24 March 2005

Multiple Recursive Random Number Generators and their APL programmes

Jean-Louis Foulley, INRA-SGQA
78352 Jouy-en-Josas Cedex, France
E-mail: jean-louis.foulley@jouy.inra.fr

As stated by many authors, the Linear Congruential Generators (LCGs) proposed by Lehmer (1951), including the most popular one, $X_i = 16807 X_{i-1} \bmod 2^{31} - 1$ (Rittaud, 2004), are known to have several serious defects such as short periods regarding our today needs and poor multidimensional structures. This is especially the case for power-two moduli which should be absolutely avoided (L'Ecuyer, 1998): see table 1 extracted from L'Ecuyer (2004) for a list of such generators.

There are now alternative tools that correct such defects and that are both portable and efficient and, in addition, may be implemented without much difficulty. Most of them belong to the class of the so-called Multiple Recursive Generators (MRG) which are linear congruential generators based on a state vector having k elements.

$$X_i = (\alpha_1 X_{i-1} + \alpha_2 X_{i-2} + ... + \alpha_k X_{i-k}) \bmod m \,. \qquad (1)$$

They can be divided into two categories according to the value of their modulus m. The "digital" ones are based on $m = 2$ and thus produce a sequence of bits. Those can then be combined by blocks of L bits to generate "words" which can be easily converted (dividing them by 2^L) into uniform (0,1) numbers. This class is the basis for the so-called Shift Register Generators eg the Tausworthe (1965) generator, the Generalized Feedback (GFSR) generators of Lewis and Payne (1973) and the Twisted GFSR generators including the most famous one, the Mersenne-Twister "MT1937" proposed by Matsumoto and Nishimura (1998). For more details see eg the reviews by Deng (1998), Hellekalek (1998), L'Ecuyer (1998) and Law and Kelton (2000).

The second class comprises generators build with a large modulus, usually a prime number to insure the largest period. Uniform random numbers on $[0,1)$ are in this case classically calculated as $U_i = X_i / m$. To avoid the occurrence of zeroes, some procedures have been proposed. For instance, Deng and Xu (2003) proposed to compute U_i as $U_i = (X_i + ½)/m$. L'Ecuyer (1999) converts $X_i = 0$ into $X_i = m$ and divides by $m+1$. We will strictly apply the procedure chosen by the author of the generator considered so as to make it perfectly reproducible.

We will restrict our attention here to such MRG's based on

i) the most common value for a prime m ie $m = 2^{31} - 1$;

ii) a large value of k.

Provided the polynomial associated to (1) $P(x) = x^k - \alpha_1 x^{k-1} - \alpha_2 x^{k-2} - ....\alpha_k$ is a primitive polynomial, the MRG in (1) reaches its maximum period $p = m^k - 1$ and also has the property of equidistribution up to dimension k. The generators described here are presented in papers by Deng and Xu (2003) and more recently by Deng (2005) following initial work on fast MRGs by Deng and Lin (2000).

DX-1597-4d

The first type of generators is a special class of MRGs (called DX-k in Deng and Xu's (2003) terminology) with the following formula (Deng, 2005, page 6):

$$X_i = B(X_{i-t} + X_{i-\lceil k/3 \rceil} + X_{i-\lceil 2k/3 \rceil} + X_{i-k}) \bmod m \qquad (2)$$

where B is the common multiplier $B = \alpha_t = \alpha_{\lceil k/3 \rceil} = \alpha_{\lceil 2k/3 \rceil} = \alpha_k$, $\lceil . \rceil$ the ceiling function, t refers to the smallest index for which the coefficient of X_{i-t} is not zero; m is the usual modulus equal to $2^{31}-1$. Here we will pay special attention to $t = 1$.

The one with the largest state vector is the DX-1597-4 ($k = \mathbf{1597}$) and, within that class, the one with the largest multiplier (desirable property with respect to the lattice structure) uses $B = 1\,073\,741\,362$ (table II, column d, last row in Deng, 2005). Here $\lceil k/3 \rceil = 533$ and $\lceil 2k/3 \rceil = 1065$ so that this generator is defined as

$$X_i = 1073741362(X_{i-1} + X_{i-533} + X_{i-1065} + X_{i-1597}) \bmod 2^{31}-1 \qquad (3)$$

This formulation requires a 64-bit integer data type. When this norm is not available, one can employ, as explained in the paper, an indirect computation based on writing B as $C_1 C_2 = B \bmod p$ with $e = (C_1 C_2 - B)/p$ having the smallest value for $0 < C_1, C_2 < 2^{19}$. This assumes that a double precision IEEE 754 standard is available. Here $C_1 = 29746$ and $C_2 = 36097$ and $e = 0$ and the DX-1597-4d is calculated as

$$\boxed{X_i = 29746\left[36097(X_{i-1} + X_{i-533} + X_{i-1065} + X_{i-1597}) \bmod 2^{31}-1\right] \bmod 2^{31}-1} \qquad (4)$$

Here are the corresponding APL programmes.

At start we need to initialize the 1597 element-seed. I suggest to do it simply via the usual linear congruent generator LCG (7^5, 2^{31} -1) with an initial value of 1 which gives
16807 282475249 1622650073 984943658 1144108930,…., 1476003502 1607251617 2028614953 1481135299 1958017916
(see the programme in appendix A for LCG)

Output: R a quasi random number between 0 and 2^{31} -2

```
R←DX15974;X;M
M←¯1+2*31
R←M|29746×M|36097×+/R1597[1597 1065 533 1]
R1597←(1↓R1597),R⍬
```

Example:Using the previous seed R1597, one obtains for 10 successive draws
221240004 2109349384 527768079 238300266 1495348915 1589596592 1437773979 813027151 401290350 1732813760

Note that this programme is not intended for a direct use in simulation works, but it may be useful to check that outputs are exactly reproducible.

Input: N: size of the vector of N uniform numbers to generate
Output: R such a vector of U(0,1) numbers calculated as $U_i = (X_i + ½)/m$

```
 R←GDX15974 N;M;I
 ⍝ DX-1597-4 MRG OF DENG 2005 WITH MULTIPLIER B=1 073 741 362
 ⍝ SEED R1597= A 1597-VECTOR OF NON ZERO INTEGERS
 ⍝ INPUT N =NUMBER OF U(0,1) VALUES TO GENERATE
 ⍝ OUTPUT THE N VECTOR OF SUCH VALUES
 M←¯1+2*31 ⋄ I←1 ⋄ R←N⍴0
STEP1:R[I]←M|29746×M|36097×+/R1597[1597 1065 533 1]
 R1597←(1↓R1597),R[I] ⋄ I←I+1
 →(I≤N)/STEP1 ⋄ R←(0.5+R)÷M
```

With the same 1597-seed as before, we get for N=50, one obtains

-as the first five

0.1030229053 0.9822423502 0.2457611634 0.1109672089 0.6963261013

-as the last five

0.3426870549 0.1907795485 0.7101110752 0.9272213492 0.5966575984

Input: a positive integer N

Output: R a random "equiprobable" draw among the integers 1,2,…,N calculated as $\left[N \,\mathrm{x}\, U_i(0,1)\right]+1$ where $[.]$ is the symbol for the integer part.

```
 R←ROLL15974 N;M
 M←¯1+2*31
 R←M|29746×M|36097×+/R1597[1597 1065 533 1]
 R1597←(1↓R1597),R
 R←⌊1+N×(R+.5)÷M
```

Example : 10 successive draws among numbers from 1 to 10000, are

1031 9823 2458 1110 6964 7403 6696 3786 1869 8070

DX-643-4d

The same can be done by applying formula (2) with $k = \mathbf{643}$, $s = 4$, $t = 1$ and $B = 1\ 073\ 740\ 543$ resulting in DX-643-4d. In that case, the recurrence is (Deng, Table II, column d, and following ones)

$$X_i = 1073740543(X_{i-1} + X_{i-215} + X_{i-429} + X_{i-643}) \bmod\ 2^{31} - 1 \qquad (5)$$

It is computed via

$$\boxed{X_i = 30559\left[105410(X_{i-1} + X_{i-215} + X_{i-429} + X_{i-643}) \bmod\ 2^{31} - 1\right] \bmod 2^{31} - 1} \qquad (6)$$

The programmes are

```
 R←DX6434;M
 M←¯1+2*31
 R←M|30559×M|105410×+/R643[643 429 215 1]
 R643←(1↓R643),R⌊
```

Example:Starting with a seed R643 equal to 16807 282475249 1622650073 984943658 1144108930 470211272 101027544…. ie based on LCG (7^5, 2^{31} -1) with an initial value of 1, one obtains for 10 successive draws:

1641505334 103236556 721745135 104437320 329533308 1025183836 1860188164 329379879 255862529 2125528287

Input: N: size of the vector of N uniform numbers to generate
Output: R such a vector of U(0,1)'s

```
 R←GDX6434 N;M;I
 ⍝ DX-643-4 MRG OF DENG 2005 WITH MULTIPLIER B=1 073 740 543
 ⍝ SEED R643= A 643 VECTOR OF NON ZERO INTEGERS
 ⍝ INPUT N =NUMBER OF U(0,1) VALUES TO GENERATE
 ⍝ OUTPUT THE N VECTOR OF SUCH VALUES
 M←¯1+2*31 ⋄ I←1 ⋄ R←N⍴0
STEP1:R[I]←M|30559×M|105410×+/R643[643 429 215 1]
 R643←(1↓R643),R[I] ⋄ I←I+1
 →(I≤N)/STEP1 ⋄ R←(0.5+R)÷M⍝
```

With the same 643-seed as before, we get for N=50, one obtains
-as the first five
0.7643854875 0.04807326782 0.3360887691 0.04863241713 0.1534509047
-as the last five
0.2580945304 0.9492599207 0.3861052375 0.1677643827 0.4536414728

Input: a positive integer N
Output: R a random draw among the integers 1,2,…,N

```
 R←ROLL6434 N;M
 M←¯1+2*31
 R←M|30559×M|105410×+/R643[643 429 215 1]
 R643←(1↓R643),R
 R←⎕IO+⌊N×(R+0.5)÷M⍝
```

Example: 10 successive draws among numbers from 1 to 10000, are

7644 481 3361 487 1535 4774 8663 1534 1192 9898

DX-47-4b

Finally, one may wish to use something simpler and lighter but of good quality. Then, we suggest the **DX-47-4** based on $B = 46281$ (see table II on page 8, column b) corresponding to

$$X_i = 46281(X_{i-1} + X_{i-16} + X_{i-32} + X_{i-47}) \bmod 2^{31} - 1. \qquad (6)$$

The corresponding programmes (made along the same lines as previously) are

Output: R a quasi random number between 0 and 2^{31} -2

```
 R←DX474;M
 M←¯1+2*31
 R←M|46281×+/R47[47 32 16 1]
 R47←(1↓R47),R⍝
```

Example : Starting from a seed R47 equal to 16807, 282475249, 1622650073, 984943658, 1144108930,… one gets

839071403 1731758405 1606050126 1443462404 2109690996 2114024150 298132109 628783979 817598807 1011726052

Input: N: size of the vector of N uniform numbers to generate
Output: R such a vector of U (0,1)'s

```
 R←GDX474 N;M;I
 ⍝ DX-47-4 MRG OF DENG 2005 WITH MULTIPLIER B=46281
 ⍝ SEED R47= A 47-VECTOR OF ON ZERO INTEGERS
 ⍝ INPUT N =NUMBER OF U(0,1) VALUES TO GENERATE
 ⍝ OUTPUT THE N VECTOR OF SUCH VALUES
 M←¯1+2*31 ⋄ I←1 ⋄ R←N⍴0
STEP1:R[I]←M|46281×+/R47[47 32 16 1]
 R47←(1↓R47),R[I] ⋄ I←I+1
 →(I≤N)/STEP1 ⋄ R←(0.5+R)÷M⍝
```

Example: With the same 47-seed as before, we get for N=50, one obtains
-as the first five
0.3907230701 0.8064128488 0.7478753697 0.6721645618 0.9824014257
-as the last five
0.8843225815 0.9192814191 0.820364061 0.02971864796 0.4020915785

Input: a positive integer N
Output: R a random draw among the integers 1,2,…,N

```
 R←ROLL474 N;M
 M←¯1+2*31
 R←M|46281×+/R47[1 16 32 47]
 R47←(1↓R47),R
 R←⎕IO+LN×(R+0.5)÷M⍝
```

Example: 10 successive draws among numbers from 1 to 10000, are
3908 8461 9571 2366 933 3699 1971 7298 5619 9900
3908 8065 7479 6722 9825 9845 1389 2929 3808 4712

MRG-1597-2

Deng (2005) also proposed MRG-k-s generators based on different construction procedures. For $k = \mathbf{1597}$ and $s = 2$ (**MRG-1597-2**), he gave this generator

$$X_i = 1057217510 X_{i-1} + 1066409146 X_{i-1597} \bmod 2^{31} - 1. \qquad (7)$$

When a 64-bit integer type is not available, it can be computed as

$$\boxed{X_i = \left[44653\left(71769 X_{i-1} \bmod 2^{31}-1\right) + 48619\left(21934 X_{i-1597} \bmod 2^{31}-1\right)\right] \bmod 2^{31}-1} \quad (8)$$

This generator, as DX-1597-4 described previously, has a period of $m^k - 1 = \left(2^{31}-1\right)^{1597} - 1$ ie about $\boxed{10^{14903}}$ an has a uniform distribution up to dimension $\boxed{k = 1597}$.
Although this generator is a little bit more complicated than the corresponding DX-1597-s, it has a more complex structure which, according to Deng, makes "harder to devise a general empirical test that such generators will fail".

The programmes are as follows:

```
R←MRG15972;M
 M←¯1+2*31
 R←M|+/(44653 48619)×M|(71769 21934)×R1597[1597 1]
 R1597←(1↓R1597),R⌷
```

Example: starting with a seed R1597 equal to 16807 282475249 1622650073 984943658 1144108930 470211272 101027544… ie based on LCG (7^5, 2^{31} -1) with an initial value of 1, one obtains for 10 successive draws :
1811133916 491217212 31477969 917602403 1251137860 2141366420 1997727199 1852033570 34235151 178125418

Input: N: size of the vector of N uniform numbers to generate
Output: R such a vector of U(0,1)'s

```
 R←GMRG15972 N;M;I
 ⍝ MRG-1597-2 MRG OF DENG 2005 WITH MULTIPLIER B=1 057 217 510
 ⍝ SEED R1597= A 1597 VECTOR OF NON ZERO INTEGERS
 ⍝ INPUT N=NUMBER OF U(0,1) VALUES TO GENERATE
 ⍝ OUTPUT THE N VECTOR OF SUCH VALUES
 M←¯1+2*31 ⋄ I←1 ⋄ R←N⍴0
STEP1:R[I]←M|+/(44653 48619)×M|(71769 21934)×R1597[1597 1]
 R1597←(1↓R1597),R[I] ⋄ I←I+1
 →(I≤N)/STEP1 ⋄ R←(0.5+R)÷M⌷
```

Example: With the same 1597-seed as before, we get for N=50,
-as the first five
0.8433749514 0.2287408396 0.01465807181 0.4272919166 0.582606467
as the last five
0.3458714908 0.3731809076 0.1382221401 0.2910157814 0.9041655634

Input: a positive integer N
Output: R a random draw among the integers 1,2,…,N

```
 R←ROLL15972 N;M
 M←¯1+2*31
 R←M|+/(44653 48619)×M|(71769 21934)×R1597[1597 1]
 R1597←(1↓R1597),R
 R←⌊1⌷0+LN×(R+.5)÷M⌷
```

Example : 10 successive draws among numbers from 1 to 10000, are
8434 2288 147 4273 5827 9972 9303 8625 160 830

MRG32k3a

Another way to deal with a large modulus in a MRG is to combine MRG's with smaller moduli (L'Ecuyer, 1998; Deng et al, 2005). The generator considered here is a standard one from L'Ecuyer (1999) known as MRG32k3a. It combines two MRG's of order $k = 3$ and is defined as follows:

$$X_{1,i} = (\alpha_{11} X_{1,i-1} + \alpha_{12} X_{1,i-2} + \alpha_{13} X_{1,i-3}) \bmod m_1 \quad (9)$$

$$X_{2,i} = (\alpha_{21} X_{2,i-1} + \alpha_{22} X_{2,i-2} + \alpha_{23} X_{2,i-3}) \bmod m_2 \quad (10)$$

with

$$\alpha_{11} = 0, \alpha_{12} = 1403580, \alpha_{13} = -810728 \text{ and } m_1 = 2^{32} - 209 \text{ ;} \quad (11)$$

$$\alpha_{21} = 527612, \alpha_{22} = 0, \alpha_{23} = -1370589 \text{ and } m_2 = 2^{32} - 22853. \quad (12)$$

The final generator combines these two according to

$$Z_i = (X_{1,i} - X_{2,i}) \bmod m_1 . \quad (13)$$

The corresponding uniform (0,1) generator is calculated as (l'Ecuyer, 1999)

$$U_i = Z_i^* / (m_1 + 1), \quad (14)$$

where Z_i^* corresponds to $Z_i^* = Z_i$ if $Z_i > 0$ and $Z_i^* = m_1$ otherwise. This avoids the occurrence of zeroes and ones that might cause problems for simulating some non-uniform distributions: eg the logistic, exponential, Weibull (Devroye, 1986; Evans et al, 1993).

According to L'Ecuyer (1999), this generator is approximatively equivalent to a MRG of order $k = 3$ with a modulus $m = m_1 m_2$, $\alpha_1 = 18446645023178547541$, $\alpha_2 = 3186860506199273833$ and $\alpha_3 = 87386136264398222622$.

It has a period of $p = (m_1^3 - 1)(m_2^3 - 1)/2 \approx 2^{191} \simeq \mathbf{3x10^{57}}$. It is also well behaved up to dimension 45. Its implementation requires as previously the IEEE 754 double precision standard.

The APL programmes proposed here take advantage of the vector operating procedures of APL to calculate the two component generators simultaneously.

At start we will need to initiate the seed R32K as a (3x2) matrix build from the usual LCG eg

```
      16807    984943658
  282475249   1144108930
1622650073    470211272
```

ie seed for X1: 16807 282475249 1622650073
and for X2: 984943658 1144108930 470211272

Input: N: size of the vector of N uniform numbers to generate
Output: R such a vector of U(0,1)'s computed as in (14).

```
 R←GMRG32K N;M;M1;C;I;X
 ⍝ COMBINED MRG 'MRG32K3A' BY L'ECUYER 1999
 ⍝ SEED R32K= A (3X2) MATRIX OF INTEGERS NOT ALL ZERO
 ⍝ INPUT N =NUMBER OF U(0,1) VALUES TO GENERATE
 ⍝ OUTPUT THE N VECTOR OF SUCH VALUES
 M←(M1←¯209+2*32),¯22853+2*32 ⋄ I←1 ⋄ R←N⍴0
 C←⍉(2 3)⍴¯810728 1403580 0 ¯1370589 0 527612
STEP1:X←M1|+/[1]C×R32K
 R32K←(1 0↓R32K),[1]X
 R[I]←(X×X≠0)+((X←M1|-/X)=0)×M1
 I←I+1 ⋄ →(I≤N)/STEP1 ⋄ R←R÷M1+1⍝
```

Example : With the same seed as before, we get for N=50, one obtains
-as the first five
0.7669364155 0.7286176883 0.5890946068 0.2480655726 0.2741894033
as the last five
0.264122945 0.1468770745 0.5614629734 0.177519304 0.7555685728

If for instance, one wishes to produce draws of 32-bit integers (numbers ranging from 0 to $2^{32}-1$), one simply has to calculate $\left\lfloor 2^{32}U_i(0,1)\right\rfloor$, ie the integer part (symbolized by $\lfloor . \rfloor$) of the product of 2^{32} by an uniform random number defined as before; this can be done in APL with: `⌊(2*32)x GMRG32K N` for N draws. Here for N=10, one has
3293966822 3129389142 2530142070 1065433521 1177634520 1644939348 3413537337 1852571700 115527021 783713440.
This example was used as a test of validation of results from our APL programmes with the original ones developed by L'Ecuyer's team (see appendix B).

Input: a positive integer N
Output: R a random draw among the integers 1,2,…,N calculated as $\left\lfloor N \,\mathrm{x}\, U_i(0,1)\right\rfloor+1$

```
 R←ROLL32K N;M;M1;C;X
 M←(M1←¯209+2*32),¯22853+2*32
 C←⍉(2 3)⍴¯810728 1403580 0 ¯1370589 0 527612
 X←M1|+/[1]C×R32K
 R32K←(1 0↓R32K),[1]X
 X←(X×X≠0)+((X←M1|-/X)=0)×M1
 R←⎕IO+⌊N×X÷M1+1⍝
```

Example: 10 successive draws among numbers from 1 to 10000, are
7670 7287 5891 2481 2742 3830 7948 4314 269 1825

A programme in C is also provided: see appendix C

Conclusions

The generators presented here offer good opportunities to update the classical LCG generators still used in many instances. There are alternatives to the modulus-two generators and their GFSR and other extensions provided by some softwares (eg "R" and the MT-19937). The formulation and programming of the MRG and c-MRG's shown here are easy provided a double precision IEEE 754 standard is available. Moreover, they have passed the "Crush" batterie of stringent tests defined in the well-known TESTU01 suite developed by Prof Pierre L'Ecuyer.

Finally, among the possible choices, I would suggest the two generators:

-the **DX-1597-4d** by Deng (2005) as it has the largest order k among the MRGs, more non-zero terms and is more efficient numerically than the MRG-1597-2;

-the **MRG32k3a** by L'Ecuyer (1999) for its intrinsic properties despite a shorter period and also due to its increasing popularity (SAS, Arena, Automod, …).

Remember the advice given by most specialists in this field: "The most prudent policy for a person to follow is to run a Monte Carlo program at **least** **twice** using quite different sources of random numbers before taking the answers of the programme seriously; this not only will give an indication of the stability of the results, it will also guard against the danger of trusting in a generator with hidden deficiencies" (Knuth, 1998).

Acknowledgements

I am indebted to Professor Lih-Yuan Deng from the Department of Mathematical Sciences of the University of Memphis, Memphis, TN for his valuable assistance in reading this manuscript, providing me with some of his recent works and also checking the numerical examples of the DX-k and MRG generators shown here.

Special thanks are also expressed to Dr Richard Simard from "Laboratoire de Simulation et d'Optimisation", Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, who checked my test examples for the MRG32ka and gave me further explanations about this generator.

I also would like to thank the Monolix ("Modèles Non Linéaires Mixtes") group for stimulating discussions on that subject, and a few other people from the APL community (S Baron, E Lescasse, E McDonnell, M Righetti, B Rutiser and my INRA colleagues JJ Colleau and L Ollivier) for their real interest in this matter. I am especially grateful to JJ Colleau for some suggestions which greatly improve the efficiency of the APL programmes.

Sites

http://www.cs.memphis.edu/~dengl/dx-rng/ Prof Lih-Yuan Deng

http://www.iro.umontreal.ca/~lecuyer/ Prof Pierre L'Ecuyer

http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html Matsumoto "Mersenne Twister"

References

Deng L.Y., Uniform random numbers, Encyclopedia of Biostatistics, in: Armitage P, Colton T (Eds), John Wiley & Sons, New York, 1998, Vol 5, pp. 4651-4656.

Deng L.Y., Efficient and Portable Multiple Recursive Generators of Large Order, *ACM Transactions on Modeling and Computer Simulation* 15 (2005) 1-13.

Deng L.Y., Lin D.K.J., Random number generation for the new century, *The American Statistician* 54 (2000) 145-150.

Deng L.Y., Xu H., A system of high-dimensional efficient, long-cycle and portable uniform random generators, *ACM Transactions on Modeling and Computer Simulation* 13 (2003) 299-309.

Deng L.Y., Li H., Shiau J-J.H., Theory and practice of combination generators, 2005 (submitted).

Devroye L., Non uniform random generation, Springer Verlag, Berlin, 1986.

Evans M., Hastings N., Peacock B., Statistical distributions, 2nd edition, J Wiley, NY, 1993.

Hellekalek P., Good random generators are (not so) easy to find, *Mathematics and Computers in Simulation*, 46 (1998) 485-5005.

Knuth D.E., The Art of Computer Programming, volume 2 : Semi-numerical Algorithms. Addison-Wesley, Reading, MA, 3nd edition, 1998.

Law A.M., Kelton W.D., Simulation modelling and analysis, 3rd edition, Mc Graw Hill, Boston, 2000.

L'Ecuyer P., Random number generation, Handbook on Simulation, Chapter 4, Wiley, 1998.

L'Ecuyer P., Good parameters and implementation for combined multiple recursive random number generators, *Operations Research* 47 (1999) 159-164.

L'Ecuyer P., Simulation: aspects stochastiques, IFT6561, Notes de cours, Chapitre III, partie I, http://www.iro.umontreal.ca/~lecuyer/cours.html, 2004

Lehmer D.H., Mathematical methods in large-scale computing units, *Annals of Computer Laboratory of the Harvard University* 26 (1951) 141-146.

Lewis T.G., Payne W.H., Generalized feedback shift register pseudo-random number, algorithm, Journal *of the ACM* 20 (1973) 456-468.

Matsumoto M., Nishimura T., Mersenne-Twister: a 623-dimensionally equidistributed uniform pseudorandom number generator, *ACM Transactions on Modeling and Computer Simulation* 8 (1998) 3-20.

Rittaud B., Fabriquer le hasard, 1. L'ordinateur à rude épreuve*, La Recherche* 381 (2004), 28-33.

Tausworthe R.C., Random number generated by linear recurrence modulo two, *Mathematics of Computation* 19 (1965) 201-209.

Appendix A:**Programme for the LCG(7^5, 2^{31} -1)**

Input: N=number of values generated; Output: the corresponding vector of pseudo-random numbers

At start, the seed ⎕RL is set to a given value

```
 R←LCGM N;I;M
 ⍝ LCG (M=(2*31)-1;A=7*5) LEWIS GOODMAN MILLER 1969
 ⍝ SEED '⎕RL' A SCALAR NON ZERO INTEGER
 ⍝ INPUT N =NUMBER OF U(0,1) VALUES TO GENERATE
 ⍝ OUTPUT THE N VECTOR OF SUCH VALUES
 M←¯1+2*31 ⋄ I←1 ⋄ R←N⍴0
STEP1:R[I]←⎕RL←M|⎕RL×16807
 I←I+1 ⋄ →(I≤N)/STEP1⍝
```

Example : With a seed of 1, one gets for the first ten draws:
16807 282475249 1622650073 984943658 1144108930 470211272 101027544 1457850878 1458777923 2007237709

Appendix B: **A test example for MRG32k3a**

17/03/2005 JL Foulley, INRA-SGQA, Jouy en Josas, France

Le générateur retourne des flottants dans (0, 1). J'inclus le programme C du générateur. Si l'on veut générer des entiers "aléatoires" de 32 bits suivant la loi uniforme, alors on doit le faire comme suit:

Yi = Entier[Ui * 2^32]

J'ai ainsi généré 10 entiers de 32 bits à partir des 6 germes que vous avez choisis. Cela donne

ulec_CreateMRG32k3a: (s10, s11, s12, s20, s21, s22) = (16807, 282475249, 1622650073, 984943658, 1144108930, 470211272)

3293966822
3129389142
2530142070
1065433521
1177634520
1644939348
3413537337
1852571700
115527021
783713440

Pour votre information, MRG32k3a est le générateur inclus dans les nouvelles versions de plusieurs logiciels commerciaux (SAS, Arena, Automod, ...) de simulation ou de statistiques très répandus. Il est important que toutes les implantations retournent les mêmes résultats pour une même simulation utilisant 32 bits ou moins.

==========

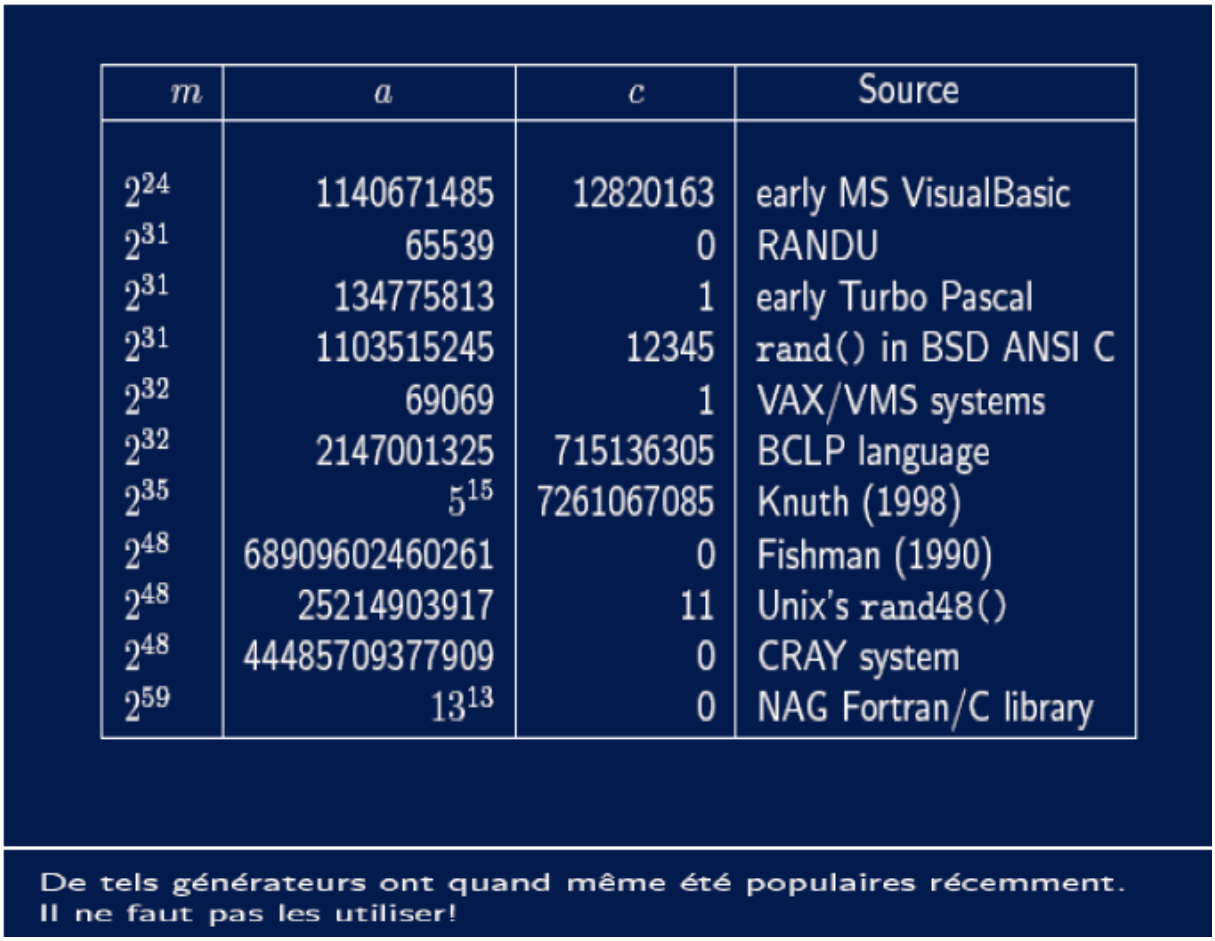
Richard Simard <simardr@iro.umontreal.ca>
Laboratoire de simulation et d'optimisation
Université de Montréal, Dépt. IRO

```
Workspace D:\Home\ugenjlf\user\RANDOM

Dyalog APL/W Version 10.0.2
Serial No : 001080 / Pentium
Thu Mar 24 12:32:30 2005
CONTINUE saved Tue Sep 28 15:26:19 2004
D:\Home\ugenjlf\user\RANDOM saved Thu Mar 24 12:32:24 2005
      ∇GMRG32K[□]∇
[0]   R←GMRG32K N;M;M1;C;I;X
[1]   ⍝ COMBINED MRG 'MRG32K3A' BY L'ECUYER 1999
[2]   ⍝ SEED R32K= A (3X2) MATRIX OF INTEGERS NOT ALL ZERO
[3]   ⍝ INPUT N =NUMBER OF U(0,1) VALUES TO GENERATE
[4]   ⍝ OUTPUT THE N VECTOR OF SUCH VALUES
[5]   M←(M1←¯209+2*32),¯22853+2*32 ⋄ I←1 ⋄ R←N⍴0
[6]   C←⍉(2 3)⍴¯810728 1403580 0 ¯1370589 0 527612
[7]  STEP1:X←M|+/[1]C×R32K
[8]   R32K←(1 0↓R32K),[1]X
[9]   R[I]←(X×X≠0)+((X←M1|-/X)=0)×M1
[10]  I←I+1 ⋄ →(I≤N)/STEP1 ⋄ R←R÷M1+1
      □RL←1⋄R32K←⍉(2 3)⍴LCGM 6
      ESSAI←⌊(2*32)×GMRG32K 10
      ESSAI
3293966822 3129389142 2530142070 1065433521 1177634520 1644939348
...    3413537337 1852571700
      115527021 783713440
```

Appendix C: **C programme for MRG32k3a** (due to Pierre l'Ecuyer and provided by Richard Simard)

```
#define norm 2.328306549295728e-10
#define m1   4294967087.0
#define m2   4294944443.0
#define a12     1403580.0
#define a13n     810728.0
#define a21      527612.0
#define a23n    1370589.0

static double s10 = 12345, s11 = 12345, s12 = 12345,
              s20 = 12345, s21 = 12345, s22 = 12345;

double MRG32k3a (void)
{
  long   k;
  double p1, p2;
  /* Component 1 */
  p1 = a12 * s11 - a13n * s10;
  k = p1 / m1;   p1 -= k * m1;   if (p1 < 0.0) p1 += m1;
  s10 = s11;   s11 = s12;   s12 = p1;

  /* Component 2 */
  p2 = a21 * s22 - a23n * s20;
  k  = p2 / m2;  p2 -= k * m2;   if (p2 < 0.0) p2 += m2;
  s20 = s21;   s21 = s22;   s22 = p2;

  /* Combination */
  if (p1 <= p2) return ((p1 - p2 + m1) * norm);
  else return ((p1 - p2) * norm);
}
```

Table 1: LCG generators with a modulus power two (from L'Ecuyer, 2004)



m	a	c	Source
2^{24}	1140671485	12820163	early MS VisualBasic
2^{31}	65539	0	RANDU
2^{31}	134775813	1	early Turbo Pascal
2^{31}	1103515245	12345	`rand()` in BSD ANSI C
2^{32}	69069	1	VAX/VMS systems
2^{32}	2147001325	715136305	BCLP language
2^{35}	5^{15}	7261067085	Knuth (1998)
2^{48}	68909602460261	0	Fishman (1990)
2^{48}	25214903917	11	Unix's `rand48()`
2^{48}	44485709377909	0	CRAY system
2^{59}	13^{13}	0	NAG Fortran/C library