A fast random number generator for stochastic simulations

Anthony J. C. Ladd

Department of Chemical Engineering, University of Florida, Gainesville, Florida 32611-6005, USA

Abstract

A discrete random number (DRN) generator for stochastic differential equations is proposed. The generator has exactly 8 states and thus 10 DRN's can be obtained from a single 32-bit random variable. This is advantageous when large numbers of DRN's are needed, as for example in fluctuating lattice-Boltzmann models. The moments of the discrete distribution match those of a Gaussian distribution (zero mean and unit variance) up to 5th-order. Numerical tests show that satisfactory statistical properties can be obtained with several 32-bit pseudo random number (PRN) generators.

Key words: Random Number Generators
PACS: 05.10.Gg,47.11.Qr,47.57.-s

1. Introduction

Stochastic simulations of a time-dependent Fokker-Planck equation require computationally efficient algorithms for generating random numbers with an appropriate probability distribution. Although the transition probability itself is a normal distribution, Gaussian random numbers are unsuitable for numerical simulation, because the occasional large displacement causes significant numerical errors [1, 2]. The solution is to generate random numbers from a bounded distribution that is constructed to approximate a normal distribution after a number of steps [1, 3]. Weak convergence of order n can be achieved by ensuring that moments up to order $2n$ are reproduced correctly [3].

A standard random number generator (RNG) samples a bounded uniform distribution, but this can only match the variance of the normal distribution, whereas for a second-order stochastic simulation the correct fourth moment is required as well. Discrete distributions of random numbers are a simple way to ensure a bounded distribution, and can be constructed to match an arbitrary number of cumulants. Two such distributions are

$$P(x) = \frac{1}{12}\delta(x+2) + \frac{1}{6}\delta(x+1) + \frac{1}{3}\delta(x) + \frac{1}{6}\delta(x-1) + \frac{1}{12}\delta(x-2) \quad (1)$$

$$P(x) = \frac{1}{6}\delta(x+\sqrt{3}) + \frac{2}{3}\delta(x) + \frac{1}{6}\delta(x-\sqrt{3}). \quad (2)$$

Both these distributions match the cumulants, $\langle x^n \rangle$, of a normal distribution with unit variance up to the 5th order ($n = 5$). Note that a short sequence of binary (±1) random numbers would not have the correct fourth moment.

The range (minimum to maximum value) of the distribution in Eq. (2) is slightly smaller than in Eq. 1, $[-1.732, 1.732]$ as opposed to $[-2, 2]$, which gives it a slight preference. A typical implementation of Eq. (1) or (2) is to scale a random number R, drawn from a uniform distribution, to the appropriate range, $0 \leq R < 12$ for Eq. (1) or $0 \leq R < 6$ for Eq. (2). The scaled value of R is then truncated to an integer and used in a table lookup for x.

In some applications, generation of random numbers can be a computational bottleneck. For example, the fluctuating lattice-Boltzmann model [4] needs up to 15 random variables to update a single lattice site [5, 6], which results in a significant overhead [7]. Here we describe an efficient method to calculate 10 DRN's from a single 32 bit variable, reducing the computational cost of the random number generation by an order of magnitude. The method will be useful in stochastic simulations where random number generation is a significant part of the overall computational effort.

2. A 3-bit RNG

An attractive property of discrete random numbers is that they require a small number of random bits, and thus, in principle, it should be possible to obtain several DRN's from a single 32-bit variable. However, the most commonly used distributions, Eqs. (1)–(2) [1, 3, 8], do not exactly match to a fixed number of bits, because the number of states (12 and 6) is not a power of two. Thus, although several DRN's can be obtained from a single 32-bit random variable, there is little computational advantage in doing so because the overhead is so high. We find that a single DRN from the 6-state distribution takes about 10 clock cycles compared with 20 clock cycles for a 32-bit random variable. However, a significant speed up can be obtained by choosing a discrete probability distribution with exactly 2^m states, where m is the number of bits:

$$P(x) = \frac{1}{2^m} \sum_{i=1}^{2^{m-1}} \delta(x + a_i) + \delta(x - a_i), \tag{3}$$

In this case we can generate a DRN in about 2 clock cycles; results of the timings are in Table 1.

For $m = 2$ there are no real values of a_1 and a_2 such that both the second and fourth moments of $P(x)$ match the normal distribution, but with $m = 3$ the moment conditions

$$\sum_{i=1}^{4} a_i^2 = 4, \tag{4}$$

$$\sum_{i=1}^{4} a_i^4 = 12, \tag{5}$$

can be satisfied by various sets of constants $a_1 - a_4$. It is not possible to match the 6th cumulant with an 8-state generator; again the coefficients turn out to be complex.

If the constants are arranged in ascending order, $a_1 \leq a_2 \leq a_3 \leq a_4$, Eq. (5) implies that the largest constant, a_4, is bounded in the range $3^{1/4} \leq a_4 \leq 12^{1/4}$; this range of a_4 can also satisfy Eq. (4). The specific set of constants can be chosen to minimize the range of the random numbers; in other words minimizing a_4 with respect to variations in a_1 and a_2 while satisfying Eqs. (4)–(5). The solution is $a_1 = a_2 = 0$, $a_3 = a_- = (2 - \sqrt{2})^{1/2}$, and $a_4 = a_+ = (2 + \sqrt{2})^{1/2}$. The proposed 3-bit discrete distribution is then

$$P(x) = \frac{1}{8}\delta(x + a_+) + \frac{1}{8}\delta(x + a_-) + \frac{1}{2}\delta(x) + \frac{1}{8}\delta(x - a_-) + \frac{1}{8}\delta(x - a_+), \tag{6}$$

with a range of $[-1.848, 1.848]$ that is only slightly larger than Eq. (2), $[-1.732, 1.732]$.

Pseudo random numbers sampled from Eq. (6) can be generated by the following algorithm:

1. Generate a 32-bit integer random number, R

2. Right shift by two bits ($R \gg 2$) to remove unneeded trailing bits (typically the least random).
3. Generate the index to the lookup table of $\pm a_i$ by R & 7
4. Right shift R by three bits ($R \gg 3$)
5. Repeat steps 3 and 4 up to 10 times, until the word is exhausted.

Random bits were generated in 32-bit words, using random number generators (RNG's) from the GSL library [9]. We tested two "Generalized Feedback Shift Register" algorithms (MT19937, GFSR4 [10]), a "Multiplicative Recursive Generator" (TAUS2), and a "Linear Congruential" generator (RAND48), which is also part of the Standard C Library. Each RNG was first tested with the "DIEHARD" suite [11, 12, 13] of statistical tests for randomness. The generators MT19937, TAUS2, and GFSR4 are considered to be among the best available RNG's and performed well in the DIEHARD tests. RAND48, which is a standard UNIX RNG was also satisfactory although it consistently failed two of the tests (DNA and OQSO). Note that it automatically discards the trailing 16 bits (out of 48 generated) to reduce serial correlations in the low-order bits.

Additional tests were carried out to detect correlations in the bit sequences, using a variant of the n-block test [14]. A single sequence of N 3-bit random numbers was generated and the cumulants $\langle x^2(t)\rangle$ and $\langle x^4(t)\rangle$ were measured as a function of the number of steps t, up to a maximum t_{max}. Different sequences were run for $t_{max} = 100$ in steps of 1 and $t_{max} = 10^4$ in steps of 100. The total number of DRN's generated was $N = 10^4 t_{max}$; *i.e.* 10^6 for the short sequence and 10^8 for the long sequence. The process was repeated with a total of 10 randomly generated seeds and the distribution of the cumulants was compared both visually and quantitatively with the statistically expected results. A sample set of data for the MT19937 RNG is shown in Fig. 1. The distributions of cumulants show no obvious correlations for sequences of up to 10^4 steps. A visual inspection of the 1000 values of Δ_2 or Δ_4 in each graph shows them to be more or less uniformly distributed, with the expected variance,

$$\sigma^2_{\langle x^{2n}(t)\rangle} = \frac{\langle x^{4n}(t)\rangle - \langle x^{2n}(t)\rangle^2}{N t^{n-1}}. \tag{7}$$

The random number generators TAUS2, GFSR4, and RAND48 show similar behavior to MT19937.

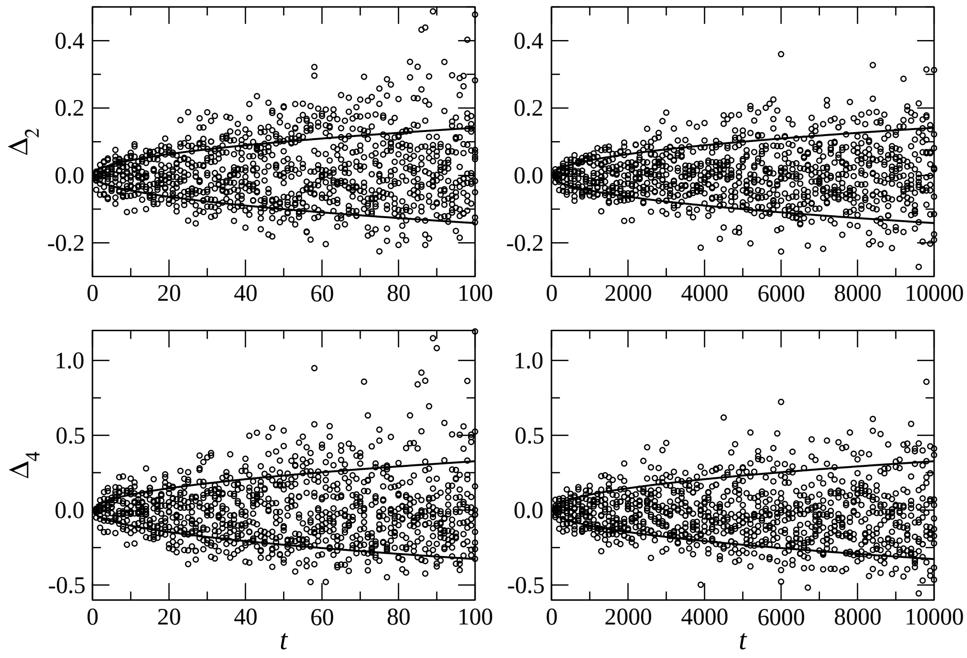

Figure 1: Deviations in the second and fourth cumulants, Δ_2 and Δ_4, of a random walk of 10^6 (left figures) and 10^8 (right figures) steps; the MT19937 RNG was used for these tests. The walk was divided up into blocks of t steps and the cumulants calculated from the displacement at the end of each block. The upper figures show the results for the Δ_2 and the lower figures the results for Δ_4. The regions bounded by a single standard deviation in Δ_n, are shown by the solid lines.

RNG	RN	RN8	RN6
MT19937	8.0	0.8	4.3
TAUS2	6.0	0.6	4.0
GFSR4	4.9	0.5	3.9
RAND48	6.9	0.7	4.3

Table 1: Timings for random number generation measured on an Intel Xeon E5410 (2.33GHz) processor; all times are in nanoseconds per random number. The column RN is the baseline for a 32-bit random number; RN8 is for a single 3-bit DRN, timing the code fragment shown in Table 2 for the 8-state distribution (a) ; RN6 is the corresponding timing for the 6-state distribution (b).

Timings for random number generation are reported in Table 1. As a baseline for each RNG, the time to generate a 32-bit random variable was determined from a sequence of 10^9 calls. Discrete random numbers can be generated more efficiently from the 8-state RNG, Eq. (6) than generators derived from Eqs. (1)–(2) because the outcome is entirely predictable. A simple application to a one-dimensional random walk is illustrated by the code fragments shown in Table 2. Timings of the code fragments in Table 2 shows that the 8-state RNG can be implemented with essentially zero overhead whereas the 6-state RNG takes 5-6 times longer. A DRN sampled from Eq. (6) can be generated in about 2 clock cycles (see Table 1), but if the number of states does not match the number of bits out of range states must be rejected in order to obtain the correct distribution. Thus it cannot be guaranteed that a 32-bit word will generate 10 3-bit DRN's, or, in an unfortunate case, even a single DRN. The additional code to count the in-range DRN's and to determine when to generate a new 32-bit PRN (Table 2b) accounts for the substantial overhead.

```
w=0;
for(i=0;i<N/10;i++){
   u=gsl_rng_get(r)>>2;
   for(k=0;k<10;k++){
      w+=table[u&7];
       u>>=3;}}
```

(a)

```
w=i=k=0;
while(i<N){
   if(k==0){
       u=gsl_rng_get(r)>>2;
       k=10;}
       if((u&7)<6){
          w+=table[u&7];i++;}
       u>>=3;k--;}
```

(b)

Table 2: Code fragments illustrating the implementation of an 8-state RNG (a) and a 6-state RNG (b). The timings for these code fragments are given in Table 1.

In this brief communication a simple algorithm to generate sequences of discrete random numbers has been proposed. By using a probability distribution with 2^m states ($m = 3$ in this case), several DRN's can be obtained from a single 32-bit random variable. The optimized code, producing 10 random numbers at a time, is more than an order of magnitude faster than the standard implementation. It is also a factor of 5–6 faster than discrete generators where the number of states does not match to a power of 2.

Acknowledgments

This work was supported by the National Science Foundation (CTS-0505929).

References

[1] Dünweg, B. and Paul, W., Int. J. Mod. Phys. C **2** (1991) 817.

[2] Öttinger, H. C., *Stochastic Processes in Polymeric Fluids*, Springer-Verlag, 1996.

[3] Kloeden, P. E. and Platen, E., Numerical solution of stochastic differential equations, in *Applications of Mathematics 23*, Springer, 1993.

[4] Ladd, A. J. C., Phys. Rev. Lett. **70** (1993) 1339.

[5] Adhikari, R., Stratford, K., Cates, M. E., and Wagner, A. J., Europhys. Lett. **3** (2005) 473.

[6] Dünweg, B. and Ladd, A. J. C., Adv. Polym. Sci. **221** (2009) 89.

[7] Dünweg, B., Schiller, U. D., and Ladd, A. J. C., Comput. Phys. Commun. (2009), In Press.

[8] Buchmann, F. M. and Petersen, W. P., BIT Numerical Mathematics **43** (2003) 519.

[9] Free Software Foundation, GSL - GNU Scientific Library - Version 1.12, http://www.gnu.org/software/gsl/.

[10] Ziff, R. M., Comput. Phys. **12** (1998) 385.

[11] Marsaglia, G., Diehard: Battery of Tests of Randomness, http://www.stat.fsu.edu/pub/diehard/.

[12] Marsaglia, G., J. Stat. Soft. **14** (2005).

[13] Brown, R. G., Dieharder: A Random Number Test Suite - Version 2.28.1, http://www.phy.duke.edu/∼rgb/General/dieharder.php.

[14] Vattulainen, I., Ala-Nissila, T., and Kankaala, K., Phys. Rev. E **52** (1995) 3205.