Modern Random Number Generators: Implementation and Statistical Analysis

Kashev Dalmia
Email: dalmia3@illinois.edu

David Huang
Email: huang157@illinois.edu

William H. Sanders
Email: whs@illinois.edu

***Abstract*—We create several PRNGs and then test them using a common battery of statistical tests.**

***Index Terms*—Random Number Generators, Psuedo Random Number Generators, PRNGs, Statistical Analysis, Dieharder**

I. Introduction

The applications of randomness are far reaching. From statistics to simulation, there is a large need for random number generators to perform quickly, generate seemingly random numbers, yet be reproducible in case something needs to be random yet reproducible. Thus, the creation of pseudo random number generators, or PRNGs, has become a large research area in modeling and computer simulation. In the field of analysis of computing systems, simulation is an extremely important part of the modeling process. Today's stochastic models become so complex that solving these models analytically becomes impossible. Simulations driven by randomness are common, and motivate the study in this paper.

TODO:

- Emphasize that here, we mean psuedo random number generators for simulation.
- Mention true hardware random number generators.
- Mention Crypographic RNGs, and emphasize that this is not that.
- Describe the plan for the paper.

This is where the rest of the paper will go. [1]

II. Testing Random Number Generators

This section will describe the difficulties and challenges to testing random number generators, including the rationale behind p value analysis done in dieharder. We will heavily reference the Dieharder manual in this section. Describing how dieharder works is essential, and perhaps describing major important tests, especially the ones from diehard.

III. Classes of Random Number Generators

Here will will describe different kinds of random number generators, the math that makes them work, give examples of each kind, and finally, give their strengths and weaknesses.

A. Linear Congruential Generators

This section will describe LCGs. http://en.wikipedia.org/wiki/Linear_congruential_generator

this section should decribe variants, such as RANDU, also http://en.wikipedia.org/wiki/Lehmer_random_number_generator

B. Lagged Fibonacci Generators

This section will describe Lagged Fib Generators. http://en.wikipedia.org/wiki/Lagged_Fibonacci_generator Also Mention Subtract with Carry: http://en.wikipedia.org/wiki/Subtract_with_carry

C. Xorshift Generators

This section will describe Xorshift generators.

http://en.wikipedia.org/wiki/Xorshift

D. Mersenne Twister Generators

This section will describe the Mersenne Twister class of generators.

http://en.wikipedia.org/wiki/Mersenne_twister

IV. Software Implementation

Describe the software approach, the implemented generators, etc.

V. Results and Analysis

Describe the results of the dieharder tests for our implementations. Discuss speed, and suitability for simulation.

VI. Conclusion

Talk about the generators.

References

[1] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, Jan. 1998. [Online]. Available: http://doi.acm.org.proxy2.library.illinois.edu/10.1145/272991.272995