Simple Random Sampling: Not So Simple

Kellie Ottoboni and Philip B. Stark

September 11, 2018

Abstract

R (Version 3.5.0) generates random integers between 1 and m by multiplying random floats by m, taking the floor, and adding 1 to the result. It is well known that quantization effects inherent in this approach result in a non-uniform distribution on $\{1, \ldots, m\}$. The difference, which depends on m, can be substantial. Because the `sample` function in R relies on generating random integers, random sampling in R is also biased. There is an easy fix, namely, construct random integers directly from random bits, rather than multiplying a random float by m. That is the strategy taken in Python's `numpy.random.randint()` function, among others.

A textbook way to generate a random integer on $\{1, \ldots, m\}$ is to start with $X \sim U[0, 1)$ and define $Y \equiv 1 + \lfloor mX \rfloor$. If X is truly uniform on $[0, 1)$, Y is then uniform on $\{1, \ldots, m\}$. However, if X has a discrete distribution derived by scaling a pseudorandom binary integer, the resulting distribution is not uniformly distributed on $\{1, \ldots, m\}$ even if the underlying pseudorandom number generator (PRNG) is perfect (unless m is a power of 2):

Theorem 0.1 (Knuth [1997])**.** *Suppose X is uniformly distributed on w-bit binary numbers, and let $Y_m \equiv 1 + \lfloor mX \rfloor$. Let $p_+(m) = \max_{1 \le k \le m} \Pr\{Y_m = k\}$ and $p_-(m) = \min_{1 \le k \le m} \Pr\{Y_m = k\}$. There exists $m < 2^w$ such that, to first order, $p_+(m)/p_-(m) = 1 + m2^{-w+1}$.*

The algorithm that R (Version 3.5.0) [R Core Team, 2018] uses to generate uniform random integers has this issue (albeit in a slightly more complicated form, because, depending on m, R uses pseudorandom binary integers of different lengths). Because `sample` relies on random integers, it inherits the problem.

R uses `unif_rand` to generate pseudorandom numbers with word size at most $w = 32$. To generate integers with a larger word size, R combines two w-bit integers to obtain an integer with 50 to 53 bits (depending the chosen PRNG).

A better way to generate random elements of $\{1, \ldots, m\}$ is to use pseudorandom bits directly. The integer m can be represented with $\mu = \lceil \log_2(m) \rceil$ bits. To generate a pseudorandom integer between 1 and m, first generate μ pseudorandom bits (for instance, by taking the most significant μ bits from the PRNG output). If that binary number M is larger than $m - 1$, discard it. Repeat until the μ bits represent an integer M that is less than m. Return $M + 1$.[1] This procedure might discard almost half the draws if m is slightly larger than a power of 2, but if the input bits were IID Bernoulli(1/2), the resulting integers will be uniformly distributed. This is how the Python function `numpy.random.randint()` (Version 1.14) generates pseudorandom integers.[2]

The R `sample` function has a branch in its logic depending on the number of elements in the population to be sampled. It uses `ru` when $m >= 2^{31}$ and `rand_unif`

[1] See Knuth [1997] p.114.

[2] However, Python's built-in `random.choice()` (Versions 2.7 through 3.6) does something else biased: it finds the closest integer to mX, where X is a binary fraction between 0 and 1.

when $m < 2^{31}$.[3] The nonuniformity of selection probabilities is largest when m is just below 2^{31}. In that case, `sample` calls `unif_rand`, which gives outputs with word size $w = 32$. The maximum ratio of selection probabilities approaches 2 as m increases to the cutoff 2^{31}, or about 2 billion. Even if m is close to 1 million, the ratio is about 1.0004.

When $m \geq 2^{31}$, R calls `ru()` to produce a pseudorandom number with word length at least $w = 50$ bits and at most $w = 53$ bits (depending the chosen PRNG). The algorithm uses two pseudorandom numbers, r_1 and r_2, to produce one with greater precision,

$$\texttt{ru}() = \frac{\lfloor Ur_1 \rfloor + r_2}{U},$$

where $U = 2^{25} - 1$. In theory, this method would produce equidistant pseudorandom numbers on $[0, 1)$ by using the entire 53-bit mantissa of floating point numbers. It fails due to floating point arithmetic.

To illustrate the issue, suppose that the PRNG generating r_1 and r_2 produces 4-bit integers and that floating point numbers are represented in base 2 with 4 bits of precision. Let $U = 2^4$. Imagine that $r_1 = 0$, represented as 0.000×2^0. Then the possible values of `ru` would be equally spaced values, separated by a distance of U^{-2}. Instead, imagine r_1 is the largest value that the PRNG can generate, represented as 1.000×2^{-1} in floating point. Multiplying by U only changes the exponent from -1 to 3. In floating point arithmetic, adding $\lfloor Ur_1 \rfloor$ to small values of r_2 will result in underflow: for $r_2 < 1.000 \times 2^0$, $\lfloor Ur_1 \rfloor + r_2 = \lfloor Ur_1 \rfloor$. Moreover, $\lfloor Ur_1 \rfloor + 1.000 \times 2^0 = \lfloor Ur_1 \rfloor + 1.001 \times 2^0 = \lfloor Ur_1 \rfloor + 1.111 \times 2^0$ and so forth. In this case, `ru` will give equally spaced values separated by a distance of U^{-1}.

Knuth's theorem no longer applies to the ratio of selection probabilities in the

[3] A different function, `sample2`, is called when $m > 10^7$ and $k < m/2$. It uses the same flawed method of generating pseudorandom integers.

branch of the logic that calls `ru`, as values aren't truly uniformly distributed. In fact, the nonuniformity of pseudorandom numbers might exacerbate the pseudorandom integer generation problem further. This could be avoided by using pseudorandom bits directly to gain precision, for example by concatenating two 32-bit pseudorandom numbers or by using a 64-bit PRNG (Marsaglia and Tsang [2004]).

Population size (m)	Word length (w)	Maximum ratio of selection probabilities
10^6	32	1.0004
10^9	32	1.466
$2^{31} - \epsilon$	32	2
$2^{31} + \epsilon$	53	$1 + 2.3 \times 10^{-7}$
10^{12}	53	1.0001
10^{15}	53	1.11

Table 1: Maximum ratio of selection probabilities for different population sizes m. R constructs random integers using random binary integers with different numbers of bits (word lengths) depending on m: if $m < 2^{31}$, the word length is between 25 and 32, otherwise it may be up to 53 bits. The maximum ratio of selection probabilities is based on Knuth's theorem; for $m > 2^{31}$, the maximum may be even larger due to nonuniformity introduced by the `ru` function.

We recommend that the R developers replace the current algorithm for generating pseudorandom integers with the masking algorithm and use pseudorandom bits directly to generate pseudorandom numbers with more than 32 bits of precision.

References

Donald E. Knuth. *Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley Professional, Reading, Mass, 3 edition edition, November 1997. ISBN 978-0-201-89684-8.

George Marsaglia and Wai Wan Tsang. The 64-bit universal RNG. *Statistics & Probability Letters*, 66(2):183–187, January 2004.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2018. URL `https://www.R-project.org`.