Key Distributions in R

Ashley I Naimi

Summer 2024

Contents

1 Distribution Functions in R

Base R comes with a wide variety of pseudo-random number generators that we can use to simulate from a wide variety of probability distributions, including the Gaussian, uniform, binomial, Poisson, and other distributions. Additionally, there are several packages written that can be used to simulate from other common and less common distributions, including the multivariate Normal, double-exponential, Gumbel, and others. Generally, functions written in R to generate data from a distribution follow a particular convention. For example, generating from a normal distribution, we have the following functions in base R:

- rnorm: generate Normal random variable
- dnorm: evaluate the Normal probability density (with a given mean/SD) at specific points
- pnorm: evaluate the cumulative distribution function for a Normal distribution
- qnorm: evaluate the inverse of the cumulative distribution function for a Normal distribution

Generally, functions that start with an "r" generate random variables. Functions that start with a "d" generate density values for a specific value of the random variable. Functions that start with a "p" and a "q" evaluate the cumulative distribution function, and the inverse of the cumulative distribution (quantile) function, respectively.

Prefix	Function
d	density
r	random variable generation
p	cumulative distribution
q	inverse cumulative distribution (quantile)

Mostly, to conduct simulation studies, we'll rely on the "r" functions, which will allow us to generate random variables. Next we'll look at specific functions that generate random variables in R and explore their use for simulation.

1.1 Gaussian (or Normal) Distribution

We can simulate data from a Normal distribution using the `rnorm` function in R. This function can be deployed as:

```
set.seed(123)

n <- 5

y <- rnorm(n, mean = 0, sd = 1)

y
```

```
## [1] -0.56047565 -0.23017749  1.55870831  0.07050839  0.12928774
```

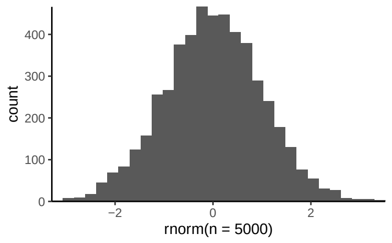

Figure 1: Histogram for Univariate Normal Distribution with Mean = 0 and Standard Deviation = 1 for 5000 Simulated Observations.

The univariate Normal distribution is fully defined by its mean and standard deviation. The default values for these in R are 0 and 1, respectively.

In a regression context, the mean of a Normally distributed random variable is often what is made conditional on other variables. For example, in a simple setting, we might have a Normally distributed outcome Y with a mean conditional on X. This can be accomplished in R in two ways[1]:

[1] Note the use of three `set.seed()` functions in this code chunk. This is not advisable, but I'm doing it here to demonstrate an equivalence between two ways of generating a Y variable conditional on X.

```
n <- 5

set.seed(123)
x <- rnorm(n)

set.seed(123)
y_version1 <- rnorm(n, mean = 1 + 2 * x, sd = 1)

set.seed(123)
y_version2 <- 1 + 2 * x + rnorm(n, mean = 0, sd = 1)

y_version1
```

```
## [1] -0.6814269  0.3094675  5.6761249  1.2115252  1.3878632
```

```
y_version2
```

```
## [1] -0.6814269  0.3094675  5.6761249  1.2115252  1.3878632
```

1.2 Multivariate Normal Distribution

It is sometimes useful to generate data from a multivariate Normal distribution. For example, if you are trying to simulate a large number of independent covariates (e.g., confounders or predictors) that will be included in a regression model, rather than copy and paste code for the univariate Normal distribution multiple times, you can use code for generating a multivariate Normal vector, and make the covariance between them zero.

Alternatively, if you are interested in exploring the impact of non-zero covariance between a set of covariates on the performance of an estimator, you can use multivariate Normal functions to do so.

There are two packages in R that can be used to generate multivariate Normal data: the `MASS` package and the `mvtnorm` package. Here's how to use the functions in `MASS`:

```
n <- 5

set.seed(123)

# create variance - covariance matrix:
sigma <- matrix(0, nrow = 3, ncol = 3)
diag(sigma) <- 1

sigma
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

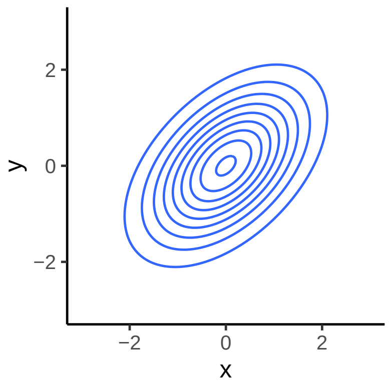

Figure 2: ,
and Standard Deviation = [1,1], and covariance [.5, .5] for 5000 Simulated Observations]Contour Plot for Multivariate Normal Distribution with Mean = [0,0], and Standard Deviation = [1,1], and covariance [.5, .5] for 5000 Simulated Observations.

```r
# create mean vector:
mu <- rep(1, 3)

mu
```

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

```r
# simulate variables
c <- MASS::mvrnorm(n, mu = mu, Sigma = sigma)

c
```

```
##           [,1]       [,2]      [,3]
## [1,] 2.2240818  2.7150650 0.4395244
## [2,] 1.3598138  1.4609162 0.7698225
## [3,] 1.4007715 -0.2650612 2.5587083
## [4,] 1.1106827  0.3131471 1.0705084
## [5,] 0.4441589  0.5543380 1.1292877
```

Here's how to do the same thing in `mvtnorm`:

```r
n <- 5

set.seed(123)

# create variance - covariance matrix:
sigma <- matrix(0, nrow = 3, ncol = 3)
diag(sigma) <- 1

sigma
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

```r
# create mean vector:
mu <- rep(1, 3)

mu
```

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

```r
# simulate variables
c <- mvtnorm::rmvnorm(n, mean = mu, sigma = sigma)

c
```

```
##           [,1]       [,2]      [,3]
## [1,] 0.4395244  0.7698225 2.5587083
## [2,] 1.0705084  1.1292877 2.7150650
## [3,] 1.4609162 -0.2650612 0.3131471
## [4,] 0.5543380  2.2240818 1.3598138
## [5,] 1.4007715  1.1106827 0.4441589
```

Note the flexibility that can be introduced into these function calls. You can specify different means for each column, different standard deviations for each column, as well as different variance-covariance relations between columns, all depending on the nature of the question you are interested in answering.

Technical Note:
Recently in our work, we have been exploring the impact of increasing or decreasing the number of variables in a regression model on the performance of a range of estimators. To do this, we are using the mvtnorm package, with code that looks like this:

```
n = 5

p <- c_number <- 3

## confounder matrix
sigma <- matrix(0,nrow=p,ncol=p)
diag(sigma) <- 1
c <- mvtnorm::rmvnorm(n, mean=rep(0,p), sigma=sigma)

# DESIGN MATRIX FOR THE OUTCOME MODEL
muMatT <- model.matrix(
  as.formula(
    paste("~(",
          paste("c[,",1:ncol(c),"]", collapse="+"),
          ")"
          )
    )
  )[,-1]

parmsC <- rep(1.5,c_number)

y <- 10 + muMatT%*%parmsC + rnorm(n)

data.frame(y,c)
```

```
##           y         X1         X2         X3
## 1 10.903684  1.7869131  0.4978505 -1.9666172
## 2  8.446040  0.7013559 -0.4727914 -1.0678237
## 3  7.935820 -0.2179749 -1.0260044 -0.7288912
## 4  8.667215 -0.6250393 -1.6866933  0.8377870
## 5 11.225158  0.1533731 -1.1381369  1.2538149
```

By changing the c_number value, you can automatically increase or decrease the number of covariates included in the model for Y.

1.3 Uniform Distribution

The uniform distribution is fully defined by its upper and lower bounds. The default values for these in R are 0 and 1, respectively.

This distribution can be useful in a number of ways. In principle, this distribution can be made conditional on other variables by specifying the bounds of the distribution as a function of some other variable. However, this is not often seen in practice. One can generate a uniform random variable in R using the following code:

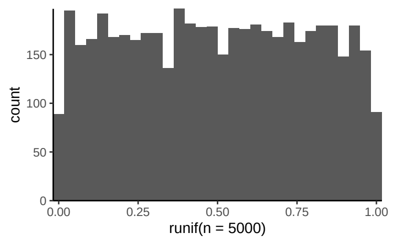

Figure 3: Histogram for the Uniform Distribution with Upper and Lower Bounds of 0 and 1, respectively for 5000 Simulated Observations.

```r
set.seed(123)

n <- 5

y <- runif(n, min = 0, max = 1)

y
```

```
## [1] 0.2875775 0.7883051 0.4089769 0.8830174 0.9404673
```

This distribution is central to the inverse transformation method, which is a generic technique used to simulate data from arbitrary distributions. We will see this method shortly.

1.4 Binomial Distribution

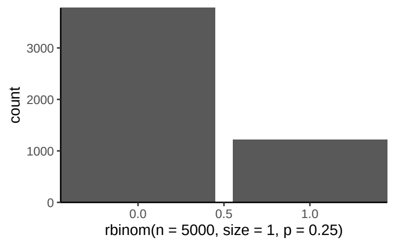

Figure 4: Barplot for the Binomial (Bernoulli) Distribution with p = 0.25 for 5000 Simulated Observations.

The binomial distribution is defined by two parameters: the probability of a success in a given "trial" and the number of trial conducted. When the number of "trials" is one, the binomial distribution is equivalent to the Bernoulli distribution, which generates a $[0, 1]$ indicator of whether an event occurred or not. In settings where logistic regression is used, the dependent variable is usually from a Bernoulli distribution with a probability depending on variables that will be included in the model. There are no default values for probability of success and the trial size, so these must be specified as follows.

```
set.seed(123)

n <- 5

y <- rbinom(n, size = 1, p = 0.5)

y
```

```
## [1] 0 1 0 1 1
```

Note what happens when we increase the size of the trial to, say, 8:

```
set.seed(123)

n <- 5

y <- rbinom(n, size = 8, p = 0.5)

y
```

```
## [1] 3 5 4 6 6
```

Each instance of the simulated y becomes a sum of all of the successes (one's) encountered over the eight trials.

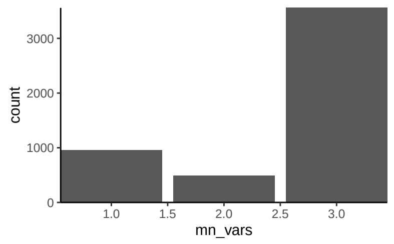

Figure 5: Barplot for the Multinomial Distribution with Three Levels and p = 0.2, 0.1, 0.7 for 5000 Simulated Observations.

1.5 Multinomial Distribution

The multinomial distribution is a generalization of the binomial distribution that can be used to generate categorical variables where the probability of each level is governed by a (possibly) unique probability value. The multinomial distribution is defined by two parameters: the set of values defining the probabilities of realizing a specific category. and the number of trial conducted. Similar to the binomial distribution, when the number of "trials" is one, the multinomial distribution is equivalent to the categorical distribution, which generates a $[0, 1]$ indicator of whether an event occurred or not. In the multinomial (categorical) case, the "event" can take on more than 2 levels. There are no default values for probabilities of success and the trial size, so these must be specified.

We can start by using the multinomial distribution in R to model the physical mechanism by which we roll a single (size = 1), six sided die. The "six-sidedness" comes from the fact we are repeating the "1/6" value six times (using the `rep()` function). Here, we roll a six-sided dice 5 times:

```r
set.seed(123)

n <- 5

y <- rmultinom(n, size = 1, p = rep(1/6, 6))

y
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    1    0    0    0
## [2,]    0    0    0    0    0
## [3,]    0    0    1    1    0
## [4,]    1    0    0    0    0
## [5,]    0    0    0    0    0
## [6,]    0    0    0    0    1
```

Alternatively, we can imagine a situation where an individual can experience one of three outcomes in a study related to cardiovascular health. Suppose the individual can be censored (y = 1), can experience a competing event such as death (y = 2), or can experience the outcome of interest, such as a heart attack (y = 3). Suppose further that these probabilities are 0.1, 0.05, and 0.15, respectively. This means that thirty percent of the sample experienced some event, and the remaining seventy percent experienced no event (such individuals would often be identified as administratively censored). We can simulate these data with the following code:

```r
set.seed(123)

n <- 5
```

```
y <- rmultinom(n, size = 1,
               p = c(0.1,0.05,0.15, 1 - sum(0.1,0.05,0.15)))

y
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    0
## [2,]    0    0    0    1    0
## [3,]    0    0    0    0    0
## [4,]    1    1    1    0    1
```

```
t(y)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    0    0    0    1
## [2,]    0    0    0    1
## [3,]    0    0    0    1
## [4,]    0    1    0    0
## [5,]    0    0    0    1
```

```
apply(t(y), 2, mean)
```

```
## [1] 0.0 0.2 0.0 0.8
```

Note here that in this simulation, the first two observations experienced a competing event, the third and the fifth observations were censored, and the fourth observation experienced a heart attack.

1.6 Poisson Distribution

The Poisson distribution is fully defined by a single parameter, usually denoted λ. This distribution is usually used to model counts of discrete events, such as the number of cigarettes smoked in one hour among smokers. A Poisson random variable can be simulated using the following code:

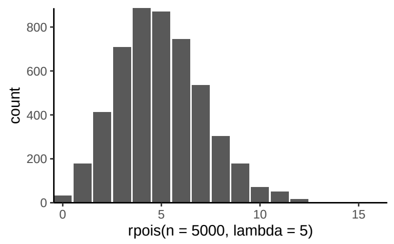

Figure 6: Histogram for the Poisson Distribution with lamba = 5 for 5000 Simulated Observations.

```r
set.seed(123)

n <- 5

y <- rpois(n, lambda = 3)

y
```

```
## [1] 2 4 2 5 6
```