15　乱数とモンテカルロ法

モンテカルロ法とは、**乱数**を用いて、現象の解析を行ったり方程式を解いたりする手法をいう。始まりは、米国 Los Alamos 研究所において、von Neumann と S. Ulam によって核反応のシミュレーションを行ったことである。乱数の発生は、サイコロを振るように予測の出来ない独立な数列を作ることであるから、有名な国営賭博場のあるモナコ公国の都市名に因んで、モンテカルロ法と名付けられた。この方法は、原子炉における中性子の遮蔽、原子核同士の核反応、宇宙線のカスケード、鎖状分子の変型、流体の動力学など広範囲の数値計算に使われている。

15.1　疑似乱数

真の乱数は、例えば放射性同位元素の崩解のような、不規則に発生する自然現象によってしか生成することが出来ない。これに対して計算機で発生させる乱数は、ある種のアルゴリズムに従い、規則性を出来るだけ排除した数列を発生させることになり、**疑似乱数**と呼び真の乱数と区別する。

モンテカルロ法の計算結果が信頼出来るものであるためには、使用する疑似乱数の一様性や独立性、周期性の無いことが十分に検証されていなければならない。この検証は一見容易に見えるが、10^{10} 以上にもなる標本の独立性を議論することは容易な事では無く、乱数の発生方法やその検定に関して多くの論文や書籍がある程興味深い話題である。しかしながら、物理学における「数値計算」は手法 (あるいは道具) に過ぎず、乱数の発生自体に深入りすることは本意ではない。そこでここでは、出来合いの乱数を使用する事にする。

15.2　一様乱数の発生方法

実用上全ての言語処理系において、最も基本的な一様乱数の発生を支援する機能が備わっており、C 言語では、ANSI 規格で乱数発生関数 `rand()` が規定されている。

FORTRAN 77 の規格では乱数発生用の組み込み関数は定義されていないが、UNIX 上の拡張 FORTRAN 77 処理系の多くでは、以下の関数を標準ライブラリに持っている (但し、使用に際して `external` 宣言が必要な場合もある)。

`rand(`*iflag*`)`	[0,1] の範囲の一様乱数を単精度実数で返す
`drand(`*iflag*`)`	[0,1] の範囲の一様乱数を倍精度実数で返す
`irand(`*iflag*`)`	[0,2147483647] の範囲の一様整数乱数を返す

ここで整数引数 *iflag* の意味は次の通り。

0	疑似乱数列の次の乱数を返す
1	疑似乱数列の生成を再開し、最初の乱数を返す
その他	*iflag* を「種」として疑似乱数列を生成し、最初の乱数を返す

教育用計算機システム上の FORTRAN 77 処理系でも乱数発生用の組み込み関数を持っており使用可能である。但し、関数を `external` 宣言しておく必要がある (`external` について知りたいものは、FORTRAN の文法書や解説書を参考にして欲しい)。以下のサンプルプログラムでは、単精度乱数を 10 個発生させて画面に表示させる。プログラムは、

```
/home/teacher/z6wt01in/SAMPLE/rand_test.f
```

として置いてある。

サンプルプログラム (rand_test.f)

```
      program rand_test
      implicit none
c local:
      integer   iflag/0/  ! 関数 rand の引数
      integer   i         ! loop 用変数
c function:
      real      rand
      external  rand      ! 教育用システムでは必要
c begin:
      do i=1,10
         write(*,'(I3,F9.5)') i,rand(iflag)
      end do
      stop
      end
```

15.3 与えられた分布関数に従う乱数の発生方法

モンテカルロ法では、[0,1] の範囲に発生させた擬似乱数 x をもとに、与えられた分布関数 $g(y)$ に従う乱数 y を発生させる必要がある。その代表的な方法をまとめておく。

15.3.1 逆変換法

一様乱数の確率密度を $f(x)$ とおくと、一般に

$$f(x)dx = g(y)dy \tag{15.1}$$

が成立する。ここで、$f(x) = 1(\int_0^1 f(x)dx = 1$ と $f(x)$ が x によらない事から自明) であるので、両辺積分して

$$x = \int^y g(y')dy' \equiv G(y) \tag{15.2}$$

が得られる (積分の下限は y の定義域に従う)。従って、求める分布関数 $g(y)$ に従う乱数 y は、一様疑似乱数で発生させた x を、$G(y)$ の逆関数 $G^{-1}(y)$ を使って

$$y = G^{-1}(x) \tag{15.3}$$

と変換すれば得ることが出来る。

例 1：指数関数に従う乱数

指数関数 $g(y) = \lambda \exp(-\lambda y)$ に従う乱数は、モンテカルロ法で粒子の崩解時間分布等を得るのに重要である。$g(y)$ が確率密度関数であることは、定義域 $[0, \infty]$ での積分が 1 であることから確かめられる。ここで、

$$G(y) = \int_0^y \lambda \exp(-\lambda y')dy' = 1 - \exp(-\lambda y) \tag{15.4}$$

であるので、$G(y) = x$ を y について解いて

$$y = -\frac{1}{\lambda}\ln(1-x) \tag{15.5}$$

により、一様疑似乱数 x を変換すれば良いことが分かる。

例 2：正規分布に従う乱数

逆変換法は 2 次元以上にも一般化できる。

$$f(x_1, x_2, \ldots)dx_1dx_2\cdots = g(y_1, y_2, \ldots)dy_1dy_2\ldots \tag{15.6}$$

$x_1, x_2, \ldots$ が区間 $[0, 1]$ で定義された一様乱数の場合には $f(x_1, x_2, \ldots) = 1$ なので、式 (15.6) は

$$\frac{\partial(x_1, x_2, \ldots)}{\partial(y_1, y_2, \ldots)} = g(y_1, y_2, \ldots) \tag{15.7}$$

という連立微分方程式になる、これを解いて、$y_1, y_2, \ldots$ を $x_1, x_2, \ldots$ の関数として表したものが求める変換式である。

ここでは重要な例として、平均 0、分散 1 の標準正規分布

$$g(y) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{y^2}{2}\right) \tag{15.8}$$

の場合を考える。連立微分方程式を解くのは厄介なので、天下り的ではあるが

$$\begin{aligned} y_1 &= \sqrt{-2\ln x_1}\cos(2\pi x_2) \\ y_2 &= \sqrt{-2\ln x_1}\sin(2\pi x_2) \end{aligned} \tag{15.9}$$

とおく。これは

$$\begin{aligned} x_1 &= \exp\left(-\frac{y_1^2 + y_2^2}{2}\right) \\ x_2 &= \frac{1}{2\pi}\arctan\frac{y_2}{y_1} \end{aligned} \tag{15.10}$$

と書き換えられるので、Jacobian を計算すると、

$$\frac{\partial(x_1, x_2)}{\partial(y_1, y_2)} = \begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} \end{vmatrix} = -\left[\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{y_1^2}{2}\right)\right]\left[\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{y_2^2}{2}\right)\right] \tag{15.11}$$

となり、式 (15.7) が満たされている事が分かる。

もとは 1 変数問題だったものを、2 変数にして格好良く解いた訳だが、式 (15.9) の変換方法は**Box-Muller** 法として知られているものである。Box-Muller 法のサンプルプログラムを以下に載せる。プログラムは、

```
/home/teacher/z6wt01in/SAMPLE/gran.f
```

として置いてある。

> 変数 `odd,fac,x2` は `save` 文により再利用出来る (再度呼ばれたときに前回の値が保持されている) ようにしている。

サンプルプログラム (gran.f)

```
      real function gran()
      implicit none
c const:
      real     pi
      parameter(pi=3.14159265)
c local:
      integer  odd/0/
      real     x1,x2,fac
c function:
      real     rand
      external rand
c save:
      save     odd,fac,x2
c begin:
      odd=1-odd                  ! 奇数 <-> 偶数
      if (odd.ne.0) then         ! 奇数回目の呼び出しなら
         x1=rand(0)              ! [0,1] の一様乱数を生成
         x2=rand(0)
         fac=sqrt(-2.0*log(x1))
         gran=fac*cos(2.0*pi*x2)
      else                       ! 偶数回目の呼び出しなら
         gran=fac*sin(2.0*pi*x2)
      endif
      return
      end
```

15.3.2 von Neumann の棄却法

逆変換法で必要となる G^{-1} は常に求められるとは限らない。力ずくだが適用範囲の広い方法として、分布関数の定義域 $[x_{\min}, x_{\max}]$ と値域 $[y_{\min}, y_{\max}]$ が張る長方形に一様に降り注ぐ乱数を発生させ、目的とする分布関数の「内側」に落ちたものを採用し、「外側」に落ちたものを棄却するという方法が考えられる。この方法を**von Neumann の棄却法**という。具体的には、

1. 範囲 $[0,1]$ の一様疑似乱数 x を発生させる (組み込み関数 `rand` 等を用いればよい)
2. x を使って、範囲 $[x_{\min}, x_{\max}]$ の一様疑似乱数 x' を

$$x' = x_{\min} + x(x_{\max} - x_{\min}) \tag{15.12}$$

   により生成する。

3. 範囲 $[0,1]$ の一様疑似乱数 y を発生させる (組み込み関数 `rand` 等を用いればよい)

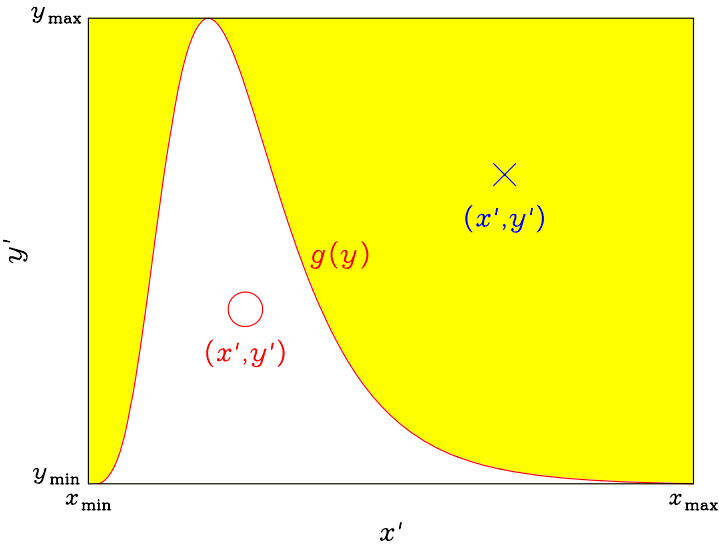

図 15.1: von Neumann の棄却法による乱数発生。$[x_{\min}, x_{\max}]$ と $[y_{\min}, y_{\max}]$ に一様分布する乱数の内、関数の内側 (◯ 印) に入るものを採用し、外側の (× 印) ものを棄却する。

4. y を使って、範囲 $[y_{\min}, y_{\max}]$ の一様疑似乱数 y' を

$$y' = y_{\min} + y(y_{\max} - y_{\min}) \tag{15.13}$$

により生成する。

5. (a) $y' > g(x')$ ならば、x' を棄てて 1. からやり直す (図 15.1 中 × 印)
   (b) $y' \leq g(x')$ ならば、x' が求める乱数 (図 15.1 中 ◯ 印)

とすればよい。

実際に使うときには、分布関数 $g(y)$ によっては捨てる乱数を少なくするような工夫 (例えば乱数の発生域を長方形ではなく階段型にする等)、あるいは再利用する方法 (例えば**Metropolis 法**等) が必要である。興味のあるものは、数値計算の参考書を参考にして欲しい。

例：棄却法による乱数発生

分布関数 $g(y) = \sqrt{y^2-1}y(y_0-y)^2$ に従う乱数[1]を発生させるサンプルプログラムを以下に載せる。簡単のため、長方形は範囲が $[1, y_0]$、高さ `YMAX` を 1 にとる。プログラムは、

`/home/teacher/z6wt01in/SAMPLE/generate.f`

として置いてある。

> `YMAX` の値は、分布関数 $g(y)$ を全て覆うように適切な値を選ばなければならない。具体的には `y0` の値によって適切に選ばなければならず、2.25 を越える場合は `YMAX` を 1 より大きな値にする必要がある。

[1] この関数型は、Fermi のベータ崩解の理論で電子のエネルギー分布を与える関数である。

サンプルプログラム (generate.f)

```
      subroutine generate(y0,y)
      implicit none
c const:
      real     YMAX
      parameter(YMAX=1.0)
c input/output:
      real     y0,y
c local:
      real     gy
      logical  success
c function:
      real     rand
      external rand
c begin:
      success=.false.
      do while (.not.success)
         y =(y0-1.0)*rand(0)+1.0       ! y0 は [1,y0] の一様乱数
         gy=sqrt(y**2-1.0)*y*(y0-y)**2 ! g(y) の計算
         success=YMAX*rand(0).le.gy
      end do
      return
      end
```

課題

半径 1 の球の表面上に一様な密度で点を分布させたい。点の緯度、経度の組 (極座標系の (θ, ϕ)) を一様乱数を用いて計算するプログラムを書け。この問題は、粒子を 3 次元空間に等方的に発生させる場合等、多くの場合に重要である。以下に、初心者によく見られる誤りの例を示す。この誤ったプログラムで発生させた (θ, ϕ) を用いてデカルト座標

$$x = \sin\theta\cos\phi, \quad y = \sin\theta\sin\phi, \quad z = \cos\theta \tag{15.14}$$

を計算して gnuplot で表示させると図 15.2 のようになり、明かに非等方 (密度の高い部分は $\theta = 0, \pi$ に対応) である事が分かる。

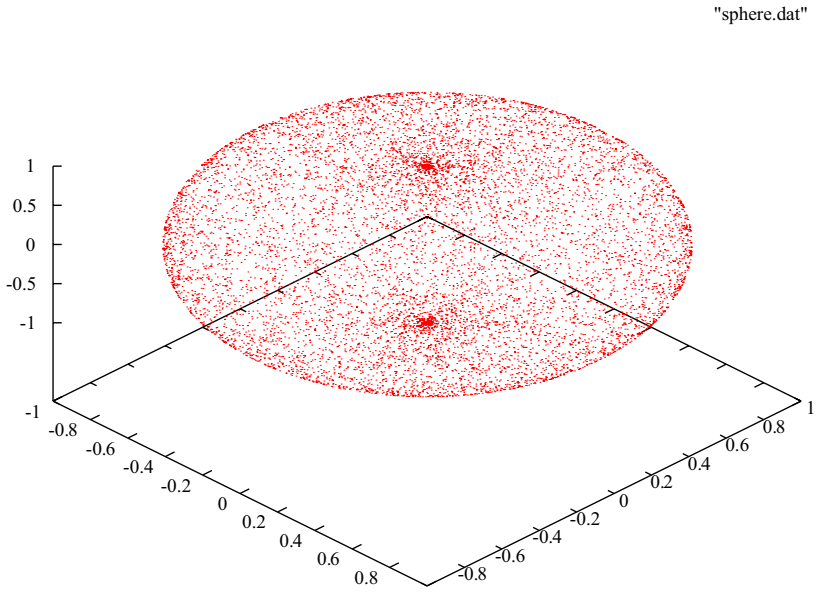

図 15.2: 誤ったプログラムで半径 1 の球の表面上に一様に点を発生させた例。$\theta = 0, \pi$ に対応する部分の密度が高く、一様ではないことが分かる。

誤ったサンプルプログラム (sphere.f)

```
      program sphere
      implicit none
c**********************************
c !!!!! 誤りの例 !!!!!
c 球面上の乱数 -緯度、経度の発生-
c**********************************
c const:
      integer  NUMBER          ! 発生させる乱数の数
      parameter(NUMBER=10000)
      real     PI
      parameter(PI=3.14159265)
c local:
      real     theta,phi       ! 緯度と経度
      integer  i               ! loop 用
c function:
      real     rand
      external rand
c begin:
      do i=1,NUMBER
         theta=PI*rand(0)      ! 誤り!まねするな!
         phi  =2.0*PI*rand(0)
         write(*,'(2F10.5)') theta,phi
```

```
      end do
      stop
      end
```

15.4 モンテカルロ積分

モンテカルロ法は、本来偶然現象ではない定積分の計算や、積分方程式や偏微分方程式を解く問題にも応用される。例えば、拡散方程式の解が、粒子の進路を数値的に多数回繰り返し追跡することにより求められたりする。ここでは身近な応用例として、モンテカルロ法による数値積分について考える。

簡単のため、1 次元積分

$$I = \int_a^b f(x)dx \tag{15.15}$$

をモンテカルロ法により評価することを考える。台形公式では、積分区間 $[a, b]$ を等間隔に分割して積分を評価した。モンテカルロ法では、積分区間 $[a, b]$ に一様に発生する乱数 x_i を N 個発生させ、

$$I = \int_a^b f(x)dx \approx \frac{(b-a)}{N} \sum_{i=1}^{N} f(x_i) \tag{15.16}$$

により積分を評価する。中心極限定理により、モンテカルロ積分は N を大きくすると、$\mathcal{O}(N^{-1/2})$ に従って精度が向上することが期待される。

式 (15.16) をそのままプログラムにしたのが以下に示すサンプルプログラムである。プログラムは、

```
/home/teacher/z6wt01in/SAMPLE/mc_int.f
```

として置いてある。

サンプルプログラム (mc_int.f)

```
      program mc_int
      implicit none
c const:
      real     A,B             ! 積分区間
      parameter(A=0.0,B=1.0)
      real     N               ! 乱数の数
      parameter(N=100000)
c local:
      integer  i               ! ループ用
      real     integral/0.0/,x
c function:
      real     f               ! 被積分関数
      real     rand
      external rand
```

```
c begin:
      do i=1,N
         x=(B-A)*rand(0)+A    ! [A,B] で一様乱数
         integral=integral+f(x)
      end do
      integral=integral*(B-A)/N
      write(*,'(A,E15.7)') 'Integral = ',integral
      stop
      end
c
      real function f(x)      ! 被積分関数
      implicit none
c input:
      real     x
c begin:
c ここに適当な被積分関数を入れる
c f=関数値という代入文で主プログラムに値を戻す
c 例えば
      f=exp(x)                ! 厳密解は e-1=1.71828...
      return
      end
```

モンテカルロ積分で注意すべきことは、

1. 関数を N 回計算させたとき、モンテカルロ法では積分の次元数 d によらず精度が $\mathcal{O}(N^{-1/2})$ で向上するが、台形則では (1 次元あたりの刻み数が $N^{1/d}$、台形則の誤差が刻み幅の 2 乗オーダーであるため) 精度は $\mathcal{O}(N^{-2/d})$ となる。したがって**4 次元以上の積分になれば、モンテカルロ法が有利**になる。

2. モンテカルロ積分は、被積分関数 $f(x)$ が一様であれば収束が早い。他方、$f(x)$ が $\delta(x)$ に近いような特異性を持っている場合、x_i がたまたまその付近をサンプルしなかったとしたら、積分値は真値からかけ離れてしまう。このような場合は、$f(x)$ を適当な重み関数で割って被積分関数を一様化するか、重要な領域に重みの付いた乱数を用いてモンテカルロ法を行うなどの工夫が必要である。

3. 多次元の積分に台形則などを適用しようとすると、積分範囲の解析的な表現が簡単に書けないという障害に出くわす。モンテカルロ法は、乱数で指定した点が、領域の内側にあるか外側にあるかさえ判定すれば積分できるため、複雑な積分境界の関数を積分するのに適している。

課題

2,4,5,10 次元空間の半径 1 の (超) 球の体積をモンテカルロ法により求めよ。d 次元球のときには、$[0,1]$ に一様に分布する乱数を d 個用意して、それらの 2 乗和が 1 より小さくなる割合を求めればよい。得られた体積を解析的に求めた結果と比較せよ。余裕があれば、モンテカルロ法以外の数値積分法でも計算して、計算精度や計算時間を比較せよ。

計算・メモ用余白