Random Number Generation with `acorni`: A Warning Note

Julián M. Ortiz (jmo1@ualberta.ca) and Clayton V. Deutsch (cdeutsch@ualberta.ca)
Department of Civil & Environmental Engineering
University of Alberta

Abstract

The use of `acorni` *in scripts is common. Initializing the random number generator with seeds that are incremented by a constant value facilitates the generation of multiple realizations. However, if the seed numbers are shifted by a constant, there is a constant difference between the random numbers generated. This can introduce artifact correlation in the results of multiple variables. Some examples are provided in this note and a simple solution is suggested to avoid this problem.*

The Additive Congruential Random Number Generator

Additive generators calculate each pseudo-random number as some additive combination of the previous numbers in the sequence [2]. R. S. Wikramaratna [3, 4, 5] proposed the `acorn` generator and its integer version `acorni`. The latter version is used in GSLIB [1].

The K^{th} order generator X_j^K combines the previous number in the sequence with a corresponding number from the $(K-1)^{th}$ order sequence. X_j^K is defined recursively from a seed X_0^0 ($0 < X_0^0 < maxint$) and a set of K initial values X_0^k, k=1, ..., K each satisfying $0 \leq X_0^k < maxint$, where $maxint = 2^{30}$, just over a billion. The generator is defined by:

$$\begin{aligned} X_n^0 &= X_{n-1}^0, && n \geq 1 \\ X_n^k &= \frac{(X_n^{k-1} + X_{n-1}^k) \text{mod } maxint}{maxint}, && n \geq 1, \quad k = 1, ..., K \end{aligned}$$

This generator has three main features:

1. It is faster to compute than other generators,
2. The period length can be set arbitrarily large, and
3. It gives the same sequence in any machine (differing only in the number of significant digits).

The current implementation of `acorni` is shown in **Figure 1**. All the numbers in the Zero Order row are the same (the seed number X_0^0 equal to a constant C). The arrows show

which previous numbers are used to calculate the current one. The numbers generated in the row of the K^{th} order are considered to be pseudo-random numbers, in this case $K = 12$.

The standard implementation of this random number generator considers K additional seeds that could be specified by the user, but that are set to zero by default, $X_0^1 = X_0^2 = \ldots = X_0^K = 0$. It is recommended to initialize the sequence not considering the first thousands. In most cases, the random number generator is initialized by running it up to a certain value, say 10000. The random numbers needed are then taken starting at the 10000^{th} value in the sequence.

Under this implementation, and considering a seed number C, the generation of random numbers would be as shown in **Figure 1**. It can be seen that the random numbers in the 12^{th} row are equal to the constant seed C multiplied by a factor m. This factor changes as we consider new terms in the sequence of pseudo-random numbers. To obtain the final sequence of random numbers in $]0, 1]$, the remainder of dividing the terms in the last row by $maxint$ must be taken, that is, we find the value $m \cdot C - p \cdot maxint$, for some p such that the new value is in $]0, maxint]$. Then this value is standardized to the interval $]0, 1]$ by dividing it by $maxint$ one more time.

$$
\begin{array}{lllccccccc}
Zero & Order: & C & \rightarrow & C & \rightarrow & C & \rightarrow & C & \rightarrow & \cdots \\
 & & & & \downarrow & & \downarrow & & \downarrow & & \\
1^{st} & Order: & 0 & \rightarrow & C & \rightarrow & 2 \cdot C & \rightarrow & 3 \cdot C & \rightarrow & \cdots \\
 & & & & \downarrow & & \downarrow & & \downarrow & & \\
2^{nd} & Order: & 0 & \rightarrow & C & \rightarrow & 3 \cdot C & \rightarrow & 6 \cdot C & \rightarrow & \cdots \\
 & & & & \downarrow & & \downarrow & & \downarrow & & \\
 & & & & \vdots & & \vdots & & \vdots & & \\
 & & & & \downarrow & & \downarrow & & \downarrow & & \\
12^{th} & Order: & 0 & \rightarrow & C & \rightarrow & 12 \cdot C & \rightarrow & 81 \cdot C & \rightarrow & \cdots
\end{array}
$$

Figure 1: Schematic showing how `acorni` generates pseudo-random numbers.

The Problem

Consider the use of `acorni` within a script, where multiple realizations are required. Common practice consists on calling the simulation algorithm several times, but changing the random number seed C. This change is commonly done by adding a constant to the previous random number seed, starting the new realization with a seed number $C + \Delta C$.

Considering the N^{th} value of two different sequences generated with seed numbers C and $C + \Delta C$, there are two possibilities:

1. If $m \cdot (C + \Delta C) - p \cdot maxint < maxint$, then:

$$
\left.
\begin{array}{l}
\text{Random Number 1} = \frac{m \cdot C - p \cdot maxint}{maxint} \\
\text{Random Number 2} = \frac{m \cdot (C + \Delta C) - p \cdot maxint}{maxint}
\end{array}
\right\} \text{Difference} = \frac{-m \cdot \Delta C}{maxint} \tag{1}
$$

(1)	(2)	(3)=(1)-(2)	(4)	(5)	(6)=(4)-(5)	(7)=(3)-(6)
Seed 1	Seed 2	Difference	Seed 3	Seed 4	Difference	
69069	69071		69073	69075		
0.8737	0.5058	0.3679	0.1380	0.7701	-0.6321	1.00
0.9092	0.6511	0.2582	0.3929	0.1347	0.2582	0.00
0.6413	0.4659	0.1754	0.2905	0.1151	0.1754	0.00
0.1061	0.9463	-0.8402	0.7865	0.6266	0.1598	-1.00
0.6178	0.5990	0.0188	0.5802	0.5614	0.0188	0.00
0.5318	0.0141	0.5177	0.4963	0.9786	-0.4823	1.00
0.5575	0.7861	-0.2286	0.0147	0.2433	-0.2286	0.00
0.5872	0.4639	0.1233	0.3406	0.2174	0.1233	0.00
0.8279	0.7783	0.0497	0.7286	0.6790	0.0497	0.00
0.2021	0.0196	0.1826	0.8370	0.6545	0.1826	0.00

Table 1: Example: the last ten numbers of four sequences of one million numbers are shown. The difference between numbers in sequences that differ in their seed are calculated. These differences are equal for two pairs of sequences, or they are shifted by one.

2. If $m \cdot (C + \Delta C) - p \cdot maxint > maxint$ then:

$$\left.\begin{array}{l} \text{Random Number 1} = \frac{m \cdot C - p \cdot maxint}{maxint} \\ \text{Random Number 2} = \frac{m \cdot (C+\Delta C) - (p+1) \cdot maxint}{maxint} \end{array}\right\} \text{Difference} = \frac{-m \cdot \Delta C + maxint}{maxint} \quad (2)$$

It can be noticed that the difference does not depend on the seed number C, but on the increment between the two seed numbers ΔC. Hence, if we take the same N^{th} number in two other sequences, with seed numbers D and $D + \Delta C$, then the difference between these two random numbers will be either $\frac{-m \cdot \Delta C}{maxint}$ or $\frac{-m \cdot \Delta C + maxint}{maxint}$, that is, the difference will be the same as the one obtained from the two original sequences, or it will differ by one.

To further illustrate this point, consider the sequences presented in **Table 1**. One million numbers were generated with each seed number and the last ten are shown on the table. Considering a constant increment of $\Delta C = 2$ between the random number seeds, four sequences are generated and the difference between the numbers in sequences 69069 and 69071, and between sequences 69073 and 69075 are calculated. From the table, it can be seen that these differences are equal for the same number in the sequences. For instance, the difference between the one millionth number in the first and second sequences is 0.1826, the same as the difference between the one millionth numbers in the third and fourth sequences.

A Proposed Solution

A simple solution to this problem is to initialize the sequences differently, say, as a function of the seed number. Instead of discarding the first 10000 numbers of the sequence, as in the current implementation of most algorithms in `GSLIB`, we propose to initialize the random sequence by discarding the first C values of it, where C is also the seed number. That is, the first number the user will see for the first sequence will be the C^{th} number in the

sequence. For the second sequence of random numbers, the first value the user will see is the $(C + \Delta C)^{th}$ number. If other two sequences are considered with seed numbers differing by ΔC, no correlation should be found between differences, since in the expressions 1 and 2, the value of m will no longer be the same for different sequences.

References

[1] C. V. Deutsch and A. G. Journel. *GSLIB: Geostatistical Software Library and User's Guide*. Oxford University Press, New York, 2nd edition, 1998.

[2] J. Ortiz C. and C. V. Deutsch. Testing pseudo-random number generators. In *Centre For Computational Geostatistics*, volume 3, Edmonton, AB, 2001.

[3] R. S. Wikramaratna. ACORN - a new method for generating sequences of uniformly distributed pseudo-random numbers. *Journal of Computational Physics*, 83:16–31, 1989.

[4] R. S. Wikramaratna. ACORN random number generator user documentation. User Documentation, October 1990.

[5] R. S. Wikramaratna. Theoretical analysis of the ACORN random number generator, 1990. SIAM Conference on Applied Probability in Science and Engineering, New Orleans, Louisiana.