Introduction to Simulation for Biologists: Probability Distributions

Jun Ishigohoka

2024-06-10

Contents

Bernoulli trials and Binomial distribution

Throw a die once. The outcome is success if the side is 1, failure otherwise. This type of trial where the outcome is either success or failure with a certain probability is called a **Bernoulli trial**. The distribution of the number of successes x out of n independent Bernoulli trials ($0 \leq x \leq n$) is the **binomial distribution**.

In `R`, we can use the `rbinom` function for binomial sampling. Confusingly, in R, the number of trials is `size` and the number of binomial samplings is `n`.

```
1 ?rbinom
```

```
## The Binomial Distribution
##
## Description:
##
##      Density, distribution function, quantile function and random
##      generation for the binomial distribution with parameters 'size'
##      and 'prob'.
##
##      This is conventionally interpreted as the number of 'successes' in
##      'size' trials.
##
## Usage:
##
##      dbinom(x, size, prob, log = FALSE)
##      pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
##      qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)
##      rbinom(n, size, prob)
##
## Arguments:
##
##     x, q: vector of quantiles.
##
##        p: vector of probabilities.
##
##        n: number of observations. If 'length(n) > 1', the length is
##           taken to be the number required.
##
##     size: number of trials (zero or more).
##
##     prob: probability of success on each trial.
##
## log, log.p: logical; if TRUE, probabilities p are given as log(p).
##
## lower.tail: logical; if TRUE (default), probabilities are P[X <= x],
##           otherwise, P[X > x].
##
## Details:
##
##      The binomial distribution with 'size' = n and 'prob' = p has
##      density
##
##                        p(x) = choose(n, x) p^x (1-p)^(n-x)
##
##      for x = 0, ..., n.  Note that binomial _coefficients_ can be
##      computed by 'choose' in R.
```

```
##
##      If an element of 'x' is not integer, the result of 'dbinom' is
##      zero, with a warning.
##
##      p(x) is computed using Loader's algorithm, see the reference
##      below.
##
##      The quantile is defined as the smallest value x such that F(x) >=
##      p, where F is the distribution function.
##
## Value:
##
##      'dbinom' gives the density, 'pbinom' gives the distribution
##      function, 'qbinom' gives the quantile function and 'rbinom'
##      generates random deviates.
##
##      If 'size' is not an integer, 'NaN' is returned.
##
##      The length of the result is determined by 'n' for 'rbinom', and is
##      the maximum of the lengths of the numerical arguments for the
##      other functions.
##
##      The numerical arguments other than 'n' are recycled to the length
##      of the result.  Only the first elements of the logical arguments
##      are used.
##
## Source:
##
##      For 'dbinom' a saddle-point expansion is used: see
##
##      Catherine Loader (2000). _Fast and Accurate Computation of
##      Binomial Probabilities_; available as
##      <https://www.r-project.org/doc/reports/CLoader-dbinom-2002.pdf>
##
##      'pbinom' uses 'pbeta'.
##
##      'qbinom' uses the Cornish-Fisher Expansion to include a skewness
##      correction to a normal approximation, followed by a search.
##
##      'rbinom' (for 'size < .Machine$integer.max') is based on
##
##      Kachitvichyanukul, V. and Schmeiser, B. W. (1988) Binomial random
##      variate generation.  _Communications of the ACM_, *31*, 216-222.
##
##      For larger values it uses inversion.
##
## See Also:
##
##      Distributions for other standard distributions, including
##      'dnbinom' for the negative binomial, and 'dpois' for the Poisson
##      distribution.
##
## Examples:
##
```

```
##      require(graphics)
##      # Compute P(45 < X < 55) for X Binomial(100,0.5)
##      sum(dbinom(46:54, 100, 0.5))
##
##      ## Using "log = TRUE" for an extended range :
##      n <- 2000
##      k <- seq(0, n, by = 20)
##      plot (k, dbinom(k, n, pi/10, log = TRUE), type = "l", ylab = "log density",
##            main = "dbinom(*, log=TRUE) is better than  log(dbinom(*))")
##      lines(k, log(dbinom(k, n, pi/10)), col = "red", lwd = 2)
##      ## extreme points are omitted since dbinom gives 0.
##      mtext("dbinom(k, log=TRUE)", adj = 0)
##      mtext("extended range", adj = 0, line = -1, font = 4)
##      mtext("log(dbinom(k))", col = "red", adj = 1)
```

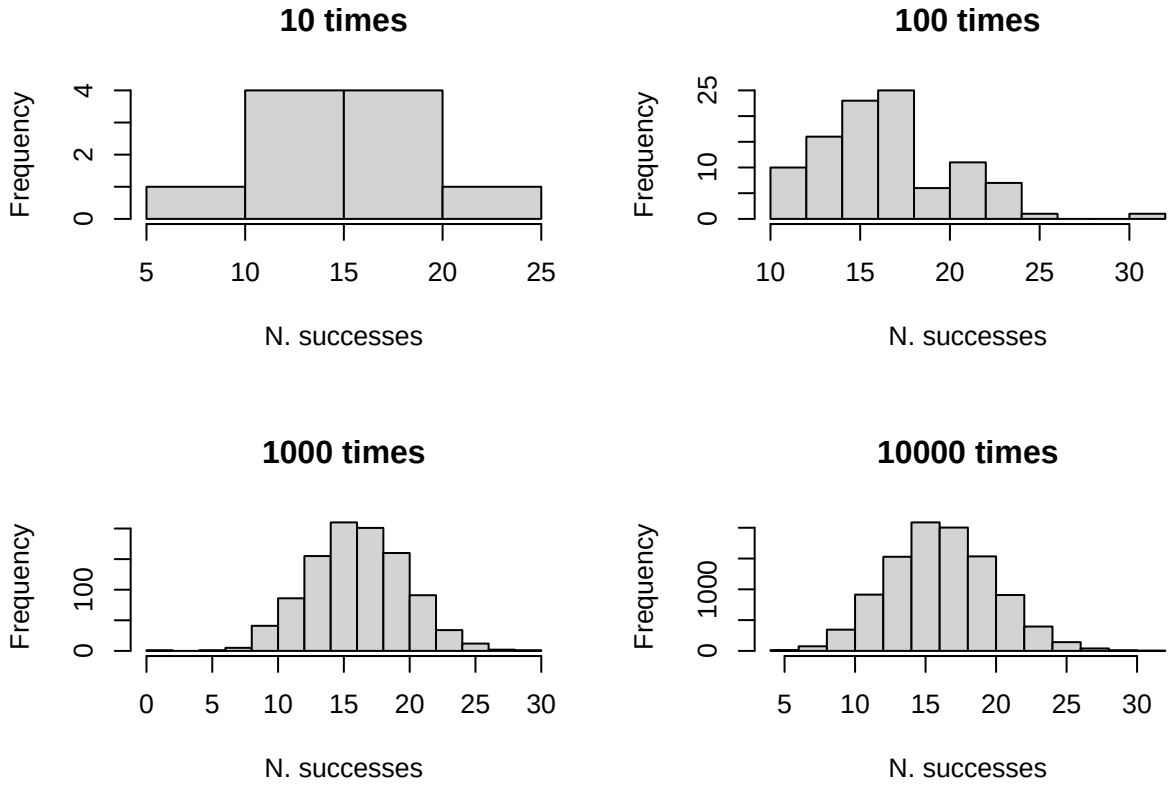
Let's throw a die 100 times and count the number of successes (e.g. the side of 1).

```
rbinom(n = 1, size = 100, 1/6)
```

```
## [1] 17
```

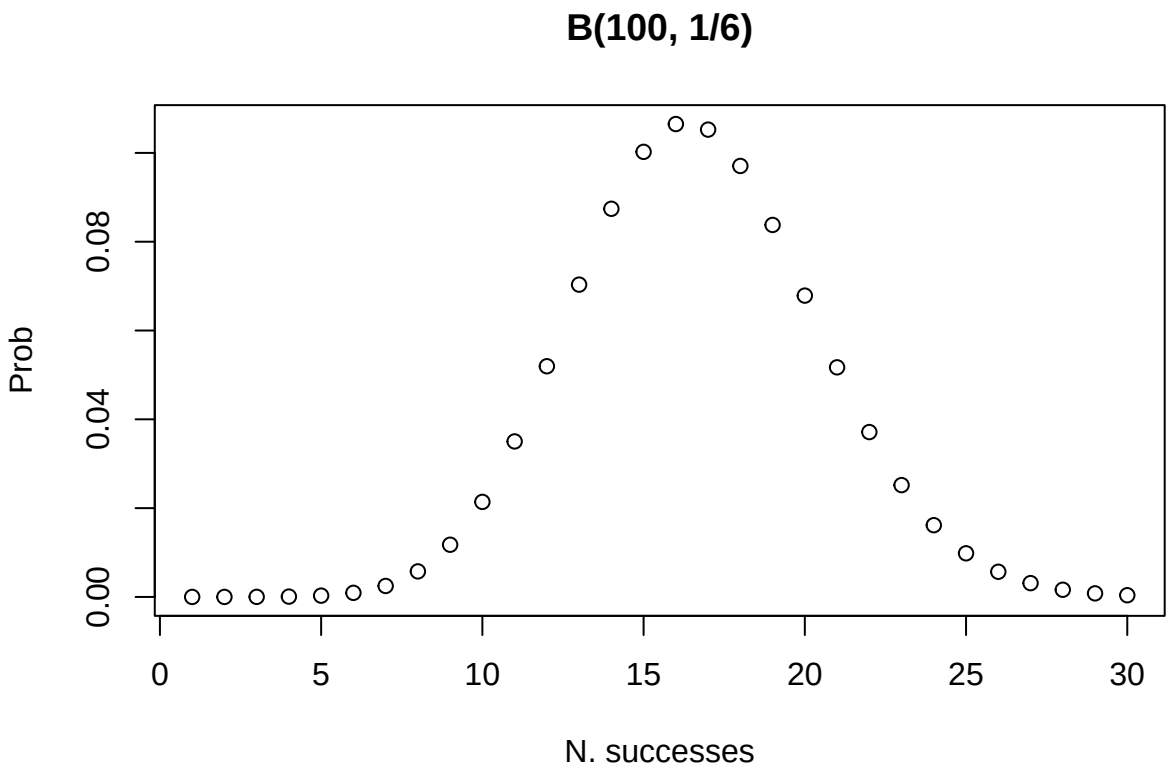
Let's repeat this 10, 100, 1,000, 10,000 times to see the distribution of the number of successes.

```
par(mfrow=c(2,2))
for(n in c(10, 100, 1000, 10000)){
        hist(rbinom(n = n, size = 100, 1/6),
             main = paste0(n, " times"),
             xlab = "N. successes"
        )
}
```



This distribution will approach the probability mass function of the binomial distribution.

```r
x <- 1:30
plot(x, dbinom(x = x, size = 100, prob = 1/6),
     main = "B(100, 1/6)",
     ylab = "Prob",
     xlab = "N. successes"
)
```



Geometric distribution

Let's keep throwing a die until the side is 1. How many times does it take?

Naively, we can loop binomial sampling until it succeeds.

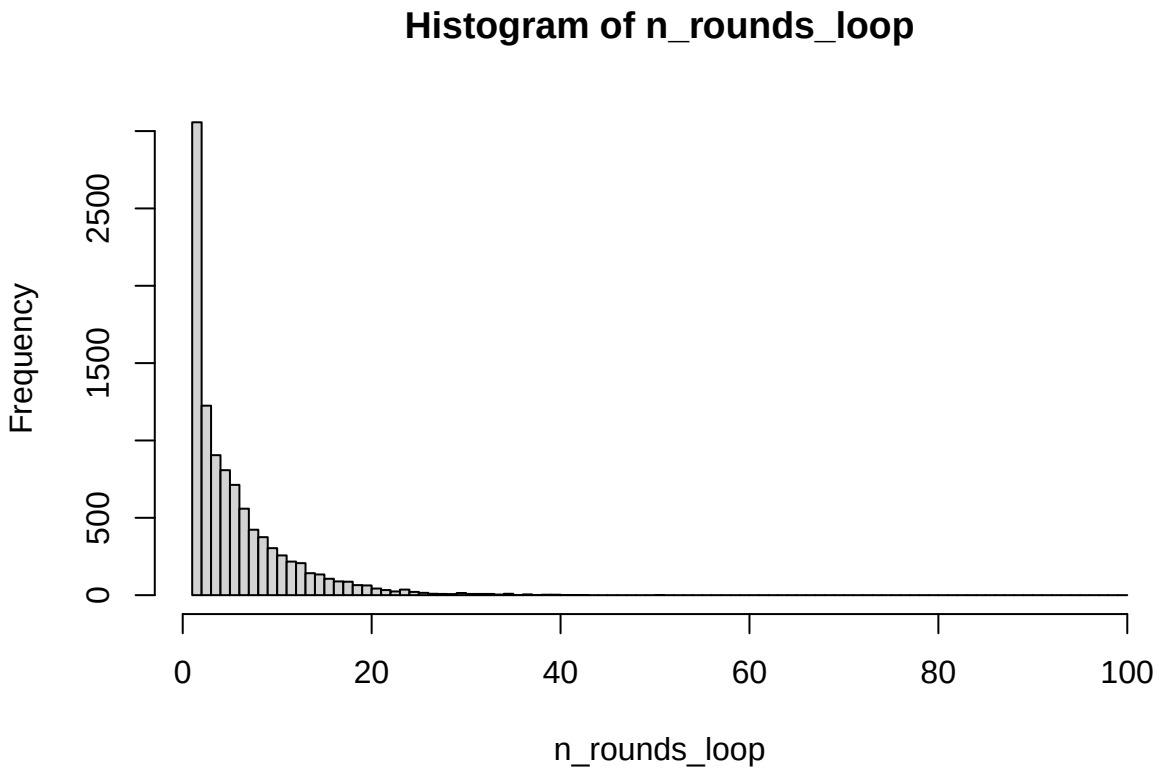
```r
get_nrounds <- function(prob){
        c <- 0
        n <- 0
        while(n == 0){
                c <- c + 1
                n <- rbinom(n = 1, size = 1, prob = prob)
        }
        return(c)
}


get_nrounds(prob = 1/6)
```

```
## [1] 1
```

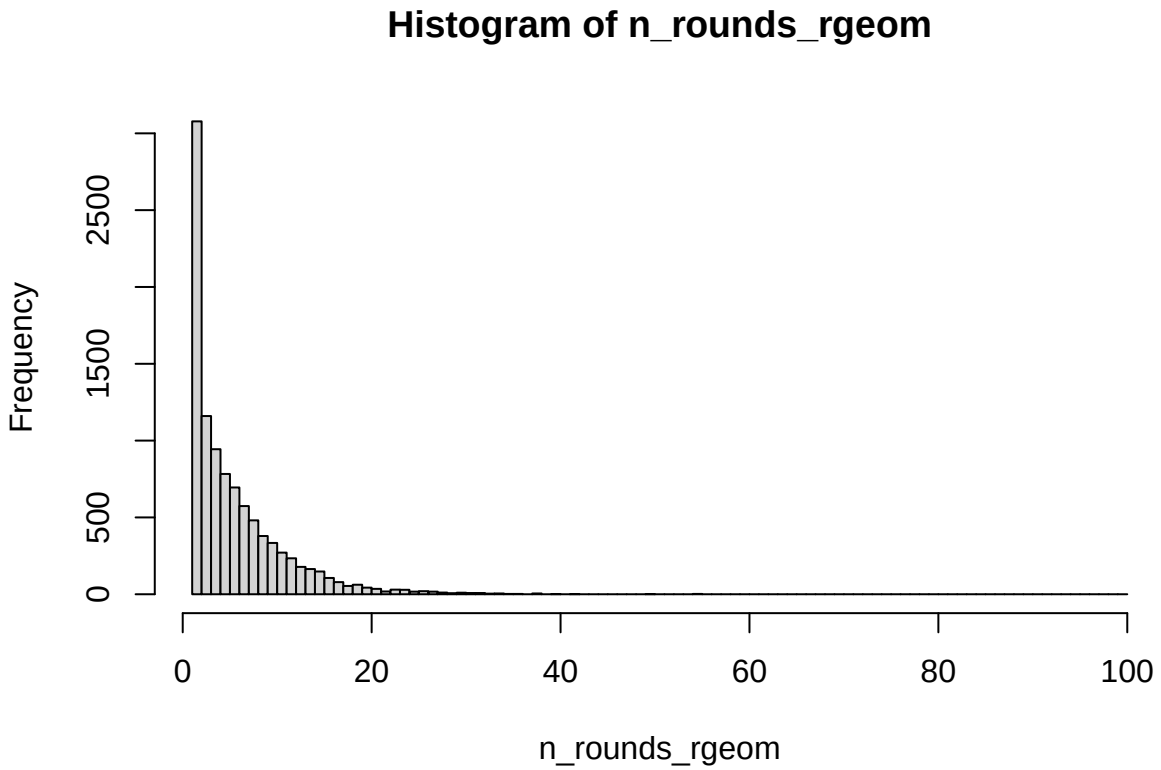
Let's visualise how this waiting time is distributed

```r
n_rounds_loop <- sapply(1:10000,
       function(x){
              get_nrounds(prob = 1/6)
       }
)
hist(n_rounds_loop, breaks = seq(1,100))
```



The number of failures before the first success is known to follow a **geometric distribution**. So, instead of looping binomial/Bernoulli trials, we can directly obtain the number of throws from a geometric distribution using `rgeom` function.

```r
n_rounds_rgeom <- rgeom(n = 10000, prob = 1/6) + 1 # + 1 for the first success
hist(n_rounds_rgeom, breaks = seq(1,100))
```



Using a geometric sampling instead of looping is more efficient especially when the probability of success is very low. Let's measure the time it takes to obtain the number of throws until the first success 100 times using the loop.

```
system.time(
            sapply(1:10,
                   function(x){
                           get_nrounds(prob = 1e-6)
                   }
            )
)
```

```
##    user  system elapsed
##   6.613   0.004   6.622
```

And if we use geometric sampling

```
system.time(
        rgeom(n = 10000, prob = 1/6) + 1 # + 1 for the first success
)
```

```
##    user  system elapsed
##   0.001   0.000   0.001
```

This is a good example that you can still implement your simulation without knowing probability distributions, yet knowing distributions makes your simulation faster and more efficient!

Poisson distribution

Consider taking a 1 mL from a cell suspension at 5,000,000 cells/L (5,000 cell/mL on average). Instead of 1 time of pipetting of 1 mL, let's pipette n times, $1/n$ mL each. When n is very large, one taken suspension ($1/n$ mL) has no cells most of the time, or has 1 cell at the probability of $5000/n$. So the number of cells in

the taken 1 mL suspension can be considered as the limit of binomial sampling of n trials with probability of success of $5000/n$ as n tends to be infinity. This limit of binomial distribution is called **Poisson** distribution, which is used to model the random number of events that happen at a constant rate λ per unit of continuous time or space. In other words, the continuous time/space version of binomial sampling is the Poisson process.

In the above example, the cell concentration corresponds to the rate parameter. Take a 1 mL of cell suspension from 5,000 cells/mL.

```
rpois(n = 1, 5000)
```

```
## [1] 5009
```

Exponential distribution

Continuing on the example of taking cells, let's keep sucking the solution until we get the first cell. The random amount of suspension to take until we get the first cell follows a continuous distribution called **exponential distribution**, which is the limit of geometric distribution where $\lambda = np$ and $n \to \infty$. As is the Poisson distribution, the exponential distribution has one parameter, the rate λ.

Let's sample the amount of suspension at 5,000 cells/mL until you get the first cell.

```
rexp(1, 5000)
```

```
## [1] 0.0001464596
```

According to Wikipedia, the lifetime probability of a fatal lightning strike is 1/60000. So the time it takes for you to die of a lightning hit can be modelled with an exponential distribution. Let's sample one such event.

```
print(paste(rexp(1, 1/60000) * 70, "years"))
```

```
## [1] "499871.886335313 years"
```

Summary

- Binomial distribution B(n, p): discrete number of successes
  - n: number of trials (`size` in `rbinom()`)
  - p: prob. of success
- Geometric distribution: discrete number of failures before the first success
  - p: prob. of success
- Poisson distribution: discrete number of events
  - λ: rate of events per unit of continuous time/space.
- Exponential distribution: continuous waiting time until the first event
  - λ: rate of events per unit of continuous time/space.

Other distributions used in the course

Hypergeometric distribution

The hypergeometric distribution describes the probability of k successes (random draws for which the object drawn has a specified feature) in n draws, **without replacement**, from a finite population of size N that contains exactly K objects with that feature, wherein each draw is either a success or a failure. This is similar to binomial distribution, but it is sampling without replacement.

The hypergeometric distribution is often seen in enrichment analysis. For example, in the context of gene ontology analysis, N is the number of all genes, K is the number of all genes of a focal gene ontology term, n is the number of significant genes, k is the number of significant genes of a focal gene ontology.

Let's sample a number of genes out of 100 significant genes that are of one GO term, when there are 2000 genes of this GO term in 10000 genes throughout the genome.

```
rhyper(1,
       m = 2000, # number of genes of a GO type in the genome
       n = 10000 - 2000, # number of genes that are not the GO type in the genome
       k = 100 # number of significant genes
)
```

```
## [1] 23
```

Let's compute the p-value of observing 50 out of 100 significant genes that are of this GO term.

```
phyper(50,  # number of significant genes of the GO type
       m = 2000, # number of genes of the GO type in the genome
       n = 10000 - 2000, # number of genes that are not the GO type in the genome
       k = 100,  # number of significant genes
       lower.tail = F
)
```

```
## [1] 3.839329e-12
```

Normal distribution

According to the central limit theorem, the average of many observations of some random variable is a random variable, whose distribution converges to a normal distribution as the number of samples increases. Quantities that are expected to be the sum of many independent processes, such as quantitative genetic traits and errors, can be modelled to follow a normal distribution.

Let's sample a normal random variable with mean of 175 and variance of standard deviation of 6.

```
rnorm(1,
      mean = 175,
      sd = 6
)
```

```
## [1] 179.1656
```