Biographies

Robert G. Brown, PI

Contact Information:

Department of Physics
Box 90305
Duke University, Durham, NC 27708-0305

Phone: (919-660-2567)
email: *rgb@phy.duke.edu*

Education: B.S. Physics: Duke University, 1977; Ph.D. Physics: Duke University, 1982 in Multiple Scattering Band Theory (Advisor: L.C. Biedenharn). Postdoctoral work: Duke University, in Multiple Scattering Theory, 1982-1988 (Advisor: M. Ciftan).

Professional Experience: Visiting Professor, Duke University, 2001-present; Senior Systems Programmer, Duke University, 1987-present, focusing on large scale cluster computing 1993-present; Visiting Assistant Professor, Duke University, 1989–2001; Instructor/Research Associate, Duke University, 1982–1988.

Five Most Relevant Publications.

1. Primary *Dieharder* project website. Dieharder is available from:
   http://www.phy.duke.edu/∼rgb/General/dieharder.php
2. "Live Random or *Dieharder*", R. G. Brown, article in *Linux Magazine*, September, 2007. This article is available online at: http://www.linux-mag.com/id/4125
3. "Critical behavior of the helicity modulus for the classical Heisenberg model", R. G. Brown and M. Ciftan, *Physical Review B* **74** 224413 (2006).
4. "Monte Carlo study of the helicity modulus of the classical Heisenberg ferromagnet", R. G. Brown and M. Ciftan, in *Condensed Matter Theories* **13** (1998, part I) and *Condensed Matter Theories* **14** (1999, part II).
5. "A high-precision evaluation of the static exponents of the classical Heisenberg ferromagnet", R. G. Brown and M. Ciftan, *Phys. Rev. Lett.* **76** 1352 (1996).

Five Other Publications.

1. "Generalized non–muffin–tin band theory," R. G. Brown and M. Ciftan, *Phys. Rev.* **B27**, 4564 (1983).
2. "A generalized theory of band structure", R. G. Brown and M. Ciftan, *Phys. Rev.* **B32**, 1339 (1985).
3. "Numerical tests of high–precision multiple–scattering band theory", R. G. Brown and M. Ciftan, *Phys. Rev.* **B33**, 7937 (1986).
4. "Multipolar expansions in the empty lattice problem", R. G. Brown and M. Ciftan, *Phys. Rev.* **B39**, 3543 (1989).

5. "The N–atom optical Bloch equations: A microscopic theory of quantum optics", R. G. Brown and M. Ciftan, *Phys. Rev.* **A40**, 3080 (1989).

Synergistic Activities

Beowulf/Cluster/Linux Computing Expert: In addition to teaching and doing research in physics, Brown has been a professional systems administrator and engineer for over twenty years. He is an internationally recognized expert and teacher in the field of cluster computing and an *extremely* active participation for the last twelve years as a teacher and mentor on the beowulf list at http://www.beowulf.org. Brown has helped hundreds of people get started in cluster-based high-performance computing worldwide and regularly teaches independent study students advanced programming.

Predictive Modeling Expert: Brown has written a proprietary neural network engine. This engine, together with a patent pending business process involving Bayesian analysis of partially restricted databases, are contributed intellectual property in the founding of two predictive modeling companies so far, Market Driven and Arcametrics.

Teaching: Brown is an active teacher of both graduate and undergraduate students. His "public good" contributions in this arena include putting all of his teaching materials in physics and computation up online for free as e.g. lecture note style textbooks. These teaching materials have been written up several times by organizations opposed to the high cost of introductory physics textbooks, and have been accessed over *four million times* over the last twelve months by students of physics and computing from all over the world.

Philosophy: In addition to physics, Brown majored in philosophy as an undergraduate. Brown's mentor in philosophy (instructor in half the courses of the major) was George Roberts, himself a student and friend of Bertrand Russell. Brown is is writing *Axioms*, a book on the fundamental axiomatic basis for the theory of knowledge. A *preliminary* draft of this book on his personal website attracts around ten thousand hits a month.

Collaborators

- Lawrence C. Biedenharn (Duke University) – graduate advisor.
- Mikael Ciftan (Duke University/ARO) – postdoctoral advisor, physics mentor and colleague, co-author on most physics papers.
- Dirk Eddelbuettel – Debian developer, especially of the *R* statistical package, collaborator on the *Dieharder* interface to *R* and Debian packager of *Dieharder*.

References

[1] Robert G. Brown. *Dieharder*, A Random Number Generator Test Suite, 2003. Project website: http://www.phy.duke.edu/∼rgb/General/dieharder.php.

[2] Donald Knuth. *The Art of Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley Professional, 3rd edition, 1998.

[3] Jesse R. Walker. IEEE P802.11 Wireless LANs: Unsafe at any key size; An analysis of the WEP encapsulation. Technical report, Intel Corporation, 2000.

[4] Dan Gauthier. Private communication.

[5] The *R* Project for Statistical Computing. Project website: http://www.r-project.org/.

[6] Dirk Eddelbeuttel. Debian *R* developer, private communication.

[7] Robert G. Brown. Live Random or *Dieharder*. *Linux Magazine*, September 2007. http://www.linux-mag.com/id/4125.

[8] Cygwin: A Linux-like Environment for Windows. Project website: http://www.cygwin.com/.

[9] George Marsaglia. The Diehard Battery of Tests of Randomness, 1995. http://www.stat.fsu.edu/pub/diehard.

[10] Rukin et. al. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications (STS). Technical Report 800-22b, National Institute of Standards (NIST), 2001. http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22b.pdf.

[11] Random Number Generator Test Suite (RNGTS). M. Rutti, M. Troyer, W. Petersen. Project website: http://www.comp-phys.org/rngts/.

[12] Scalable Parallel Random Number Generators Library (SPRNG). Project website: http://sprng.fsu.edu/.

[13] M. Mascagni and A. Srinivasan. Algorithm 806: SPRNG: A Scalable Library for Pseudorandom Number Generation. *ACM Transactions on Mathematical Software*, 26:436, 2000.

[14] A. Srinivasan, M. Mascagni, D. Ceperley. Testing Parallel Random Number Generators. *Parallel Computing*, 29:69, 2003.

[15] Pierre L'Ecuyer and Richard Simard. TestU01: A C Library for Empirical Testing of Random Number Generators. *ACM Transactions on Mathematical Software*, 33:3, 2007.

[16] M. Fischler. *Distribution of Minimum Distance among N Random Points in d Dimensions*, FERMILAB-TM-2170, May 2002, private communication.

[17] Paul Leopardi. *Testing the Tests: Using pseudorandom number generators to improve empirical tests*, Talk presented in July 2008 at MCQMC, private communication.

[18] The Gnu Scientific Library (GSL). http://www.gnu.org/software/gsl/.

[19] The Gnu Public License (GPL). http://www.gnu.org/copyleft/gpl.html.

[20] Michael Luby. *Pseudorandomness and Cryptographic Applications.* Princeton Computer Science Notes, 1996.

[21] Bob Jenkins. est.c, 1994. Unsupported, public domain tests for random number generators.

[22] S. Wegenkittl. Entropy estimators and serial tests for ergodic chains. *IEEE Transactions on Information Theory*, 47:2480, 2001.

[23] S. Wegenkittl. Dissertation and other reprints. Private communication.

Project Summary

Proposal: The proposed project is the continued development and support of *Dieharder*[1], a Gnu Public Licensed (GPL) random number generator (RNG) test suite. Funding is being requested for adding new tests and integrated RNGs to the suite; further developing and debugging existing tests; providing direct support to users of the suite and to developers of integrated statistical interfaces such as *R*; reporting on the results of tests conducted with it in suitable journals of computational statistics and otherwise advertising its existence to the broad multidisciplinary community that relies on RNGs.

Methodology: A typical test consists of numerically evaluating some *a priori* known test statistic via sampling, using the test RNG. The numerical result is transformed into a p-value – the probability of obtaining the numerical experimental result given the *null hypothesis* $\mathcal{H}_0$: that the test RNG is perfectly random. Extreme values or non-uniform distributions of p signal a failure of the null hypothesis – a “bad” (non-random) RNG.

Intellectual Merit: The foundation of science itself is distinguishing the random from the nonrandom, the discovery of pattern and subsequent inference of a rule that explains the pattern. The empirical testing of RNGs is science in a computational microcosm, but the goals are reversed – one wishes to discern the (near) *perfect absence* of pattern in the output of some *mechanism*, be it computational or physical. The invention of new RNG tests, the aggregation of existing tests into systematic series that provide specific information about modes of failure, and the discovery and repair of published tests that are themselves broken in some way is worthwhile work with many intellectual challenges. Equally important is the *application* of the resulting broad base of tests in a uniform testing interface to the best and worst of the modern RNGs to attempt to discover where and how they empirically deviate from true randomness. This information is of critical value to both RNG developers and RNG users.

Broader Impact: RNGs are extremely important tools in many branches of science, industry and government. Numerical modeling, simulation, and cryptography in particular rely on having a computationally inexpensive stream of “sufficiently random” numbers. Being able to test any given RNG for “sufficiency” in a domain specific context, however, is in and of itself a serious problem in advanced statistics.

Every numerical simulation is a compution of the desired quantities *assuming* the truth of $\mathcal{H}_0$. Bayes theorem then tells us that the probability that the answer obtained is the *right answer* depends strongly on the probability that $\mathcal{H}_0$ is correct for problems *like* the simulation problem at hand. However, this is rarely objectively assessed in the literature, because of a lack of readily available tools with which to perform such an assessement. The *Dieharder* project seeks to remedy this lack and put a single well-documented open source, easy-to-use RNG testing tool at the disposal of every consumer of random numbers for *free* to objectively assess the quality of their RNG.

Dieharder is achieving part of this goal and has many users already. Much hard work remains, however, before the tool is truly universal, capable of running all known useful tests from a single consistent interface.

Dieharder Project Description

Introduction

Random numbers are important in mathematics, in the physical and social sciences, and in the world of business and commerce. In mathematics, random numbers are used to evaluate (for example) high dimensional integrals, to study Markov processes, to test results in statistics. In the physical and social sciences, random numbers are used as the basis of simulations of complex systems, often ones that are beyond our ability to analytically compute. In the world of business, random numbers are the fundamental basis for encryption, which is the core enabling technology for virtually all of the world's e-commerce and e-banking. Many software tools and products in common use require a reliable stream of (sufficiently) random numbers, from gaming to complex statistical modeling programs.

Truly random numbers are of course difficult to come by – the term "Random Number Generator" (RNG) is an oxymoron in computation if not in physics[1]. If a number is produced by an algorithm within a software RNG from a definite initial state, it isn't truly "random". A stream of numbers produced by a software generator may, nevertheless, have many of the desirable properties of truly random numbers and can therefore be *useful.* It can be very, very difficult to perform numerical experiments that reveal a weakness in a high quality software RNG, and tests become so subtle that one is often left uncertain as to whether a failure is due to the RNG or due to a weakness in the test. Software RNGs and RNG testing are cutting edge computer science and cutting edge computational statistics, respectively, meriting an entire chapter in e.g. Knuth's *The Art of Programming*[2].

Random numbers are just as difficult to generate using hardware devices based on natural processes. Many "random" processes in nature are random in the sense that they are *unpredictable* given our incomplete knowledge of the state of the physical generator, but are not truly *random.* Even when a hardware-based stream is generated by a "truly random" process (such as radioactive decay) the resulting set of of numbers may still have bit-level biases and hence fail to be distributed correctly in accordance with the *theoretical properties* required of an RNG in order to make it useful in simulations and statistical modeling.

Hardware generators can have other flaws as well. Most are much slower than modern software generators. They are often rate limited both in the random physical process itself and by internal autocorrelation times (during which they are very definitely non-random) in that process. Also, it is often desirable to be able to *reproduce* a particular stream of random number (a process that is essential to encryption and decryption, for example), but this is generally *impossible* for hardware generators that rely on missing state information (entropy) or radioactive decay to generate sequences that are unpredictable and cannot be reproduced.

The history of computing is peppered with examples of problems that have arisen due to the use of weak RNGs (generally software RNGs). They are the bane of the simulation scientist – a Monte Carlo simulation is *always* as much a test of its underlying RNG as it is

[1]Software RNGs are therefore often called "pseudorandom number generators" to acknowledge this point. We do not differentiate because *Dieharder*[1] can be used to test anything that produces a supposedly random stream of bits.

a method of computing a result. If the result is theoretically known and a simulation fails to correctly evaluate it within the bounds allowed for by statistics itself, we would use this as grounds for rejecting $\mathcal{H}_0$ and concluding that the RNG is not random. If the result is *not* known, a Monte Carlo computation of the result is *always* conditional – *if* the RNG used to compute it is "random enough", then the result is reliable. If not, it may *not* be reliable.

They are similarly the bane of the cryptographer. A lack of sufficient "randomness" in the underlying generator in this case appears as a reduction in the size of the space that must be searched when seeking to crack an encrypted message. Subtle patterns in the digit string produced by an RNG can provide clues to both state information and the algorithm used to perform the encryption. Information *of any sort* weakens encryption, and again there have been spectacular and quite recent failures of encryption schemes (such as the Wireless Encryption Protocol – WEP[3]) that indicate the importance of having a "random enough" encryption scheme.

We see that applications of all sorts that rely on random numbers can be viewed as *tests* of the RNG used in the sense that if the RNG fails in certain ways, the application will fail to accomplish its purpose as well, be it evaluating a critical exponent correctly in a physical simulation or successfully protecting an internet transaction. This failure may occur "invisibly" and be very difficult to detect, and of course any such failure can have an enormous *cost*. It is therefore clearly desirable to have a reliable RNG testing tool that not only can test RNGs at a useful level today, but that can scale up along with the appetite of future applications for random numbers and still be a useful and reliable tool in a decade. *Dieharder* is intended to be such a tool.

Dieharder History

The *Dieharder* project[1] was initiated in January of 2003 and has slowly been carried to its current stage of development without any grant support. It grew out of the PI's interest in random numbers and RNGs in the context of twelve years of conducting Monte Carlo simulations, some of it using RNGs that in retrospect were appallingly weak. This interest was further stimulated by the discovery by a colleague in the Duke physics department of a nonlinear optical process that might serve as the basis for a hardware RNG[4]. The PI's assistance was informally requested to help test random sequences produced by this process, and the PI discovered that there simply weren't any good, open source, general purpose random number testing tools and so decided to write his own.

Over the course of five years the tool has radically changed as the PI has learned more of the subtle art of RNG testing. The entire tool has been rewritten from the ground up *three times* over this period, the first time to create the rudiments of a portable interface, the second time to make the tests far more "object-like" and implement them all in a portable library, and the third to produce a much better user/reporting interface.

The implementation of all the *Dieharder* tests in a portable library has facilitated the integration of the *Dieharder* tests directly into the *R* statistical toolset[5], permitting *R*'s integrated RNGs to be directly tested by *Dieharder* and permitting the results of *Dieharder* runs or arbitrary RNGs to be displayed or further processed with *R* itself. The *R* port is entirely due to Dirk Eddelbeuttel[6], and is not a *direct* part of the *Dieharder* project but rather a collaborative offshoot. This is an example of the synergy already being generated by *Dieharder* in the statistics community.

For some two years now *Dieharder* has been sufficiently broadly available (and sufficiently reliable and well-documented) that it has come to be adopted by an increasing base of users worldwide. For almost two years now it has been “instantly installable” under Debian and available in both binary and source RPM format for RPM-based Linux. The general design and philosophy of *Dieharder* has been written up in *Linux Magazine*[7], which brought it to the attention of many new users all over the world. Users have ported it to *Cygwin*[8] for use on Windows-based machines (with some help from the PI). It has been similarly built on freebsd and other unix/posix systems by users, with less direct support from the PI due to a lack of local systems running the operating systems in question on which to do the work.

Dieharder has already earned a reputation of being highly responsive to the requests for features and bug fixes from the community of users – its last interface redesign is less than three months old and was undertaken to provide just *one* (very large scale) user with the output reporting format they required in order to be able to most easily import the results of massive *Dieharder* testing into spreadsheets and other post-processing reporting filters.

Over the five years of the *Dieharder* project two undergraduate students have contributed to the project in the course of independent studies in computer science under the direction of the PI. Both students learned a great deal about both random numbers and RNGs and about the testing process, while gaining valuable experience working on an actual open source tool.

Recently, however, *Dieharder* development and maintenance has suffered due to a sheer lack of compensated time available to the PI, coupled with a veritable explosion in the number of users requesting support of one sort or another after the tool became universally available. To put it bluntly, the PI has time to either help users or add new tests but not both, while at the same time discharging his compensated duties. This is the primary reason support is being requested to help carry the project, which has come so far and benefited so many “for free” already, to the next stage of its planned development.

Testing RNGs

The most common methodology for testing RNGs is easily described and understood in the context of an actual example. If one sums a large number $N \gg 1$ of uniform deviates (floating point random numbers uniformly distributed on the range $[0, 1)$), the expected mean value of the sum is $N * 0.5$. From the Central Limit Theorem, one expects the *distribution* of many such independent and identically distributed (*iid*) sums to tend towards the normal distribution $\mathcal{N}$. One expects the variance of this normal in the limit of many sampled means to be $N/12$. This is one of *many* known statistics that can easily be generated from a perfectly random sequence of sufficient size.

To test a particular RNG, we begin by making the *null hypothesis* $\mathcal{H}_0$ that the RNG is a perfect RNG (at least as far as this test is concerned). If we *form* the sum of N samples and compare the result to the expected value, the normal *cumulative distribution function* (CDF) for the Gaussian (relative to the expected sample mean and for the expected variance) at this value yields the *probability* of obtaining this particular result given $\mathcal{H}_0$ – the *p-value* for the test.

A standard testing methodology (as implemented in many, if not most existing RNG testing programs[2]) would then proceed as follows. If p is small – say less than some cutoff such as 0.05 – one reasons that there would be only a $< 5\%$ chance of obtaining this result given the null hypothesis $\mathcal{H}_0$. This number is small, so we *reject* the null hypothesis and conclude that the *alternative* hypothesis (that the RNG is biased) is more likely to be true. But how much more likely?

In the *STS* documentation, it is suggested that one can reject the null hypothesis with "95% confidence", but this is not, in fact, correct. This single test result does not (or should not) give us a great deal of confidence either way that an acceptance or rejection of the RNG is correct. For one thing, if one runs the same test ten thousand times with different seeds using a perfect RNG, one *expects* to get p-values less than 0.05 roughly 500 times, to the extent that if one failed to get *approximately* 500 p-values in this "reject" range one would, in fact, have extremely sound grounds for rejecting the null hypothesis! Clearly we would not in this case reject the null hypothesis 500 times and *accept it* 9500 times

The point is that p itself is a random variable, and if $\mathcal{H}_0$ is true one in fact expects p to be uniformly distributed. The probability of a perfect RNG producing a p-value in the range 0.65-0.70 is precisely the same as the probability of it producing a p-value in the range 0.00-0.05, and we'd be precisely as justified rejecting it in the one case as in the other on the basis of a single p-value.

Clearly we cannot interpret the 5% rule as yielding a 95% probability that we would be *correct* in rejecting $\mathcal{H}_0$. We can only interpret the p-value as the thing that it literally is: the numerical experimental result inverted by the CDF so that it *should* be a uniform deviate. Just as nobody would be shocked to see a RNG produce a uniform deviate of 0.032, nobody should be shocked to see a test of some RNG produce a p-value of 0.032. The only important question is: How *often* does it produce p-values (or uniform deviates) in any given range?

The correct thing to do for any given test is therefore to run the test *many times* and determine the *distribution* of p. If p is appropriately uniformly distributed we should fail to reject the null hypothesis. If it is not uniformly distributed or is *too* uniformly distributed, we should consider the alternative hypothesis. To assess the uniformity of p in a quantitative way, a Kolmogorov-Smirnov (KS) test is applied to the distribution of p itself in comparison to a uniform distribution to generate a final p-value for the entire run of the test being applied.

This is the strategy uniformly implemented in all *Dieharder* tests. The default test run for any test generates at least 100 p-values, and the number of p-values generated per test can easily be increased to the limits of one's patience, the supply of random numbers from the RNG being tested, and one's available computational power.

Having the ability to increase the number of p-values results in a remarkable thing. Anyone who has ever used a non-*Dieharder* RNG tester that generates single p-values per test knows that judging whether or not a RNG should be rejected is usually extremely

[2]Such as *Diehard*[9], the *Statistical Test Suite* (*STS*)[10], the *Random Number Generator Test Suite* (*RNGTS*)[11], the tests included with the *Scalable Parallel Random Number Generator* (*SPRNG*)[12],[13],[14]

difficult. As George Marsaglia says in the documentation for *Diehard*, "p happens"[9]. By this he is encouraging users not to be too quick to reject a RNG for producing a p-value less than 0.05 or for that matter, 0.01, but if not then, when *should* you reject a RNG as non-random?

In the case of *Dieharder*, a reasonable cutoff might be 0.0001, or even *smaller* (especially as the number of tests grows). Suppose one runs a test a single time and generates a p-value of 0.34. This seems reasonable. One runs it again and gets 0.046. Naively we might reject it. Again and we get 0.89. *If* the RNG is in fact *not* good but is not too bad, either, we might get reasonable p-values most of the time, but the actual distribution of p might not be uniform. Accumulating 10, 30 or even 100 p-values might not be enough to reveal its non-uniformity and might return a number like 0.032 for the final KS p-value – an edgy number, but not a certain failure. If, however, one accumulates *1000* p-values, a truly weak RNG will exhibit a *clearly resolved failure* with a final p-value more like 0.000003, where a RNG that should *not* be rejected will most likely regress towards the mean. If not 1000 p-values, then 10000.

For enough p-values, the final KS test for a bad RNG (where one *should* reject $\mathcal{H}_0$) will return a totally unambiguous KS p-value of *zero*, not a hard-to-interpret p-value of 0.032, and it will do so *every time* the test is repeated at that resolution. *Dieharder* thus replaces an ambiguous art with *science* – impossible to miss, reproducible failure, where as is *also* the case in science, success cannot so easily be empirically proven.

The methodology used in *Dieharder* is so powerful that it has thus far revealed several errors in e.g. target statistics imported from *Diehard* by consistently failing (in the same way) a number of RNGs that use very different algorithms and that are supposed to be quite sound theoretically (including a source of "true random numbers" from atmospheric noise[**?**]). Because it is actually quite *difficult* to know if consistent failures of supposedly good generators is due to a failure of the test or due to the *power* of the test, one of the design goals for *Dieharder* calls for a "gold standard" RNG to be integrated with the test suite and used to validate the test suite itself.

This short description does not by any means exhaust the methodology for the RNG tester to use to accept or reject $\mathcal{H}_0$. L'Ecuyer[15], for example, goes into greater detail than this proposal does into subtleties of the testing process when applied to e.g. discrete distributions, but in general supports the method (implemented independently in *Dieharder*) of increasing the p-resolution of RNG tests until failure is unambiguous. A paper by Srinivasan, Ceperley and Mascagni available on the *SPRNG* site[12] describes some of the special problems associated with testing of parallelized RNGs and methodology that we hope to implement in *Dieharder* during the course of the proposed grant.

Dieharder **Design Goals**

Large scale Monte Carlo simulations in condensed matter and high energy physics can easily consume months to years of compute time on a beowulf cluster (a very expensive process, in aggregate). Simple economics dictates that it is well worth the investment of resources required to test the generator used in such a computation at a scale similar to that of the application, with a reliable and properly supported test suite. Failure to do so can result in expensive and embarrassing errors. Few computer scientists or physicists are so naive these days as to use any of the "known-bad" RNGs (such as *R250* or *randu*) in such a computation, but knowledge of the need for testing is not the same thing as the means at hand to test the many RNGs provided in various libraries or in source in order to be able to decide which generator is *optimal* for the problem, in that it provides a stream that is *random enough* at the greatest possible *rate*.

In order to be able to rely on any given RNG in any given application or numerical computation, a researcher must be able to test it. A good RNG test suite should not only be able to *discriminate* a "good" RNG (one that possesses all of the theoretical properties of a perfect RNG at the statistical resolution of the testing process) from a "bad" RNG (one that consistently fails one or more tests used in the testing process) – it needs to indicate *how* it fails the tests that fail, as computations may well be more sensitive to one kind of failure than to others. A truly useful test suite would therefore provide the tester with important information about the mode of any observed failure, the *way* the RNG deviated from theoretical randomness. This sort of systematic information is of critical importance to researchers seeking to design improved RNGs or debug existing ones. It is also of critical importance to programmers or scientists seeking assurance that a given RNG is random *enough* to produce a reliable result for their particular project where they may have grounds for believing that certain kinds of failure (such as a generator being a bit "too uniform" for certain ntuples but otherwise quite random) will not have a large impact on their results where others (such as not being uniform enough) will.

The design goals of Dieharder are for it to be:

Easy to Use: *Dieharder* should be easy to obtain, easy to install, and easy to use. The ideal is for a user to be able to type e.g. `yum install dieharder` or `apt get dieharder` and two minutes later start testing RNGs. *Dieharder* is the *only* RNG tester available in this packaged form as well as in portable source. It is currently has a command line interface and output modes that facilitate loading results directly into spreadsheets or second-stage analysis software. A portable *graphical* interface and Windows and Macintosh versions (including packaging) are future design goals.

Complete: As an open source tool, an ambitious design goal is to include in *Diehardér* *all* potentially useful random number test algorithms documented in the literature or incorporated into other RNG testing programs. Currently it contains 28 distinct tests and produces 106 KS p-values in addition to providing a speed benchmark for integrated RNG tests. However, many more tests are known to the PI and will be incorporated as soon as possible (see below). As a nice side effect of the open development process, users working in the field of statistics are already beginning to contribute new tests as well as helping to fix old ones[16],[17].

Scalable: A simulation running on a modern computer cluster can consume enormous numbers of random numbers. Bearing in mind that *every* such simulation *is* a test of the RNG, RNGs need to be testable at commensurate scales. To facilitate this (on top of its essentially scalable design) a parallel version of *Diehardér* is to be developed (incorporating, for example, methodology and generators in the *SPRNG*) that can run the tests on a compute cluster of essentially arbitrary size. This will facilitate the *exhaustive* testing of the best generators to reveal any systematic problems that might be lurking when 10^{16} or more random numbers are being produced in the course of a single computation using a sheaf of distinct seeds on different nodes.

Systematic: A RNG tester should do more than just flag problems in a generator. It should provide information about the *mode* of failure. *Dieharder* already incorporates several systematic series of tests but needs many more. RNG streams are often generated from iterated maps and can thereby contain extremely subtle long-period cyclic correlations. *Dieharder* needs a more sophisticated interface to RNG streams that permit it to generate numbers to be tested out of lagged recombinations of bit ntuples to reveal e.g. long range lagged correlations in the stream.

Extensible: It should be (and already is, to some extent) easy for users and the primary developers to add new tests and new RNGs (the latter to *Dieharder* itself or to the Gnu Scientific Library (GSL)[18] that provides its base of "instantly" testable RNGs). Its extensibility needs to be improved and maintained, especially as it is parallelized (as most users are unlikely to be familiar with the methodology of code parallelization). *Dieharder* will also need to be made fully 64-bit compatible as 64-bit RNGs appear on the scene over the next few years.

Portable: As development continues, *Dieharder* needs to be ported, tested and "certified" on a variety of platforms. This includes both Unix-based platforms such as Linux/Gnu (the primary development platform so far), Solaris, HP-UX, FreeBSD and Mac OSX as well as Microsoft Windows. *Dieharder* has been developed so far using POSIX compliant C, but portability issues have nevertheless consumed a disproportionate amount of developer time thus far and it is likely that this will, unfortunately, continue.

Open: This is a *critical* design goal. In order to properly support the research community, a RNG test suite needs to be *open source* and *freely available*. For researchers to be comfortable using the tool in the current age of the Digital Millenium Copyright Act, there needs to be no hint of proprietary ownership about it, no restrictions on the mode of use, no obstacle to user modification of the tool. As a Gnu Public License (GPL)[19] package, *Dieharder* is *aggressively* open. Unfortunately existing RNG test suites do not all meet this criterion; in particular *Diehard*, the *RNGTS* and L'Ecuyer's *TestU01* suite all have non-commercial restrictions on the use of their source code although their algorithms are of course published, and other openly published source code (e.g. *SPRNG*) often contains no copyright notice or license at all. *Dieharder* has received an enthusiastic welcome by the members of the statistics community that have discovered it for this reason alone.

Well Documented: RNG testing is a fairly challenging problem in statistics. In fact, it is a *textbook* problem in statistics – it incorporates the fundamental concepts of statistical testing in an extremely pure form. A design goal of *Dieharder* is to have textbook quality documentation available to the extent that a consumer of random numbers in the less technical arenas of commerce or science can educate themselves at the same time they validate the RNGs they wish to use for some task. Documentation of this quality would also facilitate the development of actual undergraduate or graduate courses in statistics built around the use of *Dieharder* to illustrate the many concepts in statistics that are fundamental to its operation.

Dieharder **Tests**

Source/Test Name	Status	Remarks
Diehard Birthdays	Installed	Good
Diehard OPERM5	Installed	Suspect
Diehard 32x32 Binary Rank	Installed	Good
Diehard 6x8 Binary Rank	Installed	Good
Diehard Bitstream	Installed	Small Error in Test
Diehard OPSO	Installed	Small Error in Test
Diehard OQSO	Installed	Small Error in Test
Diehard DNA	Installed	Small Error in Test
Diehard Count the 1s (stream)	Installed	Good
Diehard Count the 1s (byte)	Installed	Good
Diehard Parking Lot	Installed	Good
Diehard Minimum Distance (2d Circle)	Installed	Good
Diehard 3d Sphere (Minimum Distance)	Installed	Good
Diehard Squeeze	Installed	Good
Diehard Sums	Installed	Suspect
Diehard Runs	Installed	Good
Diehard Craps	Installed	Good
Marsaglia and Tsang GCD	Installed	Good
STS Monobit	Installed	Good
STS Runs	Installed	Good
STS Serial	Installed	(Generalized) Good
RGB Bit Distribution	Installed	Good
Fischler Generalized Minimum Distance	Installed	Good
Knuth Permutations	Installed	Good
RGB Lagged Sum	Installed	Good

Table 1: Tests already in *Dieharder*

As one can see from Table 1 above, *Dieharder* at this point contains all of the *Diehard* tests, several tests from the *STS*, as well as several tests that were written by the PI for *Dieharder*. The RGB bit distribution test is a non-overlapping generalization of the *STS* serial test; both have been implemented in *Dieharder* to run on an entire *series* of bit ntuple sizes. This has proven to provide useful new information even about "believed good" RNGs such as the Mersenne Twister – as one can see in the example run below, it performs a tiny bit too *well* on the bit distribution or series tests at certain ntuple sizes, detectable as a non-uniform distribution of final test p-values with a distinct high bias.

The Generalized Minimum Distance test was brought to the attention of the PI by M. Fischler[16] and subsequently added to *Dieharder* and generalizes two *Diehard* tests: the Minimum Distance test and the 3D Spheres test (which are actually the same test in 2 and 3 dimensions). The Generalized Minimum Distance is automatically run on a series of dimensions from 2 to 5 (the current upper bound supported by the test) again in harmony with the design objective of providing a user with a set of systematically generated results that permits them to pinpoint failure and obtain some insight as to the *mode* of failure and whether or not it might impact their e.g. Monte Carlo simulation.

The Knuth[2] Permutations test is the non-overlapping (simpler) form of the *Diehard* Operm5 (permutation distribution of overlapping 5-sequences). It is included for several reasons. First and foremost, the Operm5 test in *Dieharder* is almost certainly broken. It fails *all* generators tested, including hardware generators and the best of the software generators in a consistent way. The failure is subtle – it is invisible unless one accumulates order of 100 p-values of the test (or more), at which point the small bias in the distribution of p becomes apparent and the final KS test yields unambiguous failure. Since *Dieharder* contains original code and the precise method for deriving the weak inverse of the permutation correlation matrix used in the test can only be inferred from the code and is not published anywhere as far as we can tell, it is very difficult to be *certain* that the version of Operm5 test as ported from *Diehard* is actually broken, and if *Diehard* itself fails in a similar way.

However, the Permutations test tests the same basic statistic – the frequency distribution of order permutations produced by the RNG – without the confusing issue of the covariance matrix and the particular permutation sort order used in the test. It very clearly shows no flaw with the "good" RNGs as far as their production of permutations is concerned, allowing us to be certain that Operm5 is flawed at *least* as implemented in *Dieharder*. It also permits us to *easily and transparently* test the permutation distribution of random numbers not just for 5 sequential numbers but for sequences pulled 2, 3, 4, 5, 6, 7 at a time (or more) from the RNG, although human patience waiting for test results practically limits us to this general range at this time.

Rederiving and implementing a functional overlapping permutations test that can be run on at least a similar range of permutation sizes is one of the serious pieces of work planned to be conducted during the course of this grant, if funded. Work has similarly already begun to fix the *Dieharder* version of the *Diehard* Sums test, which exhibits a similar signal of test failure. Users have already contributed fixes (in the form of precise target statistics[17]) for the various "monkey" tests from *Diehard* that are incorporated into *Dieharder*, and it is expected that these fixes will be implemented before the beginning of the period of support.

In addition to all of the tests above (many of which return multiple results or a series of results) *Dieharder* currently contains *72* distinct RNGs that are *integrated* with the test so that they can be immediately tested for possible use or their test results compared to external RNGs. Integrated RNGs include "good" generators such as several variants of mersenne twister, gfsr4, taus2, ranlxd2; several "bad" generators: the infamous randu, R250; and many in between.

The simplest way to illustrate *Dieharder*'s use is via an example of its application to one of the best generators, mt19937_1999 (from the GSL). The following is a *partial* table of results from an actual run of the current 3.28.0beta snapshot:

```
rgb@lilith|B:1069>./dieharder -g 014 -a | tee dieharder-g-014-a
#=============================================================================#
#          dieharder version 3.28.0beta Copyright 2003 Robert G. Brown        #
#=============================================================================#
   rng_name    |rands/second|   Seed   |
   mt19937_1999|  1.13e+08  | 756452601|
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
   diehard_birthdays|   0|       100|     100|0.55765071|  PASSED
      diehard_operm5|   5|   1000000|     100|0.07483668|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.44568026|  PASSED
    diehard_rank_6x8|   0|    100000|     100|0.75054239|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.99487258|  PASSED
        diehard_opso|   0|   2097152|     100|0.10275334|  PASSED
        diehard_oqso|   0|   2097152|     100|0.43488786|  PASSED
         diehard_dna|   0|   2097152|     100|0.25090850|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.88790756|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.95753456|  PASSED
 diehard_parking_lot|   0|     12000|     100|0.65917069|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.08336595|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.87835233|  PASSED
     diehard_squeeze|   0|    100000|     100|0.76354402|  PASSED
        diehard_sums|   0|       100|     100|0.53274805|  PASSED
        diehard_runs|   0|    100000|     100|0.53049714|  PASSED
        diehard_runs|   0|    100000|     100|0.97339502|  PASSED
       diehard_craps|   0|    200000|     100|0.98271979|  PASSED
       diehard_craps|   0|    200000|     100|0.73525249|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.96404339|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.97777365|  PASSED
         sts_monobit|   1|    100000|     100|0.43559861|  PASSED
            sts_runs|   2|    100000|     100|0.02302364|  PASSED
          sts_serial|   1|    100000|     100|0.78088329|  PASSED
...
          sts_serial|  10|    100000|     100|0.95359118|  PASSED
          sts_serial|  11|    100000|     100|0.99993826|  FAILED
          sts_serial|  11|    100000|     100|0.47964300|  PASSED
          sts_serial|  12|    100000|     100|0.96998477|  PASSED
...
          sts_serial|  16|    100000|     100|0.61927046|  PASSED
         rgb_bitdist|   1|    100000|     100|0.05454541|  PASSED
...
         rgb_bitdist|  12|    100000|     100|0.60988172|  PASSED
rgb_minimum_distance|   2|     10000|    1000|0.08862860|  PASSED
```

```
...
rgb_minimum_distance|   5|     10000|     1000|0.15812548|  PASSED
    rgb_permutations|   2|    100000|      100|0.55135702|  PASSED
...
    rgb_permutations|   5|    100000|      100|0.69224297|  PASSED
      rgb_lagged_sum|   0|   1000000|      100|0.98183457|  PASSED
      rgb_lagged_sum|   1|   1000000|      100|0.10266242|  PASSED
      rgb_lagged_sum|   2|   1000000|      100|0.98338869|  PASSED
      rgb_lagged_sum|   3|   1000000|      100|0.76848949|  PASSED
      rgb_lagged_sum|   4|   1000000|      100|0.99816656|   WEAK
...
      rgb_lagged_sum|  32|   1000000|      100|0.99313787|  PASSED
```

Note well that *Dieharder fails* the mt13997_1999 generator on one test and returns a weak result on another – careful examination of the complete run suffices to *reject* $\mathcal{H}_0$. This is a *typical* result for this generator which is one of the theoretically *strongest* generators available, and demonstrates the power of *Dieharder*'s test series to resolve very subtle weaknesses – in this case the generator is measurably *too uniform* for true randomness, especially for certain ntuple sizes. 24 out of a total of 106 p-values generated in the test run above (each of which represents a Kolmogorov-Smirnov test based on at least 100 p-values for a given test and/or ntuple size) exceeded 0.9.

While *Dieharder* is arguably the most useful suite of RNG testing tools ever made openly available in prebuilt or ready-to-compile-and-run form, it is still *far* from complete or even fully adequate as a spanning set of RNG tests. This is especially true in the general arena of *cryptographic* quality tests, as the demands of cryptographic applications (which often have serious attendant privacy issues or monetary penalties for failure) are far greater than the more modest needs for uniformity and randomness in most Monte Carlo simulation applications[20].

Adding tests to *Dieharder* from third party sources and validating the *test suite itself* is hard work in every sense of the word and very time consuming. This work grows with the tool – it is necessary to revalidate the entire suite every time a test is added as well as to run the new test on a wide range of "presumed good" RNGs (as well as a few "known bad" ones) many, many times to be reasonably certain that the test itself is valid and conclusions drawn from it to be trusted. This work is hampered by the lack of a "gold standard" RNG that is both fast and known to be "truly" random. It requires a fine degree of judgment and a lot of effort to decide whether an apparent failure is due to a weakness in the test code or the RNG being tested.

The PI plans to add at *least* the tests in Table 2 to *Dieharder* as rapidly as possible, ideally with directed student labor contributing to the development process both to speed up the process and to educate selected undergraduate students in physics, mathematics, or the advanced computational statistical concepts involved. The tests from the *STS*[10] suite are for the most part bit-level tests targeted at validating RNGs for cryptographic purposes. Others are variants of tests from Knuth[2], tests available in open source form[21], or new tests described in the literature, especially by L'Ecuyer[15], Wegenkittl[22],[23] and in the *SPRNG* test documentation[14].

Test Name	Status	Remarks
STS Frequency within a Block	Not Installed	
STS Longest-Run-of-Ones in a Block	Not Installed	
STS Binary Matrix Rank	Not Installed	
STS Discrete Fourier Transform (Spectral)	Not Installed	
STS Non-overlapping Template Matching	Not Installed	
STS Overlapping Template Matching	Not Installed	
STS Maurer's "Universal Statistical"	Not Installed	
STS Lempel-Ziv Compression	Not Installed	
STS Linear Complexity	Not Installed	
STS Approximate Entropy	Not Installed	
STS Cumulative Sums (Cusums)	Not Installed	
STS Random Excursions	Not Installed	
STS Random Excursions Variant	Not Installed	
EST χ^2	Not Installed	
EST Uniform values	Not Installed	
EST Gaps	Not Installed	
EST Run Lengths	Not Installed	
EST Multiply Gap Lengths	Not Installed	
EST Occurrence	Not Installed	
Heinrich Maximum Spacing	Not Installed	
Wegenkittl Overlapping M-tuple	Not Installed	
Wegenkittl Problem Matching Tests	Not Installed	
Wegenkittl Entropy	Not Installed	
Marsaglia and Tsang Gorilla	Not Installed	
Marsaglia and Tsang Birthday Spacings	Not Installed	
L'Ecuyer Entropy	Not Installed	
L'Ecuyer *TestU01* Tests	Not Installed	
Knuth Tests	Not Installed	
SPRNG Tests	Not Installed	
RNGTS Tests	Not Installed	
Monte Carlo Applications	Not Installed	

Table 2: Tests to be added to *Dieharder*

The last item listed above is there to realize the design goal of providing e.g. Monte Carlo researchers with tests that are *very similar* to the intended end use of the RNG in question to provide them with the greatest possible confidence that the RNG they select will give them the correct answer instead of an incorrect one even if it "fails" some aspect of *Dieharder*'s rigorous testing. To further support the end user of RNGs, *Dieharder* automatically provides a *timing benchmark* for RNGs that are "integrated" with dieharder (in the GSL already or added in source form in the GSL-style RNG harness provided). Timing is almost as important as accuracy in the selection of the optimal RNG for a simulation that is expected to run for months to years.

One sign of a project that contains a significant research component is that one cannot know precisely what path the work will take. This is especially true of open source projects with a research component *and* the direct engagement of dozens of workers in the field. It is quite certain that during the course of this project new tests will be discovered, redundancies in the list above will be eliminated, and better ways of organizing or applying the existing tests will emerge from the active community of individuals using *Dieharder* and from the work done by the PI and his research assistants with the support of this grant.

Dieharder **Work Plan**

The worked planned for the period of requested support can be broken up into several categories in order of decreasing difficulty:

- Theoretical/Mathematical work, for example deriving a precise form for the "corrected" permutations test for overlapping sequences of N successive numbers or deriving suitable variations of the tests that search the seed space for global patterns of test results (a common test "fingerprint" for a given RNG, as it were).
- "Advanced" computational work, such as high level interface design, the parallelization of *Dieharder*, or the implementation of the aforementioned redesigned overlapping permutations test, where especially careful testing and validation is clearly called for and where the linear algebra is nontrivial.
- Writing documentation. This requires considerable expertise and experience with both testing and applying random numbers as well as the ability to write clearly, a fairly uncommon combination.
- More mundane work "porting" generators (or RNGs) into *Dieharder* either from their published descriptions (when adequate) or from actual code, when freely available and unencumbered by a restrictive license.
- Support work – debugging and fixing existing tests, helping users get started, packaging *Dieharder* for major operating systems and compute platforms.
- Communicating the existence of the tool itself as well as the results of its application to the available suite of RNGs (such as the interesting result presented above for the Mersenne Twister) to the broad multidisciplinary community of RNG users.

It is anticipated that the PI will perform all of the work done in the first two categories, certainly in the first year of support, as well as the bulk of the third. However, the project calls for hiring part-time undergraduate (student) labor (1/4 FTE/year) to help with at least the last three items on the list and, as the expertise of the students participating grows, it may be that they will be able to contribute to the higher-level work as well.

Impact of the Work

We conclude with a brief discussion of the expected impact of the work. To date, work on RNG testing has not been directed at the specific goal of producing a general purpose tool that can easily be packaged, has a simple and extensible user interface, is available as a linkable library so that it can easily be integrated with third-party tools such as *R*, and that is freely available to all users in all communities that need to test RNGs.

Although *Diehard* was originally developed with NSF support, it was subsequently distributed on a copyrighted CD-ROM and from a website that prominently displays a copyright notice on the toplevel page, but without *any* copyright statement or license contained in the code itself. It has thus never been clear whether or not the source code one can download can be freely modified or redistributed by both academic and commercial users. In private communications with Marsaglia it was indicated that *Diehard* was intended to be free for use in academic research but *not* free to commercial users; not an unfamiliar pattern but one that is ultimately self-defeating, especially when the code is actually distributed without any license statement at all so that one must basically ask the author to find out.

The *STS*, as an "internal" National Institute of Standards and Technology (NIST) project, is absolutely open but is much narrower than *Diehard* in its scope, developed primarily for *cryptographic* testing. Although it is quite well documented and is available in source, its user interface is clumsy and real packaging is nonexistent.

L'Ecuyer's recent *TestU01* suite is very sophisticated and quite well documented, but has a license that explicitly excludes its free use by commercial interests. It, too, is not packaged for general purpose use and requires considerable programming expertise simply to build.

The *EST* tests, while explicitly public domain, are in a simple unpackaged source format. The sources are not terribly easy to find on the Internet or build once they are found and are (as the sources themselves make clear) completely unsupported at this point. They have never (as far as the PI can tell) been subjected to a systematic validation process and are a bit cavalier in the way they define "success" versus "failure" in purely statistical terms.

Wegenkittl's papers on RNG testing and the tests described therein (especially his work on entropy based tests[22] and serial tests[23]) are wonderful resources and have been of great use to the *Dieharder* project PI but are extremely challenging mathematically and therefore a bit beyond the reach of many people that *use* RNGs but who are not prepared to write and test extremely sophisticated code from a description in the literature just to test them. Thus the greatest obstacle to the widespread use of Wegenkittl's tests is that they are not *implemented* in a readily available open source package with a consistent and simple test interface.

The SPRNG tests[14] are very definitely a move in the right direction and are openly distributed – but without either copyright notice or license, making it unclear whether or not modification of source code is permitted, whether the source code or binaries can be redistributed, and so on. They are also somewhat limited in the set of tests available at this time. However, there is a very definite opportunity there for collaborative work to be done integrating SPRNG methodologies with *Dieharder*, especially if the SPRNG project can be persuaded to GPL their code or permit *Dieharder* to release a GPL implementation of their code.

Providing a significant subset of these tests in a single, consistent, open and *free* packaging to all of the many users of RNGs is *already* having an impact, although the project is still far from being broadly known. Support requests, feature requests, bug reports, suggestions, and simple notes of thanks from new users arrive every month. Evidence of the impact is also available more broadly. For example, in the Wikipedia entry for the *Diehard* suite, *Dieharder* is cited as the fully GPL alternative that contains all of the same tests. It is similarly cited in the entry for pseudorandom number generators. These citations were independently added by users of *Dieharder*, not the PI.

However, when the work planned in this proposal is complete *all* of these tests (and more) will be accessible in a single, universal testing tool. It will also be easy to add new tests (and RNGs) as they are invented to the tool, have them validated *in use* by the broad random number consuming community, and then either be accepted into the "standardized" portion of the test suite or into the GSL itself if appropriate, or simply be made available as a user choice if they appear useful but only in certain contexts. Never again will anyone (academic or commercial end user alike) need to hesitate before downloading or modifying source code that was paid for, ultimately, with taxpayer dollars, or need to worry that by doing so they will be violating a copyright or license agreement.

In the end, if a *single* federally funded massive simulation project is saved from making a poor choice for its base RNG because of flaws revealed *at similar scale and in a similar programming context* by a fully parallelized *Dieharder*, any funding granted to this project will more than pay for itself. All of the benefits derived by other potential and actual users in academic research, cryptography, business, and government security will be pure "profit".

To conclude, there are two other benefits that should result from the funding of the project that, while perhaps not sufficiently valuable to justify its funding in and of themselves, are worthy of mention. The engagement of undergraduates in the *Dieharder* project has been and will continue to be a *significant benefit* of the proposed work. The undergraduates who participate leave the project with a far deeper knowledge of statistics, linear algebra, simulations in physics, cryptography, and other academic fields that use random numbers in large quantities than they entered it with *and* may have one or more publications to their name and/or opportunities to present their own work to the academic community in person. This knowledge and experience can only benefit them, and the greater community, later on in their careers.

The second is that it is the intention of the PI to if at all possible provide *very general* documentation that as far as is practical presents all of the actual theory underlying *each and every test*. A template has already been created for producing this documentation as a *book*, one that can equally well serve as a textbook for an undergraduate course in statistics that focuses on the use of random numbers and testing of RNGs or as a manual for the tool itself. Never again should it be possible for a user of one of the tests to wonder just where the target numbers used in the tests come from. By providing this documentation in an *open* format – both in freely downloadable form and in a moderately priced printed form – the statistics community will be able to readily see where the tests come from and how they work, which can only encourage their *correction* where the best efforts of the PI prove to be inadequate and inevitable bugs or errors creep into the work.