GENERATING NOISE FOR LANGEVIN DYNAMICS SIMULATIONS

by

Lorenzo Mambretti

Thesis in major, submitted for
completion of Mathematics and Scientific Computation

June 13, 2019

University of California, Davis

APPROVED

Niels Grønbech-Jensen, Ph.D.
Department of Mathematics, College of Letters and Science

Contents

Abstract

We present an analysis of detected statistical anomalies in the results of applying the GJF algorithm for simulating Langevin dynamics in discrete time to a simple one-dimensional oscillator with friction and noise such that thermal equilibrium is obtained.The anomalies are detected in, e.g., calculations of energetic averages, where particular variations around the correct value are noticed as a function of the applied time step. We found that the random number generator RAN3 used for the noise was solely responsible for the observed anomalies. We then performed both a theoretical and empirical analysis of several pseudo-random number generators that can be used a replacement, with the goal of understanding their impact on Langevin dynamics simulations. Our test results show which algorithms and transform methods can produce noise with the most correct distribution, thus allowing for good statistical results. Finally, we recommend the addition of the GJF algorithm as new empirical test for random number generators given that it's easy to implement, it's applicable to any generator, and it has direct physical applications in Langevin dynamics.

1 Introduction

To perform simulations in molecular dynamics (MD) there exist a variety of methods and algorithms which have been developed over the years. One of the most popular classes of methods is based on Langevin dynamics (LD) simulations. In LD, the physical system is based on three components: a force $f(r,t)$, a friction component with friction coefficient $\alpha \geq 0$, and a thermal white noise $\beta(t)$. Explicitly, the Langevin equation is given by [9]

$$m\dot{v} = f(r,t) - \alpha v + \beta(t) \tag{1}$$

To satisfy the dissipation-fluctuation theorem, it is assumed that the stochastic noise is Gaussian distributed, and has the following statistical properties [9]:

$$\langle \beta(t) \rangle = 0 \tag{2}$$

$$\langle \beta(t)\beta(t') \rangle s = 2\alpha k_B T \delta(t-t') \tag{3}$$

The first algorithm which can produce discrete time statistics for Langevin dynamics with exact distribution is the GJF algorithm [3]. When applied to linear systems, this algorithm has been proven to produce exact statistical distribution of the trajectory of and its derived physical measurements, such as potential energy. The algorithm defines the trajectory at a specific time step recursively as a function of the previous state of the system $\{r^n, f^n, \beta^n\}$ and the noise β^{n+1} at the current time step. Explicitly, the trajectory is given by

$$r^{n+1} = 2br^n - ar^{n-1} + \frac{bdt^2}{m} f^n + \frac{bdt}{2m}(\beta^{n+1} + \beta^n) \tag{4}$$

with

$$a \equiv \frac{1 - \frac{\alpha dt}{2m}}{1 + \frac{\alpha dt}{2m}} \qquad b \equiv \frac{1}{1 + \frac{\alpha dt}{2m}} \tag{5}$$

It is easy to see how equation 4 can be easily implemented as an algorithm and used to perform long simulations. A nice property of the GJF algorithm is given by the fact that, for linear systems such as a harmonic oscillator $f = -\kappa r$, we can analytically verify that it produces the correct statistics. In particular, for the above mentioned harmonic oscillator

$$E_p = \frac{1}{2}\kappa \langle (r^n)^2 \rangle = \frac{1}{2}\kappa_B T \tag{6}$$

This analytic solution allows us to actually verify whether the simulation codes are exact, so to compare the test results obtained by computer simulations with the theoretical solution.

1.1 Anomalies

As we know that the algorithm will produce exact solutions for linear systems, we decided to computationally test it on a harmonic oscillator by collecting measures of potential energy for values in the entire spectrum of dt. We set up the simulations with parameters $\kappa = 1, \alpha = 0.05, T = 0.2$ and performed 10^9 steps for 200 equally-spaced values of dt to cover the entire stability limit. A system with these parameters is described as in a regime of underdamped dynamics, and it should be always stable. Moreover, we used the known random number generator RAN3 [11], to create the random numbers used as source for the noise.

Notice that according to equation 6, the potential energy in a harmonic oscillator is invariant on the time-step value dt. Once we ran the simulations, however, we discovered that the simulations were producing abnormal results for some specific values of dt. We were expecting a constant value, but in the spectrum that we obtained there were clear spikes, specially for lower values of dt. The anomalies are shown in the figure below.

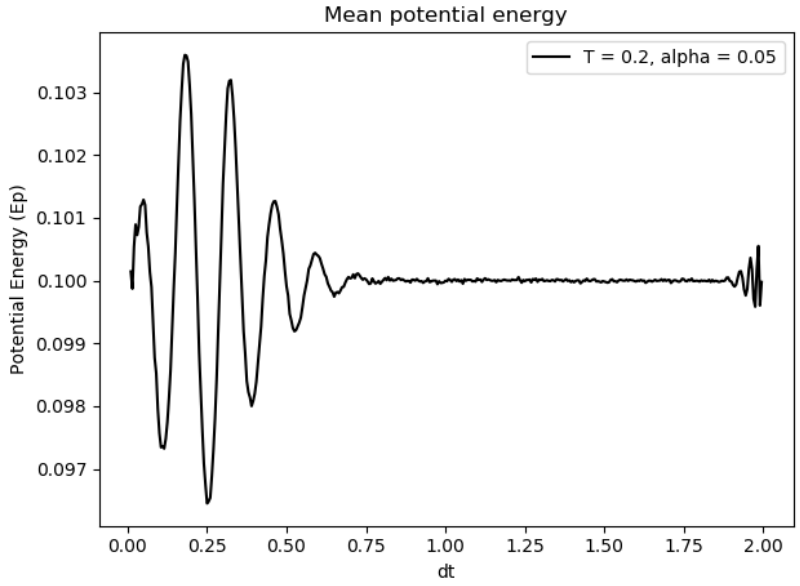

Figure 1: Mean potential energy using $\alpha = 0.05, \kappa = 1, T = 0.2$ and the random number generator RAN3 for 10^9 time-steps.

1.2 Initial hypothesis

As these anomalies were much larger than any statistical error we already accounted for, we realized it was necessary to investigate their possible cause.

As we were confident on the theoretical correctness of the algorithm, our initial hypothesis was that some unforeseen interference within the algorithm was responsible for the anomalies.

We observed that the anomalies were increasing in amplitude at decreasing friction coefficient α or increasing temperature T. The initial observations we did on the anomalies did not provide more insights on potential interference, but a new candidate emerged: the noise. The noise $\beta(t)$ is indeed related to all the above mentioned terms, and it was controlled by parts of the algorithm that were not well studied. We discovered that an incorrect generation of the noise, and in particular a poor random number generator (RAN3), was the sole component responsible for the anomalies in the simulations. This called into question how to correctly generate noise for LD simulations and which aspects of random number generators are truly important to consider. For these reasons, we decided to start a detailed investigation of how the noise is typically generated, which factors come into play, and what replacement we could use.

2 Noise

The noise is perhaps one of the most overlooked aspects of physics simulations. For the Langevin algorithms, the noise is assumed to have a Gaussian distribution and characteristics described by equations 2 and 3. The noise is commonly generated through two steps:

1. A pseudo-random number with uniform distribution between 0 and 1 is generated from an algorithm.

2. The generated uniform random numbers are converted into random numbers with a normal distribution

These steps are fundamental to create the noise with the expected characteristics, and they need to be carefully performed to ensure accurate results.

2.1 PRNGs

Pseudo-random number generators (PRNGs) constitute a class of algorithms which require one or few numbers to be initialized, and they output a new number at each function call. Over time, the sequence of numbers generated is claimed to be "random" if it has statistics which are equivalent to those of random variables. The sequence of numbers is eventually destined to be repeated with a certain period which is characteristic of the PRNG. Here we describe six possible portable random number generators, from the simple to complex.

2.1.1 Park-Miller

A very simple PRNG by Park and Miller [5]. This algorithm is based on a linear congruential generator (LCG) of the form

$$X_{i+1} = aX_i \mod m \tag{7}$$

with parameters

$$a = 7^5 = 16807 \qquad m = 2^{31} - 1 = 2147483647 \tag{8}$$

This PRNG will generate a sequence with period $\approx 2.1 \times 10^9$. This period should be long enough to perform many simulations, although we recognized that it can already be a limiting factor in the most computationally expensive applications.

2.1.2 RAN2 and RAN3

RAN2 and RAN3, described in Numerical Recipes in C,[11] are point of interest because they are trying to solve some of the possible problems of the Park-Miller generator, while still remaining relatively simple. Although more recent PRNGs have been developed, occasionally it is possible to find these PRNGs utilized in software simulations, included some specifically for Langevin dynamics. The initial implementation we had was, indeed, based on RAN3.

RAN2 is a long period ($> 2 \times 10^{18}$) PRNG of L'Ecuyer with Bays-Durham shuffle and added safeguard. At the moment of the publication, it was claimed to produce "perfect" random numbers [11]. In the spectral test that will be executed later, this PRNG is converted to a LCG with parameters

$$a = 1968402271571654650 \qquad m = 2147483563 \tag{9}$$

RAN3 is based on a *subtractive method*, also known as a lagged Fibonacci generator. RAN3 generates the next random number following the rule

$$X_i = X_{i-55} - X_{i-24} \tag{10}$$

It has a long period ($> 2 \times 10^{16}$) and it seems to have a slightly better performance in terms of execution time compared to RAN2. For the spectral test, we consider the terms as they are generated in their LCG form within the algorithm, thus on the parameters:

$$a = 161803398 \qquad m = 10^{10} \tag{11}$$

It's to be noticed that the LCG is used to produce the initial sequence of 51 numbers, while every other number is actually generated by the subtractive method.

2.1.3 RANMAR

RANMAR is a more recent PRNG which is commonly used in MD simulations. It is at the core of software programs directly utilizing the GJF algorithm, such as the LAMMPS software for Molecular Dynamics simulations [10]. Its implementation is based on the combination of two different methods: a lagged Fibonacci Generator and a LCG.

2.1.4 RANLUX

RANLUX is a very famous random number generator, first described by Lüscher in 1994[6]. It has been proven to give high-quality results for many applications. It is currently used in software around the world included for simulations at CERN, where it is regarded as a very high-quality PRNG.[4]

To later perform the spectral test on RANLUX, we can create an equivalence with a LCG based on the parameters described by Lüscher[6]:

$$a = 2^{576} - 2^{552} - 2^{240} + 2^{216} + 1 \qquad m = 2^{576} - 2^{240} + 1 \tag{12}$$

2.1.5 Mersenne-Twister

Mersenne-Twister is another highly-regarded random number generator, first described in 1998 by Matsumoto and Nishimura [8]. We decided to include it in our tests as it is currently one of the most popular PRNGs in the scientific community.

2.2 Transformations

After we obtain random numbers with uniform distribution, we need to convert them into a normal distribution so it has the desired characteristics to perform LD simulations. There are three main methods that we here describe to convert standard random uniforms into normally distributed random numbers. The first two methods are based on the Box-Muller transform [1], while the third method is the Ziggurat Algorithm [7].

2.2.1 Box-Muller Transform

The Box-Muller transform[1] is a method to generate a pair of independent normal random numbers from two standard uniform random numbers. There are two versions of this method.

The first version is based on converting the two numbers to coordinates in Cartesian form. Given two standard uniform random numbers x_1, x_2 the method will generate

$$\begin{cases} y_1 = \sqrt{-2\ln x_1}\cos 2\pi x_2 \\ y_2 = \sqrt{-2\ln x_1}\sin 2\pi x_2 \end{cases} \tag{13}$$

A second version of the method uses coordinates in polar forms. It defines the quantity

$$r = (2x_1 - 1)^2 + (2x_2 - 1)^2 \tag{14}$$

If r is within the interval (0,1], then we accept the value of r. Otherwise a new pair of random numbers x_1, x_2 is generated until r is valid. If r is accepted, then the standard normal numbers are generated as following

$$\begin{cases} y_1 = x_1\sqrt{-2\frac{\ln r}{r}} \\ y_2 = x_2\sqrt{-2\frac{\ln r}{r}} \end{cases} \tag{15}$$

2.2.2 Ziggurat algorithm

The Ziggurat algorithm [7] is a rejection sampling method. The algorithm works by dividing the area under a curve f representing a normal distribution into n rectangles of equal area, stacked from bottom to top and with their left side laying on the y axis. Each rectangle i is described by two numbers x_i, y_i, which are the coordinate of the right-top corner of the rectangle. At initialization, a table of all the rectangles is computed such that $x_n = 0$, meaning that the top rectangle reaches the distribution peak at $(0, f(0))$ exactly. The curve is given by the probability density function

$$f(x) = e^{-\frac{x^2}{2}} \tag{16}$$

The algorithm then generates two random uniform numbers U_1, U_2 and proceeds with the following steps:

1. Let $x = -\frac{lnU_1}{x_1}$
2. Let $y = -\ln U_2$
3. If $2y > x^2$, return $x + x_1$, otherwise return to step 1

This algorithm over time will generate from pairs of uniform numbers a sequence of random variables with normal distribution.

2.3 Tests

The most important properties of our noise is that it needs to be "random" enough. As PRNGs are based on mathematical operations, nothing is truly random, but we can ensure that the numbers generated have statistics which have the same distribution of random independent variables. A variety of tests have been developed over the years to test randomness [2]. Here we are reporting some of the most significant tests and the results on all the previously introduced PRNGs. Moreover, we will use the PRNGs to generate noise for a simulation in the GJF algorithm; in this way we can test whether each of them satisfy the expected properties and achieve results comparable to the analytic solution in a harmonic oscillator.

2.3.1 "Chi-Square" test

The chi-square test (χ^2 test) is one of the best known statistical test. It is used to measure the difference between a set of n observations Y_s and their expected probability distribution p_s over k classes. The statistic can be computed with the formula

$$V = \frac{1}{n} \sum_{s=1}^{k} \left(\frac{Y_s^2}{p_s}\right) - n \tag{17}$$

Since the observations Y_s are generated from random numbers that are supposed to be uniformly distributed between 0 and 1, we expect all probabilities p_s to be identical, $p_s = \frac{1}{k}$, as we consider each class an equally sized partition of $(0, 1)$.

Once we obtain the value V, we can compare it with the expected value. For this test, we take the suggestion of Knuth [2] and consider the test failed if the value V falls below the 1% or above the 99% of the distribution. For the test we consider a distribution over $k = 50$ possible classes, thus V is expected to fall between the values 29.71 and 76.15 to pass the test.

PRNG	$n = 10^6$	$n = 10^7$	$n = 10^8$
Park-Miller	0.95	0.97	0.97
RAN2	0.98	0.96	0.99
RAN3	0.99	0.98	0.98
RANMAR	0.98	0.98	0.97
RANLUX	1.00	1.00	0.99
Mersenne-Twister	0.98	0.95	0.98

Table 1: Percentage of success in the Chi-square test.

From our results we can observe minimal differences between the random generators. The simple Park-Miller generator seem to perform slightly worse at all settings, while RANLUX performed the best across all n.

2.3.2 The Kolmogorov-Smirnov test

The Kolmogorov-Smirnov test (KS test) is another popular test, but it is more suited for observations that can fall into infinitely many values. The test measures the maximum difference between an expected *distribution function* $F(x)$ and an *empirical distribution function* $F_n(x)$, where

$$F(x) = \Pr(X \leq x) = \text{probability that}(X \leq x) \tag{18}$$

$$F_n(x) = \frac{\text{number of } X_1, X_2, ..., X_n \text{ that are } \leq x}{n} \tag{19}$$

To make KS test, from n observations X_j we form the following statistics

$$K_n^+ = \sqrt{n} \max_{1 \le j \le n} \left(\frac{j}{n} - F(X_j) \right); \tag{20}$$

$$K_n^- = \sqrt{n} \max_{1 \le j \le n} \left(F(X_j) - \frac{j-1}{n} \right). \tag{21}$$

Similarly to the chi-square test, we consider the test passed if the results falls between the 1% and 99% of the distribution. In our test, we used several values of n, although we are aware that local non-randomness behaviour may disappear for larger values.

Name PRNG	$n = 10^3$	$n = 10^4$	$n = 10^5$
Park-Miller	0.952	0.96	0.973
RAN2	0.965	0.965	0.966
RAN3	0.955	0.966	0.963
RANMAR	0.960	0.958	0.968
RANLUX	0.962	0.952	0.958
Mersenne-Twister	0.965	0.97	0.965

Table 2: Percentage of success in the Kolmogorov-Smirnov test.

From our results, we can say that all PRNGs pass the test most of the times, and there isn't any significant difference between the simpler random number generators and the more complex ones.

2.3.3 Birthday spacings test

This test was introduced by George Marsaglia in 1984 and well described by Knuth[2]. It is a particularly important test as some classes of PRNGs, particularly lagged Fibonacci generators, consistently fail it. We chose this test because RAN3 was performing poorly on the GJF simulations, and we knew it was based on a Lagged Fibonacci method. For this reason we considered this test the most appropriate to verify that it had statistical problems. At the same time, the test allowed us to check whether other PRNGs had similar issues.

In the test, we select n birthdays in a year with m days, from the random numbers $(Y_1, ...Y_n)$. Once the birthdays are sorted in non-decreasing order $Y_1 \le Y_2 \le ... \le Y_n$ we calculate the spacings $S_1 = Y_2 - Y_1, ..., S_{n-1} = Y_n - Y_{n-1}, S_n = Y_1 + m - Y_n$. Finally we sort the spacings in order and we calculate the number R, the number of equal spacings. For $m = 2^{25}$ and $n = 512$ we have an expected distribution of R that we can compare to test the number. The tested is repeated 1000 times and we perform a chi-square test compared with the correct distribution.

R =	0	1	2	3 or more	Passed
Expected	*0.369*	*0.369*	*0.183*	*0.079*	
Park-Miller	0.365	0.399	0.165	0.071	yes
RAN2	0.371	0.382	0.183	0.064	yes
RAN3	0.130	0.239	0.268	0.363	no
RANMAR	0.373	0.387	0.168	0.72	yes
RANLUX	0.128	0.246	0.289	0.337	no
Mersenne-Twister	0.366	0.367	0.198	0.069	yes

Table 3: Results of the Birthday Spacing test, compared with expected distribution

From our results we can tell that all generators except RAN3 and RANLUX pass it. RAN3 was expected to fail this test; more surprising is the result with RANLUX, given that is highly regarded in the field to be one of the best generators. This makes us suspect that there is a relationship between the wrong distribution of the spacings and the performance in the GJF simulation.

2.3.4 Spectral test

This test is the most complex, but also one of the most precise to identify good random number generators. The test measures the distance between parallel hyper-planes that contain points in t dimensions, where the t coordinates of each point are t successive random numbers in a sequence. A precise implementation of the test is explained by Knuth [2].

We considered dimensions $2 \leq t \leq 6$, and for each of them we compute μ_t, which indicates the effectiveness of the multiplier a of the PRNG for its modulus m. We say that the multiplier a passes the spectral test if $\mu_t > 0.1$. An high value of μ_t in this case means that a is a good multiplier for the given m.

Name PRNG	μ_2	μ_3	μ_4	μ_5	μ_6
Park-Miller	0.41	0.51	1.08	3.22	1.72
RAN2*	0.76	0.00	0.00	0.00	0.00
RAN3**	0.84	1.89	0.13	3.04	1.58
RANLUX	2.27	3.46	3.92	2.49	2.98

Table 4: Results for the spectral test. *does not account for the added shuffling. **accounts uniquely for the initialization sequence

The results for this test are not particularly significant as they confirm that both Park-Miller and RANLUX pass the test, and there is no evidence of a complete failure for the other two either. The test on RAN2 does not account for the final shuffling, thus it might not be representative of the overall quality. Similarly, on RAN3 the subtractive step is not measured and the quality might be affected by that step.

2.3.5 Sequence covariance

Different from the more standard tests, we decided to execute a test to understand whether there is any long-term correlation between some numbers in the generated random sequence. In the sequence of random numbers $(X_n) = X_1, X_2, ..., X_n$, we measure the correlation between any element X_n and the element that is created k steps later:

$$C((X_n), k) = \frac{1}{n-k} \sum_{i=0}^{n-k} X_i X_{i+k} \tag{22}$$

and if the events X_i and X_{i+k} are not correlated, we expect for $C((X_n), k)$ to decrease as n increases. We can use this test on the random numbers both in their standard uniform form and after the transformation into standard normal.

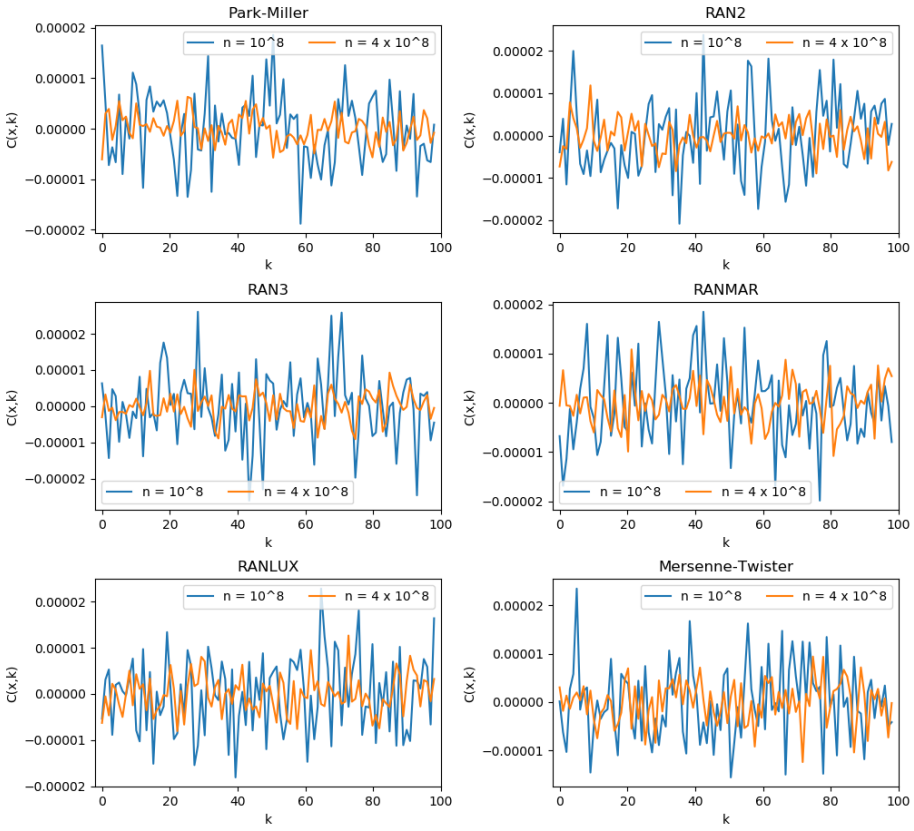

Figure 2: Sequence covariance results for all PRNGs

Our results on the sequence covariance show that all PRNGs behave as expected, showing no sign of correlation for any value of $k < 100$ in the sequence.

In the figures above, it is possible to observe that by increasing the sequence length from $n = 10^8$ to $n = 4 \times 10^8$, the values of the covariance decreased on average by a factor of two, which means that there is no correlation between any two pair of numbers. Moreover, there seem to be no fixed point between the two plots for any PRNG.

Given that this method did not show any particular correlation between the number themselves, we decided to test also the transformations methods described earlier. This allowed us to examine whether the various algorithm either create or amplify preexisting correlations between the numbers produced by the PRNGs. To better show the differences, below we are reporting the square of the values instead of the covariance itself.

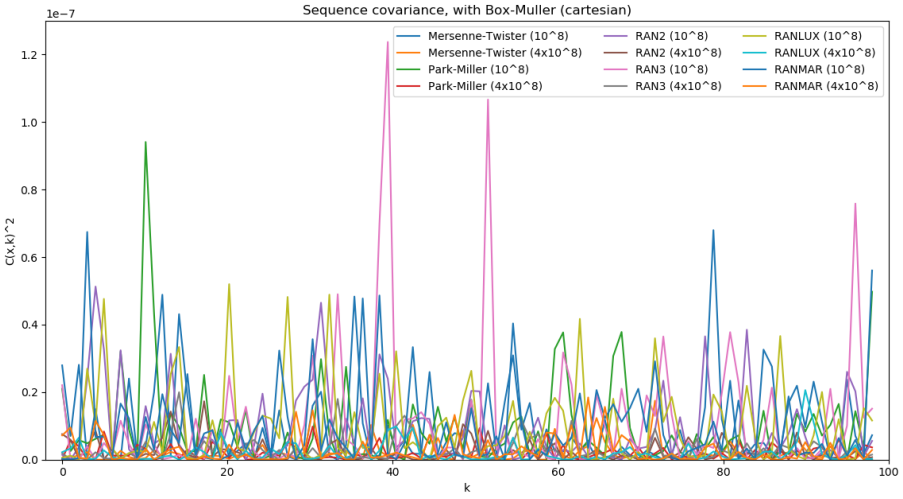

Figure 3: Sequence covariance for numbers after the Ziggurat algorithm. The highest values reported are only sporadically produced by RAN3, but disappear if we increase n

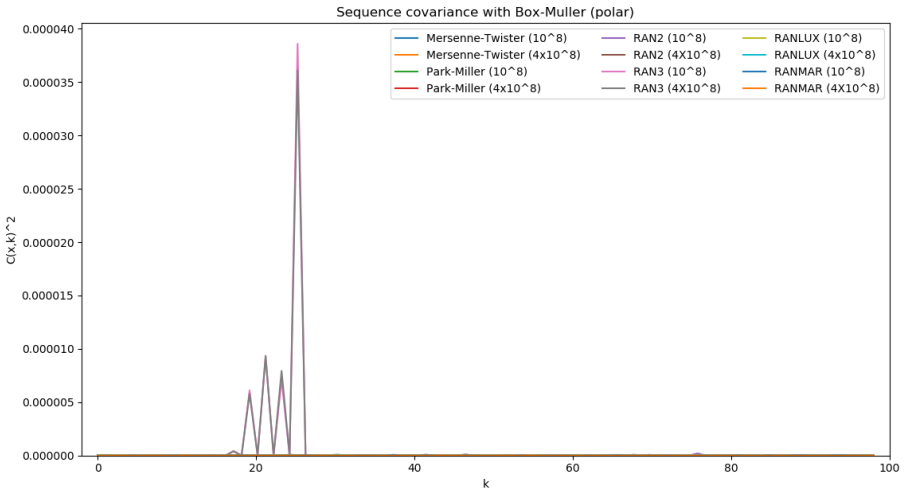

Figure 4: Sequence covariance for numbers after the Box-Muller transformation in Polar form. The highest values reported are produced by RAN3 after the transformation

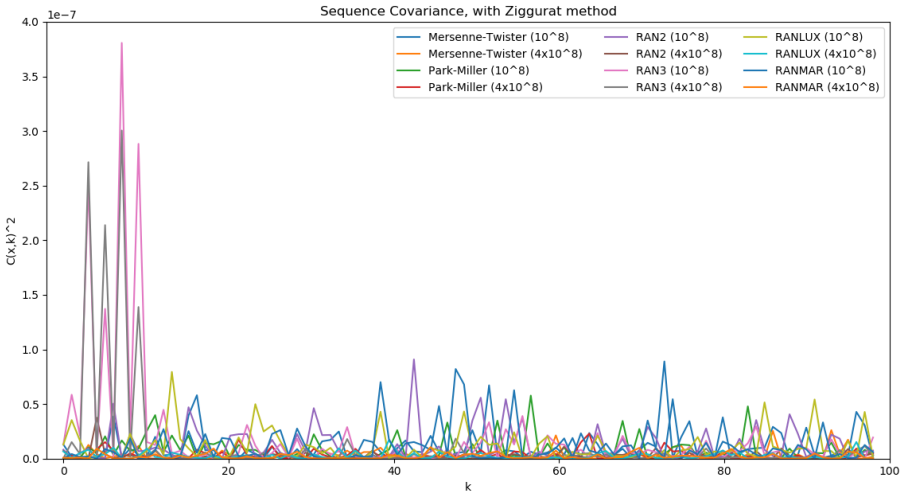

Figure 5: Sequence covariance for numbers after the Box-Muller transformation in Cartesian form. The highest values reported are produced by RAN3 after the transformation.

From our results, we can notice that the Box-Muller transform in polar form increase some existing correlations in RAN3 in a very drastic way. The Ziggurat method also increase the covariance, but with one order of magnitude

less intensity. The Cartesian form of Box-Muller seem not to present particular correlations, although RAN3 still has occasionally the highest values.

2.3.6 GJF simulation on harmonic oscillator

In the GJF simulations, the random numbers influence directly with the system. Although the exact trajectory is unpredictable, we can the expected statistics for potential energy described in equation 6 to verify that the noise has the expected characteristics. To obtain a system that is more sensitive to variations to the noise, we have chosen a lower friction coefficient $\alpha = 0.05$ so that we are operating at an under-damped regime. To allow the harmonic oscillator to stabilize we allow a long period of time of $n = 10^9$ steps of size dt. We run the test run multiple times for different values of $0 < dt < 2$ at intervals of $0.05dt$; this corresponds to the entire stability limit for the chosen parameters. Finally we plot the mean and standard deviation of the potential energy E_p for all values of dt and we compare it with the expected analytic value.

Following we plot the results in the full spectrum for the value of mean potential energy in the simulations, using the different PRNGs, and Box-Muller in polar form as transformation. For all simulations the expected value of $\langle E_p \rangle$ is 0.1.

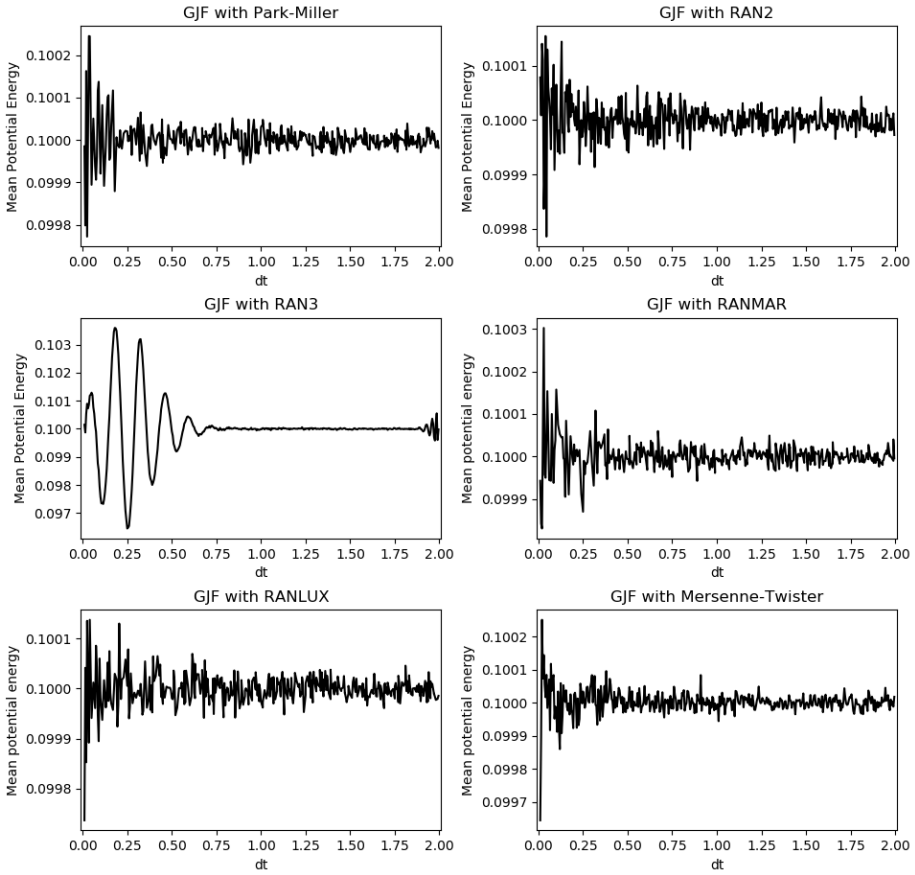

Figure 6: Mean potential energy using different PRNGs

To have a mathematical measure of error along the entire spectrum of dt we take the sum of absolute differences of the values between the computed potential energy and the expected value.

$$S(X) = \sum_{i}^{n} |x_i - \exp X| = \sum_{i}^{n} |x_i - E_p| \tag{23}$$

Name PRNG	$S(X)$ for $0 < dt < 2$
Park-Miller	0.0081
RAN2	0.0085
RAN3	0.1755
RANMAR	0.0074
RANLUX	0.0079
Mersenne-Twister	0.0087

Table 5: Sum of absolute differences for all PRNGs

Both the plots and the sum of absolute differences show clearly how RAN3 causes very significant variations in the results. Some values of the spectrum of dt clearly diverge from the expected value. Among the other PRNGs, no significant difference is noticeable.

3 Discussion

The analysis we conducted successfully identified the pseudo-random number generator RAN3 as the component responsible for anomalies that were observed in GJF simulations of a harmonic oscillator. From the discovery we started a more in-depth investigation of the noise generations, both with a theoretical and empirical point of view. Based on the results collected we are able to draw conclusions on the number generators, the transform methods, and the impact of the GJF algorithm itself.

3.1 PRNG choice

Choosing the appropriate methods to generate noise in MD simulations is non-trivial. From the theoretical analysis and the empirical test executed we make the following conclusions on the examined PRNGs:

1. *Park-Miller*, even though it is very simple to implement and computationally efficient, its period ($\approx 10^9$) is too short to guarantee quality results in MD simulations.

2. *RAN3* shows clear issues in the GJF simulations for a harmonic oscillator, which make it unusable for any MD application. Its problems are detected also with the sequence co-variance test for the Box-Muller transform.

3. *RANLUX*, although it does not show any issue in the simulations, fails our Birthday Spacing test. Since RAN3 is the only other PRNG that fails it, and RAN3 has issues in GJF, we recommend not to use this PRNG in LD simulations as a cautious measure.

4. *RAN2*, *RANMAR* and *Mersenne-Twister* pass all the tests that we decided to execute. The only uncertain result given by the spectral test

for RAN2 does not account for the added shuffling. Any choice of these PRNG works for simulations with the GJF algorithm.

3.2 Transform choice

For the choice of transform method, even though none of the techniques is causing the problem per se, some might amplify existing correlations that exist in the sequence more than others. The one which obtains the minimal error is the Box-Muller transform in cartesian coordinates. Given that it is also efficient as it is not a rejection sampling process, we suggest it as preferred transformation method. Moreover, an implementation of the method, described in Equation 13 is trivial in most programming languages.

3.3 GJF as a test for PRNGs

Finally, we conclude that the GJF algorithm can itself be used as a novel method to investigate PRNGs. Given both its simple implementation and the fact that it has an analytic solution for linear systems, we recommend its addition as new test for all PRNGs. Among the reason why this test should be used:

1. It has direct physical applications in LD, which makes it distinct from most other statistical tests.

2. It is easy to implement, as it is based on a single equation, expressed in 4

3. It is not necessary to convert the PRNG into a Linear Congruent Generator as it is required in the Spectral Test. This makes it both less theory-heavy and it does not require extra steps to compute a and m.

We do not suggest to use this test as a replacement for more complete examinations, but it can be useful to provide more insights of the quality of PRNGs for further improvements in the field.

References

[1] Box G. E. and Muller Mervin E. A Note on the Generation of Random Normal Deviates. *The Annals of Mathematical Statistics*, 29:610–611, 1958.

[2] Knuth D. E. *The Art of Computer Programming*, volume 2: Seminumerical Algorithms. Addison-Wesley, Reading, Massachussetts, 2 edition, 1981.

[3] Niels Grønbech-Jensen and Oded Farago. A simple and effective Verlet-type algorithm for simulating Langevin dynamics. *Molecular Physics*, 111(8):983–991, Apr 2013.

[4] Frederick James and Lorenzo Moneta. Review of High-Quality Random Number Generators. *arXiv e-prints*, page arXiv:1903.01247, Mar 2019.

[5] G Peter Lepage. A new algorithm for adaptive multidimensional integration. *Journal of Computational Physics*, 27(2):192–203, 1978.

[6] Lüscher M. A portable high-quality random number generator for lattice field theory simulations. *Computer Physics Communications*, 79:100, 1994.

[7] George Marsaglia and Wai Wan Tsang. The ziggurat method for generating random variables. *Journal of Statistical Software*, 5(8), 2000.

[8] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998.

[9] Giorgio Parisi. *Statistical eld theory*. Addison-wesley, 1988.

[10] Steve Plimpton. Lammps molecular dynamics simulator.

[11] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical recipes in C: the art of scienti c computing*. Cambridge University Press, 1992.