

Random number generators: pretty good ones are easy to find

Clifford A. Pickover

IBM Watson Research Center, Yorktown Heights,
NY 10598, USA

A popular conception is that random number generators are very difficult to build. I informally discuss some easily programmed, easily remembered, random number generators. Simple graphical techniques are introduced for assessing the quality of the generators with little training. To encourage reader involvement, computational recipes are included.

e-mail: cliff@watson.ibm.com

1 Introduction

"In a sense, randomness, like beauty, is in the eye of the beholder". – S. Park and K. Miller, 1988.

The prevailing view of researchers working with computers is that it is very difficult to create a good random number generator (RNG) casually. In fact, if you were to tell colleagues that the first function that came to your head would make a fine RNG for most applications, surely they would shake their heads with disbelief at such heresy. Many papers and books passionately warn of the complexities in creating random numbers (Gordon 1978; Knuth 1981; Mckean 1987; Park and Miller 1988; Voelcker 1988). For an excellent review, see Park and Miller (1988). I have found that wise caution has sometimes led to an almost mystical, exaggerated sense of difficulty in this area.

As background, many years of experience have demonstrated the usefulness of RNGs in computer programming, numerical analysis, sampling, simulation, recreation, and decision making (Knuth 1981). There has also been considerable interest in the area of computer graphics where random numbers are used to construct complex and natural textures, terrains, and various special effects (Pickover 1990).

I begin our discussion by defining what I mean by a good RNG. First, I require that the RNG produces a few million numbers that are statistically independent and uniformly distributed between 0 and 1. Second, I have the interesting requirement that the generating formula and parameters be easy to remember. By far the most popular RNGs today make use of the linear congruential method and generally have the form:

$$X_{n+1} = (aX_n + c) \bmod m, \quad n \geq 0.$$

Here, a is the multiplier; m , the modulus; c , the increment; and X_0 , the starting value. Usually, specific, multidigit numbers are required for these variables, and I have not found (even among the most seasoned computer scientists) a single person who can actually write down the parameters for one of these RNGs without consulting textbooks. Many pages have been written on the theoretical reasons for using this number or that for the modulus, multiplier, etc. Fascinating folklores have even evolved regarding some initial seeds that are better than others. I recall, along

with many colleagues, being told to use a seed with at least five digits, with the last digit being odd. This folklore is confirmed by Park and Miller (1988) who note, "For some generators this type of witchcraft is necessary".

Since many computer scientists will never have more than an occasional need to use a RNG, and on most occasions superlong periods and statistical perfection is not of paramount importance, I would like to begin with an anecdote regarding an easily remembered RNG with interesting properties. There are many areas in which "good" RNGs (by my definition) are quite useful, and I am also interested in their use in computer graphics.

2 The Cliff RNG

"Infinity is a very big place." – M. Somos

On 1 February 1994, I posted the following RNG to various computer bulletin boards. I called it the Cliff RNG, and wanted to impress researchers with how easy it is to create a good RNG. (It was the first function that popped into my head.) Consider the iteration

$$X_{n+1} = |100 \log(X_n) \bmod 1|, \quad n = 0, 1, 2, 3 \dots$$

where $x_0 = 0.1$. By "mod 1", I mean only the digits to the right of the decimal point are used. The log function is the natural log. The Cliff algorithm is easily implemented in C with double precision numbers:

```
long num, n;
double x, xi;
x = 0.1 /* seed */
num = 1e6;
for (n = 0; n < num; n++) {
    x = 100 * log(x); /* natural log */
    xi = trunc(x);
    x = fabs(x - xi); /* absolute value */
    printf("Random number is: %.16f\n", x);
}
```

(If desired, the user may add a test to check that the value for x never becomes 0. I did not find this necessary in my applications.)

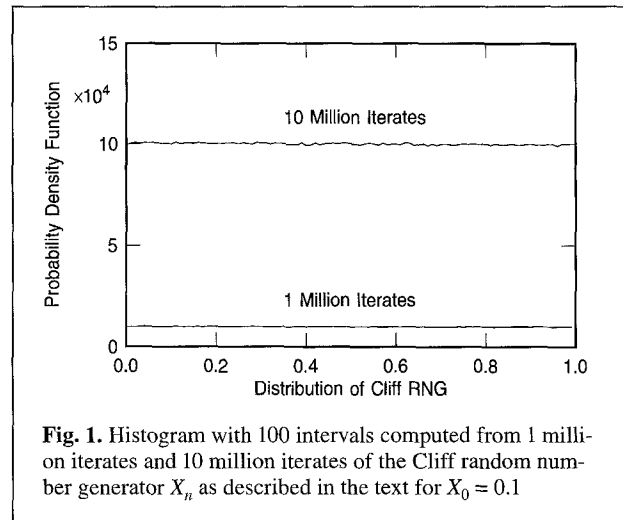


Fig. 1. Histogram with 100 intervals computed from 1 million iterates and 10 million iterates of the Cliff random number generator X_n as described in the text for $X_0 = 0.1$

Figure 1 suggests that X_n are sufficiently uniformly distributed on the interval $(0, 1)$ to be useful for a wide variety of applications. The Cliff RNG also has a theoretically infinite period; however, in practice, the period is finite due to limited machine precision. I find that we can easily generate a million uniformly distributed numbers on an IBM RISC System/6000 in about 4s, with no cycling of numbers. Obviously my goal here is not to create the most efficient RNG or one with a superlong period – but simply one that is easy to remember and adequate for many purposes.

To assess possible correlations between neighbor values in X_n , I used a highly sensitive graphical approach called the *noise sphere* (Pickover 1991a, 1991b). (In harmony with the general objectives of this article, my goal here is also to present simple, easy-to-remember, and easy-to-program approaches.) The noise sphere is computed by mapping random number triplets to the position of little balls in a spherical coordinate system, as described in the next section. High-quality RNGs do not show any particular features in the noise sphere; low-quality ones yield tendrils and various forms of nonhomogeneity. Not only do all good RNGs I have tested pass the noise-sphere test, but all famous poor generators with correlations fail it. Figure 2 (discussed in more detail in the next section) indicates that the Cliff RNG produces essentially uncorrelated numbers, because no particular features are evident in the collection of little spheres. As a counterexample, examine the

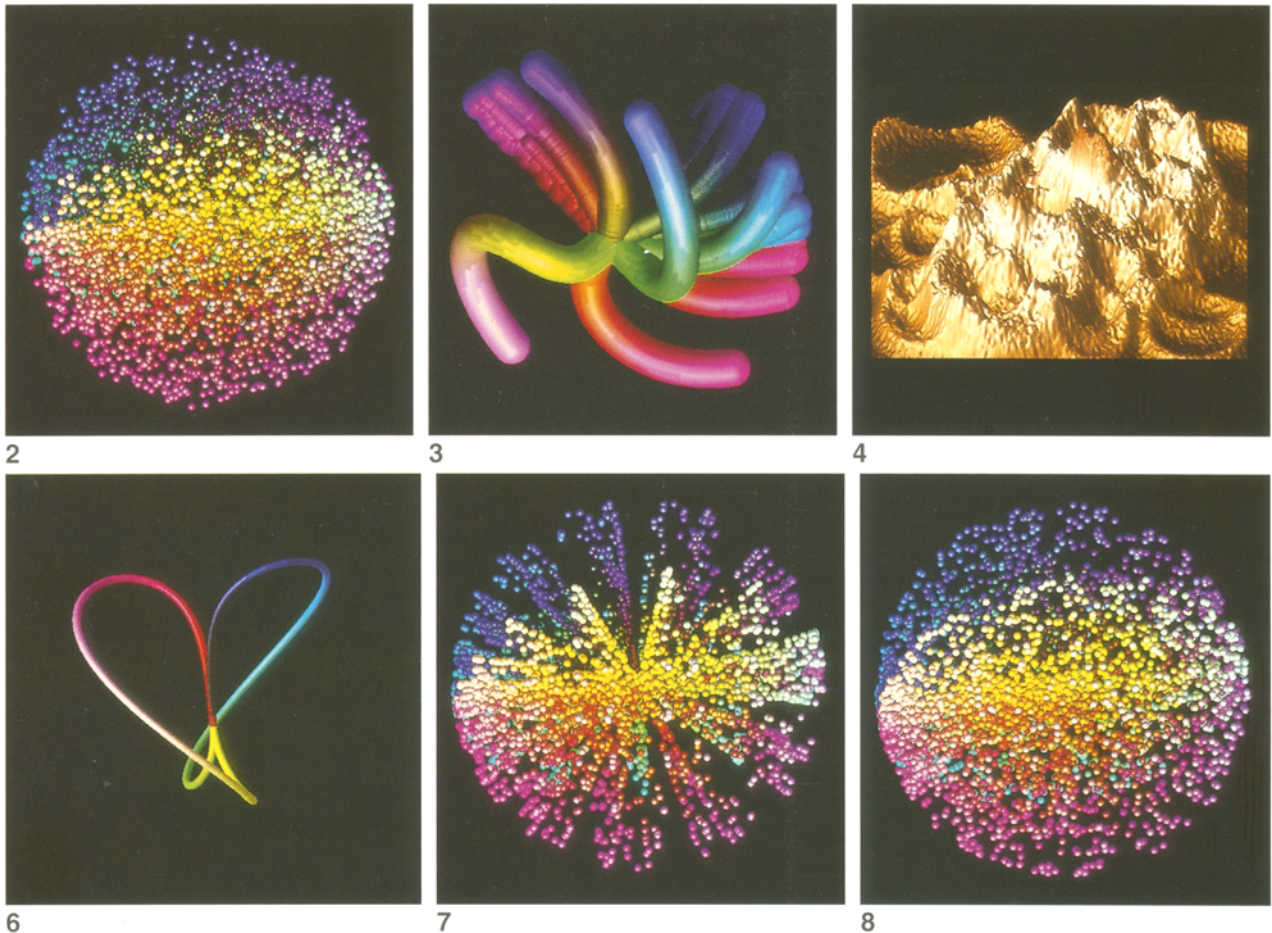


Fig. 2. Noise sphere for the Cliff random number generator. No particular unwanted correlations are evident

Fig. 3. Noise sphere for a poor random number generator. (To give a better feel for the global structure produced, spheres are larger than the ones used for the other noise-sphere figures and interpenetrate to a greater degree)

Fig. 4. Terrain generated by the radialized logit transform on the logistic variable

Fig. 6. Noise sphere for logistic variables, lag = 1. Here no points are skipped. By skipping points (Figs. 7 and 8), correlations are reduced

Fig. 7. Noise sphere for logistic variables, lag = 5. (Every fifth point is used. The others are discarded)

Fig. 8. Noise sphere for logistic variables, lag = 10

noise sphere in Fig. 3, which omits the 100 from the expression $100 \times \log(x)$. The nonindependence of the numbers is clearly seen.

In this next section, I describe the noise sphere and suggest that it is a useful and simple way to check the quality of RNGs.

3 Noise spheres

A sensitive graphical representation called a noise sphere can be used to detect many poor RNGs

with little training on the part of the observer. (Poor RNGs often show various tendrils when displayed in the noise sphere representations.) As suggested earlier in this paper, in modern science, RNGs have proven invaluable in simulating natural phenomena and in sampling data (Gordon 1978; Knuth 1981; McKean 1987; Parker and Miller 1988; Pickover 1990; Voelcker 1988). It is therefore useful to build easy-to-use graphic tools that, at a glance, help determine whether a RNG being used is poor (i.e., nonuniform and/or with dependence between various digits). Although

a graphics supercomputer facilitates the particular representation described in the following, a simpler version would be easy to implement on a personal computer. I wish to stimulate and encourage programmers, students, and teachers to explore this technique in a classroom setting. To produce the noise leave sphere figures, simply map the output of a RNG to the position of spheres in a spherical coordinate system. This is easily achieved with a computer program using random numbers $\{X_n, n = 0, 1, 2, 3, \dots, N$ ($0 < X_n < 1$)}, where X_n , X_{n+1} and X_{n+2} are converted to θ , ϕ and r respectively. For each cartesian triplet, an (r, θ, ϕ) is generated, and these coordinates position the individual spheres. The data points may be scaled to obtain the full range in spherical coordinates:

$$2\pi X_n \rightarrow \theta$$

$$\pi X_{n+1} \rightarrow \phi$$

$$\sqrt{X_{n+2}} \rightarrow r$$

The square root for X_{n+2} serves to spread the ball density though the spherical space and to suppress the tight packing for small r . Each successive triplet overlaps, so that each point has the opportunity to become an r , θ , or ϕ . (The advantage over a cartesian system is discussed later in this paper.) The resultant pictures can be used to represent various kinds of noise distributions or experimental data. As already mentioned, no particular correlations in the ball positions are visually evident with good RNGs. For surprising results when this approach is applied to a BASIC RNG, see Pickover (1991a, 1991b). In particular, the method has been effective in showing that RNGs prior to release 3.0 of BASIC had subtle problems (Pickover 1991a, 1991b).

Although such representations are fairly effective in black and white, I have found it useful to map the random number triplet values to a mixture of red, green, and blue intensities to help the eye see correlated structures. (In other words, for a particular sphere, a value of red, green, and blue from 0 to 1 is determined by the value of X_n , X_{n+1} , and X_{n+2} .) I use an IBM Power Visualization System to render and rotate the noise spheres; however, interested users without access to graphics super-

computers can render their data simply by plotting a scattering of dots and projecting them on a plane. Most importantly, I have found that the spherical coordinate transformation used to create the noise sphere allows the user to see trends much more easily, and with many fewer trial rotations, than an analogous representation that maps the random number triplets to three orthogonal coordinates. Since the noise sphere approach is sensitive to small deviations from randomness, it may have value in helping researchers find patterns in complicated data such as genetic sequences and acoustical waveforms. Some readers may wonder why I should consider the noise sphere approach over traditional brute-force statistical calculations. One reason is that this graphic description requires little training on the part of users, and the software is simple enough to allow users to implement the basic idea quickly on a personal computer. The graphics description provides a qualitative awareness of complex data trends, and may subsequently guide the researcher in applying more traditional statistical calculations. Also, fairly detailed comparisons between sets of "random" data are useful and can be achieved by a variety of brute-force computations, but sometimes at a cost of the loss of an intuitive feeling for the structures. When one is just looking at a page of numbers, differences between the data's statistics may obscure the similarities. The approach described here provides a way of simply summarizing comparisons between random data sets and capitalizes on the feature integration abilities of the human visual system to see trends.

4 Logistic and logit transformations

In this section, I extend our discussion of simple RNGS to new functions, and introduce readers to a fascinating, nonperiodic near-gaussian RNG. I find that it is useful for a variety of purposes, including the graphical generation of extraterrestrial terrain (Fig. 4), and I would be interested in hearing from readers who use these methods for other applications. The approach described may be used in laboratory and classroom demonstrations, and has the potential for producing a wide range of fanciful terrain structures for special effects in movies.

5 The logistic equation

To produce Fig. 4, I have used an approach based on the famous logistic equation, which is easily implemented by beginning programmers and computer graphics aficionados. The logistic equation or map is given by the expression:

$$x_{n+1} = 4\lambda x_n(1 - x_n)$$

where $0 < \lambda < 1$. This equation was introduced in 1845 by the Belgian sociologist and mathematician P. F. Verhulst to model the growth of populations limited by finite resources. The equation has since been used to model such diverse phenomena as fluid turbulence and the fluctuation of economic prices. [For a nice review, see Schroeder (1991).] As the parameter λ is increased, the iterates x_n exhibit a cascade of period-doubling bifurcations followed by chaos.

If we use $\lambda = 1$ then x_n are “random” numbers on the interval $(0, 1)$ with a U-shaped probability distribution P_x given by Collins et al. (1992), Ulam and von Neumann (1947) and Stein and Ulam (1964):

$$P_x = \frac{1}{\pi\sqrt{x(1-x)}}$$

Figure 5 shows a histogram with the expected clustering near $x = 0$ and $x = 1$ for $x_0 = 0.1$. Ulam and von Neumann (1964) defined a new variable y_n as:

$$y_n = (2/\pi) \sin^{-1}(\sqrt{x_n}).$$

This “ulamized” variable y_n has potential as a RNG because the numbers produced are uniformly distributed on the interval $(0, 1)$, (Collins et al. 1992; Ulam and von Neumann 1947; Stein and Ulam 1964) (Fig. 5).

Note that while the ulamized variables are distributed uniformly on the interval $(0, 1)$, unfortunately the variables do have correlations. This is also true for the logistic variables, as evidenced in Figs. 6 and 7. Therefore, users of this approach must sample the generated variables at some suitably large interval to eliminate correlations. I recommend that users skip iterates, taking every

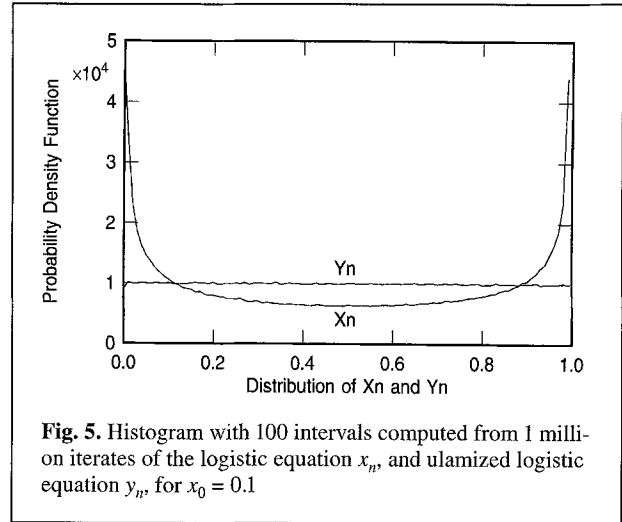


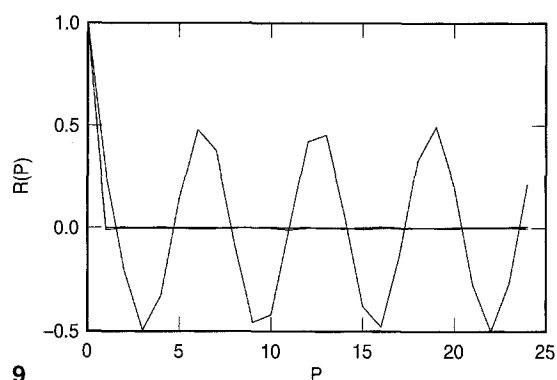
Fig. 5. Histogram with 100 intervals computed from 1 million iterates of the logistic equation x_n , and ulamized logistic equation y_n , for $x_0 = 0.1$

tenth iterate, at which point no correlations are visually apparent (Fig. 8).

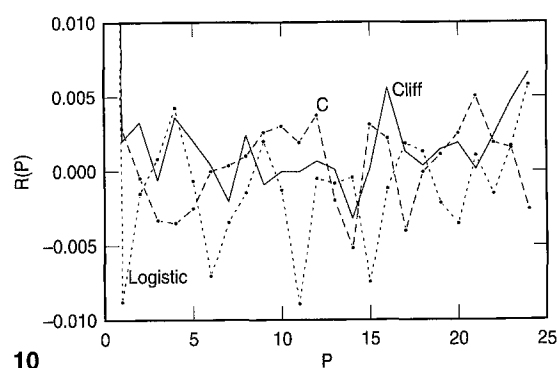
Interestingly, noise spheres are more useful than traditional correlation functions when applied to many kinds of random data. For example, Grossman and Thomae (1977) have computed correlation functions for the logistic formula, and the functions appear to indicate vanishingly small (close to 0) correlations for lags greater than 1. I have confirmed this (Figs. 9 and 10) by computing the same kind of autocorrelation function for the logistic variables (and for other RNGs):

$$R_x(p) = \frac{1}{n-p} \sum_{q=1}^{n-p} X_q X_{q+p} \quad p = 0, 1, \dots, m$$

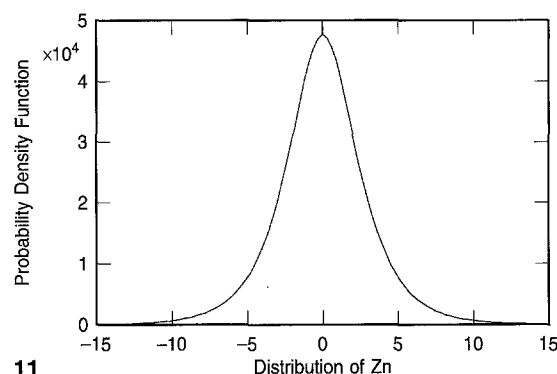
where n is the number data points. The autocorrelation function for random data describes the general dependences of the values of the data at one point in the sequence on the values at another place. It is a measure of the predictability of a fluctuating function $x(t)$. If the value of the function $x(t + \tau)$ can be predicted with a good expectation of success when the value of $x(t)$ is known, then the product $x(t)x(t + \tau)$ does not become zero when averaged over t . Grossman and Thomae (1977) examine such autocorrelation functions, note the immediate drop to zero, and say that for $\lambda = 1$ “all the iterates are ergodic, that is [the logistic formula] generates pure chaos.” I have found that many graphics specialists have taken



9



10



11

Fig. 9. Autocorrelation function for a sine wave (the rippling function), and for the logistic, Cliff, and C-language random number generator (all which immediately drop to values close to zero)

Fig. 10. Magnification of 9

Fig. 11. Histogram with 100 intervals computed from 1 million iterates of the logit transform of the logistic variable for $x_0 = 0.1$

this to suggest that the logistic formula produces uncorrelated values. (Note that the noise sphere allows humans to perceive correlations even beyond a lag of five.) In other words, individuals sometimes confuse the concept of ergodic orbits of dynamic systems with the concept of an independent, identically distributed sequence of numbers that, ideally, is produced by RNG. Sequences of numbers produced by function iteration ($X_{n+1} = f(x_n)$) can almost never be independent, identically distributed sequences, and are often highly correlated as evidenced in this paper. One standard way of revealing this correlation is simply to plot X_n versus X_{n+1} (Richards 1989). The noise spheres do a similar job, but show correlations between three successive values.

Figure 9 shows the autocorrelation function for a sine wave (the rippling function), and for the logistic, Cliff, and C-language RNGs (all of which immediately drop to values close to zero.) As can be seen, except for the sine wave, the others all have a vanishingly small correlation value for lags greater than 1. Even when the autocorrelation functions for the various RNGs are greatly magnified (Fig. 10), there is not much useful information that can be gleaned from the noisy plots. The noise spheres are a much better way to detect correlations within noisy data sets. For a sampling of brute-force statistical tests that are sensitive to correlations, see Knuth (1981).

6 The logit transform

For terrain generation, student experiments, and possible inclusions in statistical software packages, I have found that the transform given by

$$z = \ln \left(\frac{x}{1-x} \right) \quad (1)$$

to be quite useful. (In statistics, this is known as the *logit*.) The logit transform may operate *directly* on the logistic variable, and Collins et al. (1992) have shown that this transform generates a sequence of random numbers with

a near-gaussian distribution:

$$P_z = \frac{1}{\pi} \left(\frac{e^{z/2}}{1 + e^z} \right) = \frac{1}{\pi(e^{z/2} + e^{-z/2})}$$

(Students are urged to compare this to a gaussian distribution and observe the remarkable similarity.) Figure 11 shows an experimentally derived histogram for the logit transform of the logistic variable.

If readers are interested in using the previous idea for 3D extraterrestrial terrain generation, convert the z_n distribution to a radial one in polar coordinates, and use the ulamized variables y_n as angles:

$$v = z_n \alpha \sin(2\pi y_n) + cx$$

$$w = z_n \alpha \cos(2\pi y_n) + cy \quad (2)$$

For each (v, w) pair generated, a 2D $P(v, w)$ array is incremented by 1 at location (v, w) . (Care should be taken to iterate the logistic formula a few times before using it to compute both y_n and z_n to eliminate unwanted correlations.) The resultant $P(v, w)$ distribution, after many iterations, resembles a circular ridge centered at (cx, cy) with a near-gaussian cross-section. The variable α controls the diameter of the crater. The landscape in this paper was created by computing a 512×512 $P(v, w)$ array for 50 ulamized values of α , cx , and cy . In other words, I select a random (α, cx, cy) triple using ulamized logistic numbers, then Eq. 2 is iterated N times ($N \leq 10^6$), and a $P(v, w)$ histogram is created. A new triple is chosen, and the process is repeated – each time “laying down” a radial near-gaussian function as numbers accumulate in the $P(v, w)$ histogram. For more examples of terrain produced by this method, more computational details, and further analysis, see Pickover (1995a).

I enjoy the logit method because it produces a wealth of lunarlike structures not produced by standard fractal methods, and also the method is conceptually simpler than the Fourier transform and midpoint displacement methods. It is easily implemented and understood by students and programmers. There are no Fourier transforms, no erosion models, no convolutions, no complicated vertex bookkeeping. From a pedagogical

standpoint, it therefore makes a wonderful demonstration of not only how to build complex terrains, but of how the relationships between RNGs, probability theory, and chaotic dynamics are intimately intertwined. From a practical standpoint, the logit transform operating on the logistic variables produces a distribution with a slightly more gradual decay to zero than does a gaussian distribution. This may be useful in certain applications; for example, for the terrain it means that the crater meets the ground in a more gradual way, which I find aesthetically pleasing.

7 Discussion

Obviously, in some areas in physics and numerical analysis, “pretty good” RNG are not suitable, and various past papers point readers to more sophisticated algorithms. Nonetheless, I feel there is a need for simple, easy-to-remember RNGs such as the ones I have presented for various applied and pedagogical applications. Certainly the myth suggesting that adequate, simple RNGs are very difficult to create is an intriguing one to explore. Interestingly, some researchers have suggested that RNGs should be able to give the same random numbers when run on different computers and when implemented in different languages. Unfortunately, as pointed out by Colonna (1994) rounding errors often make this quite difficult, especially for generators based on function iterations of the form $X_{n+1} = f(x_n)$.

Despite the strong correlations between successive numbers – when numbers are not skipped – there are several interesting advantages of the logit approach to producing near-gaussian “random” numbers. First, these near-gaussian numbers are easily incorporated into software for stochastic modeling. Second, like the Cliff RNG, the logistic-based number generators theoretically have an infinite period – although in practice, the period is finite due to the limited precision of computers. This limitation can easily be overcome by restarting the generator with a new x_0 every billion or so iterations. For the terrain in this paper, x_0 and λ were fixed.

Collins et al. (1992) have explored the use of the logit transform for a variety of statistical

purposes, and they point out an interesting advantage that the n th number generated by the logistic equation can be determined directly without iteration:

$$x_n = \sin^2 [2^n \sin^{-1}(\sqrt{x_0})]$$

where x_0 is some initial value. The values of x_n generated in this manner can then be transformed by the logit given in Eq. 1. Again, by skipping output numbers we can reduce correlations by an arbitrary amount.

Interest in RNGs continues to grow, as evidenced by a recent explosion of papers in exciting new RNGs (Clark 1985; Ferrenberge et al. 1992; James 1990, 1994; Luscher 1994; Marsaglia and Zaman 1991) that are not based on the extensively studied, multiplicative, congruential types. For example, much longer “defect-free” sequences (James 1994) have been proposed that are based on chaotic, dynamic systems (See, for example, the luxury random numbers (RANLUX) algorithm described in James (1994) which has a period of 10^{170} .)

A report such as this can only be viewed as introductory; however, it is hoped that the discussion will stimulate future interest in the graphic representation of simple RNGs. In laboratory exercises for courses such as statistical mechanics, thermodynamics, and even computer programming, it is unlikely that undergraduate students will be required to write their own RNGs, but I feel examples such as the ones given in this paper will be of great instructional value to many students and scientists involved with computers. For information on other graphical methods for detecting patterns in random numbers, such as the use of computer-generated cartoon faces and Truchet tiles, see Pickover (1990). For information on visual methods for detecting patterns in seemingly random information-containing sequences in biology, see Jefferey (1994) and Pickover (1990, 1991b, 1992). For a fanciful treatment on RNGs in an alien society, see Pickover (1994). For the use of RNGs in science, simulation, and art see Pickover (1995b) and Pickover and Tewksbury (1994). For very recent papers in novel random number generators see Hennecke (1994) and Marsaglia and Zaman (1994).

Acknowledgements. I thank Dr. James Collins (Boston University) for bringing the logit transform of the logistic variable to my attention and for advice and encouragement.

References

1. Clark RN (1985) A pseudorandom number generator. *Simulation* 45:252–255
2. Collins J, Fancilulli M, Hohlfeld R, Finch D, Sandri G, Shtatland E (1992) A random number generator based on the logit transform of the logistic variable. *Comput Phys* 6:630–632
3. Colonna J (1994) The subjectivity of computers. *Commun ACM*. 36:15–18
4. Ferrenberge A, Landau D, Wong Y (1992) Monte Carlo simulations: hidden errors from “good” random number generators. *Phys Rev Lett* 69:3382–3384
5. Gordon G (1978) *System simulation*. Prentice-Hall, Englewood Cliffs, N.J
6. Grossman S, Thomae S (1977) Invariant distributions and stationary correlation functions. *Z Naturforsch* 32a:1353
7. Hennecke M (1994) RANEXP: experimental random number generator package. *Comput Phys Commun* 79:261
8. James F (1990) A review of pseudorandom number generators. *Comput Phys Commun* 60:329–344
9. James (1994) RANLUX: A Fortran implementation of the high-quality pseudorandom number generator of Luscher. *Comput Phys Commun* 79:111–114
10. Jefferey H (1994) Fractals and genetics in the future. In: Pickover C (ed) *Visions of the future: art, technology, and computing in the 21st century*. St. Martin's Press, New York
11. Knuth D (1981) *The art of computer programming*, Vol. 2, 2nd edn. Addison-Wesley, Reading, Mass
12. Luscher M (1994) A portable high-quality random number generator for lattice field theory simulations. *Comput Phys Commun* 79:100–110
13. Marsaglia G, Zaman A (1991) A new class of random number generators. *Ann Appl Probability* 1:462–480
14. Marsaglia G, Zaman A (1994) Some portable very-long-period random number generators. *Comput Phys* 8:117–121
15. McKean K (1987) The orderly pursuit of disorder. *Discover* 5:72–81
16. Park S, Miller K (1988) Random numbers generators: good ones are hard to find. *Commun ACM* 31:1192–1201
17. Pickover C (1990) *Computers, pattern, chaos and beauty*. St. Martin's Press, N.Y
18. Pickover C (1991a) Picturing randomness on a graphics supercomputer. *IBM J Res Devel* 35:227–230
19. Pickover C (1991b) *Computers and the imagination*. St. Martin's Press, N.Y
20. Pickover C (1992) *Mazes for the mind: computers and the unexpected*. St. Martin's Press, New York
21. Pickover C (1994) *Chaos in wonderland: visual adventures in a fractal world*. St. Martin's Press, New York
22. Pickover C (1995a) Terrain generation via the logit transformation of the logistic variable, *IEEE Comput Graph* 15:18–21
23. Pickover C (1995b) *Keys to Infinity*. Wiley, New York
24. Pickover C, Tewksbury S (1994) *Frontiers of scientific visualization*. Wiley, New York
25. Richards T (1989) Graphical representation of pseudorandom sequences. *Comput Graph* 13:261–262

26. Schroeder M (1991) Fractals, chaos, power laws. Freeman, N.Y
27. Stein P, Ulam S (1964) Rozpr 39:1
28. Ulam S, von Neumann J (1947) Bull Am Math Soc 53:1120
29. Voelcker J (1988) Picturing randomness. IEEE Spectrum. 25:13

The photograph and biographie of Clifford A. Pickover was published in Volume 11, Issue 6, p 312, 1995