

Robert G. Brown's General Tools Page



Things on the site itself that may be of interest to students or philosophers of any age or generation include complete online books of poetry, various support materials for the study of physics, and links related to beowulfery. All materials on this site that are authored by Robert G. Brown are Copyright 2004. The details of their Open Public License (modified) can be viewed [here](#). If you use or enjoy anything at all on this site -- free textbooks, stories, programs, or other resources, consider hitting [to help spread the word so others can find it as well](#). Note, Robert G. Brown is generally either `rgb` or `rgbatduke` on many external sites crosslinked here.



My Books:

Home	Top	Flashcard Program	DieHarder Program	Benchmaster Program	Jove (editor) Program	The C Book	The Tao of Programming	Your Brain: a User's Manual (draft)	CVS Mini Howto
C project template	PVM project template	LaTeX project template	HOWTO project template	Latex Manual (online)	random_pvm demo/template	The yum HOWTO (draft)	Yum Article	Contact	About
Amazon Sales Rank Toolset	A Theorem Concerning God								

Site Links

[Home](#)
[Home](#)
[Lilith](#)
[Class](#)
[Beowulf](#)
[Research](#)
[General](#)
[Poetry](#)
[Prose](#)
[Philosophy](#)
[Search](#)
[Contact](#)
[About](#)

Webalyze

[Home](#)
[Class](#)
[Beowulf](#)
[Research](#)
[General](#)
[Poetry](#)
[Prose](#)
[Philosophy](#)
[Misc](#)

[Brahma \(webalize\)](#)
[DBUG \(webalize\)](#)
[DULUG](#)
[Linux@Duke](#)

dieharder

by
Robert G. Brown
Duke University Physics Department
Durham, NC 27708-0305
Copyright Robert G. Brown, 2025

Abstract

Dieharder: A Random Number Test Suite

Version 3.31.1

Robert G. Brown (rgb)

Dirk Eddelbuettel

David Bauer

Welcome to the dieharder distribution website.

Version 3.29.4beta is the current snapshot. Some of the documentation below may not quite be caught up to it, but it should be close.

Dieharder is a *random number generator (rng) testing suite*. It is intended to test *generators*, not *files of possibly random numbers* as the latter is a fallacious view of what it means to be random. Is the number 7 random? If it is generated by a random process, it might be. If it is made up to serve the purpose of some argument (like this one) it is not. Perfect random number generators produce "unlikely" sequences of random numbers -- at exactly the right average rate. Testing a rng is therefore quite subtle.

dieharder is a tool designed to permit one to push a weak generator to unambiguous failure (at the e.g. 0.0001% level), not leave one in the "limbo" of 1% or 5% maybe-failure. It also contains many tests and is extensible so that eventually it will contain many more tests than it already does.

If you are using dieharder for testing rngs either in one of its prebuilt versions (rpm or apt) or built from source (which gives you the ability to e.g. add more tests or integrate your rng directly with dieharder for ease of use) you may want to join either or both of the [dieharder-announce](#) or the [dieharder-devel](#) mailing lists here. The former should be very low traffic -- basically announcing when a snapshot makes it through development to where I'm proud of it. The latter will be a bit more active, and is a good place to post bug reports, patches, suggestions, fixes, complaints and generally participate in the development process.

About Dieharder

At the suggestion of Linas Vepstas on the Gnu Scientific Library (GSL) list this GPL'd suite of random number tests will be named "Dieharder". Using a movie sequel pun for the name is a

Contents

Document Type	Size (K)	Last Modified
HTML (page)	N/A	06/27/24
dieharder/dieharder-2.24.1-0.i386.rpm	94	06/27/24
dieharder/dieharder-2.24.1-0.src.rpm	539	06/27/24
dieharder/dieharder-2.24.1-1.i386.rpm	97	06/27/24
dieharder/dieharder-2.24.1-1.src.rpm	427	06/27/24
dieharder/dieharder-2.24.2-0.i386.rpm	96	06/27/24
dieharder/dieharder-2.24.2-0.src.rpm	427	06/27/24
dieharder/dieharder-2.24.3-0.i386.rpm	96	06/27/24
dieharder/dieharder-2.24.3-0.src.rpm	519	06/27/24
dieharder/dieharder-2.24.4-0.i386.rpm	97	06/27/24
dieharder/dieharder-2.24.4-0.src.rpm	778	06/27/24
dieharder/dieharder-2.24.7-0.i386.rpm	96	06/27/24
dieharder/dieharder-2.24.7-0.src.rpm	886	06/27/24
dieharder/dieharder-2.24.7-1.i386.rpm	96	06/27/24
dieharder/dieharder-2.24.7-1.src.rpm	897	06/27/24
dieharder/dieharder-2.27.10-1.src.rpm	1382	06/27/24
dieharder/dieharder-2.27.10-1.x86_64.rpm	102	06/27/24
dieharder/dieharder-2.27.11-1.src.rpm	1389	06/27/24
dieharder/dieharder-2.27.11-1.x86_64.rpm	113	06/27/24
dieharder/dieharder-2.27.12-1.i386.rpm	112	06/27/24
dieharder/dieharder-2.27.12-1.src.rpm	653	06/27/24
dieharder/dieharder-2.27.12-1.x86_64.rpm	114	06/27/24

double tribute to George Marsaglia, whose "[Diehard battery of tests](#)" of random number generators has enjoyed years of enduring usefulness as a test suite.

The dieharder suite is more than just the diehard tests cleaned up and given a pretty GPL'd source face in native C. Tests from the [Statistical Test Suite \(STS\)](#) developed by the National Institute for Standards and Technology (NIST) are being incorporated, as are new tests developed by rgb. Where possible or appropriate, *all* tests that can be parameterized ("cranked up") to where failure, at least, is unambiguous are so parameterized and controllable from the command line.

A further design goal is to provide some indication of *why* a generator fails a test, where such information can be extracted during the test process and placed in usable form. For example, the bit-distribution tests should (eventually) be able to display the actual histogram for the different bit ntuplets.

Dieharder is by design extensible. It is intended to be the "Swiss army knife of random number test suites", or if you prefer, "the last suite you'll ever ware" for testing random numbers.

Dieharder Related Talks or Papers

- [TechExpo 2011 Talk \(Duke\)](#). A short talk given at a Duke's Tech Expo in 2011 as an overview of random number generator testing. Good for beginners.
- [Good Practice in \(Pseudo\) Random Number Generation for Bioinformatics Applications](#) by David Jones, UCL Bioinformatics Group (E-mail: d dot jones@cs dot ucl dot ac dot uk). A really excellent "must read" guideline for anyone thinking of using random number generators in an actual application. My own advice differs only in that I endorse using (well tested) Gnu Scientific Library random number generators as they are generally portable and open source, hence well tested. Several of Jones' implementation of Marsaglia's KISS-family rngs have been added to dieharder and will shortly be added to the GSL under the GPL for general use.

Dieharder Download Area

Dieharder can be freely downloaded from [the Dieharder download site](#). On this page there should be a long list of previous versions of dieharder, and it should tell you what is the current snapshot. The version numbers have the following *specific meaning* which is a bit different than usual:

- First number (major). Bumped only when major goals in the design roadmap are reached (for example, finishing all the diehard tests). Version 1.x.x, for example, means that ALL of diehard (and more) is now incorporated in the program. Version 2.x.x means that the tests themselves have been split off into the libdieharder library, so that they can be linked into scripting languages such as R, new UIs, or user code. 3.x.x would be expected to indicate that the entire STS suite is incorporated, and so on.
- Second number (first minor). This number indicates the number of tests currently supported. When it bumps, it means new tests have been added from e.g. STS, Knuth, Marsaglia and Tsang, rgb, or elsewhere.
- Third number (second minor). This number is bumped when significant features are added or altered. Bug fixes bump this number, usually after a few bumps of the release number for testing snapshots. This number and the release are reset to 0 when the major is bumped or a new test is added to maintain the strictly increasing numerical value on which e.g. yum upgrades rely.

The single-tree dieharder sources (.tgz and .src.rpm) files can be downloaded from this directory. In addition, binary rpm's built on top of Fedora Core whatever (for either i386 or both of x86_64) may be present. Be warned: the GSL is a build *requirement*. The current packaging builds both the library and the dieharder UI from a single source rpm, or from running "make" in the toplevel directory of the source tarball. With a bit of effort (making a private rpm building tree), "make rpm" should work for you as well in this toplevel directory.

This project is under very active development. Considerable effort is being expended so that the suite will "run out of the box" to produce a reasonably understandable report for any given random number generator it supports via the "-a" flag, in addition to the ability to considerably vary most specific tests as applied to the generator. A brief synopsis of command options to get you started is presented below. In general, though, documentation (including this page, the man page, and built-in documentation) may lag the bleeding edge snapshot by a few days or more.

An rpm installation note from Court Shrock:

I was reading about your work on dieharder. First, some info about getting dieharder working in Gentoo:

```
cd ~
emerge rpm gsl
wget
http://www.phy.duke.edu/~rgb/General/dieharder/dieharder-0.6.11-1.i386.rpm
rpm -i --nodeps dieharder-0.6.11-1.i386.rpm
```

Rebuilding from tarball source should always work as well, and if you are planning to play a lot with the tool may be a desireable way to proceed as there are some documentation goodies in the ./doc subdirectory and the ./manual subdirectory of the source tarball (such as the original diehard test descriptions and the STS white paper).

George Marsaglia retired from FSU in 1996. For a brief time diehard appeared to have finally disappeared from FSU webspace, but what had really happened is google's favorite path to it

dieharder/dieharder-2.27.13-1.src.rpm	734	06/27/24
dieharder/dieharder-2.27.13-1.x86_64.rpm	114	06/27/24
dieharder/dieharder-2.28.1-1.src.rpm	746	06/27/24
dieharder/dieharder-2.28.1-1.x86_64.rpm	115	06/27/24
dieharder/dieharder-2.6.24-1.i386.rpm	62	06/27/24
dieharder/dieharder-2.6.24-1.src.rpm	793	06/27/24
dieharder/dieharder-3.29.1beta-1.src.rpm	856	06/27/24
dieharder/dieharder-3.29.1beta-1.x86_64.rpm	106	06/27/24
dieharder/dieharder-3.29.2beta-1.src.rpm	1012	06/27/24
dieharder/dieharder-3.29.2beta-1.x86_64.rpm	106	06/27/24
dieharder/dieharder-3.29.4beta-1.src.rpm	969	06/27/24
dieharder/dieharder-3.29.4beta-1.x86_64.rpm	109	06/27/24
dieharder/dieharder-3.31.0-1.src.rpm	1105	06/27/24
dieharder/dieharder-3.31.0-1.x86_64.rpm	108	06/27/24
dieharder/dieharder-3.31.1-1.src.rpm	1115	06/27/24
dieharder/dieharder-3.31.1-1.x86_64.rpm	126	06/27/24
dieharder/dieharder.src.rpm	1115	06/27/24
dieharder/libdieharder-2.24.1.0.i386.rpm	171	06/27/24
dieharder/libdieharder-2.24.1.1.i386.rpm	173	06/27/24
dieharder/libdieharder-2.24.2.0.i386.rpm	173	06/27/24
dieharder/libdieharder-2.24.3.0.i386.rpm	173	06/27/24
dieharder/libdieharder-2.24.4.0.i386.rpm	172	06/27/24
dieharder/libdieharder-2.24.7.0.i386.rpm	187	06/27/24
dieharder/libdieharder-2.24.7.1.i386.rpm	189	06/27/24
dieharder/libdieharder-2.27.10-1.x86_64.rpm	196	06/27/24
dieharder/libdieharder-2.27.11-1.x86_64.rpm	207	06/27/24
dieharder/libdieharder-2.27.12-1.i386.rpm	203	06/27/24
dieharder/libdieharder-2.27.12-1.x86_64.rpm	207	06/27/24
dieharder/libdieharder-2.27.13-1.x86_64.rpm	209	06/27/24
dieharder/libdieharder-2.28.1-1.x86_64.rpm	211	06/27/24
dieharder/libdieharder-2.6.24-1.i386.rpm	156	06/27/24
dieharder/libdieharder-3.29.1beta-1.x86_64.rpm	224	06/27/24
dieharder/libdieharder-3.29.2beta-1.x86_64.rpm	238	06/27/24
dieharder/libdieharder-3.29.4beta-1.x86_64.rpm	241	06/27/24
dieharder/libdieharder-3.31.0-1.x86_64.rpm	259	06/27/24
dieharder/libdieharder-3.31.1-1.x86_64.rpm	278	06/27/24
dieharder/dieharder-2.24.1.tgz	424	06/27/24

had disappeared when his personal home directory was removed. Diehard is still there, at the URL <http://www.stat.fsu.edu/pub/diehard> as well as at a Hong Kong website. The source code of diehard itself is (of course) Copyright George Marsaglia but Marsaglia did not incorporate an explicit license into his code which muddles the issue of how and when it can be distributed, freely or otherwise. Existing diehard sources are *not directly incorporated into dieharder in source form* for that reason, to keep authorship and GPL licensing issues clear.

Note that the same is not true about data. Several of the diehard tests require that one use precomputed numbers as e.g. target mean, sigma for some test statistic. Obviously in these cases we use the same numbers as diehard so we get the same, or comparable, results. These numbers were all developed with support from Federal grants and have all been published in the literature, though, and should therefore be in the public domain as far as reuse in a program is concerned.

Note also that most of the diehard tests are *modified* in dieharder, usually in a way that should improve them. There are three improvements that were basically always made if possible.

- The number of test sample p-value that contribute to the final Kolmogorov-Smirnov test for the uniformity of the distribution of p-values of the test statistic is a variable with default 100, which is *much* larger than most diehard default values. This change alone causes many generators that are asserted to "pass diehard" to in fact fail -- any given test run generates a p-value that is acceptable, but the *distribution* of p-values is not uniform.
- The number of actual samples *within* a test that contribute to the single-run test statistic was made a variable when possible. This was generally possible when the target was an easily computable function of the number of samples, but a number of the tests have pre-computed targets for specific numbers of samples and that number cannot be varied because no general function is known relating the target value to the number of samples.
- Many of diehard's tests investigated overlapping bit sequences. Overlapping sequences are not independent and one has to account for covariance between the samples (or a gradually vanishing degree of autocorrelation between sequential samples with gradually decreasing overlap). This was generally done at least in part because it used file-based input of random numbers and the size of files that could reasonably be generated and tested in the mid-90's contained on the order of a million random deviates.

Unfortunately, some of the diehard tests that rely on weak inverses of the covariance matrices associated with overlapping samples seem to have errors in their implementation, whether in the original diehard (covariance) data or in dieharder-specific code it is difficult to say. Fortunately, it is no longer necessary to limit the number of random numbers drawn from a generator when running an integrated test, and non-overlapping versions of these same tests do not require any treatment of covariance. For that reason non-overlapping versions of the questionable tests have been provided where possible (in particular testing permutations and sums) and the overlapping versions of those tests are deprecated pending a resolution of the apparent errors.

In a few cases other variations are possible for specific tests. This should be noted in the built-in test documentation for that test where appropriate.

Aside from these major differences, note that the algorithms were independently written more or less from the test descriptions alone (sometimes illuminated by a look at the code implementations, but only to clear up just what was meant by the description). They may well do things in a different (but equally valid) order or using different (but ultimately equivalent) algorithms altogether and hence produce slightly different (but equally valid) results even when run on the *same data with the same basic parameters*. Then, there may be bugs in the code, which might have the same general effect. Finally, it is always possible that *diehard* implementations have bugs and can be in error. Your Mileage May Vary. Be Warned.

About Dieharder

The primary point of dieharder (like diehard before it) is to make it easy to time and test (pseudo)random number generators, both software and hardware, for a variety of purposes in research and cryptography. The tool is built entirely on top of the GSL's random number generator interface and uses a variety of other GSL tools (e.g. sort, erfc, incomplete gamma, distribution generators) in its operation.

Dieharder differs significantly from diehard in many ways. For example, diehard uses file based sources of random numbers exclusively and by default works with only roughly ten million random numbers in such a file. However, modern random number generators in a typical simulation application can easily need to generate 10^{18} or more random numbers, generated from hundreds, thousands, millions of different seeds in independent (parallelized) simulation threads, as the application runs over a period of months to years. Those applications can easily be sensitive to rng weaknesses that might not be revealed by sequences as short as 10^7 uints in length even with excellent and sensitive tests. One of dieharder's primary design goals was to permit tests to be run on very long sequences.

To facilitate this, dieharder *prefers* to test generators that have been wrapped up in a GSL-compatible interface so that they can return an *unbounded* stream of random numbers -- as many as any single test or the entire suite of tests might require. Numerous examples are provided of how one can wrap one's own random number generator so that it is can be called via the GSL interface.

Dieharder also supports file-based input three distinct ways. The simplest is to use the (raw binary) stdin interface to pipe a bit stream from *any* rng, hardware or software, through dieharder for testing. In addition, one can use "direct" file input of either raw binary or ascii formatted (usually uint) random numbers. The man page contains examples of how to do all

dieharder/dieharder-2.24.2.tgz	423	06/27/24
dieharder/dieharder-2.24.3.tgz	516	06/27/24
dieharder/dieharder-2.24.4.tgz	777	06/27/24
dieharder/dieharder-2.24.7.tgz	900	06/27/24
dieharder/dieharder-2.27.10.tgz	1415	06/27/24
dieharder/dieharder-2.27.11.tgz	1418	06/27/24
dieharder/dieharder-2.27.12.tgz	655	06/27/24
dieharder/dieharder-2.27.13.tgz	736	06/27/24
dieharder/dieharder-2.28.1.tgz	749	06/27/24
dieharder/dieharder-2.6.24.tgz	816	06/27/24
dieharder/dieharder-3.29.1beta.tgz	858	06/27/24
dieharder/dieharder-3.29.2beta.tgz	1017	06/27/24
dieharder/dieharder-3.29.4beta.tgz	972	06/27/24
dieharder/dieharder-3.31.0.tgz	1109	06/27/24
dieharder/dieharder-3.31.1.tgz	1122	06/27/24
dieharder/dieharder.tgz	1122	06/27/24

three of these things, and dieharder itself can generate sample files to use as templates for the appropriate formatting.

Note Well! Dieharder can consume a *lot* of random numbers in the course of running all the tests! To facilitate this, dieharder should (as of 2.27.11 and beyond) support large file (>2GB) input, although this is still experimental. Large files are clunky and relatively slow, and the LFS (large file system) in linux/gcc is still relatively new and may have portability issues if dieharder is built with a non-gcc compiler. It is therefore *strongly recommended* that both hardware and software generators be tested by being wrapped within the GSL interface by emulating the source code examples or that the pipe/stdin interface be used so that they can return an essentially unbounded rng stream.

Dieharder also goes beyond diehard in that it is deliberately extensible. In addition to implementing all of the diehard tests it is expected that dieharder will eventually contain all of the NIST STS and a variety of tests contributed by users, invented by the dieharder authors, or implemented from descriptions in the literature. As a true open source project, dieharder can eventually contain *all* rng tests that prove useful in one place with a consistent interface that permits one to apply those tests to many generators for purposes of comparison and validation of the *tests themselves* as much as the generators. In other words, it is intended to be a vehicle for the computer science of random number generation testing as well as a practical test harness for random number generators.

To expand on this, the development of dieharder was motivated by the following, in rough order of importance:

- To provide a readily available, rpm- or apt- installable **toolset** so that "consumers" of random numbers (who typically use *large* numbers of random numbers in e.g. simulation or other research) can test the generator(s) they are using to verify their quality or lack thereof.
- To provide a very **simple user interface** for that toolset for random number consumers. At the moment, this means a command line interface (CLI) that can easily be embedded in scripts or run repeatedly with different parameters. A graphical user interface (GUI) is on the list of things to do, although it adds little to the practical utility of the tool.
- To provide **lots of knobs and dials** and low level control for statistical researchers that want to study particular generators with particular tests in more detail. This includes full access to test sources -- no parameter or aspect of the test algorithms is "hidden" and needs to be taken on faith.
- To have the entire test code and documentation be fully **Gnu Public Licensed** and hence openly available for adaptation, testing, comment, and modification so that the testing suite itself becomes (over time) reliable.
- To be **extensible**. Dieharder provides a fairly **simple API** for adding new tests with a common set of low-level testing tools and a **common test structure** that leads (one hopes) to an *unambiguous* decision to accept or reject any given random number generator on the basis of any given test for a suitable choice of controllable test parameters.
- To allow all researchers to be able to directly test, in particular, the **random number generators interfaced with the GSL**. This is a deliberate design decision justified by the extremely large and growing number of random number generators prebuilt into the GSL and the ease of adding new ones (either contributing them to the project or for the sole purpose of local testing).
- To allow researchers that use e.g. *distributions* directly generated by GSL routines (which can in principle fail two ways, due to the failure of the underlying random number generator or due to a failure of the generating algorithm) to be able to directly validate their particular generator/distribution combination at the cost of implementing a suitable test in dieharder (using the code of existing tests as a template).
- To allow dieharder to be directly interfaced with **other tools and interfaces**. For example, dieharder can be directly called within the R interface, permitting its rngs to be tested and R-based graphics and tools to be used to analyze test results. Note well, however, that because it uses the GSL (which is GPL viral) dieharder itself is GPL viral and cannot be embedded directly into a non-GPL tool such as matlab. It can, of course, be used to generate *p-value data* that is passed on to matlab (or any other graphing or analysis tool)

Although this tool is being developed on Linux/GCC-based platforms, it should port with no particular difficulty to other Unix-like environments (at least ones that also support the GSL), with the further warning that certain features (in particular large file support) may require tweaking and that the dieharder authors may not be able to help you perform that tweaking.

Essential Usage Synopsis

If you compile the test or install the provided binary rpm's and run it as:

```
dieharder -a
```

it should run -a(ll) tests on the default GSL generator.

Choose alternative tests with -g number where:

```
dieharder -g -1
```

will list all possible numbers known to the current snapshot of the dieharder.

```
dieharder -l
```

should list all the tests implemented in the current snapshot of DieHarder. Finally, the venerable and time tested:

```
dieharder -h
provides a Usage synopsis (which can quite long) and
man dieharder

is the (installed) man page, which may or many not be completely up to date as the suite is
under active development. For developers, additional documentation is available in the toplevel
directory or doc subdirectory of the source tree. Eventually, a complete DieHard manual in
printable PDF form will be available both on this website and in /usr/share/doc/dieharder-*/.
```

List of Random Number Generators and Tests Available

List of GSL and user-defined random number generators that can be tested by dieharder:

```
#-----#
# dieharder version 3.29.4beta Copyright 2003 Robert G. Brown #
#-----#
#   Id Test Name      |   Id Test Name      |   Id Test Name #
#-----#
| 000 borosh13       | 001 cmrg          | 002 coveyou
| 003 fishman18     | 004 fishman20     | 005 fishman2x
| 006 gfsr4          | 007 knuthran      | 008 knuthran2
| 009 knuthran2002  | 010 lecuyer21    | 011 minstd
| 012 mrg            | 013 mt19937       | 014 mt19937_1999
| 015 mt19937_1998  | 016 r250           | 017 ran0
| 018 ran1            | 019 ran2           | 020 ran3
| 021 rand            | 022 rand48         | 023 random128-bsd
| 024 random128-glibc2 | 025 random128-libc5 | 026 random256-bsd
| 027 random256-glibc2 | 028 random256-libc5 | 029 random32-bsd
| 030 random32-glibc2 | 031 random32-libc5 | 032 random64-bsd
| 033 random64-glibc2 | 034 random64-libc5 | 035 random8-bsd
| 036 random8-glibc2  | 037 random8-libc5  | 038 random-bsd
| 039 random-glibc2   | 040 random-libc5   | 041 randu
| 042 ranf            | 043 ranlux          | 044 ranlux389
| 045 ranlxd1          | 046 ranlxd2         | 047 ranlxs0
| 048 ranlxs1          | 049 ranlxs2         | 050 ranmar
| 051 slatec           | 052 taus            | 053 taus2
| 054 taus13           | 055 transputer      | 056 tt800
| 057 uni              | 058 uni32           | 059 vax
| 060 waterman14        | 061 zuf             |
#-----#
| 200 stdin_input_raw  | 201 file_input_raw  | 202 file_input
| 203 ca                | 204 uvag            | 205 AES_OFB
| 206 Threefish_OFB      |
#-----#
| 400 R_wichmann_hill  | 401 R_marsaglia_multic | 402 R_super_duper
| 403 R_mersenne_twister | 404 R_knuth_taocp   | 405 R_knuth_taocp2
#-----#
| 500 /dev/random        | 501 /dev/urandom      |
#-----#
| 600 empty              |                      |
#-----#
```

Two "gold standard" generators in particular are provided to "test the test" -- AES_OFB and Threefish_OFB are both cryptographic generators and should be quite random. gfsr4, mt19937, and taus (and several others) are very good generators in the GSL, as well. If you are developing a new rng, it should compare decently with these generators on dieharder test runs.

Note that the stdin_input_raw interface (-g 200) is a "universal" interface. Any generator that can produce a (continuous) stream of presumably random bits can be tested with dieharder. The easiest way to demonstrate this is by running:

```
dieharder -S 1 -B -o -t 1000000000 | dieharder -g 75 -r 3 -n 2
```

where the first invocation of dieharder generates a stream of binary bits drawn from the default generator with seed 1 and the second reads those bits from stdin and tests them with the rgb bitdist test on two bit sequences. Compare the output to:

```
dieharder -S 1 -r 3 -n 2
```

which runs the same test on the same generator with the same seed internally. They should be the same.

Similarly the file_input generator requires a file of "cooked" (ascii readable) random numbers, one per line, with a header that describes the format to dieharder. Note Well! File or stream input rands (with any of the three methods for input) are delivered to the tests on demand, but if the test needs more than are available dieharder either fails (in the case of a stdin stream) or rewinds the file and cycles through it again, and again, and again as needed. Obviously this significantly reduces the sample space and can lead to completely incorrect results for the p-value histograms unless there are enough rands to run EACH test without repetition (it is harmless to reuse the sequence for different tests). **Let the user beware!**

List of the CURRENT fully implemented tests (as of the 08/18/08 snapshot):

```
#-----#
# dieharder version 3.29.4beta Copyright 2003 Robert G. Brown #
#-----#
Installed dieharder tests:
Test Number          Test Name          Test Reliability
-----#
-d 0                 Diehard Birthdays Test  Good
-d 1                 Diehard OPERM5 Test  Suspect
-d 2                 Diehard 32x32 Binary Rank Test  Good
-d 3                 Diehard 6x8 Binary Rank Test  Good
```

-d 4	Diehard Bitstream Test	Good
-d 5	Diehard OPSO	Good
-d 6	Diehard QSO Test	Good
-d 7	Diehard DNA Test	Good
-d 8	Diehard Count the 1s (stream) Test	Good
-d 9	Diehard Count the 1s Test (byte)	Good
-d 10	Diehard Parking Lot Test	Good
-d 11	Diehard Minimum Distance (2d Circle) Test	Good
-d 12	Diehard 3d Sphere (Minimum Distance) Test	Good
-d 13	Diehard Squeeze Test	Good
-d 14	Diehard Sums Test	Do Not Use
-d 15	Diehard Runs Test	Good
-d 16	Diehard Craps Test	Good
-d 17	Marsaglia and Tsang GCD Test	Good
-d 100	STS Monobit Test	Good
-d 101	STS Runs Test	Good
-d 102	STS Serial Test (Generalized)	Good
-d 200	RGB Bit Distribution Test	Good
-d 201	RGB Generalized Minimum Distance Test	Good
-d 202	RGB Permutations Test	Good
-d 203	RGB Lagged Sum Test	Good
-d 204	RGB Kolmogorov-Smirnov Test Test	Good

Full descriptions of the tests are available from within the tool. For example, enter:

```
rgb@lilith|B:1003>./dieharder -d 203 -h
OK, what is dtest_num = 203
=====
#           RGB Lagged Sums Test
# This package contains many very lovely tests.  Very few of them,
# however, test for lagged correlations -- the possibility that
# the random number generator has a bitlevel correlation after
# some fixed number of intervening bits.
#
# The lagged sums test is therefore very simple.  One simply adds up
# uniform deviates sampled from the rng, skipping lag samples in between
# each rand used.  The mean of tsamples samples thus summed should be
# 0.5*tsamples.  The standard deviation should be sqrt(tsamples/12).
# The experimental values of the sum are thus converted into a
# p-value (using the erf()) and a ks-test applied to psamples of them.
=====
```

Note that all tests have been independently rewritten from their description, and may be functionally modified or extended relative to the original source code published in the originating suite(s). This has proven to be absolutely necessary; dieharder stresses random number generator tests as much as it stresses random number generators, and tests with imprecise target statistics can return "failure" when the fault is with the test, not the generator.

The author (rgb) bears complete responsibility for these changes, subject to the standard GPL code disclaimer that the code *has no warranty*. In essence, yes it may be my fault if they don't work but using the tool is *at your own risk* and you can fix it if it bothers you and/or I don't fix it first.

Development Notes

All tests are encapsulated to be as standard as possible in the way they compute p-values from single statistics or from vectors of statistics, and in the way they implement the underlying KS and chisq tests. Diehard is now complete in dieharder (although two tests are badly broken and should not be used), and attention will turn towards implementing more selected tests from the STS and many other sources. A road map of sorts (with full supporting documentation) is available on request if volunteers wish to work on adding more GPL tests.

Note that a few tests appear to have stubborn bugs. In particular, the diehard operm5 test seems to fail all generators in dieharder. Several users have attempted to help debug this problem, and it tentatively appears that the problem is in the original diehard code and not just dieharder. There is extensive literature on overlapping tests, which are highly non-trivial to implement and involve things like forming the weak inverse of covariance matrices in order to correct for overlapping (non-independent) statistics.

A revised version of overlapping permutations is underway (as an rgb test), but is still buggy. A non-overlapping (rgb) permutations test is provided now that should test much the same thing at the expense of requiring more samples to do it.

Similarly, the diehard sums test appears to produce a systematically non-flat distribution of p-values for all rngs tested, in particular for the "gold standard" cryptographic generators aes and threefish, as well as for the "good" generators in the GSL (mt19937, taus, gfsr4). It seems very unlikely that all of these generators would be flawed in the same way, so this test also should not be used to test your rng.

Thoughts for the Future/Wish List/To Do

- Tests of GSL random distribution (as opposed to number) generators, as indirect tests of the generators that feed them.
- New tests, compressions of existing ones that are "different" but really the same. Hyperplane tests. Spectral tests. Especially the bit distribution test with user defineable lag or lag pattern (to look for subtle, long period correlations in the bit patterns produced).
- Collaborators. Co-developers welcome, as are contributions or suggestions from users. Note well that users have already provided critical help debugging the early code! Part of the point of a GPL project is that you are NOT at the mercy of a black box piece of code. If you are using dieharder and are moderately expert at statistics and random numbers and observe something odd, please help out!

Conclusions

I hope that even during its development, you find dieharder useful. Remember, it is fully open source, so you can freely modify and redistribute the code according to the rules laid out in the Gnu Public License (version 2b), which might cost you as much as a beer one day. In particular, you can easily add random number generators using the provided examples as templates, or you can add tests of your own by copying the general layout of the existing tests (working toward a p-value per run, cumulating (say) 100 runs, and turning the resulting KS test into an overall p-value). Best of all, you can look inside the code and see how the tests work, which may inspire you to create a new test -- or a new generator that can pass a test.

To conclude, if you have any interest in participating in the development of dieharder, be sure to let me know, especially if you have decent C coding skills (including familiarity with Subversion and the GSL) and a basic knowledge of statistics. I even have documents to help with the latter, if you have the programming skills and want to LEARN statistics. Bug reports or suggestions are also welcome.

Submit bug reports, etc. to

rgb at phy dot duke dot edu

License Info

The documents linked from this page are all provided under a modified Gnu License appropriate for the document type ([OPL](#) for text, [GPL](#) for software/source). Please read the relevant license(s) before redistributing the document(s) in any form -- an explicit agreement with the author is required for certain kinds of for-profit redistributions. In all cases the license makes the documents generally available for unlimited personal use and non-profit distributions (for example, linking or posting copies on a website, distributing paper copies to a class for free or at cost).

The author cherishes feedback. If you like or dislike the document(s) and would like to say so, wish to redistribute a version in any medium to be sold at a profit, would like to contribute or comment on material, or just want to say hi, feel free to [contact the author](#)

Home	Top	Flashcard Program	DieHarder Program	Benchmaster Program	Jove (editor) Program	The C Book	The Tao of Programming	Your Brain: a User's Manual (draft)	CVS Mini Howto
C project template	PVM project template	LaTeX project template	HOWTO project template	Latex Manual (online)	random_pvm demo/template	The yum HOWTO (draft)	Yum Article	Contact	About
Amazon Sales Rank Toolset	A Theorem Concerning God								

This page is maintained by Robert G. Brown: rgb@phy.duke.edu