Introduction to Mersenne Twister Pseudorandom number generator

Qiao Zhou

June 30, 2016

Introduction

In this introductory article, I explain the algorithm behind the Mersenne Twister (MT) Pseudorandom number generator (PRNG) designed by Nishimura and Matsumoto. The most common implementation of it uses an internal state of 624 words (integers) with word-length 32 bits. For a k-bit word length, MT generates uniform draws in the range of $[0, 2^k - 1]$, which can be translated to a [0,1) ranged uniformly distributed real number. As the name suggests, its period is given by a Mersenne Prime in the form of $M_n = 2^n - 1, n \in N$, which is $2^{19937} - 1$ in the MT19937 version. According to Wikipedia, the Mersenne Twister algorithm is based on a matrix linear recurrence over a finite binary field. The algorithm is a twisted generalised feedback shift register of rational normal form, with state bit reflection and tempering. Concretely, the algorithm works as follows:

1. An integer array of length n (n=624) is created to store the internal state of the MT generator;
2. A w-bit (w=32) word-length seed is input to the generator for state vector initialization. This seed is supplied to $x_0, ..., x_{n-1}$ by setting x_0 to the seed value and subsequent x values through a recursion expression. This forms the internal state for MT comprised of n values of w bits each. The first random realization the algorithm then generates is based on x_n.
3. Next, twisting is applied to the state and the next n values from x_i is generated to move the generator to the next state. Note that the first state (state 0) will not be used to generate any number, and the first random number is generated based on state 1.
4. Next, the number extraction function (i.e., method genrandint32() as denoted in the original C implementation) will be called. The algorithm will check for the current index i in the length-n x_i state array, and twist() will be called every n numbers. Next, tempering transformation is applied before extracting the next number. After tempering, the random number is returned and the index is incremented.

The formal technical details can be found in their original paper. The original C implementation can be retrieved from mt19937ar.c.

Given the language of choice is Python, we examine the implementation details of MT under this language. As documented, Python's basic randomization routine random() uses MT as the core generator, which produces 53-bit precision floats and has a period of $2^{19937} - 1$ before repeating itself, and 623-dimensional equidistribution property is assured. The underlying fast implementation in C is based on the 2002 version of MT19937. MT is extensively tested and passes numerous statistical randomness tests, including the Diehard tests.

Concretely, 2002-version mt19937ar.c has versions [0,1],[0,1),[0,1),(0,1) (32-bit precision), and [0,1) with 53-bit precision. For the 32-bit precision version, the discretization is 2^{-32} (i.e., $1/N$, where $N = 2^{32}$). While under the IEEE 754 double-precision binary floating-point format (64-bit floating point value), the number representation has a significand precision of 53 bits. The discretization is thus 2^{-53} (i.e., $1/N$, where $N = 2^{53}$) , and MT is able to create double precision values in the closed interval $[0, 1 - 2^{-53}]$ with increment 2^{-53}. However, this is not the same as the recurrence period $N^{'}$, which is $2^{19937} - 1$ in MT19937 version. One may wrongly argue that given 32-bit random integers can only range from 0 to $2^{32} - 1$, its recurrence period can only be as long as 2^{32} for the 32 bit precision version. However, one should make the distinction between the output random number and the internal state of the PRNG as they are different concepts. The number can be the same after 2^32 generations, but the internal state will only repeat with recurrence period $2^{19937} - 1$. In fact, MT internally stores the large state with $19968 = 32 \times 624$ bits, which is sufficient for the period of $2^{19937} - 1$. By default, the 53-bit double-precision version is used by Python, which is achieved by concatenating two 32-bit random integers (with assigned weights) and them dividing them by 2^{53} to result in a normalized random number on $[0, 1)$.