RDieHarder: An R interface to the Die*Harder* suite of Random Number Generator Tests

Dirk Eddelbuettel
Debian
edd@debian.org

Robert G. Brown
Physics, Duke University
rgb@phy.duke.edu

Initial Version as of May 2007
Rebuilt on March 15, 2018 using RDieHarder 0.1.4

1 Introduction

Random number generators are critically important for computational statistics. Simulation methods are becoming ever more common for estimation; Monte Carlo Markov Chain is but one approach. Also, simulation methods such as the Bootstrap have long been used in inference and are becoming a standard part of a rigorous analysis. As random number generators are at the heart of the simulation-based methods used throughout statistical computing, 'good' random numbers are therefore a crucial aspect of a statistical, or quantitative, computing environment. However, there are very few tools that allow us to separate 'good' from 'bad' random number generators.

Based on work that started with the **random** package (Eddelbuettel, 2007) (which provides functions that access a non-deterministic random number generator (NDRNG) based on a physical source of randomness), we wanted to compare the particular NDRNG to the RNGs implemented in GNU R (R Development Core Team, 2007) itself, as well as to several RNGs from the GNU GSL (Galassi et al., 2007), a general-purpose scientific computing library. Such a comparison is possible with the Die*Harder* test suite by Brown (2007) which extends the DieHard test suite by Marsaglia. From this work, we became interested in making Die*Harder* directly accessible from GNU R. The **RDieHarder** package presented here allows such access.

This paper is organized as follows. Section 2 describes the history and design of the Die*Harder* suite. Section 3 describes the **RDieHarder** package facilities, and section 4 shows some examples. Section 5 discusses current limitations and possible extensions before section 6 concludes.

2 Die*Harder*

Die*Harder* is described at length in Brown (2006). Due to space limitations, this section cannot provide as much detail and will cover only a few key aspects of the DieHarder suite.

2.1 DieHard

Die*Harder* reimplements and extends George Marsaglia's *Diehard Battery of Tests of Randomness* (Marsaglia, 1996). Due to both its robust performance over a wide variety of RNGs, as well as an ability to discern numerous RNGs as weak, DieHard has become something close to a 'gold standard' for assessing RNGs.

However, there are a number of drawbacks with the existing DieHard test battery code and implementation. First, Marsaglia undertook a large amount of the original work a number of years ago when computing resources were, compared to today's standards, moderately limited. Second, neither the Fortran nor the (translated) C sources are particularly well documented, or commented. Third, the library design is not

modular in a way that encourages good software engineering. Fourth, and last but not least, no licensing statement is provided with the sources or on the support website.

This led one of the authors of this paper (rgb) to a multi-year effort of rewriting the existing tests from DieHard in a) standard C in a modular and extensible format, along with extensive comments, and to b) relicense it under the common and understood GNU GPL license (that is also used for GSL, R, the Linux kernel, and numerous other projects) allowing for wider use. Moreover, new tests from NIST were added (see next subsection) and some genuinely new tests were developed (see below).

2.2 STS

The National Institute of Standards and Technology (NIST) has developed its own test suite, the 'Statistical Test Suite' (STS). These tests are focussed on bit-level tests of randomness and bit sequences.

Currently, three tests based on the STS suite are provided by Die*Harder*: *STS Monobit*, *STS Runs* and *STS Block*.

2.3 RGB extensions

Three new tests have been developed by rgb. A fourth 'test' is a timing function: for many contexts, not only do the mathematical properties of a generator matter, but so does computational cost measured in computing time that is required for a number of draws.

2.4 Basic methodology

Let us suppose a random number generator can provides a sequence of N uniform draws from the range $[0, 1)$. As the number of draws increases, the mean of the sum of all these values should, under the null hypothesis of a proper generator, converge closer and closer to $\mu = N/2$. Each of these N draws forms one experiment. If N is sufficiently large, then the means of all experiments should be normally distributed with a standard deviation of $\sigma = \sqrt{N/12}$.[1] Given this asymptotic result, we can, for any given experiment $i \in 1, \ldots, M$ transform the given sum x_i of N draws into a probability value p_i using the inverse normal distribution.[2]

The key insight is that, under the null hypothesis of a perfect generator, these p_i values should be uniformly distributed. Using our set of M probability values, we can compute one 'meta-test' of whether we can reject the null of a perfect generator by rejecting that our M probability values are not uniformly distributed. One suitable test is for example the non-parametric Kolmogorov-Smirnov (KS)[3] statistic. Die*Harder* uses the Kuiper[4] variant of the KS test which uses the combination $D_+ + D_-$ of the maximum and minimum distance to the alternative distribution, instead of using just one of these as in the case of the KS test. This renders the test more sensitive across the entire test region.

2.5 GSL framework

Die*Harder* is primarily focussed on tests for RNGs. Re-implementing RNGs in order to supply input to the tests is therefore not an objective of the library. The GNU Scientific Library (GSL), on the other hand, provides over 1000 mathematical functions, including a large number of random number generators. Using the GSL 1.9.0 release, the following generators are defined[5]:

[1] This is known as the Irwin-Hall distribution, see http://en.wikipedia.org/wiki/Irwin-Hall_distribution.

[2] Running `print(quantile(pnorm(replicate(M,(sum(runif(N))-N/2)/sqrt(N/12))), seq(0,1,by=0.1))*100, digits=2)` performs a Monte Carlo simulation of M experiments using N uniform deviates to illustrate this. Suitable values are e.g. `N <- 1000; M <- 500`.

[3] C.f. the Wikipedia entry http://www.wikipedia.org/wiki/Kolmogorov-Smirnov_test.

[4] C.f. the Wikipedia entry http://www.wikipedia.org/wiki/Kuiper%27s_test.

[5] This is based on the trailing term in each identifier defined in `/usr/include/gsl/gsl_rng.h`.

```
borosh13 coveyou cmrg fishman18 fishman20 fishman2x gfsr4 knuthran knuthran2 knuthran2002 lecuyer21
minstd mrg mt19937 mt19937_1999 mt19937_1998 r250 ran0 ran1 ran2 ran3 rand rand48 random128_bsd
random128_glibc2 random128_libc5 random256_bsd random256_glibc2 random256_libc5 random32_bsd
random32_glibc2 random32_libc5 random64_bsd random64_glibc2 random64_libc5 random8_bsd ran-
dom8_glibc2 random8_libc5 random_bsd random_glibc2 random_libc5 randu ranf ranlux ranlux389
ranlxd1 ranlxd2 ranlxs0 ranlxs1 ranlxs2 ranmar slatec taus taus2 taus113 transputer tt800 uni
uni32 vax waterman14 zuf
```

The GNU GSL, a well-known and readily available library of high quality, therefore provides a natural fit for Die*Harder*. All of these generators are available in Die*Harder* via a standardized interface in which a generator is selected, parameterized as needed and the called via the external GSL library against which Die*Harder* is linked.

Beyond these GSL generators, Die*Harder* also provides two generators based on the 'devices' `/dev/random` and `/dev/urandom` that are commonly available on Unix. They provide non-deterministic random-numbers based on entropy generated by the operating system. Die*Harder* also offers a text and a raw file input generators. Lastly, a new algorithmic generator named 'ca' that is based on cellular automata has recently been added as well.

2.6 R random number generators

To assess the quality of the non-deterministic RNG provided in the GNU R add-on package **random**, benchmark comparisons with the generators provided by the R language and environment have been a natural choice. To this end, one of the authors (edd) ported the R generator code (taken from R 2.4.0) to the GNU GSL random number generator framework used by Die*Harder*. This allows a direct comparison of the **random** generator with those it complements in R.

It then follows somewhat naturally that the other generators available in Die*Harder*, as well as the Die*Harder* tests, should also be available in R. This provided the motivation for the R package presented here.

2.7 Source code and building Die*Harder*

Recent versions of Die*Harder* use the GNU autotools. On Unix system, the steps required to build and install Die*Harder* should only be the familiar steps `configure; make; sudo make install`.

For Debian, initial packages have been provided and are currently available at `http://dirk.eddelbuettel.com/code/tmp`. Within due course, these packages should be uploaded to Debian, and thus become part of the next Debian (and Ubuntu) releases. Die*Harder* is also expected to be part of future Fedora Core (and other RPM-based distribution) releases.

On Windows computers and other systems, manual builds should also be possible given that the source code is written in standard C.

3 RDieHarder

The **RDieHarder** package provides one key function: `dieharder`. It can be called with several arguments. The first one is the name of the random number generator, and the second one is the name of the test to be applied. For both options, the textual arguments are matched against internal vectors to obtain a numeric argument index; alternatively the index could be supplied directly. The remaining arguments (currently) permit to set the number of samples (i.e. the number of experiments run, and thus the sample size for the final Kolmogorov-Smirnov test), the random number generator seed and whether or not verbose operation is desired.

The returned object is of class `dieharder`, inheriting from the standard class `htest` common for all hypothesis tests. The standard print method for `htest` is used; however not all possible slots are being filled (as there is for example no choice of alternative hypothesis).

A custom summary method is provided that also computes the Kolmogorov-Smirnov and Wilcoxon tests in R and displays a simple stem-and-leaf plot. Lastly, a custom plot method shows both a histogram and kernel density estimate, as well as the empirical cumulative distribution function.

4 Examples

The possibly simplest usage of **RDieHarder** is provided in the examples section of the help page. The code `dh <- dieharder; summary(dh); plot(dh)` simply calls the `dieharder` function using the default arguments, invokes a summary and then calls plot on the object.[6]

A more interesting example follows below. We select the `2dsphere` test for the generators `ran0` and `mt19937` with a given seed. The results based on both the Kuiper KS test and the KS test suggest that we would reject `ran0` but not `mt19937`, which is in accordance with common knowledge about the latter (the Mersenne Twister) being a decent RNG. It is worth nothing that the Wilcoxon test centered on $\mu = 0.5$ would not reject the null at conventional levels for `ran0`.

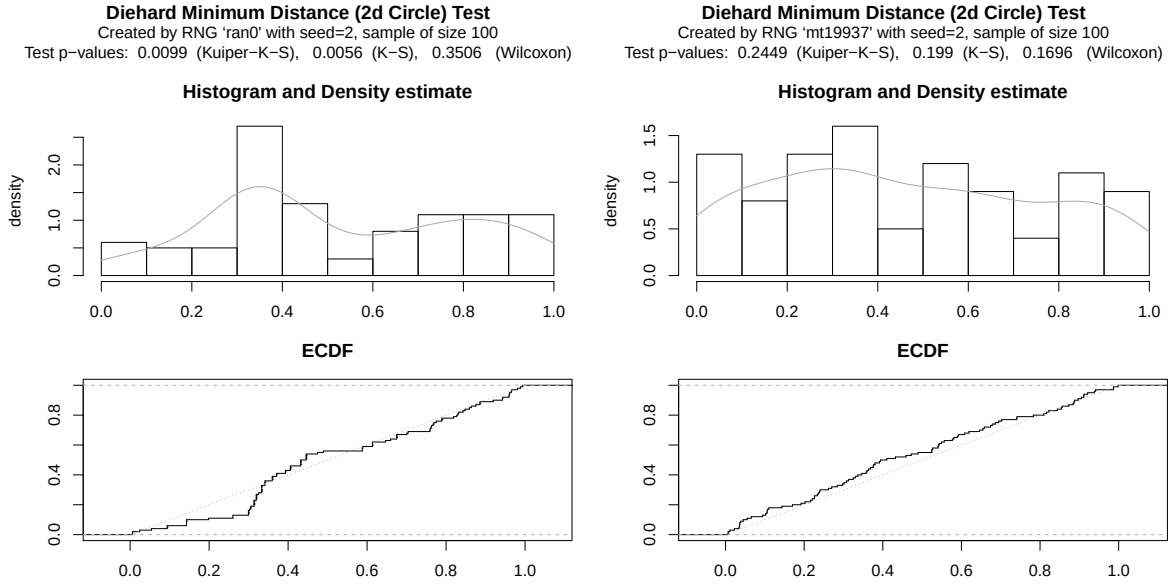

Figure 1: Comparison of `ran0` and `mt19937` under test `2dsphere`

A programmatic example follows. We define a short character vector containing the names of the six R RNGs, apply the Die*Harder* function to each of these, and then visualize the resulting p-values in simple qqplot.

All six generators provide p-value plots that are close to the ideal theoretical outcome (shown in gray). Unsurprisingly, p-values for the Kuiper KS test also show no support for rejecting these generators.

5 Current Limitations and Future Research

The implementation of **RDieHarder** presented here leaves a number of avenues for future improvement and research. Some of these pertain to Die*Harder* itself – adding new, more sophisticated, more systematic tests including those from the STS suite and tests that probe bitlevel randomness in unique new ways. Others pertain more to the integration of Die*Harder* with R, which is the topic of this work.

[6] We omit the output here due to space constraints.

```
> rngs <- c("R_wichmann_hill", "R_marsaglia_multic",
+           "R_super_duper", "R_mersenne_twister",
+           "R_knuth_taocp", "R_knuth_taocp2")
> if (!exists("rl")) rl <- lapply(rngs, function(rng) dieharder(rng, "diehard_runs", seed=12345))
> oldpar <- par(mfrow=c(2,3), mar=c(2,3,3,1))
> invisible(lapply(rl, function(res) {
+   qqplot(res$data, seq(0, 1, length.out=length(res$data)),
+          main=paste(res$generator, ":", round(res$p.value, digits=3)),
+          ylab="", type="S")
+   abline(0, 1, col='gray')
+ }))
> par(oldpar) # reset graph defaults
>
```

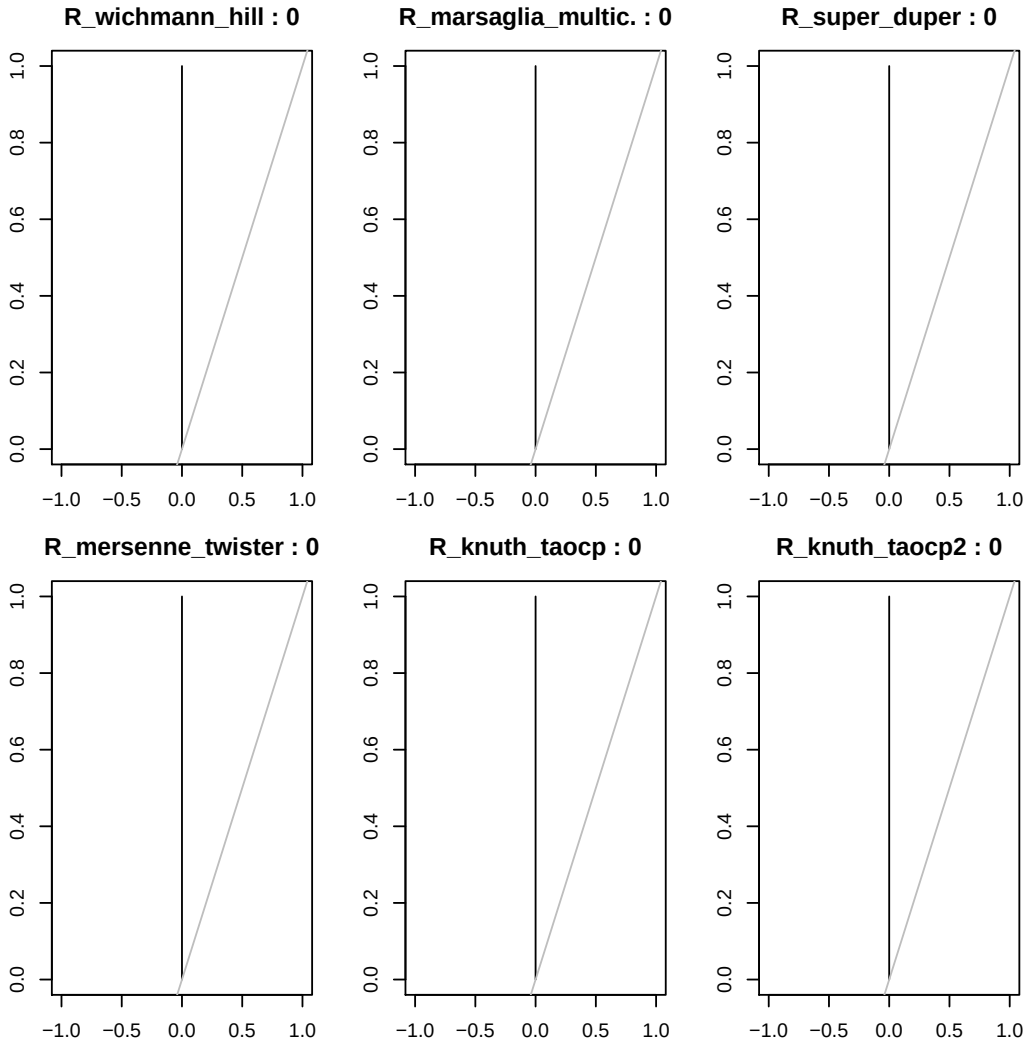

Figure 2: Comparing six GNU R generators under the `runs` test

Not all of Die*Harder*'s features are yet supported in this initial port. In the near future we expect to add code to deal with tests that support extra parameters, or that return more than one p-value per instance of a test. Ultimately, **RDieHarder** should support the full set of options of the the command-line version of Die*Harder*.

There is no direct interface from the R generators to the **RDieHarder** module for evaluation; rather, the 'ported' R generators are called from the libdieharder library. This could introduce coding/porting errors, and also prevents the direct use of user-added generators that R supports. It would be worthwhile to overcome this by directly letting **RDieHarder** call back into R to generate draws. On the other hand, the current setup corresponds more closely to the command-line version of Die*Harder*.

Next, the R generators in Die*Harder* may need to be updated to the 2.5.0 code. The GSL RNGs provided by libdieharder may as well be exported to R via **RDieHarder** given that the GSL library is already linked in. Indeed, it would be worthwhile to *integrate* the two projects and both avoid needless code duplication and ensure even more eyes checking both the quality and accuracy of the code in both.

It could be useful to also build **RDieHarder** with an 'embedded' libdieharder rather than relying on an externally installed libdieharder. This may make it easier to build **RDieHarder** for systems without libdieharder (and on Windows). Likewise, it is possible to reorganize the Die*Harder* front-end code into a common library to avoid duplication of code with **RDieHarder**.

Lastly, on the statistical side, an empirical analysis of size/power between KS, Wilcoxon and other alternatives for generating a final p-value from the vector of p-values returned from Die*Harder* tests suggests itself. Similarly, empirical comparisons between the resolving power of the various tests (some of which may not actually be terribly useful in the sense that they yield new information about the failure modes of any given RNG) could be undertaken. Lastly, there is always room for new generators, new tests, and new visualizations.

One thing that one should remember while experimenting with Die*Harder* is that there really is no such thing as a *random* number *generator*. It is therefore likely that *all* RNGs will fail any given (valid) test if one cranks up the "resolution" up high enough by accumulating enough samples per p-value, enough p-values per run.

It is also true that a number of Marsaglia's tests have target distributions that were computed empirically by simulation (with the best RNGs and computers available at the time). Here one has to similarly remember that one can do in a few hours of work what it would have taken him months if not years of simulation to equal back when the target statistics were evaluated. It is by no means unlikely that a "problem" that Die*Harder* eventually resolves is not not the quality of the RNG but rather the accuracy of the target statistics.

These are some of the things that are a matter for future research to decide. A major motivation for writing Die*Harder* and making it open source, and integrating it with *R*, is to facilitate precisely this sort of research in an easy to use, consistent testing framework. We *welcome* the critical eyes and constructive suggestions of the statstical community and invite their participation in examining the code and algorithms used in Die*Harder*.

6 Conclusion

The **RDieHarder** package presented here introduces several new features. First, it makes the Die*Harder* suite (Brown, 2007) available for interactive use from the GNU R environment. Second, it also exports Die*Harder* results directly to R for further analysis and visualization. Third, it adds adds additional RNGs from GNU R to those from GNU GSL that were already testable in Die*Harder*. Fourth, it provides a re-distribution of the Die*Harder* 'test engine' via GNU R.

References

Robert G. Brown. Die*Harder*: A Gnu public licensed random number tester. Draft paper included as file `manual/dieharder.tex` in the dieharder sources. Last version dated 20 Feb 2006., 2006.

Robert G. Brown. *dieharder: A Random Number Test Suite*, 2007. URL http://www.phy.duke.edu/~rgb/General/dieharder.php. C program archive **dieharder**, version 2.24.3.

Dirk Eddelbuettel. **random***: True random numbers using random.org*, 2007. URL http://cran.r-project.org/src/contrib/Descriptions/random.html. R package **random**, current version 0.1.2.

Mark Galassi, Brian Gough, Gerald Jungman, James Theiler, Jim Davies, Michael Booth, and Fabrice Rossi. *The GNU Scientific Library Reference Manual*, 2007. URL http://www.gnu.org/software/gsl. ISBN 0954161734; C program archive **gsl**, current version 1.9.0.

George Marsaglia. The Marsaglia random number CDROM including the diehard battery of tests of randomness. Also at http://stat.fsu.edu/pub/diehard., 1996.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. URL http://www.R-project.org. ISBN 3-900051-07-0.