CPRNGs as An Application of Chaos

Carter J. Bastian
Math 53, 15F

January 7, 2016

Purpose

The purpose of this project is to explore the use of chaotic pseudo-random number generators (CPRNGs) as an application of chaos theory. The main objective is to develop, test, and analyze an extensible[1] series of CPRNGs based in simple maps which can have chaotic orbits.

To this end, the software at
https://github.com/carterjbastian/chaotic-prngs
was developed and implemented in tandem with the dieharder test series[1] to analyze the validity of this use of chaos.

Background Information

Blackledge and Ptitsyn define a pseudo-random number generator (PRNG) as a "deterministic algorithm that, on input of a short seed (an initial condition), outputs a typically much longer sequence that is computationally indistinguishable from a uniformly chosen string"[3]. This definition spawns two important theoretical considerations that must be accounted for while designing CPRNGs.

First, it's important to differentiate between chaos and pseudo-chaos. Whereas chaotic maps consist of infinitely precise state variables, the orbits computed in any practical simulation must be represented with a finite binary length (in this case, 32 bits). This means that,

1. there is a finite number of states in any pseudo-chaotic system,
2. the result of successive iterations is an approximation subject to non-trivial roundoff error, and
3. the pseudo-chaotic system may be observed on a finite time scale or granularity[3].

This distinction is important because, as Blackledge and Ptitsyn state, pseudo-chaos is always a "poor approximation of chaos because the approximated model does not converge to the original model"[3]. This means both that systems that are theoretically chaotic may exhibit non-chaotic properties and that systems that are not formally chaotic may exhibit pseudo-chaotic properties.

This divergence between theory and application causes a series of challenges which must be taken into consideration whenever chaos theory is applied in a crypto-system. First and foremost, many of the tools used in theoretical chaos cease to be applicable. For example, while the analytic Lyapunov exponent – a measure of the rate at which orbits that are arbitrarily close to one another separate – of a chaotic map will be larger than the Lyapunov exponent of it's pseudo-chaotic counterpart [5][3].

Furthermore, the use of the Lyapunov exponent as a measure of a pseudo-chaotic system's predictability (or lack thereof) is flawed in that it does not consider the scope or resolution over which the system is observed [6]. As such, one needs to adopt an alternative measure based less in chaos theory and more in information and complexity theory [3][2].

Second, without a practical way to rigorously prove or check the indistinguishability of a long sequence from one uniformly chosen, statistical tests must be used instead to evaluate the quality of CPRNGs. Because of the inevitability of periodicity implied by the finite state space of pseudo-chaotic systems, it's important that the tests be applied with an extremely large resolution. An ideal pseudo-chaotic crypto-system would consist of a single periodic orbit iterating over the entire state space. However, for practical use of a CPRNG, we only need to require that the periodic orbit asymptotically reached from the initial condition be both long enough and structurally diverse enough as to not allow an arbitrarily large amount of knowledge of past states to yield significant insight into future behavior of the system.

The dieharder test suite was employed to provide some measurable degree of certainty regarding the practical usability strength of each developed CPRNG. The dieharder suite is a battery of tests designed to rigorously apply generators of possibly-random numbers to the point of failure [1]. As such, many of the tests performed requires a large amount of generated output. So as to not induce unnecessary and artificial periodicity into the tests (by allowing the input to the dieharder tests to repeat), 1,000,000,000 numbers were generated with each CPRNG for consumption by DieHarder[3].

Notes on Project Design

In order to accurately test a PRNG, the dieharder package requires a large sample of presumably-random 32-bit integers. These samples consist of 1,000,000,000 unsigned (32-bit) integers and can be found in the “output” directory of the source code.

Further, for reference, the corresponding trace files (in the “trace” directory of the source code) contain the floating-point values on the map's original interval (usually [0,1] unless otherwise noted in the generators.c internal documentation).

The maps implemented as PRNGs (listed and described fully in Appendix A) were chosen somewhat arbitrarily. Considerations included whether the map had been covered or seen throughout the term (such as the tent and logistic maps), the ease of computation (such as the sine map and the iterative map), or previous use in application[4].

For comparison, the industry standard random number generator – glibc's linear congruential PRNG implemented as rand() in the stdlib.h package – was tested alongside the newly developed CPRNGs. This design decision was based on the assumption that the standard could serve as an industry-grade benchmark with respect to its ability as a PRNG and it's performance metrics.

The conditions for the pseudo-chaotic systems were selected in the same manner as the seeds standard PRNGs in the GNU Scientific Library (GSL) tradition. This is because, in order for a PRNG to be usable, one must be able to seed across its entire range of possible initial conditions [7]. The implications of this for CPRNGs will be further discussed in the conclusions.

Experimental Results

In short, the CPRNGs consistently performed quite well on a small subset of the dieharder tests. There were a few outlying CPRNGs which performed quite well outside of this subset of consistently-passed tests. Overall, there was only one CPRNG which failed entirely in its capacity to generate pseudo-random output. However, even the highest-performing CPRNGs were unable to match the capabilities of the control.

The tests which were passed almost uniformly by the CPRNGs birthday spacings test (in which a set of random points chosen on a large interval is tested for an exponential distribution), and the 32x32 and 6x8 rank tests (in which the rank of multiple matrices built from random numbers is counted) [8]. Notice that these tests are the three which require the fewest random numbers of the entire test suite.

Recall that the periodic orbits eventually reached by the CPRNGs must be both unpredictably distributed and sufficiently long (approaching the size of the state space) in order for such an orbit to be capable of producing pseudo-random data. One explanation for the inconsistencies in the test results is that the idea of a “sufficiently long” periodic orbit is dependent on the amount of random data required, or the granularity at which one is analyzing the randomness of CPRNG resolution.

In other words, the passing of the tests requiring less data suggests that the pseudo-chaotic orbits achieved by almost all of the maps were sufficiently diverse in distribution to be used as good approximations to random data, but were simply too short for use in more consumptive tests.

With that in mind, there were a few notable outliers in the set of PRNGs (namely, the Chebyshev map, the piecewise map, the modified tent map, and the circle map) which were able to perform well on tests outside the set of those passed easily by the CPRNGs. While it is difficult to analytically or computationally pinpoint exactly why these CPRNGs performed better than their counterparts (for reasons fully enumerated in End Note 4), it is reasonable to believe that it has to do either with the length of the periods these maps approached or with the even distribution of these eventually periodic orbits across the state space.

This can be seen especially clearly in the successes of the modified tent and piecewise maps. While these maps failed to pass even the 6x8 rank test (which was passed by all other CPRNGs except that of the singer map), they succeeded fully in the entire set of NIST sts monobit and series tests, designed to test the series of random numbers as randomly constructed strings of bits at various resolutions (referred to in the tests and in Appendix B as ”ntups”). This suggests that, although the periodic orbits may be too confined, they still consisted of quite unpredictable selections throughout the sample space [9] and thus worked exceedingly well on a small scope.

On the other hand, it is much easier to analytically explain the shortcomings of extremely poor performers. The most clear example of this is the standard tent map (see Appendix A). As demonstrated in class, the set of all initial conditions for chaotic orbits of the tent map is the set of numbers without a finite binary expansion – the set of irrationals. Accordingly, the tent map's pseudo-chaotic counterpart (implemented and demonstrated in full in the file /src/broken/tent.c, but replaced with the "modified tent" map in actual experimentation) quickly falls into a short periodic orbit and thus fails each of the dieharder tests.

Similarly, the singer map failed for dynamical reasons; the finite initial seed provided at run time was in the basin of an attracting fixed point for the pseudo-chaotic system analog (0.6821692814), and the potentially-random orbit quickly fell into period-1 repetition.

In terms of performance, some of the CPRNGs (especially the Chebychev map) performed well, even in comparison to a highly-optimized industry standard with performance features such as specialized compilation techniques on most platforms. This suggests that, with any amount of low-level optimization, some of these maps may have the potential to match or even outperform the control PRNG. This would make them extremely valuable tools for use in situations where lightweight PRNGs are needed at a small scope.

Analysis

The observations from the dieharder tests breakdown of the CPRNGs provide us with some insights into the relationship between properties of the finite analogs of chaotic maps and their applicability and potential for use as random number generators.

First, the size and distribution of the set of pseudo-chaotic initial conditions affects the statistical consistency of the CPRNGs results. The smaller this set, the more likely it is that a CPRNG will fail entirely by falling into an unsuitably short orbit or even getting trapped into spitting out the same exact "random" number repeatedly forever. Clearly, both of these behaviors are unacceptable in PRNGs, and thus much care needs to be taken in CPRNG design to ensure that either the map has no such pitfalls or that, as suggested by Senkerik et. all, some mechanism is put in place to automatically catch problematic initial conditions and switch to the use of a different chaotic map within the CPRNG [7].

Second, the periodicity of various orbits in a pseudo-chaotic map has a large effect on the resolution or scale at which a CPRNG is useful. Taking care to design pseudo-chaotic maps in such a way that the minimum orbit is decided to be sufficiently long for use in a particular application. This practice has been the primary method of improvement of standard PRNGs since the advent of Linear Congruential Model and may prove to be highly useful in the design and engineering of CPRNGs as well [8].

Third, the distribution of points across the system's state space for each orbit within a pseudo-chaotic map seems to play a defining role in the extent to which a CPRNG succeeds at random number generation (when seeded with an initial condition for a sufficiently-long, eventually-periodic orbit).

In conclusion, it appears that pseudo-chaos is a fairly unexplored aspect of chaos theory, and yet is one with great potential for application. The results of the tests suggested that efficient, powerful CPRNGs are a very real possibility, and may be useful already as lightweight PRNGs working with limited-scope use.

Notes

[1] A major development focus for this project was to design these CPRNGs in such a way that they would be easily extensible for future development and practical use. To this end, all software was designed and constructed using software engineering practices which may seem unnecessary for the mathematics, but will be largely helpful to anyone trying to build, test, use, or extend the set of CPRNGs implemented in this assignment.

The other justification for this focus is that the implementation of the CPRNGs tested should be as similar to the implementation in which they would be applied. This is the primary justification for the choice of C (instead of MATLAB, R, or even a higher-level language such as Python).

[2]Blackledge and Ptitsyn recommend the use of Kolmogorov-Sinai entropy. It's interesting to note that this is related both to the Lyapunov Exponent of an orbit at various resolutions, and also to the algorithmic complexity of the pseudo-chaotic system. This touches one of the core issues with CPRNG development from analytically chaotic maps; while complexity is not a necessary property in chaotic systems, it is a necessary characteristic of strong PRNGs [3]. However, further analysis along these lines is beyond the scope of this project and is more fully developed in Blackledge and Ptitsyn's paper.

[3]The amount of rigor required by dieharder to accurately gauge the "randomness" of each generator was the source of many practical issues. Each attempted trial of the program had to run for 10 to 12 hours, and upon completion, there was far too much data to import into MATLAB without running out of memory, even for purposes as simple as graphing the tracefiles.

For example, in order to have calculated the Lyapunov exponent, one would needed to have read through the entirety of each 13Gb text file and run a program to estimate the Lyapunov exponent on an orbit no greater than the length of the shortest orbit found. With 20 orbits of 1,000,000,000 points each, this is a nontrivial computational challenge.

This difficulty, in addition to the weaknesses with traditional chaotic measures on pseudo-chaotic orbits (described in background information), is why it was not feasible to calculate the Lyapunov exponents of the pseudo-chaotic orbits. Furthermore, computing more-viable measures such as the orbits' Kolmogorov-Sinai Entropies is a computational challenge outside the scope of this project.

[4]An example of this is the choice of some of the maps suggested for use in Biogeography based optimization of neural computation by Saremi and Mirjalili [2]

References

[1] Brown, R. G. (n.d.). Robert G. Brown's DieHarder Page. Retrieved January 07, 2016, from http://www.phy.duke.edu/ rgb/General/dieharder.php

[2] Saremi, S., & Mirjalili, S. (2013). *Integrating Chaos to Biogeography-Based Optimization Algorithm.* International Journal of Computer and Communication Engineering IJCCE, 655-658.*Zur Elektrodynamik bewegter Körper.* (German)

[3] Blackledge, J., & Ptitsyn, N. (2010). *Encryption using Deterministic Chaos.* ISAST Transactions on Electronics and Signal Processing, vol. 4, issue 1, pp. 6-17.

[4] S.K. Park & K.W. Miller (1988). *Random Number Generators: Good Ones Are Hard To Find.* Communications of the ACM 31 (10): 1192–1201.

[5] Alligood, K. T., Sauer, T. D., & Yorke, J. A. (2000). *Chaos: An introduction to dynamical systems.* New-York: Springer.

[6] Boffetta, G. (2002). *Predictability: A way to characterize complexity.* Physics Reports, 356(6), 367-474

[7] Senkerik, R., Pluhacek, M., Zelinka, I., & Oplatkova, Z. K. (2014). *Utilization of the Discrete Chaotic Systems as the Pseudo Random Number Generators.* Advances in Intelligent Systems and Computing Modern Trends and Techniques in Computer Science, 155-164.

[8] Renyl, A. (1953). *On the Theory of Order Statistics* Acta Mathematica Hungarica Akadémiai Kiadó, vol. 4, issue 3-4, pp.191-231

[9] National Institute of Standards and Technology (2010). *A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications* NIST Special Publication 800-22rev1a http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf

Appendix A: Maps Tested

Chebyshev Map

$$x_{n+1} = \frac{\cos(i * \arccos(x_n))}{2}$$

Circle Map

$$x_{n+1} = (x_n + 0.2 - \frac{0.5 \sin(2\pi x_n)}{2\pi}) \, mod \, 1$$

Gauss Map

$$x_{n+1} = \exp(-8.0x_n^2) \; - 0.6$$

Iterative Map

$$x_{n+1} = \sin \frac{0.7\pi}{x_n}$$

Logistic Map

$$x_{n+1} = 3.57x_n(1 - x_n)$$

Modified Tent Map

$$x_{n+1} = \begin{cases} \frac{x_n}{0.7} & x_n < 0.7 \\ \frac{10}{3}(1.0 - x_n) & x_n \geq 0.7 \end{cases}$$

Piecewise Map

$$x_{n+1} = \begin{cases} \frac{1.0 - x_n}{0.4} & 0.6 \leq x_n < 1 \\ \frac{0.6 - x_n}{0.1} & 0.5 \leq x_n < 0.6 \\ \frac{x_n - 0.4}{0.1} & 0.4 \leq x_n < 0.5 \\ \frac{x_n}{0.4} & 0 < x_n < 0.4 \end{cases}$$

Sine Map

$$x_{n+1} = \sin x_n \pi$$

Singer Map

$$x_{n+1} = 1.07(-13.302875x_n^4 + 28.75x_n^3 - 23.31x_n^2 + 7.68x_n)$$

Sinusoidal Map

$$x_{n+1} = 2.3x_n^2 \sin(\pi x_n)$$

Appendix B: DieHarder Test Results

	Control	Cheby	circle	gauss	iterate	log	mtent	piece	sine	singer	sinus
birthday	0.80037713	0.61612158	0.69002300	0.14540994	0.68974158	0.00000000	0.97484422	0.46939423	0.28385343	0.00000000	0.98521939
operm5	0.87698994	0.70707689	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
rank (32 x 32)	0.71787430	0.98911707	0.38547362	0.56991125	0.07764281	0.27246476	0.51828621	0.30537308	0.47160765	0.00000000	0.42179329
rank (6 x 8)	0.98221815	0.34669293	0.98420807	0.94639648	0.29876321	0.00018419	0.00000000	0.00000000	0.04711309	0.00000000	0.47319463
bitstream	0.86164212	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.29326739	0.30765939	0.00000000	0.00000000	0.00000000
opso	0.01560613	0.01319234	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
oqso	0.02954957	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
dna	0.36274614	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
count 1s str	0.91680337	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
count 1s byte	0.38060358	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
parking lot	0.79600924	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
2d sphere	0.73163604	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
3d sphere	0.69822030	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000

	Control	Cheby	circle	gauss	iterate	log	mtent	piece	sine	singer	sinus
squeeze	0.91864526	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
runs (1)	0.78263170	0.87351971	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
runs (2)	0.44210106	0.49737442	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
craps (1)	0.85406703	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
craps (2)	0.06072170	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
marsaglia tsang gcd (1)	0.49452632	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
marsaglia tsang gcd (2)	0.12324267	0.46955378	0.03685836	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
sts monobit	0.65511215	0.54276764	0.00000000	0.00000000	0.18736829	0.00000000	0.73577198	0.82015233	0.00000000	0.00000000	0.00000000
sts runs	0.11869829	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.76663333	0.28902569	0.00000000	0.00000000	0.00000000
sts serial ntups = 1	0.65511215	0.54276764	0.00000000	0.00000000	0.18736829	0.00000000	0.73577198	0.82015233	0.00000000	0.00000000	0.00000000
sts serial ntups = 2	0.80235845	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.44412050	0.31662081	0.00000000	0.00000000	0.00000000
sts serial (1) ntups = 3	0.84397721	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.79864491	0.95372691	0.00000000	0.00000000	0.00000000
sts serial (2) ntups = 3	0.94185356	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.79810964	0.48195750	0.00000000	0.00000000	0.00000000
sts serial (1) ntups = 4	0.96709032	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.02191229	0.72869188	0.00000000	0.00000000	0.00000000

	Control	Cheby	circle	gauss	iterate	log	mtent	piece	sine	singer	sinus
sts serial (2) ntups = 4	0.20776304	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.01670098	0.20358673	0.00000000	0.00000000	0.00000000
sts serial (1) ntups = 5	0.64218821	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.01893154	0.15558838	0.00000000	0.00000000	0.00000000
sts serial (2) ntups = 5	0.73363534	0.00000000	0.56925823	0.00000000	0.00000000	0.00000000	0.98559162	0.23513045	0.00000000	0.00000000	0.00000000
sts serial (1) ntups = 6	0.84014037	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.13572548	0.15258654	0.00000000	0.00000000	0.00000000
sts serial (2) ntups = 6	0.96719431	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.86706303	0.30806653	0.00000000	0.00000000	0.00000000
sts serial (1) ntups = 7	0.49695827	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00165143	0.67421597	0.00000000	0.00000000	0.00000000
sts serial (2) ntups = 7	0.22415817	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.09253935	0.32256166	0.00000000	0.00000000	0.00000000
sts serial (1) ntups = 8	0.87314884	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.14377708	0.95403385	0.00000000	0.00000000	0.00000000
sts serial (2) ntups = 8	0.59261587	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.99516053	0.96963698	0.00000000	0.00000000	0.00000000
sts serial (1) ntups = 9	0.83471442	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.26507755	0.90679242	0.00000000	0.00000000	0.00000000
sts serial (2) ntups = 9	0.99686110	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.47290753	0.47024589	0.00000000	0.00000000	0.00000000
sts serial (1) ntups = 10	0.81931026	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.24494277	0.18140977	0.00000000	0.00000000	0.00000000
sts serial (2) ntups = 10	0.32264707	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.91558222	0.59951817	0.00000000	0.00000000	0.00000000
sts serial (1) ntups = 11	0.66225979	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.27898207	0.61880597	0.00000000	0.00000000	0.00000000

	Control	Cheby	circle	gauss	iterate	log	mtent	piece	sine	singer	sinus
sts serial (2) ntups = 11	0.85707797	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.18652946	0.90827716	0.00000000	0.00000000	0.00000000
sts serial (1) ntups = 12	0.30820180	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.56756119	0.13456096	0.00000000	0.00000000	0.00000000
sts serial (2) ntups = 12	0.37041744	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.56228444	0.60003194	0.00000000	0.00000000	0.00000000
sts serial (1) ntups = 13	0.27864397	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.74029006	0.65589206	0.00000000	0.00000000	0.00000000
sts serial (2) ntups = 13	0.46646329	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.65634770	0.80806436	0.00000000	0.00000000	0.00000000
sts serial (1) ntups = 14	0.80459771	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.16740276	0.99580894	0.00000000	0.00000000	0.00000000
sts serial (2) ntups = 14	0.95926557	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.62691250	0.29550675	0.00000000	0.00000000	0.00000000
sts serial (1) ntups = 15	0.68692471	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.70582595	0.44034454	0.00000000	0.00000000	0.00000000
sts serial (2) ntups = 15	0.43717751	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.60430976	0.34172733	0.00000000	0.00000000	0.00000000
sts serial (1) ntups = 16	0.57670831	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.42694708	0.97524520	0.00000000	0.00000000	0.00000000
sts serial (2) ntups = 16	0.95730758	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.98301047	0.58302278	0.00000000	0.00000000	0.00000000
rgb minimum distance	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
rgb permutations	0.70423102	0.95946617	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
rgb kstest test	0.43989972	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000001	0.00000000	0.00000000	0.00000000

	Control	Cheby	circle	gauss	iterate	log	mtent	piece	sine	singer	sinus
dab byte distrib	0.98114803	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.39423380	0.00000000	0.00000000	0.00000000
dab dct	0.55616945	0.51936377	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.01757265	0.00000000	0.00000000	0.00000000
dab fill tree (1)	0.29884218	0.99018824	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
dab fill tree (2)	0.46261950	0.91111412	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
dab fill tree 2 (1)	0.69606118	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
dab fill tree 2 (2)	0.43562959	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
dab monobit 2	0.59964022	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	1.00000000	1.00000000	0.00000000	1.00000000

Color Code:

green	Full Pass
yellow	Weak Pass
red	Failed

Appendix C: Performance Results

Time	Control	Cheby	circle	gauss	iterate	log	mtent	piece	sine	singer	sinus
Real	11m12.913s	13m36.957s	12m50.958s	13m56.388s	14m50.037s	15m58.258s	17m53.710s	17m52.890s	16m21.521s	18m55.896s	20m21.797s
User	9m9.341s	10m53.023s	9m58.970s	9m50.527s	9m49.312s	8m53.796s	9m17.165s	9m22.241s	9m44.204s	10m55.871s	9m57.889s
Sys	0m39.246s	0m49.477s	0m50.541s	0m50.890s	0m43.047s	0m45.368s	0m49.794s	0m49.067s	0m50.498s	0m49.914s	1m14.486s