

The Law of Large Numbers for Coin Tossing

INTRODUCTION

Suppose a coin has $P(\text{Heads}) = P(\text{H}) = p$, where $0 < p < 1$, on each toss. (We also assume that tosses of the coin are independent, so that whether we get H or T on any one toss is not influenced by the results on other tosses.)

For $i = 1, 2, 3, \dots$; let $X_i = 0$ or 1 according as the i th toss results in T or H, respectively.

Let

$$S_n = X_1 + X_2 + \dots + X_n$$

be the number of Heads seen in the first n tosses.

And let

$$R_n = S_n/n$$

be the proportion of heads in the first n tosses.

LAW OF LARGE NUMBERS

Then the Law of Large Numbers (LLN) asserts that, in a certain sense,

$$R_n \rightarrow p, \text{ as } n \rightarrow \infty.$$

As an example of this coin-tossing model:

Suppose that in 10 tosses of a fair coin ($p = 1/2$) coin we obtained the sequence shown on the second column of the table below.

Then the corresponding values of X_i , S_i and R_i , for $i = 1, \dots, 10$ are as shown in other rows.

i	Toss	X_i	S_i	R_i
1	H	1	1	1.00000
2	H	1	2	1.00000
3	H	1	3	1.00000
4	H	1	4	1.00000
5	H	1	5	1.00000
6	T	0	5	0.83333
7	H	1	6	0.85714
8	H	1	7	0.87500
9	T	0	7	0.77778
10	T	0	7	0.70000

TYPES OF CONVERGENCE

The kind of convergence you have encountered in a beginning calculus course involves numerical sequences.

For example: $a_n = (1 + 1/n)^n \rightarrow e \approx 2.71828$, where

- a_n is a *deterministic* sequence
- For example: we *always* have $a_{10} = 1.1^{10} \approx 2.5937$.

The kind of convergence involved in the LLN (“convergence in probability”) is different because

- R_n is a *random* sequence depending on coin tosses.
- In the example above, $R_{10} = 0.6$.
- But different sequences of random coin tosses give various results.

The LLN can be proved from the axioms of probability.

But for now, it is sufficient to state—and to illustrate by simulation—that for large enough n , we will likely get

$$R_n \approx p, \text{ to any desired degree of accuracy.}$$

Specifically, for a fair coin we will likely see

- $R_{10\,000} = 0.50 \pm 0.01$ and
- $R_{50\,000} = 0.500 \pm 0.005$, and so on.

(Here, *likely* means 95% of the time.)

SIMULATION

Because it is not feasible to perform so many tosses with a *real* coin, we use the software package R to *simulate* coin tosses on a computer.

- There are technical difficulties in getting random-appearing results with a computer program.
- There is reliable evidence that the "random" results generated by R are suitable for our demonstration. (R has several random number generators. We use the default.)
- But one should not necessarily trust the random number generators in commercial packages (for example, C++ or Excel) to do a satisfactory job.

All generators eventually repeat the same sequence of values — after a "period" d .

Default RNG (“Marsaglia-Multicarry”) in R has period $d > 10^{18}$.

An alternate RNG in R (“Mersenne Twister”) has period $d > 10^{8668}$.

Some commercial versions of programming languages and business software have $d < 10\text{M}$, some even have $d < 100\text{K}$. Also, d doesn't tell the whole story of quality.

R FUNCTIONS / COMMANDS

The following R script does the desired simulation and produces the corresponding graph of R_n against n .

In R the code

- `x <- sample(0:1, n, repl=T)` creates a vector **x** of **n** randomly chosen 0s and 1s based on a fair coin.
- `s <- cumsum(x)` creates a vector **s** of cumulative sums, and
- `r <- s/(1:n)` does elementwise division to produce a vector of Heads ratios as illustrated in the table above.
- The notation `1:5` gives the vector (1, 2, 3, 4, 5), etc.
- A semicolon permits more than one statement per line.
- `plot` function produces the figure shown.
- `lines` function plots the horizontal reference line.
- The last line of code prints results for the first 10 simulated tosses shown in the table above and beneath the code (`cbind` makes an $n \times 3$ matrix) and the final value of **r**.

SEEDS

A random number generator is essentially a very long list of numbers that pass all known tests looking as if they result from a truly random process.

The seed says where to begin in the list.

- The seed can be specified by the user.
(Useful in debugging.)
- Usually, the seed is set using the system clock.

Using `set.seed(1212)`, the program always does the simulation shown. (Provided the default generator is used in R.)

Without `set.seed`:

- Different runs of the program give quite variable results for small numbers of tosses.
- All runs show a path that stabilizes near 1/2 for large numbers of tosses.
- The run shown is a “lucky” one. Not all runs come this close to 1/2 after as few as 20,000 simulated tosses.

CODE

```
set.seed(1212)

n <- 50000

x <- sample(0:1, n, repl=T)

s <- cumsum(x); r <- s/(1:n)

plot(r, ylim=c(.4, .6), type="l")

lines(c(0,n), c(.5,.5))

round(cbind(x,s,r), 5)[1:10,]; r[n]
```

PRINTED OUTPUT

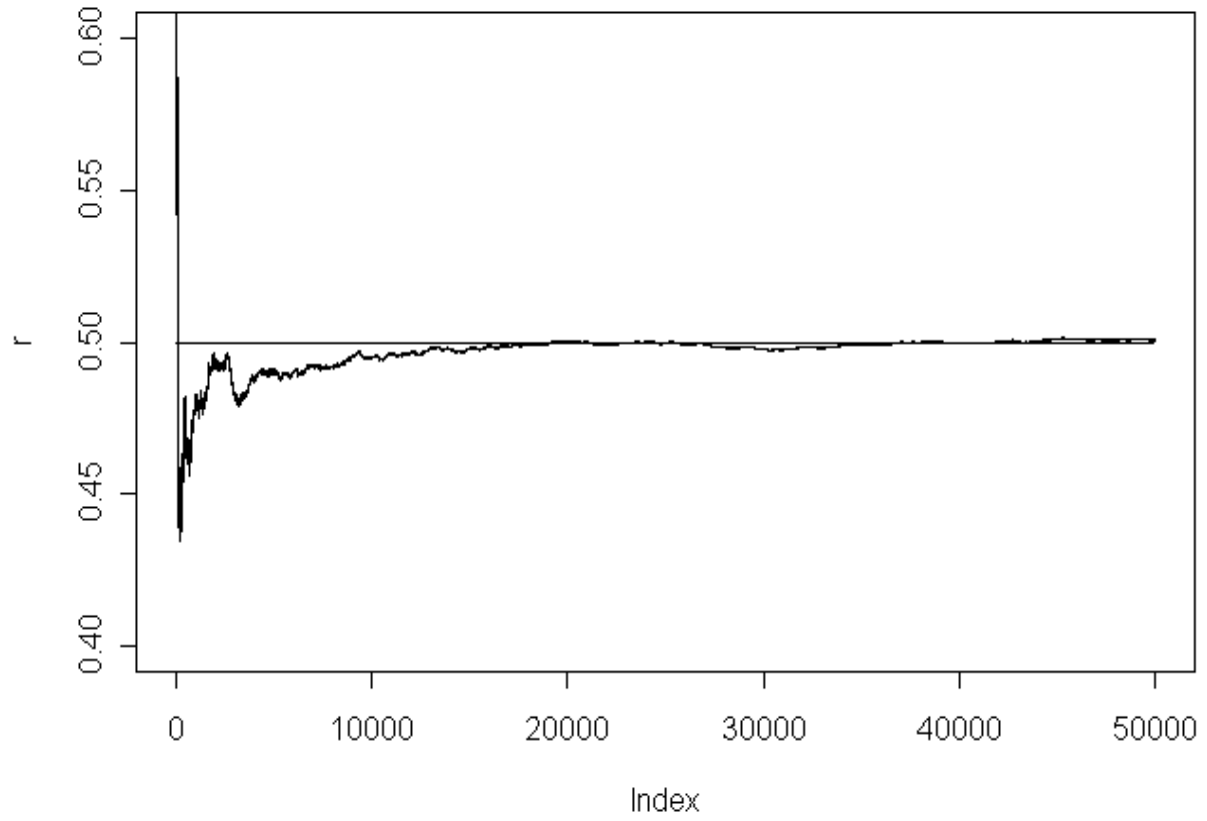
(Same as table above—plus **50,000th value.**)

	x	s	r
[1,]	1	1	1.00000
[2,]	1	2	1.00000
[3,]	1	3	1.00000
[4,]	1	4	1.00000
[5,]	1	5	1.00000
[6,]	0	5	0.83333
[7,]	1	6	0.85714
[8,]	1	7	0.87500
[9,]	0	7	0.77778
[10,]	0	7	0.70000

[1]	0.50068
-----	----------------

GRAPHICAL OUTPUT

Law of Large Numbers for Tosses of a Fair Coin



GENERALIZATION OF CODE

```
set.seed(1212)

n <- 50000

p <- 0.3

x <- sample(0:1, n, repl=T, prob=c(1-p, p))

s <- cumsum(x)

r <- s/(1:n)

lo <- max(c(0, p-0.1))

hi <- min(c(1, p+0.1))

plot(r, ylim=c(lo, hi), type="l")

lines(c(0,n), c(p,p))

r[n]
```

PROGRAMMING NOTE:

```
x <- sample(0:1, n, repl=T, prob=c(1-p, p))
```

Can be replaced either by

```
x <- rbinom(n, 1, p)
```

or by

```
x <- (runif(n) < p)
```

COMMENTS ON SIMULATION

Simulations are random:

- No two runs alike.
- Need to distinguish signal from noise.
- Multiple runs required.

Simulations can solve specific, practical probability modeling problems in which...

- No analytic solution is available;
- Analytic solution is too difficult, time consuming;
- Logic of analytic solution needs to be verified;
- Assumptions of model need to be tested.

Simulations are a basic part of research:

- Discover things to be proved;
- Dispose of ideas that don't lead anywhere.

Why are simulations increasingly used in modern probability?

- Theoreticians have discovered the value of simulations.
- Existence of more convenient and efficient software — R, S-Plus, and others.
(Note: Some of the world's best academic and industrial theoreticians and software experts have contributed to R, which is available free online: www.r-project.org.)
- Faster hardware. Yesterday's "supercomputer" problem is today's undergraduate exercise.

What simulations can't do:

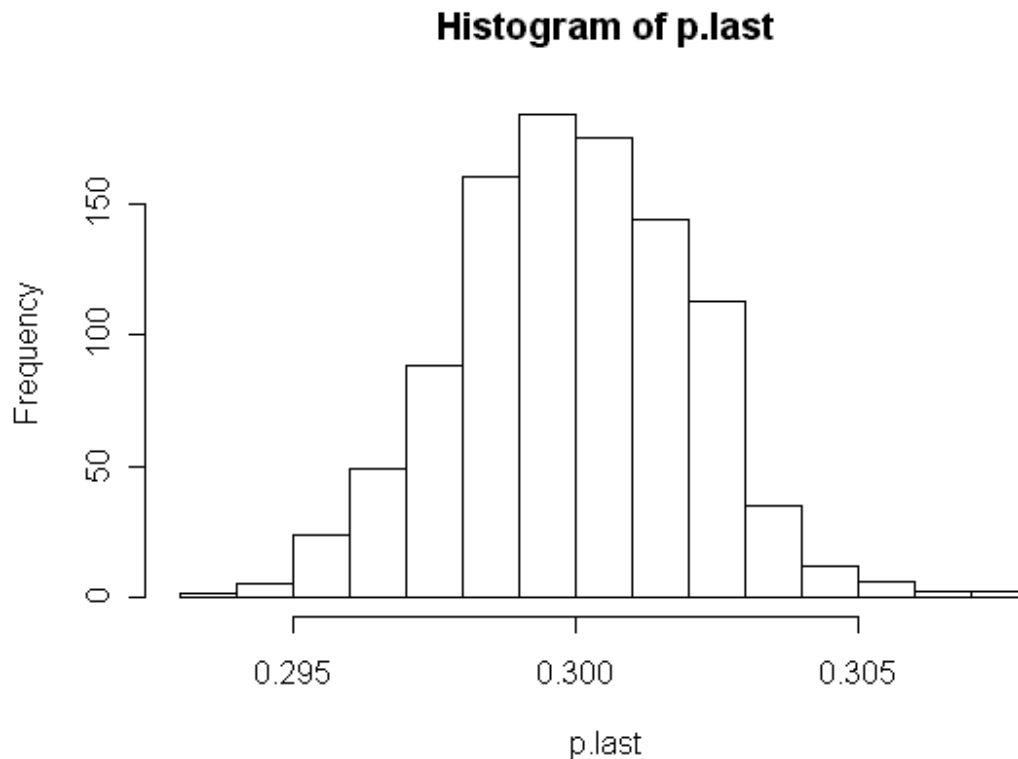
- Provide a general formula,
- Prove (not just suggest) that a result is true.

Bottom line:

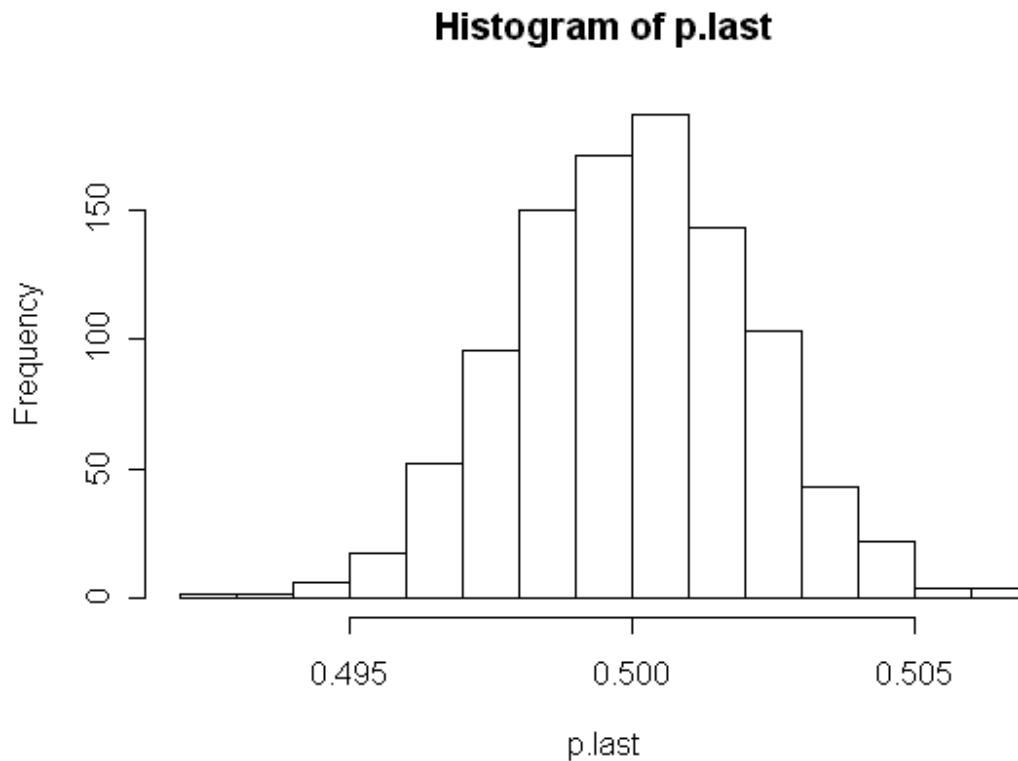
Both theory and simulation are essential in learning and practicing probability modeling.

FOR 50K TOSSES — HOW CLOSE, HOW OFTEN?

```
m <- 1000
p.last <- numeric(m)
n <- 50000
p <- 0.3
for (i in 1:m) {
  x <- (runif(n) < p)
  s <- cumsum(x)
  r <- s/(1:n)
  p.last[i] <- r[n] }
hist(p.last)
```



For $p = 0.5$.



In both cases we see that most of the results after 50 000 tosses are within 0.005 of the correct answer.

Note:

Here we used $mn = 1000(50,000) = 5 \times 10^7$ random numbers.

This simple simulation shows the need for a generator with a large period.

R CODE WITH LABELS AND CONFIDENCE INTERVAL

```
#set.seed(1212)
n <- 50000
p <- .5
x <- sample(0:1, n, prob=c(1-p,p), repl=T)
s <- cumsum(x); r <- s/(1:n)
upr <- min(1, p+.1)
lwr <- max(0, p-.1)
plot(r, ylim=c(lwr, upr), type="l")
lines(c(0,n), c(p,p), col="darkblue", lty=2)
err <- 1.96 * sqrt(p*(1-p)/n)
lines(c(1.01*n,1.01*n), c(p+err,p-err),
      col="darkgreen", lwd=2)
farb <- "darkgreen"
  if (abs(p-r[n]) > err) farb <- "red"
text(n, (lwr+p-err)/2,
paste("r =",round(r[n],3)),
adj=1, col=farb)
title(paste("Heads Ratios up to",n,
  "Tosses With P(H)=",p))
round(cbind(x,s,r), 5)[1:10,]; r[n]
```

Heads Ratios up to 50000 Tosses With $P(H) = 0.5$

