解説

コイン投げで一儲けする方法

—疑似乱数研究の現状—

松本　眞　　慶応大学理工学部数理科学科

宣伝

最近，Mersenne Twister（以下MT）と命名した疑似乱数発生法を西村拓士と共同開発しました．MTはC言語のrand()よりも約4倍高速で，周期は$2^{19937}-1$です．Cコードがただでhttp://www.math.keio.ac.jp/matumoto/mt.htmlより入手できますので，どしどし使ってください（抽象数学研究の社会的意義にご理解をいただけると，なお幸いです）．

コイン投げで一儲けする方法

疑似乱数発生法とは，計算機の中で，あたかもサイコロを振って得られたかのようなでたらめな数列を，高速に，再現性のあるように生成する方法の総称です．たとえばコイン投げのギャンブルを計算機の中で行うために用いられますし，後述のように科学・金融でも幅広い応用があります（実際，MTに対し銀行・保険・物理学者・ゲームメーカなどから200通以上問合せがありました）．

しかし，その応用上の重要性にもかかわらず，疑似乱数の研究と利用は混沌の歴史であるといっていいでしょう．端的な例を挙げます．疑似乱数発生法の「バイブル」として多くの人が参照するのが，D. E. Knuthの教科書[5]です．が，15年前に出版された第二版に対し，Knuth自身を含む専門家たちがその中のいくつかの生成法の欠陥を指摘していました．さて，1997年の第三版において「新しく推薦できる生成法」としてKnuthは*ran_array*を挙げています．この生成法は「安全率」というべきパラメータnを受け取ります．「nは最低100，安全のためには1000ぐらいが薦められる」[5] [p.186]，「ほとんどのユーザにはこの注意は不要だろうが，安全には1009を薦める」[5] [p.188]と彼は述べています．

では，*ran_array*を使って疑似乱数整数を発生し，その奇遇性を用いてコイン投げをしてみましょう．偶数をコインの表に，奇数を裏に対応させます．すると，ギャンブラがよくやる作戦，「表が続いたら次も表だ」という予想に基づいて賭けをすれば，お金がどんどん儲かることが実験で確かめられます．

詳しく言うと，毎回のコイン投げで，自分は表に賭けるかもしくは賭けをしないとします．過去100回のコイン投げの結果を見て，もし60回以上表が出たら，「今は表のツキが来ている」と思って，表に1円賭けます．もし60回未満なら，賭けをしません．そしてまた100回コイン投げを観察し，60回以上表が出たら表に1円賭けます．これを100万回繰り返します．それで，賭けに何回勝ち，何回負けたかを表にしたものが**表-1**です．た

表-1　*ran_array*で儲かるお金

安全率	勝った回数	負けた回数	儲けたお金	儲かる確率
100	14740	13594	1146	5.11816×10^{-12}
200	14527	13955	572	0.000357862
300	14456	14359	97	0.285854
400	14440	14335	85	0.269908
TT800	14250	14322	−72	0.667081

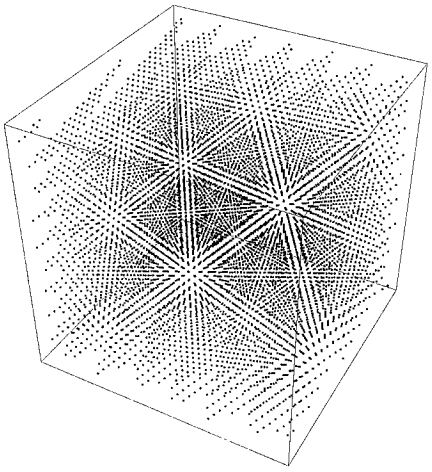


図-1　rand()の3次元プロット拡大図

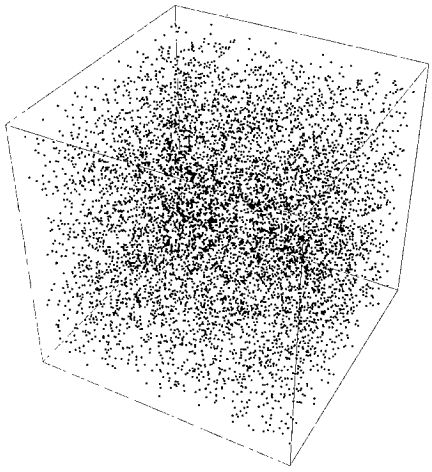


図-2　MT19937()の3次元プロット拡大図

とえば安全率が200のとき，100×1,000,000回のコイン投げの観察で14527+13955=28482回賭けをすることになり，そのうち表は14527回，裏は13955回で，儲けたお金は572円です．28482回の賭けで572円以上のお金が儲かる確率は0.036%未満です．安全率を100とすると28334回の賭けで1146円が儲かり，対応する確率は5.12×10^{-12}ほどです．

表-1から，安全率を300以上にしなければ「ほとんどのユーザにとって問題のない」疑似乱数とはいえないことが分かります[5) P.604]．表の最後にあるTT800は，比較のための別の生成法[11)]です．*ran_array*はnに比例したメモリ領域と計算時間を必要とすることを注意しておきます．

*ran_array*は，評判の良くないlagged-Fibonacci生成法により生成された数列を，100個使ってはn−100個捨てるという方法で改良したものです（捨てるというアイデアはLüscher[8)]によります．彼のオリジナルのアルゴリズムはSWB[9)]を改良したもので，*ran_array*より良いと思われます）．

この例は，疑似乱数の研究と応用の混乱を示しています．古い，あるいは新しくて悪い生成法が広く使われ，問題点を指摘する論文が出版され，小さな改良が行われ，また問題点が見つかり，論文を生産します．一方で，良い生成法は評価されぬまま無視されます．深い理論に基づく論文は，しばしば理解されないからです．このため，良い生成法がユーザの元に届きません．

疑似乱数って何するもの?

計算機科学の持つ夢の1つは，「現実世界を計算機内で完全に模倣する」シミュレーションでしょう．量子力学を持ち出すまでもなく，現実世界は本質的に不確定な要素を持っています．たとえば，連鎖核分裂をシミュレートするとすると，1つ1つの原子がある確率分布に従って崩壊し中性子を放出します．放出する方向も確率的なら，中性子と原子核の衝突の際の分裂も確率的です．このような現象のシミュレーションでは，各原子に対し各タイムクロックごとに乱数を生成して次状態を決めることになります．「巨大な並列すごろく」と思えばいいでしょう．このように，不確定性を持つシステムを乱数を用いてシミュレートすることを**モンテカルロシミュレーション**といいます．

近年のハードウェアの進歩によりシミュレーションは大規模化しましたが，使用されている疑似乱数にはあまり注意が払われていないようです．たとえばC言語の標準疑似乱数rand()は周期2^{30}の31ビット整数を生成するものですが，現在ではパソコンでも数時間で一周期を使い果してしまいます．また，rand()は線形合同法のすべてが持つ「格子構造」を持っています．**図-1**はrand()で3次元単位立方体の中にランダムに点がばらまかれるように普通にプログラムして得られる図の拡大です（ザルツブルグ大学pLab研協力）．生成された乱数列を［0, 1］区間に分布するようスケールを変えて$x_0, x_1, x_2,\ldots\in[0, 1]$とし，$xyz$空間に座標（$x_i, x_{i+1}, x_{i+2}$）の点を$i$=0, 1,...と一周期に渡って打った図の，原点近くの70倍拡大図です．**図-2**は，同じ濃度になるまでMTで点を打ったものです（全周期は長すぎて不可能だし，やれば真っ黒になることが数学的に証明されています）．この種の格子構造は応用上致命的になり得ます．たとえば，a, b, c, d, d'を実数として，問題が

$$d<ax+by+cz<d'$$

なる二平面に囲まれた部分の体積を求めることだったとします．もし，二平面が図-1の格子構造にほぼ平行であれば，体積はd, d'の平行移動に関しては不変であるのに，モンテカルロ法で打たれる点の数は大きく変わることになります．

rand()は古いので仕方がありませんが，最新のパッケ

ージにも，良い疑似乱数が入っていないことが多いのです．1997年のMicrosoft Fortran Powerstation 4.0には，rand()と同様の線形合同法のほかは，伏見[4]による非既約3項M系列が実装されているだけですが，3項M系列はすべて*ran_array*と同様の欠陥を持ちます．この事実は文献7）により1968年から知られていましたが，あまり省みられませんでした．Ferrenberg et al.[2]は，3項M系列などの最新の生成法が物理のIsing Modelシミュレーションで有意の誤差を生むことを報告し脚光を浴びましたが，その欠陥はすでに知られており，統計的検定でも棄却されることが分かっていました[10]．これも，疑似乱数業界の「良い生成法が無視され，悪いと分かっている生成法が繰り返し使われる」という不幸な体質の1つの現れと思えます．

さて，解きたい問題が，変数の少ない数値積分に帰着される場合には，上述のモンテカルロ法よりも「準モンテカルロ法[14]」と呼ばれる乱数以上によく配置された数列を使った方が，大抵劇的に計算量を減らせます．モンテカルロ法，準モンテカルロ法は科学計算のみならず，金融リスク管理[15]などでも広く応用されています．

モンテカルロ法への応用のほか，計算機通信の発達により，「暗号疑似乱数」の重要性が増しています．これは，計算量的に次の数を予想するのが不可能な数列を発生するものです[18]．高速生成されるものがあればモンテカルロ法にも有効なはずですが，現状ではまだ知られていません．多少遅くてもよければ，BBSなど，素因数分解の困難性に依存した生成法があります．

2つの流れ・整数と多項式

疑似乱数には，再現性が求められます．1つは追試のため，もう1つは，あるシステムを最適化するとき，同じ乱数列をなんども入力しながら，システムのパラメータを変えて最適化を行いたいからです．

このため，ほとんどの疑似乱数は無入力有限状態オートマトンで生成されます．つまり，Sを有限集合とし（状態集合という），$f: S\to S$を写像とし（次状態関数という），Xを有限集合とし（出力の集合），$b: S\to X$を写像とします（出力関数という）．この時，次の手順で数列を作ります．

[Step 0] 初期状態と呼ばれる元$s_0\in S$を決め，$b(s_0)\in X$を出力する．

[Step 1] 状態を次状態に動かす，すなわち$s_1:=f(s_0)$とし，$b(s_1)$を出力する．

[Step 2] 状態を次状態に動かす，すなわち$s_2:=f(s_1)$とし，$b(s_2)$を出力する．

以後これを繰り返し，状態をfにより次状態へと変換するたびbにより出力を行う．

こうして得られる数列$b(s_0), b(s_1),...$をXに値をとる一様乱数として使おう，というのがほとんどの疑似乱数です．疑似乱数は，長い周期と良い分布を持たねばなりません．これらが解析できさえすればどんなSでもfでもいいのですが，大きく分けて次の3つの方法があります．

[整数型] $\mathbf{Z}$で整数の集合を表し，$\mathbf{Z}/N$で自然数をNで割った余りの集合を表すと，自然に足し算や掛け算の構造が入る．$S:=\mathbf{Z}/N$を状態集合とし，$f: S\to S$を一次関数として得られる生成法．たとえば，定数a, bにより

$$s_n := a\, s_{n-1}+b \bmod N$$

で自然数の数列を生成する（**線形合同法**と呼ばれる，ここに$\bmod N$はNで割った余りを表す）．高次の漸化式を使う方法もあるが，ここでは省略．

[有限体多項式型] $\mathbf{F}_2$で2元体$\{0, 1\}$を表す．$\mathbf{F}_2[t]$で$\mathbf{F}_2$係数の多項式環を表す．$\varphi(t)$を多項式とし，$\mathbf{F}_2[t]/\varphi(t)$で多項式環を$\varphi(t)$で割った余りの集合を表すと，自然に足し算や掛け算の構造が入る．$S:=\mathbf{F}_2[t]/\varphi(t)$を状態集合とし，$f: S\to S$を$t$を掛ける写像とするもの．つまり，

$$s_n(t) := t\cdot s_{n-1}(t) \bmod \varphi(t)$$

で多項式の列を生成し，係数（の一部）を0, 1のビット列と見て整数化する．

[それ以外] 非線形な写像（2次多項式や分数式など）で次状態を計算する．

整数型の簡単な例を見てみます．$s_n=3s_{n-1} \bmod 7$だと，$s_0=1$, $s_1=3$, $s_2\equiv 3\times 3=9\equiv 2$, $s_3\equiv 3\times 2=6$, $s_4\equiv 3\times 6\equiv 4$, $s_5\equiv 3\times 4\equiv 5$, $s_6\equiv 3\times 5\equiv 1$で，132645132645と周期6で巡回します（ここに$\equiv$は7で割った余りが等しいことを示します）．rand()はa=1103515245, b=12345, $N=2^{31}$に対応する線形合同法です（厳密にいうとANSI CのBSD版のrand()）．

有限体型は，もう少し説明が複雑です．$\mathbf{F}_2=\{0, 1\}$には，通常の和と積において，1+1 = 2を1+1=0で置き換えて得られる演算が定義されます（つまりExclusive-or）．すると，0, 1を係数とする多項式にも和，積，差，余りつき割算が導入されます．たとえば，

$$s_n(t)=t\cdot s_{n-1}(t) \bmod t^3+t+1, s_1(t)=1$$

で多項式の列を生成するとしましょう．$s_2(t)=t\times 1=t$, $s_3(t)=t^2$, $s_4(t)\equiv t^3=(t^3+t+1)+t+1\equiv t+1$（1+1=0に注意），$s_5(t)\equiv t(t+1)=t^2+t$, $s_6(t)\equiv t^3+t^2=(t^3+t+1)+t^2+t+1\equiv t^2+t+1$, $s_7(t)\equiv t(t^2+t+1)=t^3+t^2+t=(t^3+t+1)+t^2+1\equiv t^2+1$, $s_8(t)\equiv t(t^2+1)=t^3+t=(t^3+t+1)+1\equiv 1$で，周期7で循環します（ここに$\equiv$は$t^3+t+1$で割った余りが等しいことを表す）．$t^2, t, 1$の係数のみを書くと，001, 010, 100, 011, 110, 111, 101, 001で循環します．

今回発表したMTは，$\varphi(t)$としてある19937次の多項式を使い，周期が$2^{19937}-1$となるような有限体多項式型の疑似乱数発生法です．

線形合同法では周期はNを超えません．掛け算をす

るので，Nは大体計算機の1ワード整数に近い数がとられます．Nを2^wの形にとると高速ですが，乱数性はあまりよくありません（rand()など）．というのは，$k<w$に対して，生成数列s_nはmod 2^kでも同じ漸化式を満たすことになり，s_nの下kビットの周期は2^k以下で，（場合の数がそれしかないから），小さなkに対しては乱数性が損なわれるからです．

なんにせよ，N=2^{32}程度では現在の大規模利用には向きません．たとえば，核分裂シミュレーションでは，2^{20}個の原子核の反応をシミュレートし，それぞれの原子のシミュレーションに最大2^{30}個程度の乱数を使用します．都合，2^{50}程度の乱数を使ってしまいます（こういったシミュレーションのために，Pentiumを数千個つなげた並列計算機がアメリカに現存するそうです）．全周期を使うと格子構造があらわになってしまいます．経験則により「M個の疑似乱数を使いたいときは，周期はM^3以上のものを使うべし」と言われています．これを認めますと，周期2^{150}程度の発生法が現に必要になっているわけです．

有限体多項式型の中で広く使われているのが，3項M系列またはGFSRと呼ばれている生成法です．これは，$\mathbf{x}_n$を0, 1からなるワード長の横ベクトルとし，⊕でビットごとのExclusive-or（つまり$\mathbf{F}_2$ベクトルとしての和）を表したとき，漸化式

$$\mathbf{x}_{j+n}=\mathbf{x}_{j+m}\oplus\mathbf{x}_j \qquad (j=1, 2, 3,...)$$

でワード長の数列を生成する方法です（$n>m$は適当な自然数．これがなぜ本質的には上述の有限体多項式型と思えるのかは文献16）[Ch.4]を参照してください）．1ビット目，2ビット目，...，それぞれが同じ$\mathbf{F}_2$内の漸化式

$$x_{j+n}=x_{j+m}+x_j$$

を満たします．この漸化式の作る01列は，状態集合$S:=\{(x_n, x_{n-1}, ..., x_1)\,|\,x_i\in\mathbf{F}_2,\}$，次状態関数$f: S\to S: (x_n, x_{n-1},..., x_1)\mapsto(x_{n+1}, x_n,..., x_2)$，ただし$x_{n+1}:=x_{m+1}+x_1$，出力関数$b:S\to\mathbf{F}_2:(x_n, x_{n-1},..., x_1)\mapsto x_1$で与えられるオートマトンで生成されます．状態数は$2^n$で，(0,0,...,0)は不動点ですから周期はたかだか2^n-1です．この上限を達成する0, 1の列のことを**3項M系列**といいます．3は漸化式の項の数，Mは周期maximalのMです．周期が極大になるようなn, mに対して，対応する特性多項式t^n+t^m+1を原始3項式といいます（現在見つかっている最大のものはn=859433, m=288477です[6), 5) [P.29]]）．n=521くらいがよく使われています．ベクトル列$\{\mathbf{x}_j\}$の周期もたかだか2^n-1です．この生成法は高速で周期も長く，初期化をうまくやれば[3)]比較的良い乱数が作れるのですが，0, 1の個数について，n次元を超えたところで偏差がおき，ランダムウオークなどで狂いを生むことが知られていました[7), 10), 11), 5) [P.193 Exercise 14]]．

非線形な次状態関数を用いた生成法もさかんに研究されており，同じ周期の線形の生成法より乱数性が優れているようですが，計算時間がかかること（線形のものの数倍），長周期のものが作りにくいなど，現時点での応用性はそれほど高くないと思います．

新世代の疑似乱数

1990年代に入って，非常に長い周期を持ちかつメモリを浪費しない生成法がいくつか開発されました．1つはCombined Tausworth[1)]という，いくつかのm系列をExclusive-orで混ぜ合わせるという方法です．周期はそれぞれの周期の積となり，高位ビットの高次元での均等分布性（k-distribution性[3)]と呼ばれる）も最適化されています．

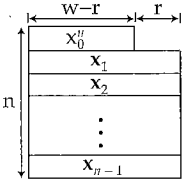
もう1つはSubtract with borrow（SWB[9)]）という方法で，線形合同法のNを非常に大きくして1つの配列の状態集合を$\mathbf{Z}/N$と対応させ，aをうまく選んでa倍がポインタのつけ替えだけでほとんど求められるようなデータ構造を使います．そのままでは高位ビットの乱数性が悪い[2), 17)]ので改良が必要です．1つの方法は，速度を犠牲にして生成列の大部分を捨てること[8)]です．

最後は，Twisted GFSR[10), 11)]という方法です．MT[12)]はこれの変形版で，次の漸化式を使って疑似乱数を生成します．wを計算機のワード長とし，rを$0\le r<w$なる自然数定数，n, mを自然数定数とします．

$$\mathbf{x}_{k+n}:=\mathbf{x}_{k+m}+(\mathbf{x}_k^u\,|\,\mathbf{x}_{k+1}^l)A, (k=0, 1,...). \qquad (1)$$

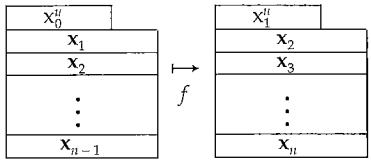
ここで，$(\mathbf{x}_k)_{k=0, 1,...}$は$w$次元の$\mathbf{F}_2$係数横ベクトルの列，$(\mathbf{x}_k^u\,|\,\mathbf{x}_{k+1}^l)$は$\mathbf{x}_k$の成分のうち左から$w-r$個とって，$\mathbf{x}_{k+1}$の成分のうち右から$r$個とってつなげて$w$次元のベクトルとしたもの，$A$は適当な$w\times w$行列（twist行列[10)]）です．さらに，$\mathbf{x}_k$に右から調律行列$T$(tempering[11)])を掛けて出力列とします．

オートマトンでいうと，状態集合Sは次の形に並べられた$nw-r$個の0, 1のパターン全部です．



この図形のことを「欠けた配列」と呼びましょう．欠いた理由は，$2^{nw-r}-1$を素数周期としたいためです．周期が素数なら，大きな数でも確認方法があるのです．

次状態関数fは次のように表されます．



ここで$\mathbf{x}_n$は漸化式（1）の$k=0$の場合から得られます．出力関数bは欠けた配列の最後の1行に，右から行列Tを掛けるという写像です．

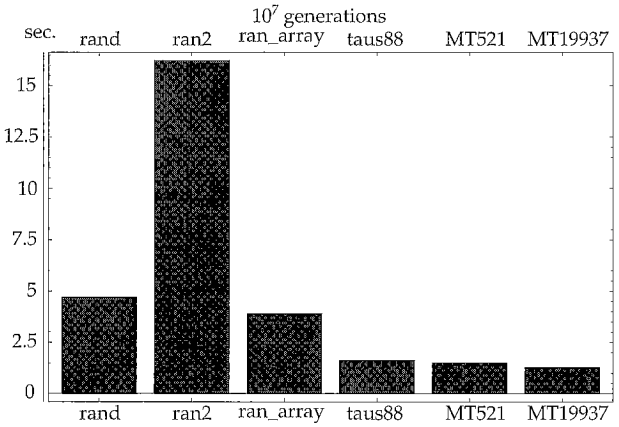


図-3　速度の比較

fは欠けた配列の中身を上にずらして最後の一行を書き換えて得られますが，計算機の中では中身を上にずらすかわりにポインタを下にずらし，ポインタが下からはみでたら上に持っていくという方法の方が高速です．配列の一番上とm番目を指すポインタ1，2を用意します．ポインタ1の行と次の行，ポインタ2の行を読んで演算をし，ポインタ1の行に書き込み，2つのポインタを下にずらせば，計算量は配列の長さに依存しません．

MTはビット演算と和のみを使い，掛け算や割算を含まない上，一度に配列全部に乱数をまとめて生成することでキャッシュメモリやパイプラインに合った実装が可能です．**図-3**はいくつかの生成法のスピードの比較です．MT19937はS. Cokusにより最適化されたもので，だいぶ高速であることが分かります．randは前述，ran2はCの数値計算パックNumerical Recipeにあるもの（線形合同法の組合せ），*ran_array*はKnuthの薦めた大部分を捨てたLagged Fibonacci, taus88はL'EcuyerらのCombined Tausworth, MT521は17ワードを使う周期$2^{521}-1$のMTです．SWBも，文献8）の改良をすると*ran_array*以上に遅くなります．

課題は山積み

疑似乱数をめぐる今後の課題は山積みです．なんと言っても，「疑似乱数性を評価する決定的基準」の不在が，研究を困難にしています．格子構造に関してはk-distributionやspectral test[5]など理論的基準が存在しますが，それで十分とは限りません．一方，多数の統計的検定法が提唱されていますが，それらの有効性はよく分かっていません．

将来の1つの方向性は，計算量的に安全な疑似乱数発生法で高速なものの開発です．そこまででなくても，非線形な次状態関数で良い乱数性を持つ高速算法の可能性はあります．

別方向の課題として，並列システム上での疑似乱数発生が挙げられます．数千のCPUを持つ並列システム上で，どう疑似乱数を生成するかは難しい問題です．我々は，MTのパラメータ探索の高速性を生かして，Dynamic Creation（DC）という，乱数生成法を動的に生成する作戦を提唱しました[13]．DCは，プロセッサIDやプロセスIDなどを受け取って，動的にMTのパラメータを探します．特性多項式にIDが埋め込まれるよう探索するので，違うIDの発生法は互いに素な特性多項式を持つようになっています．必要となる発生器の数の上限があらかじめ分かっている場合には，MTのパラメータをその数だけ表にしておけば初期化の時間が短縮されます．この方法は，素粒子シミュレーションなどで活用できるのではと思います．

謝辞　Twisted GFSRは1996年4月に亡くなられた米田信夫先生とのdiscussionから生まれたものです．MTを開発・実現した西村拓士君に感謝します．

参考文献

1) Couture, R. and L'Ecuyer, P. and Tezuka, S.: On the Distribution of k-dimensional Vectors for Simple and Combined Tausworthe, Sequences, Math. Comp., 60, pp.749-761 (1993).
2) Ferrenberg, A.M., Landau, D.P. and Wong, Y.J.: Monte Carlo Simulations: Hidden Errors from "good" Random Number Generators Phys. Rev. Lett., 69, pp.3382-3384 (1992).
3) Fushimi, M. and Tezuka, S.: The k-distribution of Generalized Feedback Shift Register Pseudorandom Numbers. Commun. ACM, 26, pp.516-523 (1983).
4) Fushimi, M.: Random Number Generation with the Recursion $X_t=X_{t-3p}\oplus X_{t-3q}$. J. Comp. and Appl. Math., 31, pp. 105-118 (1990).
5) Knuth, D.E.: Seminumerical Algorithms, The Art of Computer Programmings 2, 3rd Ed., Addison Wesley (1997).
6) Kumada, T., Hannes, L., Kurita, Y. and Matsumoto, M.: New Primitive t-nomials (t=3, 5) over GF(2) Whose Degree is a Mersenne Exponent, Keio Science and Technology SUURI Research Report 98-009 (1998).
7) Lindholm, J. H.: An Analysis of the Pseudo-Randomness Properties of Subsequences of Long m-sequences, IEEE Trans. Inform. Theory, 14, pp.569-576 (1968).
8) Lüscher, M.: A Portable High-Quality Random Number Generator for Lattice Field Theory Simulations., Comput. Phys. Commun., 79, pp.100-110 (1994).
9) Marsaglia, G. and Zaman, A.: A New Class of Random Number Generators, Ann. Appl. Prob., 1, pp.462-480 (1991).
10) Matsumoto, M. and Kurita, Y.: Twisted GFSR Generators, ACM Trans. Model. Comput. Simul., 2, pp.179-194 (1992).
11) Matsumoto, M. and Kurita, Y.: Twisted GFSR Generators II, ACM Trans. Model. Comput. Simul., 4, pp.254-266 (1994).
12) Matsumoto, M. and Nishimura, T.: Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudorandom Number Generator, ACM Trans. on Model. Comput. Simul., Vol. 8, pp.3-30 (1998).
13) Matsumoto, M. and Nishimura, T.: Dynamic Creation of PseudoRandom Number Generator, 第3回モンテカルロ・準モンテカルロ法国際会議（Claremont, 1998）口頭発表，論文準備中.
14) Niederreiter, H.: Random Number Generation and Quasi-Monte Carlo Methods, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania (1992).
15) 二宮祥一，田島　玲，水田秀行: 金融リスク管理におけるITの最前線，情報処理，Vol.39, No.8, pp.794-799 (Aug.1998).
16) Tezuka, S.: Uniform Random Numbers: Theory and Practice, Kruwer Academic Publishers (1995).
17) Tezuka, S., L'Ecuyer, P. and Couture, R.: On the Lattice Structure of the Add-with-Carry and Subtract-with-Borrow Random Number Generators, ACM Trans. Model. Comput. Simul., 3, pp.315-331 (1993).
18) 渡辺　治: 一方向関数の基礎理論，離散構造とアルゴリズムIII，近代科学社，pp.77-114 (1994).

（平成10年8月21日受付）