Manipulation of Discrete Random Variables with discreteRV

by Eric Hare, Andreas Buja, and Heike Hofmann

Abstract A major issue in statistics education is the sometimes large disparity between the mathematical and theoretical coursework, and the computational coursework. **discreteRV** is an R package for manipulation of discrete random variables which uses clean and familiar syntax similar to that which is found in introductory probability courses. It is simple enough for those less experienced with statistical programming, but has more advanced features which are suitable for a large number of more complex applications. In this paper, I introduce and motivate **discreteRV**, describe its functionality, and provide reproducible examples illustrating its use.

Introduction

One of the primary hurdles in teaching probability courses in an undergraduate setting is the missing link between theoretical statements from textbooks and lectures, and the notation used in statistical software required in more and more classes. Depending on the background of the student, this missing link can manifest itself in a couple of different ways. Some students will master the theoretical concepts and notation, but struggle in a computing environment. Others will feel very comfortable with statistical programming, but sometimes struggle to translate that back to the classroom probability setting.

discreteRV is an attempt to bridge this gap. It provides a comprehensive set of functions to create, manipulate, and simulate from discrete random variables. **discreteRV** is designed for introductory probability courses. It uses syntax which in most cases closely matches probability textbooks to allow for a more seamless connection between a probability classroom setting and the use of statistical software. **discreteRV** is available for download on the Comprehensive R Archive Network (CRAN).

The functions of **discreteRV** are organized into two logical areas, termed probabilities and simulations.

Probabilities

discreteRV includes a suite of functions to create, manipulate, and compute distributional quantities for discrete random variables. A list of these functions and brief discriptions of their functionality is available in Table 1.

Creating random variables

The centerpiece of **discreteRV** is a set of functions to create and manipulate discrete random variables. A random variable X is defined (see e.g. Wild and Seber, 1999) as a theoretical construct representing the value of an outcome of a random experiment. A discrete random variable is a special case that can only take on a countable set of values. Here, we will make the additional constraint to only allow a finite number of values. In practice, this constraint has minimal impact. Discrete random variables are associated with probability mass functions, which map the set of possible outcomes of the random experiment to probabilities. Probability mass functions must therefore define probabilities which are between zero and one, and must sum to one.

Throughout this document, we will work with two random variables, a simple example of a discrete random variable representing the value of a roll of a fair die, and one representing a realization of a Poisson random variable with mean parameter equal to two. Formally, we can define such random variables and their probability mass functions as follows:

Let X be a random variable representing a single roll of a fair die; i.e., the sample space $\Omega = \{1, 2, 3, 4, 5, 6\}$ and X is the identity, mapping the upturned face of a die roll to the corresponding number of dots visible. Then,

$$f(x) = P(X = x) = \begin{cases} \frac{1}{6} & x \in 1, 2, 3, 4, 5, 6 \\ 0 & \text{otherwise} \end{cases}$$

Name	Description
Creation	
`as.RV`	Turn a probability vector with possible outcome values in the names() attribute into a random variable
`make.RV`	Make a random variable consisting of possible outcome values and their probabilities or odds
Manipulation	
`margins`	Marginal distribution of a joint random variable
`mult`	Joint probability mass function of random variables X and Y
`multN`	Probability mass function of X^n
`SofI`	Sum of independent random variables
`SofIID`	Sum of independent identically distributed random variables
Probabilities	
`E`	Expected value of a random variable
`KURT`	Kurtosis of a random variable
`P`	Calculate probabilities of events
`probs`	Probability mass function of random variable X
`SD`	Standard deviation of a random variable
`SKEW`	Skewness of a random variable
`V`	Variance of a random variable
Methods	
`plot.RV`	Plot a random variable of class RV
`print.RV`	Print a random variable of class RV
`qqnorm.RV`	Normal quantile plot for RVs to answer the question how close to normal it is

Table 1: Overview of functions provided in **discreteRV** ordered by topics.

Let Y be a random variable distributed according to a Poisson distribution with mean parameter λ. In this case, the sample space $\Omega = \{0, 1, 2, ...\}$ (the non-negative integers). Then,

$$f(y) = P(Y = y) = \begin{cases} \frac{\lambda^y e^{-\lambda}}{y!} & y \in 0, 1, 2, ... \\ 0 & \text{otherwise} \end{cases}$$

In **discreteRV**, a discrete random variable is defined through the use of the `make.RV` function. `make.RV` accepts a vector of outcomes, a vector of probabilities, and returns an RV object. The code to create X, a random variable representing the roll of a fair die, is as follows:

```
X <- make.RV(outcomes = 1:6, probs = 1/6)
X

## Random variable with 6 outcomes
##
## Outcomes    1    2    3    4    5    6
## Probs     1/6  1/6  1/6  1/6  1/6  1/6
```

Outcomes can be specified as a range of values, which is useful for distributions in which the outcomes that can occur with non-zero probability are unbounded. This can be indicated with the range parameter, which defaults to TRUE in the event that the range of values includes positive or negative infinity. To define our poisson random variable Y, we specify the outcomes as a range and the probabilities as a function:

```
pois.func <- function(y, lambda = 2) { return(lambda^y * exp(-lambda) / factorial(y)) }

Y <- make.RV(outcomes = c(0, Inf), probs = pois.func)
Y

## Random variable with 15 outcomes
##
## Outcomes    0    1    2    3    4    5    6    7    8    9    10    11
```

discreteRV	Casella and Berger
E(X)	$E(X)$
P(X == x)	$P(X = x)$
P(X >= x)	$P(X \geq x)$
P((X < x1) %AND% (X > x2))	$P(X < x_1 \cap X > x_2)$
P((X < x1) %OR% (X > x2))	$P(X < x_1 \cup X > x_2)$
P((X == x1) | (X > x2))	$P(X < x_1 \| X > x_2)$
probs(X)	$f(x)$
SD(X)	$sd(X)$
V(X)	$var(X)$

Table 2: Probability functions in **discreteRV** and their corresponding syntax in introductory statistics courses.

```
## Probs    0.135 0.271 0.271 0.180 0.090 0.036 0.012 0.003 0.001 0.000 0.000 0.000
##
## Displaying first 12 outcomes
```

The make.RV function also allows the definition of a random variable in terms of odds. One can construct a loaded die in which a roll of one is four times as likely as any other roll as:

```
X.loaded <- make.RV(outcomes = 1:6, odds = c(4, 1, 1, 1, 1, 1))
X.loaded

## Random variable with 6 outcomes
##
## Outcomes   1   2   3   4   5   6
## Odds      4:5 1:8 1:8 1:8 1:8 1:8
```

Structure of an RV Object

The syntactic structure of the included functions lends itself both to a natural presentation of elementary probabilities and properties of probability mass functions in an introductory probability course, as well as more advanced modeling of discrete random variables. In Table 2, we provide an overview of the notational similarities between **discreteRV** and the commonly used probability textbook by Casella and Berger (2001)

A random variable object is constructed by defining a standard R vector to be the possible outcomes that the random variable can take (the sample space Ω). It is preferred, though not required, that these be encoded as integers, since this allows the computation expected values, variances, and other distributional properties. This vector of outcomes then stores attributes which include the probability of each outcome. By default, the print method for a random variable will display the probabilities as fractions in most cases, aiding in readability. The probabilities can be retrieved as a numeric vector by using the probs function:

```
probs(X)

##      1      2      3      4      5      6
## 0.1667 0.1667 0.1667 0.1667 0.1667 0.1667
```

Probability-based Calculations

By storing the outcomes as the principle component of the object X, we can now make a number of probability statements in R. For instance, we can ask what the probability of obtaining a roll greater than 1 is by using the code P(X >1). R will check which values in the vector X are greater than 1. In this case, these are the outcomes 2, 3, 4, 5, and 6. Hence, R will return TRUE for these elements of X, and then we can encode a function P to compute the probability of this occurrence by simply summing over the probability values stored in the names of these particular outcomes. Likewise, we can make slightly more complicated probability statements such as $P(X > 5 \cup X = 1)$, using the %OR% and %AND% operators. Consider our Poisson random variable Y, and suppose we want to obtain the probability that Y is with a distance δ of its mean parameter $\lambda = 2$:

```
delta <- 3
lambda <- 2

P((Y >= lambda - delta) %AND% (Y <= lambda + delta))

## [1] 0.9834
```

In addition, conditional probabilities are also supported. For instance, to calculate the probability of obtaining a roll of one given that the roll is no more than 3, you would use the code `P(X == 1 | X <= 3)`. The use of the pipe operator may be less intuitive to the seasoned R programmer, but overcomes a major notational issue in that conditional probabilities are most commonly specified with the pipe. This issue led to the creation of the `%OR%` and `%AND%` operators specified previously.

Several other distributional quantities are computable, including the expected value and the variance of a random variable. As in notation from probability courses, expected values can be found with the `E` function. To compute the expected value for a single roll of a fair die, we run the code `E(X)`. The expected value for a poisson random variable is its mean, and hence `E(Y)` will return two.

Joint Distributions

Aside from moments and probability statements, **discreteRV** includes a powerful set of functions used to create joint probability distributions. Once again letting X be a random variable representing a single die roll, we can use the `multN` function to compute the probability mass function of n trials of X. Table 3 gives the first eight outcomes for $n = 2$, and Table 4 gives an the first eight outcomes for $n = 3$. Notice again that the probabilities have been coerced into fractions for readability. Notice also that the outcomes are encoded by the outcomes on each trial separated by a period.

Outcome	1,1	1,2	1,3	1,4	1,5	1,6	2,1	2,2
Probability	1/36	1/36	1/36	1/36	1/36	1/36	1/36	1/36

Table 3: First eight Outcomes and their associated Probabilities for a variable representing two independent rolls of a die.

Outcome	1,1,1	1,1,2	1,1,3	1,1,4	1,1,5	1,1,6	1,2,1	1,2,2
Probability	1/216	1/216	1/216	1/216	1/216	1/216	1/216	1/216

Table 4: First eight Outcomes and their associated Probabilities for a variable representing three independent rolls of a die.

discreteRV also includes functions to compute the sum of independent random variables. If the variables are identically distributed, the `SofIID` function can be used to compute probabilities for the sum of n independent realizations of the random variable. In our fair die example, `SofIID(X,2)` would create a random variable object with the representation given in Table 5

| Outcome	2	3	4	5	6	7	8	9	10	11	12
Probability	1/36	1/18	1/12	1/9	5/36	1/6	5/36	1/9	1/12	1/18	1/36

Table 5: Outcomes and their associated Probabilities for a variable representing the sum of two independent rolls of a die.
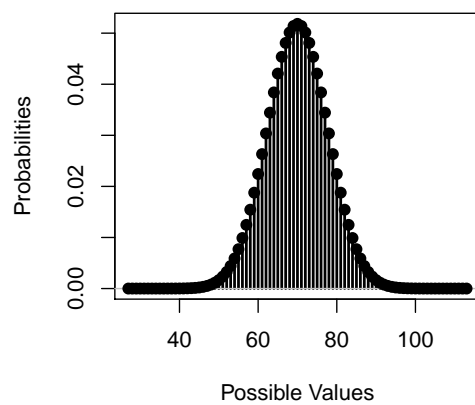
Plotting

discreteRV includes a `plot` method for random variable objects so that a visualization of the outcomes and probabilities can be made simply by calling `plot(X)`. The result of plotting the random variable representing a fair die is given in Figure 1. The x axis includes all outcomes, and the y axis includes the probabilities of each particular outcome. The result of plotting a random variable representing the sum of two independent rolls of a die is given in Figure 2. The result of plotting a random variable representing the sum of 20 independent rolls of a die is given in Figure 3.
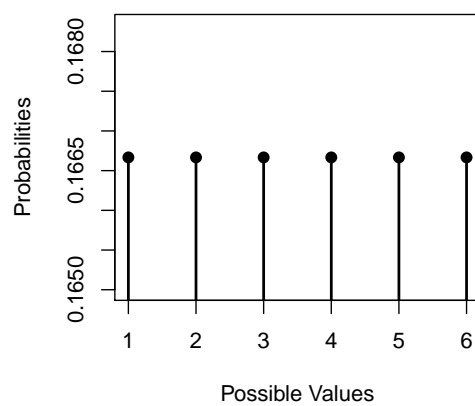
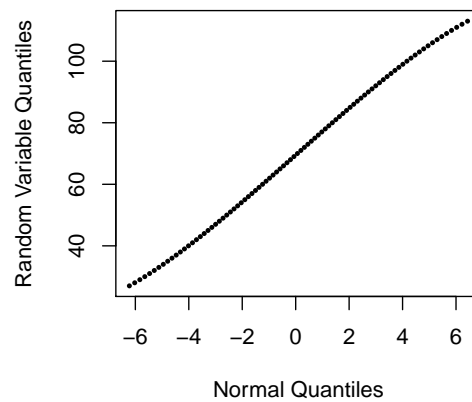**Figure 1:** Plot method called on a fair die random variable.
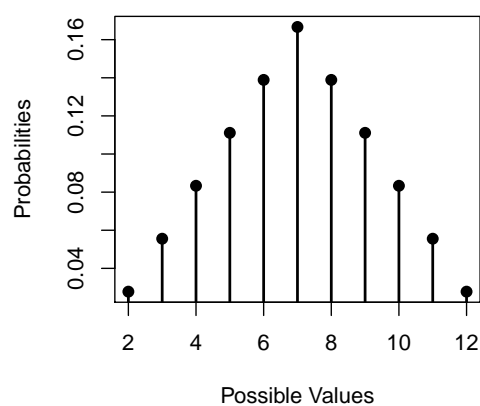
Figure 2: Plot method called on a sum of two fair die random variables.

Figure 3: Plot method called on a sum of 20 fair die random variables.

In addition to a plotting method, there is also a method for qqnorm to allow assessment of normality for random variable objects, as displayed in Figure 4.

Figure 4: qqnorm method called on a sum of 20 fair die random variables.

Simulation

discreteRV also includes a set of functions to simulate trials from a random variable. A list of these functions and brief discriptions of their functionality is available in Table 6.

Name	Description
plot.RVsim	Plot a simulated random vector
Prop	Proportion of an event observed in a vector of simulated trials
props	Proportions of observed outcomes in one or more vectors of simulated trials
rsim	Simulate n independent trials from a random variable X
skewSim	Skew of the empirical distribution of simulated data

Table 6: List of the simulation functions contained in **discreteRV**.

Creation

Creating a simulated random vector is done by using the rsim function. rsim accepts a parameter n representing the number of independent trials to simulate, and a parameter X representing the random variable with which to simulate from. For example, suppose we'd like to simulate ten trials from a fair die. We have already created a random variable object X, so we simply call rsim as follows:

```
X.sim <- rsim(10, X)
X.sim

##  [1] 2 2 2 5 6 5 5 6 5 1
## attr(,"RV")
## Random variable with 6 outcomes
##
## Outcomes    1    2    3    4    5    6
## Probs     1/6 1/6 1/6 1/6 1/6 1/6
## attr(,"class")
## [1] "RVsim"
```

The object returned is a vector of simulated values, with a class attribute to contain the random variable that was used for the simulation. If we would like to retrieve only the simulated values and

exclude the attached probabilities, we can coerce the object into a vector using R's built-in `as.vector` function.

```
as.vector(X.sim)
```

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

It is also possible to retrieve some quantities from the simulation. We can retrieve the empirical distribution of simulated values with the `props` function. This will return the outcomes from the original random variable object, and the observed proportion of simulated values for each of the outcomes. We can also compute observed proportions of events by using the `Prop` function. Similar to the `P` function for probability computations on random variable objects, `Prop` accepts a variety of logical statements.

```
props(X.sim)
```

```
## RV
##   1   2   3   4   5   6
## 0.1 0.3 0.0 0.0 0.4 0.2
```

```
Prop(X.sim == 3)
```

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

```
Prop(X.sim > 3)
```

```
## [1] 0.6
```

Case Study: Playing Craps

Craps is a common card game played in casinos. The game begins with what is called the "Come Out" roll, in which two fair dice are rolled. If a sum of seven or eleven is obtained, the player wins. If a sum of two, three, or twelve is obtained, the player loses. In all other cases, the roll obtained is declared the "Point" and the player rolls again in an attempt to obtain this same point value. If the player rolls the Point, they win, but if they roll a seven, they lose. Rolls continue until one of these two outcomes is achieved.

discreteRV allows for a seamless analysis and simulation of the probabilities associated with different events in Craps. Let us begin by asking "What is the probability that the game ends after the first roll?" Let us construct random variables to answer this question. We construct a random variable X representing the single roll of a fair die, and then we use the `SofIID` function to sum over two rolls.

```
X <- make.RV(outcomes = 1:6, probs = 1/6)
roll <- SofIID(X, n = 2)
```

```
roll
```

```
## Random variable with 11 outcomes
##
## Outcomes     2    3    4    5    6    7    8    9   10   11   12
## Probs     1/36 1/18 1/12  1/9 5/36  1/6 5/36  1/9 1/12 1/18 1/36
```

Recall that the game ends after the first roll if and only if a seven or eleven is obtained (resulting in a win), or a two, three, or twelve is obtained (resulting in a loss). Hence, we calculate the probability that the game ends after the first roll as follows:

```
win_rolls <- c(7, 11)
loss_rolls <- c(2, 3, 12)

end_rolls <- c(win_rolls, loss_rolls)

P(roll %in% end_rolls)
```

```
## [1] 0.3333
```

Now suppose we would like to condition on the game having ended after the first roll. Using the conditional probability operator in **discreteRV**, we can obtain the probabilities of winning and losing given that the game ended after the first roll:

```
P(roll %in% win_rolls | roll %in% end_rolls)

## [1] 0.6667

P(roll %in% loss_rolls | roll %in% end_rolls)

## [1] 0.3333
```

Now, let's turn our attention to calculating the probability of winning a game in two rolls. Recall that we can use the `multN` function to generate joint distributions of independent and identically distributed random variables. In this case, we would like to generate the joint distribution for two independent rolls of two dice. Now, we will have possible outcomes (11^2), and our job is to determine which outcomes result in a win. We know that any time the first roll is a seven or eleven, we will have won. We also know that if the roll is between four and ten inclusive, then we will get to roll again. To win within two rolls given that we've received a four through ten requires that the second roll match the first. We can enumerate the various possibilities to calculate the probability of winning in two rolls, which is approximately 30%.

```
two_rolls <- multN(roll, 2)

winning_outcomes <- c(paste(7, 2:12, sep = ","), # Seven on the first roll
                      paste(11, 2:12, sep = ","), # Eleven on the first roll
                      paste(4:10, 4:10, sep = ",")) # Pair between four and ten
winning_outcomes

##  [1] "7,2"   "7,3"   "7,4"   "7,5"   "7,6"   "7,7"   "7,8"   "7,9"
##  [9] "7,10"  "7,11"  "7,12"  "11,2"  "11,3"  "11,4"  "11,5"  "11,6"
## [17] "11,7"  "11,8"  "11,9"  "11,10" "11,11" "11,12" "4,4"   "5,5"
## [25] "6,6"   "7,7"   "8,8"   "9,9"   "10,10"

P(two_rolls %in% winning_outcomes)

## [1] 0.2994
```

Finally, suppose we are interested in the empirical probability of winning a game of Craps. Using the simulation functions in **discreteRV**, we can write a routine in order to test this. Using the `rsim` function, we simulate a single game of Craps by rolling from our random variable representing the sum of two dice. We simulate 100 games, and then perform this simulation 1000 times. The results indicate that the player wins a game of craps approximately 49% of the time.

```
craps_game <- function(RV) {
    my.roll <- rsim(1, RV)
    if (my.roll %in% c(7, 11)) {
        return(1)
    } else if (my.roll %in% c(2, 3, 12)) {
        return(0)
    } else {
        new.roll <- 0
        while (new.roll != my.roll & new.roll != 7) {
            new.roll <- rsim(1, RV)
        }

        return(as.numeric(new.roll == my.roll))
    }
}
```

```
sim.results <- replicate(5000, sum(replicate(1000, craps_game(roll))))
(emp.prob <- mean(sim.results / 1000))

## [1] 0.4931
```

Conclusion

The power of **discreteRV** is truly in its simplicity. Because it uses familiar introductory probability syntax, it can allow students who may not be experienced or comfortable with programming to ease into computer-based computations. Nonetheless, **discreteRV** also includes several powerful functions for analyzing, summing, and combining discrete random variables which can be of use to the experienced programmer.

Bibliography

A. Buja. *discreteRV: Functions to create and manipulate discrete random variables*, 2013. R package version 1.0.2. [p]

G. Casella and R. L. Berger. *Statistical Inference*, volume 2. Cengage Learning, 2001. [p3]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. URL http://www.R-project.org/. [p]

C. Wild and G. Seber. *CHANCE ENCOUNTERS: A First Course in Data Analysis and Inference*. John Wiley & Sons, 1999. [p1]