Improvement algorithm of random numbers generators used intensively on simulation of stochastic processes

Edisel Navas Conyedo¹*, Carlos Torres Pupo¹, Giuseppe B. Suffritti², Jorge Gulín González³

¹ Grupo de Matemática y Física Computacional. Universidad de las Ciencias Informáticas, Carretera a San Antonio de los Baños, km 2½, Torrens, Boyeros, La Habana, Cuba. CP.: 19370
² Dipartimento di Chimica, Università degli Studi di Sassari, and Consorzio Ineruniversitario Nazionale per la Scienza e Tecnologia dei Materiali (INSTM), Unità di ricerca di Sassari, Via Vienna, 2, I-07100 Sassari, Italy
³ Dirección de Investigaciones. Universidad de las Ciencias Informáticas, Carretera a San Antonio de los Baños, km 2½, Torrens, Boyeros, La Habana, Cuba. CP.: 19370

*Autor para la correspondencia: enavas@uci.cu

Resumen

La elección de algoritmos eficaces y eficientes para la generación de números aleatorios es un problema clave en simulaciones de procesos estocásticos; siendo la difusión uno de ellos. El modelo del caminante aleatorio y la ecuación dinámica del Langevin son las formas más sencillas para el estudio computacional de la difusión. Ambos modelos, donde las partículas no interactúan y se mueven libremente, se utilizan para probar la calidad de los generadores de números aleatorios que se van a utilizar en simulaciones computacionales más complejas. En principio, la generación de números aleatorios a través de ordenadores es imposible porque los ordenadores funcionan a través de algoritmos deterministas, sin embargo, se pueden utilizar generadores deterministas cuyas secuencias de números que para las aplicaciones prácticas podrían considerarse aleatoria. En el presente trabajo se presenta una combinación de los generadores de números aleatorios reportados por Numerical Recipes y GNU Scientific Library...
con el que utiliza el sistema operativo Linux (basado en hardware). Los resultados obtenidos utilizando nuestra herramienta computacional permite mejorar las características aleatorias de los generadores en estudio, con la mejora subsiguiente de la exactitud y la eficiencia de simulaciones computacionales de los procesos estocásticos.

**Palabras clave:** Difusión, la ecuación dinámica de Langevin, generador de números aleatorios, paseo aleatorio, procesos estocásticos.

**Abstract**

Choice of effective and efficient algorithms for generation of random numbers is a key problem in simulations of stochastic processes; diffusion among them. The random walk model and the Langevin’s dynamical equation are the simplest ways to study computationally the diffusion. Both models, in the non-interacting free particles approximation, are used to test the quality of the random number generators which will be used in more complex computational simulations. In principle, generation of random numbers via computers is impossible because computers work through determinist algorithms; however, there are determinist generators which generate sequences of numbers that for practical applications could be considered random. In the present paper we present a improve algorithm random number generator obtained from a combination of those reported by Numerical Recipes, GNU Scientific Library, and that used by Linux operating system (based on hardware). The results obtained using our computational tool allows to improve the random characteristics of any pseudorandom generator, and the subsequent improving of the accuracy and efficiency of computational simulations of stochastic processes.

**Keywords:** Diffusion, random walk, langevin’s dynamical equation, random number generators, stochastic processes.

**PACS:** 05.40.-a, 05.40.Fb, 05.10.Gg.

**Introduction**

Random Number Generator (RNG) is a key point for the simulation of stochastic processes, particularly when the Monte Carlo method is used. Diffusion is among most common phenomena in nature; moreover it is suitable to be computationally studied. The simplest way to describe the diffusion is by using the random walk model (RWM) and the Langevin’s dynamical equation (LDE). Both models, in the non-interacting free particles approximation, are used to test the quality of the random number generators (Janke, 2002; Passerat-Palmbach, 2013). In the first model, RNG is used to simulate the molecular displacement by jumping; in the second one, to simulate the force on each particle, when the thermal noise is considered. The results obtained by RWM and LDE allow to test the quality of the RNGs which will be used in more complex computational simulations.
Generating random numbers by using computers is, in principle, unmanageable, because computers work with deterministic algorithms. However, there are deterministic algorithms that produce sequences of random numbers which for practical proposes can be considered random; these algorithms are named pseudorandom. For many applications some statistical and empirical test are consider for select a RNG with desirable characteristics, the quality of the generators is often estimated by the number of statistics or empirical test that are satisfactory resolved; however, it is possible the case in which a good selection of a RNG taken into account the behavior in many tests, do not produce good results in computational simulations (Sánchez-Pajares and Yuste, 2002; David P. Rosin, 2013; Yates and Klingbeil, 2013). Large simulation processes need good accuracy of results and low run time consumption as criteria of RNG selection.

In this paper, we study the behavior of the solutions in case of diffusion of free non interacting particles by using the RWM and LDE; to generate random numbers we use some of the most popular RNG, they are: the reported in Numerical Recipes text (NR) (Press, et al., 2007), GNU Scientific Library (Galassi, et al., 2011), and finally, that used by the operating system LINUX, (Love, 2010) based on hardware. Here, we propose a new algorithm to improve the random characteristic of any pseudorandom generator, and subsequently improving the accuracy and efficiency of computational simulations of stochastic processes.

Computational methodology

In the selection of a generator of pseudorandom numbers (PRNG) to be used in simulations it is desirable the existence of good random properties. The last should be undertaken as an independent sequence of random numbers with the same probability of occurrence. Besides they have a long period and computational efficiency taking into account: time of calculation, used memory, and portability (David, 2013).

The computational algorithms for generating a pseudorandom numbers can be classified as: one-step (Park and Miller, 1988; Kirkpatrick and Stoll, 1981), many recurrence steps (Deng, et al., 2008), shift registers, linear feedback shift registers and non-linear generators (González and Pino, 1999). Overall, all the PRNGs generate a sequence depending on starting value called seed and, consequently, whenever they are initialized with a same value the sequence is repeated.

To improve the characteristics of a random generator it uses several mechanisms as PRNGs combination of lower quality, one example is the method of dual randomness in which one PRNG generate a vector of values
\( \mathbf{V} = (V_1, V_2, \cdots, V_s) \) and other PRNG selects it while their values are updated obtaining a PRNG that has a longer period.

A detailed explanation of all of these algorithms can be found in the appendix of the book of a reference (Fishman, 1996).

The operating system GNU/Linux was the first to implement a PRNG on system kernel level, this implementation uses the ambiental noise from various devices such as network card, video, mouse, keyboard, read and write instructions in memory and disks that are combined and transformed using secure hash functions yielding a sequence of random bits that can be accessed by /dev/random and /dev/urandom devices. These devices are differed because /dev/random block read operations until the system has enough entropy in its noise sources to ensure the randomness of the bits to read, this is very positive when we get results with a high degree of randomness regardless of the computing time; /dev/urandom is not blocked allowing a faster response time at the expense of obtaining random numbers with less random properties (Gutterman et al., 2006; Love, 2010). The implementation of this PRNG is very simple follow a algorithms represented on a function GetUrand to obtain a uniform generator on \([0;1]\) interval, that depends of the number \(N\) of random bits that was read.

**Function GetUrand**

Read N bits from /dev/urandom file and set on X var

Return double \(X/(2^N-1)\)

End Function GetUrand

The use of the generator core operating system GNU/Linux on the other hand has better features when the time between calls to the generator is sufficient enough to cause cumulative noise from the devices, is therefore not appropriate to use this generator for simulations with intensive random number generator calls, being necessary to design a strategy to mix the use of this generator with some others algorithms. One of the major deficiencies that have the PRNG is its sequences are determined by the random seed, this may be a mechanism that can be used to improve the characteristics of the PRNG if after a set of calls, optimized in correspondence with the computational architecture, the seed is restart using other PRNG of operating system, in each case by optimizing the number of iterations for which there is sufficient accumulated environmental noise, this method breaks the sequence of decreasing PRNG long-term correlation between the values of the sequence and increasing the random statistical properties. The algorithms to use this mechanism of improvements that we propose can use any PRNG, represented as \(\text{Rand}\) function, and depend of the number \(M\) of iterations to do the reseed as show on function GetBetRand. For the \(M\) estimation the relation between the mean speed of call of \(\text{Rand}\) and \(\text{GetUrand}\) are used as
that guaranty no a great impacts on run time calls and enough accumulative entropy on /dev/urandom device.

Function GetBetRand
Count I
IF (I equal to M) then
Read N bits from /dev/urandom file and set on Seed var
Reseed Rand with Seed var
I=0
else
I=I+1
EndIF

Return call of Rand
End Function GetBetRand

Basic models for the simulation of stochastic processes.
Diffusive processes are stochastic processes whose behavior can be simply simulated through the random walker model (RW) and Langevin dynamics equation (DL).

The RW model is a basic model for the description of Brownian motion developed by Einstein (Einstein, 1956), this is consider a system of independent particles each time a small interval \( \tau \) perform a jump in its value \( l \) position, so each particle has a course based on the number of intervals considered as \( \bar{R} = \sum_{i=1}^{N} \Delta \vec{r}_i \) whose rms value is:

\[
< R^2 > = Nl^2 + \sum_{i \neq j}^{N} \Delta \vec{r}_i \cdot \Delta \vec{r}_j = Nl^2 + R_m^2 = R_{2m}
\]

where \( R_m = < |\vec{R}| > \) and \( N \) is the iterations number, if the displacements are uncorrelated and all particles initially are in the origin marks the central limit theorem and \( R_m \to 0 \) and thus \( R_{2m} \to Nl^2 \).

In practice, a computer simulation model RW is to build a system \( S \) which particles move with displacements
\[
\Delta \tilde{r}^2 = \frac{(\text{rand}(-1, 1)_1, \text{rand}(-1, 1)_2, \text{rand}(-1, 1)_3) l}{\sqrt{\text{rand}(-1, 1)_1^2 + \text{rand}(-1, 1)_2^2 + \text{rand}(-1, 1)_3^2}}
\]

where \(\text{rand}(-1, 1)_1, 2, 3\) are calls to a uniform selected PRNG on \([-1, 1]\) interval. It is known that the RW model is sensitive to long periodicity and correlation between the displacements generated, which if you do not have enough randomness can lead to absurd results, so it can be used as a tool to test the qualities of a RNG (Proykova, 2000; Sánchez-Pajares and Yuste, 2002; Janke, 2002).

The DL model is a simplified approach to describe the dynamics of a molecular system, this takes into account the interaction of each molecule with the environment in which broadcasts which is treated as a viscous medium and includes a term corresponding to the thermal agitation in the case of particles that do not interact with each other, it has the form:

\[
m \ddot{\vec{R}} = -\gamma \dot{\vec{R}} + \sqrt{6\gamma k_B T} \vec{\eta}(t)
\]

where \(M\) are the particles mass, \(\gamma\) is the viscosity coefficient, \(T\) is the temperature and \(\vec{\eta}(t) = (\text{rand}(-1, 1)_1, \text{rand}(-1, 1)_2, \text{rand}(-1, 1)_3)\)

is the noise function (Schlick, 2002). When rms is calculating this gives:

\[
R_{2m}(t) = \frac{2k_B T}{\gamma} \left( t + \frac{m}{\gamma} \ln(1 - \frac{\gamma t}{m}) \right)
\]

for \(t \gg m\gamma^{-1}\) has linear behavior \(R_{2m}(t) \approx \frac{2k_B T}{\gamma} \left( t - \frac{m}{\gamma} \right)\).

Results and discussion

To test the influence of PRNG on diffusion simulation of noninteracting free particles we have selected a set of 20 PRNGs where 7 correspond to the distribution of NR in its third version, 10 to a package of GSL in the version 1.15, 1 using the Linux kernel generator whose implementation for C that was shown before in section:2 and 2 whose corresponding with a mixture implementation using a one RNG of each distributions with seeds changing every 1024 calls by Linux kernel PRNG those are shown in table 1.
Table 1. PRNGs used on simulations.

<table>
<thead>
<tr>
<th>ID</th>
<th>Key</th>
<th>Distribution</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ran</td>
<td></td>
<td>NUMERICAL RECIPES</td>
</tr>
<tr>
<td>2</td>
<td>Ranq1</td>
<td>Numerical</td>
<td>The Art of Scientific Computing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Recipes</td>
<td>Third Edition Chapter 7.1 (Press et al., 2007)</td>
</tr>
<tr>
<td>3</td>
<td>Ranq2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Ranhash</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Ranbyte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Ranfib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Ranlim32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>gsl_rng_mt19937</td>
<td></td>
<td>(Matsumoto and Nishimura, 1998)</td>
</tr>
<tr>
<td>9</td>
<td>gsl_rng_ranld1</td>
<td></td>
<td>(Lüscher, 1994),</td>
</tr>
<tr>
<td>10</td>
<td>gsl_rng_ranld2</td>
<td></td>
<td>(James, 1994)</td>
</tr>
<tr>
<td>11</td>
<td>gsl_rng_cmrg</td>
<td></td>
<td>(L’Ecuyer, 1996a)</td>
</tr>
<tr>
<td>12</td>
<td>gsl_rng_mrg</td>
<td></td>
<td>(L’Ecuyer et al., 1993)</td>
</tr>
<tr>
<td>13</td>
<td>gsl_rng_taus</td>
<td></td>
<td>(L’Ecuyer, 1996b)</td>
</tr>
<tr>
<td>14</td>
<td>gsl_rng_taus2</td>
<td></td>
<td>(L’Ecuyer, 1999)</td>
</tr>
<tr>
<td>15</td>
<td>gsl_rng_gfsr4</td>
<td></td>
<td>(Ziff, 1998)</td>
</tr>
<tr>
<td>16</td>
<td>gsl_rng_r250</td>
<td></td>
<td>(Kirkpatrick and Stoll, 1981)</td>
</tr>
<tr>
<td>17</td>
<td>gsl_rng_minstd</td>
<td></td>
<td>(Park and Miller, 1988)</td>
</tr>
<tr>
<td>18</td>
<td>/dev/urandom</td>
<td></td>
<td>Linux Kernel (Love, 2010)</td>
</tr>
<tr>
<td>19</td>
<td>14 with 18 seeding</td>
<td></td>
<td>Develop on this article.</td>
</tr>
<tr>
<td>20</td>
<td>2 with 18 seeding</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Importantly, the expressions (1) and (3) that are used to generate shifts in the RW model and noise in DL are highly influenced by the quality of the generator used, because the generation of random numbers corresponding to three consecutive calls are needed and implies that the sets of possible values generated can be limited by the correlations, the ability to generate 3 calls at least 2 components of equal value is almost null then all possible directions as $(1, 1, 0, 0)(1, 1, 1, 1)(0, 1, 0, 0)$ may not be generated.
For the simulation model RW we used a step $l = 1$, thus the theoretical value of the mean square displacement is $R_{2m}^2 = MN$ with $M = 1$. In the case of the simulation model DL we used the following parameters: $m = 1$, $k_B T = 10^2$, $\gamma = 10^{-3}$ where we use dimensionless $t' = \frac{m}{\gamma} t$ and $R' = \sqrt{\frac{2 k_B T}{\gamma}} R$ thus obtaining for $t' \gg 1$ that $R_{2m}^2 = M(t' - 1)$ with $M = 1$, Euler method on (2) was used with $\Delta t' = 4 \times 10^{-4}$ steep (Lambert, 1991).

The simulation was developed on a Intel(R) Xeon(R) CPU X5570 2.93 GHz 16 Cores 24 GB RAM PC with CENTOS LINUX 5.8 for the X86_64 architecture with kernel 2.6.18-348.3.1.el5, $S = 6000$ particles and 100000 iterations for each PRNG with 100 independents runs. By reduction we refer each PRNG as RNG on result analysis.

In Figure 1 we see the computational speed of each PRNG in correspondence with its algorithmic complexity, the slower PRNG corresponds to the Linux kernel PRNG because of multiple consecutive calls to the /dev/urandom virtual device are made. PRNGs with ID 19 and 20 which are a mixture using the random seeding procedure using the Linux kernel PRNG do not have a noticeable difference with the unmodified original PRNGs 14 and 2.

![#rng evaluation per second](image-url)
Analyzing the graphs presented in Figure 2 a) and b) we note two important aspects, firstly that compliance through the central limit $S_m = \sum_{i=1}^{100000} R_m/100000$ depends not only on the implementation of the PRNG, also the model used as RW or DL, in this case DL being more consistent with the expected results, even though some PRNG shows similar behavior in comparison with the rest, however for the generators 19 and 20 shows a better performance in all cases.

Figure 3 show the average deviation of the theoretical value using two ways to adjust the linear behavior of the equation $4$ on each PRNG, $S_m = Ml + b$ and $S_m = M * l$, in this case shows that the results for the RW model are more closely to theory expected results, however the PRNGs 19 and 20 have a notable improvement over the corresponding PRNG without seeding Figure 3 show the average deviation of the theoretical value using two ways to adjust the linear behavior of the equation $4$ on each PRNG, $S_m = Ml + b$ and $S_m = M * l$, in this case shows that the results for the RW model are more closely to theory expected results, however the PRNGs 19 and 20 have a notable improvement over the corresponding PRNG without seeding.

More details of other statistical tests for PRNGs can be consulted on the url: http://random.mat.sbg.ac.at/tests/ and (Maurer, 1992; Rukhin, et al., 2001; Passerat-Palmbach, 2013).
Conclusions

In general we observe that the influence of the PRNG in diffusion simulation for the proposed system is different, not only depending of used model, the information to be obtained also affects. In the study of central limit average behavior the DL model was better and the study of the standard deviation of the theoretical value was more appropriate RW model for the proposed system.

In all the cases we observe that the PRNG give better results when using PRNG seeding with the Linux kernel PRNG, this result is confirmed for all proposed PRNGs when the number of calls to reset is optimized such that time to gather enough operating system noise with the expression proposed, without affecting significantly the response speed of the PRNG, a factor which is principal for the development of long runs. We only show illustratively only two of the most widely PRNGs used. One per software distribution.

References


L’ECUYER, P.; BLOUIN, F. and COUTRE, R. A search for good multiple recursive random


- RUKHIN, A.; SOTO, J.; NECHVATAL, J.; et al. A statistical test suite for random and


