# Testing Pseudo-Random Number Generators

Kirk L. Byers
*Department of Physics and Astronomy, Texas A&M University, College Station, Texas 77843-4242, USA*
(Dated: April 29, 2015)

This paper will cover my study on Pseudo-Random Number Generators (RNGs) imitating Ferrenberg and Landau[1] using the 2D Ising Model and the Wolff cluster algorithm. The test consists of sweeping a square Ising lattice a large number of time at the critical temperature and comparing the heat capacity with the known expected value. While this has been done before, I will test new generators and verify the statistics of previously tested generators.

## I. INTRODUCTION

As computers become faster and capable of larger computations, the need for better RNGs becomes more important. With more computations, RNGs need to have longer periods while producing numbers very quickly. It has been found that some RNGs that were thought to be "good" actually have correlated numbers when looked at on a long period[1]. Using an RNG to assign spins to a 2D Ising lattice, I swept each lattice $10^7$ times using the Wolff Cluster Algorithm. With the Wolff Cluster Algorithm adding individual spins with a probability of

$$P(spin) = 1 - e^{-2*J/T*k_b} \tag{1}$$

where $J$ is the spin coupling constant and $k_b$ is the Boltzmann constant. For my tests, $J$ and $k_b$ were set to 1, and $T$ was set to the transition temperature. Since multiple spins are checked per sweep, we can look at the results to see if the period of the tested RNG and the integrity of the generator is suitable for general 2D Ising simulations not just single-spin-flip Monte Carlo methods.

## II. METHODS

### A. Wolff Cluster Algorithm[2]

The Wolff Cluster Algorithm is a self-referencing algorithm where a spin is checked to be added to the cluster. If the spin is added to the cluster, its neighbors are then checked to be added. All neighbors that are successfully added to cluster are then put through this process. This continues until no more spins are added to the cluster, and then the entire cluster is flipped and the energy of the lattice is calculated. Spins are added with the probability from equation (1) to the cluster.

The Wolff Algorithm method is preferred over single-spin-flip Monte Carlo method due to the fact that the Wolff Algorithm can overcome the critical slow down seen at the critical temperature. Other clustering methods have been used and tested, but the Wolff Algorithm has shown the best performance.[2]

The lattice was swept $10^7$ times at $T_c = 2.269185$ as it had been done previously. The average energy per spin, the average energy squared per spin, and the specific heat capacity per spin were then recorded. After completing

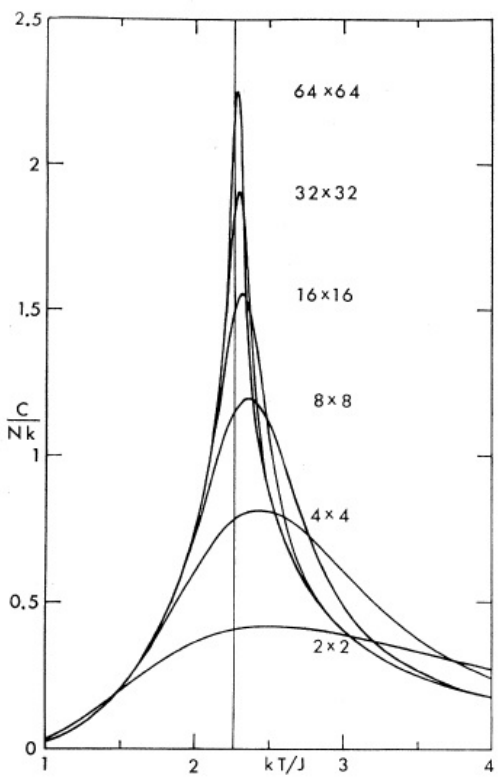


FIG. 1: The specific hear per spin plotted as a function of temperature for nXn lattices for n=2, 4, 8, 16, 32, 64 from Ferdinand and Fisher. The vertical line is $T_c$. [3]

20 tests of a single RNG, the average of the tests was taken, and approximation of error was found using the jackknife method. We can compare the found value for heat capacity per spin with the previously calculated and known value[3]. This value can be verified and calculated for other lattice sizes by evaluating the partial functions found in Ferdinand and Fisher. [3].

The energy was taken by evaluating the Hamiltonian of the 2D Ising model assuming there is no external magnetic field

$$H = \sum_{i \neq j}^{N} J_{ij} s_i s_j \tag{2}$$

This was taken after every sweep, and then averaged to

get the expected value of energy. Similarly, the expected value of the energy squared was recorded, so we could find the heat capacity per spin, $\{C_v/N\}$ :

$$C_v = \frac{< E^2 > - < E >^2}{T^2} \tag{3}$$

In our case, $T = T_c = 2.2691853$ .

### B. The Generators[4]

All of the generators with exception of r1279 were selected out of the GSL. RNGs were selected based on their performance benchmarks provided in the GSL manual or due to their historical significance.

#### 1. *fishman20*

A multiplicative generator with a sequence of

$$X_{n+1} = (ax_n) \mod m \tag{4}$$

Fishman20 has the values a=48271 and$m = 2^{31} - 1$.

#### 2. *mt19937*

Also known as "Mersenne Twister." It has a period of $2^{19937} - 1$ which is a mersenne prime and is equidistributed in 623 dimensions. It was been tested extensively in many applications. One of the most widely used random number generators. It is the third fastest generator in the GSL. The biggest known flaw with MT is also it's large period. The generator preforms better the longer it is sampled. In addition, the state space may be excessive for some tasks and use substantially more cache than generators that can produce similar statistical output for the small period.

#### 3. *gfsr4*

A lagged-fibonacci generator, and produces each number as an xord sum of four previous values.

$$X_n = X_{n-A} \oplus X_{n-B} \oplus X_{n-C} \oplus X_{n-D} \tag{5}$$

Gfsr4 has a period of $2^{D-1}$. In the GSL, the values A=471, B=1586, C=6988, and D=9689 are used to produces a period of about $10^{2917}$. Gfsr4 is ranked as the second fastest RNG in the GSL.

#### 4. *ranlx*

Ranlx series generators are the second-generation of the RANLUX algorithm that generate "luxury random numbers". The higher the number at the end of RANLX the higher the "luxury level". Higher luxury levels produce less correlation between samples. The RANLUX based generators have shown the best mathematical quality [4] . RANLXD produces doubles, and all RANLX can also return integer values.

#### 5. *r250*

R250 was tested as it was the RNG that was found to be bad in Ferrenber and Landau. R250 is a shift register generator with a period of $2^{249}$. It's still very fast, but produces correlated data.

## III. RESULTS

| *Name* | *Avg* | $\sigma$ |
|---|---|---|
| cmrg | 1.497 98 | 0.000 164 778 964 327 |
| fishman20 | 1.500 19 | 0.000 159 796 853 31 |
| gfsr4 | 1.498 59 | 0.000 182 544 967 224 |
| mt19937 | 1.498 83 | 0.000 225 099 131 294 |
| r1279 | 1.498 52 | 0.000 187 858 583 566 |
| r250 | 1.453 32 | 0.000 194 827 446 562 |
| ranlxd2 | 1.500 07 | 0.000 190 679 215 108 |
| ranlxs0 | 1.498 80 | 0.000 227 317 671 245 |
| ranlxs1 | 1.499 91 | 0.000 193 571 746 174 |

TABLE I: Found $\frac{C_v}{N}$ for RNGs over 20 tests. The expected average of the 20 runs is 1.498704959.

As expected, mersenne twister, gfsr4, and r1279 all did very well coming within one sigma of the expected value. Also as expected, r250 is still bad and shouldn't be used ever. It wasn't even within 100 sigma of the expected value. When using the GSL, there doesn't seem like there is much reason to use anything other than the top three rated RNGs for Monte Carlo simulations. Not only are they the fastest, but they performed the best. An interesting note is the difference in performance in RANLXS0 and RANLXS1. From their description, it seems like RANLXS1 should be perform better having a higher "level of luxury." In addition, RANLXS0 out-performed not only RANLXS1, but also RANLXD2. I cannot recommend fishman20, cmrg, RANLXD2, or RANLXS1 for Monte Carlo simulations. Producing an expected value that is about 10 sigma away from the expected value is not very useful. This is fairly disappointing. However, it was quite surprising at how well RANLXS0 produced such close results. Even if RANLXS0 had one of the larger sigmas, it would be a good alternative to mersenne twister if cached memory was a scarce resource.

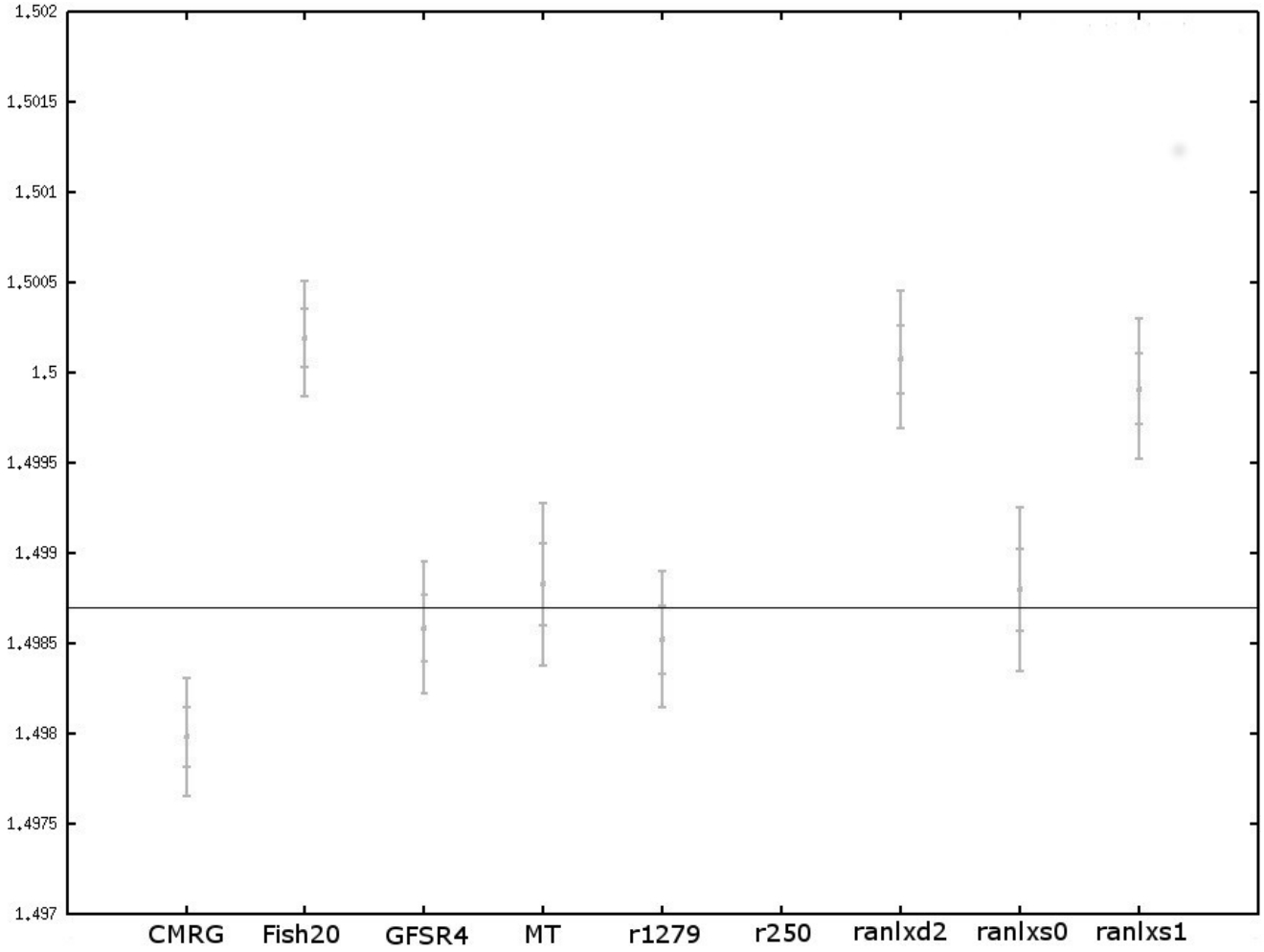


FIG. 2: Figure 2:Each RNG tested is listed with the measured $C_v$ per spin in the latice after $10^7$ sweeps. The expected known value is 1.498704. Shown error bars are at $1\sigma$ and $2\sigma$.

## IV. SUMMARY AND CONCLUSIONS

With the small handful of RNGs tested, I had planned to test more. I had attempted to test Randomph, but could not get MPI to work with me nicely. Also, I had contacted Random.org in hopes of getting a large library of random numbers. I got access to their daily archive of numbers, but after stringing a few files together I realized that working with such a large file wasn't very practical. In addition to learning about the Wolff Clustering Algorithm and RNGs, the majority of the code was developed on my personal computer running windows 7 with cygwin. While frustrating at times, it did give me the opportunity to learn some more UNIX and more than just the rng libraries of the GSL.

## ACKNOWLEDGMENTS

We thank the NSF (Grant No. DMR-1151387) for financial support and Texas A&M University for access to their Eos cluster.

[1] D. P. L. A. M. Ferrenberg and Y. J. Wong, *Monte Carlo simulations: Hidden errors from "good" random number generators*, Phys. Rev. Lett. **69**, 9 (1992).

[2] U. Wolff, *Collective Monte Carlo updating for spin systems*, Phys. Rev. Lett. **62**, 9 (1989).

[3] A. E. Ferdinand and M. E. Fisher, *Bounded and inhomogeneous ising models. i. specific-heat anomaly of a finite lattice*, Phys. Rev. **185**, 9 (1969).

[4] *GNU Scientific Libraryrandom number generation*, `https://gnu.org/software/gsl/manual/gsl-ref_18.html`, accessed: 2015-04-19.