Pseudorandom Number Generation

Mark M. Fredrickson (mfredric@umich.edu)

Computational Methods in Statistics and Data Science (Stats 406)

Random Number Generation

Example from First Lecture: Sample Maximum Magnitude

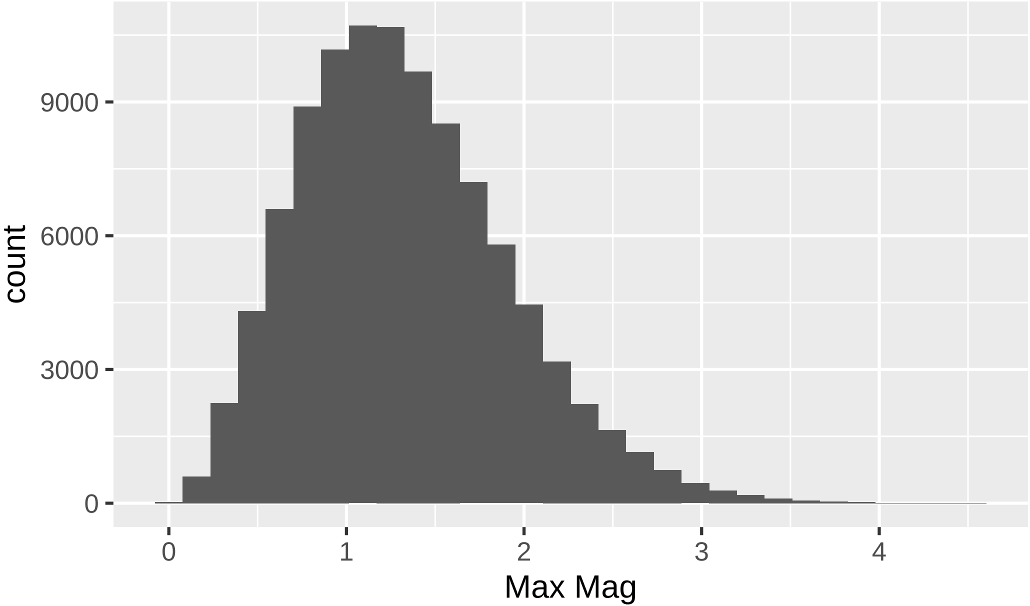
What is the distribution of the **sample maximum magnitude**?

$$M_3 = \max_{i=1,2,3} |X_i|, \quad X_i \overset{\text{iid}}{\sim} N(0,1)$$

Generate many replicates of (X_1, X_2, X_3) and find the **empirical distribution**.

In R, the **random number generator** for standard Normal variables is `rnorm`:

```
> sample_max_mag <- function(n) {
+       x <- rnorm(n)
+       max(abs(x))
+ }
> empirical_distrib <- replicate(100000, sample_max_mag(3))
```



count
9000
6000
3000
0
0
1
2
3
4
Max Mag

Why do we need random numbers?

Question: Aren't our data random enough? Why do we need to generate RVs?

- Estimate properties of **random variables**, especially **sampling distributions** of statistics
- Monte Carlo methods for **integration**, **hypothesis tests**, **estimation**
- **Resampling methods** such as the **bootstrap**, **permutation tests**, **cross validation**
- **Random assignment** in controlled trials, **random sampling** from enumerated populations
- **Simulating** natural processes
- Adding **noise** to data for privacy reason
- **Cryptography**
- Non-deterministic **optimization**

Random vs. Pseudorandom

We will say that an event is **random** if, given all possible information, we cannot predict it.

Deterministic vs. random:

$$f(x) = a + bx \quad \text{vs} \quad f(x) = a + bx + \epsilon, \; \epsilon \sim N(0, 1)$$

True randomness can be achieved with **physical devices**, but these are

- costly
- slow
- cannot be easily reproduced

We will use **pseudorandom numbers** that are "close enough" to true random numbers.

Aside: Getting true random numbers in R

Using `random.org`:

```
> library(random) # you need to install this with `install.packages`
> system.time(rand.org <- randomNumbers(10, 0, 255))

   user  system elapsed
  0.030   0.012   1.667
```

```
> system.time(devrand <- replicate(10,
+   as.integer(packBits(rawToBits(charToRaw(
+   system("dd if=/dev/random bs=1 count=1 status=none", TRUE)))))))

   user  system elapsed
  0.056   0.042   0.104

> system.time(pseudo.randoms <- runif(5, 0, 255))

   user  system elapsed
  0.000   0.001   0.001
```

Pseudrandom Number Generators

Our goal is **uniform, IID** bits (0 or 1) with $\Pr(B_i = 1) = 0.5$.

As computer numbers are represented as groups of **bits** (0 or 1) this is equivalent to getting IID

$$U \sim \text{Uniform}(0, 1) : 0 < U < 1, \; f(u) = 1, \; F(u) = u$$

A **pseudorandom number generator (PRNG)** is a function that takes a **seed** and produces **a (pseudo)random number and a new seed**:

```
> prng <- function(seed) {
+     # do something
+     c(random_number, new_seed)
+ }
```

A very simple PRNG

```
> lcg <- function(seed) {
+   a <- 5 ; c <- 1 ; m <- 8
+   new_number <- (a * seed + c) %% m
+   c(new_number, new_number) # using the new number as the seed
+ }
> lcg(7)

[1] 4 4
```

Using the `lcg`

```r
> rngs <- numeric(length = 16)
> seed <- 7
> for (i in 1:16) {
+     rs <- lcg(seed)
+     rngs[i] <- rs[1] # pull out the first item in the vector
+     seed <- rs[2]
+ }
```

```
 [1] 4 5 2 3 0 1 6 7 4 5 2 3 0 1 6 7
```

```r
> rngs / 7 # scale to [0, 1]
```

```
 [1] 0.5714 0.7143 0.2857 0.4286 0.0000 0.1429 0.8571 1.0000
 [9] 0.5714 0.7143 0.2857 0.4286 0.0000 0.1429 0.8571 1.0000
```

Evaluating PRNGs

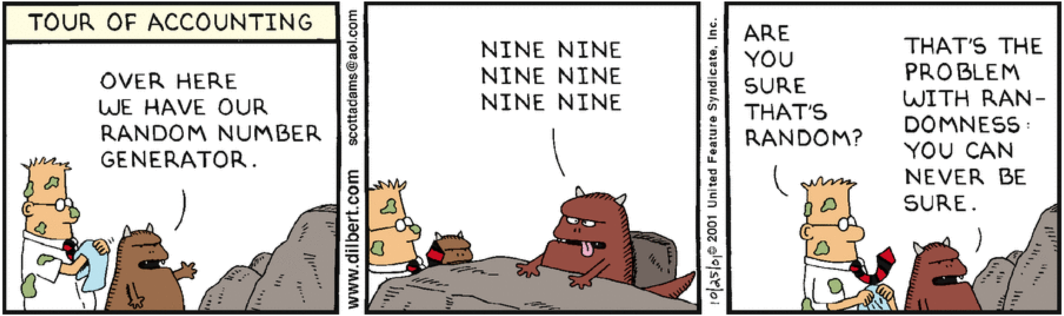


Figure 1: ©2001 Scott Adams

But you can **reject with high confidence** hypothesis tests.

Test suites exist that implement these tests (`TestU01` and `diehard`).

PRNGs in R

R implements several pseudorandom number generators. See the help page for `Random`.

The default ("Marsenne-Twister") has a period of $2^{19937} - 1$ (compared to our example of 2^8) and it maps multiple seeds to the same values. It is reasonably fast as well.

You can set the random seed with

```
> set.seed(3949392)
```

The full seed

In fact, for the default MT the real seed is composed of 624 integers!

```
> length(.Random.seed) # first two items are book keeping
[1] 626
> .Random.seed[3:6] # just a few elements
[1]  1173446700 -1537720515  1428012250  1749109907
```

PRNGs: Summary

Pseudorandom Number Generators provide:

- Fast
- Cheap
- Reproducible

streams of **numbers that look almost random**.

A PRNG is a **deterministic function** that takes a **seed** and returns a **random number and a new seed**.

We will develop tools for turning **uniform (0,1) RVs** into other distributions.