Pseudo Random Numbers

by Frank Bjørnø

The purpose of this article is to discuss the random number generator that should be distributed along with it.

The linear congruential generator

The heart of the RNG is the linear congruential generator

$$X_{n+1} = (aX_n + c) \mod m$$

where $a = 4706\,8445_{\text{HEX}}$, $c = 0010\,16\text{B}5_{\text{HEX}}$ and $m = 7\text{FFF}\,\text{FFFF}_{\text{HEX}}$. The number a is chosen so that $a \mod 8 = 5$ and $0.01m < a < 0.99m$. A good candidate for c should have no factors in common with m and should be odd. In this case c is the product of the arbitrarily chosen primes $\{3, 7, 23, 37, 59\}$. The modulus, m should be as large as possible, relatively prime and should, if possible, be strategically chosen to speed up computation. The comparatively slow division operation can be avoided by observing that $A \texttt{ mod } 2^n = A \texttt{ and } (2^n - 1)$.

This is based on the method examined in Knuth, Donald E. (1997), 'The Art of Computer Programming 3rd ed., Vol. 2 - Seminumerical Algorithms', p. 10.

A severe limitation of the LCG is that the odd/eveness of successive numbers follows a certain pattern. Suppose the seed, X, is odd. Multiplying it with a (odd) produces an odd number and adding c (also odd) gives an even number. The modulo operation by m will not affect the least significant bit, and therefore will not change the odd/evenness of the number. The next iteration will however produce an odd number so that the LCG produces a series of alternating odd and even numbers.

Setting a or c to even numbers will make the LCD produce only even or only odd numbers.

To remedy this, some of the bits are shifted around after generation of the new seed. This breaks the pattern of odds and evens while preserving the uniqueness of the number.

Numbers are drawn from a uniform distribution

The program 'period.c' demonstrates that the period of the RNG is $2\,147\,483\,648$ (2^{31}) or $8000\,0000_{\text{HEX}}$.

Because the RNG can't generate the same seed twice in a period, and because it can't produce a seed larger than $2^{31} - 1$, every number between 0 and $2^{31} - 1$ will be generated. Keeping in mind that the RNG is deterministic, this means that the RNG will produce exactly the same sequence of numbers every time, only starting from a different position in the sequence.

For this reason it is a good idea to reseed the RNG frequently to make it jump around in the sequence.

Suppose, for example, that an RNG has period 10 and, and with initial seed 5, generates the sequence $\{9, 8, 4, 0, 1, 6, 2, 3, 7, 5\}$. If the initial seed was instead 0, the generator would produce the sequence $\{1, 6, 2, 3, 7, 5, 9, 8, 4, 0\}$, which is the same sequence, only starting from a different position

If the RNG produces every number in the range of the function exactly once in one period, this means that numbers are drawn from a uniform distribution, and therefore that there are simple formulas for calculating population mean and variance:

$$\text{Mean:} \quad \mu = \frac{a+b}{2} \qquad \text{Variance:} \quad \sigma^2 = \frac{(b-a)^2}{12}$$

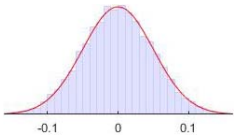
a and b are the minimum and maximum values, respectively.

Statistical tests

A pseudo random number generator is an algorithm which produces a sequence of numbers whose properties approximate the properties of sequences of random numbers. To determine whether a sequence of numbers indeed exhibits the characteristics of a random sequence, we can perform some statistical tests that will provide us with some quantitative

measures for randomness. Although there is practically no end to the number of tests that can be conceived, we will limit ourselves to only handful of them. Of course, if a sequence performs well on these tests, there is no guarantee that it won't fail miserably on others.

1. Test for Autocorrelation



Distribution of autocorrelations produced by the program 'acdist.c'. The figure is generated by MATLAB® using the program 'acdist.m'.

As we are dealing with a series of numbers, there is a potential for correlation between successive numbers. This is called serial- or autocorrelation and would, in most cases, render the generator useless. Most of the time, when working with series, we don't know the population mean and variance because we usually only have access to one of infinitely many possible realisations of the series. In our case, however, we do have access to all possible realizations because they are, as mentioned, all the same. We can therefore calculate the mean and variance using the formulas for a uniform distribution.

The autocorrelation coefficient is, by the central limit theorem, normally distributed around ρ, with standard deviation (standard error of the autocorrelation) $1/\sqrt{N}$, where N is the sample size. Therefore, we can test for autocorrelation performing a hypothesis test under the following assumptions:

$$\mathrm{H}_0 : \rho_k = 0.0$$
$$\mathrm{H}_\mathrm{A} : \rho_k \neq 0.0$$

ρ_k is the autocorrelation at lag k.

The program named 'autocorr.c' takes a sequence of 400 numbers, computes the correlations between a number and 20 sequential observations, and performs a hypothesis test on each coefficient. Below (fig. 1) is the output from the program. Note that even though the null hypothesis is rejected for the autocorrelation at lag 6, it does not mean that there is systematic correlation between x_n and x_{n+6}. Setting alpha at the 5 percent level translates to a 5 percent probability of rejecting the null hypothesis when it is actually true (a Type I error). In other words; 1 in 20 rejections is exactly what we would expect and the correlation, although relatively large, could just be caused by random variation. It is only a problem if a hypothesis is rejected more than 5 percent of the time in repeated runs of the test.

Test for autocorrelation in a data set.

Alpha: 0.05

Lag	n	Coeff.	P	Std.Err.	[95% conf. interv]	Reject H0
1	399	-0.0349	0.4847	0.0501	[-0.13, 0.06]	
2	398	-0.0469	0.3482	0.0502	[-0.15, 0.05]	
3	397	0.0373	0.4552	0.0503	[-0.06, 0.14]	
4	396	0.0512	0.3055	0.0503	[-0.05, 0.15]	
5	395	0.0037	0.9403	0.0504	[-0.09, 0.10]	
6	394	-0.1129	0.0240	0.0504	[-0.21, -0.01]	X
7	393	-0.0412	0.4101	0.0505	[-0.14, 0.06]	
8	392	-0.0421	0.3998	0.0506	[-0.14, 0.06]	
9	391	0.0274	0.5839	0.0506	[-0.07, 0.13]	
10	390	-0.0869	0.0824	0.0507	[-0.19, 0.01]	
11	389	-0.0583	0.2438	0.0508	[-0.16, 0.04]	
12	388	0.0390	0.4354	0.0508	[-0.06, 0.14]	
13	387	-0.0123	0.8050	0.0509	[-0.11, 0.09]	
14	386	0.0216	0.6664	0.0510	[-0.08, 0.12]	
15	385	0.0727	0.1457	0.0510	[-0.03, 0.17]	
16	384	-0.0824	0.0994	0.0511	[-0.18, 0.02]	
17	383	0.0280	0.5756	0.0512	[-0.07, 0.13]	
18	382	0.0566	0.2578	0.0512	[-0.04, 0.16]	
19	381	0.0433	0.3867	0.0513	[-0.06, 0.14]	
20	380	-0.0096	0.8477	0.0514	[-0.11, 0.09]	

Fig. 1: Output from 'autocorr.c'

2. Runs Test

A run can be defined as a segment of a sequence consisting of adjacent elements with equal properties. The properties in question are often binary in nature, such as heads or tails, zero or one, above the mean or below etc. Consider, for example, the sequence

$$\{2, 7, 3, 18, 11, 13, 13, 2, 15, 5, 16, 11, 1, 17, 14, 12\}, \qquad \mu = 10$$

It starts with a run of three numbers below the mean followed by four numbers above the mean. Then follows three single number runs, etc. for a total of eight runs. If we count numbers above the mean as positive and numbers below the mean as negative, we can define a run as a series of consecutive positive or negative observations and denote the sequence as follows.

$$\{- \; - \; - \; + \; + \; + \; + \; - \; + \; - \; + \; + \; - \; + \; + +\}$$

The Wald-Wolfowitz runs test is a statistical test used to test the hypothesis that a sequence is random.

$$\mathrm{H}_0 : \text{The sequence is random.}$$
$$\mathrm{H}_\mathrm{A} : \text{The sequence is not random.}$$

The number of runs, R, in a (binary) sequence is normally distributed with mean, $\bar{R}$, and and variance, s_R^2:

$$\bar{R} = \frac{2n_1n_2}{n_1 + n_2} + 1$$

$$s_R^2 = \frac{2n_1n_2\left(2n_1n_2 - n_1 - n_2\right)}{\left(n_1 + n_2\right)^2\left(n_1 + n_2 - 1\right)}$$

If n_1 or n_2 is 0, the test is invalid.

Where n_1 and n_2 are the number of positive and negative observations.

The null hypothesis is rejected if $|z| > z_c$ where the test statistic

$$z = \frac{R - \bar{R}}{s_R}$$

Usually, the significance level, α, is set at 0.05, which for a two tailed test corresponds to a critical z-score of $z_c = 1.96$.

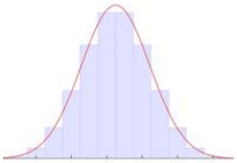
The program 'wwruns.c' uses the runs test to analyze a sequence of 40 numbers drawn by the RNG:

```
          Wald-Wolfowitz runs test.

+  +  +  -  -  +  -  +  -  -  -  -  +  +  +  -  -  -  -  +  -  -  +  +  -  -  +  -  -  -  +  +  +  -  -  +  -  -  +  -

H0: The number of runs indicates randomness.
HA: The number of runs indicates non-randomness.

Alpha: 0.05

Positive values:                17
Negative values:                23
Observed number of runs:        20
Expected number of runs:        20.55
Standard deviation:              3.05
P-value:                         0.86
Reject H0 (P < Alpha)?          NO
```

Fig. 2: Output from 'wwruns.c'

'wwruns.c' uses the p-value method, in which H_0 is rejected if the probability of seeing a deviation from $\bar{R}$, by at least as much as the observed deviation, $\left|R - \bar{R}\right|$, is less than α.

2.1. Runs test - A chi squared approach



Expected distribution of 16 elements binary runs produced by the program 'runs_exp.c'. The figure is generated by MATLAB® using the program 'runsdist.m'.

The Wald-Wolfowitz runs test rejects the null hypothesis if the sequence under scrutiny is statistically unlikely to occur, based on the number of runs. However, if events are independent of each other, then any combination of positive and negative values is possible, including long runs of only positive or only negative observations.

Consider for example a 16 elements long binary sequence. Such a sequence could form $2^{16} = 65\,536$ unique combinations of positive and negative values, each with equal chance of occuring. The number of runs, however, is normally distributed.

A good RNG should behave as if numbers were independently drawn and should therefore produce a frequency distribution of runs theoretically equal to the expected distribution. To test whether the RNG satisfies this criterion, we can employ a chi-square goodness of fit test.

The test is usually applied to binned data and is used to test the hypothesis that a sample of data came from a population with a specific distribution, in this case a normal distribution.

H_0 : The data follow the normal distribution.

H_A : The data do not follow the normal distribution.

The data are divided into k bins and the test statistic is defined as

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

The test statistic follows a chi-square distribution with $k - 1$ degrees of freedom

The null hypothesis is rejected if $\chi^2 > \chi_c^2$. Where O_i and E_i are the observed- and expected frequency for bin i, respectively.

The critical chi-square score, χ_c^2, depends on degrees of freedom and α. For 15 degrees of freedom and $\alpha = 0.1$, $\chi_c^2 = 22.307$.

Pearson's chi-square test:

H0: Observed frequencies do not differ significantly from expected frequencies.
HA: Observed frequencies differ significantly from expected frequencies.

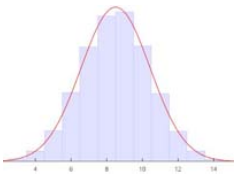
X² = 9.5990
df = 15
P = 0.8442

runs	observed	expected	residual	component
1	2	2.00	0.00	0.00
2	37	30.00	7.00	1.63
3	202	210.00	-8.00	0.30
4	906	910.00	-4.00	0.02
5	2704	2730.00	-26.00	0.25
6	6013	6006.00	7.00	0.01
7	9967	10010.00	-43.00	0.18
8	12944	12870.00	74.00	0.43
9	12799	12870.00	-71.00	0.39
10	10028	10010.00	18.00	0.03
11	5935	6006.00	-71.00	0.84
12	2821	2730.00	91.00	3.03
13	947	910.00	37.00	1.50
14	200	210.00	-10.00	0.48
15	30	30.00	0.00	0.00
16	1	2.00	-1.00	0.50

Reject H0 (P < 0.1)? NO

Fig. 3: Output from 'chiruns.c'

Above is the output from the program 'chiruns.c' that performs a chi square goodness of fit test on $65\,536$ 16-elements long binary sequences to confirm that the runs produced by the RNG does indeed follow the expected normal distribution.



Observed distribution of 16 elements binary runs produced by the program 'runs_obs.c'. The figure is generated by MATLAB® using the program 'runsdist.m'.

3. Poker Test

A poker hand consists of 5 cards drawn at random from a deck of 52. Hands are rated according to how rare they are, with certain combinations of cards occuring more infrequently than other. The frequencies with which each hand occurs can be calculated, leading to a frequency distribution. The Idea behind the poker test is to draw thousands of hands and examine, using the chi square goodness of fit test, how well to observed distribution fits the hypothetical distribution.

For example, to calculate the probability of drawing a hand with one pair, we first observe that there are 6 possible combinations of the 2 suits that form a pair for each of the 13 values in a deck, meaning that there are 78 possible pairs. For the remaining 3 cards in the hand, there are 220 ways to combine 3 values from the remaining 12, and 4 suits for each of the 3 cards.

$$\binom{13}{1}\binom{4}{2}\binom{12}{3}\binom{4}{1}^3 = 1\,098\,240$$

$\binom{n}{r} = \dfrac{n!}{r!\,(n-r)!}$
The number of possible combinations when taking r items from a larger set, n, if the order of the items is irrelavant.

Finding the total number of possible poker hands is more straightforward; we simply calculate the number of combinations when drawing 5 cards from a deck of 52:

$$\binom{52}{5} = 2\,598\,960$$

When drawing five cards without exchanges.

Dividing the number of hands with one pair by the number of possible hands yields a 42.2569% chance of drawing a pair.

The program 'poker.c' draws $2\,598\,960$ poker hands and uses the goodness of fit test to test whether the observed frequency distribution of hands matches the theoretical distribution.

```
Pearson's chi-square test:

H0: Observed frequencies do not differ significantly from expected frequencies.
HA: Observed frequencies differ significantly from expected frequencies.

   X² =   4.8771
   df =   9
    P =   0.8449

hand             observed     expected     residual     component
-------------------------------------------------------------------
High Card         1303340   1302540.00       800.00          0.49
One Pair          1097606   1098240.00      -634.00          0.37
Two Pair           123267    123552.00      -285.00          0.66
Three of a kind     54811     54912.00      -101.00          0.19
Straight            10294     10200.00        94.00          0.87
Flush                5185      5108.00        77.00          1.16
Full house           3804      3744.00        60.00          0.96
Four of a kind        614       624.00       -10.00          0.16
Straight flush         35        36.00        -1.00          0.03
Royal flush             4         4.00         0.00          0.00
-------------------------------------------------------------------
Reject H0 (P < 0.1)?      NO
```

Fig. 4: Output from 'poker.c'

The deck is sorted between each hand and in the following order:

$$\{2\clubsuit,\ 2\diamondsuit,\ 2\heartsuit,\ 2\spadesuit, 3\clubsuit,\ \ldots,\ A\heartsuit,\ A\spadesuit\}.$$

Cards are then drawn from the deck at random with numbers from 0 to 51 representing positions in a full deck.

Sorting the deck between each hand and not shuffeling it reduces implementation bias, that is bias due to how the test is implemented. Not sorting the deck between drawing each hand or shuffeling it introduces a second layer of randomness that will most likely mask potential weaknesses in the RNG.

While the autocorrelation- and runs tests were targeting specific properties, the poker test is a more general approach to testing as deviations from the hypothetical distribution can be caused by any number of reasons. The previously mentioned flaw in the linear congruential generator that causes alternating odd and even numbers, for example, will cause an overabundance of flushes if hands are drawn from a deck sorted in the manner described above.

4. Spectral Test

Linear congruential generators have a property that when plotted in two or more dimensions, lines or hyperplanes will form on which all possible outcomes can be found. The purpose of the spectral test is to examine the structure of the hyperplanes as a way to evaluate the quality of the generator. In short, the further apart the hyperplanes are, the worse the generator is.

The MATLAB® program 'plot_randu.m' demonstrates the formation of hyperplanes by plotting points generated by the IBM® subroutine RANDU in three dimensions. When plotting points consisting of three consecutive numbers scaled to the interval $[0, 1]$, 15 planes are formed, on which all possible outcomes can be found.

IBM's RANDU is, according to Wikipedia, widely considered to be one of the most ill-conceived random number generators ever designed.

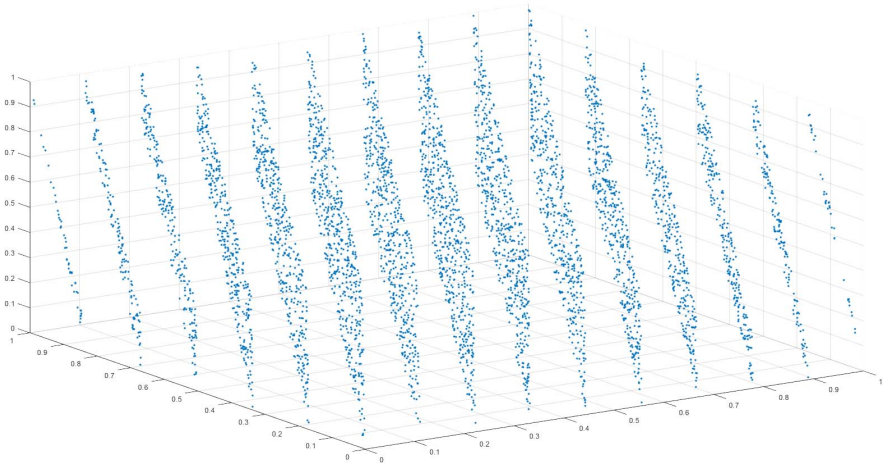


Fig. 5: Output from 'plot_randu.m'

In comparison, the points generated by the RNG form 202 hyperplanes in three dimensional space when each number is rounded to two decimals and can only be seen when viewed, as in Fig. 6, from an angle perpendicular to the planes. Of course, a satisfying performance in $\mathbb{R}^3$ does not guarantee that the generator won't fail spectacularly in higher dimensions.

The systematic behavior of linear congruential generators might lead us to conclude that they are essentially worthless, but Knuth* points out that even if we take truly random numbers between 0 and 1, and round or truncate them to finite accuracy, then the t-dimensional points we obtain will have an extremenly regular character.

**Knuth, Donald E. (1997), 'The Art of Computer Programming 3rd ed., Vol. 2 - Seminumerical Algorithms', p. 94.*

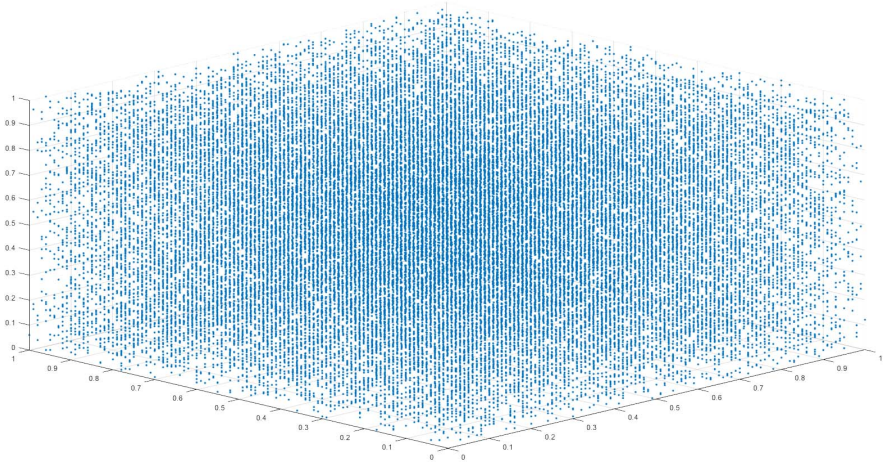


Fig. 6: Output from 'plot_rng.m'

'plot_rng.m reads from a datafile generated by the program 'spec-data.c'