モンテカルロコードにおける乱数発生方法の研究

大矢 賢太郎^{1,*}, 北田 孝典¹, 田中 慎一¹

Study on Random Number Generator in Monte Carlo Code

Kentaro OYA^{1,*}, Takanori KITADA¹ and Shinichi TANAKA¹

¹*Division of Sustainable Energy and Environmental Engineering, Graduate School of Engineering, Osaka University,
2-1 Yamadaoka, Suita-shi, Osaka 565-0871, Japan*

(Received November 2, 2010; accepted in revised form June 29, 2011; published online October 15, 2011)

The Monte Carlo code uses a sequence of pseudo-random numbers with a random number generator (RNG) to simulate particle histories. A pseudo-random number has its own period depending on its generation method and the period is desired to be long enough not to exceed the period during one Monte Carlo calculation to ensure the correctness especially for a standard deviation of results. The linear congruential generator (LCG) is widely used as Monte Carlo RNG and the period of LCG is not so long by considering the increasing rate of simulation histories in a Monte Carlo calculation according to the remarkable enhancement of computer performance. Recently, many kinds of RNG have been developed and some of their features are better than those of LCG. In this study, we investigate the appropriate RNG in a Monte Carlo code as an alternative to LCG especially for the case of enormous histories. It is found that xorshift¹⁾ has desirable features compared with LCG, and xorshift has a larger period, a comparable speed to generate random numbers, a better randomness, and good applicability to parallel calculation.

KEYWORDS: *Monte Carlo code, random number generator, xorshift, linear congruent generator*

I. 緒 言

統計論的手法であるモンテカルロ法は炉心解析手法として長年使われ続けている。モンテカルロ法は乱数を用いて確率変数のサンプリングを行うことで数値計算の解を求める手法であり、おおよそ考えられるすべての体系を正確に取り扱えることやエネルギーを多群化することなく連続的に取り扱えることから、モデル化誤差を排除することが可能であり、決定論的手法に対する参照解として頻繁に用いられている。しかしながらモンテカルロ法での計算結果に統計誤差を含むため、モンテカルロ法によって統計誤差の小さい解を得るには追跡する粒子の数(ヒストリ数)を増やし、大量の粒子の挙動を追跡する必要があり決定論的手法に比べて多大な計算時間が必要となる。

モンテカルロ法での粒子の追跡においては、発生する事象を確率的に取り扱うために乱数が多用されている。モンテカルロコードで用いられている乱数は、一般的には真の乱数ではなく、算術的計算を用いることによって生成され

る疑似乱数である。疑似乱数は周期を必ずもっており、周期長を超えると始めと同じ乱数列になるため、追跡する事象の独立性を保証するためにも周期長は十分に長いことが望ましい。

一方、近年の計算機の性能の向上や並列計算の導入によってモンテカルロコードに用いるヒストリ数は増加傾向にあり、計算に用いられる疑似乱数の数もそれに伴って増加している。モンテカルロコードは固有値問題や固定粒子源問題などを計算することができるが、疑似乱数の周期長は特に固定粒子源問題において注意が必要となる。固有値問題では世代ごとに核分裂中性子分布が更新されるため、中性子発生位置情報は世代とともに変化する。つまり乱数の生成数が周期長に達し同じ乱数列が生成された場合においても、乱数は異なる事象の選択に使用され、周期長を超えることによる影響は小さいと考えられる。しかし固定粒子源問題においては、中性子や光子のランダムウォーク開始位置やエネルギーが固定されているため、確率は大きくないが同じ乱数列が同じ事象の選択に使われる可能性があり、周期長を超えることが計算結果へ思わぬ悪影響を与える可能性がある。つまり乱数が周期長を超えた場合、粒子のランダムウォークの独立性が保証できなくなる恐れがあ

¹ 大阪大学大学院 工学研究科環境・エネルギー工学専攻

* Corresponding author, E-mail: oya@ne.see.eng.osaka-u.ac.jp

り、望んだヒストリ数における計算結果とは異なる解を得る危険性が存在し得る。

疑似乱数の周期が十分に長く、シミュレーションに用いられる乱数の総数が周期に達しないことが保証されれば上記のような危険性はなく、結果の信頼性も向上できると考えられる。

現在よく用いられている粒子輸送モンテカルロコードとそれぞれのコードに用いられている疑似乱数生成手法の周期を **Table 1** に示す²⁾。Table 1 に示したすべてのモンテカルロコードでは、乱数生成手法として線形合同法が採用されている。線形合同法は乱数生成速度が非常に高速であり、並列計算への適応も可能であるため、古くに開発された手法でありながら現在でも多くのモンテカルロコードで採用されている。一例として、MCNP5³⁾はデフォルトで 2^{46} の周期長をもつ疑似乱数を用いており、オプションにより最大 2^{63} ($\equiv 10^{19}$)の周期長をもつ疑似乱数を発生させることができる。モンテカルロ計算で必要とされる乱数の数について、近年では可能となっている10億ヒストリで燃料格子体系における固有値問題を計算する場合を例にとれば、1ヒストリ当たりの乱数生成数は1万個に満たないため、乱数の数は 10^{13} 個未満となり 2^{63} の周期長は十分大きく問題ない。しかしながら今後さらに大きなヒストリ数の計算が実行されるであろうことは容易に想像できるため計算機の性能向上の速度を考慮すれば、 2^{63} が将来的に十分な周期長ではなくなる可能性がある。例として、十数年前に開発されたMVP⁴⁾は周期長が 2^{30} ($\equiv 10^9$)の疑似乱数を用いており、100万程度のヒストリ数で計算されていたが現在では普通となっている10億ヒストリの計算において十分な周期長とはいえなくなっている。これを考慮すれば数十年後には兆単位 of ヒストリ数による計算が行われる可能性は十分にあるため、さらに周期長の長い疑似乱数が望まれる。

しかしながら線形合同法による疑似乱数の周期長は、計算機のビット数に依存しており、64ビットより大きなビット数で稼働する計算機が出てこない限り、計算速度を犠牲にすることなく周期長を大きくすることは困難である。一方、線形合同法以外にも様々な乱数生成手法が開発されてきており、乱数生成速度が高速なものや乱数性がよいもの

のなどそれぞれ固有の特徴を有している。

モンテカルロコードで用いる乱数の性質としては乱数の取り得る値が一様な確率で出現しその平均値が正規分布に従う性質と、以前生成した乱数との相関をもたずに同じ分布に従う独立同分布(i.i.d)である性質をもつことが必要である。そこで本研究ではこれらの性質をもつ乱数生成手法の内、周期長、乱数生成速度、並列計算への適応性に着目し、様々な疑似乱数生成手法と線形合同法を比較することで、モンテカルロコードに用いることが妥当な疑似乱数生成手法を検討した。

II. 乱数生成手法

乱数生成手法としては様々な手法が存在し、目的に応じて採用する手法を選択する。計算機を用いた場合は完全にi.i.dであることは不可能であるが、i.i.dに近い性質をもち、乱数の出現が等確率である性質をもつ乱数生成手法をこの章で取り上げ、それぞれの乱数発生手法の特徴を簡単に述べる。

1. 物理乱数

物理乱数とは確率的な物理現象に基づいて生成される乱数であり、ダイオードに電圧を印加することにより生じる熱電子雑音や放射線を用いたものが広く用いられている⁵⁾。物理乱数は周期が無限であるものの、乱数生成速度が以下に述べる算術乱数に比べて非常に遅いという弱点がある。

2. 算術乱数

算術乱数とは、ある計算アルゴリズムに従って生成される乱数である。算術乱数は必ず有限の大きさの周期長をもち、周期長や乱数生成速度は用いる計算アルゴリズムにより異なる。よく用いられている算術乱数を以下に示す。

(1) 線形合同法 (Linear Congruential Generator : LCG)

最も広く用いられてきた算術乱数であり、定数 a, c, M を用いて一次の漸化式(1)によって乱数を生成する。 $\text{mod } M$ は M で割り剰余を取ることを意味する。

$$x_{n+1} = (ax_n + c) \text{ mod } M \quad (n=0, 1, 2, \dots) \quad (1)$$

乱数生成速度は非常に高速であり、任意個先の乱数を生成するアルゴリズムが存在し並列化に適している。短所として周期長は他の乱数生成手法に比べて小さく(詳細はIII章2で述べる)、生成した乱数をxy座標として平面にプロットした場合に直線状にしか現れないという性質(多次元疎結晶構造)をもつことや、乱数の下位ビットの乱数性が悪い(詳細はIV章の2で述べる)ことなどが挙げられる⁶⁾。

(2) 平方採中法

n 桁の整数 x を初期値とし、 x^2 を求めて得られた $2n$ 桁の整数の上下 $n/2$ 個を除いたものを生成された乱数とす

Table 1 RNGs of Monte Carlo codes and their period

Monte Carlo code	period
MVP (version 1.0)	2^{30}
MVP 2.0	2^{63}
MCNP-4	2^{46}
MCNP5 (default/optional)	$2^{48}/2^{63}$
VIM	2^{46}
RACER	2^{45}
RCP	2^{48}
MORSE	2^{45}

る手法である。例として初期値を1234とすれば x^2 は01522756となり生成される乱数は5227となる。計算は非常に単純であり計算は高速である。しかし累乗で得られた数値の半分の桁を用いるためにその周期は線形合同法よりも短く、また初期値の選択を誤れば乱数の値として0が生成され続ける危険性がある⁷⁾。

(3) TGFSR

TGFSR は以下の式により乱数を生成する。

$$x_{l+n} = x_{l+m} \oplus Ax_l \quad (l=0, 1, 2 \dots) \quad (n>m>0) \quad (2)$$

ただし TGFSR は通常の整数の四則演算とは異なりガロア体 GF(2) 上の計算を用いる。GF(2) とは 0 と 1 からなる集合体であり、加算を排他的論理和として扱う。 x が 32 bit 整数であれば 2 進数で表現した長さ 32 のベクトルとみなし A は 0 と 1 からなる 32×32 の変換行列を示す。つまり整数を 0 と 1 のビット列とみなし、1 と 0 の並びを行列で変換して他に保存していたビット列と排他的論理和を行うことによって生成されたビット列を再び 10 進数に変形して乱数として用いる。計算には x_0 から x_{n-1} までの n 個の初期値を保存し、そのうち 2 つの値を用いて新たな乱数を生成する。線形合同法と異なり高次の漸化式により乱数を生成しているため周期長は線形合同法に比べて非常に大きくすることが可能であり、線形合同法がもつ多次元疎結晶構造や下位ビットの問題を有さない。計算処理は線形合同法に比べて多少複雑である^{8,9)}。

(4) Mersenne Twister

TGFSR を改良した手法であり、以下の式によって乱数を生成する。ただし計算は GF(2) で行っており A, B はビット列を変換させる行列である。

$$x_{l+n} = x_{l+m} \oplus Ax_{l+1} \oplus Bx_l \quad (l=0, 1, 2 \dots) \quad (n>k>m>0) \quad (3)$$

TGFSR の手法に加えて、周期長が素数となるような役割をもつ行列 B を用いることによって周期長を非常に大きな素数の形にしており、素数周期の判定が容易となり TGFSR よりもさらに周期長が長く、最も普及している MT19937 の周期は $2^{19937}-1$ と非常に長い。乱数性も非常によく 623 次元均等分布の性質をもつ。計算処理は複雑であり計算速度は線形合同法に劣る¹⁰⁾。

(5) WELL

GF(2) 上における高次の複雑な漸化式によって乱数を生成している。 r 個の初期値のうち 5 つの値と 8 種類の行列を組み合わせて新たな乱数を生成する。周期長は非常に大きく $r=1391$ とした WELL44497a の周期は $2^{44497}-1$ となり、乱数性もよく Mersenne Twister を大きく上回る 1390 次元均等分布の性質をもつ。次の乱数生成のためには r 個の値を保存する必要があるため使用メモリは大きく、同様の処理を有する Mersenne Twister よりも計算処理が複雑であるため計算速度は劣る¹¹⁾。

(6) Xorshift

GF(2) 上で計算を行っており、周期長は Mersenne

Twister や WELL 以下ではあるが線形合同法よりは周期長が大きい。乱数生成速度が非常に高速である。詳細は IV 章で述べる。

(7) RANROT

RANROT は以下の式によって乱数を生成する。

$$x_n = ((x_{n-j} \text{rotr } r_1) + (x_{n-k} \text{rotr } r_2)) \bmod 2^b \quad (4)$$

(A rotr B) は A を B ビット右に循環シフトを行うことを意味しており、循環シフトと加算した値の剰余を乱数としている。RANROT は数学的に周期長を計算することが困難であり、初期値によって周期長は大きく変化し非常に短い周期となることがあり、周期長は実験的に求める必要があるが b が大きい場合の周期長の確認を行うのは計算機のコスト的に困難であるため実際は線形合同法と同程度かそれ以下の周期長となる可能性が大きい。周期がわからない点で並列化にも不向きである。乱数生成は非常に高速であり必要なメモリも Mersenne Twister に比べて小さい¹²⁾。

(8) Blum-Blum-Shub

以下の式によって乱数を生成する。

$$x_{n+1} = x_n^2 \bmod pq \quad (5)$$

ただし p と q は互いに素であり、 x_{n+1} の最低 bit のみを用いる。算術乱数であるにも関わらず前の乱数から次に生成される乱数を予測することが理論的に不可能なため、暗号分野で用いられることが多い。しかしその他の算術乱数法と比較して計算速度は遅い¹³⁾。

III. 乱数生成手法の選択

I 章で述べたようにモンテカルロコードに用いる疑似乱数としては、周期長が大きいこと、乱数生成が高速であること、並列計算への適応性などが望まれる。これらの性質以外にも、乱数列の再現性があることやメモリの使用量が少ないことなども、望ましい性質として挙げることができる。この章ではこれらの観点から乱数生成方法を比較し、モンテカルロコードで使用するに望ましい乱数生成手法について検討した結果を示す。

1. 再現性・予測不可能性

算術演算により生成されるほとんどの疑似乱数は、初期値を同じにすれば同じ乱数列を出力する。そのため初期値を同じとすることで常に同じ結果を得ることができる。一方、物理乱数を採用した場合には、算術乱数での初期値という概念もなく常に異なる乱数列を生成するため、計算を実施するたびに異なる結果が得られる。しかしながらモンテカルロコードに新しいルーチンを導入することによる影響を評価したい場合や計算の確認を行う場合には疑似乱数が再現性をもっていること望ましいことから、本検討では結果の再現性をもつ算術乱数を対象とした検討を実施した。

また算術乱数の中にも、過去の乱数列の出力から先の乱数を予測できない性質(予測不可能性)をもつ手法として Blum-Blum-Shub があり物理乱数を採用した場合と同じ

く、結果の再現性がないものの、モンテカルロコードに導入すべき手法からは除外した。

2. 周期・メモリの使用量

算術演算で得られるほとんどすべての疑似乱数は有限の大きさの周期性をもっているが、モンテカルロシミュレーションに用いる疑似乱数は I 章で述べたように十分に長い周期をもつことが望ましい。また同程度の周期長の場合には、所要メモリ量が少ないことが望ましい。この節では、これらの観点から乱数発生法を比較した結果を述べる。

線形合同法よりも周期長の大きい手法の採用を考えているため、周期長が線形合同法と同程度もしくはそれ以下である平方採中法や RANROT は今回の検討対象からは除外した。残りの手法について比較検討するため、比較対象となる線形合同法について、少し詳しく説明しておく。

線形合同法は一次の漸化式 (1) によって乱数を生成する。つまり、1 つ前に発生させた乱数 x_n と乗数 a の積に c を加算して M で除算することにより求められた剰余の値が次の乱数 x_{n+1} となる。 a, c, M は定数であり、この定数の値によって乱数 x の周期長や乱数性が決まるが、周期長に関しては最大値が M となる。初期値を x_0 とすれば、漸化式 (1) によって乱数列 $x_0, x_1, x_2, \dots$ が生成され、 i 番目に初期値 x_0 と同じ値が生成された場合は始めに生成した乱数列 $x_0, x_1, x_2, \dots$ と全く同じ並びの乱数列 $x_i, x_{i+1}, x_{i+2}, \dots$ が生成され、この場合の疑似乱数は $i-1$ の周期をもつ。つまり線形合同法を用いた疑似乱数が最大の周期をもつためには乱数 x が表現可能な値のすべてのパターンの乱数を一度も同じ値が重複することがないように生成する必要がある、そのときに周期長は最大値 M を取る。Fig. 1 に 2 進数計算機における 32 bit 符号なし整数の状態の表現方法と表現し得る状態の数を示す。1 つの枠が 1 bit に相当しており、それぞれ 0 か 1 の数を表現可能である。Fig. 1 に示すように 32-bit 符号なし整数であれば 2^{32} パターンの状態を表現可能で疑似乱数 x を 32 bit とした場合の最大周期は 2^{32} となり、一般に y -bit 符号なし整数を用いる場合には最大周期は 2^y 以下となる。線形合同法による乱数が最大周期をもつための (1) 式のパラメータの条件は明確に示されている⁷⁾。

- $c \neq 0$ の場合 (最大周期: M)
 c と M が互いに素

Fig. 1 Number of representable state of 32-bit unsigned integer

$a-1$ が M を割り切るすべての素数の倍数

M が 4 の倍数の場合 $a-1$ も 4 の倍数

- $c=0$ の場合 (最大周期: $M/4$)

$M=2^e (e \geq 4)$

初期値 x_0 が奇数

$a \pmod{8} = 3$ または 5

これに対して、II 章で挙げた算術乱数の中で TGSFR, Mersenne Twister, WELL, xorshift は線形合同法に比べて非常に大きな周期長をもつ。よく使われる周期長を Table 2 に示す。これらの手法には周期の短いものや長いものなど様々な種類が存在するが、Table 1 の周期長と比べて非常に大きな周期長をもつことがわかる。これらの手法の周期長が大きくなり得るのは、疑似乱数が (6) 式のような GF(2) 上における高次の漸化式を用いて複数のビット列を保存しているからである。

$$x_{i+n} = Ax_i \oplus Bx_{i+m} \oplus Cx_{i+k} \oplus \dots \quad (n > k > m > 0) \quad (6)$$

$$(i=0, 1, 2, \dots)$$

(6) 式では n 個のビット列を保存しており、 x のビット列の長さが 32 である場合 (x は 32 ビット整数)、GF(2) 上で整数をビット列として扱うため保存している n 個のビット列を $32 \times n$ の長さをもつ 1 つのビット列 $N_i(x_{i+0}, x_{i+1}, \dots, x_{i+n-1})$ とみなせば (6) 式は

$$N_{i+1} = WN_i \quad (i=0, 1, 2, \dots) \quad (7)$$

と表すことができる。 W は $(32n) \times (32n)$ の行列であり、要素として (6) 式の A, B, C をもつ。したがって 32 bit 整数で線形合同法が 2^{32} 通りの状態まで表現可能であるのに対し 2^{32n} 通りの状態の表現が可能となり周期長は線形合同法よりも大きくなる。

しかし高次の漸化式によって乱数を発生させるには複数の乱数のデータを保存する必要がある、線形合同法が 1 つのデータを保存していたのに対し、Table 2 に示すように保存しなければならないデータが増加する。高次の漸化式による乱数生成手法をモンテカルロコードに用いる場合には、保存するメモリ量が不必要に大きくならないものを選択するべきである。メモリ使用量が大きい手法は計算手法も遅くなる傾向があることも考慮に入れる必要がある。

Table 2 において最も周期が短い xorshift の xor128 の周期は $2^{128}-1 (\approx 10^{38})$ であり、MCNP を用いて燃料格子体系において 10 億ヒストリで計算する場合の乱数の生成

Table 2 Period and data memory of pseudo random numbers generated by RNG that has long period

Method	Version	Period	Conservative data (byte)
TGSFR ⁸⁾	TT800	$2^{800}-1$	100
Mersenne Twister ¹⁰⁾	mt19937ar	$2^{19937}-1$	2,500
WELL ¹¹⁾	WELL44497a	$2^{44497}-1$	5,568
Xorshift ¹⁾	xor128	$2^{128}-1$	16

数が 10^{13} 以下となることを考えると25桁の余裕があり将来的にも十分長い周期長であると考えられることから、次節の乱数生成速度の評価は xorshift を中心に行うこととした。

3. 乱数生成速度

乱数生成に要する時間はモンテカルロコードによる計算時間の一部であり、乱数生成に要する時間を短縮することでモンテカルロコードの計算時間は短縮できる。前節までの検討から、線形合同法の代替として有望と考えられる xorshift について、線形合同法との乱数生成速度の比較を行った。また xorshift に比べてメモリ使用量が多いものの広く使われている Mersenne Twister についても乱数生成速度を測定し比較した。乱数生成速度は、使用する CPU やコンパイルオプションで大きく変化する。そのため **Table 3** では MCNP コードのコンパイルに用いられているオプションを使用した場合、**Table 4** では計算速度を最大にするオプションを用いた場合の結果を示している。計算機は 64 bitCPU の Intel®Core™2 Quad CPU 3.40 GHz と 32 bitCPU の Intel®Pentium®4 CPU 3.00 GHz で実施した場合であり、それぞれ80億個の乱数を生成するのに要する時間を計測した結果を示している。Table 3, Table 4 において xor64, xor96, xor128 は周期が $2^{64}-1$, $2^{96}-1$, $2^{128}-1$ の xorshift による結果、lcg48, lcg63 は 64 bit 整数を用いた線形合同法による結果であり、(1)式のパラメータ a, c, M を lcg48 では $a=5^{19}$, $c=0$, $M=2^{48}$, lcg63 では $a=9219741426499971445$, $c=1$, $M=2^{63}$ とした場合に対応している。また Mersenne Twister は周期が $2^{19937}-1$ である MT19937 での結果である。Table 3 に示すように xorshift での計算速度は周期長を長くしても変化が小さ

いことがわかる。さらに xorshift での計算速度はすべて線形合同法よりも高速であり、特に Intel®Pentium®4 を使用した場合は線形合同法よりも3倍近く高速となる。これは用いた計算機の CPU の仕組みの違いに起因している。線形合同法である lcg48 や lcg64 は 64 bit 整数を用いており、64 bitCPU であれば 64 bit の整数を一度の命令で扱うことが可能であるが、32 bitCPU は一度に扱うことが可能な整数は最大で 32 bit である。そのため線形合同法に用いる乗算は 64 bitCPU では一度の 64 bit 整数の乗算で済むのに対し 32 bitCPU では 64 bit の整数を 32 bit に分割し、32 bit 整数の乗算を3回と加算を2回必要とし、変数のメモリへの保存やメモリからの読み出しも分割して行うために CPU の処理数は 64 bitCPU に比べて倍増する。それに対して xorshift である xor128 は 32 bit 整数を用いており 32 bitCPU であっても 64 bitCPU と同程度の処理数で計算しているため、64 bitCPU である Intel®Core™2 Quad に比べて 32 bitCPU である Intel®Pentium®4 を用いた方が線形合同法と xorshift の計算時間の比が大きくなる。

また異なるコンパイルオプションを用いた場合においても xorshift は他の手法に比べて高速であることがわかる。Mersenne Twister はオプションの違いによる計算速度の変化が大きく、特に MCNP に用いられるコンパイルオプションを使用した場合は $1/2 \sim 1/6$ 倍の計算速度であり、xorshift や線形合同法に比べてコンパイルオプションの影響が大きい。この傾向は Mersenne Twister と計算手法が似通っている TGFSR や WELL でも同様であると考えている。

Table 3 Calculation time (using MCNP option)

CPU		xor64	xor96	xor128	lcg48	lcg63	mt19937
CQ	Time (second)	20.65	20.20	21.07	30.30	30.30	126.99
	Rate to xor128	1.0	1.0	—	1.4	1.4	6.0
P4	Time (second)	61.70	68.17	71.96	215.12	215.30	124.53
	Rate to xor128	0.9	0.9	—	3.0	3.0	1.7

- CQ : Intel®Core™2 Quad CPU 3.40 GHz, P4 : Intel®Pentium®4 CPU 3.00 GHz
- time for 8billion random numbers generation
- using compile option for ifort9.0 : -O2 -lm -fpp0 -pc64 -W0 -Vaxlib -static-libcxa.

Table 4 Calculation time (using fast option)

CPU		xor64	xor96	xor128	lcg48	lcg63	mt19937
CQ	Time (second)	20.65	20.15	21.06	24.16	27.23	21.58
	Rate to xor128	1.0	1.0	—	1.1	1.3	1.0
P4	Time (second)	32.17	33.54	38.47	46.80	50.08	42.43
	Rate to xor128	0.8	0.9	—	1.2	1.3	1.1

- CQ : Intel®Core™2 Quad CPU 3.40 GHz, P4 : Intel®Pentium®4 CPU 3.00 GHz
- Time for 8billion random numbers generation
- using compile option for ifort9.0 : -fast.

4. 乱数生成手法の選択

以上の検討より、モンテカルロコードの乱数生成手法として望ましい性質を有している乱数生成手法としては、xorshift が選択される。xorshift は、線形合同法よりも十分大きな周期長をもつものの保存するデータ量も少なく、コンパイル方法の違いによる乱数生成速度への影響をあまり受けずに高速で乱数生成が可能であるなど、モンテカルロコードの乱数生成手法として望ましい性質を有していると判断した。V 章では xorshift の中でも周期長が $2^{128}-1$ である xor128 を乱数生成手法として選択し、モンテカルロコードに導入し、その有用性を検討した結果を示す。

IV. Xorshift

1. Xorshift の理論

Xorshift は Marsaglia 博士によって考案された疑似乱数生成方法である¹⁾。32 bit 符号なし整数を用いて r 個のビット列を保存して用いる場合は疑似乱数の周期は $2^{32r}-1$ ($r=1, 2, 3, 4, 5 \dots$) となり、線形合同法を用いた場合の最大周期が 2^{32} であるのに対して非常に長い周期をもつ。乱数生成の数式は GF(2) に基づいた (6) 式で表される多次元の漸化式で表現される。(6) 式のビット列を変換する行列 A, B, C の複雑さにより計算処理数が異なり、xorshift では片側の副対角成分と対角成分が 1 でその他の成分が 0 である正則行列を a 乗した行列を用いている。この行列をビット列に掛けるということは、右もしくは左に a ビットシフト演算を行い演算される前の値と排他的論理和の演算を取る操作に相当し、xorshift はどの周期のものにおいてもこの操作を三度施している。つまり排他的論理和演算とビットシフト演算のみを用いており、一般的にビット演算は四則演算に比べて高速であるので計算速度は高速である。xorshift により生成される疑似乱数は線形合同法のように多次元結晶構造をもたず乱数性もよい。ビット列を変換させる行列が単純であるため乱数発生ルーチンは Mersenne Twister や WELL に比べて非常に短くシンプルであり、モンテカルロコードへの組み込みも容易である。例として 32 bit 符号なし整数を用いた周期 $2^{128}-1$ の xorshift のプログラム(C 言語)を以下に示す。

```
unsigned long xor128( ){
    static unsigned long x=123456789, y=362436069,
                        z=521288629, w=88675123;
    unsigned long t;
    t=(x^(x<<11)); x=y; y=z; z=w;
    return(w=(w^(w>>19))^(t^(t>>8)));
}
```

このプログラムから乱数生成のアルゴリズムが非常にシンプルでありに乗算の性能がよくない計算機においても高速で乱数生成が可能であることが理解できる。

2. DIEHARD による乱数性の評価

Xorshift をモンテカルロコードに導入するに当たって、xorshift によって生成した疑似乱数の乱数性の確認を実施した。乱数性とは生成した乱数の数値分布の偏りを表しており、偏りが小さいことが望ましい。乱数性が悪い場合には、真の乱数を用いた場合に比べてサンプリングされる状態が異なり、モンテカルロ計算結果にもバイアスがかかる恐れがある。このような危険性がないことを確認するために、生成される乱数を統計的に検定する方法がいくつか考案されており、ここでは Marsaglia 博士により考案された DIEHARD¹⁴⁾を用いた。DIEHARD は Table 5 に示す 17 種類の統計テストから構成されている乱数性の検定方法である。

Table 3, Table 4 に計算速度結果を示した線形合同法の lcg48 の場合では、乱数として得られるビット列の上位 32 bit を検定すると、Gorilla, OPSO, OQSO, DNA, Craps test の 5 つの統計テストに不合格であった。これは線形合同法によって生成される乱数は、その乱数性に問題があるためであり、下位 bit の周期が短いことに起因している。lcg48 の場合、上位から 15~20 bit より下の bit 列の乱数性が悪く、乱数の範囲が $[0, 1)$ ならば $10^{-5} \sim 10^{-7}$ 以下の乱数性が悪いということになる。しかしながら非常に小さい値における乱数性のみが問題であるために、乱数による事象選択数が非常に大きい場合以外は、モンテカルロ計算の結果にほとんど影響しない。また lcg63 で生成された乱数の場合、上位 32 ビットについてはすべての検定に合格した。lcg63 でも lcg48 と同じ問題が生じるものの、下位 32 bit の範囲であるため上位 32 ビットではすべての検定に合格している。また、参考文献 2) では MCNP5 に組み込まれている線形合同法に関してオプション(パラメータ)を変更して何ケースかの統計テストが詳しく行われており、

Table 5 DIEHARD TEST

-
- Birthday spacings test
 - GCD test
 - Gorilla test
 - Overlapping permutations test
 - Ranks of 31×31 and 32×32 matrices test
 - Rank of 6×8 matrices
 - Monkey tests on 20-bit words
 - Monkey tests OPSO, OQSO, DNA test
 - Count the 1' in a stream of bytes test
 - Count the 1's in specific bytes test
 - Parking lot test
 - Minimum distance test
 - Random spheres test
 - The squeeze test
 - Overlapping sums test
 - Runs up and down test
 - The craps test
-

正しくパラメータを選択すれば線形合同法はモンテカルロシミュレーションの使用においては問題ないことが報告されている。

これに対して、今検討で選択した xorshift(xor128)に関して、パラメータや初期値を変更しながら複数ケースで検定し、同様の確認を行った結果、Table 5 のすべての検定に合格しており、xorshift は下位 bit においても乱数性に問題がなくモンテカルロにおける使用に問題ないことが確認できた。

3. 並列計算への適応

モンテカルロ計算では計算時間を短縮させる方法として並列計算が効果的でありよく使われている。モンテカルロコードを用いて並列計算を行う場合に疑似乱数の SEED 値(初期値)の決定には注意を払う必要がある。並列計算は単一の計算機で行っていた処理を分割して複数の計算機によって同時に計算を進める。単一の計算機によって計算を実行する場合は処理を逐次的に実行するため、計算を進めながら疑似乱数も逐次的に生成すればよいが、複数の計算機を用いて処理を並列化させた場合は各計算機では独立に計算を進めていくことになるため各計算機に異なる SEED を与えなければならない。複数の計算機に SEED を与える方法として疑似乱数などを用いて適当に割り当てる方法は望ましくない。例えば、2 つの計算機 A と B で並列計算を実行する場合に、ある SEED の x_0 から生成した乱数列を $x_0, x_1, x_2, x_3, \dots, x_{sA}, \dots, x_{sB}, \dots$ と表し A と B に SEED として x_{sA}, x_{sB} を与えたとする。計算機 A における計算処理の実行に伴い乱数を生成し、 x_{sB} を生成する前に計算が終了すれば問題ないが、計算途中で x_{sB} が現れてしまった場合は以後 A では B で用いていたものと同じ乱数を使用することとなり、各計算機の処理が独立でなくなってしまう。このような問題を避けるには、乱数列上において各計算機の SEED は各計算機が消費する乱数の数以上の間隔をもつようにする必要があり、消費する乱数の個数に応じて任意先の乱数を発生させる手法が必要となる。線形合同法では簡単な計算によって任意個先の乱数を発生させることが可能であるため、MCNP では並列計算に対応した SEED 作成を行い各計算機での乱数の独立性を確保している。つまり、今回選択した xorshift を並列計算に適用するためには、各計算機での計算結果の独立性を確保できる SEED 設定手法が必要である。以下では SEED の設定手法について、周期が $2^{32}-1$ であり 32 bit 整数を用いている xorshift を対象として検討した結果を述べる。

乱数の SEED 値を 1×32 の横ベクトル y とし、ビットシフト演算と排他的論理和演算を組み合わせたものを 3 回行うという操作は、要素が 0 と 1 からなる 32×32 行列 T を y に掛け合わせることに相当する。つまり生成される乱数は $y, yT, yT^2, yT^3, \dots, yT^{k-1}, yT^k$ ($k=2^{32}-1$) となる。したがって、 i 個先の乱数を発生させるためには T^i を

SEED 値 y に掛け合わせればよい。何種類かの T^i のセットを用意することにより、何個先の乱数を生成するかを任意に決定できる。例えば、 $T^{100}, T^{1000}, T^{10000}$ を用意しておくことで、 T^{100} を 5 回、 T^{1000} を 2 回、 T^{10000} を 2 回 SEED 値に掛け合わせることで 22,500 個先の乱数を得ることができる。Xorshift が大きい周期長をもっていることを考えれば、何個先の乱数を用いるかを細かく考える必要はなく、計算機ごとに割り当てる SEED 値が周期上において十分に大きな間隔をもってさえいればよいので、十分に大きな i に対する T^i を掛ける操作を一度行うだけで十分であると考えられる。ただし、2 値ベクトルではなく乱数整数であるため、行列を掛けるという考え方を実際に用いるには工夫が必要である。ここで提案するのは xorshift の演算を応用したビット演算を用いた方法である。始めに行列 T を対角成分に分ける。そしてそれぞれの成分と整数である SEED 値 Y と論理積を取り、ビットシフト演算を行う操作は、各対角成分をもつ行列を掛け合わせる操作を行うことに相当する。それらで得られた値を逐次的に和(ただし 0, 1 成分のみの空間でビットの繰り上がりを考慮しないため和を取る操作は排他的論理和を行うことに相当する)を取れば y に行列 T を掛け合わせて得られた y' を整数に直した値に一致する整数 Y' が得られる。このような処理により xorshift も並列計算用の SEED 値設定が高速に実行可能であると考えている。

V. Xorshift のモンテカルロコードへの導入

1. 計算速度の比較

Figure 2 に示す燃料格子体系に対する固有値計算を対象として、モンテカルロ計算に要する計算時間を乱数発生方法として線形合同法(lcg48)と xorshift (xor128)を用いた場合について比較した。計算コードは MCNP5 を使用し計算は 200 万ヒストリ (2 万 $\times$ 100 バッチ) で行い、得られた結果を Table 6 に示す。Intel®Core™2 Quad を用いた場合は計算速度に有意な差はみられなかったが、Intel®Pentium®4 を用いた場合には xorshift が 2% 程度である

Fig. 2 Pincell geometry and material composition

が高速であった。この計算で消費された乱数の数は線形合同法と xorshift の両方の場合で約17億個と同じであり、乱数の発生数から乱数生成に要した時間を評価すれば Intel®Pentium®4 では線形合同法で約45秒、xorshift で約15秒であり、この差が Table 6 の結果に現れている。乱数生成に要する時間は、これは全計算の数%程度であるため、モンテカルロ計算時間の短縮はほとんどみられない。乱数生成に要する時間の割合が大きくなるような体系を予想し、いくつかのケースで計算を実行した結果、乱数生成に要する時間は多くとも5パーセント程度であったため、xorshift を導入してもモンテカルロ計算の大幅な高速化には至らないと考えられる。

Table 6 Time and the number of generated random numbers

	Xorshift	LCG	Time ratio
	Computer time (minute)		
Intel®Core™2 Quad	10.86 ± 0.01	10.88 ± 0.01	1.00
Intel®Pentium®4	21.91 ± 0.01	22.29 ± 0.01	0.98
Generated random Numbers (million)			
	1,710.6 ± 0.1	1,710.6 ± 0.1	

2. 実効増倍率の比較

確認のため、線形合同法と xorshift のそれぞれで計算された実効増倍率の比較を行った結果を示す。線形合同法は lcg48, xorshift は xor128 を用いた。計算体系は上記と同じく Fig. 2 に示した体系を対象とした。計算は100万ヒストリ(1万×100バッチ)とし、乱数の初期値を変化させて100ケースの計算を行って得られた実効増倍率を統計処理し、ヒストグラムと実効増倍率の平均値および標準偏差を Fig. 3 に示す。度数分布は xor128 と lcg48 は同じ形状をしており、「xor128 と lcg48 の実効増倍率の分散に差がない」を帰無仮説として F 検定を行ったところ F 値は1.166 であり有意水準を5%とした場合の棄却限界値1.394を下回り、帰無仮説は棄却できず分散は等しいといえる。また、「xor128 と lcg48 の実効増倍率の平均値に差がない」を帰無仮説として実効増倍率の平均値に関して t 検定を行ったところ、棄却域を5%としたとき t は0.456 となり t 境界値は片側で1.653, 両側で1.972 となり両方とも t よりも大きくなり、帰無仮説は棄却されず平均値は一致しているといえる。これらの結果から線形合同法の代わりに xorshift を用いた場合に実効増倍率に関しては一致することを確認した。

VI. 結 論

モンテカルロコードに用いる乱数発生方法として従来用

Fig. 3 Frequency of keff

いられてきた線形合同法よりもモンテカルロコード用の乱数発生方法として有用な手法がないかを検討した。その結果, xorshift は線形合同法に比べて周期長の大きい疑似乱数を生成することが可能であり, 乱数生成速度も高速であるだけでなく乱数性にも優れていることがわかり, xorshift が線形合同法に代わる乱数生成法として選択すべきであると判断した。

モンテカルロコードにおいて全計算時間のうち乱数生成に費やされる計算時間は最大で5%程度であるため, モンテカルロコードの計算時間を大きく削減させることはできなかったが, 将来予想されるモンテカルロ計算における大幅なヒストリ数の増大に耐え得るほど十分大きい周期長をもっているだけでなく, ビット演算を用いて任意の個数先の乱数を発生させることが可能であり, 並列計算への適応が可能である。これらの点から xorshift がモンテカルロコードにおける乱数発生方法として望ましいと考えられる。

— 参考文献 —

- 1) G. Marsaglia, "Xorshift RNGs," *J. Stat. Software*, **8**[14], 1–6 (2003).
- 2) F. B. Brown, *Fundamentals of Monte Carlo Particle Transport*, LA-UR-05-4983, Los Alamos National Laboratory (LANL) (2005).
- 3) X-5 Monte Carlo Team, *MCNP-A General Monte Carlo N-Particle Transport Code Version 5*, LA-UR-03-1987, Los Alamos National Laboratory (LANL) (2003).
- 4) T. Mori, M. Nakagawa, *MVP/GMVP: General Purpose Monte Carlo Codes for Neutron and Photon Transport Calculations Based on Continuous Energy and Multigroup Methods*, JAERI-Data/Code 94-007, Japan Atomic Energy Research Institute (JAERI) (1994).
- 5) S. Kishimoto, M. Fukue, "Generation of physical random digits by using diode noise and its statistical properties," *Trans. Inst. Electronics, Info. Commun. Eng.*, **J82-A**, 1704–1709 (1999), [in Japanese].
- 6) 伏見正則, 乱数, 東京大学出版会, 東京 (1989).
- 7) D. E. Knuth, *The Art of Computer Programming 3 準数値算法/乱数*, サイエンス社, 東京 (1981).
- 8) M. Matsumoto, Y. Kurita, "Twisted GFSR generators," *ACM Trans. Model. Comput. Simulation*, **2**[3], 179–194 (1992).
- 9) M. Matsumoto, Y. Kurita, "Twisted GFSR generators II," *ACM Trans. Model. Comput. Simulation*, **4**[3], 254–256 (1994).
- 10) M. Matsumoto, T. Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simulation*, **8**[1], 3–30 (1998).
- 11) F. Panneton, P. L'Ecuyer, M. Matsumoto, "Improved long-period generators based on linear recurrences modulo 2," *ACM Trans. Math. Software*, **32**[1], 1–16 (2006).
- 12) A. Fog, *Chaotic Random Number Generators with Random Cycle Lengths*, <http://www.agner.org/random/theory/chaosran.pdf>.
- 13) L. Blum, M. Blum, M. Shub, "A simple unpredictable pseudo random number generator," *SIAM J. Comput.*, **15**[2], 364–383 (1986).
- 14) G. Marsaglia, *Diehard Battery of Randomness v0.2 beta*, <http://i.cs.hku.hk/~diehard/>.