平成１２年度

卒業論文

乱数発生法

指導教員

山本　哲也　助教授

電子・光システム工学科

学籍番号：１０１０２７６

木村　麻友子

目次

１．　はじめに

本稿では、プログラミング言語の主流言語ともいえる C 言語を用いて、基本的な算法プログラムを中心に、乱数発生法について検討した。乱数発生法は、確率現象をシミュレーションする重要かつ有効な方法である。確率現象には無数のものがあり、ランダムに生起する各種の素過程が複雑に組み合わせられて観察される現象はすべてこれに属する。この種の対象で工学的に特に興味のあるのは、比較的単純な要素が結合して作られる多要素系で、しかも各要素が何らかの確率的要因を含むとき、それらの集合体としての大システムはどのようにふるまうかという問題である。具体例としては、比較的かんたんなスウィッチ要素が多数集まってできたディジタル・システムや、分岐点（node）が枝線（arc、bond あるいは edge）で複雑につながり合っている回路網（たとえば、大送配電網、親子局をいくつも含む電話回線網、大都市の交通網など）や一般的に大きなシステムの信頼度などの問題がある。

１．１　目的

“ 乱数 ” とは、不規則に生起する数を意味するものである。その “ 乱数 ” を利用するにあたっては、“ 乱数表 ” を使用するという方法もあるが、多数のデータを必要とする場合、あまり便利といえるものではない。一方、プログラムによってコンピュータ上で “ 乱数 ”（疑似乱数）を発生させるという方法がある。この場合、“ 一様性 ” が満たされていると考えられている “ 疑似乱数 ” を発生させる。本研究の目的は “ 乱数発生 ” を数学的な面からも、理解し、その特徴を把握することである。検定法についても、定量的に議論する。

１．２　課題

乱数発生法について、乗算合同法と混合合同法の二つを考える。まず、各々の数学的な内容を理解した後に、実際に C 言語によるプログラミングを行う。一般に広く用いられるのは、混合合同法である。本稿では、それがなぜなのか、比較検討する。加えて、それぞれの乱数の精度に注目し、円周率といった無理数についてどの程度、正確に数値を求められるかについて検討した。

２．乱数発生

本章では、はじめに最も一般的に使われる乱数発生法について議論する。次に C 言語を利用したプログラム、検定を行う。

２．１　乱数とは

サイコロの目や宝くじのあたり番号などは、不規則に生起する。このような、規則性がなく、不規則に変化する数値の列を“乱数”と呼ぶ。サイコロを実際に振って乱数を発生させることもできるが、それを何十万個も必要とするのであれば、現実的ではない。

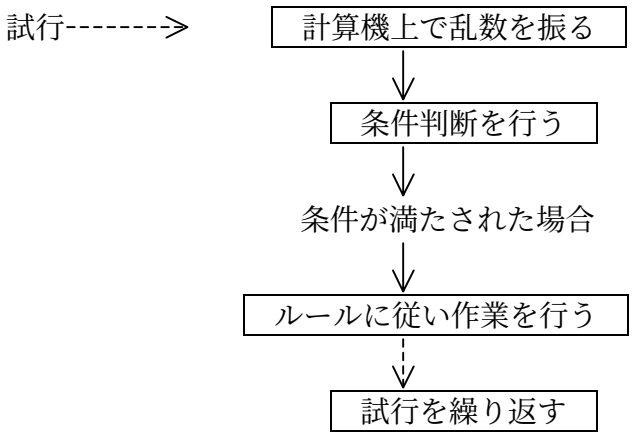
本研究では、コンピュータ上で乱数関数を用いることによって、整数を不規則に発生させる。ただし、ここで発生させた乱数は厳密に言えば、ある数式に基づいた（規則を使う）ものであり、一様な理想的なものではないが、それを模倣した疑似乱数も乱数と呼ぶことにする。

２．２　モンテカルロ法

モンテカルロ法は、フォン・ノイマンとユラムによって 1945 年頃から用いられはじめたものであり、当初は「決定論的な数学の問題を乱数を用いて解くこと」が目的であった。しかし、最近では精度の高い乱数が得られることにより、それを用いた模擬実験（シミュレーション）が盛んとなり、現在では、それをモンテカルロ法と呼ぶ。モンテカルロ法は、確率論的モデルに基づいた確率過程を問題とするのである。

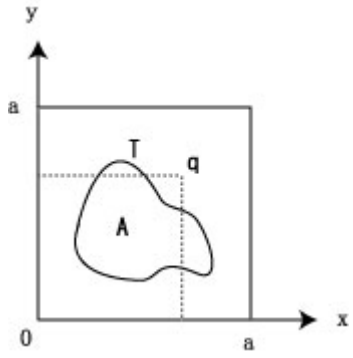
モンテカルロ法では、現象のモデルを構築し、その様子を調べようというものである。例としては、形状の複雑な領域の面積の計算などがある。

シミュレーションでは、乱数を発生しながら確率過程を計算機上で試行していく。その最も基本的なアルゴリズムを以下に示す。



コンピュータ上で発生させた乱数は先にも述べたように、一様に理想的な乱数ではなく、疑似乱数と呼ばれるものである。疑似乱数というのは、理想的な一様乱数を模倣したものであるので、やはり完全に一様なものではない。そこで、性質の良い乱数をいかに多数発生させることができるかどうかが、モンテカルロ・シミュレーションにおいて良質の結果を得るためのポイントとなる。

モンテカルロ法の考え方を紹介する。例として、閉曲線 T で囲まれた部分 A の面積をこの方法で求めるとする。閉曲線 T をすべてその中に含むように１辺 a の正方形をつくり、相隣る２辺をそれぞれ x および y 軸とする。0 以上１以下の乱数 n を２個つくり、これらをそれぞれ n_1、n_2 とする。点 q (an_1, an_2) を正方形内に記入する。（点 q はでたらめにつくったのでランダム点とする）（図 2.2.1）



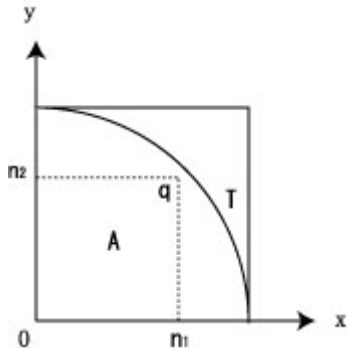
（図 2.2.1）

正方形内に M 個のランダムな点をつくり、T の内にある点の個数を m とする。M の値が十分と近似され、大きいとき、ランダムな点が A 内にある確率は、

$$\frac{\text{Aの面積}}{\text{正方形の面積}} = \frac{\text{Aの面積}}{a^2} \quad \frac{m}{M} \qquad \text{---（2.2.1）}$$

$$\text{Aの面積} \quad \frac{m}{M}a^2 \qquad \text{---(2.2.2)}$$

となる。これが、モンテカルロ法によるシミュレーション効果である。
例として、後にも詳しく述べるがこの A が半径１の円の 1/4 としよう。(図2.2.2)



（図 2.2.2）

$$\text{Aの面積} = \frac{\pi}{4} \quad \frac{\mathrm{m}}{\mathrm{M}} \qquad \text{---（2.2.3）}$$

となり、これよりπの近似値$\hat{\pi}$が求められる。

この場合、閉曲線 T、図形 A が単純なので、モンテカルロ法でπの近似値を求めるというのは、解析的な方法に比べ不利である。高次元の図形の体積を求める場合にはモンテカルロ法が有利といえる。

２．３　一様乱数

一様乱数とは、ある区間において（[0,1] など）数値が同じ確率で出現する、確率分布に従う乱数である。たとえば、サイコロの目は１から６までの数値がどれも 1/6 の確率で出現する一様乱数である。一方、一様でない乱数には、確率が正規分布に従う正規乱数、指数分布に従う指数乱数などがあるが、今回それについては省略する。

一様乱数は、サイコロから生成できるが、前にも述べたように、何十万個も数列を必要とする場合には、大変不便である。そこで乱数を得るために、コンピュータを利用する方法が最もよく使われる。コンピュータで計算することにより、乱数を生成させられるので、一見でたらめな数の集まりに見えるが、厳密に言えば一様に理想的なものではなく、一定の周期性が存在することを注意しておく。

２．４　合同法乱数

モンテカルロ法を用いる場合に、まず乱数を発生させなければならない。一様乱数の生成法として最も一般的に使われてきたのは、レーマーが 1948 年頃に発表した線形合同法（linear congruential method）である。漸化式は、

$$X_n = aX_{n-1} + b \pmod{\mathrm{M}} \qquad \text{---（2.4.1）}$$

で表され、非負整数列X_nを生成する。

以下に、その発生法及び、C 言語によるプログラミングの内容を述べる。

２．４．１　乗算合同法

乗算合同法は２数の積をある数で割って、余りをとる方法である。式 2.4.1 において、$b=0$の場合を乗算合同法といい、漸化式は

$$X_n = aX_{n-1} \pmod{\mathrm{M}} \qquad \text{---（2.4.2）}$$

で表される。この漸化式より、数列x_0、x_1、x_2…を発生させる。aは定数、M は係数を示す。（a、M は整数）なお、aと M の選択法については後で述べることにする。乗算合同法の周期は M に等しくすることは出来ない。よって、aの値をどのように選んでも、周期は M より小さくなる。

ここで、乗算合同法を用いた計算例を例１として示す。

例１． 乗算合同法を用い、一桁の一様乱数を作る。ただし、定数a＝2045、係数 M＝2^{15}＝32768、x_0＝257 の値を用いる。

【計算例】

漸化式 $X_n = aX_{n-1} \pmod{M}$ において、$n=1$から順にx_1、x_2…を計算する。

実際、

$$\begin{aligned} x_1 &= a \cdot x_0 \pmod{2^{15}} \\ &= 2045 \times 257 \ (\mathrm{mod}\ 2^{15}) \\ &= 2045 \times 257 - \left[\frac{2045 \times 257}{32768} \right] \times 32768 \\ &= 2045 \times 257 - 16 \times 32768 \\ &= 1277 \end{aligned}$$

となる。［ ］はガウスの記号を表し、［ ］内の数値の整数部をあらわす。

$\frac{2045 \times 257}{32768}$ 16.038 なので、上例では 16 となる。

ここで、一桁の乱数y_1を求めたい場合は、x_1の値を、M の値を 10 で割った値で割ればよい。

$$y_n = \frac{x_n}{3276.8}$$

これにより、一桁の乱数列y_nが求められる。

よって、y_1＝1277÷3276.8 0.38 となり、0 という数値が得られる。同様にして、さらにx_2を求めてみる。

$$\begin{aligned} x_2 &= 2045 \times 1277 - \left[\frac{2045 \times 1277}{32768} \right] \times 32768 \\ &= 2045 \times 1277 - 79 \times 32768 \\ &= 2611465 - 2588672 \\ &= 22793 \end{aligned}$$

よって、y_2＝22793÷3276.8 6.95 となり、6 という数値が得られる。

この様にして、順次１桁の乱数列y_nを求めていく。この乱数は、整数区間［0,9］の一様分布乱数となる。

次に、C 言語によるプログラミングを行う。アルゴリズムとしては整数型乱数をソフトウェア上で発生させ、係数で割ることにより［0,1］上の一様乱数を得るというものである。定数、係数、x_0の値については、上記の計算例と同様のものを使用する。

プログラム１　乗算合同法による一様乱数の発生

```
#include <math.h>
#include <stdio.h>
#define RDX 10

main() {
        int rb , rmax=32768;
        int arb[100];
        int i,nmax,number;
        int rnd_b();
        printf(" uniformity random number generation\n");
        printf("-----------------------------------\n");
        printf("multiplication congruence method\n");
        printf("-----------------------------------\n");
        printf("number of data= "); scanf("%d",&nmax);
        for(i=0;i<100;i++) arb[i]=0;
        while(nmax--) {
                        rb=rnd_b();
                        arb[rb/(rmax/RDX)]+=1;
         }
        for(i=0;i<RDX;i++)
                printf("no=\t%d\t\t%d\n",i,arb[i]);
                printf("type any key:");
}

int rnd_b()    /* multiplication congruence method  */
{
static long unsigned int x=257, a=2045, p=32768;
        x=((a*x)%p);
        return((int)x);
}
```

〔変数の宣言〕

整数型のデータは、先頭の１ビットで数値の符号、残りのビットで数値自身の大きさを示す２の補数表現で表される。なお、負の整数値を扱わない場合は、整数型のデータ型を修飾子unsignedで修飾すると、すべてのビットが数値の表現に用いられる。

```
unsigned　　整数型のデータ宣言　　変数/配列の並び；
```

〔static 変数〕

キーワード static を付けて変数を宣言することにより、その変数は static の属性をもつ static 変数となる。static 変数とは、値が変わるとその値を保持する。一方、宣言文で static がつかない変数は、auto 変数とよばれ、副関数の中の auto 変数は、関数が呼び出される毎に初期値が設定される。

```
static long unsigned int x=257, a=2045, p=32768;
```

〔引数のなしの関数呼び出し／定義〕

関数側へデータを渡す変数を引数という。引数なしの関数を定義するには、関数名の後に（　）をつける。呼び出す場合も、同じく（　）をつける。

```
int rnd_b()
```

プログラム１より、0から９までの整数を試行回数10^6回までランダムに発生させ、各回数ごとにヒストグラムを作成したところ（参考資料①参照）、乱数発生回数が増えるにつれ、各数値の出現回数が一様になっていく様子がわかる。発生させた乱数を一様乱数と認められるかの検定については後で詳しく説明する。

乗算合同法は混合合同法にくらべ、決して広く用いられていないが、その理由の一つとして、数列の周期が容易に計算できないことが指摘されている。乗算同法を用いる場合には、非常に多くのサンプリングを行う必要がある。

２．４．２　混合合同法

混合合同法とは、乗算合同法を加数bを加えて修正したものであり、漸化式

$$X_n = aX_{n-1} + b \quad (\text{mod M})$$

で表される。要するに、$aX_{n-1}+b$の結果をMで割った余りをX_nとする。aは定数、Mは係数、bは加数である。a、M、bはそれぞれ整数とする。
上記の漸化式よりx_0からx_1、x_2…を発生させていく。x_0の値はもちろん整数とする。

では、混合合同法を用いた計算例を例２として示す。
例２．混合合同法を用い、一桁の一様乱数を発生させる。ただし、定数a＝2045、係数M＝2^{15}＝32768、x_0＝257、b＝9378の値を用いる。
【計算例】
漸化式　$X_n = aX_{n-1} + b \quad (\text{mod M})$　において、n＝0から順にx_1、x_2…を計算していく。

$$\begin{aligned} x_1 &= 2045\times257+9378\ (\text{mod}\ 2^{15}) \\ &= 2045\times257+9378-\left[\frac{2045\times257+9378}{32768}\right]\times32768 \\ &= 534943-16\times32768 \\ &= 534943-524288 \\ &= 10655 \end{aligned}$$

ここで、一桁の乱数y_1を求めたいので、乗算合同法の場合と同様、x_1の値を、Mの値を10で割った値で割ればよい。よって、

$$y_1 = 10655 \div 3276.8 \quad 3.25$$

より、この整数部をり、３が乱数として得られる。同様にx_2を求める。

$$\begin{aligned} x_2 &= 2045\times10655+9378\ (\text{mod}\ 2^{15}) \\ &= 2045\times10655+9378-\left[\frac{2045\times10655+9378}{32768}\right]\times32768 \\ &= 21798853-665\times32768 \\ &= 21798853-21790720 \\ &= 8133 \end{aligned}$$

となり、一桁の乱数y_2は、y_2＝8133÷3276.8　2.48より、２が乱数として得られる。以後、次々に発生させていくと整数区間［0､9］の一様乱数が得られる。

C言語によるプログラミングを行う。定数、係数、x_0の値については、上記の計算例と同様のものを使用する。

プログラム２　混合合同法による一様乱数の発生

```
#include <math.h>
#include <stdio.h>
#define RDX 10

main() {
        int rc, rmax=32768;
        int arc[100];
        int i,nmax,number;
        int rnd_c();
        printf(" uniformity random number generation\n");
        printf("-----------------------------------\n");
        printf("mixed congruence method\n");
        printf("-----------------------------------\n");
        printf("number of data= "); scanf("%d",&nmax);
        for(i=0;i<100;i++) arc[i]=0;
        while(nmax--) {
                        rc=rnd_c();
                        arc[rc/(rmax/RDX)]+=1;
        }
        for(i=0;i<RDX;i++)
                        printf("no=\t%d\t\t%d\n",i,arc[i]);
                        printf("type any key:");
}

int rnd_c()     /*  mixed congruence method  */
{
 static long unsigned int a=2045 ,p=32768 ,x=257 ,b=9378;
        x=(((a*x)+b)%p);
        return((int)x);
}
```

プログラム２より、０から９までの整数を試行回数１０^{6}回までランダムに発生させ、各回数ごとにヒストグラムを作成する（参考資料②参照)。

プログラム２の乱数関数は、計算例にしたがって作成したものである。参考までに、一般に広く使われている混合合同法を用いた乱数関数を使用したプログラムを紹介する。

プログラム３　混合合同法を用いた一様乱数の発生

```
#include <stdio.h>
#include <stdlib.h>
int main() {
        int n; /*number of data */
        float rnd();
        float mat[5000000]; /*variable of matrix */
        float imat[5000000];
        int i,j; /*number if itration */

        int x0,x1,x2,x3,x4,x5,x6,x7,x8,x9;
        float temp; /* variable of exchange */
        x0=0;x1=0;x2=0;x3=0;x4=0;x5=0;x6=0;x7=0;x8=0;x9=0;

    printf("\n");
    printf("input number of data n=  ");
    scanf("%d",&n);

    for(i=0; i<n; i++)
    {
        mat[i]=rnd();
        imat[i]=(int)(10.0*mat[i]);
        if(imat[i]==9) x9=x9+1;
        else if(imat[i]==8) x8=x8+1;
          else if(imat[i]==7) x7=x7+1;
            else if(imat[i]==6) x6=x6+1;
              else if(imat[i]==5) x5=x5+1;
                else if(imat[i]==4) x4=x4+1;
```

```
                        else if(imat[i]==3) x3=x3+1;
                          else if(imat[i]==2) x2=x2+1;
                           else if(imat[i]==1) x1=x1+1;
                            else if(imat[i]==0) x0=x0+1;
        }

/*the number of occurrence for each number*/
        printf("\n");
        printf("0=%d\n",x0);
        printf("1=%d\n",x1);
        printf("2=%d\n",x2);
        printf("3=%d\n",x3);
        printf("4=%d\n",x4);
        printf("5=%d\n",x5);
        printf("6=%d\n",x6);
        printf("7=%d\n",x7);
        printf("8=%d\n",x8);
        printf("9=%d\n",x9);
}

/*random function */
float rnd()
{
                static long ix=1643;
                long ia=1664525;
                float r;
        ix=ia*ix+37313;
        r=ix*2.32830643658698e-10;
                if(ix<0) r=r+1.0;
                return r;
}
```

プログラム３の乱数関数は、初期値ix＝1643、定数ia＝1664525、加数＝37313とした、0以上１未満の「疑似一様乱数」を発生させるものである。さらに、プログラム上で得られた疑似乱数を10.0倍し、int型で宣言することにより、0以上９以下の乱数を発生させている。

２．５　法Mの選択

線形合同法　$X_n = aX_{n-1} + b \pmod{M}$　から生成される非負整数列X_nの周期は、M より長くはなりえない。したがって、M はかなり大きくとっておく必要がある。そして、疑似乱数において、周期は M に等しくなることが望ましいのである。よく行われる選択法としては、使用する計算機（もしくはプログラム言語）で表現可能な数（最大の正整数＋１）をとる。実際には、使用するコンピュータの能力や、計算時間に合わせる。d 進法e 桁であれば、M＝d^e である。しかし、M＝d^e とした場合には、数列X_nの下位の桁があまりランダムにならないという欠点がある。

乗算合同法の M の選択法としては、M が素数であり、a^M ＝１（mod　M）を満たすことが望ましいとされる。一方、混合合同法の場合は、一般に M は取り得る最大の整数である M＝2^{31}－１＝2147483647 を採用する。なお、プログラム１、２で採用した M＝2^{15}＝32768 という値は、今回プログラムを行う上での表現可能な最大の整数である。

２．６　定数 a の選択

定数aは、$0 < a <$ M であり、非負整数列X_nの周期がなるべく長く、かつランダムになるように選ぶ必要がある。この値をどう選ぶかによって、乱数の精度のよさが決まる。乗算合同法の場合、a^M ＝１（mod　M）を満たすことと、$0 < m <$ M になるすべての整数mに対して、$a^m \neq 1$が成り立つことが望ましいとされる。良い乱数を得るための、aの選び方について、長年さまざまな研究が行われてきた。たとえばクヌース（D.E.Knuth）によれば、aは$a \bmod 8$＝５、$M/100 < a < M - \sqrt{M}$ を満たし、その２進数表示が特殊なパターンをもたないことが良いとされている。

次に混合合同法における、加数bの選択法についてだが、その前に一様乱数の検定について、議論しよう。法M、定数a、加数bの三つの値と、数列の初期値に使用する値は、それぞれつり合いが取れていないと、プログラム上でうまく疑似乱数が発生しないことが判明した。よって、必ず発生させた乱数の検定を行う必要がある。

２．７　乱数の検定

コンピュータ上で乱数を生成した場合、乱数を検定する必要がある。それは、分布の模様、周期、系列相関などの検定を必要とする。代表的な検定法として、たとえば次のようなものがある。

（a）　頻度検定 (frequency test)
（b）　連の検定 (run test)
（c）　ランダムウォーク検定 (random walk test)

２．８　一様性の検定

プログラム１、２により発生させた疑似乱数が理想的な乱数であるかどうか、一様性の検定を行う必要がある。今回は、一様乱数の発生を行ったので、平均、積率を調べる方法を採用した。区間[0,1]で発生させた疑似一様乱数を、x_n とすると、

平均　　$\sum_{n=1}^{N} \frac{x_n}{N} = \frac{1}{2}$

２次積率　　$\sum_{n=1}^{N} \frac{x_n^2}{N} = \frac{1}{3}$

３次積率　　$\sum_{n=1}^{N} \frac{x_n^3}{N} = \frac{1}{4}$

の値に近づくことが知られている。N は乱数発生の試行回数である（n＝１、２、…、N)。では実際に、乗算合同法により、発生した一様乱数の検定を行う。

乗算合同法による乱数発生と、その検定を行うプログラムは参考資料③を参照して頂きたい。初期値、乗数、係数については、プログラム１と同様のものを使用する。試行回数 1000 回（N＝1000）までの検定結果を図 2.8.1 に示す。

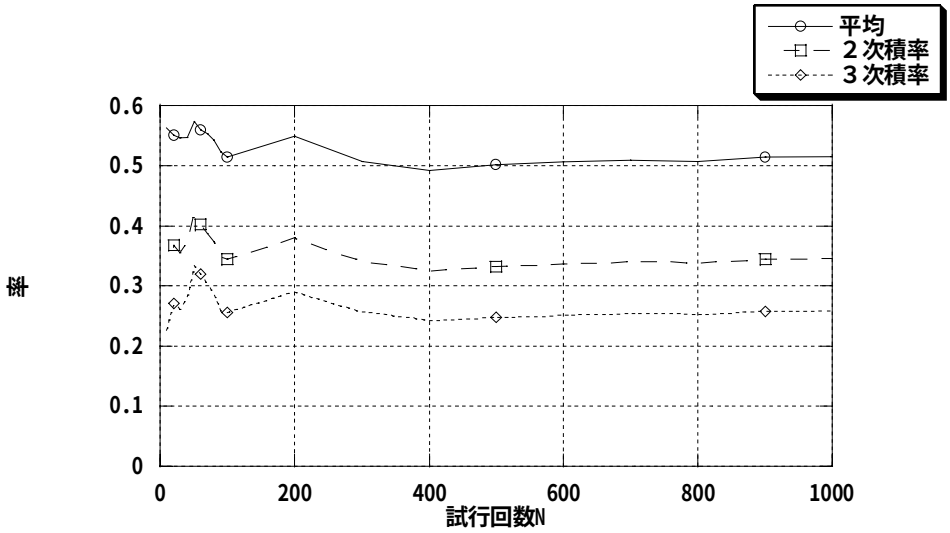


図 2.8.1　一様乱数の検定

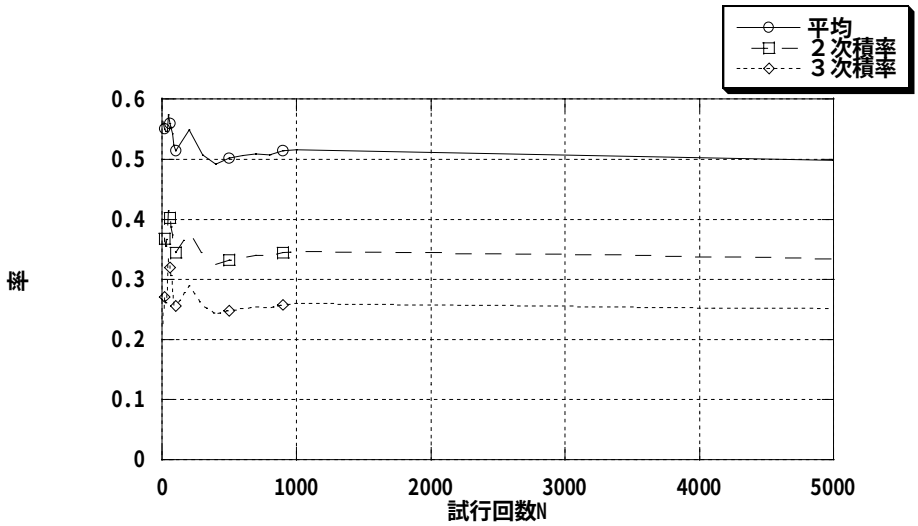


図 2.8.2　一様乱数の検定

図 2.8.1 によると、試行回数 500 回ほどで、平均、２次積率、３次積率、共に値が、それぞれ望ましい値に収束している。よって、プログラム１で発生させた乱数は疑似一様乱数と認められる。同様に、プログラム２の混合合同法により発生させた一様乱数の検定も行う。乱数発生と検定を行うプログラムは参考資料④、検定結果をグラフ化したものは、参考資料⑤で参照して頂きたい。

２．９　加数ｂの選択

加数bの値は、得られる乱数のランダムネスに大きく影響しないとされているが、本当にそうなのであろうか。一般に加数bは、$0 \leq b <$M の範囲で決められる。プログラム２（混合合同法における乱数発生プログラム）において、加数bに 9378 という値を用いている。

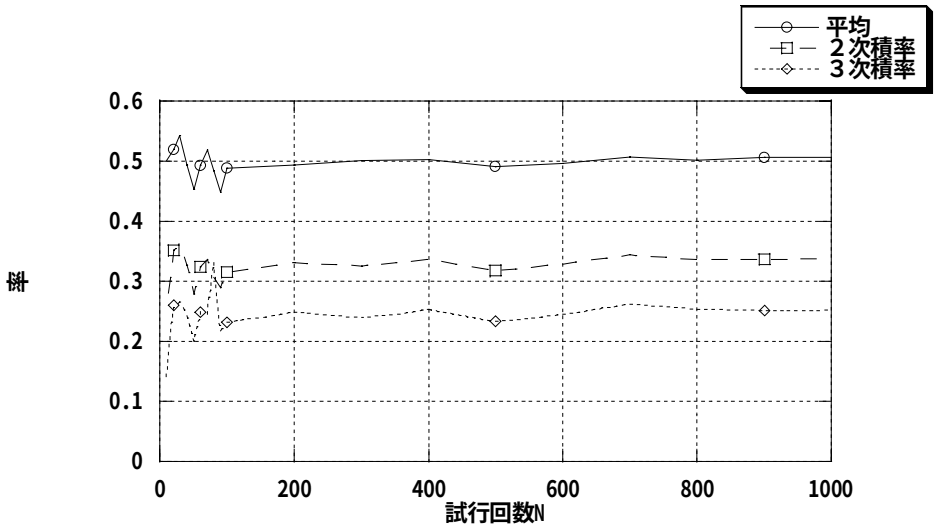


図 2.9.1　一様乱数の検定

図 2.9.1（参考資料⑤より抜粋）の検定結果より、各数値は試行回数 900 回ほどで収束している。では、プログラム２（混合合同法における乱数発生プログラム）において、加数bの値が、極端に小さい場合と大きい場合で一様乱数の検定を行ってみる。まず、加数b＝５とした場合の、試行回数 1000 回の結果をグラフ化してみる（図 2.9.2 参照)。

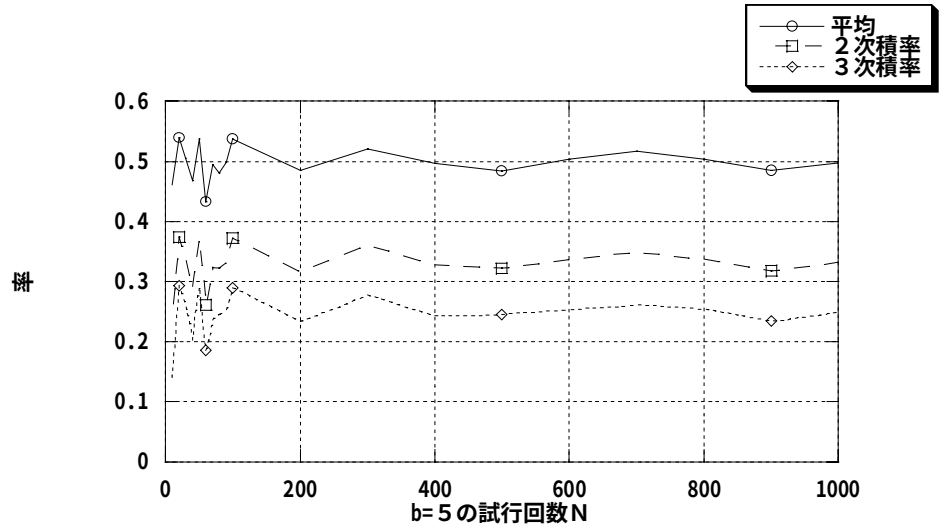


図 2.9.2　加数＝５の場合の検定

b＝５の場合、試行回数 1000 回では各値が、それぞれ望ましい値に収束していない。では、加数＝999999 とした場合はどうか。

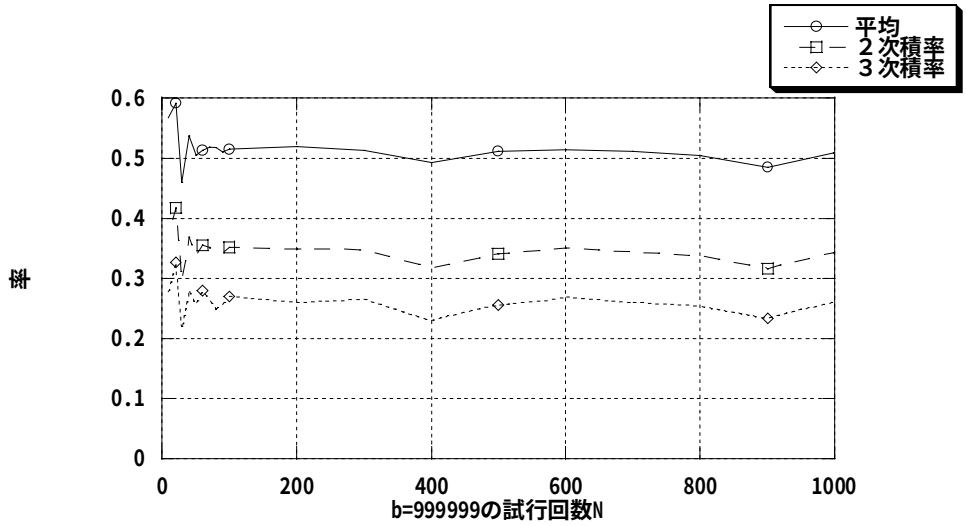


図 2.9.3　加数＝999999 の場合の検定

この場合も同様、試行回数 1000 回では各値は望ましい値に収束していない（図 2.9.3 参照）。ただし、加数が小さい場合も大きい場合も試行回数 5000 付近になると、収束する（参考資料⑥参照）。

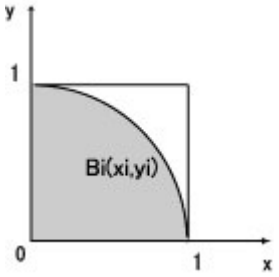
以上のことより、加数bの値は試行回数が多い場合はランダムネスには大きく影響しないが、プログラム２の乱数発生プログラムでは、数値の振動が見られないため、9378 という値を用いるのが、適当ではないだろうか。

３．　モンテカルロ法の応用

これまでの項では、一様乱数を発生させることを目的とした。当項からは、乱数を得ることにより、それを用いた模擬実験（シミュレーション）を行うモンテカルロ法の簡単な応用を行う。

３．１　円周率の計算

モンテカルロ法は決定論的問題にも応用される。その解析の一例として円周率πの計算を考える。例えば半径aの円の面積はπa^2であるが、このときのπの値は円周と直径の比という形で定義される。０と１の間にある乱数列をとり、そのうち２つずつを組とする。それらを(x_i, y_i)とし、$i = 1, 2\cdots$とする。この乱数の組(x_i, y_i)は、(x, y)平面上で１つの点に対応する。対応する点をB_iとする。多数の乱数組をとることで、多数の点B_iが得られる。点B_iは正方形の内部にほぼ一様に分布する。この内部でb個の点を得たとする。そのうちの中心が原点であり半径１の円の内部にあるものの数をnとする。
このとき、正方形の面積は４、円の面積はπなので、n/bは$\pi/4$の近似値を与える（図ウ）。



（図 3.1.1）

モンテカルロ法による円周率πの計算では、出来る限りの点B_iをつくり、それら点B_iのうち円の内部にあるものの個数を計算すればよい。点B_iの一つ一つが異なるサンプルとすれば、その点の数を大きくしていけば、原理的には結果は、正確なπの期待値に近づくことは言うまでもない。

３．２　乗算合同法によるπの計算

一般的にπ（円周率）の計算を行う場合、プログラム３で使用した乱数関数（混合合同法）を使用する。では、乗算合同法を使用した場合はどうか。πを求めることに何か不都合が生じるのであろうか。実際にπの計算を行ってみようではないか。

そこで、まず確認しておきたいのが、乱数関数で使用する各数値である。シミュレーションを行う際には、確かな疑似一様乱数を発生させたい。0 以上１以下の疑似一様乱数を乗算合同法によって発生させるので、関数プログラムは次のようになる。

プログラム４　乗算合同法による乱数関数のプログラム

```
/* rand function */
float rnd()
{
        static long ix=1643;
        long  a=69069;
        float r;
 ix=a*ix;
r=ix*2.328306436538698e-10;
if (ix<0)r=r+1.0;
        return r;
}
```

---

ix＝1643 が乱数列の初期値、a＝69069 が定数とし、係数には 2^{-32} を使用している。2^{-32}＝2.328306436538698e-10 は e 符号付（底 10 の指数部付き）浮動小数点表現で、$2.328306436538698\times10^{-10}$ を意味している。その値は 2^{32} 分の１である。では、これらの値を採用するべきか、検定を行うとしよう。試行回数は 10^6 回とする。検定プログラムは参考資料⑦を参照して頂きたい。

試行回数 10^3 回までと 5×10^3 回までのグラフを次に示す。

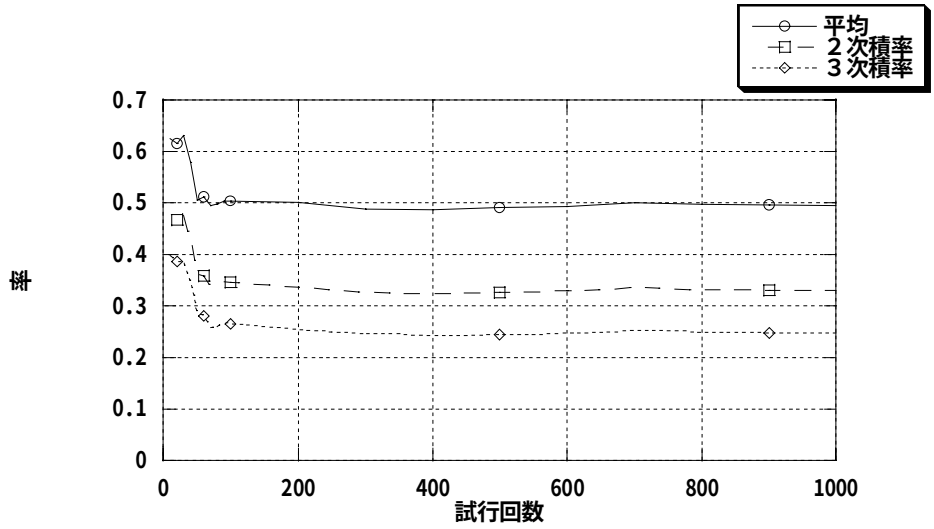


図 3.2.1　乗算合同法による一様乱数の検定

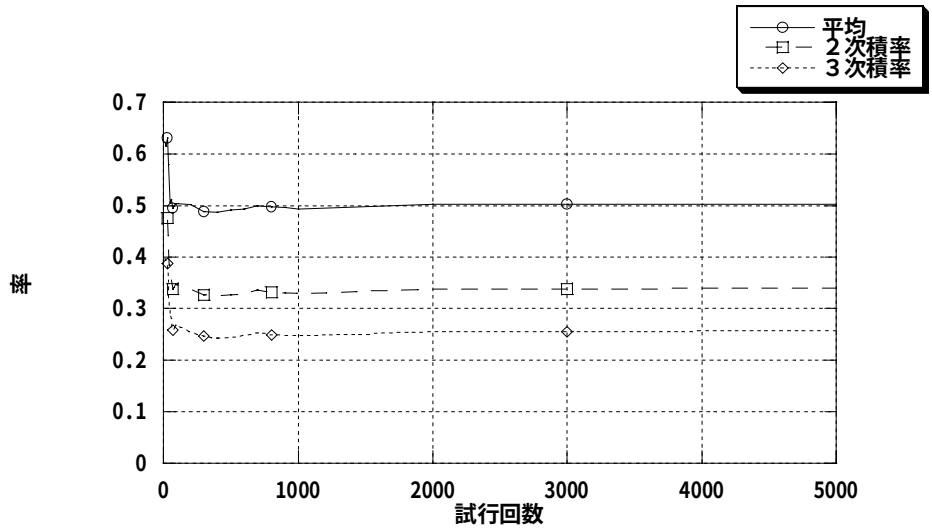


図 3.2.2　乗算合同法による一様乱数の検定

試行回数 500 回までは、一様性は認められないが、試行回数が 5000 回までになると、各値に収束してきているのがわかる。つまり、プログラム４の乗算合同法を用いた乱数関数から、発生される乱数は、疑似一様乱数と認められる。(試行回数 300 回、500 回、10^4 回のグラフを参考資料⑧として添付しておく。)

では、以上の乗算合同法による乱数発生プログラムを用いて、π（円周率）の計算に移る。作成したプログラムを次に示す。

プログラム５　乗算合同法による円周率（π）の計算

```
#include <stdio.h>
#include <math.h>
              float fan();
              float rnd();
main(){
              int n,i;
              float   b,s;
        printf("the number of scatter sands ");
        scanf(" %d", &n);
        b=0;
        for (i=1;i<=n;i++){
              if  (fan(rnd(),rnd())<=1)  b=b+1.0;
        }
        s=b/n;
        printf("the number of scattered sands    =%d\n",n);
        printf("the number of sands in fan  =%f\n", b);
        printf("area of circle     =%f\n",4.0*s);
}
/* fan function  */
        float   fan (x,y)   float x,y; {
        return x*x+y*y;
        }
/* rand function */
float rnd()
{
              static long ix=1643;
              long  a=69069;
              float r;
        ix=a*ix;
        r=ix*2.328306436538698e-10;
        if (ix<0)r=r+1.0;
        return r;
}
```

プログラム５を実行し、円の面積（πの値）が、3.1415926 に限りなく近い値になるまで、試行を繰り返す。実行結果の一例を次に示す。

プログラム５の実行結果

```
--------------------------------------------------------------------------------
the number of scatter sands 10
the number of scattered sands    =10
the number of sands in fan  =8.000000
area of circle      =3.200000

the number of scatter sands 1000
the number of scattered sands    =1000
the number of sands in fan  =779.000000
area of circle      =3.116000

the number of scatter sands 100000（＝10^5）
the number of scattered sands    =100000（＝10^5）
the number of sands in fan  =78597.000000
area of circle      =3.143880

the number of scatter sands 1000000（＝10^6）
the number of scattered sands    =1000000（＝10^6）
the number of sands in fan  =785209.000000
area of circle      =3.140836
--------------------------------------------------------------------------------
```

実行結果より、試行回数 10^5～10^6 回ほどで、理想値に近づく。つまり、試行回数が多いぶん（乱数を多く発生させる場合）には、問題はないということだ。比較するために、混合合同法より乱数を発生させ、円周率を計算してみよう。乱数関数はプログラム３のものを使用し、他の計算プログラムはプログラム５と同様である（参考資料⑨参照）。

結果は、試行回数 10^4 回ほどで理想値に、近づく。よって、混合合同法を用いたほうが乱数を発生させる回数が、少なくて済むといったことが本研究より明確となった。

４．　まとめ

第２章では、最も一般に広く使われている「乱数発生法」について、学習、C言語でのプログラミング、第３章では、それを用いての簡単なシミュレーションについて述べた。本章では、それについてまとめよう。

“乱数”とは、規則性がなく、不規則に変化する数値の列である。コンピュータ上で、“乱数”を発生させるとき、それは厳密に言えば一様な理想的なものではない。しかし、乱数を用いたシミュレーションを行う場合、多数の乱数を必要とするので、コンピュータ上で、いかに質の良い乱数を発生させるかが、課題である。

【一様乱数】

一様乱数は、ある区間において（[0,1]など）数値が同じ確率で出現する、確率分布に従う乱数である。しかし、それらの乱数は数学的には一様な理想的な乱数ではなく、疑似一様乱数とよばれる。

疑似一様乱数は理想的な一様乱数を模倣したものであるから、個々の値には非常に長いが、周期が見られる。周期は長ければ長いほど、望ましい。

ソフトウェア上で、乱数を発生させる場合、整数型乱数をレーマー法により発生させる方法が有名である。これは漸化式 $X_n = aX_{n-1} + b \pmod{M}$で表される。$b = 0$ のときを乗算合同法、$b \neq 0$ のときを混合合同法という。

乗算合同法では、法Mが素数で $a^M = 1 \pmod{M}$ を満たし、定数aにかんしては、$0 < m < M$ になるすべての整数mに対して、$a^m \neq 1$が成り立つことが望ましい。

一方、混合合同法では、一般にMは取り得る最大の整数である $M = 2^{31} - 1 = 2147483647$ を採用する。加数bの値は、得られる乱数のランダムネスに大きく影響しないとされている。実際に、bの値が極端に小さい場合と、極端に大きい場合で、プログラミングしてみたところ、多数の乱数を用いる場合には、大きく影響しないとみてかまわないであろう。しかし、実際小さい値のほうが理想的な乱数発生に近いことが、本研究での結論である。

定数、係数、加数、乱数の初期値は、それぞれ理想的な値があり、それは一様乱数の検定を行うことによって、確認できる。

【一様乱数の検定】

コンピュータにより発生させた疑似乱数を乱数と認めてよいかどうか、検定を行う必要がある。乱数の検定には、分布の模様、周期、系列相関などの検定が必要であるが、一様乱数の場合には、平均、２次積率、３次積率を調べる方法がある。0 以上１以下の区間で発生させた疑似一様乱数を x_n とした場合、

平均　$\sum_{n=1}^{N} \frac{x_n}{N} = \frac{1}{2}$

２次積率　$\sum_{n=1}^{N} \frac{x_n^2}{N} = \frac{1}{3}$

３次積率　$\sum_{n=1}^{N} \frac{x_n^3}{N} = \frac{1}{4}$

の値に近づくことが知られている。

【モンテカルロ法】

モンテカルロ法とは、乱数を用いたシミュレーションで値を近似する方法をいう。今回は、円周率の計算を行った。半径１の円の面積を求めることにすると、答えは円周率となる。

円周率を求めるプログラムを、乗算合同法、混合合同法の二種類作成する。

それぞれの結果から、コンピュータ上で乱数を発生させるためには、用いる乱数が極端に多い場合は、どちらの方法を用いても、大した影響は見られないといえる。しかし、本研究でわかったことは、乱数の発生数が少なくても信頼がおける乱数を必要とする場合には、混合合同法を用いる方が、適当であることがわかった。これが、広く用いられている理由ではないかと判断した。

参考文献

今回、この論文を書くにあたって、参考とした文献を紹介する。

【1】 定道宏監修、布上康夫・中原明宏共著、
文系のための C 基礎実習－算法とプログラミング－（オーム社、1994）

【2】 宮武修・脇本和昌共著、　乱数とモンテカルロ法（森北出版、1978）

【3】 伏見正則著、　乱数（東京大学出版会、1994 第二版）

【4】 川添良幸・三上益弘・大野かおる著、
コンピュータシミュレーションによる物理科学－分子動力学とモンテカルロ法
（共立出版、1997 第二版）

【5】 藪下信著、現代の数理科学シリーズ①計算物理（Ⅰ）（地人書館、1982）

【6】 津田孝夫著、モンテカルロ法とシミュレーション　電子計算機の確率論的応用
（培風館、1977）

おわりに

正確な乱数を生成することは簡単ではない。私たちが使用する乱数とは、常に有限のものである。そして、その有限の数列を乱数らしくなるように発生させなければならない。そのために、検定を行うことが重要かつ必要である。本研究では、乱数を発生させるにあたり、混合合同法と、あまり用いられていない乗算合同法に注目した。あえて、乗算合同法に注目したのは、その理由を軽々しくは判断できないからである。

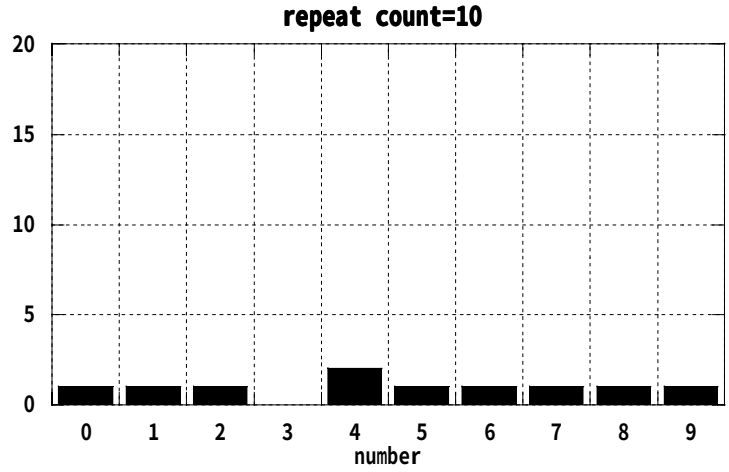
モンテカルロ法は、乱数を用いた模擬実験（シミュレーション）であるが、これによって得られた解は、多くの場合粗雑なものである。ただ、本研究のように、問題が確率論的なものであるときは、サンプルの個数を増やすことが、解の精度を上げる直接的な方法であることがわかった。

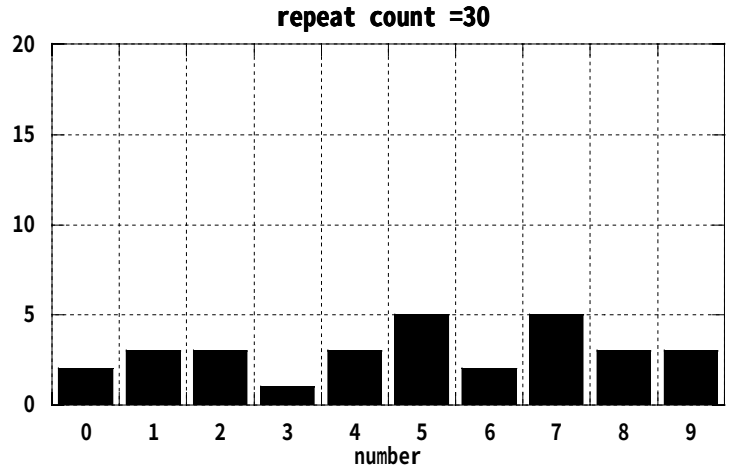
本研究を行い始めた当初は、調査研究であったが、後半は自分なりの研究ができた。無から有を創り出すことは稀であり、多くの場合は既成の理論の批判から始まるものである。ただ、その批判を自分でやらなければ、結果は何も生まれないのではないだろうか。

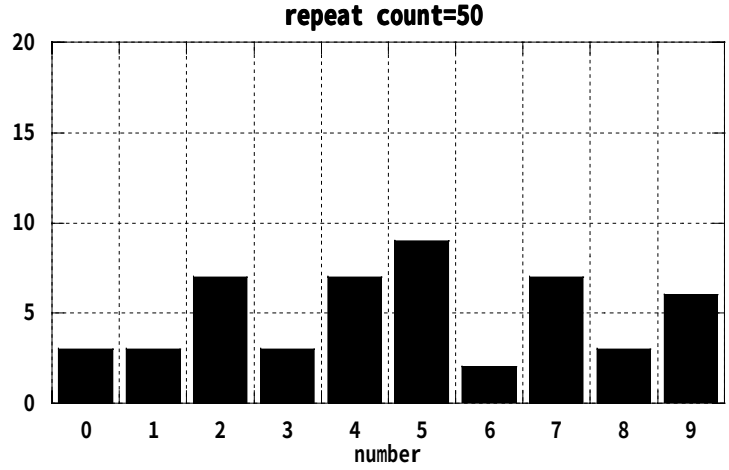
謝辞

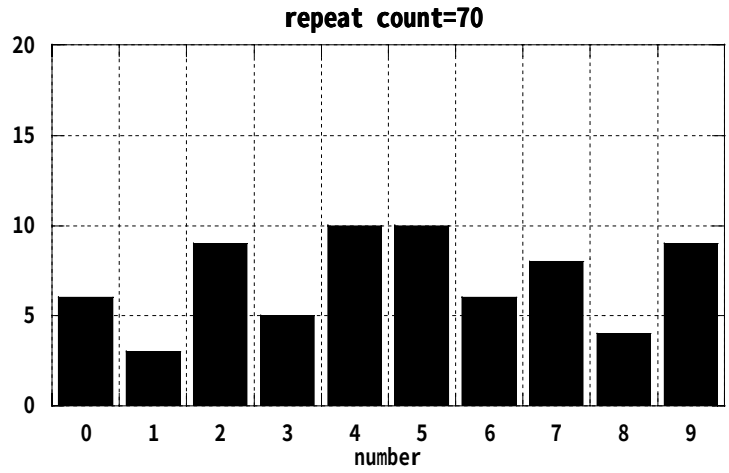
本論文を書くにあたっては、指導教員である山本哲也助教授にたくさんの御助言、ご指導をいただいた。山本哲也助教授と直接参考にさせて頂いた、参考文献の著者の方々に心から感謝の意を表したい。

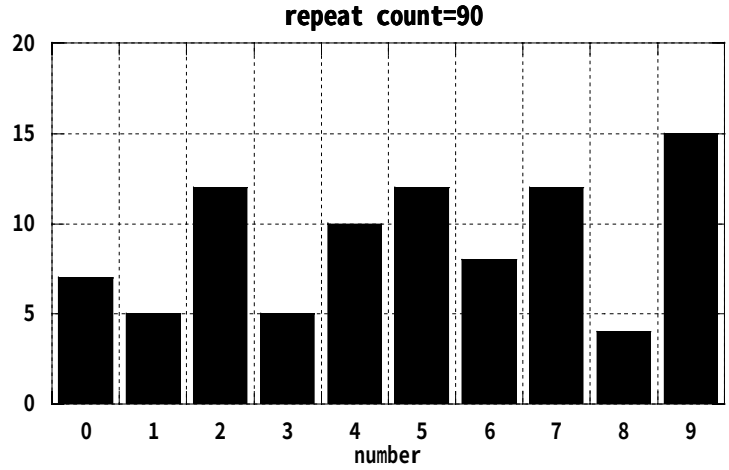
参考資料①　乗算合同法による一様乱数発生/各数値の出現回数

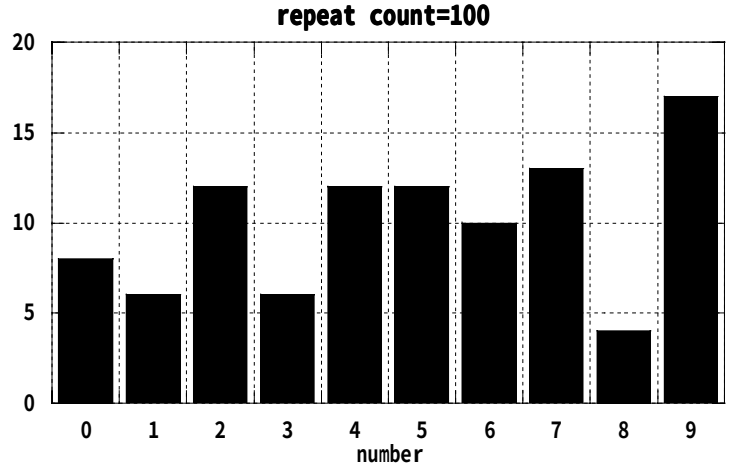


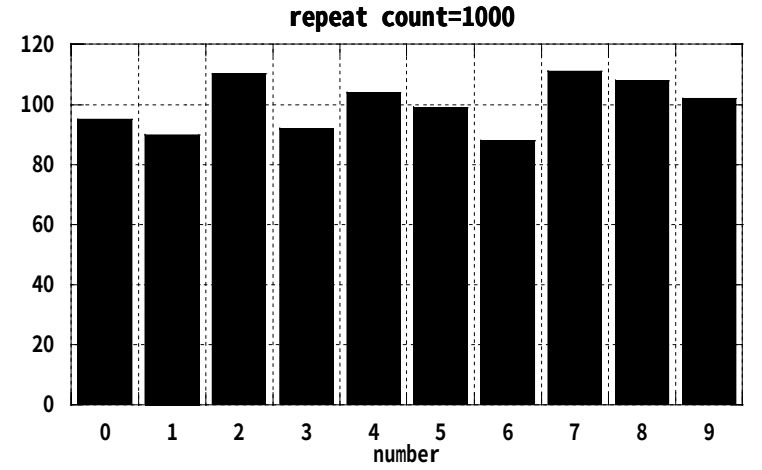


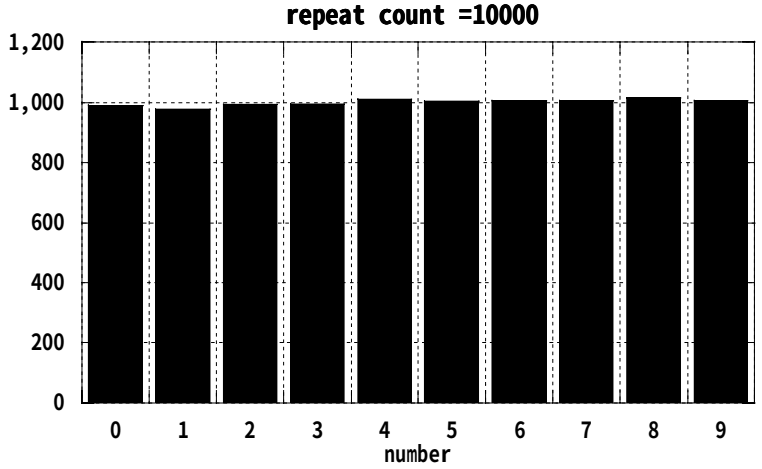


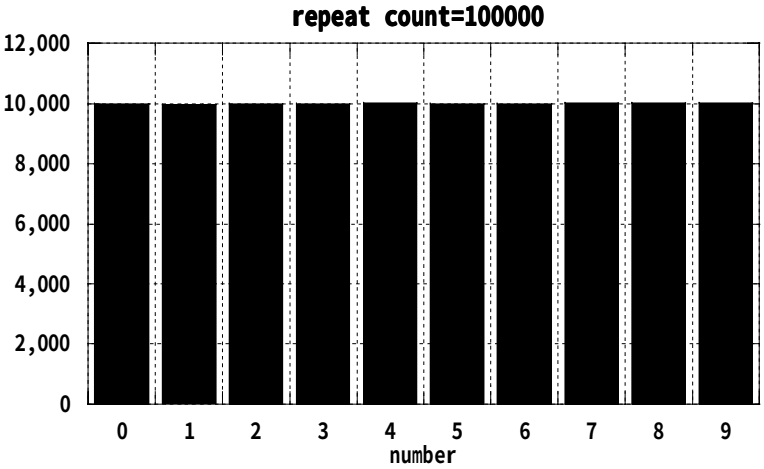

repeat count=70
20
15
10
5
0
0 1 2 3 4 5 6 7 8 9
number

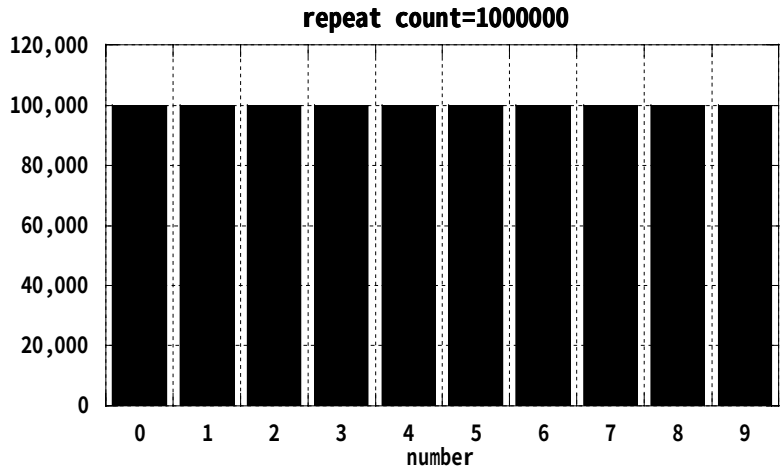

repeat count=90
20
15
10
5
0
0 1 2 3 4 5 6 7 8 9
number


repeat count=100
20
15
10
5
0
0 1 2 3 4 5 6 7 8 9
number

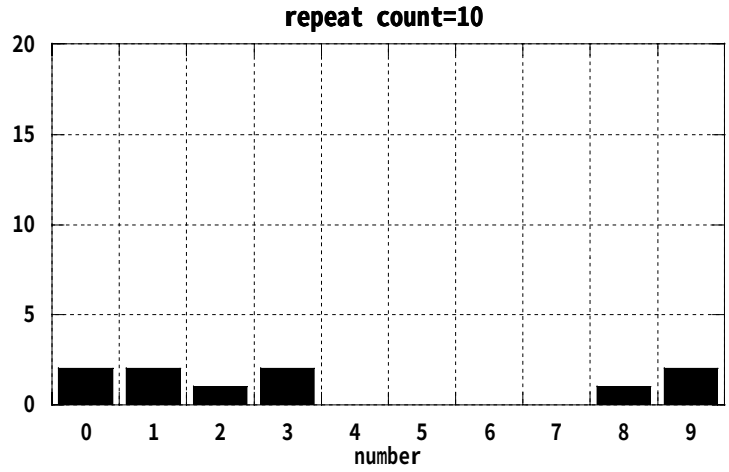

repeat count=1000
120
100
80
60
40
20
0
0 1 2 3 4 5 6 7 8 9
number

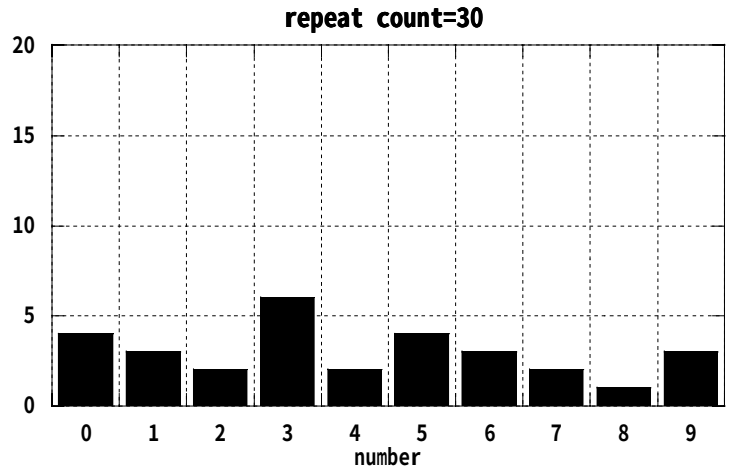

repeat count =10000
1,200
1,000
800
600
400
200
0
0 1 2 3 4 5 6 7 8 9
number

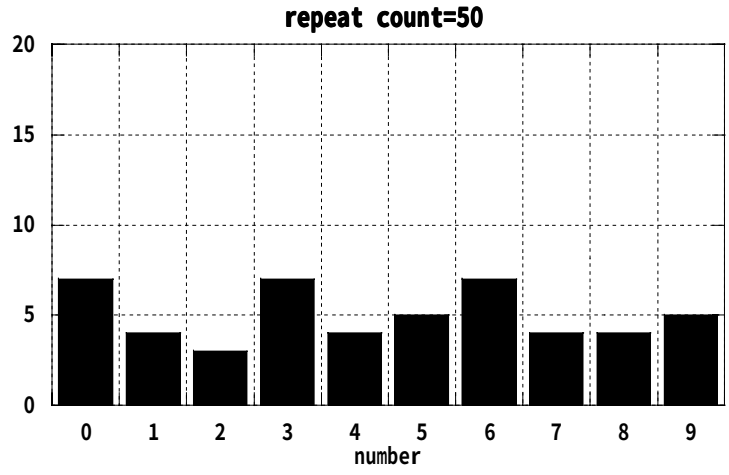

repeat count=100000
12,000
10,000
8,000
6,000
4,000
2,000
0
0 1 2 3 4 5 6 7 8 9
number

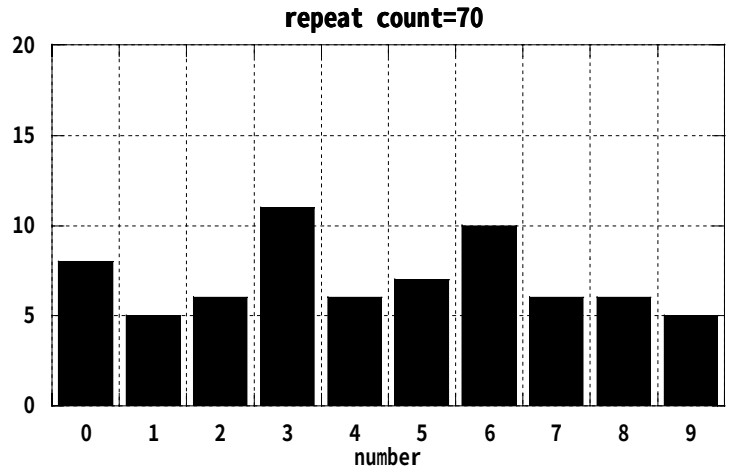

repeat count=1000000
120,000
100,000
80,000
60,000
40,000
20,000
0
0
1
2
3
4
5
6
7
8
9
number

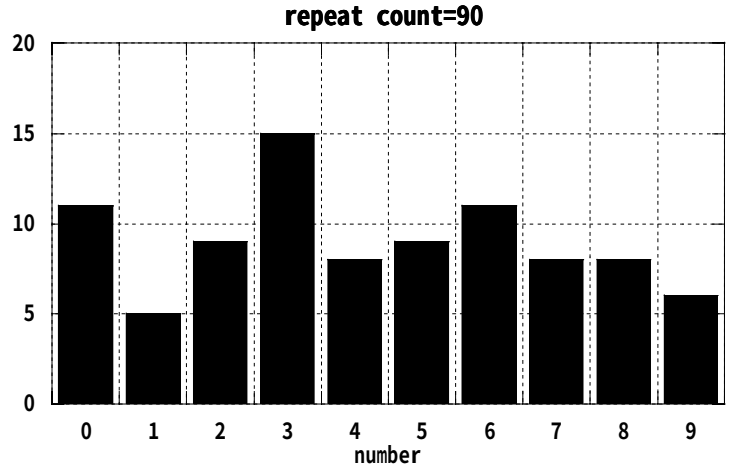
参考資料②　混合合同法による一様乱数発生/各数値の出現回数

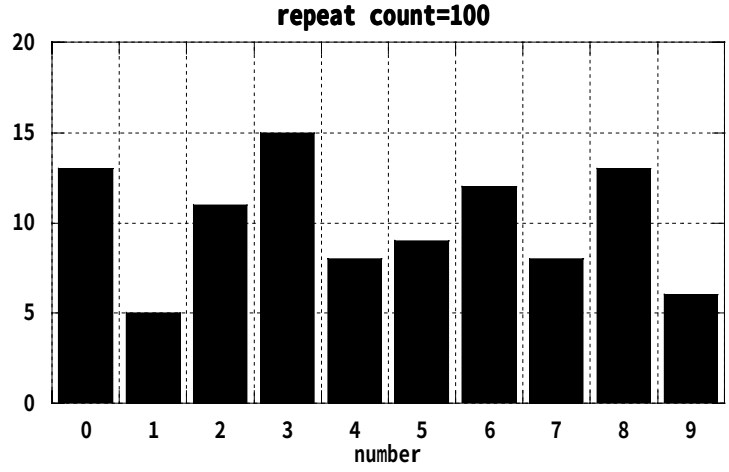


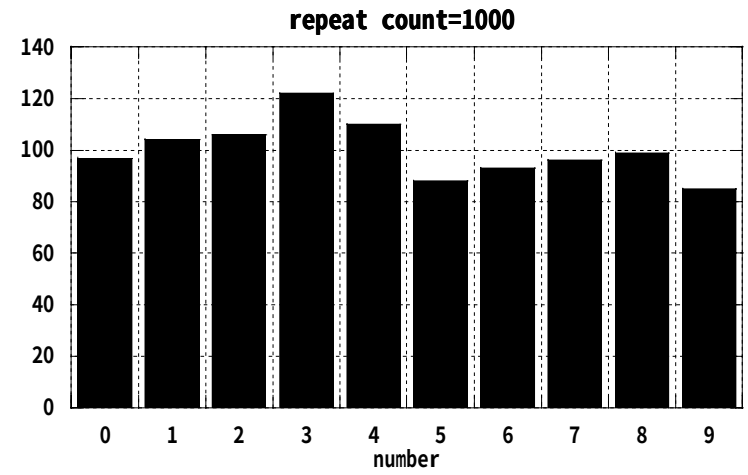


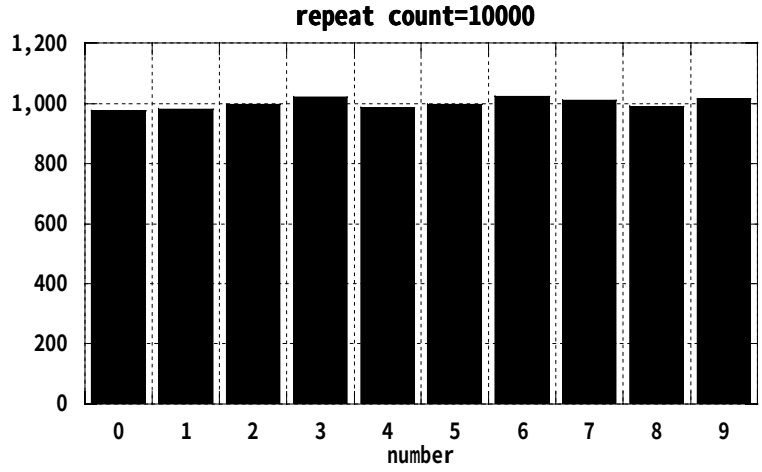


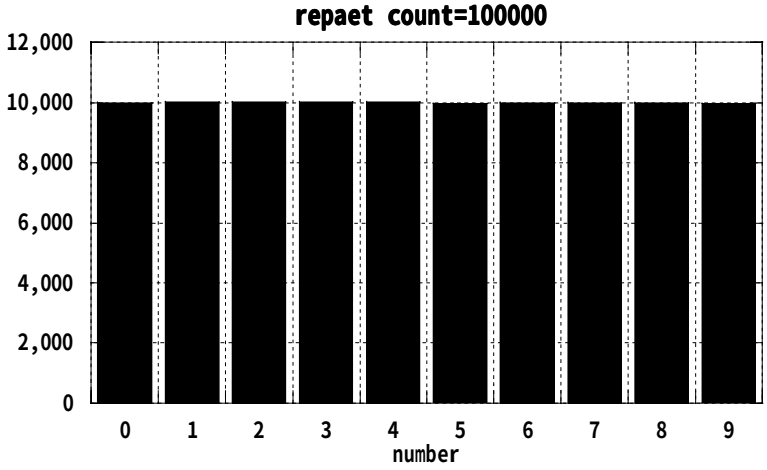

repeat count=70
20
15
10
5
0
0 1 2 3 4 5 6 7 8 9
number

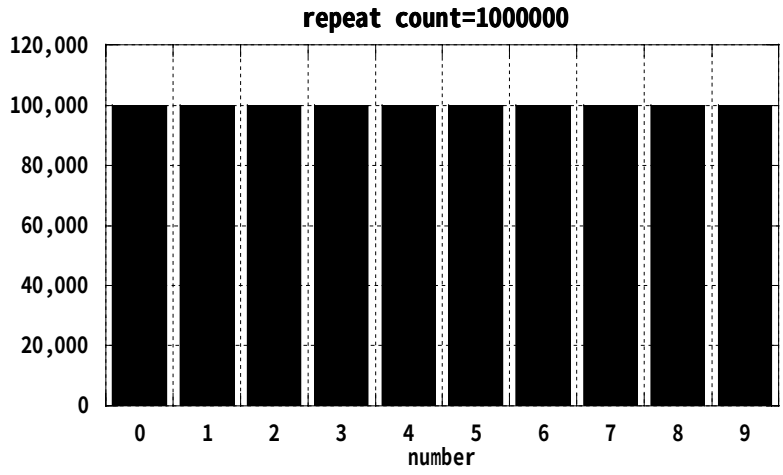

repeat count=90
20
15
10
5
0
0 1 2 3 4 5 6 7 8 9
number


repeat count=100
20
15
10
5
0
0 1 2 3 4 5 6 7 8 9
number


repeat count=1000
140
120
100
80
60
40
20
0
0 1 2 3 4 5 6 7 8 9
number


repeat count=10000
1,200
1,000
800
600
400
200
0
0 1 2 3 4 5 6 7 8 9
number


repaet count=100000
12,000
10,000
8,000
6,000
4,000
2,000
0
0 1 2 3 4 5 6 7 8 9
number


repeat count=1000000
120,000
100,000
80,000
60,000
40,000
20,000
0
0
1
2
3
4
5
6
7
8
9
number

参考資料③　乗算合同法による一様乱数の発生と検定

```
#include <math.h>
#include <stdio.h>
#define RDX 10

main() {
        int rb , rmax=32768;
        int arb[100];
        int i,nmax,number;
        int rnd_b();
        printf(" uniformity random number generation\n");
        printf("-----------------------------------\n");
        printf("multiplication congruence method\n");
        printf("-----------------------------------\n");
        printf("number of data= "); scanf("%d",&nmax);
        for(i=0;i<100;i++) arb[i]=0;
        while(nmax--) {
                        rb=rnd_b();
                        arb[rb/(rmax/RDX)]+=1;
        }
        for(i=0;i<RDX;i++)
                printf("no=\t%d\t\t%d\n",i,arb[i]);
                printf("type any key:");
                test();
}

int rnd_b()     /* multiplication congruence method  */
{
                static long unsigned int x=257, a=2045, p=32768;
                x=((a*x)%p);
                return((int)x);
}

test(){
                float rb,t,
```

```
            rb2,rb3;
        int i,nmax,rnd_b(),rmax=3276.8;
        int cls();
        printf("assay of unformity random number\n");
while(1){
        printf("number of data, or 0 for QUIT:");scanf("%d",&nmax);
        if(nmax<=0) break;
        printf("multiplication congruence method\n");
        rb=rb2=rb3=0.0;
        for(i=0;i<=nmax;i++){
                t=(float)rnd_b()/32768;
                rb +=t;rb2+=t*t;rb3 +=t*t*t;
        }
               rb /=nmax;
               rb2/=nmax;
               rb3/=nmax;
               printf("Moment\n");
               printf("1st \t\t%0.7f\n",rb);
               printf("2nd \t\t%0.7f\n",rb2);
               printf("3rd \t\t%0.7f\n",rb3);
               printf("\n");
        }

}
```

参考資料④　混合合同法による一様乱数の発生と検定

```
#include <math.h>
#include <stdio.h>
#define RDX 10

main() {
        int rc, rmax=32768;
        int arc[100];
        int i,nmax,number;
        int rnd_c();
        printf(" uniformity random number generation\n");
        printf("-----------------------------------\n");
        printf("mixed congruence method\n");
        printf("-----------------------------------\n");
        printf("number of data= "); scanf("%d",&nmax);

        for(i=0;i<100;i++) arc[i]=0;
        while(nmax--) {
                        rc=rnd_c();
                        arc[rc/(rmax/RDX)]+=1;
        }
        for(i=0;i<RDX;i++)
                printf("no=\t%d\t\t%d\n",i,arc[i]);
                printf("type any key:");
                test();
}


int rnd_c()     /*  mixed congruence method  */
{
        static long unsigned int a=2045 ,p=32768 ,x=257 ,b=9378;
                x=(((a*x)+b)%p);
                return((int)x);
}
```

```
test(){
                float rc, t,
                rc2,rc3;
                int i,nmax,rnd_c(),rmax=3276.8;
                int cls();
        printf("assay of unformity random number\n");
        while(1)
        {
         printf("number of data, or 0 for QUIT:");scanf("%d",&nmax);
         if(nmax<=0) break;
         printf("mixed congruence method\n");
         rc=rc2=rc3=0.0;

         for(i=0;i<=nmax;i++){
                  t=(float)rnd_c()/32768;
                  rc +=t;rc2+=t*t;rc3 +=t*t*t;
         }
                rc /=nmax;
                rc2/=nmax;
                rc3/=nmax;
                printf("Moment\n");
                printf("1st \t\t%0.7f\n",rc);
                printf("2nd \t\t%0.7f\n",rc2);
                printf("3rd \t\t%0.7f\n",rc3);
                printf("\n");
        }

}
```

参考資料⑤　混合合同法による一様乱数の検定結果

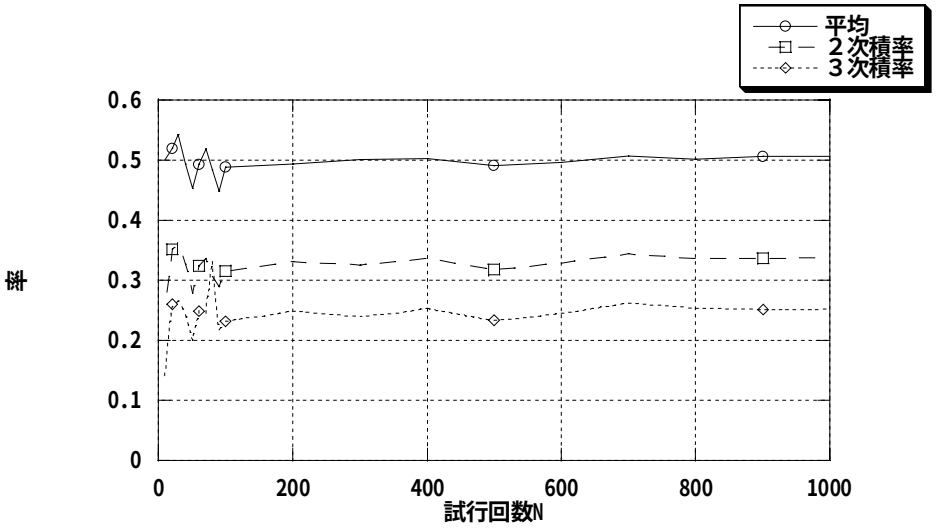


図 5.1　一様乱数の検定

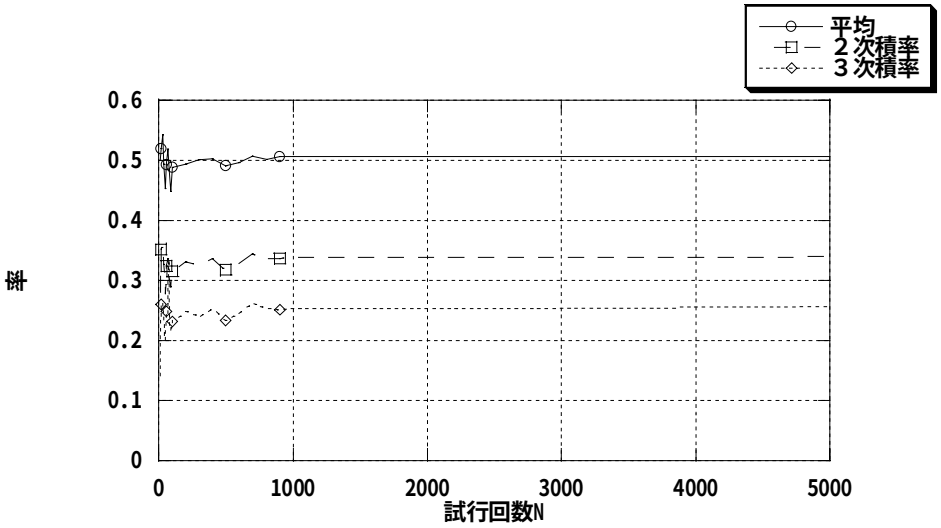


図 5.2　一様乱数の検定

参考資料⑥　混合合同法の加数を変えた検定結果

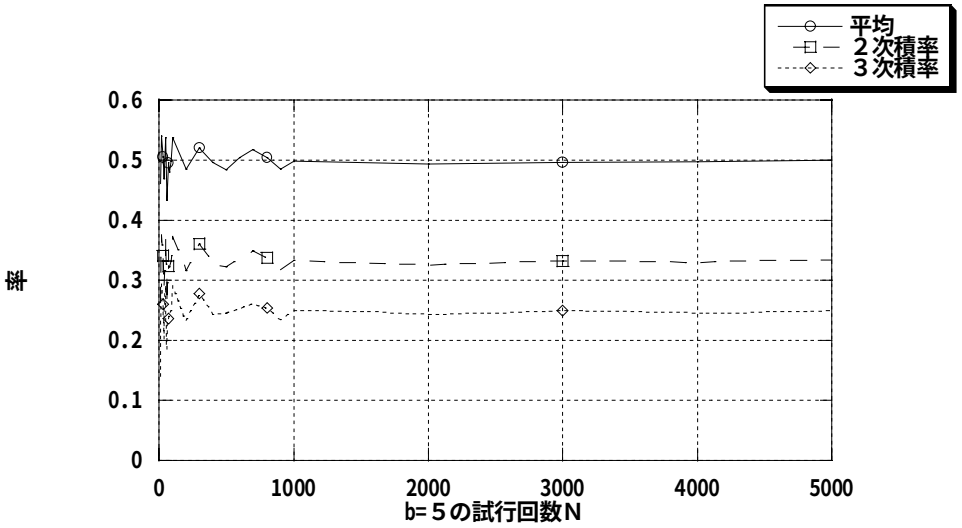


図 6.1　加数＝５の場合の検定

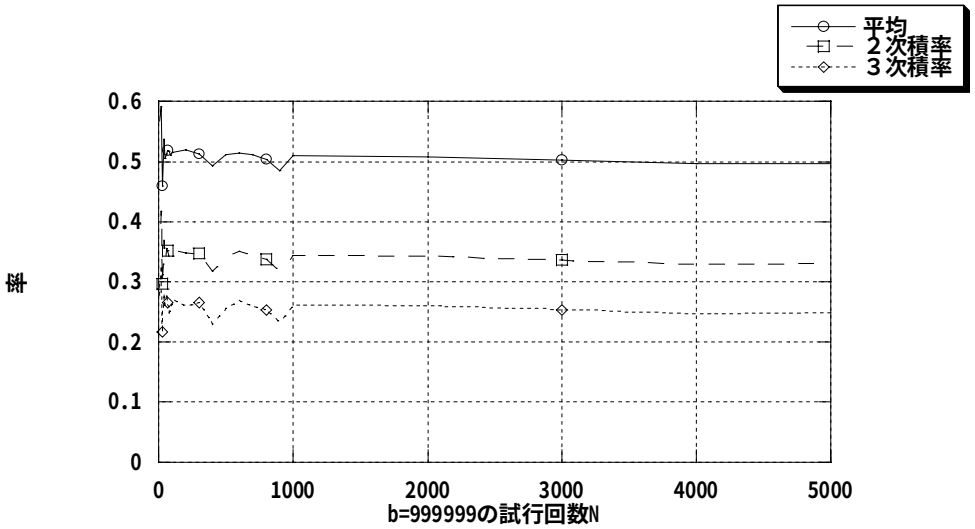


図 6.2　加数＝999999 の場合の検定

参考資料⑦　乗算合同法による一様乱数の発生と検定

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
        int n;      /*number of data */
        float rnd();
        float mat[5000000];   /*variable of matrix*/
        int imat[5000000];
        int i,j;      /*number if ittration*/
        float  y1,y2,y3;      /* y1:average, y2:2nd, y3:3nd */
        float  a1,a2,a3;
        int x0,x1,x2,x3,x4,x5,x6,x7,x8,x9;
        float temp;      /*variable of exchange*/
        y1=0.0; y2=0.0; y3=0.0;
        x0=0;x1=0;x2=0;x3=0;x4=0;x5=0;x6=0;x7=0;x8=0;x9=0;

                printf("\n");
                printf("\n input number of data n= ");
                scanf("%d", &n);

        for(i=0; i<n; i++)
        {
                mat[i]=rnd();
                y1=y1+mat[i];
                y2=y2+mat[i]*mat[i];
                y3=y3+mat[i]*mat[i]*mat[i];
                imat[i]=(int)(10.0*mat[i]);
            if(imat[i]==9) x9=x9+1;
            else if(imat[i]==8) x8=x8+1;
              else if(imat[i]==7) x7=x7+1;
```

```
                else if(imat[i]==6) x6=x6+1;
                  else if(imat[i]==5) x5=x5+1;
                    else if(imat[i]==4) x4=x4+1;
                      else if(imat[i]==3) x3=x3+1;
                        else if(imat[i]==2) x2=x2+1;
                          else if(imat[i]==1) x1=x1+1;
                            else if(imat[i]==0) x0=x0+1;
        }
    printf("-----the number of occurrence for each i-----");
    printf("\n");
                a1=y1/(float)n;
                a2=y2/(float)n;
                a3=y3/(float)n;
        printf("1st average= %f\n",a1);
        printf("2nd average= %f\n",a2);
        printf("3rd average= %f\n",a3);
        printf("\n");
        return(0);

}

/* random function */
float rnd(){
                static long ix=1643;
                long ia=69069;
                float  r;
                ix=ia*ix;
        r=ix*2.32830643658698e-10;
        if(ix<0) r=r+1.0;
        return r;
}
```

参考資料⑧　乗算合同法による一様乱数発生の検定

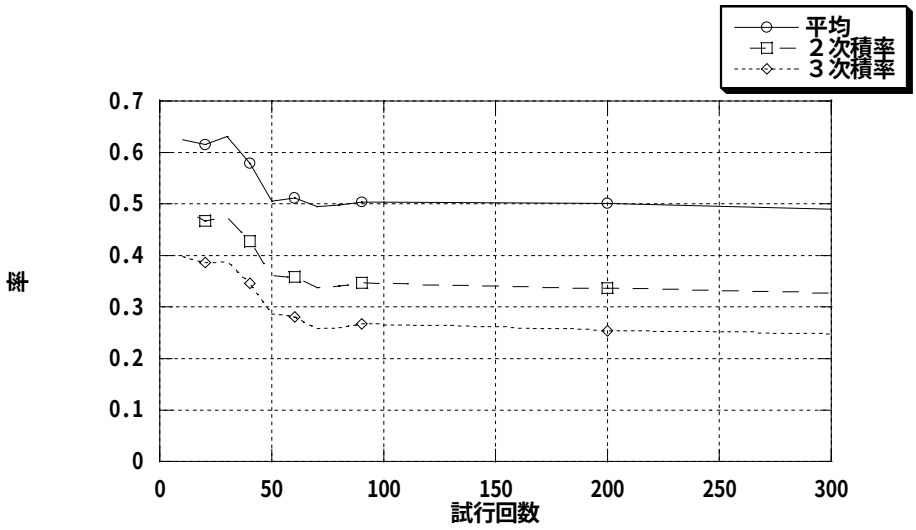


図 8.1　乗算合同法による一様乱数の検定

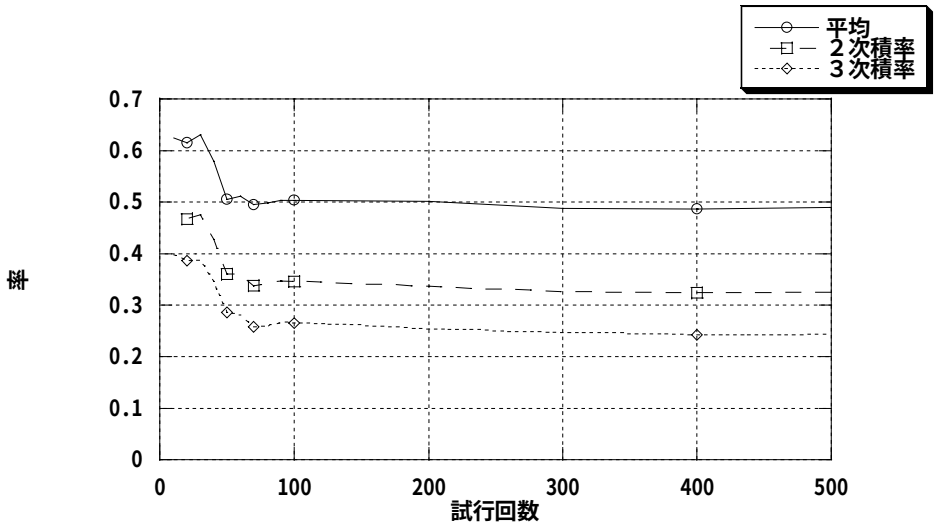


図 8.2　乗算合同法による一様乱数の検定

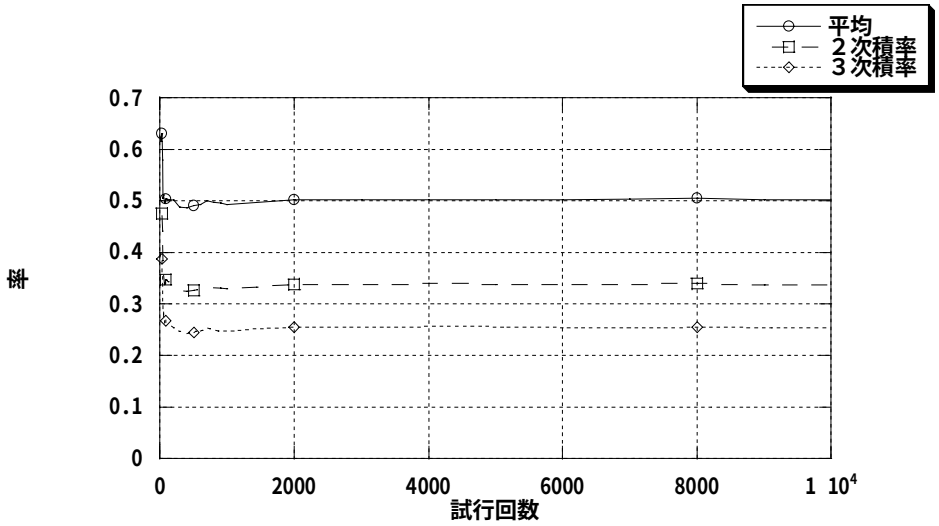


図 8.3　乗算合同法による一様乱数の検定

参考資料⑨　混合合同法を用いた円周率の計算

```
#include <stdio.h>
float rnd();
float fan();
main(){
       int n,i;
       float   b,s;
    printf("the number of scatter sands ");
    scanf(" %d", &n);
    b=0;
    for (i=1;i<=n;i++){
      if  (fan(rnd(),rnd())<=1)  b=b+1.0;
    }
    s=b/n;
    printf("the number of scattered sands    =%d\n",n);
    printf("the number of sands in fan  =%f\n", b);
    printf("area of circle     =%f\n",4.0*s);
}

/* fan function  */
float   fan (x,y)   float x,y; {
   return x*x+y*y;
}

/* rand function */
float   rnd(){
       static   long  ix=1643;
      long   ia=1664525;
      float  r;
   ix=ia*ix+37313;
   r=ix*2.32830643658698e-10;
     if  (ix<0)  r=r+1.0;
     return r;
}
```

実行結果

```
the number of scatter sands 10
the number of scattered sands    =10
the number of sands in fan  =8.000000
area of circle      =3.200000

the number of scatter sands 100
the number of scattered sands    =100
the number of sands in fan  =78.000000
area of circle      =3.120000

the number of scatter sands 1000
the number of scattered sands    =1000
the number of sands in fan  =777.000000
area of circle      =3.108000

the number of scatter sands 10000 (=10^4)
the number of scattered sands    =10000 (=10^4)
the number of sands in fan  =7842.000000
area of circle      =3.136800

the number of scatter sands 100000 (=10^5)
the number of scattered sands    =100000 (=10^5)
the number of sands in fan  =78431.000000
area of circle      =3.137240

the number of scatter sands 1000000 (=10^6)
the number of scattered sands    =1000000 (=10^6)
the number of sands in fan  =785416.000000
area of circle      =3.141664
```
