Dieharder Usage and Results

Essential Usage Synopsis

Dieharder is autodocumenting and is simple to use right away. In fact, if you compile the test or installed the provided binary rpm's, you should be able to immediate run it by entering:

```
dieharder -a
```

This should run -a(ll) tests on the default GSL generator. Let's review the features of dieharder that document its usage; then you should be able to figure the rest out on your own.

Choose alternative random number generators with -g number. To learn what generators are available within your current version of the Gnu Scientific Library (GSL -- which can vary as more or added in future versions) enter:

```
dieharder -g -1
```

or enter `diehard` on a line by itself. Note that a couple of the generators listed are not in the GSL but were added by me. It is quite simple to add your own random number generators to be tested by using these as templates.

```
dieharder -l
```

should list all the tests implemented in the current snapshop of DieHarder.

Finally, the venerable and time tested:

```
dieharder -h
```

provides a Usage synopsis (which can quite long)

Also following the *nix tradition:

```
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 DieHarder manual in printable PDF form will be available both on this website and in /usr/share/doc/dieharder-*/ (when it is finished, that is).

Back to top

List of Currently Available Generators and Tests

Here is the list of generators that were available at the time this documentation was written. There are quite a lot of them, including many of the most famous and useful generators currently known!

```
rgb@lilith|B:1344>dieharder
              Listing available built-in gsl-linked generators:            |
 Id Test Name           | Id Test Name           | Id Test Name            |
===========================================================================|
  0 borosh13            |  1 cmrg                |  2 coveyou              |
  3 fishman18           |  4 fishman20           |  5 fishman2x            |
  6 gfsr4               |  7 knuthran            |  8 knuthran2            |
  9 lecuyer21           | 10 minstd              | 11 mrg                  |
 12 mt19937             | 13 mt19937_1999        | 14 mt19937_1998         |
 15 r250                | 16 ran0                | 17 ran1                 |
 18 ran2                | 19 ran3                | 20 rand                 |
 21 rand48              | 22 random128-bsd       | 23 random128-glibc2     |
 24 random128-libc5     | 25 random256-bsd       | 26 random256-glibc2     |
 27 random256-libc5     | 28 random32-bsd        | 29 random32-glibc2      |
 30 random32-libc5      | 31 random64-bsd        | 32 random64-glibc2      |
 33 random64-libc5      | 34 random8-bsd         | 35 random8-glibc2       |
 36 random8-libc5       | 37 random-bsd          | 38 random-glibc2        |
 39 random-libc5        | 40 randu               | 41 ranf                 |
 42 ranlux              | 43 ranlux389           | 44 ranlxd1              |
 45 ranlxd2             | 46 ranlxs0             | 47 ranlxs1              |
 48 ranlxs2             | 49 ranmar              | 50 slatec               |
 51 taus                | 52 taus2               | 53 taus113              |
 54 transputer          | 55 tt800               | 56 uni                  |
 57 uni32               | 58 vax                 | 59 waterman14           |
 60 zuf                 |
                   Listing available non-gsl generators:                   |
 Id Test Name           | Id Test Name           | Id Test Name            |
===========================================================================|
 61 /dev/random         | 62 /dev/urandom        | 63 empty                |
```

Note that the last three tests are examples of random number generators that have been wrapped up in GSL compatible clothes and linked to the GSL so that the standard GSL interface works for them. *Any* random number generator that one wishes to test can thus easily be added for testing using these as prototypes, and can likely be submitted to the GSL for inclusion if they pass the tests as well or better than the tests that are already there. That makes this a *very convenient tool* for testing new RNGs.

To list the current fully implemented tests, enter the following:

```
rgb@lilith|B:1346>dieharder -l

                     DieHarder Test Suite
========================================================================
The following tests are available and will be run when diehard -a is
invoked.  Special options or suggested parameters are indicated if
they are needed to get a satisfactory result (such as completion in a
reasonable amount of time).

            Diehard Tests
   -d 1 Diehard Runs test
   -d 2 Diehard Birthdays test (-t 100, or less than 200)
   -d 3 Diehard Minimum Distance (2D Spheres) test
   -d 4 Diehard 3D Spheres (minimum distance) test

             RGB Tests
   -r 1 Bit Persist test
   -r 2 Bit Ntuple Distribution test suite (-n ntuple for 1-8)

      Statistical Test Suite (STS)
   -s 1 STS Monobit test
   -s 2 STS Runs test

            User Tests
No user-developed test are installed at this time.
```

Full descriptions of the tests are available (as you can see) from within the tool and source documentation. All tests are completely and independently rewritten from their description alone, and may be functionally modified or extended relative to the original source code published in the originating suite. The author (rgb) bears complete responsibility for these changes, subject to the standard GPL code disclaimer (in essence, yes it's my fault if they don't work but using the tool is at your own risk).

Back to top

Screen Shots (sort of)

An example: The result of running the Diehard runs test on a generator that clearly passes (mt19937) and one that clearly fails (randu):

```
rgb@lilith|B:1350>dieharder -d 1 -g 12

#==================================================================
#                Diehard "runs" test (modified).
# This tests the distribution of increasing and decreasing runs
# of integers.  If called with reasonable parameters e.g. -s 100
# or greater and -n 100000 or greater, it will compute a vector
# of p-values for up and down and verify that the proportion
# of these values less than 0.01 is consistent with a uniform
# distribution.
#==================================================================
# Random number generator tested: mt19937
# size of vector tested = 10000 (100000 or more suggested)
# p = 0.102725 for diehard_runs test from Kuiper Kolmogorov-Smirnov
#     test on 200 pvalues (up runs + down runs).

rgb@lilith|B:1352>dieharder -d 1 -g 40

#==================================================================
#                Diehard "runs" test (modified).
# This tests the distribution of increasing and decreasing runs
# of integers.  If called with reasonable parameters e.g. -s 100
# or greater and -n 100000 or greater, it will compute a vector
# of p-values for up and down and verify that the proportion
# of these values less than 0.01 is consistent with a uniform
# distribution.
#==================================================================
# Random number generator tested: randu
# size of vector tested = 10000 (100000 or more suggested)
# p = 0.006163 for diehard_runs test from Kuiper Kolmogorov-Smirnov
#     test on 200 pvalues (up runs + down runs).
```

In the latter case, the value of p indicates that there is only a 0.6% chance that a perfect random number generator could have produced the observed distribution of p-values from 100 independent runs of 10000 samples each. It is thus unlikely (in a manner of speaking) that randu is a good random number generator, but it is still possible -- even a perfect generator *could* have produced the observed result, it just isn't likely. One advantage of dieharder is that one can easily crank up the number of samples per test (-t) to make failure certain:

```
rgb@lilith|B:1355>dieharder -d 1 -g 40 -t 1000000

#==================================================================
#                Diehard "runs" test (modified).
# This tests the distribution of increasing and decreasing runs
# of integers.  If called with reasonable parameters e.g. -s 100
# or greater and -n 100000 or greater, it will compute a vector
# of p-values for up and down and verify that the proportion
# of these values less than 0.01 is consistent with a uniform
# distribution.
#==================================================================
# Random number generator tested: randu
# size of vector tested = 1000000 (100000 or more suggested)
# p = 0.000000 for diehard_runs test from Kuiper Kolmogorov-Smirnov
#     test on 200 pvalues (up runs + down runs).
# Generator randu FAILS at 0.01% for diehard_runs.
```

randu *fails* this test. To be more precise, the probability that randu is a "good" generator but produced the observed distribution is less than 0.000001, according to this test. mt19937, on the other hand, still returns perfectly reasonable values of p from the final KS test, even when run repeatedly with still larger -t.

All the tests have a similar call format, and share control parameters to the extent possible. At this point I expect to live with the overall structure and encapsulation of the tests for a while (after completing a fairly major overhaul) and will spend time in the near term just adding tests.