J. Plasma Fusion Res. Vol.96, No.6 (2020) 290-291

講座

シミュレーションのための乱数入門

Introduction to Random Numbers for Simulations

1．はじめに

1. Introduction

宇佐見俊介
USAMI Shunsuke
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）

非平衡系であるプラズマ中の現象を解析するには，コンピュータによるミュレーションが非常に有力な手段となっています．シミュレーションには様々なモデルや計算手法がありますが[1]，そのうち，いわゆるモンテカルロ法に基づいた輸送シミュレーションでは，乱数を用いた計算を繰り返すことから，大量の乱数データを必要とします．他にも粒子シミュレーションでは，粒子の初期位置や速度分布（Maxwell分布など）を与える際に，乱数が用いられるのが一般的です．また，乱数は，これらのような科学技術計算だけに留まらず，インターネットショップでの情報通信の暗号化，二段階認証におけるワンタイムパスワード生成などにも用いられ[2]，現代社会で必要不可欠なツールとなっています．

本講座では，そのような乱数について学んでいきます．講座を読み始めるにあたって，最初に思い浮かぶ疑問は，「そもそも乱数とは何か？」ではないでしょうか．乱数が，どのように定義されているのか調べてみると，大辞林（第三版）では，「乱数とは出現する値に規則性のない数」とあります．この「規則性がない」という定義から，乱数が満たすべき性質が導かれるように思いますが，数学的に厳密に定義しようとすると簡単ではありません[3]．そこで，まずは，以下のような例を用いて，乱数の性質の一端を覗いてみましょう．0と1だけからなる数列として，

(a) 0，1，0，1，0，1，0，1，0，1，0，1，0，1，0，1
(b) 1，0，1，1，1，0，1，0，0，0，0，1，0，0，1，0

の2種類を挙げてみます．(a)は，明らかに0と1が順番に出現していて，その規則性ゆえに乱数には見えません．一方，(b)は，私自身がコインを投げて表を1，裏を0として記録した結果ですので，この数列は乱数に近いと言えるでしょう（＊）．一見，(b)では，同じ値が連続することが頻繁にあって作為的な感じがしますが，乱数にはこのような並び方が起こり得ます．0と1が規則性なく出現するということから，同じ値が連続で出現する率はどのくらいかという統計的性質が導けます．そのため，逆に言えば一見乱数に見える数列も，乱数が満たすべき統計的性質から外れていると，乱数とは言えないでしょう．ただし，厳密には，無限の数列がないと，ある数列が乱数かどうかは判定できません．実は(a)の場合も，無限に続く乱数列中に含まれうることに留意する必要があります．ならば，科学技術計算で(a)も乱数として使ってよいのか，という疑問が出てきますが，答えはNOです．科学技術計算では，有限の数列しか利用しないので，その有限の中で乱数としてふさわしい統計的性質を満たしていることが望ましい，つまり(b)が望ましいということになります．乱数発生法において，この点は，議論のあるところではないでしょうか．

（＊）コインの動きは古典力学に従いますので，表裏どちらが出るのかは決定論的であり，ランダムではないという反論もあるかもしれません．一方，コインの動きは，条件のわずかな違い（コインを投げる速度，机の凸凹など）に非常に敏感なので，結果は事実上予測できません．このことから，表裏どちらが出るのかはランダムと言ってよいでしょう．

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

author's e-mail: usami.shunsuke@nifs.ac.jp

上記で述べたような性質は，コンピュータで乱数を用いる際に，極めて重要な要素です．一般的なコンピュータは真の乱数を作れず，様々なアルゴリズムにより「疑似乱数」を発生させています．コンピュータ（すなわち人間）が作り出す疑似乱数の値は，一見ランダムに見えますが，決定論的な演算（数式）によって算出されたものですので，原理的に周期性や，偏りを持っています．そのため，乱数の性質を満たしているかどうかが大きなチェック対象となります．

コンピュータシミュレーション分野では，黎明期から現在まで，より長周期で，偏りが少なく，かつ高速に疑似乱数を生成できるアルゴリズムが開発されてきました[4]．現在では，一般のユーザはプログラミング言語に標準で備えられている，あるいは数値ライブラリで提供される乱数発生ルーチンをブラックボックスのように利用することができます．しかし，その疑似乱数がどのように生成されていて，その品質がどのようなものか，また間違った利用方法をしていないか，ということを一度振り返って考えてみることは非常に重要なことです．

本講座では，実際に自ら，乱数を使ったシミュレーションモデルを開発して実行している研究者が，乱数が満たすべき条件，乱数発生方法のいくつかのバリエーション，検定方法，高速化，応用例について，詳しく説明します．ただし，著者らは乱数そのものの専門家ではないことから，数学的に厳密であることを追求しすぎることは避け，シミュレーション研究者としての実践的な説明をすることを心がけました．

掲載内容は，以下のように予定しています．

（第 1 回目）

第 2 章では，コンピュータにおいて，疑似乱数列を生成する様々な手法を主に周期長という観点から紹介し，実際の使い方も説明します．さらに，コンピュータで真の乱数を発生させる「物理乱数」に関しても触れることにします．

（第 2 回目）

第 3 章では，第 2 章で紹介した様々な乱数発生法を高速化し，並列実行のプログラム中で用いる方法について説明します．

第 4 章では，乱数の品質と検定法を解説します．乱数らしさについて，周期長や多次元均等分布性といった数学的な観点から説明し，乱数が満たすべき統計的性質を利用して，乱数の検定方法をいくつか紹介します．

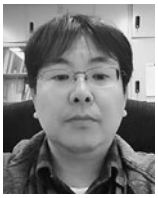
（第 3 回目）

最後に第 5 章では，乱数の変換例・利用例を示します．まずは，様々な乱数をいかに生成するか，つまり「乱数の変換」について論じ，一様乱数から任意の確率分布に従う乱数を作る方法を説明します．また，基礎的なモンテカルロシミュレーションに加えて，これまであまり焦点の当たっていなかった，輸送方程式（非線形偏微分方程式）の解を求める過程を大域的最適化問題に帰着させるという解法を紹介し，乱数がどのようにシミュレーションにおいて利用されているか具体的に解説します．

本講座が，これからシミュレーションで乱数を用いようとする研究者，学生，さらには，乱数自体に興味のある人々にとって入門的な役割を果たせたなら幸いです．

参 考 文 献

［1］プラズマ・核融合学会（編）：プラズマシミュレーション（京都大学学術出版会，2018）．
［2］「ランダムと乱数の奇妙な世界」Newton 2018 年 9 月号 pp.70-83.
［3］D.E. Knuth, *The Art of Computer Programing Volume 2 Seminumerical Algorithms, Third Edition* (Addison-Wesley, 1997).
［4］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）．



宇佐見俊介（うさみしゅんすけ）

自然科学研究機構 核融合科学研究所 ヘリカル研究部基礎物理シミュレーション研究系 准教授．

粒子シミュレーションを用いて，プラズマの様々な複雑な現象を調べていて，現在の主な研究テーマは磁気リコネクションです．SF が好きなためか，現実世界はコンピュータ上のシミュレーションでは？と密かに妄想しています．今回の講座をとりまとめている際に，世界がシミュレーションなら，量子ゆらぎに基づく乱数も実は疑似乱数で，何らかの規則性があったりしないか？とさらに妄想を膨らませています．

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

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

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

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．

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

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 $\boldsymbol{xA}$ 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．

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

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$\{\boldsymbol{x}_i\}$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

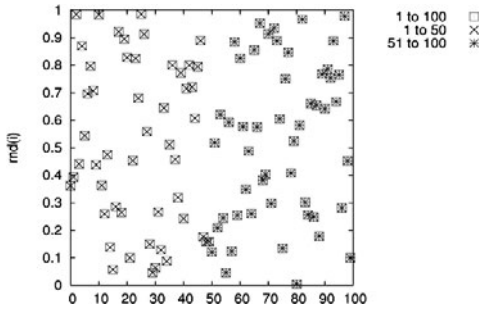


図1　MT法で生成した乱数のサンプル．100個を1度に生成した場合と，50個ずつ2回継続して生成した場合をプロットしてあります．

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．

$$
\begin{aligned}
0 \oplus 0 &= 0\\
0 \oplus 1 &= 1\\
1 \oplus 0 &= 1\\
1 \oplus 1 &= 0
\end{aligned}
$$

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

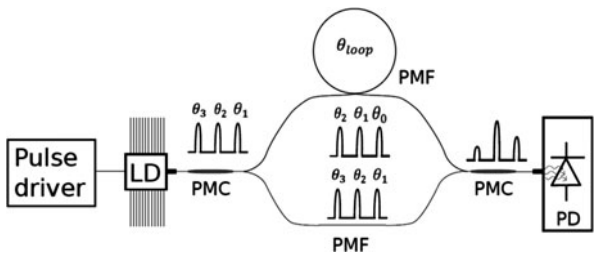
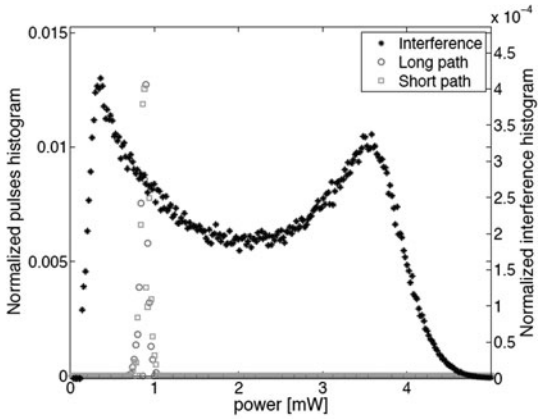


図2　レーザー光を使った量子乱数発生器の概念図（[16]のFig.2を引用）．Adapted with permission from [16] © The Optical Society.

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，



(a) Histograms

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．

J. Plasma Fusion Res. Vol.96, No.7 (2020) 367-371

講座 シミュレーションのための乱数入門

3．乱数発生の高速化と並列化技法

3. Techniques of Acceleration and Parallelizing the Random Number Generation

佐竹真介
SATAKE Shinsuke
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）

現代のスーパーコンピュータを用いた大規模シミュレーションは，多数の計算ノードを用いた並列計算が標準的になっています．本章では，MPI分散並列プログラムにおける疑似乱数の並列発生方法について，数値計算ライブラリKMATH_RANDOMを例に解説します．また，乱数発生の高速化チューニングの実例や，物理乱数発生器を使う場合の高速化，並列化についてのアイデアについても紹介します．

Keywords:
pseudo random number, physical random number, parallel computing

・はじめに

本章では，大規模シミュレーションでの乱数の利用を想定し，乱数生成を効率的に行うためのプログラミング上の技法について解説します．現代の大規模シミュレーションでは特に，MPIやOpenMPといった分散並列，スレッド並列を駆使したプログラムを利用することが一般的になっていますから，そのような並列化プログラムの中で乱数を利用する上での留意点や工夫について紹介したいと思います．なお，本章では疑似乱数発生法として第2章で紹介したMersenne-Twister（メルセンヌ-ツイスター）法（MT法）を 対象として説明しますが，物理乱数器の利用を想定した場合の並列化や高速化に関するアイデアも紹介したいと思います．

まず乱数の並列化の説明に先立って，MPIによる分散並列と，OpenMP（あるいはコンパイラの自動並列化機能による）スレッド並列の違いを明確にしておきましょう．分散並列は，計算を実行するCPUがそれぞれ物理的，あるいは論理的に分割された異なるメモリ領域に配列や変数を記憶し，シミュレーションに必要な計算を各CPUが分担して行い，必要に応じて他のCPUと明示的な通信命令によってデータをやり取りする並列計算です．一方，スレッド並列は，CPUの中にある複数のコアが，同一のメモリ上のデータを使った演算を分担するやり方で，基本的にはFortranのdo loopやC言語のfor loopなどの繰り返し計算を分割してそれぞれのコアが並列計算する方法です．この両者を組み合わせたハイブリッド並列も大規模シミュレーションでは一般的に行われています．以下，分散並列における並列化の区切りを「プロセス」，スレッド並列におけるそれを「スレッド」と呼んで区別します．

さて，並列シミュレーションにおいて乱数を利用する上で留意しなければならないことは，プログラムの詳細にも依存しますが，基本的に分散並列の各プロセスは異なる乱数列を利用しなければならないということです．例えば，分子動力学（MD）シミュレーションでタンパク質の折り畳み構造を調べる計算を，10プロセスの分散並列で行うとしましょう．異なる初期配置を持ったタンパク質分子を，乱数を利用して多数用意する際に，それぞれのプロセスで同じ疑似乱数発生法，例えばMT法で，第2章で説明したseedの値も同じに取ってしまったら，10個のプロセスで全く同じ計算を開始してしまうことになり，並列化の意味がありません．そんなことはわかりきったことであり，seedの値をプロセス毎に変えればよい，と考える人もいるかも知れませんが，それはお勧めできません．疑似乱数のseedの値は，利用する乱数列の開始点を一周期のどこに取るかを決めるものだと2.1節で説明しましたが，例えばseedの値を10個のプロセスででたらめに（人手で決める，時計を使う，他の疑似乱数を使うなど）与えても，それらの部分列の開始点が互いに十分離れているという保証は全くありません．seedの選び方がたまたま悪くて2つのプロセスの乱数列の開始点が数個しかずれてなかった，ということも考えづらいですが，起こりえない話ではありません．各プロセスが参照する乱数の部分列が確実に重複しないように選ぶことは，プロセス間で擬似乱数によるおかしな相関関係が生じる懸念を排除するために必要なことです．それでは，具体的にどのように乱数発生を並列していけばよいのか見ていきましょう．

・Mersenne-Twister法による乱数生成の並列化

分散並列計算におけるMT法の乱数発生の並列化については，以前にもプラズマ・核融合学会誌の講座で紹介させていただいたことがあります[1]．その時に紹介したのは，

National Institute for Fusion Science, Toki, GIFU 509-5292, Japan
author's e-mail:satake@nifs.ac.jp

Dynamic Creator と呼ばれる，MT 法の考案者の松本眞氏ら自身が開発した技法[2]で，ソースコードも公開されています[3]．Dynamic Creator では，第 2 章で説明した，MT 法の漸化式の形を決めるパラメータのうち，ベクトル $(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c})$ を変えることで，特性多項式が互いに素で，最大周期長がある Mersenne 素数 p に対し 2^p-1 となり，かつ高次元に均等分布する疑似乱数の漸化式の型を多数用意する，という方法を取ります．数学的に，特性多項式が互いに素な漸化式同士は相関を持たないとされているため，Dynamic Creator は言わば互いに独立な乱数表を複数用意することに相当します．ただし，特性多項式が互いに素なパラメータ $(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c})$ の探索はトライ＆エラー的になされるため，数千並列の並列計算のための準備には非常に長い時間がかかります（筆者が十数年前に1024個用意するのに，ワークステーション 1 台を 1 か月以上稼働し続ける必要がありました）．

そこで，超多並列計算のための実用的な MT 法の並列化として今回紹介したいのは，Jump Method と呼ばれるものです[4]．MT 法の特徴である長大な周期長を活かし，1 つの疑似乱数列の中の，遠く離れた複数の点を並列計算で乱数を利用する開始点として明示的に指定する方法です．これは，先に述べたような seed の値を単に変化させるのとは異なり，開始点間の距離を $N \gg 1$ 個ずつ飛ばしに指定することができます．N 個飛ばしした後の漸化式の内部状態を並列プロセス数 P だけ事前に用意するには，原理的には N 個乱数を発生させては内部状態を記録する（第 2 章のサンプルプログラムにおける"mt_rand.txt"のデータに相当）という操作を P 回繰り返せば可能です．ですが，P 個の乱数列の部分区間が重ならないように N を十分大きく（100兆など）取り，かつ P が数千となると，実用的な方法とは言えません．しかし，Jump Method に使われる SIMD-oriented Fast Mersenne Twister（SFMT）という新しいタイプの MT 法[5]では，[6]に示される原理に基づいて N 個飛ばし後の MT 法の漸化式の内部状態に一気にジャンプできる計算ルーチンが用意されているので，それを利用することで乱数列の遠く離れた先の開始点から始めるための内部状態を高速に取得できます．SFMT は名前から推察できるように，高速化チューニングされた乱数発生ルーチンです．従って Jump Method の利用に限らず，逐次的に MT 法で疑似乱数を発生させる用途としても，従来のバージョンよりも SFMT を用いた方が高速実行が期待できます．

さて，実際の利用法についてですが，ここでは[5]のオリジナルのソースではなく，C，C++，Fortran90での実装が用意されている，理化学研究所の大規模並列数値計算技術研究チームが整備・公開しているライブラリ"KMATH_RANDOM"[7]をベースに紹介したいと思います．このライブラリの利用許諾についてはマニュアルの記載事項を参照してください．まず前準備として，NTL という数値計算ライブラリを別途ダウンロード，インストールする必要がありますが，[8]に説明があるので詳細は省きます．次に，KMATH_RANDOM を展開して，random/Makefile.machine というファイルを編集します．必要なのは，利用するコンパイラとコンパイルオプションの選択，NTL をインストールしたディレクトリの指定です．random/arch/以下にいくつかのコンピュータにおけるサンプルがあるので，それを参考にご自身の環境に合わせて設定します．なお，コンパイルするには MPI 環境も必須です．また，MT 法の周期長 2^p-1 はここでコンパイルオプション〝-DDSFMT_MEXP=p〟で指定します．make コマンドを実行し，コンパイルが完了すると，random ディレクトリ下の c/，c++/，f90/ 以下に libkm_random.a というそれぞれの言語用のライブラリができるので，自分のプログラムで利用するにはこれをリンクしてコンパイルすることになります．

次に，ジャンプファイルという N 個飛ばしの疑似乱数列の開始点の状態を記録したファイルを生成します．まず，make ptool を実行すると，random/ptool/に km_rand_gen_jump ができるので，これを実行します．

（例）km_rand_gen_jump -seed S1 S2 -max_ranks R -rand_range M

ここで，-max_ranks の値 R は想定される並列計算の最大プロセス数，-rand_range の値 M はジャンプの間隔 $N=2^M$ の指数部分を指定します．-seed の値 S1，S2 は，いわゆる通常の意味での seed の値を S1，S1＋1，…，S2 と変えたものを（S2－S1＋1）個生成するということです．これは N 個飛ばしで R 個用意する開始点全体の起点が異なるジャンプファイルを複数作成することを意味します．通常，1 つのシミュレーションに使うジャンプファイルはどれか 1 つを選べばよいですが，異なるジャンプファイルは，乱数を使ったシミュレーションを複数回行って統計平均を取りたい場合などに利用します．試しに S1＝1，S2＝2，R＝200，M＝100で実行すると，jump/ディレクトリに file_00001，00002 の 2 つのジャンプファイルのセットができます．これで，MT 法の疑似乱数に対し，2^{100} 飛ばしの開始点の情報が200個× 2 系列作られました．この計算は例えば最近の Xeon CPU 上では数秒で終わってしまいます．これは逐次計算で $2^{100}\times 200$ 個× 2 回，通常の MT 法で乱数を発生させるのとは比較にならないほど速いです．

これで前準備は完了です．次にこのジャンプファイルを使って実際に並列計算で SFMT を利用する方法を見ていきましょう．第 2 章と同様に，サンプルプログラム rand_test.f90 と km_rand_wrapper.f90 を GitHub に挙げておきます[8]．ifort コンパイラを利用する場合のコンパイル用 makefile のサンプルが用意してあるので，ご自分の環境に合わせて修正して make し，実行プログラム kmrand_time を作成してください．実行方法ですが，このサンプルプログラムは MPI とスレッド並列のハイブリッドコードになっているので，環境変数 OMP_NUM_THREADS にスレッド並列数を設定し，また KMATH_RAND_JUMP_FILE_PATH に先ほど作成した file_00001 等のジャンプファイルが格納されているディレクトリを絶対 path で設定します．そして，〝mpirun -np N kmrand_time〟のような MPI 実行コマンドで計算を開始してください．ここで N は試したい MPI 並列数（N はジャンプファイル作成時の-max_ranks の値 R 以下）です．

このテストプログラムでは，rand_test.f90 の冒頭に指定された，1 プロセス当たり ntest 個の乱数を，ntime 回発生させてその平均時間を計測します．そして，そのテストを nseed 個の異なるジャンプファイルに対して繰り返します．計測結果は"log.txt"に書き出されます．rand_test.f90 の主要部分を抜粋すると以下のようになっています．

```
----------------------------------------
call MPI_INIT(IERR)
call MPI_COMM_SIZE(MPI_COMM_WORLD,nmpi,ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
call KMATH_Random_Init(handle1,MPI_COMM_WORLD, ierr)

do k=1,nseed
   call KMATH_Random_Seed(handle1, k , ierr)
  do j=1,ntime
      call grnd_dsfmt(rnd1, ntest)
  end do
end do

call KMATH_Random_Finalize(handle1, ierr)
call MPI_Finalize(IERR)
--------------------------------
```

また，上のコードの流れは，以下のようになっています．

1．MPI_INIT で MPI 並列計算を起動
2．KMATH_Random_Init で KMATH_RANDOM の利用準備
3．KMATH_Random_Seed で k 番目のジャンプファイルを読み込む
4．grnd_dsfmt(rnd1, ntest)で配列 rnd1 に ntest 個の乱数を発生させる
5．KMATH_Random_Finalize で KMATH_RANDOM の利用終了
6．MPI_Finalize で MPI 並列計算の終了

このプログラムでは nseed 個のジャンプファイルについて，各プロセスで発生した乱数の最初の ntest 個を，"rand_s.txt"に全プロセス分をまとめて書き出します（全部を出力すると膨大になるので 1/100 に縮めてあります）．プロセス毎，seed 毎に異なる乱数列が生成されることが確認できると思います．

・KMATH_RANDOM の高速化チューニング

さて，このサンプルプログラムで発生速度が計測できるので，次に高速化について考えましょう．乱数の発生は grnd_dsfmt というサブルーチンで行っていますが，これは km_rand_wrapper.f90 の中に定義されています．実は，KMATH_RANDOM が用意する乱数発生ルーチン KMATH_Random_Vector は，高速化のために区間 (1, 2] の 8 バイト実数を，386個以上の偶数個発生するという少し変わった作りになっています．なので，1 回の必要な発生数が少ない場合や，奇数の場合の対処法が必要です．また，通常シミュレーションで使われる乱数は (0, 1] 区間に発生させた一様乱数を利用することが前提になっていることが多いですから，KMATH_Random_Vector で発生させた乱数を (0, 1] 区間に変換する操作が必要です．スレッド並列が有効になっている場合，ある程度の量の乱数列に対する変換をスレッド並列で一気に行った方が効率が良さそうです．更に，シミュレーションの中で不定長の乱数を発生させるために KMATH_Random_Vector を高頻度に呼ぶより，ある程度の大きさの，固定長 nbuff の乱数列を一気に生成し，必要な分だけ切り崩して使っていき，足りなくなったらまた一気に作る，というやり方にします．井戸水を使う人は，使うたびに必要な量を井戸まで汲みに行くのではなく，一定量水瓶にため込んでおいてそこから必要な量だけ柄杓ですくって使うのと同じように，ある一定量の乱数を貯めこんでから使う方法を，私は「水瓶方式」と呼んでいます．そのような一連の作業をしているのが grnd_dsfmt になります．

ここで，最も効率が良くなる nbuff を選ぶことで高速化につながりますが，その値は計算機の環境によります．**図 1** は，IFERC-CSC のスーパーコンピュータ JFRS1 上で，8 MPI×5 スレッド並列で kmrand_time を様々な nbuff と ntest に対して実行した際の，乱数100万個当たりの平均発生時間を表しています．ntest が大きくても小さくても，nbuff=64000 の時に最も乱数発生時間が短くなりました．この例では，チューニングの効果はあまり大きくありませんが，他の計算機環境では，特に ntest が小さい時に nbuff の選び方で数十%の速度の差が出ることもありました．なおこの水瓶方式ですが，grnd_dsfmt は MPI の各プロセスで独立に動くので，それぞれのプロセスで乱数の発生数やタイミングがずれていても問題ありません（並列化乱数発生法とは当然そうあるべきものですが）．また，2.2節で紹介したような物理乱数を利用する際も，物理乱数のデバイスにプログラムからアクセスして乱数データを取得するオーバーヘッドタイムを考慮すると，必要に応じて乱数を取得するよりも，水瓶方式を用いてある程度の量の乱数を一気に取得する方が効率がよいと考えられます．

・物理乱数発生器を利用した並列化

さて物理乱数の話が出たので，次に分散並列計算を行い

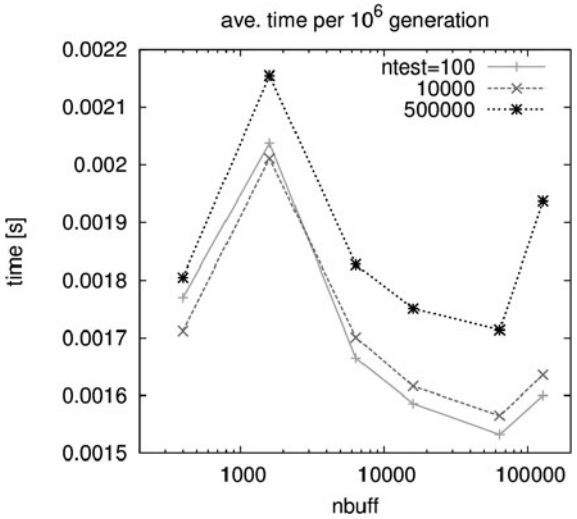


図 1　水瓶方式でバッファの大きさ nbuff を変えた場合の，乱数の発生時間の変化（百万個当たり）．nbuff = 64000 で極小値となります．

たいが物理乱数発生器が1つしかない場合，乱数の並列化をどうするか考えてみましょう．**図2**のように，計算の1ステップがCalc．A，B，C，…といったルーチンからなり，Calc.Bにおいて乱数が必要である場合を想定します．この時，例えば量子乱数発生器（QRNG）がrank0のMPIプロセスが走るノードにのみあるとします．考えられる並列化の方法としては，rank0は物理乱数の取得と，それを他のプロセスに分配することに専念し，他のプロセスが計算を担当するというやり方です．この際，「1ステップ当たり必要とされる乱数の最大個数（プロセス毎に異なる場合も含め）」×「計算に従事するプロセス数」の乱数をrank0のプロセスが乱数発生器から取得し，乱数が必要になるCalc.Bの手前のCalc.Aルーチンの間にMPI_ISCATTER（あるいはISEND/IRECVの組み合わせ）（＊）による非同期通信によって各プロセスに乱数を配信します．このやり方では，乱数を利用しないルーチンにかかる時間が，必要な乱数の発生にかかる時間に比べて長ければ，ほぼ待ち時間なしで並列計算が進められるため，計算の高速化にもつながります．各プロセスでは受け取った乱数を1ステップで全部利用しない場合もありますが，コードの実装を単純にするために，利用しなかった乱数は捨てて，次のステップで新しい乱数をrank0から受けとる形にします．この方法を私は「源泉かけ流し方式」と名付けました．温泉で源泉から汲んだお湯を，下流の各浴槽の使用量にかかわらず常に溢れるぐらい十分な量を流し込み続け，使わなかったお湯は再利用せずに捨ててしまう方式をイメージしたものです．Master-Slave形式の並列計算（パラメータ並列計算で，Masterプロセスが計算全体の管理を行い，手の空いたSlaveプロセスに次の計算パラメータを順次伝える形式）で初期条件に乱数が必要な場合にも，この源泉かけ流し方式でSlaveプロセスのどれかが計算を終えるまでに，Masterプロセスが次に使う乱数をため込んで準備しておく，という形での利用が可能です．また，乱数発生法が物理乱数に限定されるわけでもなく，疑似乱数を使った並列計算をしたいが，先に説明したJump Methodを準備するのが面倒という場合にも，一つのMT法で発生させた疑似乱数列を源泉かけ流し方式で各プロセスに分配して使うことも可能です．

次にMPI×OpenMPのハイブリッド並列の場合を考えます．もし**図2**のCalc.Aの計算時間が，QRNGで全MPIプロセスが利用する乱数を発生する時間より十分長い場合，rank0を乱数生成だけに専念させるのはCPUコアを無駄に遊ばせていることになりますので，rank0には計算にも従事してもらいたいです．そこで，Calc.Aの並列実行の仕方を**図3**のように変更してみましょう．この計算のプログラムのイメージは以下のようになります（GitHub[8]にもgensen_hybrid_img.f90として上げてあります）．

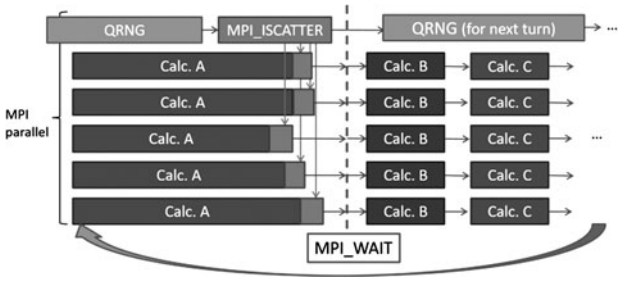


図2　MPI並列プログラムで量子乱数発生器（QRNG）が1つしかない場合の「源泉かけ流し方式」におけるプログラムの流れの模式図．rank0のプロセスは乱数生成に専念し，Calc.Aが終わった他のMPIプロセスは乱数を受信し，乱数を利用するCalc.Bに備えます．

```
   integer :: n,nmpi
   real(kind=8) :: rand(n*nmpi), rand_local(n)
!$OMP PARALLEL
!$OMP MASTER ! only master thread does the follow-
ing process
   if (myrank==0) then !assume MPI rank 0 has the
   QRNG board
      call QRNG_GEN(rand,n*nmpi) !generate random
      numbers
   end if
 call MPI_SCATTER(rand,n,mpi_real8,rand_local,n, &
 &              mpi_real8,0,mpi_comm_world,ierr)
     !scatter random numbers ( n per each MPI rank)
!$ OMP END MASTER
!$ OMP DO schedule (dynamic)     !thread parallel
   do j=1,m
      …  ! (Calculation A)
   end do
!$ OMP END DO
!$ OMP END PARALLEL
!... Both communication and calculation A are com-
pleted here.
   call sub_calc_B (n, a, b, c, ●●●, rand_local)
     ! Calc B which uses the random numbers
   call sub_calc_C (.....)
  ●●●
```

各MPIプロセスには，0〜k-1のk個のスレッドがあります．このうちの0番スレッドがMasterスレッドであり，Masterスレッドだけが，$OMP MASTER 〜 END MASTERの区間を実行します．Rank0のMPIプロセスのMasterスレッドは乱数を発生器から取得し，MPI_SCATTERでn個ずつ各プロセスに分配します．一方，Rank0以外のMPIプロセスのMasterスレッドは，乱数の受信に携わります．その間，全てのMPIプロセスにおいて，Masterスレッド以外の

（＊）**図2**では簡単のためMPI_ISCATTERを使っていますが，これではrank0自身にも乱数を分配することになります．この無駄を省くには，総プロセス数をnmpi，1プロセス当たりに分配する乱数の数をnとした場合，まずn×(nmpi−1)個の物理乱数をrank0で生成し，それを配列rand(n*nmpi)のn+1番目以降に保存します．配列randの先頭のn個の値は与えなくてよいです．そして，この配列randをMPI_ISCATTERで配信すれば，rank0自身には無意味なデータが渡り，rank1以降に生成した乱数がn個ずつ分配されます．

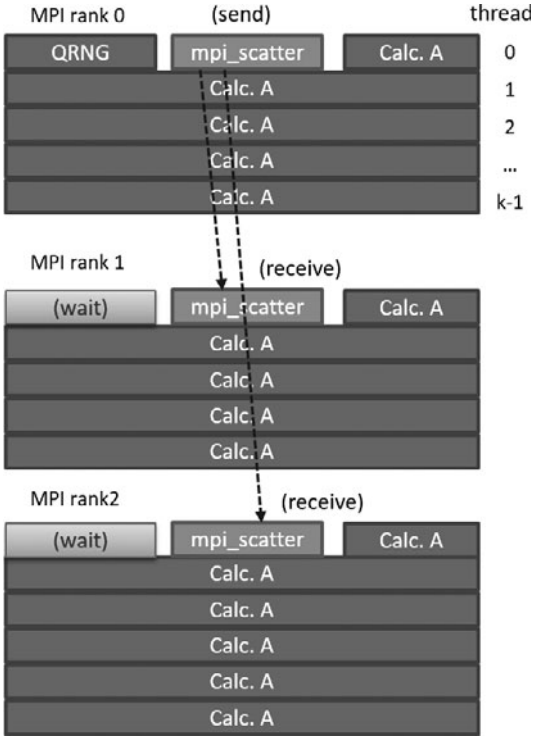


図 3 MPI-OpenMP ハイブリッドコードにおける源泉かけ流し方式の模式図. 各プロセスの Master スレッドは, 乱数の送受信が終わったところから Calc.A のスレッド並列計算に途中参加します.

1〜k-1 番スレッドは，Calc．A の Do loop を並列実行し始めています．$OMP DO に SCHEDULE (dynamic) が指定されているため，Master スレッドが乱数の生成と分配を終えて，まだ他のスレッドが Calc.A を終えていなければ，**図 3** に示すように Master スレッドは途中から Calc．A の並列計算に参加することになります．このようにすれば，源泉かけ流し方式でも乱数発生器を持つプロセスも計算に参加し，かつ乱数の分配の MPI 通信時間を Calc.A の演算に隠ぺいすることが可能です．もちろん，この方式が効率よく稼働するには MPI 並列数とスレッド並列数を適切に選び，Calc.A にかかる時間より乱数生成時間が短くなるようにしなければならないので，どんなプログラム・計算環境にも使える万能な方法ではないことに留意してください．

・KMATH_RANDOM における継続計算の方法

最後に，KMATH_RANDOM を使った計算を複数回継続する方法を説明します．第 2 章でもみたように，1 回のジョブの最後に MT 法の内部状態をファイルに書き出し，継続ジョブの最初にそれを読み直せばよいのですが，MPI 並列プログラムではそれを各プロセスが行う必要があります．更に，grnd_dsfmt のように水瓶方式を使っている場合，「水瓶」の中身も保存し，継続計算に引き継がなければなりません．そうでなければ，例えば100ステップを 3 回つないだ乱数を使った計算と，150ステップを 2 回つないだ計算とで途中から利用する乱数がずれてしまいます．

それでは，サンプルプログラムを使って，継続の仕方を具体的に説明していきましょう．km_rand_wrapper.f90 には，MT 法の内部状態と水瓶方式のバッファの中身を save，load するためのサブルーチン save(load)_grnd_dsfmt が用意してあります．その利用法のサンプル cont_test.f90 が GitHuB に上げてありますのでご参照ください [9]．はじめに，input_cont.txt 中の njob を 0 にして kmrand_cont を MPI 並列で実行すると，インデックスが iseed のジャンプファイルを使って新規に ntest 個の疑似乱数を各 MPI プロセスが生成し，"out_r**R_s**I_i000" に書き出します．ここで **R は MPI のランク，**I は選択したジャンプファイルの seed 番号です．また，乱数の継続用のファイル "rand_cont_s**I_i000"，"grnd_buff_ s**I_i000" も作られます．これらはそれぞれ，MT 法の内部状態と水瓶方式のバッファの状態を全 MPI ランク分まとめて書き出したものになっています．次に，njob=1 として再度 kmrand_cont を実行すると，上の継続用ファイル 2 つを読み込み，疑似乱数列の続きを ntest 個作ります．以降，njob=2,3,…とするごとに，"rand_cont_s**I_i**N"，"grnd_buff_ s**I_i**N" (N=njob-1) のデータを読んで疑似乱数列を一つ前のジョブの続きから生成します．ntest の数をいろいろ変えて実行し，各 MPI ランクが作る疑似乱数がそれぞれ異なる系列の乱数列で，かつ各々はジョブの区切りを跨いで連続した乱数列になることを確認してください．

駆け足になりましたが，これで KMATH_RANDOM ライブラリを利用した，並列化 Mersenne-Twister 法の利用と高速化と，物理乱数発生器を並列プログラムで利用する方法の解説はおしまいです．本章の内容とサンプルプログラムが，乱数を利用した多並列シミュレーションコードを書く方々の一助になれば幸いです．

参 考 文 献

［1］講座「核融合プラズマシミュレーションの技法　5．粒子シミュレーションのコーディング技法」プラズマ・核融合学会誌 **89**, 245 (2013).
［2］M. Matsumoto and T. Nishimura, "Dynamic Creation of Pseudorandom Number Generators", Monte Carlo and Quasi-Monte Carlo Methods 1998, Springer, 2000, pp. 56-69.
［3］http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/DC/dc.html
［4］http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/JUMP/index-jp.html
［5］http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/index-jp.html
［6］H. Haramoto *et al.*, "A Fast Jump Ahead Algorithm for Linear Recurrences in a Polynomial Space", Lecture Notes in Computer Science, vol. 5203, Springer, Berlin, Heidelberg (2008).
［7］https://www.r-ccs.riken.jp/labs/lpnctrt/projects/kmath-random/
［8］https://www.shoup.net/ntl/index.html
［9］https://github.com/satakeshinsuke/random_number/blob/master/Chapter3.tar

J. Plasma Fusion Res. Vol.96, No.7 (2020) 372-378

講座 シミュレーションのための乱数入門

4．乱数の品質と検定法

4. Quality and Verification Methods of Random Numbers

佐竹真介
SATAKE Shinsuke
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）

疑似乱数には様々な発生法がありますが，良い疑似乱数，悪い疑似乱数の判断はどのように考えられているでしょうか．本章ではまず，結晶構造や多次元均等分布性について説明した後，乱数の満たすべき統計的な性質に基づくいくつかの検定テスト法と，それを2.1節と3章で紹介した疑似乱数の発生法に対して試した結果を紹介します．

Keywords:
pseudo random number, quality of random number, verification of pseudo random number

本章では，「良い乱数」「悪い乱数」とはどういうものか，疑似乱数の品質の判断基準と検定の仕方について概説します．第2章でも繰り返し述べてきたことですが，疑似乱数をシミュレーションに利用するにあたって最も気にするべきことの1つが，この乱数の「品質」です．疑似乱数の発達の歴史は，いかに品質の良い疑似乱数を効率よく決定論的なアルゴリズムから生み出すかの探求であると言っても過言ではないでしょう．また，疑似乱数に対する品質の不満や不信が物理乱数発生器の開発を促したとも言えます．

とは言え，何をもって「良い」疑似乱数であるとするかというと，これは非常に難しい問題でもあります．例えば偏微分方程式や数値積分の数値解法であれば，より高精度で誤差の蓄積が少ない，保存すべき量を保存する，数値不安定性を起こしにくい，といった明確な良し悪しの基準をその数値アルゴリズムの用途に応じて定められます．しかし乱数の場合，確率論的な統計量に対しての判定の議論が主となり，「この試験をパスした疑似乱数は真の乱数に近い」と言い切れるものがある訳ではありません．むしろ，発生した数列が，真の乱数であるならば満たす，考えうる限り多くの統計的な性質A，B，C，…を満たすならばそれは良い乱数だ，という少し漠然とした判断基準になります．また，数学的に厳密な乱数の品質の議論をするには高度な数学（統計学だけでなく整数論，Galois（ガロア）体等）の知識を必要とするため，この講座で全てを説明することはできません．そこで，ここではわかりやすい検定法の実例を挙げながら解説したいと思います．

・疑似乱数の結晶構造と均等分布性

「とても良い」乱数を定義することは難しいですが，「とても悪い」疑似乱数を見つけることはそれほど難しくありません．2.1節で線形合同法（混合合同法）の問題点は，周期長の短さと，多次元分布における結晶構造であることを触れました．この結晶構造とは，乱数列$\{r_i\}$を用いて，n次元空間に点列$(r_i, r_{i+1}, \cdots, r_{i+n})\,(i = 1, 2, \cdots)$を並べた時に，その点列が$n$次元空間を一様に埋め尽くすのではなく，ある決まった大きさのn次元格子の頂点にのみ点列が存在することを意味します．**図1**に2.1節で紹介した線形合同法による乱数列$r_{i+1} = 1103515245 r_i + 12345 \,(\mathrm{Mod}\, 2^{32})$を3次元空間に並べた場合の例を示します．見る角度を選ぶと，図のように点列(r_i, r_{i+1}, r_{i+2})がある平面上に整列していることがわかります．このようなパターンは，真に一様な乱数であれば出ないため，線形合同法は品質が劣る疑

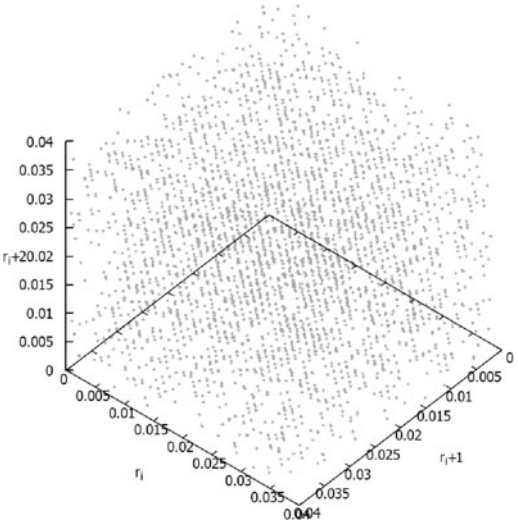


図1　線形合同法 $r_{i+1} = 1103515245\, r_i + 12345\ (\mathrm{Mod}\ 2^{32})$で発生した乱数から，3次元空間に点列$(r_i, r_{i+1}, r_{i+2})$を並べた場合に見られる結晶構造の例．

National Institute for Fusion Science, Toki, GIFU 509-5292, Japan
corresponding author's e-mail:satake@nifs.ac.jp

似乱数だと判断されるわけです．

線形合同法については，次元 n をどのように取っても必ず結晶構造を持つことが数学的に証明されています[1]．対照的に，2.1節で紹介したTausworthe（トーズワース）法やMersenne-Twister（メルセンヌ・ツイスター）法（MT法）は，長周期性だけでなく，多次元空間における分布の均等性が良いという特徴も持ちます．例えば，最もよく使われている，周期長が $2^{19937}-1$ の32ビット整数バージョンのMT法の場合，623次元空間に均等分布するとされています．しかし，実際にこのような超多次元空間の分布を，長い疑似乱数の一周期に渡って調べることは不可能です．そのため，これらの疑似乱数における多次元均等分布性は，乱数発生法の原理から以下のように説明されます[2]．

まずTausworthe法の場合，漸化式

$$\boldsymbol{x}_{i+p} = \boldsymbol{x}_{i+p} \oplus \boldsymbol{x}_i \qquad (1)$$

（$\boldsymbol{x}_i$ は$\{0,1\}$を元とするwビットの行ベクトル）に対応する特性多項式 $x^p + x^q + 1$ がGalois体GF(2)上で定義され，これが $t = 2^p - 1$ の時は多項式 $x^t - 1$ を割り切るが，$t < 2^p - 1$ に対しては割り切れない場合に，この特性多項式を p 次の原始多項式と呼び，生成される数列の各ビットの周期がそれぞれ $2^p - 1$ となると2.1節で説明しました．また，$2^p - 1$ が（1）式の形の漸化式が取りうる最大周期長でもあります．このように最大周期長を取っている場合，$\boldsymbol{x}_i$ の j 番目のビット $x_{i,j}$ に着目して，それを p 個ずつの組にして並べたもの $\{x_{i,j}, x_{i+1,j}, \cdots, x_{i+p-1,j}\}$ $(i = 1,2,\cdots,2^p-1)$ を考えると，これらの組の取りうるパターンは，全てのビットが0というパターンを除いた $2^p - 1$ 通りです．実は，特性多項式が p 次の原始多項式となる場合には，この $x_{i,j}$ を p 個並べた組 $\{x_{i,j}, \cdots, x_{i+p-1,j}\}$ が取りうる $2^p - 1$ 通りの全パターンが，乱数列の一周期の中にちょうど1回ずつ含まれることが証明されています．Tausworthe乱数の各ビットはこのような意味で p 次元に均等分布している，と言われます．

ところでTausworthe法では w ビットベクトル $\boldsymbol{x}_i$ の各ビットは独立に決まるため，$\boldsymbol{x}_i$ の初期値を上手く選ばないと w ビットベクトル全体としての周期を各ビットの最大周期長 $2^p - 1$ に一致させられないと2.1節で説明しました．では w ビットベクトル $\boldsymbol{x}_i$ に対する均等分布性に関してはどうでしょうか？その説明をするために，まず定義を明確にしましょう．疑似乱数の周期を P とした時，w ビットベクトル $\boldsymbol{x}_i$ を n 個並べた組 $\{\boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \cdots, \boldsymbol{x}_{i+n-1}\}$ $(i = 1,2,\cdots,P)$ が v ビット（$v \le w$）の精度で n 次元均等分布するとは，$\{\boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \cdots, \boldsymbol{x}_{i+n-1}\}$ の上位 v ビットが取りうる $2^{nv} - 1$ 個（全てゼロは除く）のパターンが，一周期中に同じ回数だけ現れることを意味します．Tausworthe法では，$\boldsymbol{x}_i$ ベクトルを p 個（ここで p は周期長ではなく，（1）式の階数）並べた組 $\{\boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \cdots, \boldsymbol{x}_{i+p-1}\}$ が，全wビットの精度で均等分布しうる次元は最大で $\lfloor p/w \rfloor$ であることが知られています（$\lfloor a \rfloor$ は a 以下の最大整数の意味）．しかし，実際に何次元で均等分布するかは，周期長と同様に初期条件 $\boldsymbol{x}_i$ $(i = 1,2,\cdots,p-1)$ の各ビットの選び方に依存します．2.1節でも述べましたが，世の中に出回っているTausworthe法のプログラムは適切な初期化ルーチンと通常セットになっているのでユーザがあまり気に病まずともよいですが，周期長や均等分布性のよい初期値の設定の仕方について詳しく知りたい方は[3,4]を参考にして下さい．

ところでwビットベクトルの組 $\{\boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \cdots, \boldsymbol{x}_{i+p-1}\}$ が取りうるビットの組み合わせは $2^{pw} - 1 > 2^p - 1$ 通りあるので，これらの組み合わせが一周期中に1回ずつ出てくるような数列を作ることができれば，より長周期でより高次元に均等分布する疑似乱数列を作ることができそうです．そこで，Tausworthe法では各ビットが（1）式から独立に決まっていたものを，ビット間でのシャッフルを取り入れることによってそれを実現したものがMT法です．MT法の漸化式は

$$\boldsymbol{x}_{i+n} = \boldsymbol{x}_{i+m} \oplus (\boldsymbol{x}_i^{w-r} | \boldsymbol{x}_{i+1}^{r})\boldsymbol{A} \qquad (2)$$

でした（式の詳細は2.1節を参照）．この場合，$\boldsymbol{x}_i$ ベクトルを w ビット乱数として先ほどと同様に漸化式の階数 n だけ並べた組 $\{\boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \cdots, \boldsymbol{x}_{i+n-1}\}$ を考えると，取りうるビットの組み合わせは $2^{nw-r} - 1$ 通りになります．MT法ではこれらが一周期の中で1通りずつ出るようにパラメータ $(m, n, r, \boldsymbol{A})$ が上手く選択されています．例えば，2.1節で紹介したサンプルコードmt19937ar.fでは，$n = 624$，$m = 397$，$r = 31$，$w = 32$ であり，周期長 $P = 2^{624\times 32 - 31} - 1 = 2^{19937} - 1$ になっており，w ビット精度で $\lfloor (nw - r)/w \rfloor = 623$ 次元均等分布することになります．なお，MT法では w ビット精度の均等分布性だけでなく，各上位 v ビットに対する均等性についても工夫されています．上位 v ビットに関する均等分布性は，理論上 $\lfloor (nw - r)/v \rfloor$ 以下にしかなり得ませんが，MT法では $v = 1,2,\cdots,w-1$ のそれぞれについても極力最大値に近づくように，$\boldsymbol{x}_i$ ベクトルに追加の"Tempering"と呼ばれる操作を付け加えています（2.1節の（4）式を参照）．各 v ビット精度での均等分布の次元 $k(v)$ が具体的にどの様になっているかは，[5]に示されています．このように，MT法は単に周期長が長いだけでなく，多次元分布の均等性についてもTausworthe法より良好になるように作られています．

・統計的性質による疑似乱数の検定

ここまで多次元分布の均等性について見てきましたが，それさえ良ければ品質のよい疑似乱数であるという訳ではありません．実際に我々が疑似乱数を使う時には，w ビット整数乱数 $r_i \in [0, 2^w - 1]$ を $2^w - 1$ で割って $[0,1]$ 区間の一様乱数として使うことが一般的です．以下，$x_i = r_i/(2^w - 1)$ としてこの実数疑似乱数列 $\{x_i\}$ の品質について調べていきましょう．$\{x_i\}$ が「良い一様乱数」として満たすべき統計的な性質は，いくらでも挙げることができますが，例として以下のようなテストを考えてみましょう．

（ｉ）**連検定**：例えば，$x_{i-1} > x_i < x_{i+1} < x_{i+2} < \cdots < x_{i+m} > x_{i+m+1}$ となるような区間のことを，長さ m の上昇連と呼び

ます（逆は下降連）．このような連が全体の長さ N の数列の中に出現する数を l_m^N と表します．あるいは，平均値 1/2 以上（未満）の値が m 回続く連を考え，その出現数を f_m^N とします．$\{x_i\}$ が一様乱数であるなら，それらの期待値は以下のように求まります[1,6]．

$$\mathrm{E}[l_m^N]=\frac{2}{(m+3)!}[N(m^2+3m+1)-(m^3+3m^2-m-4)],$$

$$\mathrm{E}[f_m^N]=\frac{N+3-m}{2^{m+1}}.$$

十分な長さ N の乱数列を発生させ，その中の長さ m の連の出現回数と，上記の期待値との間に統計的に有意な差が生じていないかを確認します．

（ii）近接値の出現率：2 つの連続した乱数の組 $\{x_i, x_{i+1}\}$ $(i=1,2,\ldots)$ が，区間 $\Delta_{m-1}<|x_{i+1}-x_i|\le\Delta_m$ に現れる回数 c_m^N の期待値は，

$$\mathrm{E}[c_m^N]=(N-1)[(2\Delta_m-\Delta_m^2)-(2\Delta_{m-1}-\Delta_{m-1}^2)]$$

となります．$\Delta_m \ll 1$ に対して調べることで，近接した値が不自然に連続して出ていないかを確認します．

（iii）平均値と二乗平均値の分布：これは最も単純なテストの一つと言えます．一様乱数 $\{x_i\}$ の平均，二乗平均の期待値はそれぞれ $\mathrm{E}[x_i]=1/2$，$\mathrm{E}[x_i^2]=1/3$ ですが，大きさ N_s の標本を多数取った場合，$\{x_i\}$ が一様乱数であれば x_i，x_i^2 の平均値はそれぞれ，正規分布 $\mathrm{N}(1/2, 1/(12N_s))$，$\mathrm{N}(1/3, 4/(45N_s))$ に従って分布するはずです．生成された疑似乱数列の標本平均が，この分布に従うかを確認します．

なお，連検定と近接値の出現率の検定については，（i）の連の長さ，あるいは（ii）の近接値の区間 Δ_m のインデックス $m=1,2,\cdots$ に対する出現数分布の期待値が上の $\mathrm{E}[l_m^N]$，$\mathrm{E}[f_m^N]$，$\mathrm{E}[c_m^N]$ のように求められているので，長さ N の標本中の長さ m の連の出現数の分布が，期待値の分布とどの程度よく一致しているかをチェックします．このような「分布の当てはまりのよさ」のチェックには χ^2 検定が使われます．ある確率的事象が k 個の互いに背反なクラス $C_1, C_2, \cdots, C_k$ に分別される時，S 個の標本中の i 番目の事象 C_i の観測度数 n_i，その期待度数 E_i（$\sum_{i=1}^k n_i=\sum_{i=1}^k E_i=S$）に対して次式から求められる値を χ^2 値と呼びます．

$$\chi^2=\sum_{i=1}^k\frac{(n_i-E_i)^2}{E_i} \qquad (3)$$

もし観測度数の分布（例えば，疑似乱数列に含まれる長さ m の上昇・下降連の個数の分布）が，疑似乱数が一様乱数であると仮定して求めた期待値の分布に実際に従っているなら，（3）式に従って長さ N の標本を S 個用意して評価した χ^2 値は，自由度 $\nu=k-1$ の χ^2 分布に従うと期待されます．ここで自由度 ν の χ^2 分布（確率密度関数）は次式で定義されます．

$$f(\chi^2,\nu)=\frac{1}{2^{\nu/2}\Gamma(\nu/2)}(\chi^2)^{\nu/2-1}\mathrm{e}^{-\chi^2/2} \qquad (4)$$

今回の検定では，(i) の連検定については $m=1,2,\cdots,5$ と $m\ge 6$ の計 6 ケースの出現率，(ii) の近接値の出現率についても $\Delta_k=\{0.00, 0.01, 0.02, 0.05, 0.10, 0.20, 1.00\}$ とした場合の計 6 区間について調べるので，それぞれ自由度 = 6 − 1 = 5 の χ^2 分布に従うかどうかをチェックします．

一方，(iii) の平均値，二乗平均値の検定は Z 検定によって行います．ある確率変数 X の母集団の平均値と分散が (μ_0, σ^2) と仮定される時，大きさ N_s の標本 $\{x_i\}$ の標本平均を $\bar{x}$ とし，次式で定義される値 Z によって「母集団の平均値が μ_0 である」という仮定が正しいか否かを判定します．

$$Z=\frac{(\bar{x}-\mu_0)\sqrt{N_s}}{\sigma} \qquad (5)$$

仮説が正しければ，Z は平均 0，分散 1 の正規分布 $\mathrm{N}(0,1)$ に従うはずです．もし標本から評価した $|Z|$ が 1.96 以上の場合，仮説が正しければその確率は 5 %なので，この場合「有意水準 5 %で仮説（母平均が μ_0 である）は棄却された」ことになります．同様に，$|Z|>2.58$ となる場合は有意水準 1 %で棄却されます．ここでは，疑似乱数 $\{x_i\}$ の平均値，二乗平均値について，母集団が $[0,1]$ の一様乱数であるとの仮説から得られる平均と分散の期待値 (μ_0, σ^2) を用いて Z 検定します．

なお通常，Z 検定は N_s 個の標本のサンプリング 1 回について Z を評価し，仮説が正しいか否かを判定しますが，ここではそのようなサンプリングを S 回行い，評価した Z が実際に $\mathrm{N}(0,1)$ の正規分布をしているか，5 %，1 %の棄却域に入るサンプル数が実際に $0.05S$，$0.01S$ に近いかを調べます．

乱数検定の対象としては，2.1節で紹介した線形合同法 $(x_{i+1}=a_0x_i\,(\mathrm{Mod}\,P), a_0=1103515245, b=12345, P=2^{32})$，Tausworthe 法（$\boldsymbol{x}_{i+p}=\boldsymbol{x}_{i+p}\oplus\boldsymbol{x}_i$，$p=250$，$q=103$），Mersenne-Twister 法（mt19937ar.f）と，第 3 章で紹介した KMATH_RANDOM 版の SFMT の 4 つとします．なお，SFMT については Jump Method によって 2^{100} 飛ばしで作った初期値から開始した10本の乱数列それぞれについてテストしました．それぞれの疑似乱数から大きさ $N=10$万の標本を $S=1$ 万回サンプリングし，上記の検定を行いました．

まず，2 種類の連と，近接値の出現頻度の分布を見てみましょう．結果はどの疑似乱数もほぼ同じものになったため，代表として mt19937ar の例を**図 2** に示しました．この図では 3 つのテストそれぞれにおける，連や近接値の実際の出現数（図中点で表示）と期待値（線で表示）を比較していますが，一様乱数を仮定して得られる期待値とよい一致をしていることがわかります．この出現率の分布の期待値との適合度を見るために，（3）式で評価した χ^2 値の度数分布をグラフにしてみると，**図 3** のようになり，確かに自由度 5 の χ^2 分布に従っていることがわかります．紙面の都合上全ての結果について図は載せませんが，SFMT の10本の乱数列だけでなく，Tausworthe 法や，乱数としての品質が劣るとされている線形合同法についても**図 2** や**図 3** をプロットしても有意な差は見られませんでした．

次に疑似乱数の平均値，2 乗平均値の分布について見て

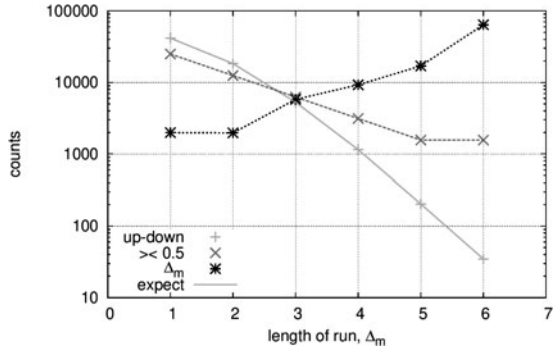


図 2　mt19937ar の疑似乱数に対する 2 種類の連と近接値出現回数の分布．“up-down”が上昇 - 下降連，“> < 0.5”が 0.5 以上又は以下の連，が $\Delta_{m-1} < |x_{i+1} - x_i| \leq \Delta_M$ の近接値の出現数であり，点がそれぞれの観測値，線が期待値を表します．長さ10万の乱数列を 1 万回サンプリングして評価しました．

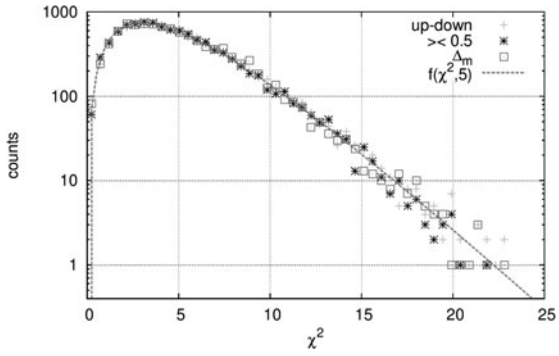


図 3　mt19937ar の疑似乱数に対する 2 種類の連と近接値出現回数の出現数分布に対する χ^2 値の度数分布（サンプル数＝1 万）．点がそれぞれの観測値，曲線が（4）式による自由度 5 の χ^2 分布を表します．

みましょう．**図 4** に線形合同法の結果を代表として示します．こちらも，一様乱数を仮定して求めた正規分布とよい一致を示していることがわかります．この場合も，乱数の発生法による有意な差は現れませんでした．（5）式による Z 検定の結果についても見てみましょう．**図 5** では，平均値と二乗平均値について，有意水準 5 %，1 %で棄却されるサンプルが 1 万個のうち実際にいくつあったかをプロットしてあります．また，**図 3** で見た χ^2 値の分布についても，χ^2 値が大きくて連などの分布が期待値によく一致していると言えない標本数（自由度 5 の場合 $\chi^2 > 11.07$ が 5 %，$\chi^2 > 15.09$ が 1 %の棄却域）も合わせて示しました．どの乱数発生法のどの検定結果も，実際にほぼ 5 %，1 %の標本がそれぞれの棄却域に入っています．

これらの結果から，ここで取り上げた 4 つの疑似乱数は，どれも今回紹介した検定をパスしたと言えそうです．驚くべきことに，線形合同法は明らかに結晶構造という乱数らしからぬ性質を持っているのに，これらの検定はパスできました．2.1節で整数の線形合同法乱数が偶・奇，偶・奇，のパターンを繰り返すことを紹介しましたが，このような乱数らしからぬ性質も今回のテストでは見逃されてしまいます．本章の冒頭でも述べた通り，「このテストさえ合格すれば OK」と言い切れるような，万能な乱数の検定法がある訳ではなく，考えうる限り多くのテストを合格で

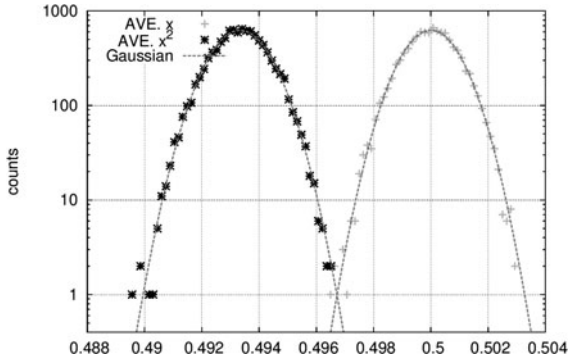


図 4　線形合同法で発生させた疑似乱数（標本の大きさ10万）の平均値と 2 乗平均値の分布（サンプル数 1 万）．なお 2 乗平均値は 0.16 右にオフセットしてプロットしてあります．“Gaussian”は一様乱数の場合に期待される正規分布を表します．

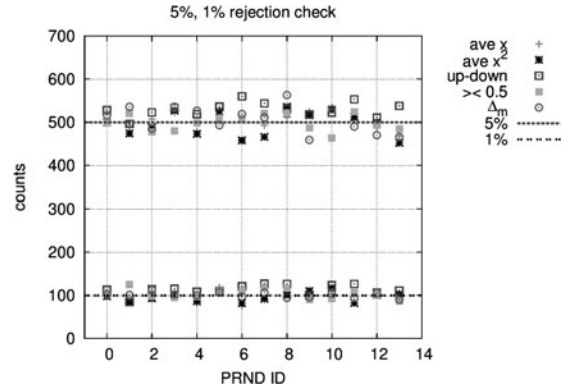


図 5　平均値，2 乗平均値に対する Z 検定と，2 種類の連と近接値の出現率に対する χ^2 検定で，それぞれ有意水準 5 %，1 %で棄却された標本数．横軸は乱数の発生法を表し，ID = 1～10 が10個の SFMT，11が線形合同法，12が Tausworthe 法，13が mt19937ar の結果です．また，ID = 0 は10個の SFMT の平均値を表します．

きる疑似乱数の発生法が「より乱数らしい」と言えるだけです．また，様々な検定を組み合わせても，「実際上この種の検定で棄却できるのは異常に不都合なアルゴリズムで，非常によいものを積極的に見出すことは，はなはだ難しい」（[1]の p. 32より抜粋）ことが以前から指摘されており，それゆえにより複雑な乱数検定法が多く考案され続けています．例えば，NIST SP 800-22[7]と呼ばれる乱数検定用のツール群には15種類もの検定法が含まれています．しかしながら，その検定法の中には問題点を指摘されているものも存在しているため[8]，利用には注意が必要です．

なお，NIST の検定法では疑似乱数に対する各テストの合否判定は，棄却域に入るサンプル数に対する Z 検定によって行っており，具体的には各テストにおいて 1 %棄却域に入った標本数が全体数 S に対して $S \times \left[0.01 \pm Z\sqrt{\frac{0.01(1-0.01)}{S}}\right]$，（$Z = 2.58$）の範囲に入っていれば，疑似乱数が「期待通りに有意水準 1 %の棄却域でテストに不合格した」ため「乱数検定のテストに合格した」という判断をします．ここでも，乱数の品質の検定はあくまで統計学的なスタンスでなされることが現れています．

NIST の合否判定を今回参考として行ったテストに当て

はめると，$S = 1$ 万なので合格の範囲は100±25ということになります．テスト結果の内訳を見ると，10個の SFMT の内の 2 つと線形合同法の上昇・下降連テストの χ^2 検定において126あるいは127個の標本が 1 %の棄却域に入った他は，全て 100±25 の範囲に収まりました．上昇連，下降連のテストについては筆者自身の経験として，一標本当たりの長さ N をかなり大きく取らないと，乱数の発生法に依らずテストから評価した χ^2 値が期待される χ^2 分布より若干上振れする傾向があります．これは長さ N の数列に含まれる連の総数自体が確率変数であることや，**図 2** からわかるように平均値以上・以下の連の分布と比較して，上昇・下降連の分布は長さ 5 や 6 以上の連の出現率が長さ 1， 2 の連に比べて極端に少ないため，χ^2 値の誤差（分散）が大きく出やすいためではないかと予想しています．いずれにしても，棄却されたサンプル数の 1， 2 程度の差をもって直ちに SFMT の内の 2 つと線形合同法がテストに不合格だとみなすのは早計でしょう．実際，1 サンプルの長さを10万から100万に延ばすと，全ての Z 検定，χ^2 検定において 1 %の棄却域に入った標本数が 100±25 の合格圏に収まりました．このように，検定法によって適切な 1 サンプルの大きさが異なったり，χ^2 値の出方に癖があったりするので，NIST やその他，世の中に出回っている乱数の検定法を使う際，テストのサイズ設定や合否判定の解釈には注意が必要です．

検定プログラム群としては他にも，TESTU01[9]と呼ばれる乱数の品質検定プログラム群が知られています．こちらはテストの規模に応じて SmallCrush，Crush，BigCrush いう 3 つの検定プログラム群が用意されており，最も詳細な検定をする BigCrush では106種類もの検定法による160通りのテストが行われます．文献[9]の Table.I には多くの乱数発生法に対する検定テストの結果が載っており，棄却域 10^{-8}% で不合格だったテストの数が示されています．残念ながら本節で試したものと同じ線形合同法はありませんが，パラメータの異なる様々な線形合同法の結果が載っており，おしなべて多くのテストで不合格になっています．Tausworthe 法（本節と同じものは文献[9]の Table.I 中の GSFR (250, 103)），MT 法（同 Table 中の MT19937）の結果も載っています．Tausworthe 法に比べて MT 法の方が不合格になったテストの数が少ないことがわかりますが，MT 法でも BigCrush の160個のテスト中 2 つで不合格になっています．これを「たった 2 つ」とみなすか，「 2 つも不合格」とみなすかは，先ほども述べたように万能な乱数のテストというものがないため難しいところです．なお MT 法については初期のバージョンには，初期化の seed の値を変えても初期ベクトルの状態があまり変わらない，一旦漸化式中のベクトルのビットが 0 が過多な状態に陥るとそこからなかなか抜け出せない，といった問題点が指摘されていました．これらが 2 つのテストで不合格になったことと関係があるかについては，筆者が調べた限りではっきり述べられたものはありませんでした．初期化の問題については本講座で紹介した mt19937ar において既に改善されており，また 0 超過状態からの回復についても並列化 MT 法として紹介した SFMT において改善されています．もし，BigCrush テストの結果を重視するのであれば，MT 法を改良して省メモリ化し，かつ BigCrush テストも全て通るように出力を工夫した TinyMT[10] が MT 法の作者らによって発表されているので，それを使うという選択肢もあります．ただし均等分布性の次元は元の MT 法より低くなっています．TinyMT はまだ新しく一般的に普及していないため本講座では紹介しませんでしたが，Dynamic Creator や Jump Method も備えているので，最新の疑似乱数を試したい方は利用してみてはいかがでしょうか．

・並列乱数の品質について

ここまで，1 つの疑似乱数発生法に対する品質の検定について説明してきましたが，最後に，第 3 章で紹介したような並列疑似乱数についての品質について少し述べたいと思います．乱数を並列化する上で重要な性質は，それらの乱数列同士が互いに独立で相関関係を持たないことです．第 3 章では MT 法の並列化の方法として，過去に本学会誌[11] で紹介した Dynamic Creator の他に，1 本の MT 法乱数列のそれぞれ遠く離れた点を出発点として，並列計算において異なる部分区間の乱数列を利用するという Jump Method を新たに取り上げました．並列計算によるシミュレーションで，これら並列疑似乱数を利用する時には，それらの乱数列が互いに独立であるという前提で利用するわけですが，その保証はどこにあるのでしょうか？

Dynamic Creator の場合は，MT 法を特徴づける漸化式の形が異なるものを必要な数だけ用意するというものでした．その独立性については，作者自身も厳密な数学的証明はなく，漸化式を特徴づける特性方程式が互いに素な疑似乱数同士は独立である，という仮説に基づいていると説明しています[12]．一方，Jump Method では 1 本の MT 法の疑似乱数列を例えば 2^{100} 飛びに使うという方法であり，それぞれの部分区間が統計学的な意味で独立とみなせるかどうかについて，筆者が調べた限り具体的な説明は見当たりませんでした．しかし，MT 法が持つ一般的な意味での疑似乱数の品質の良さや高次元均等分布性を踏まえると，部分区間 A と B から取った数列 $\{a_j\}$ と $\{b_j\}$ に対して，例えば $a_j b_{j+k}$ $(k = 0, 1, \dots)$ に対する相関係数を具体的に評価したところで，何かおかしな相関が現れることはないと期待して良さそうです．このように，MT 法による疑似乱数並列化に関しては，まず間違いなく大丈夫だろう，という感じで実応用に供されているというのが実情のように思われます．

筆者自身は15年くらい前から MT 法（当時は Dynamic Creator，現在は Jump Method）を並列シミュレーションに応用してきましたが，当時から並列乱数列の独立性について議論や検証の話が見当たらないことに不安を感じつつ使い始めました．最終的には，160本（後に1024本）の Dynamic Creator の MT 法乱数列について，${}_{160}C_2$ (${}_{1024}C_2$) 通りの全ての組み合わせに対して，相関チェックを自ら試すという実力行使に出ました．この際に考えた Checker-

board test と名付けた検定法[13]では，2つの乱数列 $\{a_j\}\{b_j\}\in[0,2^n]$ に対し，$(2^n\times 2^n)$ の2次元のマス目を考え，(a_j,b_j) を順次打っていき，iステップ目まで進んだ時に，全部で $N=2^{2n}$ 個のマス目のうちまだ1度も点を打たれていない数 $m(i)$ を観測します．$\{a_j\}\{b_j\}$ が無相関な一様乱数だと仮定した場合の期待値とその分散は

$$\mathrm{E}[m(i)]=N\left\{1-\left(\frac{N-1}{N}\right)^i\right\}\simeq N\exp\left(-\frac{i}{N}\right),$$

$$\sigma^2[m(i)]=N\left\{N+(N-1)\left(\frac{N-2}{N}\right)^2-(2N-1)\left(\frac{N-1}{N}\right)^i\right\}$$

と与えられます．これに対し，観測した $m(i)$ が $E[\mathrm{m}(i)]\pm(1\sim 3)\times\sigma(i)$ の範囲に現れる割合が，上式の期待値と分散を持つ正規分布に従うかどうかをチェックしました．もしテストの結果がそうなっていなければ $\{a_j\}\{b_j\}$ が無相関な一様乱数だという仮定が棄却される，即ち相関があると考えられるわけです．MT 法は32ビットの精度を持っていますが，その精度でチェックすることは時間的に不可能だったので，$\{a_j\}\{b_j\}\in[0,2^{14}]$ にビットを落として $i=20N$ まで追跡しました．その結果は[13]に示すように，独立な乱数として期待される通りに $m(i)$ が i と共に減少するということが，Dynamic Creator で作った全ての MT 法の乱数列の組み合わせについて確認できました．

このようなテストをわざわざするほど並列乱数の独立性について心配する必要は今にして思えばなかったかも知れません．また，このテストに限らず乱数の検定全般についてよく指摘される点ですが，いくら乱数のテストを行ったとしてもそれはテストを行った部分区間における品質をチェックしたに過ぎないということは否めません．しかしながら，並列乱数の独立性については自らテストをしなければ何も分からないという程に，検定法や検定結果が発表されていないというのが実情です．並列計算に疑似乱数を応用するプログラマは，その事実を頭の片隅に留めておいてもよいのではないか，と私は考えます．よりシビアに乱数の独立性を求めたいユーザは，第2.2節で紹介したような物理乱数の利用を検討するのも一つの手段だと思います．

・おわりに

最後に本章の補足として，実際にシミュレーションを行う際に乱数を検定してから使うべきか否か，という問題について少し述べたいと思います．例えば粒子シミュレーションで1空間セル当たり N 個のテスト粒子の初期速度を乱数で与える際に，大きさ N の疑似乱数に対して本節で例を示したような統計学的な検定テストを行えば，5％，1％など任意の有意水準で「一様乱数らしくない」疑似乱数のセットを排除することも可能です．しかし，疑似乱数が実際に真の乱数に近い挙動をしている場合，1％の棄却域に入るような N 個の疑似乱数のセットは，実際に約1％の確率で生じて然るべきものでもあります．空間セルの数が例えば10万個ある場合，統計的には1000個のセルで疑似乱数が1％の棄却域に落ちる可能性がある，ということです．

この例のように，初期分布を作るのに乱数を使う場合は1回きりのことなので，検定を掛けて棄却域に入った場合は乱数を作りなおしたとしても，総計算時間に対する計算時間の増加は無視できるでしょう．また，テスト粒子の平均初期速度がなるべく仮定した初期条件通りの状態から計算をスタートさせたいといった要請があるかも知れません．しかしそれならば，平均値が棄却域に入るような乱数でもそのまま N 個のテスト粒子の初速の決定に使い，後で粒子の速度を全体的にずらして仮定した初期条件に粒子の平均速度を合わせてもよいわけです．また，時間発展シミュレーションの中で，ランダムウォークや衝突計算を乱数で模擬する場合，1％の棄却域に入る乱数を一々排除していると計算時間の増大につながりかねませんし，何より本当に1％の確率で起こるはずの「レアイベント」を人為的に排除してしまう可能性があります．

筆者自身は初期化においても時間発展計算の最中においても，疑似乱数の検定はしてもいいが，それは乱数の品質の確認のためだけであり，棄却域に入る乱数もそのまま使うというスタンスを取りますが，これはシミュレーションコードが扱う問題の性質や研究者個々人の乱数に対する考え方によって変わりうるものだと思います．ここで強調しておきたいことは，乱数の検定法が統計的な性質に基づくものである以上，真の乱数でも疑似乱数でも，ある確率で検定の棄却域に入る乱数の標本は必ず生じ，むしろそれは乱数として自然な振る舞いであるということです．

本章では筆者自身の過去の経験を交えながら，疑似乱数の品質について概説してみました．乱数を使うシミュレーションを行う研究者の方々が，ふと「自分が使っているこの疑似乱数は大丈夫なのだろうか？」という疑問を感じた時にこの記事で紹介した内容が参考になれば幸いです．

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］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**, 86-95 (1993).

［3］伏見正則：乱数（東京大学出版会，1989）．

［4］B.J. Collins, G. Barry Hembree, J. Association for Computing Machinery 33, 706-711 (1986).

［5］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

［6］H. Levene and J. Wolfowitz, Ann. Math. Stat. **15**, 58-69 (1944).

［7］https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software

［8］吉田等明 等：電子情報通信学会 信学技報 **112**, 13 (2012).

［9］P. L'ecuyer and R. Simard, ACM T. Math. Software, 33, Article 22 (2007).

［10］M. Saito *et al.,* RFC 8682, DOI 10.17487/RFC8682 (https://www.rfc-editor.org/info/rfc8682).
［11］講座「核融合プラズマシミュレーションの技法 5．粒子シミュレーションのコーディング技法」プラズマ・核融合学会誌 **89**, 245 (2013).
［12］M. Matsumoto and T. Nishimura, "Dynamic Creation of Pseudorandom N umber Generators", Monte-Carlo and Quasi-Monte Carlo Methods pp. 56-69 (Springer, Berlin, Heidelberg, 1998).
［13］S. Satake *et al.,* Lecture Notes in Computer Science book series Vol. 4759, pp. 344-357 (Springer-Verlag, Berlin, Heidelberg, 2006).

なお、本講座の執筆のために統計数理研究所の研究者に意見を聞きに伺ったことがきっかけで、朝日新聞の記者から乱数のシミュレーション研究への応用について取材を受けました。その記事は2020年 6 月 3 日の朝日新聞夕刊および朝日新聞デジタルに『現場へ！「乱数の世界へようこそ」』というタイトルで掲載されました。

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．

J. Plasma Fusion Res. Vol.96, No.8 (2020) 405-417

講座 シミュレーションのための乱数入門

5．乱数の変換と利用例

5. Transformation of Random Numbers and its Applications

佐竹真介，菅野龍太郎，本多　充[1)]
SATAKE Shinsuke, KANNO Ryutaro and HONDA Mitsuru[1)]
自然科学研究機構核融合科学研究所，[1)]量子科学技術研究開発機構

（原稿受付：2020年5月22日）

この章では，プラズマ・核融合分野，特に，磁場によって閉じ込められた核融合プラズマの理工学におけるシミュレーションにおいて，乱数がどのように利用されているのかを紹介します．シミュレーションにおいては，一様乱数だけでなく，正規乱数など様々な乱数が目的に応じて利用されています．そこで，まずは，乱数を用いたシミュレーションを行う際に必須である，様々な乱数を生成する「乱数の変換」について説明します．次に，シミュレーションにおいて，具体的にどのように乱数が用いられているのかを解説します．乱数を用いるシミュレーション手法は，一般に，モンテカルロ法と呼ばれます．ここでは，モンテカルロ法に基づくシミュレーションについて，基礎的な例を挙げて説明します．また，近年，様々な科学技術分野において注目を集めている遺伝的アルゴリズムと，そのアルゴリズムを用いた大域的最適解を求めるシミュレーションについても説明します．

Keywords:
random number transformation method, Monte-Carlo method, Dirichlet-Poisson solver, stochastic differential equation, genetic algorithm, global optimization

5.1　乱数の変換

本講座の第2章から4章で紹介してきた疑似乱数や物理乱数は基本的に $x_i \in [0,1]$ の実数値一様乱数でした（利用するサブルーチンによって $(0,1)$, $[0,1)$ ないし $(0,1]$ の場合もありますが，ここでは厳密な区別はしません）．しかし，様々なシミュレーションに乱数を利用する際には，一様乱数以外の様々な確率密度関数に従う乱数が必要になります．本節では，そのような任意の確率密度関数に従う疑似乱数を発生させる方法を紹介します．

任意の確率密度関数に従う乱数を生成する方法としては，大きく分けて2つの代表的な方法が知られています．まず1つ目は，逆変換法（直接法）と呼ばれる方法で，確率密度関数 f に対し累積分布関数

$$x = F(y) = \int_a^y f(t)\,\mathrm{d}t$$

を考えることで，一様乱数 $\{x_i\}$ から F を通じて f に従う乱数 $\{y_i\}$ を生成するという方法です．以下，具体的に見ていきましょう．

ここでは簡単のため，確率密度関数の定義域が $[a,b]$ の連続な関数であるとします．もし下限，上限の両方あるいはいずれかが $-\infty$, $+\infty$ となる場合は，$t<a$, $t>b$ で十分に $f(t)\simeq 0$ とみなせるような下限，上限を設定しておきましょう．さて，確率密度関数の性質として，$f(t)\geq 0$ であること，定義域全体で積分すると1になることの2つがあります．したがって，累積分布関数 $F(y)$ は，値域を $[0,1]$ に持つ単調増加関数になります（f が不連続な場合については後述）．よって，y は F の逆関数 F^{-1} を使って

$$y = F^{-1}(x)$$

のように表すことができます．さて，今 X が区間 $[0,1]$ に一様分布する確率変数とすると，$X \leq x$ となる確率は $\Pr\{X \leq x\} = x = F(y)$ です．この時，これに対応して，$Y = F^{-1}(X) \leq y = F^{-1}(x)$ となる確率も同じく $F(y)$ です．すなわち，

$$\Pr\{F^{-1}(X) \leq y\} = \Pr\{X < F(y)\} = F(y)$$

が成り立ちます．上式より，区間 $[0,1]$ の一様乱数 $\{x_i\}$ を用いて $y_i = F^{-1}(x_i)$ として与えた $\{y_i\}$ は累積分布関数 $F(y)$ に従う分布をとる，つまり確率密度関数 f に従う分布になることがわかります．

ところで，$F^{-1}(x)$ は常に解析的に表せるわけではなく，不連続な関数である場合も考えられます．そのような場合は，生成したい乱数 $\{y_i\}$ の値域 $[a,b]$ を十分細かく分割し，数値積分を使うなどして $x_k = F(y_k)$ $(k = 0,1,2,\cdots,N)$，ただし　$x_0 = 0$，$x_N = 1$，の離散データのテーブルを用意します．そして，一様乱数ルーチンで生成した x_i の値が $x_k \leq x_i < x_{k+1}$ に入る時，例えば線形補間を用いて，

$$y_i = y_k + \frac{y_{k+1} - y_k}{x_{k+1} - x_k}(x_i - x_k)$$

National Institute for Fusion Science, Toki, GIFU 509-5292, Japan
corresponding author's e-mail:satake@nifs.ac.jp

のようにして $y_i = F^{-1}(x_i)$ の近似値を得ることで $\{y_i\}$ を生成することが可能です．

なお，$\{y_i\}$ が離散値しか取らない場合も，上と同様の考えから逆変換法を用いて生成できます．確率変数 y が m 通りの離散値 y_k（$k = 1, 2, \cdots, m$）を取り，それぞれの発生確率が p_k であるとします（ただし $\sum_{k=1}^{m} p_k = 1$）．この時，区間 $[0, 1]$ 一様乱数 x_i に対して，

$$\sum_{k=0}^{n-1} p_k < x_i \leq \sum_{k=0}^{n} p_k$$

を満たす n を求め，この n に対して $y_i = y_n$ を採用することで，$\{y_i\}$ を生成します．ただし，上式の総和を取る時に便宜的に $p_0 = 0$ とおきます．

もう 1 つの代表的な方法は棄却法と呼ばれるもので，一様乱数を使った試行錯誤によって得たい分布に従う乱数を発生させるものです．まず，先ほどと同様に，生成したい乱数 $\{y_i\}$ の範囲が $[a, b]$，その確率密度関数を f とし，さらに $f(t) < c$ であることが既知であるとします．ここで，c は厳密な最大値である必要はなく，確実に $f(t) < c$ となるような適当な定数を設定すれば大丈夫です（近い方が乱数の生成効率はよいですが）．次に，一様乱数発生ルーチンを使って，二組の一様乱数 $\xi_j \in [a, b]$ と $\eta_j \in [0, c]$ を用意します．これに対して，$f(\xi_j)$ と η_j の大小関係を判定します．もし $f(\xi_j) > \eta_j$ であれば $y_i = \xi_j$ を採用し，そうでなければ破棄し，次の (ξ_j, η_j) に対して同様の判定を行います．そして，必要な個数の $\{y_i\}$ が採用されるまでこの判定を繰り返せば，確率密度関数を f に従う乱数 $\{y_i\}$ が生成されます．

この他にも，$f(t) = e^{-t}$ や正規分布の場合など，特定の関数の場合に使える乱数の変換法が色々と知られています．ここでは代表的なものとして，2 次元空間における正規分布 $f(x, y)$ を与える，Box-Muller 法を紹介します．

Box-Muller 法では区間 $[0, 1]$ 一様乱数を二組用意します．それらを $\{u_i, v_i\}$ とします．この手法では u_i と v_i は無相関で独立な一様乱数であることを前提としますが，第 4 章で説明したような「品質の良い」疑似乱数であれば，実際に独立な疑似乱数列を 2 系統使う必要はなく，1 系統の疑似乱数列からその奇数番目を u_i，偶数番目を v_i，という具合に取れば十分です．さて，平均が $(0, 0)$，分散共分散行列が $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ となる 2 次元正規分布関数は

$$f(x, y) = \frac{1}{2\pi} e^{-\frac{x^2+y^2}{2}}$$

で与えられますが，これを極座標系（$r = \sqrt{x^2+y^2}$，$\theta = \tan^{-1}(y/x)$）で表現すると，

$$f(r, \theta) = \frac{r}{2\pi} e^{\frac{-r^2}{2}}$$

であり，θ に関しては $[0, 2\pi)$ に一様に分布していることになります（ここで因子 r は $(x, y) \to (r, \theta)$ の座標変換のヤコビアンから出てきます）．また，$\int_0^\infty \mathrm{d}r\, r e^{-r^2/2} = 1$ を満たすことは容易に確認できます．そこで，r 方向に関する分布を乱数から生成する方法を考えると，累積分布関数

$$u = F(r) = \int_0^r \mathrm{d}r' r' e^{-\frac{r'^2}{2}} = 1 - e^{-\frac{r^2}{2}}$$

の値域が $[0, 1]$ なので先ほどの逆変換法が使えそうです．しかしこのままでは逆関数が簡単に記述できないので，$u' = 1 - u$ と置き換え，$u' = e^{-\frac{r^2}{2}}$ とすると，その逆関数は $r = \sqrt{-2\ln(u')}$ であることがわかります．ところで $(r, \theta) \to (x, y)$ の逆変換は

$$x = r\cos\theta, \quad y = r\sin\theta$$

で与えられるので，一様乱数 $\{u_i, v_i\}$ を用いて，

$$x_i = \sqrt{-2\ln(u_i)}\cos(2\pi v_i),$$
$$y_i = \sqrt{-2\ln(u_i)}\sin(2\pi v_i),$$

と与えると，$\{x_i, y_i\}$ は 2 次元空間の正規分布 $f(x, y)$ に従う分布を持つ乱数となります．

他にも様々な関数形の確率分布を一様乱数から生成する方法が知られていますが，ここでは紹介しきれないので，興味のある方は文献[1, 2]などを参照してください．

（佐竹）

5.2 乱数の利用例

ここからは，磁場によって閉じ込められた核融合プラズマの理工学における乱数を用いたシミュレーション手法をいくつか紹介します．シミュレーションにおいてどのように乱数が用いられているのかを説明する前に，なぜシミュレーションで乱数を用いるのかについて考えてみます．

まずは，シミュレーションによって扱われるプラズマの諸現象が，どのような方程式により記述されているのか，その概略を見ていきましょう．議論を単純化するために，完全電離プラズマが，それぞれ $N/2$ 個の水素イオンと電子で構成されているとします．ここで，$N \sim 10^{21}$ 程度です．このとき，荷電粒子の運動は，Newton 方程式（および電磁場を解くために Maxwell 方程式）で記述することができます［3-5］．電場と磁場をそれぞれ $\mathbf{E}$ および $\mathbf{B}$ とし，j 番目の荷電粒子の質量を m_j，電荷を q_j として，Newton 方程式から粒子の位置 $\mathbf{x}_j(t)$ および速度 $\mathbf{v}_j(t)$ は，

$$\dot{\mathbf{x}}_j(t) = \mathbf{v}_j(t)$$

および

$$\dot{\mathbf{v}}_j(t) = \frac{q_j}{m_j}\{\mathbf{E}(t, \mathbf{x}_j(t)) + \mathbf{v}_j(t) \times \mathbf{B}(t, \mathbf{x}_j(t))\}$$

で与えられます．ここで，$j = 1, 2, 3, \cdots, N$ です．この Newton 方程式と Maxwell 方程式を解くことができれば，原理的には，プラズマの温度，密度，閉じ込め時間など，磁場閉じ込め核融合炉のプラズマ性能を評価するために必要な情報を得ることができるはずです．しかし，現実には，この方程式系を解くことは不可能です．なぜなら，全粒子間の相互作用を考慮して，N 個すべての荷電粒子の Newton 方程式を解く必要があるからです．

そこで，プラズマの状態を近似的に記述する基礎方程式

としては，水素イオンと電子のそれぞれに対して，2体相互作用のみを考慮した6次元位相空間内の1体粒子分布関数 $f_\alpha(t,\mathbf{x},\mathbf{v})$ の時間発展方程式であるBoltzmann方程式：

$$\left\{\frac{\partial}{\partial t}+\mathbf{v}\cdot\frac{\partial}{\partial \mathbf{x}}+\frac{q_\alpha}{m_\alpha}(\overline{\mathbf{E}}(t,x)+\mathbf{v}\times\overline{\mathbf{B}}(t,x))\cdot\frac{\partial}{\partial \mathbf{v}}\right\}f_\alpha = C_\alpha(f_\alpha)$$

が用いられます．ここで，α は粒子種を表し，$\overline{\mathbf{E}}$ および $\overline{\mathbf{B}}$ は外場とプラズマの集団運動が作り出す電磁場とします．N 個のNewton方程式からBoltzmann方程式を導出する議論については，文献[3-5]を参照ください．上式の右辺は，2体相互作用（Coulomb衝突）によって起こる速度空間での拡散を表現するLandau衝突項で，Fokker-Planck型の式（＊1）として記述されます[3]．さらに，左辺において，荷電粒子のジャイロ運動に対して近似的な取り扱いを行うことで，5次元位相空間の1体粒子分布関数の運動論方程式（近似の取り方によって，ジャイロ運動論方程式，またはドリフト運動論方程式）を得ることができます[5,6]．この運動論方程式によって，現実的にプラズマの諸現象を扱うことができるようになります．

プラズマの振る舞いは，運動論方程式によって記述されるわけですが，次に，シミュレーションにおいてなぜ乱数が用いられるのかをLandau衝突項に注目して考えていきます．例えば，プラズマ性能を決める主な要因の1つである微視的乱流輸送を扱うシミュレーションにおいては，運動論方程式の主に左辺の寄与により生じる1体粒子分布関数の微細揺動に対する数値計算の高精度化のために，分布関数そのものを5次元の連続体として扱うことが多く[7]，Landau衝突項は，速度空間の微積分方程式として，差分スキームなどを使って数値的に解きます．このような微視的乱流シミュレーションは，理化学研究所のスーパーコンピュータ「京」などの大規模計算機を利用して100時間程度の計算時間を必要とします[8,9]．一方，最近の研究[10]で，プラズマ性能を劣化させる不純物の輸送において，Coulomb衝突の寄与による輸送現象（新古典輸送）の重要性が指摘されていますが，そのような現象に対するシミュレーションにおいては，微視的乱流シミュレーションほどの速度空間分解能は要求されていないので，計算コストの低減を優先した粒子的なシミュレーション手法を用いることが多いです[11]．その場合，Landau衝突項による拡散現象は，分布を構成するテスト粒子の速度空間におけるランダムウォークによってモデル化されます．そのようなCoulomb衝突の効果を考慮したシミュレーションにおいて，粒子のランダムウォークを具現化するため，乱数が用いられるのです．このような乱数の利用に対する簡単な例を5.2.1節で紹介します．また，このシミュレーション手法を発展させると，科学技術分野で扱われることが多いDirichlet-Poisson混合問題に適用することができ，そのような応用についても紹介します．

上記のような運動論方程式を解く際には，多くの場合，プラズマを閉じ込めている磁場や，プラズマの温度および密度の分布は，既知である，もしくは，初期に与えたものに近いと仮定しています．一方，プラズマの力学的平衡状態における磁場や，温度・密度分布そのものは，計算コストを考えて，通常，Boltzmann方程式から導かれる3次元空間の流体方程式とMaxwell方程式を自己無撞着に解くことで求めます[12]．この流体方程式に基づいて温度・密度分布を決定する輸送方程式が非線形偏微分方程式であるために，解を数値的安定に求めることが困難になる場合があります[13]．そのような場合に対処するため，5.2.2節で紹介する遺伝的アルゴリズムを用いた手法が開発されています．この場合，乱数は，物理現象のモデル化に用いるというより，非線形偏微分方程式を解くアルゴリズムにおけるツールとして使用されているわけです．

以上のように，乱数は，運動論方程式を解く際の膨大な計算コストを減じるためのモデリングにおいて利用されたり，流体方程式により記述される非線形な輸送方程式を解くために利用されたりしています．

（菅野・本多）

5.2.1　拡散過程とその応用

本節では，乱数を用いた基礎的なモンテカルロシミュレーションの代表的な例として，拡散方程式の解法とDirichlet-Poisson混合問題への適用の2つを取り上げ，乱数がどのように利用されているのか紹介します．紹介するシミュレーション手法の数学的基盤は，確率過程論です．「確率過程」とは，時間と共に変化する確率変数（確率的に値が定まる変数）です．この後に示すように，シミュレーションにおける乱数の役割は，数値計算上の単なるツールではなく，確率過程論に基づいて拡散現象を粒子的にモデリングしたときに導かれる数学的性質の具現化を担っています．

以下では，確率過程論における数学的結果（定理など）を用いた手法の説明に重点を置きますので，数学用語の厳密な定義や数学的結果の導出については，文献[14-17]などを参照ください．なお，本節で扱う手法では，簡単のため，空間を1次元としますが，n 次元空間（$n>1$）へ拡張できます．

● 拡散方程式の解法

プラズマ・核融合分野におけるモンテカルロシミュレーションの代表的なものの1つは，拡散方程式の解法ではないでしょうか．ここでは，荷電粒子のCoulomb衝突や，熱拡散などのランダムネスが内在する物理現象に対する粒子的なモデリングに基づいたシミュレーション手法を紹介します．以下のような拡散方程式（Fokker-Planck方程式）を解くことを考えることにします．

$$\frac{\partial}{\partial t}f(t,x)=-\frac{\partial}{\partial x}[A(x)f(t,x)]+\frac{1}{2}\frac{\partial^2}{\partial x^2}[D(x)f(t,x)] \quad (1)$$

ここに，時間パラメータを $t\geq 0$，空間を x とし，$t,x\in\mathbf{R}$ で，$\mathbf{R}$ は実数とします．また，$A(x)$，$D(x)\in\mathbf{R}$ および $D(x)\geq 0$ です．式(1)を解くため，分布 f を構成する粒子の位置 $x(t)$ という量を導入し，$x(t)$ の時間変化の式を

（＊1）　Fokker-Planck型とは，どのようなものかについては，5.2.1節で紹介します．

$\mathrm{d}x(t)/\mathrm{d}t = A(x(t)) +$ノイズ項，と表現しましょう．ここで，この「ノイズ項」から生じるランダムウォークの歩幅が拡散係数 $D(x)$ を与えるように設定します．多数の粒子に対して，この式を解くと，時刻 t におけるそれぞれの粒子の位置 $x(t)$ が求まりますので，粒子の位置分布も得られ，それが式(1)の解です．数学的に，もう少し厳密に言えば，式(1)の解 $f(t,x)$ を求めるために，式(1)に対する粒子描像の微分方程式：

$$\mathrm{d}x(t) = A(x(t))\mathrm{d}t + \sigma(x(t))\mathrm{d}W(t) \qquad (2)$$

に注目し，この式を解いて得られる「確率過程 $x(t)$」の分布が解 $f(t,x)$ であること（＊2）を利用します．ここで，$D(x) = \sigma(x)\sigma(x)$ です．この確率過程 $x(t)$ は，拡散過程とも呼ばれます．式(2)の右辺第2項における $W(t)$ は，数学における意味での「Brown 運動」で，Wiener 過程とも呼ばれます［14］．Wiener 過程 $W(t)$ は，①連続で，②離散時刻 $0 = t_0 < t_1 < t_2 < \cdots < t_{k-1} < t_k$ を適当に取って，$W(0) = 0$ として，任意の時刻 t_k に対して $W(t_k) - W(t_{k-1})$ が平均0で分散 $(t_k - t_{k-1})$ の正規分布 $N(0, t_k - t_{k-1})$ に従い，③また，$k \neq j$ として $W(t_k) - W(t_{k-1})$ と $W(t_j) - W(t_{j-1})$ が互いに独立となるという性質を持ちます[14]．式(2)の「ノイズ項」において $\mathrm{d}W(t)/\mathrm{d}t$ のように書かない理由は，$W(t)$ が至るところ微分不可能であるためです．

式(2)のような方程式は，確率微分方程式と呼ばれ，日本の数学者，伊藤 清氏によって，1942年に数学的基礎が与えられました[18]．式(2)は，確率過程 $x(t)$ を求めるための「積分」を表現していて，

$$x(t) = x(0) + \int_0^t A(x(s))\mathrm{d}s + \int_0^t \sigma(x(s))\mathrm{d}W(s) \qquad (3)$$

のように与えられます．式(3)の右辺第3項は，伊藤積分と呼ばれ，$\sigma(x(s))$ における時刻 s の取り方に気を付けて，

$$\int_0^t \sigma(x(s))\mathrm{d}W(s) = \lim_{\Delta t \to 0} \sum_{k=0}^{n-1} \sigma(x(t_k))\{W(t_{k+1}) - W(t_k)\} \qquad (4)$$

のように定義されます．後で数値計算することを意識して，$\Delta t = t/n$ とし，$t_k = k\Delta t$，$W(t_{k+1}) = W(t_k) + \Delta W$ とします．ここで，ΔW は，平均0で分散1の正規分布 $N(0,1)$ に従う正規乱数 γ を用いて，$\Delta W = \gamma\sqrt{\Delta t}$ と表現できます．確率微分方程式(2)の解の存在とその一意性のため，$A(x)$ および $\sigma(x)$ の満たすべき条件は，適当な定数 C_0 および C_1 に対して

$$|A(x)| + |\sigma(x)| \leq C_0\{1 + |x|\} \qquad (5a)$$

および

$$|A(x) - A(y)| + |\sigma(x) - \sigma(y)| \leq C_1|x - y| \qquad (5b)$$

です．ここで，$x, y \in \mathbf{R}$ です．詳しくは，文献[14, 15]などで論じられていますので，そちらを参照ください．

具体的な例で，拡散方程式を解くモンテカルロシミュレーションがどのようなものかを見ていきましょう．プラズマ中のテスト粒子（例えば，電子とします）が，背景プラズマ（水素イオンとします）との Coulomb 衝突により，速度空間をピッチ角散乱する場合を考えます．電子の速さを v，磁力線方向の速度を $v_\parallel$ として，$\xi = v_\parallel/v$ で速度空間を表現することにします．ξ は，電子のピッチ角 α（速度ベクトルと磁力線のなす角）の余弦 $\cos\alpha$ を表します．ここで，ピッチ角散乱によって速さ v は変化しないと仮定しています．ξ の取り得る値は，$-1 \leq \xi \leq 1$ です．電子の速度空間 ξ における分布 f を与える拡散方程式は，以下のように表現されます[19]．

$$\frac{\partial}{\partial t}f(t,\xi) = -\frac{\partial}{\partial \xi}[-\nu_{\mathrm{ei}}\xi f] + \frac{1}{2}\frac{\partial^2}{\partial \xi^2}[\nu_{\mathrm{ei}}\{1-\xi^2\}f] \qquad (6)$$

簡単のため，ここでは，電子-イオン衝突周波数 ν_{ei} を定数と仮定します．式(6)に対応する拡散過程 $\xi(t)$，つまり，テスト粒子の速度空間における位置の時間発展は，時間ステップを Δt として，式(2)のように与えられる確率微分方程式から自然に導かれる計算スキーム：

$$\xi(t + \Delta t) = \xi(t) - \nu_{\mathrm{ei}}\xi(t)\Delta t + \sqrt{\nu_{\mathrm{ei}}\{1 - \xi^2(t)\}}\Delta W \qquad (7)$$

により与えることができます．ここで，$\mathrm{d}\xi(t) \approx \xi(t + \Delta t) - \xi(t)$ としました．また，テスト粒子は，時刻 $t = 0$ で位置 $\xi(0)$ から出発するとします．式(7)のような計算法は，Euler-丸山スキームと呼ばれています[20, 21]．もちろん，常微分方程式の解を求める数値計算スキームと同様に，Runge-Kutta スキーム（後で紹介します）など高次の近似法もあります[20-22]．ここでは，Euler-丸山スキームを利用することにします．

式(7)のような計算スキームは，確率微分方程式の時間離散近似です．時間離散近似には，大別して，確率微分方程式の解の経路 $\{\xi(t)\}_{t \geq 0}$ をよく近似する「強い近似」と解の平均や分散などの分布特性を近似する「弱い近似」があり［20］，ΔW に対して，強い近似では正規乱数 γ を用いて $\Delta W = \gamma\sqrt{\Delta t}$，弱い近似では，例えば，$\Delta W = \pm\sqrt{\Delta t}$ のように取ります．ここで，弱い近似における“±”は，コイン投げなどで発生させた，値が $+1$ または -1 の二値乱数を用いて，それぞれ確率 1/2 で与えられるとします．（弱い近似においては，ΔW に対して他の取り方もあります．）どちらの近似を採用するかは，この後に示すように，問題ごとに適切に判断すればよいでしょう．

拡散方程式(6)を数値シミュレーションで解く場合には，Euler-丸山スキーム（7）における強い近似の採用は，あまり有効ではないことがわかっています．以下では，このことについて説明します．式(7)において強い近似を採用すると，数値的に $|\xi| > 1$ となる不具合が生じます．すなわち，数値計算において Δt は有限な値であるので，1ステップの計算の際に発生させた正規乱数の絶対値が，ある値 γ_0 より大きくなると $|\xi| > 1$ が発生します．ここで，γ_0

（＊2）証明については，例えば，文献[14]の問題8.3を参照ください．この後で紹介する，式(10)を用いる計算法との関係も説明されています．

は，ある時刻 t における ξ の値を $\xi(t)=\pm(1-\varepsilon)$ として，

$$\gamma_0=\frac{(1-\theta)\varepsilon+\theta}{\sqrt{\theta\varepsilon(2-\varepsilon)}} \tag{8}$$

です．ただし，$0<\theta=\nu_{\mathrm{ei}}\Delta t<1$，$0<\varepsilon<2$ で，また，$\xi=\pm(1-\varepsilon)$ の正符号は $\xi>1$ となるケース，負符号は $\xi<-1$ となるケースで選ぶこととします．$\xi=1-\varepsilon$ から出発するテスト粒子の1ステップの計算で不具合（$\xi>1$）が発生する確率 P_{err} は，$P_{\mathrm{err}}=(1/2)\mathrm{erfc}(\gamma_0/\sqrt{2})$ です．出発点を $\xi=-1+\varepsilon$ とした場合も同じです．ここに，$\mathrm{erfc}(x)=1-\mathrm{erf}(x)$ で，$\mathrm{erf}(x)$ は，誤差関数です．発生確率 P_{err} から，不具合は，$|\xi|=1$ の近傍で比較的起こりやすいことがわかります（＊3）．時間ステップ Δt を小さくすることで，1ステップ毎の発生確率 P_{err} は小さくできますが，ゼロにはならないので，根本的に解決したとは言えません．数学的に $\Delta t\to 0$ の極限を取れば，$P_{\mathrm{err}}\to 0$（つまり，式(7)を確率微分方程式として見たときには測度ゼロのイベント）となりますが，シミュレーションにおいてはゼロでない確率で発生するレアイベントとなります．そのため，$|\xi|>1$ となった場合の処理をどうするか考える必要がありますし，どのような処理でも，拡散方程式(6)には含まれていない操作なので，$|\xi|=1$ 近傍のテスト粒子の分布への影響が心配です．また，拡散方程式(6)の基となるLandau衝突項のモデリングでは，小角度散乱を前提としていますので[3,19]，有限な Δt の場合に，$\Delta W=\gamma\sqrt{\Delta t}$ における γ の値によって散乱効果が大きくなりすぎること自体，レアイベントとは言え，モデリングの前提に反するのではないかという疑問もあります．

以上の議論から，この問題では，Euler‐丸山スキームにおける強い近似の採用は，あまり有効ではないことがわかりました．そこで，文献[23]に従い，弱い近似で扱うことにし，$\Delta W=\pm\sqrt{\Delta t}$ として，

$$\xi(t+\Delta t)=\xi(t)-\xi(t)\nu_{\mathrm{ei}}\Delta t\pm\sqrt{\{1-\xi^2(t)\}\nu_{\mathrm{ei}}\Delta t} \tag{9}$$

のように計算することにします．ここで，± については，例えば，1ステップごとに0から1の範囲の一様乱数をTausworthe（トーズワース）法で発生させ，その値が1/2未満なら負符号，1/2以上なら正符号とします．式(9)のように計算すれば，$|\xi|>1$ となる不具合は起こりません．時刻 $t=0$ で初期分布 $f(0,\xi)$ となるようにテスト粒子の初期位置 $\xi(0)$ を設定すれば，式(9)によって与えられる時刻 t におけるテスト粒子の位置の分布が，拡散方程式(6)の数値解 $f(t,\xi)$ になります．時間ステップ Δt を十分小さく取れば，分布の計算に関して，式(9)は，確率微分方程式として見たときの式(7)の良い近似となり，シミュレーションにおいて，分布 $f(t,\xi)$ は，初期分布 $f(0,\xi)$ に依らず，1衝突時間 τ_{ei} $(=\nu_{\mathrm{ei}}^{-1})$ 以降に理論通り一様な分布に緩和します．ただし，どのような初期分布であっても，$\xi(t)$ の数値計算における最初の1ステップ以降は，テスト粒子は $[-1+\delta,1-\delta]$ の範囲に存在することになります．ここで，$\delta=\nu_{\mathrm{ei}}\Delta t/2$ です[23]．また，有限個のテスト粒子を用いた手法なので，得られた分布に統計的な誤差は生じます．統計誤差は，テスト粒子数 N_{t} に対し，$1/\sqrt{N_{\mathrm{t}}}$ に比例して小さくなります．

以上が，テスト粒子の位置分布によって拡散方程式の解を与えるモンテカルロシミュレーションの紹介ですが，これとは別の計算法もあります．関数 $u(t,x)$ を $t>0$ で，

$$u(t,x)=\mathrm{E}^x[\Phi(x(t))] \tag{10}$$

であると定義すると，このとき，$u(t,x)$ に対して以下の関係式が成り立つことを利用します[14]．

$$\frac{\partial u}{\partial t}=\mathcal{L}u \text{ および } u(0,x)=\Phi(x) \tag{11}$$

ここで，演算子 $\mathcal{L}$ は，式(2)に対応して，

$$\mathcal{L}=A(x)\frac{\partial}{\partial x}+\frac{1}{2}D(x)\frac{\partial^2}{\partial x^2} \tag{12}$$

であり，$\mathrm{E}^x[\Phi(x(t))]$ は，式(2)に従って時刻 $s=0$ に位置 x から出発した確率過程 $\{x(s)\}_{s\ge 0}$ の分布による $\Phi(x(t))$ の期待値で，この場合には，式(2)が与える時刻 t における x から y への遷移確率密度を $p(t,x,y)$ として，$\int\Phi(y)p(t,x,y)\mathrm{d}y$ と表現することもできます．最初に紹介した計算法と式(10)を用いる計算法は，Kolmogorovの前進方程式と後退方程式の関係になっています[14]．

先ほどのピッチ角散乱の問題に適用してみましょう．解くべき方程式は

$$\frac{\partial}{\partial t}f(t,\xi)=-\nu_{\mathrm{ei}}\xi\frac{\partial f}{\partial\xi}+\frac{1}{2}\nu_{\mathrm{ei}}\{1-\xi^2\}\frac{\partial^2 f}{\partial\xi^2} \tag{13}$$

で，対応する拡散過程は，式(7)（数値計算する際には，式(9)）です．例えば，初期分布を $f(0,\xi)=\Phi(\xi)=\{2-\sin(\pi\xi)\}/4$ であるとすると，各時刻 t における解 $f(t,\xi)=\mathrm{E}^{\xi}[\Phi(\xi(t))]$ は，**図1**のようになります．ここで，速度空間の位置 ξ における時刻 t の分布の値 $f(t,\xi)$ は，式(9)により与えられる，時刻 $t=0$ に ξ から出発したテスト粒子の時刻 t での位置 $\xi(t)$ における $\Phi(\xi(t))$ の平均値です．当然のことながら，**図1**の分布は，時刻 $t=0$ でテスト粒子を初期分布 $\Phi(\xi)$ となるように配置し，式(9)によって時間発展させたテスト粒子の位置の分布と同じものになります．最初に紹介した計算法の方が，シンプルでわかりやすく感じるかもしれませんが，式(10)を用いる計算法の考え方は，より一般的な初期値境界値問題への応用に発展させることができ，次に紹介する手法とも関連しています．詳しくは，例えば，文献[15]で論じられていますので，そちらを参照ください．

● Dirichlet-Poisson 混合問題への適用

この節の後半では，科学技術分野で扱われることが多い

（＊3）出発点を $|\xi|=1$ とした場合は，$|\xi|>1$ となる不具合は発生しませんが，式(5b)を満たしていないため，解の一意性に問題があるように見えます．しかしながら，後で示す式(9)に従って，有限な時間ステップ Δt で数値的に解く際には，右辺第2項があるために，そのような問題は生じません．

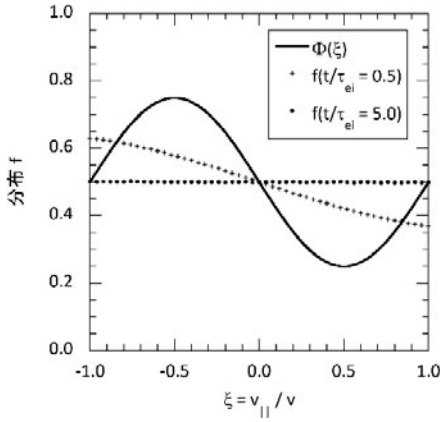


図1 ピッチ角散乱による電子分布 $f(t,\xi)$の時間発展を式(10)に基づき計算．このシミュレーションにおいて，$\nu_{ei}\Delta t = 10^{-3}$ と設定し，$-1\le\xi\le1$ の範囲の格子点（分割数50）のそれぞれに対して，10^5 個のテスト粒子を用いて期待値 $E^{\xi}[\phi(\xi(t))]$ を求めました．実線は，初期分布 $\phi(\xi)=\{2-\sin(\pi\xi)\}/4$，記号＋は時刻 $t/\tau_{ei}=\nu_{ei}t=0.5$ における分布，記号●は時刻 $t/\tau_{ei}=5.0$ における分布です．

Dirichlet-Poisson 混合問題に対して，モンテカルロ法をいかに適用するか紹介します．次の方程式を考えます．

$$\{\mathcal{L}+\eta(x)\}u(x)=F(x) \tag{14}$$

ただし，演算子$\mathcal{L}$および境界条件は，以下の通りとします．

$$\mathcal{L}u=\frac{1}{2}D(x)\frac{\partial^2 u}{\partial x^2}+A(x)\frac{\partial u}{\partial x} \tag{15a}$$

境界条件：$x=x_0,x_1$ において，$u(x)=G(x)$ (15b)

ここで，空間 x については，$x\in[x_0,x_1]$ を仮定しています．式 (15a) から，これに対応する確率微分方程式は，次のようになります[14, 15]．

$$\mathrm{d}x(t)=\sigma(x(t))\,\mathrm{d}W(t)+A(x(t))\,\mathrm{d}t \tag{16}$$

ただし，$D(x)=\sigma(x)\sigma(x)$ です．方程式(14)の解は，式(16) に従う確率過程 $x(t)$ により

$$u(x)=\mathrm{E}^x\left[\mathrm{G}(x(\tau))\exp\left\{\int_0^\tau\eta(x(s))\,\mathrm{d}s\right\}\right]-\mathrm{E}^x\left[\int_0^\tau F(x(s))\exp\left\{\int_0^s\eta(x(r))\,\mathrm{d}r\right\}\mathrm{d}s\right] \tag{17}$$

と与えられます[14, 15]．ここで，$\mathrm{E}^x[\cdot]$ は，式(16)に従って，時刻 $t=0$ に位置 x から出発した確率過程 $\{x(t)\}_{t\ge0}$ の分布による期待値を意味します．また，τ は，式(16)に従って動くテスト粒子（モンテカルロ法におけるサンプル粒子）それぞれに対して，区間 $[x_0,x_1]$ の外へ最初に出た時刻によって与えられます．式(17)により数値解 $u(x)$ を求めるには，すべてのテスト粒子それぞれに対する時刻 τ の値が必要となるので，式(16)の $A(x)$ の値などから，計算時間がどれくらいか推定できても，乱数を用いてシミュレーションしているために，計算時間を正確に定めることはできないという欠点があることに注意ください．

式(17)を用いる計算法では，各テスト粒子の経路 $\{x(t_k)\}_{k=1,2,3,\cdots}$ と境界に最初に到達した時刻 τ が重要であり，Runge-Kutta スキームを採用することで，これらを与える式(16)に対する計算精度を向上させることができます．ここで，式(16)に対する Runge-Kutta スキームは，以下の通りです[21, 22]．

$$x(t+\Delta t)=x(t)+\frac{1}{2}\Delta t(A_1+A_2)+\frac{1}{2}\sqrt{\Delta t}\,(\sigma_1+\sigma_2)\gamma_1 \tag{18a}$$

$$A_1=A(x(t)) \tag{18b}$$

$$\sigma_1=\sigma(x(t)) \tag{18c}$$

$$A_2=A(x(t)+A_1\Delta t+\sigma_1\sqrt{\Delta t}\gamma_2) \tag{18d}$$

$$\sigma_2=\sigma(x(t)+A_1\Delta t+\sigma_1\sqrt{\Delta t}\gamma_2) \tag{18e}$$

ただし，γ_1 と γ_2 は，$N(0,1)$ に従う互いに独立な正規乱数です．これらの正規乱数は，例えば，Tausworthe 法で発生させた0から1の範囲の一様乱数を使って，5.1節で説明した Box-Muller 法で生成します．

以下の例題に適用してみましょう．

$$\left(x^3+8x^2+\frac{1}{2}x\right)\frac{\mathrm{d}^2f}{\mathrm{d}x^2}-(4+x)\frac{\mathrm{d}f}{\mathrm{d}x}-2(1+x)f=5x(1-x)-2x \tag{19}$$

ただし，$x\in[0,1]$ として（$x_0=0,x_1=1$），境界条件は $G(0)=1$ および $G(1)=0$ とします．式(19)の解は，$f(x)=1-(x/2)-(x^2/2)$ です．シミュレーションの結果を**図2**に示します．ここで，**図2**では，得られた数値解の平滑化のため，最小二乗法を用いた多項式近似を行っています．多項式の項数の決定には，AIC（Akaike Information Criterion，赤池情報量規準）[24]を利用しました．AIC は，統計モデルの良さを評価し，良いモデルを選択するための指標です．数値的に得た解 f を微分したい場合には，そのままでは数値解の統計誤差がノイズになるので，AIC などを利用しながら数値解を平滑化することが有効です．また，得られた数値解の妥当性は，異なる乱数列を使って複数回，数値解を求めて確認します．例えば，2回計算して得られた数値解をそれぞれ f_1，f_2 として，以下のような相

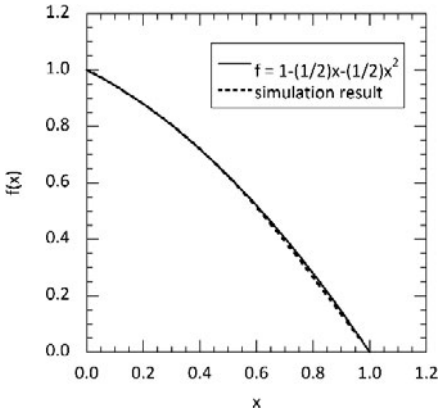


図2 方程式(19)の解析解と数値解．解析解を実線，数値解を破線で示しています[25]．数値解は，式(17)に基づき計算．区間[0, 1]における格子点（分割数100）のそれぞれに対して 10^3 個のテスト粒子を用いて，数値解を求めました．また，$\Delta t=10^{-4}$ と設定．得られた数値解の相対誤差は，$\epsilon<1/200$ でした．

対誤差

$$\epsilon = \int_{x_0}^{x_1} |f_1 - f_2| \, \mathrm{d}x \Big/ \int_{x_0}^{x_1} |f_1| \, \mathrm{d}x \tag{20}$$

が十分小さいなら，数値解として妥当と言えます．

例題(19)では解がわかっていましたが，もちろん，事前に解がわからない問題を解くのが通常です．第4章で議論した，乱数の統計的品質が良いと判定される疑似乱数を利用していれば，疑似乱数の性質が計算結果に影響することは一般的には考えづらいです．それでも乱数の検定法で見抜けないような，疑似乱数の隠れた非ランダム性が計算結果に影響を与える可能性を排除したいのであれば，物理乱数を利用することで，数値解の妥当性を判断する際に，そのような影響を考えなくてよくなります．

ところで，例題(19)では，式(17)における指数関数の引数に含まれる η が負の値で，計算の実行に問題は無かったのですが，η が正の値を取る場合には，式(17)のままでは不具合（テスト粒子によっては，指数関数部分の数値計算が破綻するなど）が発生することがあります．そのような不具合への対処の1つとして，式(14)において $\eta(x)u(x)$ を右辺に持っていき，ソース項 $F(x)-\eta(x)u(x)$ として処理すると，数値解が落ち着くまで再帰的に計算を行う必要はありますが（つまり，ソース項における $u(x)$ として，最初は適当な近似解を与え，それにより得られた近似解をまたソース項の $u(x)$ に代入して計算を繰り返すことで），うまく克服できる場合があります[25]．例として，以下のような方程式を考えます．

$$\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} + \pi^2 f = 0 \tag{21}$$

ただし，$x \in [0,1]$ として，境界条件は $G(0)=1$ および $G(1)=-1$ とします．式(21)の解析解は，$f(x)=\cos(\pi x)$ です．計算結果を**図3**に示します．ここで，$f(x)$ として最初に与えた近似解は，$f(x)=-2x+1$ です．シミュレーションにおけるその他の設定は，**図2**と同じです．**図3**に示したように，式(17)で $F(x)=0$ かつ $\eta=\pi^2$ とすると計算に失敗し，$F(x)=-\pi^2 f(x)$ かつ $\eta=0$ とすれば解析解と良く一致する数値解が得られました．もちろん，例題(19)においても，左辺第3項を右辺に持っていき，ソース項として扱って問題無く解くことができます[25]．

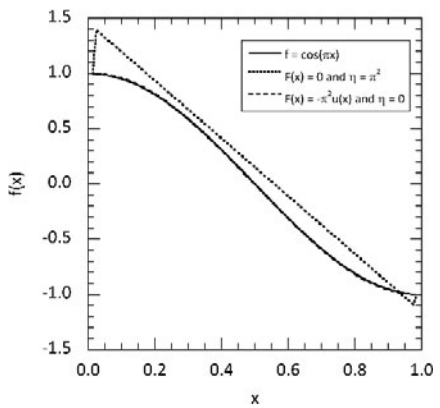


図3　方程式（21）の解析解と数値解．解析解を実線，式(17)で $F(x)=0$ かつ $\eta=\pi^2$ とした場合は点線，式(17)で $F(x)=-\pi^2 f(x)$ かつ $\eta=0$ とした場合の数値解（得られた数値解の相対誤差 ϵ = 1/100）を破線で示しています[25]．

以上のように，本節では，確率過程論に基礎づけられたモンテカルロシミュレーションを紹介しました．プラズマ・核融合分野では，ここで紹介したもの以外に，いろいろなモンテカルロ手法が開発・利用されています．例えば，文献[26-44]などです．文献[26-31]は Coulomb 衝突のモデリング，[32-38]は運動論方程式の解法，[39-42]は流体方程式の解法，[43]はプラズマ - 壁相互作用研究における動的モンテカルロ法，[44]は中性子・光子輸送に関して書かれています．興味を持たれた方は，これらの文献を参照ください．

（菅野）

5.2.2　輸送方程式求解への応用

プラズマ輸送研究や統合モデリングにおいて，プラズマの巨視的な時間発展や定常状態における分布を求めるために輸送コードが広く用いられています．とりわけ，運転シナリオ開発や実験解析においては定常状態を対象とすることが多く，非定常輸送コードでプラズマの分布が変化しなくなるまで時間発展させて定常状態の分布を求めています．輸送コードは放物型偏微分方程式である1次元の輸送方程式に基づいており，通常は有限差分法や有限要素法によって時空間方向に離散化して数値的に解きます．温度分布を求めたい場合，圧力（内部エネルギー）を従属変数とした輸送方程式を解きます．拡散係数（輸送係数とも言います）がプラズマの内部状態に対して不変である場合，またはその応答が線形である場合は求解は容易です．しかし，現実のプラズマの輸送は新古典輸送や乱流輸送によって支配され，それらは温度勾配などの熱力学的力に非線形に依存しています．そのため，輸送方程式は非線形の偏微分方程式となり，解析的に一般解を求めることは困難ですし，数値的に解けたとしても数値的安定に求めることも困難になります．従来型解法の枠内で数値的安定に定常解を求めるアルゴリズムが提案されてきましたが[45,46]，期待した結果が得られないこともしばしばです．

本節では，輸送方程式を解いてプラズマの定常分布を求めるという問題を，定常状態を満たす温度と温度勾配の適切な組み合わせを求める大域的最適化問題へと転換し，その解法としてメタヒューリスティックなアルゴリズムである遺伝的アルゴリズム[47,48]を適用した例を紹介します．ここまで読まれてきて，この話が一体どう乱数と関係するのか疑問に思われたかもしれませんが，遺伝的アルゴリズムの中で乱数は重要な働きをします．以下では，まず遺伝的アルゴリズムの簡単な紹介をしたのちに，遺伝的アルゴリズムを用いた輸送方程式の求解法へと進んでいきます．

●遺伝的アルゴリズムとは

遺伝的アルゴリズムは進化的アルゴリズムの中でも代表的なものであり，自然界における自然淘汰，遺伝，交叉，突然変異などの生命の進化過程を数値的アルゴリズムで模したものになります．遺伝的アルゴリズムでは必然的に進化

遺伝学の用語が多く使われますが，あくまで数値的な最適化アルゴリズムであり，生物学の文脈で使われる概念とは必ずしも厳密に一致しませんのでその点はご留意ください．一口に遺伝的アルゴリズムと言っても実装は様々かと思いますが，以下ではPIKAIA[49,50]というFortran言語における実装を基に，遺伝的アルゴリズムがどのようにして大域的最適解を求めるのかを見ていきましょう．

最適解，すなわち解を求めるためには，当然ですが最適さの基準が必要になります．この品質の尺度を適応度と呼び，適応度を求めるための関数を適応度関数$f(\boldsymbol{x})$と言います．大域的最適化問題を解くとは，パラメータ空間$\boldsymbol{x}\in[0,1]$で$f(\boldsymbol{x})$を最大化するパラメータ$\boldsymbol{x}$を見つけることを言います．なお，$\boldsymbol{x}$は適当な規格化によって$[0,1]$に収まるようにしています．

まず，これから生存競争を繰り広げる個体（表現型とも言います）の集団を生成します．扱う問題にも依りますが，一般的に多くのテスト粒子数を必要とするモンテカルロ法とは異なり，個体数は典型的に100程度です．アルゴリズムの流れは以下の通りです．(1)パラメータ空間の中で個体をランダムに生成し，各個体の適応度を評価します．(2)次に，適応度に応じて現在の集団からつがいを選択し交配させ，次世代の集団を作ります．(3)次世代の集団を現世代と置き換え，(4)新しい集団の個体に対して適応度を評価します．集団の中で最適な（最大の適応度を持つ）個体が，設定した基準を満たすか設定した世代数に到達するまで，(2)-(4)の過程を繰り返します．最終世代の集団で最適な個体の持つパラメータ$\arg\left(\max_{\boldsymbol{x}\in[0,1]} f(\boldsymbol{x})\right)$が最適解となります．

以上が遺伝的アルゴリズムの“アルゴリズム”になりますが，これだけだと乱数との関わりがはっきりしないかも

乱 数 茶 話

5.2.1節で紹介したモンテカルロシミュレーションでは，拡散方程式を解く際に，対応する確率微分方程式に従うテスト粒子の運動を利用しました．例えば，拡散方程式(13)に対応した確率微分方程式（の計算スキーム）として式(7)(数値計算する際には，式(9))を用いましたが，これ以外にも式(13)に対応する確率微分方程式はあるのでしょうか？このことに関して，文献[14]に興味深い問いが載っています．それは，「いつ伊藤過程は拡散過程となるか？」です．伊藤過程とは，次のように与えられる確率過程$y(t)$を指します．より正確な定義は，文献[14]を参照ください．

$$\mathrm{d}y(t)=\alpha(t)\mathrm{d}t+\beta(t)\mathrm{d}W(t)$$

ただし，$\alpha(t)$および$\beta(t)$も確率過程で，任意の時刻$t\geq 0$に対して確率1で以下を満たすとします．

$$\int_0^t|\alpha(s)|\mathrm{d}s<\infty \text{ および } \int_0^t\beta(s)^2\mathrm{d}s<\infty$$

文献[14]による「問い」への答えですが，以下の拡散過程$x(t)$

$$\mathrm{d}x(t)=A(x(t))\mathrm{d}t+\sigma(x(t))\mathrm{d}W(t)$$

と$y(t)$との間に，$\alpha(t)$に対する次の条件付き期待値について

$$\mathrm{E}^x[\alpha(t)|\mathcal{F}_t]=A(y^x(t))$$

かつ，$\beta(t)$について

$$\beta(t)\beta(t)=\sigma(y^x(t))\sigma(y^x(t))$$

が，ほとんどすべての時刻t，およびほとんどすべてのテスト粒子に対して成立する場合，$x(t)$と$y(t)$は，法則の意味で一致する（つまり同じ分布を持つ）というのです．ここで，$y^x(t)$は$t=0$で位置xから出発した$y(t)$を意味し，$\mathcal{F}_t$は$\{y(s);s\leq t\}$の生成するσ-加法族[14]です．

試しに，拡散方程式(13)で考えてみましょう．$\mu(t)$を$-1/10$から$+1/10$の範囲の一様乱数として，次のように式(9)に加えてみます．

$$\xi(t+\Delta t)=\xi(t)-\{\xi(t)+\mu(t)\}\nu_{\mathrm{ei}}\Delta t\pm\sqrt{\{1-\xi^2(t)\}\nu_{\mathrm{ei}}\Delta t}$$

この式を用いて，$\mathrm{E}^{\xi}[\Phi(\xi(t))]$を計算すると，例えば，$t/\tau_{\mathrm{ei}}=0.5$における分布は，**図4**のように**図1**と同じ結果になります．$|\xi|>1$とならない範囲で，$|\mu(t)|$の最大値を変えてみても，結果は変わりません．つまり，このような一様なノイズは，分布$f(t,\xi)=\mathrm{E}^{\xi}[\Phi(\xi(t))]$に影響しません．

ここで紹介した文献[14]の数学的結果は，微視的運動から巨視的スケールの分布が従う方程式へのモデリングにとって，重要な意味を持つように思えるのですが，いかがでしょうか．

（菅野）

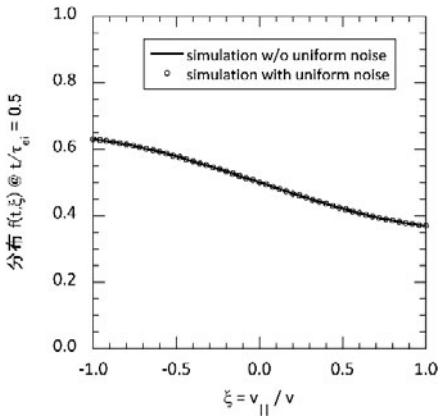


図4 ピッチ角散乱によって時間発展する電子分布に対する一様なノイズの影響．実線は図1の時刻$t/\tau_{\mathrm{ei}}=0.5$における分布，記号○は一様なノイズを加えた場合の時刻$t/\tau_{\mathrm{ei}}=0.5$における分布．一様乱数$\mu(t)$は，Tausworthe法を利用して，−1/10から+1/10の範囲で与えました．初期分布，各パラメータの設定値，およびテスト粒子の総数は，図1と同じです．

しれません．そのため，乱数との関連という観点から各段階をもう少し詳細に見ていきたいと思います．

まず，個体とは何かについて説明します．個体とは遺伝子の集合からなるものとして捉えられており，最適化したいパラメータの組の数だけ遺伝子の組があると思ってください．例えば，2次元空間(x,y)（ここで，$x,y\in[0,1]$）において$(0.5,0.5)$に最も近い$\boldsymbol{x}\equiv(x,y)$を求めるとします．念のためですが，理想的な答えは$(0.5,0.5)$なので自明です！あくまで遺伝的アルゴリズムの概念をわかりやすく伝えるために設けた問題です．適応度関数の設定方法は色々ありますが，ユークリッド距離の二乗の逆数$f(\boldsymbol{x})=[(x-0.5)^2+(y-0.5)^2]^{-1}$とすれば良いことがわかります．$x$，$y$は何桁かの数字（遺伝子）からなっており，今回の例では座標値になる訳ですが，その初期の座標値を$[0,1]$の一様乱数から生成し，個体数分だけ繰り返します．これが(1)で行っている作業です．オリジナルのPIKAIAでは線形合同法による乱数を用いていますが，筆者はそれをMersenne-Twister（メルセンヌ-ツイスター）に換装して使用しています．

(2)の過程は遺伝的アルゴリズムの中でも最も重要なものです．個体の集団の中から，1組のつがい，すなわち父と母を選択します．父と母をランダムに選択するのではなく，適応度に応じて選ばれる確率を変える必要がありそうです．そのために，まずルーレット選択と呼ばれるサンプリング手法を用いると，どうなるか見てみましょう．その方法ではまず，全個体の適応度の和Fと，個体番号が若い方から順に定義される適応度の累積分布を作成し準備しておきます．そして，一様乱数$R\in[0,F]$を生成すると，その乱数Rは必ず離散的な累積分布のどこかの区間に対応する値を取り，その区間を示すインデックス値から特定の個体番号と一対一対応が付けられるために，乱数によって個体を選択できます．ルーレット盤は個体数で分割されており，各個体の適応度の大きさによって区間の面積が決定されているイメージです（イメージしにくい方は[48]の図2.5をご覧ください）．これで問題は全くないように思いますが，実は適応度の大きさに応じて決めてしまうと，多くの問題が生じることが知られています．わかりやすい例を挙げると，1つか2つの個体が極端に高い適応度を持っておりその他がそうでない場合，このサンプリング方式ではほぼ常に適応度の高い同じ個体が選ばれてしまうことになり，最適化過程にとって重要な集団の多様性が失われてしまいます．それを避けるために，ランキング選択と呼ばれる方法が使われます．適応度の高い順に個体を並べるのですが，個体の選ばれやすさはランク（順位）の高さで決まり，適応度の値そのものでは決まらない，という方式です．つまり個体間での選択確率の差が同じである，というわけです．ランクを横軸に取り，相対適応度を縦軸に取ると，右肩下がりの直線を引くことができますが，この勾配を制御することによって，淘汰圧の強さを調整することができます．

このようにして選ばれたつがいの父をP1:$(x,y)=(0.1234,0.5678)$,母をP2:$(x,y)=(0.8642,0.7531)$としましょう．座標値は有効数字4桁で取ったので，合わせて8つの遺伝子を持った個体から各々の染色体SP1: 12345678, SP2: 86427531を作ります．このエンコード過程は自明かと思います．次に，一点交叉と呼ばれる演算を行います．両親の染色体のつがいに対して乱数によって決まった位置で染色体を切断し，付け替えます．この場合ですと，一様整数乱数$K\in[1,8]$を生成し，例えば4が選ばれたとすると，3番目と4番目の遺伝子座の間で組み替えが起こるため，染色体はそれぞれSO1: 12327531, SO2: 86445678となります．PIKAIAでは世代ごとに集団の数は増減しないため，2人の親から2人の子が生まれることになります．つまり，これらは子の染色体となります．更に，ある変異率（通常は0.005程度です）で突然変異を起こします．染色体を構成する遺伝子ごとに生成した一様乱数$R\in[0,1]$が変異率を下回った場合に，その遺伝子を一様整数乱数$K\in[0,9]$で置き換えます．これを一様な一点突然変異と呼びます．突然変異のさせ方は他にも色々ありますが，詳細な説明は割愛します．ここではSO1に対しては6番目の遺伝子が9に，SO2に対しては2番目の遺伝子が3になったとしましょう．これをデコードすると，子はそれぞれO1:$(0.1232,0.7931)$，O2:$(0.8344,0.5678)$となります．こうして生まれた子世代は一旦プールされ，親世代と同数になるまで上記の過程が繰り返されます．なお，ランキング選択方式からわかるとおり，適応度の高い個体は1つの世代で何度でも親に選ばれることがあります．

子世代に個体が出揃うと，(3)の世代交代を行います．最も簡単で広く使われている方法は，集団を完全に子世代の集団で置き換えてしまう，というものです．但し，エリート選択という手法を使う場合は，親世代で最も適応度が高い個体は無条件で子世代へと引き渡されます．交叉や突然変異によってその世代での最適個体を壊してしまうのを避けることで，収束を悪くしないための工夫です．子世代からは1個体がランダムに排除されてしまいますが，仮に選ばれた子世代の個体が親世代の最適個体よりも適応度が高い場合は，親世代の最適個体は次世代に引き継がれません．

最後に，こうして作られた子世代の適応度を評価するのが(4)です．P1，P2，O1，O2の適応度を評価すると，O2が最も大きな値を示していることがわかります．これはO2が$(0.5,0.5)$に最も近い個体であることを示しており，この例では，1世代を経てより“優秀”な子どもが生まれたことになります．参考までに，このシミュレーションの結果を**図5**に示します．**図5**では各世代において最適個体の持つ座標値が世代ごとに示されています．既に第1世代から正解にかなり近く，10世代程度で正解と見なして申し分ない精度に到達し，第25世代で完全に正解に到達します．なお，適応度関数fの定義から，$(0.5,0.5)$における適応度は無限大になりますが，数値的に工夫することで無限大を避けるようにしています．ただし，遺伝的アルゴリズムはメタヒューリスティックアルゴリズムであるため，正解に必ず到達する保証はないことに注意してください．この例では$(0.5,0.5)$にたどり着きましたが，選択する乱数

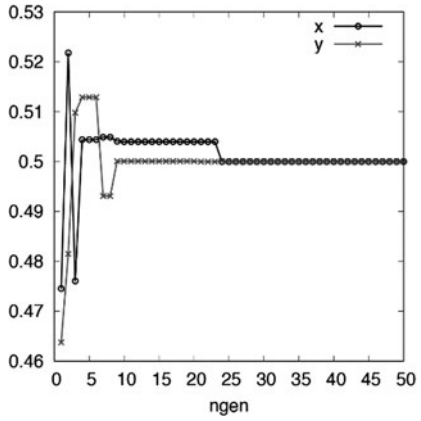


図5　各世代における最適個体の持つ遺伝子（x, y座標）．最初の10世代でほぼ正解の(0.5, 0.5)に収斂し，25世代で完全に正解と一致します．

シードによっては0.4999などで止まってしまうこともあります．しかし，それでも十分正解と見なせるでしょう．ここまで簡単に遺伝的アルゴリズムの実装を見てきましたが，数多の過程において一様乱数が用いられていることがわかるかと思います．

●大域的最適化手法を用いた輸送方程式の求解

遺伝的アルゴリズムは大域的最適化手法の一つとして位置付けられており，多峰性を持ったパラメータ空間において最適解(最大値とします)を見つけることを得意としています．多峰性の例として，$x\in[0,1]$ における正規分布の重ね合わせからなる関数 $y(x)=\sum_{j=1}^{2}A_j\exp[-(x-x_j)^2/\sigma_j^2]$ を考え，その最大値を見つけてみましょう．ここで，$[A_1,x_1,\sigma_1,A_2,x_2,\sigma_2]=[0.9,0.2,0.025,0.3,0.7,0.15]$ とします．最大値0.9は $x=0.2$ を中心とする急峻な正規分布の頂点に位置する一方，幅広い裾野を持った正規分布に極大値あります．Newton法などに代表される局所最適化手法では，極大値ではなく最大値を見つけられるかどうかはひとえに初期値の選択に掛かってきますが，問題を解く前に最大値にたどり着く適切な初期値を知っていることは稀です．局所勾配に依存する最適化手法では，$x\gtrsim0.282$ に初期値を取ると極大値0.3へと向かってしまい真の最大値0.9を見つけることはできません．言い換えると，ランダムに初期値を選んだ場合，7割強の確率で最大値を求められないことになります．大域的最適化手法は局所最適化手法と比べ一般に計算コストが掛かる一方で，アルゴリズムにも依りますが局所最適化手法よりも広くパラメータ空間を俯瞰する特徴があります．極大値近傍に適応度の高い個体が集まっている状況下でも，突然変異などで $x=0.2$ 近傍に個体が生じるとそちらの方が高い適応度を持つため，局所最適解から抜け出して最大値を見つけ出せます．

遺伝的アルゴリズムは最大値を見つける際に微分値を必要としません．そのため，微分を高精度に評価することが困難であったり，僅かな微分値の違いが結果を大きく左右してしまったりするような問題であっても，安定に解を見つけることができます．そのような問題の例としては，分布の勾配に強い依存性を持つ輸送モデルを用いる場合が挙げられます．分布の勾配という微分の精度が，輸送モデルが算出する輸送係数に大きな影響を与えてしまいます．そこで微分に依らない解法である遺伝的アルゴリズムに着目し，大域的最適化手法を用いた定常輸送方程式の解法の開発に取り組みました[18]．

以下では，簡単のため1粒子種のみを考えます．定常の輸送方程式は

$$0=-\frac{1}{V'}\frac{\partial}{\partial\rho}V'\left(-n\langle|\nabla\rho|^2\rangle\chi\frac{\partial T}{\partial\rho}\right)+S \tag{22}$$

と書けます．ρ は規格化小半径，V は体積，$'$ は ρ 微分，n は密度，χ は拡散係数，T は温度，S は加熱パワーなどに対応するソース項になります．$\langle|\nabla\rho|^2\rangle$ は平衡計算で求められるメトリックであり，ここでは既知であるとします．$\chi=\chi(T,T',n,n',\cdots)$ は一般に密度，温度やそれらの勾配などに非線形に依存しているため，式(22)は非線形の偏微分方程式となります．V,n,S が既知である場合に，この方程式を満たす T を求める，というのがここで考える問題です．ここで，式(22)の括弧内は熱流束 Q であり，

$$Q(\rho)=-n\langle|\nabla\rho|^2\rangle\chi\frac{\partial T}{\partial\rho} \tag{23}$$

を使って

$$0=-\frac{\partial}{\partial V}(V'Q)+S \tag{24}$$

と書き換えられます．式(24)を空間積分すると，

$$V'Q(\rho)=\int_0^\rho V'S\,\mathrm{d}\rho\equiv P(\rho) \tag{25}$$

となります．ある磁気面を横切って外側に流れ出る熱流束は，その磁気面より内側に吸収されたパワーと等しいことを表しています．式(23)の温度 T を変化させることで式(25)を満たすように Q を変えることから，P/V' を目標熱流束と呼びます．

では，どのように式(25)を満たす T を見つければ良いのでしょうか．通常の手法ですと，空間方向にメッシュを作成し，有限差分法や有限要素法で空間離散化し，非線形項である χ を線形化して T に対する連立一次方程式を作り，それを直接法ないし間接法で解き，得られた解（温度）を再び χ に代入して連立一次方程式を解き直し，その作業を温度が収束するまで繰り返します[51]．ここではそのような一般的な方法を採らずに解 T を求める方法を考えてみましょう．温度勾配を規格化した $1/L_T\equiv(-\partial T/\partial\rho)/T$ で式(23)を表記すると，

$$Q(\rho)=-n\langle|\nabla\rho|^2\rangle\chi(T,1/L_T)\,T\frac{1}{L_T}=\frac{P}{V'} \tag{26}$$

となります．最後の等式は式(25)から来ました．式(26)では χ の $T,1/L_T$ の依存性を陽に書きました．温度と温度勾配には物理的に現実的な定義域がありますから，そのパラメータ領域内で各 ρ において式(26)を満たす $(T,1/L_T)$ の組み合わせを見つけることができれば，それが式(22)の解と

なります．この手法だと，直接温度の微分値である$1/L_T$を探しているため，微分方程式の解を求めるのに微分が不要であることがわかると思います．

解の組み合わせを求める手法ですが，極端なことを言えば，そのパラメータ領域内を極度に細かく分割して全ての組み合わせに対して当てはめを行えば，どこかで解は見つかるでしょう．現実的にはそこまで細かな分割はできませんので，計算精度と掛かる計算時間を秤にかけて妥当な刻み幅を決めざるを得ません．非線形性の強い輸送モデルを用いる場合，僅かな勾配の差が大きな熱流束の差を生んでしまうため，先程まで極端に細かくないにしてもやはり十分細かな分割が必要になります．規格化温度勾配の定義域を$[0, 10]$として，刻み幅を0.01とかなり粗く取ったとしても，1000メッシュ必要になります．温度に対しても同じく1000メッシュ用意すると，組み合わせは10^6にも上ります．これを，空間分割数分繰り返さなくてはいけないため，膨大な試行が必要になります．さらに，この組み合わせの中には解として全くあり得そうに無い組み合わせも存在しているため，無駄の多い手法であることがわかります．そのような無駄を省くために，この解の組み合わせを見つけるアルゴリズムとして大域的最適化手法である遺伝的アルゴリズムを用いるのです．遺伝的アルゴリズムでは，最初こそモンテカルロ的にランダムに値を生成しますが，そこからはより解に近い（適応度の高い）組み合わせを持った個体がより生き残るため，急速に最適解へと収斂していきます．

熱流束を求める時は，“正解”である目標熱流束P/V'が事前にわかっていたため，Qを目標熱流束に合わせることができたのですが，正解がわかっていない温度Tはどのように求めたら良いでしょうか．一つの考え方は，$1/L_T$の値が定まったのだったら，それは温度に対する常微分方程式になるため，境界の温度さえわかれば後は全て積分で求められる，というものです．しかし，この手法は上手くいかないことがあることがわかっており[18]，別の手法を考える必要があります．式(23)で$V'Q = P$であることに留意して外側境界から積分してみましょう．すると，

$$T(\rho) = T(\rho_\mathrm{b}) + \int_\rho^{\rho_\mathrm{b}} \frac{P}{n\langle|\nabla\rho|\rangle^2 \chi(T, 1/L_T)} \frac{\mathrm{d}\rho}{V'} \tag{27}$$

という関係が得られます．$T(\rho_\mathrm{b})$は計算領域の外側境界$\rho = \rho_\mathrm{b}$での固定された温度であり，Dirichlet条件に相当するものです．式(27)は非線形の積分方程式であり，被積分関数中のχに陰に含まれるT（T_rとします）と左辺のT（T_l）が同一となるようなTを見つけなければなりません．本来このような解を見つけることは難しいのですが，大域的最適化手法によって比較的容易に見つけ出すことができます．ところで，式(27)は$1/L_T$にも陰に依存しているため，$1/L_T$も適切に定まらなければ収束解にはたどり着きません．つまり，式(26)も同時に満たさなくてはいけません．各個体の持つ$(T, 1/L_T)$に対して式(26)および(27)を計算し，得られたQとTに対して

$$f_1 = \left[\frac{P - V'Q}{P}\right]^2, \quad f_2 = \left[\frac{T_\mathrm{l} - T_\mathrm{r}}{T_\mathrm{l}}\right]^2 \tag{28}$$

を評価し，最終的に遺伝的アルゴリズムによって

$$f = [\max(f_1, f_2)]^{-1} \tag{29}$$

を最大化する個体を見つけます．その個体の持つ$(T, 1/L_T)$が最終的な式(22)の解となる訳です．f_2の意味はわかりにくいかもしれませんが，f_2が小さいとはすなわち，ある温度T_rを入力として用いて式(27)を計算した時に得られるT_lが入力のT_rとがほぼ同じである，ということを意味しています．そのようなTは式(27)を満たしている，ということがわかるかと思います．

● 計算例

上記のアルゴリズムで温度分布を解いた計算例を一つ示します．計算は，上記のアルゴリズムに基づく定常輸送コードGOTRESSで行いました．加熱分布や平衡はJT-60Uのとある放電実験のデータから持ってきました．輸送モデルにはCDBM[52]を用い，径方向50点で1粒子種（電子に相当します）の温度分布の計算をPCクラスタで実行したところ，約2.77秒で計算を終えました．平均すると，各径方向点当たり212世代で収束解が得られていることになります．なお，世代ごとの個体数は100であり，CDBMの評価を行った回数の総数は106万6千回になります．**図6**が示すとおり，各径方向位置において目標熱流束に対応する加熱パワー累積分布P（実線）と輸送熱流束に対応するパワー累積分布$V'Q$（丸印）が一致していることがわかります．これは，式(26)を満たしていることを示しています．その時の規格化温度勾配分布は**図7**に，温度分布は**図8**に示されています．温度分布や規格化温度勾配分布が共に滑らかに得られていることがわかるかと思います．この時の規格化温度勾配分布は温度勾配を何らかの微分スキームで微分したものではなく，温度分布と独立に得られたものであることを強調しておきます．また，**図8**には拡散係数分布も示されています．GOTRESS内部では新古典熱拡散係数も計算しているため，CDBMで評価された乱流熱拡散係数との和が示されています．なお，磁気軸においては規格化温度勾配，つまり輸送熱流束は0であることが自明であ

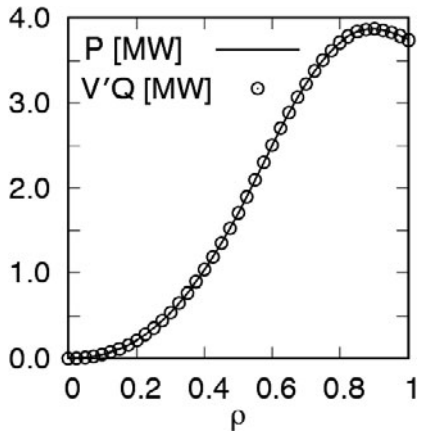


図6　目標熱流束に対応する加熱パワー累積分布Pと輸送熱流束に対応するパワー累積分布$V'Q$．

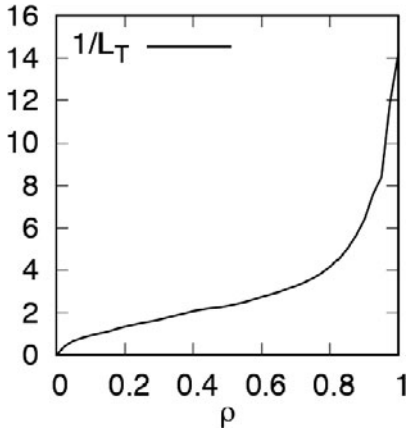


図7　規格化温度勾配 $1/L_T$ の定常解．

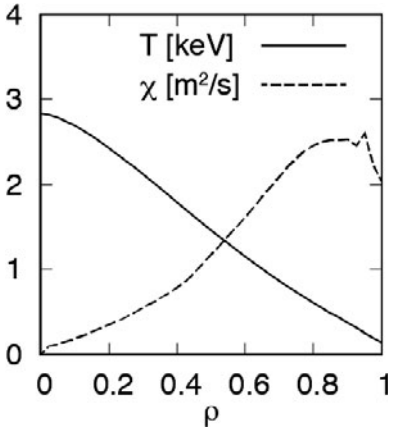


図8　温度 T の定常解とその時の拡散係数 χ．拡散係数は CDBM で評価された乱流熱拡散係数に新古典熱拡散係数が重畳されたものです．磁気軸での拡散係数は 0 としています．

るため，磁気軸では計算しておらず，磁気軸での温度は温度勾配零の条件から算出しています．そのため磁気軸での拡散係数を算出する必要はなく，0 と置いています．

本節では，一見およそ繋がりがなさそうな遺伝的アルゴリズムと偏微分方程式の求解が，解法の工夫によって結びつけられ，輸送方程式を解くという実際の問題へと適用できることをご紹介いたしました．輸送コードとしての GOTRESS は複数粒子種を扱えたり並列計算が行えたりするなど多くの特徴を備えています．詳細は[53]を参考にしてください．

（本多）

謝　辞

5.2.1節におけるシミュレーションで利用した Tausworthe 法に基づく疑似乱数発生プログラムを提供いただいた，高丸尚教氏（中部大学 工学部 教授）に感謝いたします．また，5.2.1節に対して，松山顕之氏（量子科学技術研究開発機構 六ヶ所核融合研究所 主幹研究員）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］D.R. Nicholson, *Introduction to Plasma Theory* (Wiley, 1983); 小笠原正忠，加藤鞆一［訳］：プラズマ物理の基礎（丸善，1986）．
［4］D.G. Swanson, *Plasma Kinetic Theory* (Chapman and Hall/CRC, 2008).
［5］プラズマ・核融合学会［編］：プラズマシミュレーション（京都大学学術出版会，2018）．
［6］渡邉智彦，洲鎌英雄：プラズマ・核融合学会誌 **81**, 534 (2005).
［7］渡邉智彦：プラズマ・核融合学会誌 **81**, 686 (2005).
［8］前山伸也：プラズマ・核融合学会誌 **91**, 589 (2015).
［9］S. Maeyama *et al.,* Phys. Rev. Lett. **114**, 255002 (2015).
［10］M. Nunami *et al.,* Phys. Plasmas **27**, 052501 (2020).
［11］K. Fujita *et al.,* Plasma Fusion Res. **14**, 3403102 (2019).
［12］J.P. Freidberg, *Ideal Magnetohydrodynamics* (Plenum, 1987).
［13］M. Honda, Comput. Phys. Commun. **231**, 94 (2018).
［14］B. Øksendal, *Stochastic Differential Equations: An Introduction with Applications* (Springer, 2003); 谷口説男［訳］：確率微分方程式―入門から応用まで（丸善，1999）．
［15］A. Friedman, *Stochastic Differential Equations and Applications* (Dover, 2004).
［16］保江邦夫：確率論（数理物理学方法序説 4）（日本評論社，2001）．
［17］保江邦夫：物理数学における微分方程式（数理物理学方法序説別巻）（日本評論社，2002）．
［18］高橋陽一郎［編］：伊藤清の数学（日本評論社，2011）．
［19］P. Helander and D.J. Sigmar, *Collisional Transport in Magnetized Plasmas,* Chapter 3 (Cambridge University Press, 2002).
［20］P.E. Kloeden and E. Platen, *Numerical Solution of Stochastic Differential Equations* (Springer, 1999).
［21］T. Misawa and H. Itakura, Phys. Rev. E **51**, 254 (1995).
［22］J.R. Klauder and W.P. Petersen, SIAM J. Numer. Anal. **22**, 1153 (1985).
［23］A.H. Boozer and G. Kuo-Petravic, Phys. Fluids **24**, 851 (1981).
［24］坂元慶行 他：情報量統計学，第 4 章（共立出版，1983）．
［25］R. Kanno *et al.,* Plasma Fusion Res. **6**, 2403066 (2011).
［26］T. Takizuka and H. Abe, J. Comput. Phys. **25**, 205 (1977).
［27］K. Nanbu and S. Yonemura, J. Comput. Phys. **145**, 639 (1998).
［28］C. Wang *et al.,* J. Comput. Phys. **227**, 4308 (2008).
［29］Y. Masaoka and S. Murakami, Plasma Fusion Res. **8**, 2403106 (2013).
［30］G. Zhang and D. del-Castillo-Negrete, Phys. Plasmas **24**, 092511 (2017).
［31］S. Satake *et al.,* Comp. Phys. Comm. **255**, 107249 (2020).
［32］X.Q. Xu and M.N. Rosenbluth, Phys. Fluids B **3**, 627 (1991).
［33］W.X. Wang *et al.,* Plasma Phys. Control. Fusion **41**, 1091 (1999).
［34］S. Brunner *et al.,* Phys. Plasmas **6**, 4504 (1999).
［35］S. Murakami *et al.,* Nucl. Fusion **40**, 693 (2000).
［36］S. Murakami *et al.,* Nucl. Fusion **46**, S425 (2006).
［37］S. Matsuoka *et al.,* Phys. Plasmas **22**, 072511 (2015).

［38］H. Yamaguchi and S. Murakami, Nucl. Fusion **56**, 026003 (2016).
［39］Y. Feng *et al.,* J. Nucl. Mater. **266-269**, 812 (1999).
［40］A.M. Runov *et al.,* Phys. Plasmas **8**, 916 (2001).
［41］M. Kobayashi *et al.,* Contrib. Plasma Phys. **44**, 25 (2004).
［42］R. Tatsumi *et al.,* Plasma Fusion Res. **15**, 1403003 (2020).
［43］プラズマ・核融合学会［編］：プラズマシミュレーション，第7. 4. 1節（京都大学学術出版会，2018）.
［44］桜井 淳：日本原子力学会和文論文誌 **2**, 555 (2003).
［45］S.C. Jardin *et al.,* J. Comput. Phys. **227**, 8769 (2008).
［46］G. V. Pereverzev and G. Corrigan, Comput. Phys. Commun. **179**, 579 (2008).
［47］J.H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence,* second ed. (MIT Press, Cambridge, 1992).
［48］C.R. Reeves, J.E. Rowe, *Genetic Algorithms: Principles and Perspectives* (Kluwer Academic Publishers, New York, 2002).
［49］P. Charbonneau, Astrophys. J. Suppl. Ser. **101**, 309 (1995).
［50］http://www.hao.ucar.edu/modeling/pikaia/pikaia.php
［51］J. Candy *et al.*, Phys. Plasmas **16**, 060704 (2009).
［52］M. Honda and A. Fukuyama, Nucl. Fusion **46**, 580 (2006).
［53］M. Honda and E. Narita, Phys. Plasmas **26**, 102307 (2019).

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．

本多 充（ほんだみつる）

量子科学技術研究開発機構 那珂核融合研究所 先進プラズマ研究部 上席研究員．主にトカマク中の輸送現象や輸送シミュレーションを研究対象としていますが，機械学習やベイズ推定，最適化問題にも興味を持っていて研究への応用を進めています．家族でカラオケボックスに行くことが多かったのですが，昨今の情勢でなかなかままなりません．95点以上を獲れる曲を増やすのが目標です．

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．

J. Plasma Fusion Res. Vol.96, No.8 (2020) 418–419

講座 シミュレーションのための乱数入門

6．おわりに

6. Summary

宇佐見俊介
USAMI Shunsuke
自然科学研究機構 核融合科学研究所

（原稿受付：2020年5月22日）

本講座では，コンピュータシミュレーションにおける乱数の実践的な入門書として，様々な乱数発生方法の原理から始めて，その高速化・並列化手法，乱数の品質・検定法，そして，乱数の変換・利用例まで紹介しました．以下では，これらの内容を振り返って，本講座のまとめとさせていただきます．

第2章では，コンピュータにおいて乱数を生成する原理を，決定論的な演算によって作られる疑似乱数とランダムな自然現象を利用する物理乱数の2つに分けて説明しました．2.1節では，疑似乱数の発生法として，線形合同法，Tausworthe（トーズワース）法，Mersenne-Twister（メルセンヌ・ツイスター）法（MT法）を取り上げました．特に，MT法については，サンプルプログラムを例示して実際の使い方を解説しました．一方2.2節では，物理乱数発生器として，定電圧ダイオードの出力信号に含まれるゆらぎを利用したものと，レーザー光の偏光における量子ゆらぎを利用したもの（量子乱数）をピックアップし，それぞれの原理を紹介しました．また，それらの物理乱数が利用できるサービスについても紹介しました．

第3章では，並列プログラムで乱数を使用することを想定して，乱数発生の高速化チューニング方法を解説しました．MT法による乱数生成の並列化方法としてDynamic Creator，およびJump Methodを挙げ，特に後者については，数値計算ライブラリKMATH_RANDOMをベースにして，その使い方を説明しました．また，乱数利用の効率化のため，「水瓶方式」というアイデアを示しました．使える物理乱数発生器が1個の場合においても乱数を並列プログラムで効率よく利用する方法として，「源泉かけ流し方式」というアイデアも説明しました．これら「水瓶方式」，および「源泉かけ流し方式」は，本講座の著者である佐竹真介氏が考案したオリジナル手法です．

第4章では，乱数らしさについて，周期性，ビット，多次元均等分布性（あるいは結晶構造）といった観点から詳しく論ずるとともに，乱数が持つべき統計的性質を利用しての乱数の検定方法を解説しました．具体的には，上昇連，下降連，近接値の出現率，平均値，二乗平均値が挙げられました．その一方で，「これを通過したら，その乱数の品質は保証される」という万能検定は存在せず，多くの検定をクリアすることで，「良い乱数らしい」という確度を上げるしかない，ということが強調されました．また，乱数を並列化する場合，各並列プロセスに配分された乱数列同士の独立性が重要な品質ファクターとなることを論じました．その独立性を議論する文献がないことから，著者が独自に行った独立性に関するテストについて紹介しました．

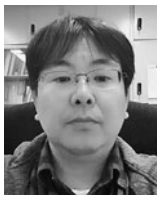
第5章では，乱数の変換例と利用例を示しました．まず5.1節では，一様乱数から任意の確率密度分布に従う乱数を作る方法として，逆変換法（直接法）と棄却法を紹介しました．逆変換法では，作りたい確率密度分布の累積分布関数を利用しました．棄却法では，2組の一様乱数を用意し，一方は，他方の乱数要素の採用・棄却を判定するために用いました．その後，例としてBox-Muller法を取り上げ，2次元空間で正規分布を生成する具体的な方法を解説しました．次に5.2節では，乱数をシミュレーションにおいて，どのように活用しているのかについて紹介しました．5.2.1節では，拡散方程式の解法とDirichlet-Poisson混合問題を例として，乱数を用いた基礎的なシミュレーション手法がどのようなものかを見ました．ここでは，乱数は，Coulomb衝突や熱拡散といったランダムな物理過程を，確率過程論に基づいて具現化するために用いられていました．また，5.2.2節では，プラズマ・核融合分野のみならず，様々な科学技術分野で注目を集めている「遺伝的アルゴリズム」を取り上げました．その適用例として，非線形な輸送方程式を解く問題を大域的最適化問題へと転換し，遺伝的アルゴリズムを用いて解を求める方法を紹介しました．こちらでは，乱数は，偏微分方程式を解くためのアルゴリズムにおけるツールという役割を果たしていました．

この第6章は，本講座のすべての原稿が出揃った段階で書いていますので，いわば本講座の紹介一覧，あるいは目次のような役割を果たしているとも言えます．読者の皆様が，本講座を振り返って読み返したい箇所，または自身の研究に必要な箇所にたどり着きやすいことを目標に，まと

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

author's e-mail: usami.shunsuke@nifs.ac.jp

めました．最後に，第１章で述べたことの繰り返しになりますが，乱数を用いようとする，あるいは，乱数自体に興味がある研究者，学生などにとって，本講座が入門としての役割を果たし，知的好奇心を刺激するものとなっていることを願っております．



宇佐見俊介（うさみしゅんすけ）

自然科学研究機構 核融合科学研究所 ヘリカル研究部基礎物理シミュレーション研究系 准教授．

粒子シミュレーションを用いて，プラズマの様々な複雑な現象を調べていて，現在の主な研究テーマは磁気リコネクションです．SF が好きなためか，現実世界はコンピュータ上のシミュレーションでは？と密かに妄想しています．今回の講座をとりまとめている際に，世界がシミュレーションなら，量子ゆらぎに基づく乱数も実は疑似乱数で，何らかの規則性があったりしないか？とさらに妄想を膨らませています．