0」は除く)を与えてよい。main()は最初の50個の乱数を出力する。このプログラムは説明のための例であって、高速化への努力は全く払われていない。

```
#include <stdio.h>
#define N 25
#define M 8

double
genrand()
{
    static int k = 0;
    static unsigned long x[N]={
        0x4af926d5, 0x05b4290a, 0x73b66573, 0x579f611c, 0x38afd691,
        0x1252c856, 0x34f25af7, 0x5fa2b0a0, 0x4b5e0dbd, 0x53defc12,
        0x60ef3adb, 0x442c54e4, 0x16d43b49, 0x5b2bfcee, 0x7fee454f,
        0x4090ed38, 0x45c11f65, 0x442e82fa, 0x271066a3, 0x2d4d6aec,
        0x28960601, 0x7542be66, 0x2660e987, 0x4448d450, 0x535bd56d
    };
    unsigned long y;
    y = x[k];
    if (x[k]%2 == 0) {
        x[k] = x[(k+M)%N] ^ (x[k] >> 1);
    } else {
        x[k] = x[(k+M)%N] ^ (x[k] >> 1) ^ 0x6c6cb38c;
    }
    k++;
    if (k == N) k=0;
    return((double) y/0x7fffffff);
}


main()
{
    int k;
    for (k=0; k<50; k++)
        printf("%f \n", genrand());
}
```

Program 1. A C-program for a TGFSR generator T775.

L521は[8]で提唱された最初のGFSR法をそのまま写してきたもの、F521は[3]による初期化の改良を施したもの、G607は[17]に提唱された均等分布の次数から見て最良なGFSR generatorである。PF89、PF521はGFSRで三項式の代わりに五項式を用いたものである。

Generator		KS test $N = 2048,\ r = 512,\ t = 64$					RUN test $N = 65536, r = 128, t = 64, k = 6$					WD test $N = 1024,\ r = 8192,\ t = 64$				
ID	Parameters (p means period)	K_t^{+++} K_t^{++-}	K_t^{-++} K_t^{+-}	K_t^{+-+} K_t^{+--}	K_t^{--+} K_t^{---}	5% (1%)	K_{up}^{++} K_{dn}^{++}	K_{up}^{+-} K_{dn}^{+-}	K_{up}^{-+} K_{dn}^{-+}	K_{up}^{--} K_{dn}^{--}	5% (1%)	K^+	K^-	5% (1%)	$[M_3]$	$[M_5]$ $\times 10^{-3}$
T400	$(w,n,m) = (16,25,11),\ p = 2^{400}-1$ a = A875	43.5 86.4	19.3 5.0	68.1 70.2	33.8 70.9	0(0)	90.8 90.4	45.9 40.7	13.2 38.9	96.9 86.4	1(0)	47.1	73.9	0(0)	−13	−31
T403	$(w,n,m) = (31,13,2),\ p = 2^{403}-1$ a = 6B5E CCF6	82.7 32.3	40.5 41.7	43.9 8.9	82.6 71.0	0(0)	79.5 42.7	34.3 12.1	3.2 45.6	95.2 71.0	1(0)	81.2	19.8	0(0)	−36	−101
T775	$(w,n,m) = (31,25,8),\ p = 2^{775}-1$ a =6C6C B38C	93.2 18.5	7.5 87.0	35.2 43.0	91.7 15.5	0(0)	33.9 8.5	69.2 54.9	68.4 57.0	39.5 44.0	0(0)	47.7	51.8	0(0)	13	8
T800	$(w,n,m) = (32,25,7),\ p = 2^{800}-1$ a =8EBF D028	12.9 25.5	87.9 52.2	75.7 34.0	35.1 94.8	0(0)	29.3 53.7	95.4 22.3	0.7 20.8	74.6 83.6	2(1)	17.7	75.9	0(0)	−2	−6
T1600	$(w,n,m) = (64,25,3),\ p = 2^{1600}-1$ a =B380 C13A A838 387E	37.2 32.7	57.0 21.1	58.6 66.1	74.3 34.9	0(0)	54.6 82.7	33.4 14.5	34.0 10.3	39.3 72.0	0(0)	71.8	11.1	0(0)	−3	29
L521	$X^{521}+X^{158}+1,\ p = 2^{521}-1$ original GFSR	84.5 99.0	76.1 14.7	1.4 43.3	100.0 100.0	4(3)	0 0	100.0 100.0	100.0 100.0	0 0	8(8)	100.0	0	2(2)	−416	−1140
F521	$X^{521}+X^{32}+1,\ p = 2^{521}-1$ Fushimi[4]	77.1 36.7	45.7 73.7	48.2 73.5	48.2 66.8	0(0)	88.6 95.3	11.7 2.9	1.6 13.6	85.9 87.3	3(0)	100.0	0.3	2(2)	−373	−927
G607	$X^{607}+X^{273}+1,\ p = 2^{607}-1$ Tootill[16]	39.3 75.9	84.4 56.0	48.3 15.4	24.7 96.7	1(0)	49.8 25.9	17.9 97.6	44.1 94.0	79.1 4.7	2(0)	100.0	1.7	2(1)	−338	−840
PF89	$X^{89}+X^{72}+X^{53}+X^{17}+1$ $p = 2^{89}-1$	34.4 34.0	29.9 50.1	49.0 17.6	75.5 32.0	0(0)	39.8 71.2	37.1 12.0	8.8 7.3	89.5 89.7	0(0)	96.4	11.6	1(0)	−25	−71
PF521	$X^{521}+X^{424}+X^{236}+X^{111}+1$ $p = 2^{521}-1$	72.9 71.2	79.7 2.0	86.8 5.3	16.5 66.3	1(0)	3.2 71.1	89.8 58.9	76.3 67.7	9.0 63.0	0(0)	31.2	45.9	0(0)	28	66
LM	$X_i = a \cdot X_{i-1} \bmod M,\ M = 2^{31}-1$ $a = 2\ 100\ 005\ 341$[7], $p = 2^{31}-2$	67.6 26.9	34.1 68.5	38.7 9.6	35.9 83.4	0(0)	79.7 54.5	11.7 73.1	42.9 50.3	37.9 19.5	0(0)	89.6	14.6	0(0)	21	53

Table 1. Five TGFSR and six other type generators and their results of statistical tests

例えばPF89のみたす漸化式は(1)のかわりに

$$x_{89+l} = x_{72+l} \oplus x_{53+l} \oplus x_{17+l} \oplus x_l \ (l \geq 0)$$

となる[7]。これらのIDに埋め込まれた数は、周期がおよそ2の何乗かを表している。たとえば、T521、F521、PF521の周期はみな同じ$2^{521}-1$である。最後のID "LM"は古典的なLehmerの合同法によるものであるが、スペクトル検定で良い乗算因子および法を選んだ[6]。

統計的検定は3種類行なった。最初の検定はKolmogorov-Smirnov（KS）検定である。これはほとんど[5]から引き写したものであるが、KS検定を3重に繰り返した点で違っている。2048個の一様乱数を生成し、sortしてKS値K^+,K^-を求める。これを512回続けて繰り返して、512組の$K^{\pm}$値を求め、それをKS検定することで4個の値$K^{\pm\pm}$を求める。これをランダムに選んだ64種の初期値に対して行ない、得られた4組の64個の$K^{\pm\pm}$値をKS検定をすることで8個の値$K^{\pm\pm\pm}$を求める。これをKS確率分布のパーセント値になおす。この結果、L521は棄却された。

次の検定はRUN testである。これは、一様乱数のなかで単調に増加する部分列（連）の長さを見るものである。これもほぼ[5]から引き写してきたものなので、詳しい説明はそちらを参照されたい。L521は完全に棄却される。T800がやや疑わしい（が棄却するほどではない）。

最後の検定はWeight Distribution検定である。これは一様乱数から二項分布する乱数をつくり、それをχ^2検定するものである。

まず1024個の一様乱数を生成し、そのなかで1/2以上のものの個数を数える。この数は$r = 1/2$の二項分布に従うはずである。[0,1024]を適当に8つにカテゴリーわけし、それぞれのカテゴリーに落ちてくるものの数が同じぐらいになるようにする。1024個ずつの一様乱数を8192組連続して生成することで、8192個の二項分布する乱数を得る。これに対して自由度7のχ^2検定を行なう。これをランダムに選んだ64種の初期値に対して行ない、得られたχ^2値64個にKS検定を行ない、KS確率分布のパーセントに値に直した。また三次、五次の中心モーメントM_3, M_5も求めた。いずれも理論値は0である。

三項式に基づく生成法（L521、F521、G607）はWD検定で全て棄却された。なぜ三項式が悪いかについてはいろいろな理由がある[9][11]が、その悪さの宣伝はまだ十分でないように思われる。現に、今だに三項式を使った乱数パッケージを作っている会社もあるらしい。なお、三項式を変形して見かけ上多くの項をもつ漸化式を得る方法も研究された[2, 1章2.2.5], [4]が、これらは本質的に三項式で生成された系列から何個かおきにサンプル抽出して得られる系列と一致し、三項式の本質的問題点を解決してはいない(cf.[9])。

TGFSR法、五項式GFSR法、Lehmer法はどの検定でも良好な結果をおさめた。

この章の最後にGFSRとTGFSRの比較について簡単にまとめておく。G607とT800を例にとろう。

- メモリ効率：　G607は607ワードのワーキングエリアを消費するが、T800は25ワードだけである。周期はそれぞれ$2^{607}-1$, $2^{800}-1$である。周期を同じにした時、TGFSRが消費するワーキングエリアはGFSRのワード長分の1（32ビットマシンなら32分の1）ですむ。これは同時にたくさんのgeneratorをインプリメントする必要のある大規模シミュレーションに適している。

- Weight Distribution：　Table 1.に見られる通り、TGFSR法では三項GFSRに見られるWDの偏りはない。

- 初期化：　Table 1に見られる通り、L521はF521に大きく劣る。この二つは初期値が違うだけで生成法は全く同じGFSR法である。よい初期値を選ぶアルゴリズムは[3]にあり、その計算量は大きくはないが何度も異なる初期値を与えたい時にはover headとなる。これに対し、TGFSR法では初期値は全く任意に与えることができる。（但し"全て0"は除く。）

- 生成の速さ： TGFSR 法は GFSR 法に比べて少しビット操作命令が増えているだけで、その生成速度はほんのわずか遅くなるだけである。

TGFSR 法の代数的背景やパラメーターの探し方については、[12] を参照されたい。

2 Tempered TGFSR 法

さて、TGFSR 法による一様乱数は先の検定を全てパスしたが、[16] で指摘された通り、[12] で提唱された TGFSR 法（すなわち、A として有理標準形のものをえらんだ TGFSR）には v-bit 精度 ($v \ll w$) での均等分布の次数 $k(v)$ に欠点があった。すなわち、周期から推論される $k(v)$ の上限は

$$k(v) \leq \lfloor nw/v \rfloor$$

であるのに対し、[12] の TGFSR の $v = 2$ での $k(v)$ の値は

$$k(2) = n \ll \lfloor nw/v \rfloor$$

で、上限からは程遠い。これの意味するところは、TGFSR 法で生成した一様乱数の上位 2 ビットだけに着目する、いいかえれば、[0,1] 区間を 4 等分してどこに落ちたかのみを考えた時、周期から期待される独立性に程遠い独立性しか持っていないことを示している。これを実験的に確かめるため、次のような（一般化されたやや人工的な）Weight Distribution 検定を考案した。先に検定結果を Table 2. に示す。

Generator	LM	T400	T403	T775	T800	TT400	TT403	TT775	TT800
KS^+ (%)	84	100	100	100	100	91	54	32	4
KS^- (%)	22	0	0	0	0	3	26	49	85
M_3	−21	−44	−46	−46	−44	−24	−24	−23	−24
cpu time(s)	97	74	63	72	72	80	70	80	79

Table 2. The result of χ^2-test on the deviation from the binomial distribution of $\#\{i | 1 \leq i \leq N, x_i > 1/4\}$ with uniform random variables $x_i \in [0,1]$, $N = 256$

Generators の名前についてまず説明する。LM は先の Table 1. と同じ Lehmer の合同法のうち特によいパラメーターを選んだもの、T400–T800 は先の TGFSR 法である。TT400–TT800 は A として有理標準形でない行列を使うことに対応する後に述べる改良 “Tempering” を T400–T800 に施したものである。この改良は元の TGFSR プログラムに数行を追加することで簡単に行なうことができる。

（拡張された）Weight Distribution 検定は次のようなものである。まず 256 個の一様乱数を生成する。このうち、1/4 を超えたものの個数を数えると、これは二項分布に従う。そこで、[0,256] を適当に 8 つのカテゴリーにわけ、それぞれのカテゴリーに落ちる確率がほぼ等しくなるようにしておく。256 個の乱数をつづけて 8192 組生成し、8192 個の二項分布に従うはずのサンプルを得る。これに対し 8 つのカテゴリーを使って自由度 7 の χ^2 検定を行ない、二項分布とみなせるか否かを検定する。この検定をランダムに選んだ 64 種の初期値についておこない、得られた 64 個の χ^2 値を KS 検定してパーセント値におとす。あわせて二項分布の中心 3 次モーメント M_3 を求める（この場合の理論値は −24）。

この結果、前章の検定では良い結果をおさめていた TGFSR 法がすべて棄却されてしまった。この検定は上述の欠陥を選択的に抽出するものであるから、当然ともいえる。LM および改良後の TGFSR Generators は検定をパスしている。では、Tempering について具体的に述べよう。それは、先のインプリメントに、y というワード長整数変数を追加し、Step 3 を次の Step 3′ にとりかえることである。

Step 3′. $y \leftarrow x[l] \oplus ((x[l] << s)\&\mathbf{b})$; $y \leftarrow y \oplus ((y << t)\&\mathbf{c})$; output y.

ここに $(y << t)$ は y を左に t ビットシフトした値を表し、& はビットごとの AND 演算を表す。s, t はある自然数、$\mathbf{b}, \mathbf{c}$ はビットマスクとして使われるあるワード長 16 進整数である。Step 3′ によって与えられる写像 $x[l] \mapsto y$ は GF(2) 線形写像であり、ある正則行列 P によって $y := x[l]P$ と表される。このとき、改良前の系列がパラメータ (w, n, m, A) で与えられていたとすると、改良後の系列はパラメータ $(w, n, m, P^{-1}AP)$ で与えられる TGFSR 系列となることが容易に確かめられる。定数 $(s, t, \mathbf{b}, \mathbf{c})$ は、次の定理の上限を達成するように計算機で探す。

定理 1　パラメータ (w, n, m, A) の TGFSR の v-bit 精度での均等分布の次数 $k(v)$ は次を満たす。

$$n | k(v) \text{ かつ } k(v) \le n\lfloor w/v \rfloor \quad (v = 1, 2, \ldots, w).$$

さらに、$k(v)/n$ は A, v, w のみで決まり、n, m によらない。

これは周期から導かれる上限 $\lfloor nw/v \rfloor$ より小さいが n に関するオーダーは同じであり GFSR の上限 $\lfloor n/v \rfloor$ よりはずっと良い。上限を満たすパラメーター $(s, t, \mathbf{b}, \mathbf{c})$ を T400–T800 に対して探し、Table 3 にリストした。

Generator		The order of equidistribution							
ID	Parameters	$k(1)$	$k(2)$	$k(3)$	$k(4)$	$k(5)$	$k(6)$	$k(7)$	$k(8)$
		$k(9)$	$k(10)$	$k(11)$	$k(12)$	$k(13)$	$k(14)$	$k(15)$	$k(16)$
		$k(17)$	$k(18)$	$k(19)$	$k(20)$	$k(21)$	$k(22)$	$k(23)$	$k(24)$
		$k(25)$	$k(26)$	$k(27)$	$k(28)$	$k(29)$	$k(30)$	$k(31)$	$k(32)$
TT400	$(w, n, m) = (16, 25, 11)$	400	200	125	100	75	50	50	50
	a= A875	25	25	25	25	25	25	25	25
	s=2, b= 6A68	*	*	*	*	*	*	*	*
	t=7, c= 7500	*	*	*	*	*	*	*	*
TT403	$(w, n, m) = (31, 13, 2)$	403	195	130	91	78	65	52	39
	a=6B5E CCF6	39	39	26	26	26	26	26	13
	s=8, b=102D 1200	13	13	13	13	13	13	13	13
	t=14, c=66E5 0000	13	13	13	13	13	13	13	*
TT775	$(w, n, m) = (31, 25, 8)$	775	375	250	175	150	125	100	75
	a= 6C6C B38C	75	75	50	50	50	50	50	25
	s=6, b=1ABD 5900	25	25	25	25	25	25	25	25
	t=14, c=776A 0000	25	25	25	25	25	25	25	25
TT800	$(w, n, m) = (32, 25, 7)$	800	400	250	200	150	125	100	100
	a=8EBF D028	75	75	50	50	50	50	50	50
	s=7, b=2B5B2500	25	25	25	25	25	25	25	25
	t=15, c=DB8B0000	25	25	25	25	25	25	25	25
T800	$(w, n, m) = (32, 25, 7)$	800	25	25	25	25	25	25	25
	a=8EBF D028	25	25	25	25	25	25	25	25
		25	25	25	25	25	25	25	25
		25	25	25	25	25	25	25	25
F521	$(w, n, m) = (32, 521, 32)$	521	260	170	130	102	81	72	64
	(a nearly optimal	57	49	41	40	37	33	32	32
	GFSR of 521 words)	30	26	26	24	22	22	22	19
		18	17	16	16	16	16	16	16

Table 3**. k-distribution of four tempered TGFSR(R), one plain TGFSR(R), and one GFSR

ここで、$\mathbf{a}, \mathbf{b}, \mathbf{c}$ は 16 進表記してある。参考のため TT800 の C によるインプリメント例を Program 2. にあげておく。速さのためには、n 回呼ばれるごとに一回、全配列を書き換えたほうが

**If $w = 31$, the most significant bit is always zero in the 32-bit words.

良い (Program 3.) 。

Table 2. の残りの部分は、v-bit 精度での均等分布の次数 $k(v)$ $(v = 1, 2, ..., w)$ の具体的数値である。Tempering によってこれらが定理の上限を満たすようになったことが示されている。T800 は temper 前の元の Generator、F521 はほとんど最良の均等分布を満たす GFSR[2] である。

定理の証明やその他の諸結果、Step 3′ の選択の理由、パラメータ探索のアルゴリズム等については、[13] を参照されたい。(e-mail ででもご連絡いただければ preprint をお送りします。)

Remark (1) 有理標準形を使った TGFSR は [1] の検定のなかの一つに (他の多くの generator とともに) はねられている。Temper 後の TGFSR がどうなのかは確かめていない。
(2) TGFSR に近いメモリ効率をもつ generator が [10] に提唱されたが、[1] によれば乱数性はわるいようである。

```
#include <stdio.h>
#define N 25
#define M 7

double
genrand()
{
    static int k = 0;
    static unsigned long x[N]={
        0x95f24dab, 0x0b685215, 0xe76ccae7, 0xaf3ec239, 0x715fad23,
        0x24a590ad, 0x69e4b5ef, 0xbf456141, 0x96bc1b7b, 0xa7bdf825,
        0xc1de75b7, 0x8858a9c9, 0x2da87693, 0xb657f9dd, 0xffdc8a9f,
        0x8121da71, 0x8b823ecb, 0x885d05f5, 0x4e20cd47, 0x5a9ad5d9,
        0x512c0c03, 0xea857ccd, 0x4cc1d30f, 0x8891a8a1, 0xa6b7aadb
    };

    unsigned long y;

    y = x[k];
    y ^= (y << 7) & 0x2b5b2500;
    y ^= (y << 15) & 0xdb8b0000;
    if (x[k]%2 == 0) {
        x[k] = x[(k+M)%N] ^ (x[k] >> 1);
    } else {
        x[k] = x[(k+M)%N] ^ (x[k] >> 1) ^ 0x8ebfd028;
    }
    k++;
    if (k == N) k=0;
    return((double) y/(unsigned long) 0xffffffff);
}

main()
{
    int k;
    for (k=0; k<50; k++)
        printf("%1.10f \n", genrand());
}

/*
main()
{
    long int k;
    for (k=0; k<5000000; k++) genrand();
}
*/
```

Program 2. A C-program for a Tempered TGFSR generator TT800.

```
#include <stdio.h>
#define N 25
#define M 7

double
genrand()
{
    unsigned long y;
    static int k = 0;
    static unsigned long x[N]={
        0x95f24dab, 0x0b685215, 0xe76ccae7, 0xaf3ec239, 0x715fad23,
        0x24a590ad, 0x69e4b5ef, 0xbf456141, 0x96bc1b7b, 0xa7bdf825,
        0xc1de75b7, 0x8858a9c9, 0x2da87693, 0xb657f9dd, 0xffdc8a9f,
        0x8121da71, 0x8b823ecb, 0x885d05f5, 0x4e20cd47, 0x5a9ad5d9,
        0x512c0c03, 0xea857ccd, 0x4cc1d30f, 0x8891a8a1, 0xa6b7aadb
    };
    if (k==N) {
        genrand1(x);
        k=0;
    }
    y = x[k];
    y ^= (y << 7) & 0x2b5b2500;
    y ^= (y << 15) & 0xdb8b0000;
    k++;
    return( (double) y / (unsigned long) 0xffffffff);
}

genrand1(x)
unsigned long x[];
{
    int k;
    for (k=0;k<N-M;k++) {
        if (x[k] % 2 == 0) {
            x[k] = x[k+M] ^ (x[k] >> 1);
        } else {
            x[k] = x[k+M] ^ (x[k] >> 1) ^ 0x8ebfd028;
        }
    }
    for (; k<N;k++) {
        if (x[k] % 2 == 0) {
            x[k] = x[k+(M-N)] ^ (x[k] >> 1);
        } else {
            x[k] = x[k+(M-N)] ^ (x[k] >> 1) ^ 0x8ebfd028;
        }
    }
}


main()
{
    int k;
    for (k=0; k<50; k++)
        printf("%1.10f \n", genrand());
}
```

Program 3. A faster C-program for a Tempered TGFSR generator TT800.

References

[1] L'Ecuyer, P. Testing random number generators. *the 1992 Winter Simulation Conference.*

[2] 伏見正則　乱数：UP 応用数学選書 12 東京大学出版会, 1989.

[3] Fushimi, M., and Tezuka, S. The k-distribution of generalized feedback shift register pseudorandom numbers. *Commun. ACM* 26, 7(1983), 516–523.

[4] Kashiwagi, H., and Uchimura, T. A simple method for obtaining primitive pentanomials over GF(2)(in Japanese). *J. of the Soc. of Instrument and Control Engineers* 18, 7(July 1982), 747–750

[5] Knuth, D. E. *The art of Computer Programming*, Vol 2: *Seminumerical Algorithms*, 2nd ed. Addison-Wesley, Reading, Mass., 1981.

[6] Kurita, Y. Choosing parameters for congruential random numbers generators. *Recent development in statistics.* Barra, J.R., Ed. North-Holland Publishing Company, 1977

[7] Kurita, Y. and Matsumoto, M. Primitive t-nomials ($t = 3, 5$) over GF(2) whose degree is a Mersenne exponent ≤ 44497. *Math. Comp.* 56 (1991), 817–821.

[8] Lewis, T. G., and Payne, W. H. Generalized feedback shift register pseudorandom number algorithms. *J. ACM* 20, 3(July 1973), 456–468.

[9] Lindholm, J. H. An analysis of the pseudo-randomness properties of subsequences of long m-sequences. *IEEE Trans. Inform. Theory*, IT-14(July 1968), 569–576.

[10] Marsaglia, G. and Zaman, A. A new class of random number generators. *Annals of Applied Probability* 1(1991), 462–480.

[11] M. Matsumoto and Y. Kurita, The Fixed Point of an m-sequence and Local Non-Randomness, technical report 88-027, Department of Information Science, University of Tokyo (1988).

[12] Matsumoto, M. and Kurita, Y. Twisted GFSR generators. *ACM Trans. on Modelling and Computer Simulation* 2(1993), 179–194.

[13] ibid. Twisted GFSR generators II. submitted to *ACM Trans. on Modelling and Computer Simulation.*

[14] 高嶋 恵三　疑似乱数のランダムウォークに依る統計的検定, この講究録.

[15] Tezuka, S. A Heuristic Approach for Finding Asymptotically Random GFSR Generators, JIP. 10, (1987), 178–182.

[16] Tezuka, S. A unified view of long-period random number generators. *A manuscript.*

[17] Tootill, J. P. R., Robinson, W. D., and Eagle, D. J. An asymptotically random Tausworthe sequence. *J. ACM* 20, 3(July 1973), 469–481.