J. Plasma Fusion Res. Vol.96, No.6 (2020) 292‐299

講座 シミュレーションのための乱数入門

2．乱数発生の原理

2. Random Number Generation

佐竹真介，菅野龍太郎
SATAKE Shinsuke and KANNO Ryutaro
自然科学研究機構 核融合科学研究所

（原稿受付：2020年3月17日）

乱数は，発生法により大きく2つに分けることができ，1つは，決定論的な演算による疑似乱数，もう1つは，ランダムな自然現象を乱数源として利用する物理乱数です．乱数を利用するシミュレーションは，当然のことながら利用した乱数に，その計算結果が影響されます．そのため，利用する乱数について，事前に理解しておくことが大切です．本章では，まずは，疑似乱数，物理乱数それぞれの発生法について概説します．

Keywords:
pseudo random number, physical random number, quantum random number generator

2.1 疑似乱数の発生法

コンピュータシミュレーションでは，様々な用途で乱数が必要となります．必要な数が10個程度であれば，人の手でサイコロを振るなり乱数表を引くなりすればよいですが，乱数を使うシミュレーションで要求される乱数の数は一般的に数万から数億個といった膨大な量になります．これは乱数を利用するシミュレーションの多くが，ある種のランダムな事象をプログラムで模擬し，その統計的な性質に基づいて解を求めるという考え方，言い換えれば，ある種のアンサンブル平均を数値計算で評価することによって解を得るという考え方に基づいて作られているからです．一般的に，確率変数のアンサンブル平均の統計誤差は標本数 N に対し $1/\sqrt{N}$ に比例して小さくなります．しかし，もし乱数と思って使っている数列に予期せぬ偏りや相関がある場合，それを使っていくら多くの標本を用意したとしても，得られる答え（統計量）の信頼性は上がりません．よって，性質の良い乱数を効率よく発生させることは，乱数を使うシミュレーションの規模が大きくなるほど，またより高い精度の計算が要求されるほど重要になると言えるでしょう．

ところで，我々が現在利用できるコンピュータは，内部的にはデジタルに表現される数値の四則演算と論理演算（AND，OR など），ビット操作（2進数で表現された数値を全体に右または左にシフトする，特定のビットの値を変更するなど）を組み合わせた手続きによって動いており，完全に決定論的な演算しか行えません．一方で，乱数というのはその名の通りランダムで予測不可能な数列ということですから，決定論的な計算機のアルゴリズムからこれを生成するというのはおかしな話にも聞こえます．

では我々がコンピュータプログラムで利用している乱数とは何なのでしょうか？本節ではまず，我々が通常プログラムで利用している「疑似乱数」の発生原理について説明します．疑似乱数とは決定論的なアルゴリズムから生成される「乱数のように見える」数列のことです．本節ではいくつかの代表的な疑似乱数の生成方法と，それらの利点・欠点を紹介します．なお本節の内容は[1-3]を参考に書かれています．一方2.2節では，まだあまり一般的に普及していませんが，疑似乱数とは異なり現実の確率的事象の観測値から乱数列を取得する「物理乱数」について紹介します．また，疑似乱数の発生法の高速化・並列化といった数値的な技法や，疑似乱数の品質，つまり「乱数らしさ」をどう定義し，評価するかについてはそれぞれ第3章，第4章で詳しく解説します．

● 線形合同法

さて，決定論的な計算機の演算から乱数のように見える数列を生み出す方法ですが，一般的に使われる方法は全て何らかの漸化式の計算によって生成します．また，コンピュータの内部では数字は2進数で表現されますから，ここで説明する疑似乱数とは4バイトや8バイトの2進数で表現された整数だと考えて下さい．単純で古くからよく知られた疑似乱数は「線形合同法」と呼ばれるもので，$\{x_i\}: i = 1, 2, \ldots$ を疑似乱数列とした時，一般的に以下のような漸化式で表現されるものを指します．

$$x_{i+1} = a_0 x_i + a_1 x_{i-1} + \cdots + a_j x_{i-j} + b \,(\mathrm{Mod}\, P) \qquad (1)$$

ここでパラメータ a_j，b，P はいずれも非負の整数で $a_j < P$，$b < P$ であり，Mod P は（1）式右辺を P で割った

National Institute for Fusion Science, Toki, GIFU 509-5292, Japan

corresponding author's e-mail:satake@nifs.ac.jp

ⓒ2020 The Japan Society of Plasma Science and Nuclear Fusion Research

余りを取る，という意味です．したがって疑似乱数x_iはPより小さい非負の整数になります．

これがどのような疑似乱数となるかはパラメータ(a_j, b, P)の選び方に強く依存します．疑似乱数のアルゴリズムを考える上で重要となる性質はその「統計的性質」，「周期性」，「再現性」の3つです．統計的性質とは生成された性質がいかに乱数っぽく振る舞うかのことで，詳しい説明は第4章で行いますが，ここでは私たちが一般的な確率的事象に対して期待するような振る舞いを，疑似乱数がどれだけ上手く模擬できているかのことだと考えて下さい．周期性についてはこの線形合同法に限らず，有限の有効桁数で演算されるコンピュータの漸化式による疑似乱数発生法には必ず固有の周期長がある，というのが重要な点です．つまり，(1)式のような漸化式で数列を生成していくと，どこかでまた同じ数列の繰り返しになるということです．また，決定論的なアルゴリズムなので，同じ初期条件$\{x_{i-j}, x_{i-j+1}, \cdots, x_{i-1}, x_i\}$を与えれば同じ乱数列$x_{i+1}, x_{i+2}, \cdots$が毎回得られる，つまり疑似乱数には再現性があります．乱数なのに再現性があるというのもおかしく聞こえますが，同じ計算を同じ条件で走らせた場合に同じ結果が得られるかどうかは，シミュレーションコードの開発時のバグ取りや検証のためには必要不可欠な性質です．

線形合同法のうち，特に$j=0$で，$x_{i+1} = a_0 x_i \,(\mathrm{Mod}\, P)$という最も簡単な形をとったものは乗積合同法，$x_{i+1} = a_0 x_i + b \,(\mathrm{Mod}\, P)$の形のものは混合合同法とも呼ばれ，一つ前の値$x_i$のみ記憶していればよいので，メモリや計算速度に制限の多かった古い時代からよく使われた疑似乱数です．なお，現代では混合合同法のことを線形合同法と呼ぶことが多いようです．

C言語やJAVA，Visual Basic，ExcelやUNIX OSの標準的なライブラリーに入っている乱数ルーチンの中身は，現在でも混合合同法であることが多いようです．こうした標準化された乱数ルーチンでは，疑似乱数列の初期化のためにseedという値を最初にユーザが指定することになっていますが，これは乗積合同法や混合合同法における最初のx_0を与えているに過ぎません．係数(a_0, b, P)は，発生する乱数列の周期長がなるべく長く，乱数としての性質がよくなるように選ばれていますが，原理的にその周期長はPを超えることはありません．例えば，C言語のstdlibに実装されているRAND( )関数は4バイト整数の疑似乱数ルーチンですが，そのパラメータは$a_0 = 1103515245$，$b = 12345$，$P = 2^{32}$であり，周期長は2^{31}です．

さて，古くから使われてきた線形合同法ですが，その乱数としての性質にはいくつかの問題があることが知られています．「乱数らしさ」とは何かについては，また第4章で詳しく述べますが，線形合同法に関する問題点としては，2進数で表現した乱数の下位ビットに，乱数列全体としての周期よりも短い周期性が現れること，「結晶構造」と呼ばれる分布の偏りが見られること，そして最近使われるようになった他の新しい疑似乱数ルーチンに比べて周期長が短いことが挙げられます．「結晶構造」とは，例えば$0 < x_i < M$の整数疑似乱数列を$r_i = x_i/M$により$(0, 1)$区間の実数に変換した場合，n次元空間に$(r_{nk+1}, r_{nk+2}, \cdots, r_{n(k+1)})$，$k = 0, 1, 2, \ldots$と点を並べていくと，結晶のように規則的な格子状に点が並んでしまうことを指します．例えば混合合同法の場合，2次元平面に並べるとこの結晶構造が現れます．これは係数(a_0, b, P)をどのように選んでも現れます．また，Pは周期長が最大になるように，疑似乱数のバイト長と同じ（4バイト整数なら$P = 2^{32}$とするなど）に取ることが一般的ですが，Pが偶数の場合，一番下位のビットが必ず0，1が交互に現れる，つまり偶数，奇数が交互に現れるといった性質もあります．乱数を用いたシミュレーションの中には，このような規則性が計算結果に影響を与えてしまうものもあるため，疑似乱数として望ましい性質とは言えません．周期長が$2^{31} \approx 20$億というのも，100万個の粒子を使ったモンテカルロシミュレーションを2000ステップ進めただけで1周期使い切ってしまうということで，現在の大規模計算においては十分に長い周期とは言えなくなってしまいました．

そこで，線形合同法とは異なる疑似乱数がいろいろと考案されてきました．ここではその代表例としてTausworthe（トーズワース）の方法と，その発展形であるMersenne-Twister（メルセンヌ・ツイスター）法について簡単に紹介します．

●Tausworthe 法

Tausworthe法は上に述べた多次元分布における結晶構造を示さない疑似乱数として知られており，M系列と呼ばれる疑似乱数の仲間の一つです．$\boldsymbol{x}_i$をwビットの2進数を表す横ベクトルとするとき，以下の漸化式

$$\boldsymbol{x}_{i+p} = \boldsymbol{x}_{i+q} \oplus \boldsymbol{x}_i \qquad (\text{ただし } p > q > 0) \tag{2}$$

によって生成されます．ここで，$\oplus$は各ビットごとに排他的論理和（XOR）を取ることを意味します．M系列乱数について詳しく説明するには，$\{0, 1\}$である2つの要素に対するガロア体GF(2)における議論が必要になりますので，詳しくは[1]を参照ください．ここではごく簡単な説明にとどめておきます．(2)式に対してGF(2)上の多項式$x^p + x^q + 1$を特性多項式と呼びます．もし，$x^t - 1$という式が$t = 2^p - 1$の場合この多項式で割り切れ，$t < 2^p - 1$に対しては割り切れない場合，この特性多項式を特にp次の原始多項式と呼びます．このとき，(2)式から生成される数列の各ビットに着目すると，それぞれの$\{0, 1\}$の並びのパターンの周期が$2^p - 1$となることが知られています．原理的に，(2)式の漸化式が作る数列の周期は，$2^p - 1$より長くなることはないので，原始多項式から生成される数列は最大周期長を持つことになります．

線形合同法と同様にTausworthe法でも，原始多項式となるような(p, q)の組み合わせを上手く選ぶ必要があります．代表的な例として，$(p, q) = (607, 273)$，$(1279, 418)$等が知られています．さらにTausworthe法ではp個のwビットベクトル$\boldsymbol{x}_i$を初期条件として与える必要があり，漸化式を回すためにp個分の過去の情報を保存しておく必要があります．一つ留意しなければならないのは，原始多

項式によって生成される数列が最大周期長を与えるのは $w=1$ ビットの場合であり，$w \geq 2$ ビットになった場合には，$\boldsymbol{x}_i$ ベクトルの初期条件を上手く設定しないと，w ビットベクトル $\boldsymbol{x}_i$ 全体のビットパターンの周期長が 2^p-1 より短くなってしまう場合がある点です．幸い適切な初期ベクトルの与え方は Tausworthe 法のサンプルプログラムに付随していますので，ユーザは自分で適当に初期ベクトルを定義せずにそれを利用しましょう．サンプルプログラムは[3-5]などで見つけることができます．

Tausworthe 法の疑似乱数の性質については，多次元分布における一様性が良好であることが特徴です．また，線形合同法の周期長 P はプログラムが扱える最大の整数 (2^{16} や 2^{32}) を超えられませんが，Tausworthe 法の場合，(p,q) の上手い組み合わせさえ見つけられれば，$p \gg w$ と取れるため格段に長い周期長 2^p-1 を実現できます．XOR 演算もCPUで高速実行されますし，使用するメモリ量も線形合同法より少し増える程度です．しかしながら，多次元分布の均等性以外の乱数としての性質については必ずしもよいというわけではありませんでした[6]．また，(2)式において各 w ビットが独立に計算されるため，ビット間で情報のやり取りがないことが欠点として指摘されています．

● Mersenne-Twister 法

そこで，更なる改良として考案されたのが Mersenne-Twister 法 [7] (以下 MT 法と略す) です．これは，日本の数学者，松本 眞氏によって考案された方法で，非常に長い周期性を持つ疑似乱数列が生成でき，乱数としての性質も良いことから，今では世界中の乱数を使うシミュレーションで幅広く利用されるようになりました．

MT 法の漸化式は形式的に以下のように記述されます．

$$\boldsymbol{x}_{i+n} = \boldsymbol{x}_{i+m} \oplus (\boldsymbol{x}_i^{w-r} | \boldsymbol{x}_{i+1}^{r}) \boldsymbol{A} \tag{3}$$

ここで $n > m > 1$ です．また，$\boldsymbol{x}_i$ は(2)式と同様に w ビットの2進数を表す横ベクトルであり，A は $\{0,1\}$ を要素とする $w \times w$ の正方行列で，次のような形を取ります．

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 & & \\ 0 & 0 & 1 & & \\ 0 & 0 & 0 & & \\ & & & \cdots & \\ & & & & 1 \\ a_0 & a_1 & a_2 & \cdots & a_{w-1} \end{bmatrix}$$

ここで，$\boldsymbol{a}$ はある定数ベクトルで，演算 $\boldsymbol{x}\boldsymbol{A}$ は以下のような演算に相当します．

$$\boldsymbol{xA} = \begin{cases} \boldsymbol{x} \text{ を左に1ビットシフト} & (\boldsymbol{x} \text{ の最下位ビットが0}) \\ \boldsymbol{x} \text{ を左に1ビットシフト} \oplus \boldsymbol{a} & (\boldsymbol{x} \text{ の最下位ビットが1}) \end{cases}$$

また，$\boldsymbol{x}_i^{w-r} | \boldsymbol{x}_{i+1}^{r}$ は $\boldsymbol{x}_i$ の上位 $w-r$ ビットと $\boldsymbol{x}_{i+1}$ の下位 r ビットをつなぎ合わせて1つの w ビットベクトルにするという操作です．MT 法の漸化式は(3)式ですが，乱数として利用する際には，更に以下のような一連のかき混ぜ操作を行ったベクトル $\boldsymbol{z}$ を出力として取ります．

$$\begin{aligned} \boldsymbol{y}_1 &= \boldsymbol{x} \oplus (\boldsymbol{x} \gg u), \\ \boldsymbol{y}_2 &= \boldsymbol{y}_1 \oplus (\boldsymbol{y}_1 \ll s) \ \& \ \boldsymbol{b}), \\ \boldsymbol{y}_3 &= \boldsymbol{y}_2 \oplus (\boldsymbol{y}_2 \ll t) \ \& \ \boldsymbol{c}), \\ \boldsymbol{z} &= \boldsymbol{y}_3 \oplus (\boldsymbol{y}_3 \gg l), \end{aligned} \tag{4}$$

ここで u, s, t, l はある整数定数，$(\boldsymbol{b},\boldsymbol{c})$ は w ビットの定数ベクトル，そして $\gg u$，$\ll l$ はそれぞれ u ビット右シフト，l ビット左シフトを表します．(3)式と(4)式は一見複雑な演算を行っているように見えますが，XOR，AND，ビットシフトなど簡素な2進数演算のみで構成されており，コード化すると思いの外短く記述できます．

MT 法の特徴の1つはその周期長の長さにあります．適切にパラメータ m，n，r，$\boldsymbol{a}$ を選ぶことにより，生成される数列 $\boldsymbol{x}_i$ の周期長が $2^p-1=2^{nw-r}-1$ になることが数学的に証明されています[7]．先に挙げた Tausworthe 法と同様，その肝は漸化式(3)を決めるパラメータの選び方にあります．その詳細については数学の専門家ではない筆者の理解を超えるため，考案者の講義資料[3]を参照いただくことにします．ごく簡単に説明すると，数列の漸化式の形を表す特性多項式が x^p-1 の原始多項式になる，そのようなパラメータの組み合わせを見つけ出すことが一般的には非常に難しいのですが，p が Mersenne 数と呼ばれる，$p=2^n-1$ の形を取る特別な素数の場合には，特性方程式が原始多項式か否かの判定が容易になるという性質を使っているそうです．Mersenne-Twister の名前もこれが由来になっています．また，(4)式のような操作は乱数の高次元分布の均等性をより高めるための工夫です．「高次元分布の均等性」の説明は第4章で改めてしますが，例えば下に示す MT 法のサンプルでは周期長が $2^{19937}-1$ で，623次元空間に均等分布する疑似乱数を生成します．

MT 法については，様々な人が Fortran，C，Java，Python など様々な言語で実装しており，MT 法の考案者の松本氏のwebページにはそれらへのリンクがまとめてあります[8]．なお，初期のバージョンの MT 法のソースコードには，乱数の初期化のseedの与え方があまり良くないものが使われており，紹介されているコードの中にも旧い初期化の仕方のままになっているものもあるので注意が必要です．C言語については，MT法の作者自身が2002年に公開した修正版 (mt19937ar) [9]か，更にSIMD化して改良された SFMT 版（[9]中のリンク）をダウンロードして利用するのがよいでしょう．

ここでは Fortran 版[10]の利用法について紹介します．なお，他の疑似乱数と同様，MT 法でも漸化式の形を決めるパラメータ $(m,n,r,u,s,t,l,\boldsymbol{a},\boldsymbol{b},\boldsymbol{c})$ を適切に選ぶ必要がありますが，これらは定数としてサンプルコード中に指定されているのでユーザが特に気にする必要はありません．
文献[10]の mt19937ar.f は初期化と発生のルーチンのみ含まれていますので，これらを利用するためのメインプログラムを書きましょう．以下にサンプルプログラム mtsample.f90を記載します (GitHubにも同じコードを公開してあります[11])．

```
program mtsample
  implicit none
  integer(kind=4), parameter :: N=624
  integer(kind=4) :: seed,i,nrnd
  real(kind=8),allocatable :: rnd(:)
  integer(kind=4) :: mti,initialized, mt(0:N-1)
  character(len=1) :: yesno
  real(kind=8) :: genrand_real1
  common /mt_state1/ mti,initialized
  common /mt_state2/ mt
!
  print *,"input seed number (0 to continue)"
  read(5,*) seed
  if(seed>0) then
    call init_genrand(seed)
  else
    call mt_initln
    open(7,file="mt_cont.dat",action="read")
    read(7,*) mti,initialized,mt
    close(7)
  end if
!
  print *,"input number to be generated"
  read(5,*) nrnd
  allocate(rnd(nrnd))
!
  do i=1,nrnd
    rnd(i)=genrand_real1() ! [0,1]-real8
  end do
!
  open(8,file="mt_rand.txt",action="write")
  write(8,'(f18.15)')rnd(:)
!
  print *,"record the MT status? (y/n)"
  read(5,*) yesno
  if(yesno=="y") then
    open(7,file="mt_cont.dat",action="write")
    write(7,*) mti,initialized,mt
    close(7)
  end if
!
 end program mtsample
```

実行するためには，[10]からダウンロードしてきたmt19937ar.fとこのmtsample.f90を合わせてコンパイルしてください．例えばgfortranなら

```
gfortan mt19937ar.f mtsample.f90
```

とすれば良いです．できあがったa.outを実行すると，

```
input seed number (0 to continue)
```

と聞かれるので，適当な自然数（4567など）を入れます．次に，

```
input number to be generated
```

と聞かれるので，例えば100と入れてみましょう．すると即座に100個の倍精度実数の[0,1]区間疑似乱数が生成され，"mt_rand.txt"に格納されます．最後に，

```
record the MT status? (y/n)
```

と聞かれるので，yと入力すると乱数列を次回の計算に継続するための情報が"mt_cont.dat"に記録されます．前回の続きから疑似乱数を発生させたい場合は，seedに0を入れて下さい．なお，mt_rand.txtとmt_cont.datは実行する度に上書きされるので注意して下さい．

ここで，「疑似乱数を継続する」というのは，例えばあるseedからスタートして1万個を一気に生成したのと，同じseedから始めて5千個の乱数を2回発生させたので同じものになるようにするということです．これは，実際に疑似乱数を使って，1回のジョブに実行時間制限のあるスパコンなどで，非常に長いシミュレーションを複数回繋いで計算する場合に必要となります．サンプルのMT法の場合は，項数624の漸化式になっているのでその状態を記録して，次に実行する時に読み直すことで継続できます．**図1**はseed=4567で100個一気に生成した場合と，50個ずつを2回継続で生成した場合の乱数の分布をプロットしたものですが，両者が完全に一致していることが確認できます．なお，mt19937ar.fにはgenrand_real1の他にも，(0,1)，[0,1)など様々な区間の乱数を生成するfunctionが定義されているので，用途に応じてrnd(i)=genrand_real1()の所を切り替えてください．

なお，サンプル中で適当に与えたseedの値の役割ですが，これは乱数の初期値を設定するものです．Tausworthe法やMT法では漸化式をスタートさせるために，数百の漸化式の初期値$\{x_i\}$を指定する必要があります．MT法の初期化では1項だけの漸化式によるシンプルな疑似乱数ルーチンを使ってその初期値を与えています（線形合同法よりは凝ったものですが）．seedの値を変えることは漸化式の初期値のセットを変えることに相当し，より具体的に言えば周期長$2^{19937}-1$の数列のどこを開始点にするかを疑似乱数で決めることになります．Tausworthe法の場合，初期値を上手く設定しないと乱数列の周期長が期待通りの長さにならない場合があると述べましたが，MT法ではビット間で情報をやり取りするために，初期値の設定と実現される周期長に依存性はありません[6]．

ところで，このサンプルプログラムで生成する数を百万など大きめに取ると，それなりに計算時間が掛かることが体感できると思います．実はこのサンプルでは，1回につき1つの疑似乱数を返すfunctionを必要な回数だけcallする形になっています．実用上は，配列rnd(nrnd)に一気にnrnd個の疑似乱数を生成できた方がコードも書きやす

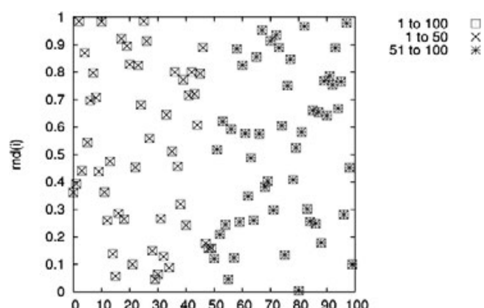

図1 MT 法で生成した乱数のサンプル．100個を１度に生成した場合と，50個ずつ２回継続して生成した場合をプロットしてあります．

いですし，効率もよさそうですよね？また，現在では大規模シミュレーションは MPI などの並列環境で実行するのが当たり前になっていますが，疑似乱数の並列化はどのようにすればよいのでしょうか？こうした効率化，並列化に関するテクニックについては次章で解説したいと思います．

2.2 物理乱数

本節においては，乱数の生成にランダムな物理現象を利用した「物理乱数」について紹介します．乱数の生成に放射線や，ダイオードの出力雑音，量子ゆらぎなどの物理現象を利用しようというアイデアは，古くからあり，日本では，遅くとも1957年には物理乱数発生器を計算機に接続し，モンテカルロ計算に物理乱数が利用されていたようです[12]．以下では，定電圧ダイオード（ツェナーダイオード）の出力信号のゆらぎ，およびレーザー光の偏光の量子ゆらぎを利用した乱数発生の原理について説明します．

●ダイオードの出力雑音による乱数発生

まず，定電圧ダイオードを利用した乱数発生器についてですが，出力信号に含まれるわずかな（数 μV 程度の）ゆらぎ（ノイズ波）をノイズ源として利用します[13]．このゆらぎを ADC（Analog to Digital Converter）の変換範囲まで増幅し，ある一定間隔の離散時刻でノイズ波のデジタル値を取得します．各時刻に得られたデジタル値を0000～1111までの16個の２進数のうちの１つに対応させることで，各時刻における４ビットの乱数（０または１の値で表現される二値乱数）が生成されます．２回のデジタル値の取得で８ビット（つまり１バイト）の乱数となり，これを繰り返すことで，必要なバイト数の二値乱数を作ることができます．Fortran や C などのプログラミング言語における単精度浮動小数点型の区間［0, 1）に含まれる一様乱数を生成す

乱数茶話

少し余談になりますが，疑似乱数は数値シミュレーションの世界で利用されるだけではありません．一般の人々が疑似乱数と係わるのは，例えば，コンピュータゲームで遊んでいる時ではないでしょうか．敵キャラクタの出現位置や行動などを，プレイヤーが飽きないようにランダムにするなどの用途に疑似乱数が使われています．ところで筆者（S.S.）が子どもの頃のコンピュータと言えば，８ビットの CPU に 32 kB のメモリという，非常に貧弱な計算資源の中でゲームを動かしていました．当然，乱数発生にメモリを割いたり複雑なアルゴリズムを採用したりする余裕はありません．当時のパソコンのメモリ（RAM）は放っておくと電気的に記録されたビット情報が減衰してしまうので，時々データを書き直す必要がありました．このタイミングをコントロールするために CPU 内部にリフレッシュカウンタというレジスタがあり，タイマのように CPU のクロックごとに値が増えていきました．当時の CPU はせいぜい数 kHz で動作していましたが，それでも人間から見れば高速にレジスタの値が変わるので，このレジスタの値が疑似乱数としてよく使われていたようです．他にも，CPU には，命令を実行するたびにその結果に応じてレジスタの特定のビットが０になったり１になったりする，フラグレジスタというものがあります．これは本来，直前に行った計算がオーバーフローしたとか，論理演算の真偽値を確認するためのものですが，このフラグレジスタの値を乱数として使うこともありました．古いゲームの攻略法などで，「乱数調整」と称して一見無意味な行動をしてからイベントを起こすと（宝箱を開けるなど）決まった結果が起こる，というようなことをたまに聞きますが，これらは乱数の代わりに使われている上に述べたようなデータの隠れた規則性を見抜いて，都合のよい結果になるように「調整」しているわけです．

今回の記事の執筆のために調べていて興味深かったのが，８ビット PC の時代から線形合同法や XOR を使った M 系列の乱数も利用されていたことを知ったことです．ただし８ビットなので周期が非常に短いですが．線形合同法を使ったゲームでも，偶奇が交互に出る性質を理解せずにくじのイベントの当たり判定に使ってしまったために，プレイヤーに癖を見抜かれてしまったという事例もあったようです．筆者のいとこは以前パチンコ機器のプログラマーをしていましたが，少ないプログラム行数で統計性のよい疑似乱数を発生させる方法はないかと相談されたことがありました．パチンコでは不正防止のために，当たりの確率が規定された通りになっていることが認可を受けるために必要な一方，プログラムに使えるメモリ量にも制約があるため疑似乱数ルーチンの選択で悩んでいたようです．

このようにゲームやギャンブルの世界では疑似乱数は広く使われていますが，人工知能が発達すると，やがては疑似乱数に潜む隠れた規則性をあっさり見抜いて，囲碁や将棋だけでなくコンピュータゲームでも人間のプレイヤーを凌駕してしまうかもしれませんね．

る場合は，例えば，4バイトの乱数を用いて得た正整数をその最大値に1を足した数で割ることで得られます．以上は，4ビットの乱数を基準に説明しましたが，一般に N ビットの乱数としても同様です．

ところで，上記の乱数発生法を実際の装置で実現化する際には，得られる二値乱数の一様性が気になるのではないでしょうか．そこで，乱数の一様性を向上させる方法について紹介します．乱数生成のスピードを落とさずに一様性を向上させる手法として，排他的論理和による補正手法が考え出されました[13]．文献[13]に従い，この手法を2ビットの乱数を例として説明します．ADCが2ビットの変換値として0～3を出力するとし，変換値 k の発生確率を p_k とします．ここで，$k=0,1,2,3$ で，$\sum_{k=0}^{3} p_k = 1$ です．あらかじめ与えた2ビットの一様乱数表から等しい確率 ϕ で補正値0～3を生成し，ADCの変換値との排他的論理和を取ります．**表1**にADCの変換値と補正値との排他的論理和を行ったときの補正後の値（上段）とその生成確率（下段）を示しました．ここで，排他的論理和は，2.1節でも使いましたが，以下のような演算（$\oplus$ と表記）であり，**表1**では，2ビットの値00～11について，ビットごとに演算して補正後の値を得ています．

$$0 \oplus 0 = 0$$
$$0 \oplus 1 = 1$$
$$1 \oplus 0 = 1$$
$$1 \oplus 1 = 0$$

例えば，変換値1と補正値3で考えると，$01 \oplus 11 = 10$ ですので，補正後の値は，2となります．**表1**を見ると，補正後の値が2となるケースは全部で4つあり，その生成確率の和は，$p_0\phi+p_1\phi+p_2\phi+p_3\phi=\phi$ となるので，補正後の二値乱数10は確率 ϕ で生成されることがわかります．他の補正後の値についても，すべて同じ確率 ϕ で生成され，このことから，生成された2ビットの乱数の一様性が向上していることが分かります．上記の説明は，簡単のために2ビットの場合で説明しましたが，2を N に一般化して，N ビットの乱数とした場合も同様な補正を行えば，生成する乱数の一様性が向上します．また，補正値の生成方法として，一様乱数表を利用する方法をここでは紹介しましたが，別の方法もあります．詳しくは，文献[13]や関連する特許情報などを参照ください．

表1　ADCの変換値と補正値との排他的論理和により与えられる補正後の値（上段）とその発生確率（下段）．例えば，変換値1と補正値3の補正後の値は，2行-4列目の上段に表示され，2です．この表は，文献[13]の表1を参考に作成しました．

		補正値			
		0	1	2	3
変換値	0	0	1	2	3
		$p_0\phi$	$p_0\phi$	$p_0\phi$	$p_0\phi$
	1	1	0	3	2
		$p_1\phi$	$p_1\phi$	$p_1\phi$	$p_1\phi$
	2	2	3	0	1
		$p_2\phi$	$p_2\phi$	$p_2\phi$	$p_2\phi$
	3	3	2	1	0
		$p_3\phi$	$p_3\phi$	$p_3\phi$	$p_3\phi$

このように乱数発生原理の概略がわかると，次に乱数発生の速度が気になるのではないかと思います．この発生速度については，近年，640 MB/s（≈5.1 Gbps）程度になっているようです．複数台の乱数発生器を用意できるのであれば，大規模並列計算機におけるシミュレーションでの利用に（必要とする乱数の数や呼び出す頻度にも依りますが）ほぼ問題の無いレベルの発生速度であると言えます．

定電圧ダイオードを用いた物理乱数発生器による乱数を利用したい場合は，例えば，統計数理研究所の乱数ライブラリー[14]における乱数取得サービスから乱数をダウンロードする方法があります．この記事を書いている時点では，東京エレクトロンデバイス製の物理乱数発生器による乱数を取得できるようです．取得できる乱数のデータ型は，区間$[-2^{31}, 2^{31}]$の32ビット符号付整数，区間$[0, 1)$の単精度浮動小数点数および倍精度浮動小数点数です．

● レーザー光の偏光の量子ゆらぎによる乱数発生

次に，光学的な装置を使った「量子乱数」と呼ばれる物理乱数発生器の原理について紹介します．量子力学的な不確定性を持つ物理過程の観測値を乱数のソースとして利用しようというアイデアは1950年代には既にありましたが[14]，初めに考えられたのは放射性物質から出てくる放射線をガイガー計測器で計測した時の，2つのパルスの発生間隔がPoisson分布の確率分布をとるという性質を利用するものでした．しかし放射性物質を線源として内蔵することの問題や，ガイガー計測器の計測速度に原理的な限界があったこと（数kbps程度），小型のレーザー発振器が実用化されたことなどから，現在では光学デバイスによる量子乱数発生器が主流になってきています．文献[15]のTable IIにいくつかの異なる手法による光学的な量子乱数発生器の一覧が載っていますが，乱数の生成速度は数Mbpsから数Gbpsに達しています．

レーザー光を使った量子乱数の例としてここでは，[16]のレーザーパルスの位相差を利用したものを紹介します．**図2**のようなレーザー光の回路を組み，発振したレーザーパルスを一度2つの光ファイバーのケーブルに分岐させ，片方の経路を長く取ることで，一方をちょうど1パルス分ずらします．その後，2つのパルスを再び合流させてから

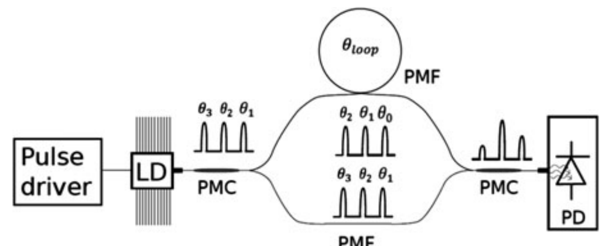

図2　レーザー光を使った量子乱数発生器の概念図（[16]のFig.2を引用）．Adapted with permission from [16] © The Optical Society.

その光の強度を計測します．レーザー光のパルスは量子力学的揺らぎにより，一つ一つ異なる位相θ_jを持った偏光になっています．したがって２つのパルスを合成した後のレーザー光の強度は，干渉によって$\cos(\theta_i-\theta_{i-1})$の依存性を持ちます．$\theta_i$と$\theta_{i-1}$が量子力学的な揺らぎから決まるため，$\theta_i$と$\theta_{i-1}$は互いに独立で一様分布する確率変数であると期待できるので，そこから観測光の強度の確率分布が計算できます．仮に観測光の強度を$x\in[-1,1]$に規格化したとすると，その確率分布は$1/\sqrt{1-x^2}$に比例すると期待されます．しかし，実際の観測値には観測のノイズやパルス幅に関係する補正項が入るので，単純に理論通りに分布するわけではなく，**図3**のようなひずんだ凹型の分布になります．この分布は，２つのパルスに何らかの相関関係があるため生じているわけではなく，上に述べたパルス間の位相差がランダムであることに起因しています．しかし，この測定値には計測誤差やレーザー光源の揺らぎなどの誤差による期待値からのずれも含まれており，これらの誤差は必ずしもランダムであるとは言えません．また，非一様な強度分布のままでは乱数として使うには適さないので，デジタル化した計測データをWhirlpool-Hash関数[17]というものを使って変換したものを乱数として使います．これは，入力されたデータをある決まった規則に従って変換し，入力データからは予測できない固定長のデータ列（Hash値）を返すもので，暗号化の技術の一つとして使われているものです．この際，ある確率分布を持つ確率変数の最低エントロピー状態という理論を基に，デジタルサンプリングした元データのビット数より低いビット数のHash値を乱数として採用することで，元のデータの情報量を一部落とす代わりに一様性の高い乱数列を得るという手法[18]を取っています．具体的には，14ビットでサンプリングした1.2×10^8個の観測データをWhirlpool-Hash関数で1.25×10^8個の７ビットのバイナリデータに変換しています．こうして得られた数列は非常によい一様性を示し，

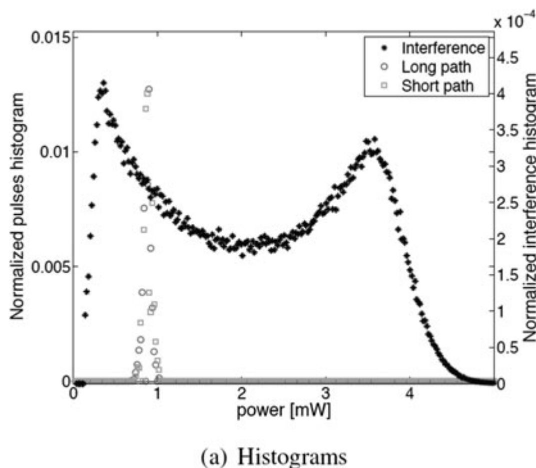

図３　短，長２つの経路を通ったそれぞれのパルス強度（○と□の印，左の縦軸）と，２つを合わせて観測した場合の強度（十印，右の縦軸）の分布．２つのレーザーパルスを合成すると干渉によって広がりを持った分布になります（[16]のFig.3(a)を引用）．Adapted with permission from [16] © The Optical Society.

NISTテストと呼ばれる一連の乱数検定テストもパスしています．このケースでの量子乱数の発生速度は理論的には40 Gbpsを超えるとされていますが，実際には，Hash関数の演算処理による時間を含めると実用上のスピードは数Mbps～Gbps程度のようです[19]．

今回紹介したレーザー光の偏光を利用した手法の他にも，レーザー光を２つにスプリットしたものを再度合わせて計測することで，観測に混じる量子力学的な真空のゆらぎを増幅して観測するという原理に基づいた光学的量子乱数発生器も開発されています[20]．この手法も信号をデジタルサンプリングした生データの情報の一部を捨てる代わりに統計的性質を向上させる手法を使って，計測データに含まれる量子的ゆらぎ以外の計測ノイズの影響を落とす後処理を使います．この原理に基づく乱数発生器がオーストラリア国立大学の研究グループによって公開運用されており[21]，乱数発生速度は5.7 Gbpsに達するそうです．インターネットを通じて生成された乱数を利用でき，様々なプログラム言語からこの乱数サーバのデータを取得するルーチンが公開されているので，関心のある方は[21]のリンクから情報をたどってみてください．

ところで比較参考のためにMT法の発生速度を見ると，2.1節のサンプルプログラムをXeonのワークステーション（クロック周波数2.5 GHz）で実行した場合，32ビット乱数を１千万個生成するのに約0.2秒，発生速度に直すと約1.6 Gbpsでした．筆者（S.S）は数年前に[16]の量子乱数発生器の試作機のテストに参加させてもらいましたが，当時はまだ乱数発生器からコンピュータのメインメモリにデータを転送するところが律速となり，理論性能ほどの速度は出ていませんでした．しかしそれでも当時のワークステーション用CPUでMT法を走らせた場合の10分の１程度の速度が出ていました．物理乱数発生器は，まだあまり世の中に出回っていませんが，[14]や[21]の乱数サーバの実績値を見ると，物理乱数の発生速度は疑似乱数に比較しうる時代になっていると言え，今後の展開が期待されるところです．

謝　辞

第2.2節における定電圧ダイオードを利用した物理乱数の発生原理の詳細についてご教示いただいた，田村義保氏（情報・システム研究機構 統計数理研究所 特任教授・名誉教授）に感謝いたします．また，第2.2節に対して，河村学思氏（自然科学研究機構 核融合科学研究所 助教）より，貴重なコメントをいただきました．ここに感謝申し上げます．

参 考 文 献

［1］津田孝夫：モンテカルロ法とシミュレーション（三訂版）（培風館，1995）．
［2］伏見正則，逆瀬川浩孝（監訳）：モンテカルロ法ハンドブック（朝倉書店，2014）．
［3］松本 眞：有限体の擬似乱数への応用，
http://www.math.sci.hiroshima-u.ac.jp/~m-mat/

TOKYOHOMEPAGE/TEACH/1011.pdf
[4] 統計数理研究所 乱数ライブラリー 乱数について http://random.ism.ac.jp/info01.html
[5] 宮村 修，牧野 純：ベクトル化乱数発生プログラム，大阪大学大型計算機センターニュース **75**, 39 (1989). http://hdl.handle.net/11094/65855
[6] 松本 眞，栗田良春：Twisted GFSR：新しい乱数発生法，京都大学数理解析研究所講究録 **85**, 86 (1993). http://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/0850-08.pdf
[7] M. Matsumoto and T. Nishimura, ACM T MODEL COMPUT S **8**, 3 (1998).
[8] http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/VERSIONS/versions.html
[9] http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/mt19937ar.html
[10] http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/VERSIONS/FORTRAN/mt19937ar.f
[11] https://github.com/satakeshinsuke/random_number
[12] 統計数理研究所 統計科学技術センター 計算機の歴史 https://www.ism.ac.jp/computer_system/jpn/outline/hist.html
[13] 田村義保 他：日本統計学会誌 **35**, 201 (2006).
[14] 統計数理研究所 乱数ライブラリー http://random.ism.ac.jp/
[15] Miguel Herrero-Collantes *et al.*, Rev. Mod. Phys. **89**, 015004 (2017).
[16] C. Abellán *et al.*, Optics Express **22**, 1645 (2014).
[17] P.S.L.M. Barreto and V. Rijmen, *The Whirlpool hashing function*, https://www.researchgate.net/publication/228610491
[18] N. Nisan and A. Ta-Shma, J. Comput. System Sci. **58**, 148 (1999).
[19] Carlos Abellán *et al.*, Optica **3**, 989 (2016).
[20] T. Symul *et al.*, Appl. Phys. Lett. **98**, 231103 (2011).
[21] The ANU Quantum Random Numbers Server, https://qrng.anu.edu.au/

佐竹真介（さたけ しんすけ）

自然科学研究機構 核融合科学研究所 ヘリカル研究部 核融合理論シミュレーション研究系 准教授，2003年総合研究大学院大学 博士（学術）．
モンテカルロ法を使った3次元磁場配位中の新古典輸送現象，新古典粘性のシミュレーションや最適化配位の研究が主なテーマ．乱数については深い思い入れがありますが，特にギャンブル好きというわけではありません．

菅野龍太郎（かんの りゅうたろう）

自然科学研究機構 核融合科学研究所 ヘリカル研究部 核融合理論シミュレーション研究系 准教授．
乱数を用いた確率論的な計算手法であるモンテカルロ法全般に興味があります．最近の研究では，プラズマの衝突輸送現象に対するモンテカルロ法に基づいたドリフト運動論シミュレーションを行っています．