Rの乱数について

Ei-ji Nakama

Kanazawa.R#3

おしながき

自己紹介

プロンプトより

```
> grep( "Nakama",
+       readLines(file.path(R.home("doc"), "THANKS")),
+       value=TRUE)
[1] "Magnusson, John Maindonald, David Meyer, Ei-ji Nakama, Jens"
```

- R歴 ... たぶん27〜28年?

Rの乱数は32bit整数

RNGのヘルプより

```
> help(RNG)

 Do not rely on randomness of low-order bits from RNGs.
 Most of the supplied uniform generators return 32-bit
 integer values that are converted to doubles, so they
 take at most 2^32 distinct values and long runs will
 return duplicated values
```

RNGの下位ビットのランダム性に依存しないでください．
提供されている一様乱数生成器のほとんどは，32ビットの整数値を返し，それをdouble型に変換します．
そのため，最大で2^32個の異なる値しか取りません．
また，長い実行では重複した値が返されます．

RNGの下位ビット

RNGの下位ビット

```
> mydisp<-function(x) cbind ( dec = sprintf('%.22g',x),
+                             hex = sprintf('%.13A',x) )
> set.seed(0)
> mydisp( runif(5) )
     dec                          hex
[1,] "0.8966972001362591981888" "0X1.CB1BE53A00000P-1"
[2,] "0.2655086631420999765396" "0X1.0FE180C400000P-2"
[3,] "0.3721238996367901563644" "0X1.7D0E0C2C00000P-2"
[4,] "0.5728533633518964052 2"  "0X1.254D093A00000P-1"
[5,] "0.9082077899947762489319" "0X1.D1009C8800000P-1"
> mydisp( 1 + .Machine$double.eps ) # 参考
     dec                          hex
[1,] "1.0000000000000022045" "0X1.0000000000001P+0"
```

整数を変換

32ビットの整数値をdouble型に変換する. $[0, 1)$を表す

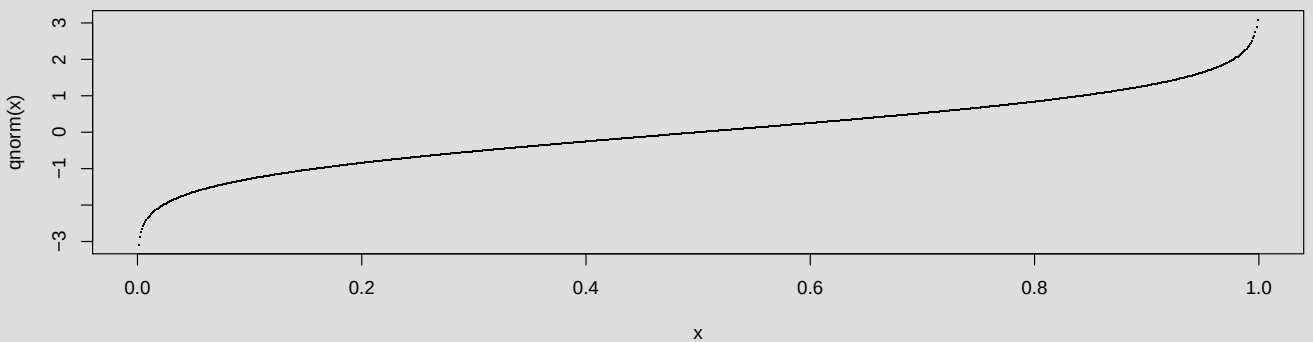
変換されたMTの最小値と最大値

```
> set.seed( -1891743248 ) # 最初の乱数が0のseed
> mydisp( Umin <- runif( 1 ) )
     dec                                  hex
[1,] "1.164153218540398429159e-10" "0X1.00000000FFFFFP-33"
> Umin * 2^32                  # 0の変わりに0.5
[1] 0.5
> set.seed( 1287433146 )  # 最初の乱数が最大値
> mydisp( Umax <- runif( 1 ) )
     dec                                hex
[1,] "0.99999999976716935634613" "0X1.FFFFFFFE00000P-1"
> Umax * 2^32                  # 2^32 - 1
[1] 4294967295
```

0は屡々都合が悪い

例えば確率密度関数 qnorm

```
> x <- seq( from = 0, to = 1, length.out = 1001 )
> plot( x, qnorm( x ) , pch = '.' )
> fivenum( qnorm( x ) )
[1]       -Inf -0.6744898  0.0000000  0.6744898        Inf
```



rnormの場合

rnorm

```
> set.seed( -1891743248 ) # 最初の乱数が0のseed
> mydisp( rnorm( 1 ) )
     dec                                hex
[1,] "-5.898834592211020577679" "-0X1.7986818683D44P+2"
> set.seed( -1891743248 ) # 最初の乱数が0のseed
> num <- ( as.integer( runif(1)*2^27)
+           + runif(1) ) / 2^27
> mydisp( qnorm( num ))
     dec                                hex
[1,] "-5.898834592211020577679" "-0X1.7986818683D44P+2"
```

乱数(疑似乱数)とは

The Art of Computer Programming - より

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin. - John von Neumann(1951)

訳) 四則演算によって乱数を作り出そうと試みる者は, 言うまでもなく, 神に背こうとしているのである. [1]

[1] 準数値算法/乱数 - 渋谷政昭訳

乱数の生成

統計に必要な乱数を考える. “0と1の間で一様に分布する実数U_n”を考えれば, 計算機は有限精度の実数しか表せないから, 実際には 0とある整数mとの間の整数X_nを生成する.

$$U_n = \frac{X_n}{m}$$

そうするとU_nは0から1の間に収まる. 一般にmは$X_n < m$となる整数であり, 型の変更を伴う場合, 乱数の性質を保持するならば

$$X_n = U_n \times m$$

を満たす必要がある

倍精度実数で表現可能な値

倍精度実数型で表現可能な値を見てみる

```
> n <- seq( from = 0, to = 8, by = 1) ^ 2
> base <- 2^-n
> mydisp( base +.Machine$double.eps * base )
      dec                                hex
 [1,] "1.00000000000000022045"     "0X1.0000000000001P+0"
 [2,] "0.50000000000000011102 23"  "0X1.0000000000001P-1"
 [3,] "0.06250000000000001387779"  "0X1.0000000000001P-4"
 [4,] "0.00195312500000000433681"  "0X1.0000000000001P-9"
 [5,] "1.52587890625000033881 3e-05" "0X1.0000000000001P-16"
 [6,] "2.98023223876953191174 4e-08" "0X1.0000000000001P-25"
 [7,] "1.45519152283668550378 1e-11" "0X1.0000000000001P-36"
 [8,] "1.77635683940025085910 8e-15" "0X1.0000000000001P-49"
 [9,] "5.42101086242752337374 3e-20" "0X1.0000000000001P-64"
> gmp::as.bigq(base) # 有理数に変換
Big Rational ('bigq') object of length 9:
[1] 1                    1/2                  1/16
[4] 1/512                1/65536              1/33554432
[7] 1/68719476736        1/562949953421312    1/18446744073709551616
```

指数部の値によって表現可能な範囲が変わる

乱数の検定

そもそも疑似乱数を乱数とみなして良いかというのは置いといて

参考：乱数の検定

- D.E.Knuth - The Art of Computer Programming 2
- NIST - SP 800-22

  https://csrc.nist.gov/Projects/random-bit-generation/Documentation-and-Software
- George Marsaglia - Diehard tests

  https://web.archive.org/web/20160125103112/http://stat.fsu.edu/pub/diehard/
- Pierre L'Ecuyer - TestU01

  https://simul.iro.umontreal.ca/testu01/tu01.html

Concluding remarks

Rの乱数について

- おおよそ統計処理に適切な乱数が実装されている
- 好ましくは異なる乱数生成アルゴリズムを用いて特定の生成器に依存しない
- 並列処理では clusterSetRNGStream を用いて "L'Ecuyer-CMRG" を使う
- 物理乱数っぽいものもあるにはあります

  https://cran.r-project.org/src/contrib/Archive/Rrdrand/