技　法

原子炉雑音研究のための簡易数理計算プログラムのコード構造

Structure of a Simple Mathematical Simulation Code for the Reactor Noise Study

近畿大学原子力研究所
芳原新也
Sin-ya Hohara

抄録

近年、福島第一原子力発電所のデブリ処理の現場において、原子炉雑音解析手法の適用が検討されるなど、古い原子炉実験手法に改めて注目が集まっている。しかしながら、多くの原子炉実験手法は1960年代～1980年代にかけて確立されており、それらの実験解析結果の統計精度等については検証が不十分なものもある。現在の日本においては、規制要求レベルの向上により、試験研究用等原子炉自体が絶滅寸前に追い込まれており、原子炉を用いた実験・解析手法検討を実機で気軽に行うこと自体が難しくなってきている。そこで本研究室では、時間領域に注目したモンテカルロ計算コードを開発して、原子炉雑音測定の基礎検討を行っている。

本論文では、実験解析手法の検討を目的として開発した簡易な数理計算プログラムのコード構築について述べる。

Abstract

In recent years, the old nuclear reactor experimental methods, such as the application of the reactor noise analysis method, are noted in the field of debris processing at the Fukushima Daiichi nuclear power plant. However, many experimental methods for nuclear reactors have been established in the 1960s and 1980s, and some of them are insufficient to verify the statistical accuracy of the experimental results. In the present Japan, because of the improvement of the level of regulatory requirements, the research reactor itself has been driven to the brink of extinction, and it is becoming difficult to easily examine the experimental and analytical methods using nuclear reactors by actual equipment. In our laboratory, we have developed a Monte Carlo computation code focused on the time domain, and a fundamental studying of the reactor noise measurement.

In this paper, we describe the code structure of a simple mathematical computation program developed for the purpose of examining experimental analysis methods.

Keyword: Reactor Noise, Mathematical Code, Time Series, Multiplication Simulation

1．序論

近年、1960 ～ 1980年代に提案された原子炉実験手法に再び注目が集まっている。福島第一原子力発電所におけるデブリ等の臨界管理がその一例である。通常、発電プラントにおいては、その設計・建設段階において要求機能とコストの兼ね合いから採用手法を決定し、具体的な設備設計へと落とし込んでいく。しかしながら、福島第一原子力発電所においては、2011年の原子力事故の処理を実施する必要があり、そこではコストよりも既存故障設備等による現場へのアクセス制限と要求機能の実現可能性が優先されている。

その様な状況において、通常のプラント設計ではコストや確実性の観点から不採用となり得る観測手法までもが採用検討対象として机上に並ぶわけである。過去に提案された様々な原子炉実験手法もその対象となっているが、それらの手法の全てが適用範囲や結果の統計的範囲の厳格な検証が完了しているわけではない。

一方、現在の日本国内においては、平成25年の原子炉施設の規制基準改定（原子炉施設への要求性能向上及び品質管理体制の強化）及び平成29年4月公布の原子炉等規制法の改定（検査制度の大幅変更及び品質管理体制の強化）等により、原子炉施設に対する規制要求は向上の一途を辿っている。これらの法令改正は、「グレーテッドアプローチ」の掛け声に対して些か消極的な現場実現しかされておらず、リスクの小さな施設に対しては過剰規制をかけている現状がある。これらの過剰規制は、施設規模の小さい試験研究用等原子炉に対しては施設維持コストの上昇を要求するものであり、このコスト上昇は既存の試験研究用等原子炉に対して（法令作成者の意図か否かは別として）廃炉という選択肢へと暗に追いやりつつ、新規の試験研究用等原子炉の建設に対する障壁ともなっている。この様な現状において、既存の試験研究用等原子炉や核燃料取扱施設のみで、原子力事故処理手法の検討検証を実施するのは非常に効率が悪い。

ここで、注目すべきなのが計算機実験であるが、時間発展型の臨界計算コードについては軍事転用可能性からその開発・適用に厳格な制限がかけられており、現に日本国産のモンテカルロ臨界計算コードMVPは、時間方向に対して一点近似を行うコード構造となっている。一方で原子炉実験手法自体の様々な検証の一部については、前述の臨界計算コードほどの空間再現性は必要なく、その検証が原理的であるほど数理的に簡略化された計算機上の実験体系のみがあれば良い。

本稿では、原子炉実験手法のうちの一つであるFeynman-α 法の検討検証を目的として開発した簡易な数理計算プログラム[1]のコード構造について解説を行う。

2．基礎理論

原子炉からの生成中性子の測定は、基本的に原子炉内で発生する核分裂反応の間接測定と言い換えることが出来る。多くの原子炉実験手法の原理は、原子炉内での核分裂反応の動特性を基礎として構築されている。このため、数理的に原子炉実験手法を検証するためには、核分裂連鎖反応自体の再現は不可欠となる。

しかしながら、実験における検出器効率等の検討以外の検証、つまり手法自体の検証のみに留まる場合、空間的な拡がりについては二次的な要素となるため、空間一点炉近似を行うことも可能であり、シミュレーション構造自体を簡略化することが出来る。これにより一般的に流布している原子力計算コードに比べてはるかに簡易にコードを構築できる可能性がある。

前述した通り、日本で広く使用されるモンテカルロ臨界計算コードであるMVPについては、時間的拡がりを一点近似し、空間的な拡がりを重視したコード構造を採用している。近畿大学ではMVPとは逆のアプローチ、つまり空間的拡がりについては

一点近似を行い、時間的拡がりについて再現を試みる数理計算プログラムを開発した[1]。

この様なコード構造を再現するためには、核分裂連鎖反応の原理に注目する必要がある。核分裂連鎖反応は、周知のとおり、任意の時刻における核分裂現象を種とし、反応の発生数を、正方向進展の時間的な拡がりにおいて連鎖させて維持するというものである。この現象の種となる核分裂現象は、外部由来の中性子を起因としており、これはとりもあえず「通常の放射性崩壊等」を親として持つということになる。別の観点から見ると、原子炉内における核分裂連鎖反応は、ポアソン事象から分岐した枝事象とも言えるわけである。

ここでは、まず幹事象となるポアソン事象の再現について述べた後に、枝事象である連鎖反応事象（以下では「増幅事象」とよぶ）の再現について述べる。ここで述べる事象の再現では、反応自身の時刻情報を出力するコード構造とするため、一般的な原子炉実験手法で用いられる計数情報への変換が必要となる。近年では時刻情報を記録する原子炉物理実験手法も広まってきているが、旧来の実験手法においては、時刻情報から計数情報への変換は実験装置を用いて実現する。ここでは、時刻情報から計数情報への計算機処理により実現方法についても述べる。その他にも、一様乱数を重み付乱数に変換する方法についても解説する。

２．１　ポアソン事象の時間間隔分布

ポアソン分布は、所与の時間間隔を持って発生する離散事象が従う確率分布の一つで、放射性壊変はポアソン過程の代表的な例である。通常の原子炉における外部中性子の発生は殆どが中性子放出RI（例えば、Pu-BeやAm-Be）を起源としており、今回のシミュレーションコードの幹事象はポアソン過程であると仮定する。

ポアソン過程では、その時間間隔の確率分布は指数関数に従う。事象発生率（計数率）をr［cps］と於いた場合、その時間間隔の確率密度関数p（t）は以下の式により表せる[2]。

$$P(t_{seed})=exp(-r \cdot t_{seed})$$

ここで、$tseed$［sec］は隣り合う発生事象の時間間隔を示す。

この法則に従う乱数を発生させることで、疑似的なポアソン事象を計算機上に再現することが可能となる。

２．２　増幅事象の発生と時間間隔分布

外部中性子による核分裂反応の発生は、微視的に観測すると非常に複雑な過程を経る。しかしながら、巨視的な観測結果である核データ等については、基本的に多数事象の平均確率という形式でまとめられて一般的に利用されている。多数事象の平均確率を利用するということは、一様乱数により再現可能であるということを意味する。

炉物理分野においては、前世代の核分裂反応から次世代の核分裂反応への回帰割合（feedback rate）は、実効増倍率k_{eff}と呼ばれる。また、外部中性子源を種として発生する核分裂反応からの中性子の発生時刻は、厳密には核分裂反応の発生時刻とは異なる。実際の物理現象に注目した場合、核分裂反応により生成する中性子は、先行核（precursor）と呼ばれる核分裂片の中性子放出壊変事象により生成される。物理的な拡がりを一点に圧縮近似して考えた場合（一点炉近似）、種中性子の発生場所から核分裂反応発生場所までの中性子の移動時間（traveling time）は0と近似されることになる。

この時、種中性子の発生時刻（これは前述の通りポアソン過程となる）と核分裂中性子の発生時刻との間には、先行核の生成－壊変時間差が繰り込まれることとなる。先行核の壊変が通常の放射性壊変と同様であると仮定すると、この時間差はポアソン過程に則るため、確率密度関数は以下の様に表すことが出来る。

$$p(t_{multiple})=\exp(-\lambda\cdot t_{multiple})$$

$$\lambda=\frac{ln2}{T_{1/2}}$$

ここで、λは先行核の崩壊定数、$T_{1/2}$は先行核の半減期となる。

以上より、幹事象であるポアソン事象が増幅事象を生成するか否かは一様乱数によって再現でき、種中性子の発生時刻と核分裂生成中性子の発生時刻の時間差は指数乱数により再現することが可能となる。また、この様な増幅事象判定を新たに生成した増幅事象に対しても適用することで、実現象で起こっている連鎖反応を再現することが出来る。

2.3　ポアソン事象と増幅事象の統合

上述の手法により生成したポアソン事象と増幅事象は、計算機上では独立に計算がされることになるが、実現象においては同一時間軸上において同時に観測される。このため、独立に算出した時刻情報はその時刻に順じて統合して並び替える必要がある。この際に生成される増幅事象の数は、前述の増幅確率に応じて異なるが、概ね等比級数により見積もることが出来る。これは、ポアソン事象から一次的に派生した増幅事象が親となって生成されるn次的な増幅事象の数は、増幅確率のn乗に比例するという発想に基づくものである。

このため、幹事象のポアソン事象数をn［個］、増幅確率をa［%］とおくと増幅事象の数$n_{multiple}$［個］と幹事象数との合計数は、以下の無限級数により見積もることが出来る。

$$n+n_{multiple}=n+\frac{a}{100}\cdot n+\frac{a}{100}\cdot\left(\frac{a}{100}\cdot n\right)+\cdots+\left(\frac{a}{100}\right)^m\cdot n+\cdots$$

$$=\frac{n}{1-\left(\frac{a}{100}\right)}$$

こうやって見積もる事象総数が、計算機上で確保すべきメモリアレイの最大数の目安となる。

2.4　時刻情報と計数情報

上述の計算手法によるイベント事象は、時刻情報として生成・記録される。原子炉雑音解析[3]では、Rossi-α 法やFeynman-α 法等の各手法における評価投影軸への変換プロセスが必要となる。Rossi-α 法であれば任意の事象から複数の後続事象までの時間間隔が評価投影軸となり、Feynman-α 法であれば任意の計測時間幅における計数の分散対平均比が評価投影軸となる。

Orndoffのゲート法[4]に代表される原理的なRossi-α 法の場合、時間間隔の起点事象は観測された全事象が対象となる。Rossi-α プロットの縦軸は時間間隔のヒストグラム数であるため、上記時間間隔のヒストグラムプロセスが必要となる。この様な解析手順を踏むためRossi-α 法では、ヒストグラム幅を調整することで低周波数領域におけるトレンド成分の影響を大幅に軽減することが可能となる。しかしながら、ヒストグラム幅の微細化はヒストグラムプロット形成に必要なイベント数の増大を必要とするため、Roosi-α 法においては計測時間と出力変動への冗長性はトレードオフの関係にある。

本来のFeynman-α 法では各ゲート時間幅における時系列事象は互いに独立である必要があるため、Feynman-α プロットのプロット間隔を微細にしようとする場合には、非常に長い計測時間を必要とする。Feynman-α 法の計測時間を短縮するために考案・導入された手法がバンチング法[5]であるが、当該手法は解析対象時系列の周波数成分に起因すると考えられる疑似トレンド現象を発生させる。また、当該手法と本来のFeynman-α 法との理論的互換性については厳密には検討されていないため、採用にあたっては注意が必要である。いずれの場合においても、時刻情報から計数情報への変換が必要となるが、変換処理過程に伴う保持情報は先のRossi-α 法と比べると格段に少なくて済むプログラム構成を選

択可能なため、より少ないメモリ容量でも解析を行うことが可能である。

2．5　確率分布の判定方法

ある時系列事象が従う確率分布を判定する方法は数多くある。原子炉中性子は核分裂連鎖反応に基づき生成されるため、saturated PMZBB（Pal-Mogilner-Zolotukhin-Bell-Babala）分布と呼称されるガンマ分布の一種に従う。しかしながら、実際に中性子検出器で計測される事象は、炉内での連鎖反応時間情報を幾分か喪失した状態で観測される。これらの事象は観測する時間領域に対して異なった様相を呈し、この変化から原子炉の特性パラメータを取得しようとする試みが原子炉雑音解析法である。

確率分布の判定に用いられる最も簡単な方法が平均値周りの二次モーメントである分散（Variance）を用いる方法である。これは、原点周りの一次モーメントである平均（Mean）と平均値周りの二次モーメントである分散との比を算出する手法であり、この値から従う確率分布を推測する方法である。また、同様の手段で平均値周りの3次モーメントである歪度（Skewness）や平均値周りの4次モーメントである（Kurtosis）と平均値との関係を見る方法が並行して用いられる場合もある。参考までに、表1に代表的な確率分布の平均、分散、歪度、尖度の一覧を示す。

表1：代表的な確率分布の平均、分散、歪度、尖度[6]

確率分布	平均 Mean	分散 Variance	歪度 Skewness	尖度 Kurtosis
ポアソン分布 Poisson Distribution	λ	λ	$\frac{1}{\sqrt{\lambda}}$	$3+\frac{1}{\lambda}$
負の2項分布 Negative Binominal Distribution	$\frac{kq}{p}$	$\frac{kq}{p^2}$	$\frac{1+q}{\sqrt{kq}}$	$3+\frac{6}{k}+\frac{p^2}{kq}$
	$0<k<\infty \quad 0<p<1 \quad q=1-p$			
ガンマ分布 Gamma Distribution	$\alpha\beta$	$\alpha\beta^2$	$\frac{2}{\sqrt{\alpha}}$	$3+\frac{6}{\alpha}$
	$\alpha>0 \quad \beta>0$			
幾何分布 Geometric Distribution	$\frac{1}{p}$	$\frac{q}{p^2}$	$\frac{1+q}{\sqrt{q}}$	$9+\frac{p^2}{q}$
	$0<p<1 \quad q=1-p$			
2項分布 Binominal Distribution	np	npq	$\frac{q-p}{\sqrt{npq}}$	$3+\frac{1-6\cdot pq}{npq}$
	n : *positive integer* $\quad 0<p<1 \quad q=1-p$			
カイ2乗分布 Chi-square Distribution	m	$2m$	$\sqrt{\frac{8}{m}}$	$3+\frac{12}{m}$
逆ガウス分布 Inverse Gaussian Distribution	μ	$\frac{\mu^3}{\lambda}$	$3\cdot\left(\frac{\mu}{\lambda}\right)^{\frac{1}{2}}$	$3+15\cdot\left(\frac{\mu}{\lambda}\right)$
	$\mu>0 \quad \lambda>0$			

2．6　一様乱数から任意乱数への変換方法

乱数を用いたモンテカルロ計算を行う際に必ず必要に技術が、一様乱数から任意の関数に従う乱数への変換技術である。当該技術については様々な解説書が出版されているが、最も判り易いものはW. H. Pressらが執筆した「Numerical Recipes in C」[7]であろう。詳細については文献を参照してもらうこととするが、放射性壊変の再現に用いられる手法について概説を再度ここで行う。

放射性壊変はポアソン事象であるため、互いに隣り合う事象間の時間間隔分布は2．1で述べた通り指数関数に従う。通常のプログラム開発環境等において入手できる乱数メソッドは一様乱数を返す仕様となっているものが一般的である。放射性壊変に係る時間間隔分布は次数1の指数関数であるため、不定積分の逆関数が存在しており、これにより累積密度関数（Cumulative Density Function）を用いた変換手法を採用することが出来る。簡潔に述べると、確率密度関数（Probability Density Function）の積分形の逆関数を用いる方法で、以下の式により指数乱数$Rand_{EXP}$を0～1の一様実数乱数$Rand_{UNI}$から生成することが出来る。

$$Rand_{EXP}=-1\times\frac{ln(Rand_{UNI})}{r}$$

ここで、得られる指数乱数$Rand_{EXP}$の単位は［秒］である。

なお、即発中性子先行核の崩壊時間も同様の手順で生成する。

3．Visual Studio Community 2015 / C#における コーディング

本章では、前述の2に沿ったコーディング例をVisual C#を用いて記述していく。コード例の記載については、Formのロード等に係る記述は省略したコア部分のみの記述とする。また使用する変数等について型宣言等は記述するが、読者の必要に応じて異なる変数型で運用しても構わない。

3．1　ポアソン事象及び増幅事象の時系列データの再現

ポアソン事象及び増幅事象の時系列データを再現するVisual C#でのコード例をコード1に示す。このコードは、ポアソン事象時系列と増幅事象時系列との統合並び替えも含んでいる。なお、並び替え部分については、理解の容易さを優先して記述している。

コード1：ポアソン事象及び増幅事象の時系列データを再現するコード例

```
int j, sorting_int, sorting_searching_int;
int No_of_PoissonEvent, No_of_MultipleEvent;
double present_time, Poisson_Interval, Count_Rate, Time_Length;
double multiple_time, Multiple_Probabirity, Multiple_Interval, Multiple_Decay_Const;

double[,]Time_Stamp;

Random random_No = new Random();


//Time Stamp Generating Part

present_time = 0.0;
No_of_PoissonEvent = 0;
No_of_MultipleEvent = 0;
for (;;)
{
    Poisson_Interval = (-1) * Math.Log(random_No.NextDouble()) / Count_Rate;
    present_time = present_time + Poisson_Interval;
    if (present_time > Time_Length) { break; }
    Time_Stamp[0, No_of_PoissonEvent] = present_time;
    No_of_PoissonEvent++;
    multiple_time = present_time;
    for (;;)
    {
        if (random_No.NextDouble() > Multiple_Probabirity) { break; }
        Multiple_Interval = (-1) * Math.Log(random_No.NextDouble()) / Multiple_Decay_Const;
        multiple_time = multiple_time + Multiple_Interval;
        if (multiple_time > Time_Length) { break; }
        Time_Stamp[1, No_of_MultipleEvent] = multiple_time;
        No_of_MultipleEvent++;
    }
}


// Time Stamp Sorting Part

for (sorting_int = 0; sorting_int < No_of_MultipleEvent; sorting_int++)
{
    for (sorting_searching_int = 0; sorting_searching_int < No_of_PoissonEvent; sorting_searching_int++)
    {
        if (Time_Stamp[0, sorting_searching_int] > Time_Stamp[1, sorting_int])
        {
            No_of_PoissonEvent++;
            for (j = No_of_PoissonEvent - 1; j > sorting_searching_int; j--)
            {
                Time_Stamp[0, j] = Time_Stamp[0, j - 1];
            }
            Time_Stamp[0, sorting_searching_int] = Time_Stamp[1, sorting_int];
            break;
        }
    }
}
```

3．2　時刻列から計数列への変換（Feynman-α/バンチング法）

バンチング法のための時刻列から計数列への変換アルゴリズムをVisual C#で記述した場合のコード例をコード2に示す。このコードは分散対平均比の算出も含んでいる。なおこのコードでの単位ゲート時間幅は1msecに設定してある。

コード2：時刻列から計数列への変換コード例（Feynman-α/バンチング法）

```
int gateTime_roop, gateTime_roop_max;
int time_series_roop, time_series_roop_max;
int gate_number, boot_gate_number, present_gate_count;
double monitor_time_period, present_monitor_time;
double mean_sum, squre_sum;

double[] Time_Series, bunching_Mean, bunching_Variance, bunching_VtoM;

for (gateTime_roop = 0; gateTime_roop < gateTime_roop_max; gateTime_roop++)
{
    gate_number = 0;
    present_gate_count = 0;
    monitor_time_period = (double)(gateTime_roop + 1) / 1000;
    present_monitor_time = monitor_time_period;

    mean_sum = 0.0;
    squre_sum = 0.0;

    for (time_series_roop = 0; time_series_roop < time_series_roop_max; time_series_roop++)
    {
        if (Time_Series[time_series_roop] > present_monitor_time)
        {
            boot_gate_number = (int)((Time_Series[time_series_roop] - (present_monitor_time - monitor_time_period)) / monitor_time_period);
            gate_number = gate_number + boot_gate_number;
            present_monitor_time = present_monitor_time + (monitor_time_period * boot_gate_number);

            mean_sum = mean_sum + present_gate_count;
            squre_sum = squre_sum + Math.Pow(present_gate_count, 2.0);

            present_gate_count = 1;
        }
        else
        {
            present_gate_count++;
        }
    }

    bunching_Mean[gateTime_roop] = mean_sum / gate_number;
    bunching_Variance[gateTime_roop] = (squre_sum / gate_number) - Math.Pow(bunching_Mean[gateTime_roop], 2.0);
    bunching_VtoM[gateTime_roop] = bunching_Variance[gateTime_roop] / bunching_Mean[gateTime_roop];

}
```

３．３　時刻列から計数列への変換（Feynman-α/移動バンチング法）

移動バンチング法のための時刻列から計数列への変換アルゴリズムをVisual C#で記述した場合のコード例をコード3に示す。このコードは分散対平均比の算出も含んでいる。なおこのコードでの単位ゲート時間幅及び移動バンチング法のための初期ゲート移動時間幅は1msecに設定してある。

コード3：時刻列から計数列への変換コード例（Feynman-α/移動バンチング法）

```
int gateTime_roop, gateTime_roop_max;
int time_series_roop, time_series_roop_max;
int gate_number, boot_gate_number, present_gate_count;
double monitor_time_period, present_monitor_time;
double mean_sum, squre_sum;

int bunching_shift_roop , bunching_shift_roop_max, moving_head_boot_number;
double moving_head_shift_period;

double[] Time_Series;
double[] moving_bunching_Mean, moving_bunching_Variance, moving_bunching_VtoM;

for (gateTime_roop = 0; gateTime_roop < gateTime_roop_max; gateTime_roop++)
{
    bunching_shift_roop_max = (gateTime_roop + 1) / 1;
    mean_sum = 0.0;
    squre_sum = 0.0;
    gate_number = 0;

    present_gate_count = 0;
    monitor_time_period = (double)(gateTime_roop + 1) / 1000;
    present_monitor_time = monitor_time_period;
```

```
    for (bunching_shift_roop = 0; bunching_shift_roop < bunching_shift_roop_max; bunching_shift_roop++)
    {
        if (bunching_shift_roop == 0)
        {
            moving_head_shift_period = 0.0;
            time_series_roop = 0;
        }
        else
        {
            moving_head_shift_period = Math.Pow(10.0, (-3.0)) * bunching_shift_roop;
            moving_head_boot_number = 0;
            for (time_series_roop = 0; time_series_roop < time_series_roop_max; time_series_roop++)
            {
                if (Time_Series[time_series_roop] > moving_head_shift_period)
                {
                    moving_head_boot_number = (int)((Time_Series[time_series_roop] - moving_head_shift_period) / monitor_time_period);
                    gate_number = gate_number + moving_head_boot_number;
                    present_gate_count = 1;
                    present_monitor_time = moving_head_shift_period + ((1.0 + moving_head_boot_number) * monitor_time_period);
                    break;
                }
            }
        }

        for (; time_series_roop < time_series_roop_max; time_series_roop++)
        {
            if (Time_Series[time_series_roop] > present_monitor_time)
            {
                boot_gate_number = (int)((Time_Series[time_series_roop] - (present_monitor_time - monitor_time_period)) / monitor_time_period);
                gate_number = gate_number + boot_gate_number;
                present_monitor_time = present_monitor_time + (monitor_time_period * boot_gate_number);

                mean_sum = mean_sum + present_gate_count;
                squre_sum = squre_sum + Math.Pow(present_gate_count, 2.0);
                present_gate_count = 1;
            }
            else
            {
                present_gate_count++;
            }
        }
    }

    moving_bunching_Mean[gateTime_roop] = mean_sum / gate_number;
    moving_bunching_Variance[gateTime_roop] = (squre_sum / gate_number) - Math.Pow(moving_bunching_Mean[gateTime_roop], 2.0);
    moving_bunching_VtoM[gateTime_roop] = moving_bunching_Variance[gateTime_roop] / moving_bunching_Mean[gateTime_roop];
}
```

4．構築コードによるシミュレーション結果

上記コードによる生成時系列の計算結果を図1及び図2に示す。図1はバンチング法により算出したポアソン事象に対する計数値の分散対平均比であり、図2は同一時系列に対して移動バンチング法により算出した計数値の分散対平均比である。各図の黒い実線は、分散対平均比に対する68%誤差である[1]。計算条件を表2に示す。

ポアソン事象である場合、通常、分散対平均比は1となる。また分散対平均比のバラつきは統計的に予測される範囲内に一様に分布するはずだが、図1及び図2から明らかな様に、バンチング法及び移動バンチング法の解析結果はゲート時間方向に向かって何らかのトレンドを有していることがわかる。

理論的には存在し得ない当該トレンド成分は、同一時系列を何回も使い回すバンチング法及びその派生解析法に起因するトレンド成分であると考えられる。計算機によるモンテカルロ計算は比較指標とする理想値を設定しやすいため、この様な現象を解析する際にも大いに役立つと期待される。

なお増幅時の計算結果等については、参考文献[1]を参照して頂きたい。

表2　表示例の計算条件

項目	設定値
計数率［cps］	100
計測時間［sec］	1,000
ゲート時間幅［msec］	1 ～ 500（every 1msec）

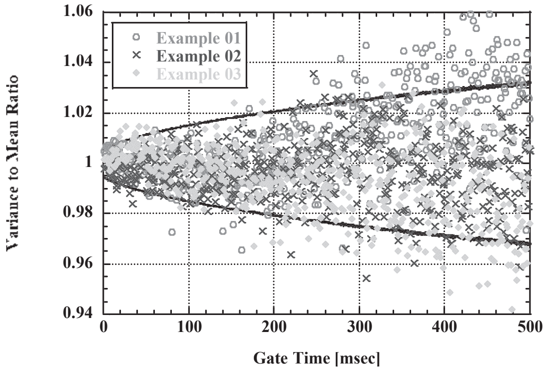

図1：計算により生成したポアソン事象に対する分散対平均比（通常バンチング法により算出）

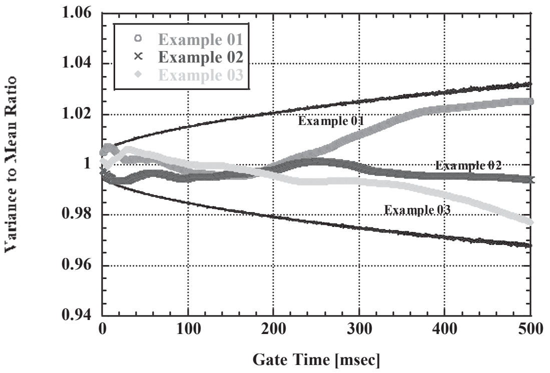

図2：計算により生成したポアソン事象に対する分散対平均比（移動バンチング法により算出）

5．まとめ

本稿では、原子炉雑音解析の基礎研究を目的として本研究室で開発した時間領域モンテカルロ計算コードの基礎理論の解説とコードの例示を行った。一般的に物理系の研究分野では、理論からコードへの落とし込みについては各研究者の知的財産が集約される箇所であることから、一般に開示・明示されることは少ない。しかしながら、計算機を用いた基礎研究においては、理論とコーディング技術が研究進展の両輪の役割を果たす。この様な意味合いにおいては、様々な物理系実処理コードの開示は次世代へ向けた知財管理・継承において重要であり、積極的に行われるべきである。

今回構築したコードは構造が非常に単純でありながら、従来の解析手法の基礎的な研究を行うための強力な道具になり得る。特にこれまでに詳細な調査が行われていなかった原子炉雑音解析手法の基礎特性調査に対して大きな効果を発揮することが期待される。

参考文献

(1) S.Hohara, K.Nakajima, A.Sakon, K.Hashimoto, “An Applied Limit of the Bunching Method for the Feynman-α Analysis”, Journal of Nuclear Science and Technology, Vol.55, No.11, pp1309-1316 (2018)

(2) Glenn F. Knoll, “Radiation Detection and Measurement Third Edition”, John Wiley & Sons, Inc., New York (2000)

(3) M. M. R. Williams, “Random Processes in Nuclear Reactors”, Pergamon Press, Oxford (1974)

(4) John D. Orndoff, “Prompt Neutron Periods of Metal Critical Assemblies”, Nuclear Science and Engineering, vol.2, pp450-460 (1957)

(5) T. Misawa, S. Shiroya, K. Kanda, “Measurement of Prompt Decay Constant and Subcriticality by the Feynman-α Method”, Nuclear Science and Engineering, vol.104, pp53-65 (1990)

(6) 蓑谷 千凰彦, “統計分布ハンドブック（増補版）”, Asakura Publishing Co., Ltd., Tokyo (2010)

(7) William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, “Numerical Recipes in C -The Art of Scientific Computing-

Second Edition", Cambridge University Press,
Cambridge (1992)