A Simple Monte Carlo Experiment

September 10, 2026

Simple Monte Carlo Experiment

Conceptual Model
We have an arbitrarily large urn of marbles labeled with distinct values in $[0, N-1]$ (only 1 marble with each number). We would like to estimate the value of N (and not by inspecting every single marble).

Specification Model We can estimate N by drawing z marbles (x_is, **with replacement**) and calculating their average value $\bar{x}$. We expect this to be near the true average, μ, which can be written using Gauss' Law

$$\bar{x} = \frac{1}{z}\sum_{i=1}^{z} x_i \approx \mu = \frac{1}{N}\sum_{i=0}^{N-1} i = \frac{1}{N}\left(\frac{(N-1)(N)}{2}\right) = \frac{N-1}{2} \quad \rightarrow \quad N \approx 1 + \frac{2}{z}\sum_{i=1}^{z} x_i$$

We are **given** N, we will simulate the above sample and averaging procedure arriving at a **simulated estimate** for N — if our simulation can be validated, we can use this approach in the real world!

Simple Monte Carlo Experiment

Conceptual Model
We have an arbitrarily large urn of marbles labeled with distinct values in $[0, N-1]$ (only 1 marble with each number). We would like to estimate the value of N (and not by inspecting every single marble).
Specification Model We can estimate N by drawing z marbles (x_is, **with replacement**) and calculating their average value $\bar{x}$. We expect this to be near the true average, μ, which can be written using Gauss' Law

$$\bar{x} = \frac{1}{z}\sum_{i=1}^{z} x_i \approx \mu = \frac{1}{N}\sum_{i=0}^{N-1} i = \frac{1}{N}\left(\frac{(N-1)(N)}{2}\right) = \frac{N-1}{2} \quad \rightarrow \quad N \approx 1 + \frac{2}{z}\sum_{i=1}^{z} x_i$$

We are **given** N, we will simulate the above sample and averaging procedure arriving at a **simulated estimate** for N — if our simulation can be validated, we can use this approach in the real world!
2 minutes — Your language library doesn't have a $Random()$ providing $u \in (0,1)$, instead it has `RandomInteger()` with with range $[0, \texttt{RAND\_MAX}]$.
How would your group design a computational model?

Simple Monte Carlo Experiment

Computational Model

Program `simple-monte-carlo.c` takes as input N and calculates the averages from small samplings ($z = \lfloor 0.15N \rfloor$, $\{x_i\}_{i=1}^{z}$) of integral values on $[0, N-1]$.

We use a pRNG library with a `RandomInteger()` function that returns values within $[0, \texttt{RAND\_MAX}]$, $\texttt{RAND\_MAX} > N$.

We simulate the drawing and replacement of a labeled marble x_i with

$$x_i \leftarrow \texttt{RandomInteger()} \mod \texttt{N}$$

We calculate the average of each k sample and track how many $\bar{x}_k < \mu$, and how many $\bar{x}_k \geq \mu$ in a `counts[2]` array.

"...view the source, Luke!"

Simple Monte Carlo Experiment

Verification

For verification, we'll simply observe if many $\bar{x}_k$s fall symmetrically about the known, true μ.

Simple Monte Carlo Experiment

Verification

For verification, we'll simply observe if many $\bar{x}_k$s fall symmetrically about the known, true μ.

Demonstrate...

```
# n not a variable, it's a command; N=10000, implicit sample size z=0.15N, SEED?
$ ./simple-monte-carlo n 10000 SEED
```

Simple Monte Carlo Experiment

Verification

For verification, we'll simply observe if many $\bar{x}_k$s fall symmetrically about the known, true μ.

Demonstrate...

```
# n not a variable, it's a command; N=10000, implicit sample size z=0.15N, SEED?
$ ./simple-monte-carlo n 10000 SEED
```

Adding a fourth command line parameter allows many "samples" to be drawn, each with their own $\bar{x}_k$ estimate of μ. In simulation speak, we call these ***replications*** (n). In this case intermediate console reports show:

- a particular k^{th} experiment's results,
- the total number of sample $\bar{x}_k$s that have been to the left and right of μ

```
# N=10000, SEED?, replications=100
$ ./simple-monte-carlo n 10000 SEED 100
```

If we collect many samples ($k = 1, \ldots, B$) for a single experiment, each with an $\bar{x}_k$, how do we expect this set of $\{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k, \ldots, \bar{x}_B\}$ to be distributed around the true μ?

drumroll please

Oops

Clearly

$$\texttt{RandomInteger()} \bmod \texttt{N}$$

doesn't perform quite as advertised for some `N`s.

Why? We'll answer this later in lecture, first math (yay!)

`equilikely-monte-carlo n 1717600 SEED 1000`

The author says to select random array elements with an `[Equilikely(0,n-1)]`...

This sounds a lot like drawing $[0, N-1]$ labeled marbles out of an urn...

Could it be that $Equilikely(a,b)$ fixes our `simple-monte-carlo` problems?
Demonstration...

`equilikely-monte-carlo n 1717600 SEED 1000`

The author says to select random array elements with an `[ Equilikely(0,n-1) ]`...

This sounds a lot like drawing $[0, N-1]$ labeled marbles out of an urn...

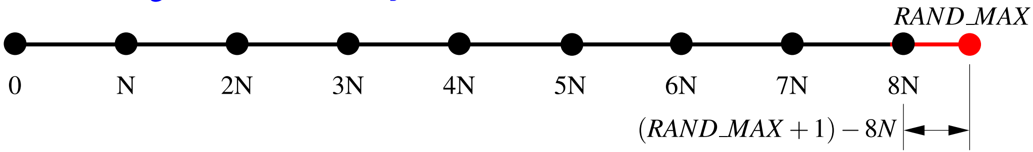
Could it be that $Equilikely(a, b)$ fixes our `simple-monte-carlo` problems? ***Demonstration***...

But `./equilikely-monte-carlo` working doesn't explain why

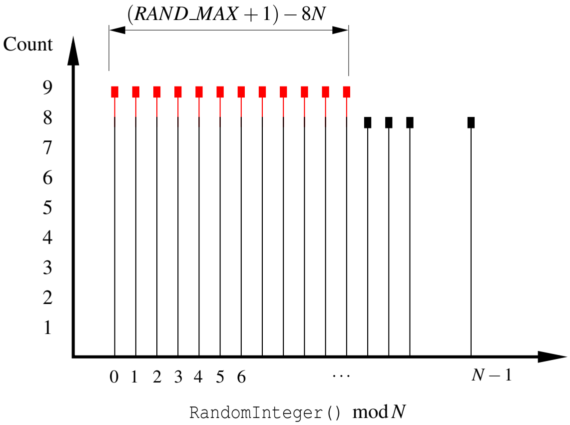
$$x_i \leftarrow \texttt{RandomInteger()} \mod \texttt{N}$$

was wrong!

Range of `RandomInteger()` Partitioned into N Sized Chunks



Data Histogram of `RandomInteger()` mod N



The red residual of the `RandomInteger()` range contributes disproportionally to the smaller values of the

`RandomInteger()` mod N

histogram.

Depending on `RAND_MAX` and N, and the use case of results, this *might* be negligible.

If it is not, drastically wrong results can occur.

`RAND_MAX`, the largest value returned by `RandomInteger()` (not always ρ), is largely independent of the underlying pRNG. It is tied to the machine+software architecture!
So choosing a “better” pRNG doesn’t make this technique OK!

$Equilikely(a,b)$: A Solution to `RandomInteger()` mod N...

Is there a **safe range** to use the simple

`RandomInteger()` mod N

technique for random integral values?

Who cares! — just use the correct algorithm (`Equilikely(a,b)`)

1. Independent of the pRNG used, its period (ρ), and the architecture (`int` size)
2. Independent of `N`
3. You still have to generate one random number (no savings there)
4. Proper `Random()` functions return $u \in (0,1)$ already, **not integers**
5. "Technically correct" is the best kind of correct `:)`

Why care? — Consider the canonical **Fisher-Yates** shuffling algorithm, web examples for which almost always use the `RandomInteger()` mod N technique for choosing the next array element. While this technique may be sufficient for small array sizes, this demonstration suggests **it does not scale** to large arrays. "Big Data" anyone?