付録 G　乱　数

ATM では、いくつかの箇所で擬似乱数を扱っている。例えば、ATM 本体に含まれる拡散過程（第 2 章）や、ESP を用いて初期値（第 3 章）を作成する際のトレーサーの配置や粒径分布などである。Fortran には擬似乱数を生成する組込みサブルーチンとして `random_number` が用意されているものの高速で高品質な擬似乱数を生成し、また、MPI 並列化（付録 H.1）における各プロセスごとに異なる擬似乱数を扱うためには、組込みサブルーチンとは別の擬似乱数生成器が必要である[1]。そこで ATM では、一様乱数は現在 xorshift（Press *et al.*, 1996; Marsaglia, 2003）と呼ばれる擬似乱数生成器で作成し、正規乱数は xorshift で作成された一様乱数を使って、Box-Muller 法（Box and Muller, 1958）を用いて作成している。正規乱数については、以前のモデル（GATM, RATM）で使われていた方法であるが、xorshift は今回のモデル更新の際に新たに採用した擬似乱数生成器であり、比較的最近の手法であるため、ここで簡単に特徴を述べる。

G.1　一様乱数（Xorshift）

Marsaglia (2003) によって開発された擬似乱数生成器である xorshift のアルゴリズムは、排他的論理和とビットシフトのみから構成され処理速度が速く[2]、高速な計算が求められる ATM に適していると考え採用した。また、xorshift で作られる擬似乱数は乱数としての精度（規則性のなさなど真の乱数が満たすべき性質の再現性）も高く、擬似乱数を多用する ATM には適している。Marsaglia (2003) によると xorshift は、周期に関して 3 つのオプションが存在し、それぞれの周期は、$2^{32}-1$、$2^{64}-1$、$2^{128}-1$ である。ATM では、最も周期の短い $2^{32}-1$ を採用しているが、それでも 43 億程度である。この周期は、数 10 万から数 100 万程度の粒子数を計算するには十分な長さであるが、将来的により周期の長いオプションに変更する可能性はある[3]。

Xorshift は、乱数シード（$s = s_0$）を入力値として与え、そのシードに対応した乱数 x_0 を出力し、同時にシードが $s = s_1$ に更新される仕様になっている。新たに乱数を取得したいときは、更新されたシード $s = s_1$ を xorshift に入力すると、新たな乱数 x_1 が出力され、同時にシードが $s = s_2$ に更新される。更新されたシード $s = s_2$ は次の擬似乱数 x_2 を取得するために使われる。このように、xorshift では、初期のシード（0 以外の整数）を決定し、そのシード s を更新しながら対応する新たな乱数を取得する仕組みとなっている[4]。

Figure G.1 に実際に ATM のコード（抜粋）を示す[5]。ATM で行う計算では、トレーサーの数だけの乱数が必要となる状況が多いため、`n_max` 個の 1 次元配列の擬似乱数（0 以上 1 以下）を返り値とするサブルーチンを実装している[6]。また、`seed`（乱数シード）が入力値であり、作成された擬似乱数の数だけ更新されるので、`n_max` 回更新された `seed` が返り値になる。なお、Figure G.1 の 29 から 33 行目までは本来ならば不要である処理であるが、どのような状況になっても異常終了させないための例外処理であり、0 未満または 1 より大きい擬似乱数が生成されてしまった場合は 0.5 を返すようにしている（実際には、この `if` 文は通過しない）。

---

[1] プロセスごとに異なる擬似乱数を使うためには、`seed` を陽に管理する必要があり `random_number` は使えない。

[2] 気象研究所の現有サーバ「リモートセンシングによる噴煙状態解析装置」（Intel® Xeon® Gold 6138 (3.7 GHz) 搭載）で実行したところ、1 億個の乱数を 10 回作成するのにかかる時間は 4 秒程度であった。

[3] 一回の計算の中で同じ乱数が使われても直ちに問題になるわけではない。例えば、初期値の粒径分布に使われる乱数と、予報の途中で拡散計算に使われる乱数に一部重複があっても問題にはならないだろう。

[4] プロセスごとに初期の乱数シードを異なる値にする必要があることに注意。

[5] コード内の整数・実数の精度については、`sp`=4, `rp`=8 である（Table C.4）。また装置番号は、非 MPI 環境では、`io_mpi_log`=6 である。

[6] つまり、このサブルーチンは `n_max` はトレーサー数（`n_tracer`）で呼ばれることがほとんどである。

```
! ============================================================
!   Based on Marsaglia (2003) "Xorshift RNGs"
! ============================================================
subroutine xorshift32_rng( harvest, n_max, seed )

  real   (rp), intent(out)  :: harvest(1:n_max)
  integer(sp), intent(in)   ::           n_max
  integer(sp), intent(inout):: seed

  integer(sp)           :: nn
  integer(sp)           :: y                     !! 32bit Integer
  integer(sp), parameter:: n_32bit = 2147483647 !! 2**32/2-1
  real   (rp), parameter:: inv     = 1._rp / ( 2._rp * real(n_32bit, rp) )

  ! -----------------------------------------------
  !   Initialize
  ! -----------------------------------------------
  y = seed
  ! -----------------------------------------------
  !   Xorshift algorithm for seed y
  ! -----------------------------------------------
  do nn = 1, n_max
    y = ieor(y, ishft(y, +13))
    y = ieor(y, ishft(y, -17))
    y = ieor(y, ishft(y, + 5))

    harvest(nn) = inv * real(y, rp) + 0.5_rp

    if ( harvest(nn) < 0._rp .or. 1._rp < harvest(nn) ) then
      write(io_mpi_log, *) "WARNING: RANDOM NUMBER GENERATION"
      write(io_mpi_log, *) "SEED, N, Y, HARVEST = ", seed, nn, y, harvest(nn)
      harvest(nn) = 0.5_rp
    end if
  end do
  ! -----------------------------------------------
  !   Update seed for next call
  ! -----------------------------------------------
  seed = y

  return
end subroutine xorshift32_rng
```

Figure G.1　Xorshift RNGs implemented in the JMA-ATM