Generation of Uniform Random Numbers

Two different random number generators are available:

- **SPSS 12 Compatible.** The random number generator used in SPSS 12 and previous releases. If you need to reproduce randomized results generated in previous releases based on a specified seed value, use this random number generator.
- **Mersenne Twister.** A newer random number generator that is more reliable for simulation purposes. If reproducing randomized results from SPSS 12 or earlier is not an issue, use this random number generator.

Specifically, the Mersenne Twister has a far longer period (number of draws before it repeats) and far higher order of equidistribution (its results are “more uniform”) than the SPSS 12 Compatible generator. The Mersenne Twister is also very fast and uses memory efficiently.

SPSS 12 Compatible Random Number Generator

Uniform numbers are generated using the algorithm of (Fishman and Moore, 1981). It is a multiplicative congruential generator that is simply stated as:

```
seed(t+1) = (a * seed(t)) modulo p
rand = seed(t+1) / (p+1)
```

where a = 397204094 and p = $2^{31}-1$ = 2147483647, which is also its period. Seed(t) is a 32-bit integer that can be displayed using `SHOW SEED`. `SET SEED=`*number* sets seed(t) to the specified number, truncated to an integer. `SET SEED=RANDOM` sets seed(t) to the current time of day in milliseconds since midnight.

Mersenne Twister Random Number Generator

The Mersenne Twister (MT) algorithm generates uniform 32-bit pseudorandom integers. The algorithm provides a period of $2^{19937}-1$, assured 623-dimensional equal distribution, and 32-bit accuracy. Following the description given by Matsumoto and Nishimura (1998), the algorithm is based on the linear recurrence:

$$\mathbf{x}_{k+n} = \mathbf{x}_{k+m} \oplus \left(\mathbf{x}_k^u | \mathbf{x}_{k+1}^l\right) \mathbf{A},\ k = 0, 1, \cdots \tag{1}$$

where

$\mathbf{x}$	is a word vector; a *w*-dimensional row vector over the two-element field $\mathbf{F}_2 = \{0, 1\}$
n	is the degree of recurrence (recursion)
r	is an integer, $0 \leq r \leq w-1$, the separation point of one word
m	is an integer, $1 \leq m \leq n$, the middle term
$\mathbf{A}$	is a constant $w \times w$ matrix with entries in $\mathbf{F}_2$
$\mathbf{x}_k^u$	is the upper (*w*–*r*) bits of $\mathbf{x}_k$
$\mathbf{x}_{k+1}^l$	is the lower *r* bits of $\mathbf{x}_{k+1}$; thus
$\mathbf{x}_k^u \vert \mathbf{x}_{k+1}^l$	is the word vector obtained by concatenating the upper (*w*–*r*) bits of $\mathbf{x}_k$ and the lower *r* bits of $\mathbf{x}_{k+1}$
$\oplus$	Bitwise addition modulo two (XOR)

Given initial seeds $\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_{n-1}$, the algorithm generates $\mathbf{x}_{n+k}$ by the above recurrence for *k*=0, 1, ...

A form of the matrix **A** is chosen so that multiplication by **A** is very fast. **A** candidate is

$$\mathbf{A} = \begin{pmatrix} & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ a_{w-1} & a_{w-2} & \cdots & \cdots & a_0 \end{pmatrix} \tag{2}$$

where $\mathbf{a} = (a_{w-1}, a_{w-2}, \cdots, a_0)$ and $\mathbf{x} = (x_{w-1}, x_{w-2}, \cdots, x_0)$; then $\mathbf{xA}$ can be computed using only bit operations

$$\mathbf{xA} = \begin{cases} shiftright\,(\mathbf{x}) & if \quad x_0 = 0 \\ shiftright\,(\mathbf{x}) \oplus \mathbf{a} & if \quad x_0 = 1 \end{cases} \tag{3}$$

Thus calculation of the recurrence is realized with bitshift, bitwise EXCLUSIVE-OR, bitwise OR, and bitwise AND operations.

For improving the *k*-distribution to *v*-bit accuracy, we multiply each generated word by a suitable $w \times w$ invertible matrix $\mathbf{T}$ from the right (called tempering in (Matsumoto and Kurita, 1994)). For the tempering matrix $\mathbf{z} = \mathbf{xT}$, we choose the following successive transformations

$$\mathbf{y} := \mathbf{x} \oplus (\mathbf{x} >> u) \tag{4}$$

$$\mathbf{y} := \mathbf{y} \oplus ((\mathbf{y} << s)\ AND\, \mathbf{b}) \tag{5}$$

$$\mathbf{y} := \mathbf{y} \oplus ((\mathbf{y} << t)\ AND\, \mathbf{c}) \tag{6}$$

$$\mathbf{z} := \mathbf{y} \oplus ((\mathbf{y} >> l) \tag{7}$$

where

l, *s*, *t*, *u*	are integers
b, **c**	are suitable bitmasks of word size
$\mathbf{x} >> u$	denotes the u-bit shiftright
$\mathbf{x} << u$	denotes the u-bit shiftleft

To execute the recurrence, let **x**[0:*n*−1]be an array of *n* unsigned integers of word size, *i* be an integer variable, and $\mathbf{x}, \mathbf{v}, \mathbf{a}$ be unsigned constant integers of word size vectors.

Step 0 $\mathbf{u} \leftarrow \underset{w-r}{1\cdots1}\,\underset{r}{0\cdots0}$; bitmask for upper (*w*−*r*) bits
$\mathbf{v} \leftarrow \underset{w-r}{0\cdots0}\,\underset{r}{1\cdots1}$; bitmask for lower *r* bits
$\mathbf{a} \leftarrow a_{w-1}a_{w-2}\cdots a_0$; the last row of matrix $\mathbf{A}$

Step 1 $i \leftarrow 0$
Initialize the state space vector array $\mathbf{x}[0], \mathbf{x}[1], \cdots, \mathbf{x}[n-1]$.

Step 2 $\mathbf{y} \leftarrow (\mathbf{x}[i]\ AND\, \mathbf{u})\ OR\ (\mathbf{x}[(i+1)\ mod\, n])\ AND\, \mathbf{v}$; computing $\left(\mathbf{x}_i^u | \mathbf{x}_{i+1}^l\right)$

Step 3 If the least significant bit of $\mathbf{y}$ equals to zero then
$\mathbf{x}[i] \leftarrow \mathbf{x}[(i+m)\ mod\, n]\ \oplus\ (\mathbf{y} >> 1)\ \oplus\ 0$
If the least significant bit of $\mathbf{y}$ equals to one then
$\mathbf{x}[i] \leftarrow \mathbf{x}[(i+m)\ mod\, n]\ \oplus\ (\mathbf{y} >> 1)\ \oplus\ \mathbf{a}$

```
Step 4  calculate $\mathbf{x}[i]\,\mathbf{T}$
        $\mathbf{y} \leftarrow \mathbf{x}[i]$
        $\mathbf{y} \leftarrow \mathbf{y} \oplus (\mathbf{y} >> u)$
        $\mathbf{y} \leftarrow \mathbf{y} \oplus ((\mathbf{y} << s)\ AND\,\mathbf{b})$
        $\mathbf{y} \leftarrow \mathbf{y} \oplus ((\mathbf{y} << t)\ AND\,\mathbf{c})$
        $\mathbf{y} \leftarrow \mathbf{y} \oplus (\mathbf{y} >> l)$

Step 5  $i \leftarrow (i+1)\ mod\ n$

Step 6  Go to Step 2.
```

SPSS Usage

The MT algorithm provides 32 random bits in each draw. SPSS draws 64-bit floating-point numbers in the range [0..1] with 53 random bits in the mantissa using

Draw = $(2^{26}*[k(t)/2^5]+[k(t+1)/2^6])/2^{53}$

There are two options for initializing the state space vector array. `SET RNG=MT MTINDEX=`*x* accepts a 64-bit floating point number *x* to set the seed. `SET RNG=MT MTINDEX=RANDOM` uses the current time of day in milliseconds since midnight to set the seed.

```
init_genrand(unsigned32 s,unsigned32 &x[])
{

    $\mathbf{x}[0] = s$ ;
    $f$ = 1812433253; $f$ is an unsigned long interger from i=0 to n
    $\mathbf{x}[i+1] = f\,(\mathbf{x}[i] >> 30)\,mod 2^w$

}
```

k[0]: 8*d+4*c+2*b+a

k[1]: y = trunc($z*2^{26}$)

k[2]: $z*2^{53} - y*2^{27}$

where

x	is the argument
a	is 1 if x == 0, or 0 otherwise
b	is 1 if x<0, or 0 otherwise
c	is 1 if |x| >= 1, or 0 otherwise

d	is an integer such that
	if |x| > 1, .5 <= $\|x\|/2^d$ < 1,
	else if |x| > 0, .5 <= $\|x\|*2^d$ < 1

```
            else x == 0 and d == 0.
e    is d if |x| <= 1, else -d
z    is |x|*2^e
```

```
init_by_array(unsigend32 init_key[ ] ,int key_length, unsigned32 &x[])
{

    init_genrand(19650218, x);

    $i = 1, j = 0, k = \max(key\_length, n)$

        for ($;k;k--$)

            $\mathbf{x}[i] = (\mathbf{x}[i] \oplus ((\mathbf{x}[i-1] \oplus (\mathbf{x}[i-1] >> 30))\, f_1))$
            $+ init\_key[j] + j;$
            if $i >= n$ then
            $\mathbf{x}[0] = \mathbf{x}[n-1];$
            $i = 1;$
            if $(j >= key\_length)$ then
            $j = 0$ ;

        end for

        for ($k = n-1; k; k--$)

            $\mathbf{x}[i] = (\mathbf{x}[i] \oplus ((\mathbf{x}[i-1] \oplus (\mathbf{x}[i-1] >> 30))\, f_2)) - i$ ;
            if $i >= n$ then
            $\mathbf{x}[0] = \mathbf{x}[n-1];$
            $i = 1;$

        end for

}
```

f_1 = 1664525 is an unsigned long interger;
f_2 = 1566083941 is an unsigned long interger;

References

Fishman, G., and L. R. I. Moore. 1981. In search of correlation in multiplicative congruential generators with modulus 2**31-1. In: *Computer Science and Statistics, Proceedings of the 13th Symposium on the Interface,* W. F. Eddy, ed. New York: Springer-Verlag, 155–157.

Knuth, D. E. 1981. *The Art of Computer Programming, volume 2, p. 106.* Reading, MA: Addison-Wesley.

Matsumoto, M., and T. Nishimura. 1998. Mersenne Twister, A 623–dimensionally equidistributed uniform pseudorandom number generator. *ACM Trans. on Modeling and Computer Simulation*, 8:1, 3–30.

Matsumoto, M., and Y. Kurita. 1994. Twisted GFSR generators II. *ACM Trans. on Modeling and Computer Simulation*, 4:3, 254–266.