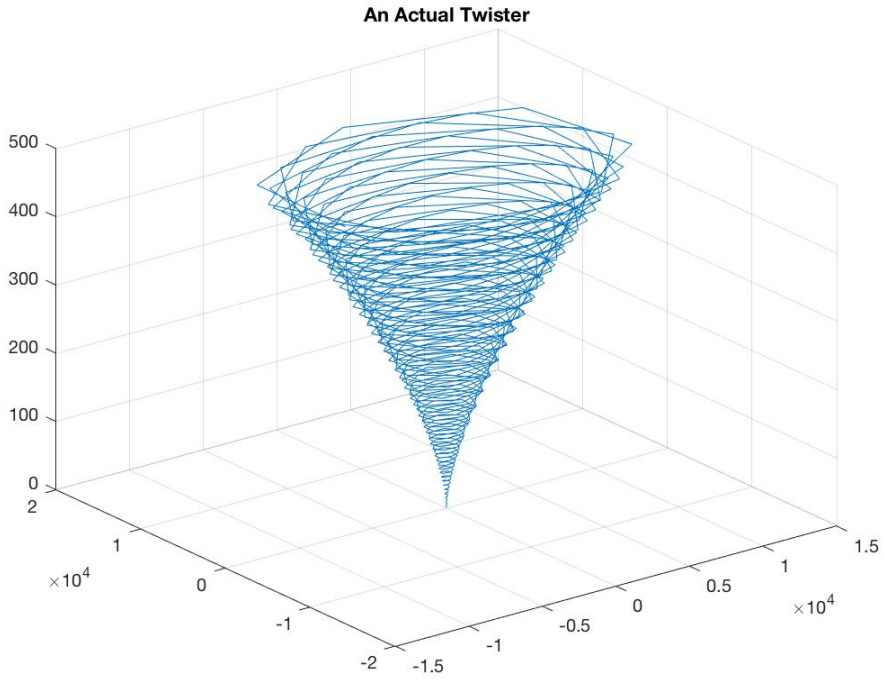
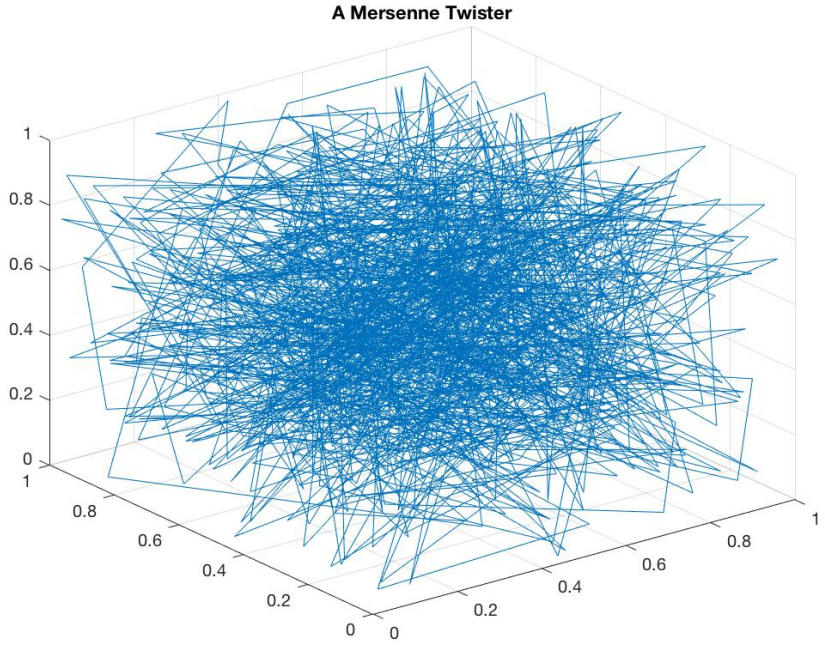
The Mersenne Twister Algorithm

Ammar Elsheshtawy





Ammar Elsheshtawy
Research Project (Draft 1)
Math 320
28 November 2016

The Mersenne Twister Algorithm

The "random" numbers generated in software packages such as MATLAB are not truly random – they are "pseudorandom." That is, they are from sequences of numbers with properties similar to those of truly unpredictable sequences, but they do not exactly match these properties. They are sufficiently unpredictable to give the appearance of randomness but they do not meet the formal criteria for this property. Such pseudorandom sequences are generated using algorithms which take in a set number of inputs and output a pseudorandom sequence. The outputs of these algorithms are deterministic – given the same input (the "seed") they will always produce the same output. There are a wide variety of RNG algorithms whose simulations of true randomness are deficient in different ways. This paper discusses the Mersenne Twister (MT) algorithm which is one of the most widely used and is the default option in MATLAB.

The MT was created by Makato Matsumoto and Tajuki Nishimura and the original paper describing the algorithm was published in 1998[1]. The preferred MT specification as originally described was notable for the large period of its pseudorandom sequences ($2^{19937} - 1$), having a high-dimensional "equidistribution," being computationally efficient with a small working memory. Various adaptations and specifications of the MT have been made since the original published paper. This paper focuses on the MT19937 specification as described in the original published paper, although with a revised "initialization procedure" that was proposed in 2002.

The rest of the paper is as follows: an explanation of the MT algorithm (including selection of parameters), a custom MATLAB implementation of MT, a discussion of how the MT's "randomness" properties are verified, a discussion of the pros and cons of the MT, and a discussion of how the MT's qualities are suited or unsuited to certain applications.

Note: This is a first draft of the paper and is incomplete. A few sections are yet to be written. Planned additions/clarifications for the next draft include (but are not limited to): tidying up formatting, inclusion of more/proper citations, inclusion of a bibliography, a fuller explanation of "k-distribution to v-bit accuracy", a fuller explanation of why Tempering is done, some discussion/demonstration of different seed values, a more developed Serial Correlation section, completing sections 3, 4, and 5, a conclusion, and revising existing wording/discussions throughout.

[1] Matsumoto and Nishimura (1998), pg. 1

1 – Explanation of Algorithm

We first define the parameters that are used in the algorithm. These parameters can be divided into three classes:

I - Period parameters:

- $w - integer\ (machine\ word\ size)$
- $n - integer\ (degree\ of\ recursion\ and\ initialization\ array\ length)$
- $m - integer\ such\ that\ 1 \leq m \leq n\ (middle\ term\ used\ in\ the\ recursion)$
- $r - integer\ such\ that\ 1 \leq r \leq w\ (for\ truncation)$
- $A - constant\ w \times w\ \ matrix\ with\ entries\ in\ \mathbb{F}_2$
- $\boldsymbol{a} - vector\ of\ size\ w\ (last\ row\ of\ A)$

II - Tempering parameters:

- $u - integer$
- $s - integer$
- $t - integer$
- $l - integer$
- $\boldsymbol{b} - vector\ of\ size\ w\ (bitmask)$
- $\boldsymbol{c} - vector\ of\ size\ w\ (bitmask)$

III - Initialization parameter:

- $f - integer\ (32\ bit)$

The purpose of each parameter will be clearer after reading the explanation of the algorithmic process which follows.

The MT algorithm is designed to generate a sequence of "word vectors" – w-dimensional row vectors that are elements over the field $\mathbb{F}_2 = \{0,1\}$. Word vectors are given a bold typeface throughout this paper (e.g. $\boldsymbol{x}$, $\boldsymbol{a}$). The elements of these word vectors correspond to bits of binary integers in base-2. The terms of the MT output sequence are designed to be uniform pseudorandom integers in the interval $[0, 2^w - 1]$. Each term of the sequence can then be divided by $2^w - 1$ so that they are uniform pseudorandom real numbers in the interval $[0, 1]$.

The MT algorithm itself is based on the following linear recurrence:

$$\boldsymbol{x_{k+n}} \coloneqq \boldsymbol{x_{k+m}} \oplus \left(\boldsymbol{x_k^u} \middle| \boldsymbol{x_{k+1}^l}\right) A, \quad (k = 0,1, \dots).$$

Where $\boldsymbol{x_k^u}$ is the upper $w - r$ bits of $\boldsymbol{x_k}$ and $\boldsymbol{x_{k+1}^l}$ is the lower r bits of $\boldsymbol{x_k}$. The operation $\oplus$ refers to bitwise addition modulo two, while the operation | refers to concatenating the word vector to the left of the symbol with the word vector to the right (in that order) which yields a word vector of dimension $w - r + r = w$. This concatenation can be computed using the bitwise OR operation.

The form of the matrix A is chosen so that the multiplication in the recurrence can be computed quickly. The form is:

$$A = \begin{pmatrix} 0 & I_{w-1} \\ a_{w-1} & (a_{w-2}, \dots, a_0) \end{pmatrix}$$

where I_{w-1} is the $w-1$ identity matrix and $a_{w-1}, a_{w-2}, \dots, a_0$ are elements of a word vector $\boldsymbol{a}$.

With this form $\boldsymbol{x}A$ can be computed as follows using only one or two bit operations:

$$\boldsymbol{x}A = \begin{cases} shiftright(\boldsymbol{x}), & if\ x_0 = 0 \\ shiftright(\boldsymbol{x}) \oplus \boldsymbol{a}, & if\ x_0 = 1 \end{cases}$$

where the word vector $\boldsymbol{x} = (x_{w-1}, x_{w-2}, \dots, x_0)$.

Next is the "tempering" step. This step is done so that MT sequences achieve a higher value for a criterion that is seen as one measure of the strength of a pseudorandom sequence – "k(v) distribution to v-bit accuracy." Each $\boldsymbol{x_{k+n}}$ generated by the recurrence above is multiplied by a $w \times w$ invertible matrix T (called a "tempering matrix"). The multiplication $\boldsymbol{x}T \coloneqq \boldsymbol{z}$ is achieved through the following computations carried out in sequential order:

$$\begin{aligned} \boldsymbol{y} &\coloneqq \boldsymbol{x} \oplus (\boldsymbol{x} >> u) \\ \boldsymbol{y} &\coloneqq \boldsymbol{y} \oplus ((\boldsymbol{y} << s)\ AND\ \boldsymbol{b}) \\ \boldsymbol{y} &\coloneqq \boldsymbol{y} \oplus ((\boldsymbol{y} << t)\ AND\ \boldsymbol{c}) \\ \boldsymbol{z} &\coloneqq \boldsymbol{y} \oplus (\boldsymbol{y} >> l) \end{aligned}$$

where u, s, t, and l are fixed tempering parameters and $\boldsymbol{b}$ and $\boldsymbol{c}$ are fixed word vectors. The operators $>>$ and $<<$ are the bitwise right and left shifts respectively (the number of shifts is the integer on the right side of the operator).

To implement the algorithm as described above, a sequence of n word vectors $\{\boldsymbol{x_0}, \boldsymbol{x_1}, \dots, \boldsymbol{x_{n-1}}\}$ is needed so that the recurrence can be performed to generate $\boldsymbol{x_n}$ onwards. These word vectors are generated using the following recurrence relation which only requires a single w-bit integer "seed" $\boldsymbol{x_0}$ in order to be executed:

$$\boldsymbol{x_i} \coloneqq \left(f \times \left(\boldsymbol{x_{i-1}} \oplus \left(\boldsymbol{x_{i-1}} >> (w-2)\right)\right) + i\right) AND\ (2^w - 1), \quad (i = 1,2, \dots, n-1).$$

where f is a fixed w-bit integer. The purpose of the AND operation in the recurrence is to select the last w bits of the expression to its left. This step ensures that each $\boldsymbol{x_i}$ is a w-dimensional word vector. The multiplier parameter f is taken to be 1812433253. [2]

Given the algorithm above, the question then becomes how to choose the parameters optimally so that the sequences generated by it have the most desirable qualities possible. We now comment on how the period and tempering parameters are chosen in turn:

[2] From http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/CODES/mt19937ar.c

I – Period Parameters

The period parameters are chosen such that $2^{nw-r} - 1$ is a Mersenne prime. Mersenne primes are prime numbers of the form $2^p - 1$, where p is called a "Mersenne-exponent."

The justification for this choice comes from a "general theorem of linear recurrence"[3] which states that for a linear recurring sequence produced by a linear transformation B (the main MT recurrence as described above) with characteristic polynomial $\varphi_B(t)$, "the sequence attains the maximal period $2^p - 1 = 2^{nw-r} - 1$, if and only if $\varphi_B(t)$ is primitive." Testing $\varphi_B(t)$ for primitivity (using "inverse-decimation methods") is greatly simplified when $2^p - 1$ is prime since the factors of this number do not have to be computed. Hence, choosing p to be a Mersenne-exponent allows a large period for the MT to be attained and verified relatively easily.

Fixing $w = 32$ at the machine word length for the 32-bit version of the MT (specification MT19937), we are left with choosing the parameters n and r such that $32n - r = p$ where p is a large Mersenne-exponent (so that the MT period will also be large). For a fixed p, several different choices of the parameters n and r could satisfy this equation. Hence, choosing them is not entirely trivial. The parameter n "is the order of equidistribution,"[4] and a larger equidistribution generally means a better pseudorandom generator, so it would be desirable to make this parameter as large as possible. However, n is also the number of machine words the algorithm must store so a larger n means that executing the algorithm requires more memory. Hence, there is a trade-off between making the properties of the MT as desirable as possible and adhering to practical constraints. MT19937 uses $n = 624$ indicating that this choice results in a good balance between the two competing objectives. Given p and n, r can be found using the equation above.

As in the selection of n, the selection of p is subject to a trade-off. A larger p means a larger period for the MT sequences, which is a desirable property, but it also means that the inverse-decimation method for the primitivity test to verify the period takes longer to compute (the paper mentions that this could be "several years"). The creators of the MT were able to run the test for $p = 19937$ in just two weeks. This value corresponds to a period of $2^{19937} - 1$, which is massive and hence sufficient for a wide variety of practical purposes, so $p = 19937$ is a suitable choice. Solving $32n - r = p$ with this value of p and $n = 624$ gives $r = 31$.

II – Tempering Parameters

The techniques used to select the tempering parameters are rather complex and not quite as precise as those used to select the period parameters. The tempering parameters are chosen to achieve as good of a "k-distribution property" as feasible. Theory suggests that an upper bound for the k-distribution $k(v) \leq [\frac{nw-r}{v}]$, although the MT creators were not able to find parameters which achieved this upper bound. Their search for tempering parameters (process described in

[3] Matsumoto and Nishimura (1998), pg. 9
[4] Matsumoto and Nishimura (1998), pg. 9

Matsumoto and Kurita 1994) yielded values which give a good k-distribution, but not the maximum theoretically possible. These values are provided in the table below.

Table 1: MT19937 Parameter Values

Period Parameter	Value
w	32
n	624
m	397
r	31
w	32
$\boldsymbol{a}$	2567483615
Tempering Parameter	Value
u	11
s	7
t	15
l	43
$\boldsymbol{b}$	2636928640
$\boldsymbol{c}$	4022730752
Initialization Parameter	Value
f	1812433253

2 – MATLAB Implementation

What follows is a custom MATLAB implementation of the MT algorithm. Note that the following implementation is intended to demonstrate the algorithmic process and the properties of the MT output sequences but not to be as efficient as possible. The complete MT algorithm only require 624 machine words of working memory in order to operate (older terms of the sequence are written over when possible) while, for simplicity, no significant effort to limit working memory was made when creating the implementation below. The process for determining each pseudorandom number is as specified for the MT algorithm, so the output sequences should have identical properties, although the storing of these numbers is done slightly differently.

The function takes in an input seed (used to start the initialization recurrence) as well as the desired number of pseudorandom terms. The first part of the code defines the constants for MT19937. Then, a check is done to ensure that the input seed is a 32-bit integer. If this check is passed, the initialization recursion as defined above is executed and the 623 outputs are stored along with the input seed in a "state vector." Then, the actual MT algorithm begins using this state vector. The truncation and concatenation steps are carried out first, followed by multiplication by A, and then followed by bitwise addition. The word vector produced is then

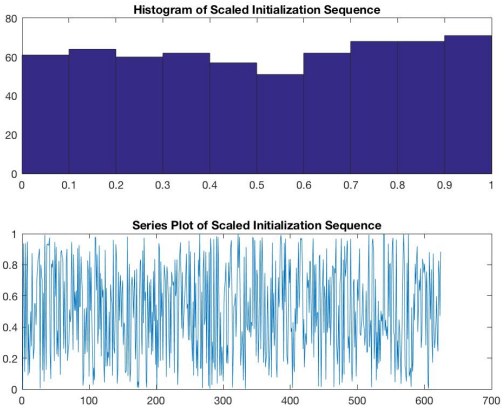
multiplied by the tempering matrix T through a series of bitwise operations as described above. Finally, this term is added to the end of the state vector and the recursion loops until the desired number of pseudorandom terms are computed. The terms produced from the MT recursion are then extracted from the state vector and transformed to real numbers in the interval [0,1] before being output.

```
function seq = myMT(seed, length)
% seq = myMT(seed, length) outputs a uniformly distributed pseudo-random
% sequence of real numbers in the interval [0,1] with the nunmber of terms
% specified through the input length. The seed input is used to start the
% initialzation recurrence. The output of this recurrence is then fed into
% the Mersenne twister recurrence from which the myMT output numbers are
% generated.
% input:
%   seed = seed value for initialization algorithm (32 bit unsigned integer)
%   length = number of terms desired for output sequence (positive integer)
% output:
%   seq = vector of pseudorandom real numbers in the interval [0,1]

% hard coded constants for MT19937
w = 32;   % machine word size
n = 624;  % degree of recursion and initialization array length
r = 31;   % for truncation step
m = 397;  % middle term used in the recursion

a = uint32(hex2dec('9908B0DF')); % last row of matrix A

u = 11;                 % tempering constant
s = 7;                  % tempering constant
b = uint32(2636928640); % tempering bitmask
t = 15;                 % tempering constant
c = uint32(4022730752); % tempering bitmask
l = 43;                 % tempering constant

f = uint64(1812433253); % initialization constant

% ensures seed is a 32 bit integer
if ((seed <= 0) | (seed >= (2^w - 1)) | ((floor(seed) - seed) ~= 0) |
((floor(seed) - seed) ~= 0))
    error('Seed must be a 32 bit integer')
end


% begin initialization
mask = uint64((2^w) - 1); % defining truncation mask
state = [uint64(seed)];   % putting chosen seed in state vector

% executing initialization recursion
for i = 1:(n-1)
    x_i = (f * (bitxor(state(i), (bitshift(state(i), -(w - 2), 'uint64')))))
+ i;
    x_i = bitand(x_i, mask); % lowest w bits
    state = [state x_i];
end

state = uint32(state); % converting to 32 bit integers for next step
% end initialization
```

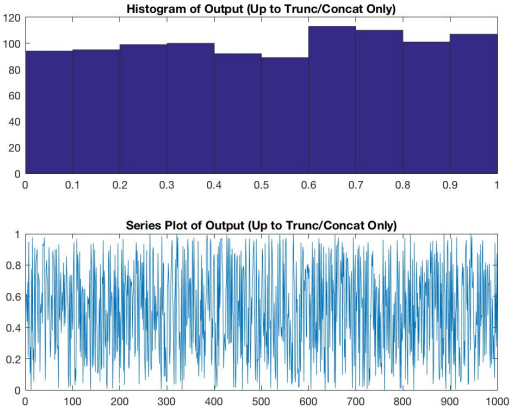
```
% start MT algorithm
% defining truncation masks
mask_u = uint32((2^(w) - 1) - (2^(r) - 1));
mask_l = uint32(2^(r) - 1);

% loop to generate specified number of terms
for i = 1:length
    % truncation and concatenation step
    y = bitor((bitand(state(i), mask_u)),(bitand(state(i+1),mask_l)));

    % executing recursion
    % multiplying by A
    yA = bitshift(y, -1); % yA if lowest bit of y is 0
    if (mod(y,2) ~= 0)    % yA if lowest bit of y is 1
        yA = bitxor(yA, a);
    end

    x_km = state(m + i - 1); % middle term
    x_i = bitxor(x_km, yA);  % bitwise addition

    % tempering steps
    y = x_i;
    y = bitxor(y, bitshift(y, -u));
    y = bitxor(y, bitand(bitshift(y, s),b));
    y = bitxor(y, bitand(bitshift(y, t),c));
    y = bitxor(y, bitshift(y, -l));

    state = [state y]; % adding term to state vector
end


output = state(625:(624 + length)); % removing initialization terms

seq = double(output) / ((2^w) - 1); % transforming output to [0,1]
end
```

The following images show what the output looks like if the code is only executed up to certain stage for a seed value of 481516 and a desired sequence length of 1000:

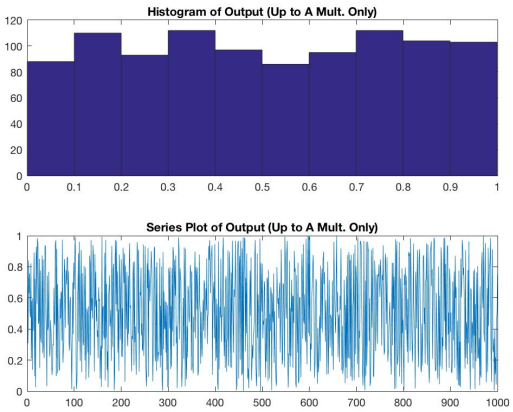
Stage 1 – Initialization Sequence (624 terms)



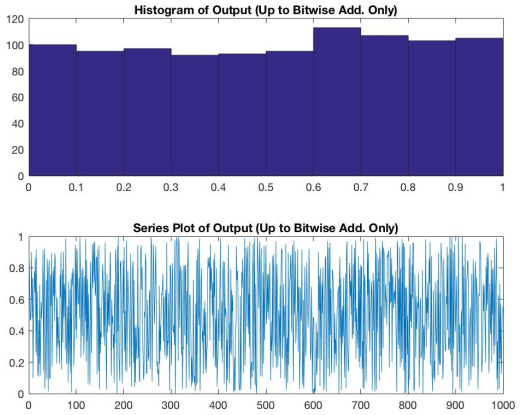
Stage 2 –Trunc., Conc. (1000 terms)



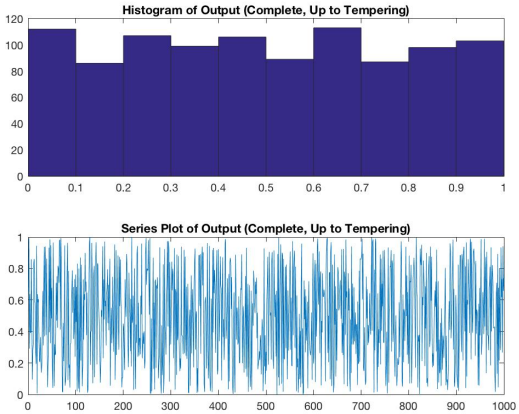
Stage 3 – Multiplying by A (1000 terms)



Stage 4 – Bitwise Add. (1000 terms)



Stage 5 – Tempering (1000 terms)



Just from looking at what the sequence distribution and plot looks like at each stage, it is difficult to appreciate what the purpose of each step is. All of the series plots look like what random, white noise should look like and the sequence terms at each stage appear to be fairly uniformly distributed in the interval [0,1] as intended. These descriptions even apply to the initialization sequence – the sequence before any of the MT-specific steps are executed.

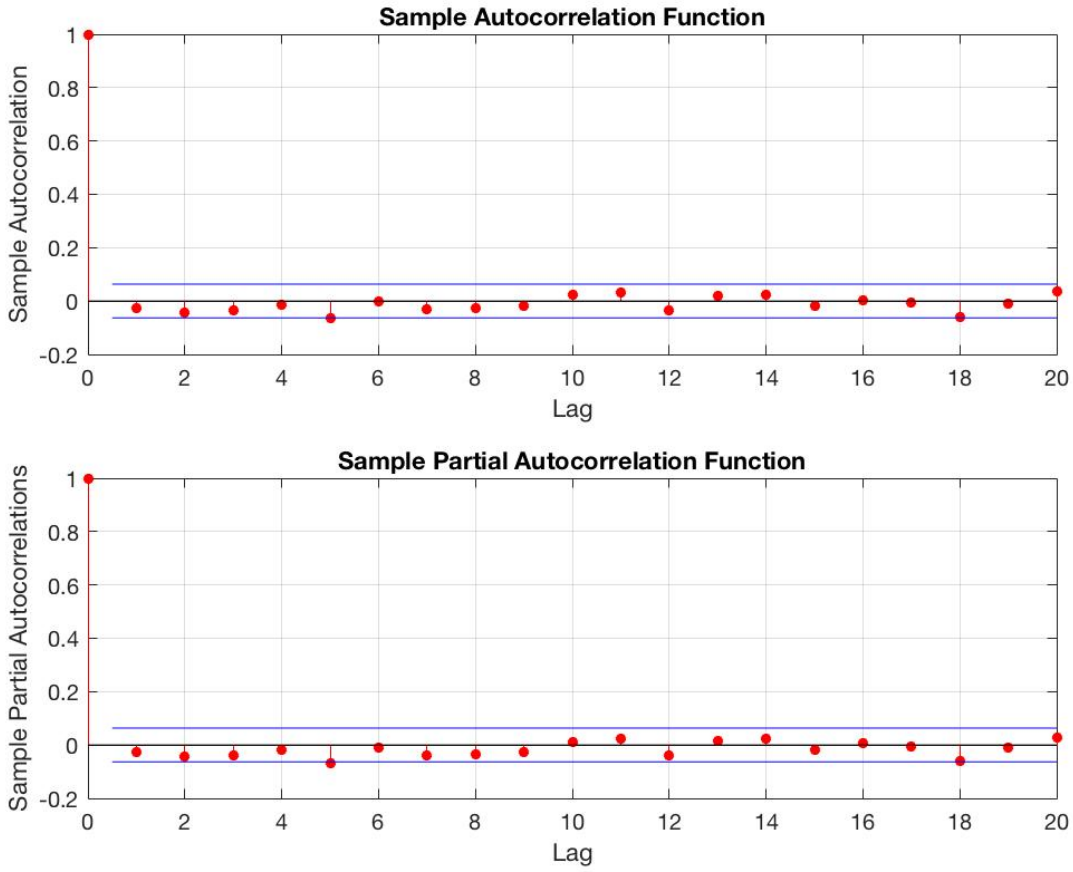
These similarities highlight just how subtle the differences between RNGs are. While it may not be visually apparent what exactly the value of each step is in making the output sequence "more random," each step helps to ensure that the sequence satisfies some statistical and mathematical criteria that have been shown, through theory and practice, to be good measures of randomness.

3 – Evaluating the Randomness of MT Sequences

I – Serial Correlation

Serial correlation is a fairly intuitive statistical concept which can be used to evaluate the randomness of a sequence. It refers to the degree by which current values of a sequence are correlated with previous values of the same sequence. For example, the strength of the correlation between all the terms of a sequence and all the terms two-terms behind them can be evaluated. If there is sufficient evidence for such a correlation, then the sequence is not truly random since a future value of the sequence can be predicted (albeit perhaps not perfectly) using values of the sequence which have already been observed.

A correlogram is a chart typically used to evaluate serial correlation. It visualizes computed autocorrelation coefficients for terms of the sequence various displacements away from each other (the correlation between terms of the sequence that displacement apart) as well as the partial autocorrelation coefficients, which measures the correlation between terms of the sequence specified displacements apart after controlling for correlations between the intermediate terms. A correlogram for an MT sequence (seed = 48118766, length = 1000) is provided below:



The blue lines in the plot denote the 95% Bartlett bands – under the null hypothesis of the sequence truly being random white noise (no serial correlation), it is expected that the autocorrelation and partial autocorrelation coefficients will exceed these bands only 5% of the time. The vast majority of the calculated coefficients fall within these bands in both plots, indicating that there is no evidence to reject the null hypothesis of the sequence exhibiting no serial correlation.

A Ljung-Box test can be used to provide a single numerical test statistic to test the null hypothesis.

There are more mathematically rigorous evaluations of randomness that the MT passes:

II – k-Distributed to v Bit Accuracy

III – Characteristic Polynomial of Linear Transformation with Many Terms

IV – Diehard Tests

4 – MT Pros and Cons

Pros:
I – Fast Computation
II – Small Working Memory
III – Large Period
…

Cons:
I – Not Good for Cryptography
II – Finite, Fixed Period (as opposed to natural sources of randomness)
…

5 – Discussion of Good Applications for MT and Bad Applications