Manipulation of Discrete Random Variables with discreteRV

Eric Riemer Hare

January 6, 2014

1 Introduction

Dr. Andreas Buja, professor of Statistics at the University of Pennsylvania, created a set of R functions implementing discrete random variables[REF]. These functions were compiled and documented in a package called discreteRV. discreteRV is available for download on the Comprehensive R Archive Network (CRAN)[REF].

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

2 Probabilities

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

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

Table 1: List of the probability functions contained in discreteRV.

2.1 Creation

The centerpiece of discreteRV is a set of functions to create and manipulate discrete random variables. A random variable maps a set of possible outcomes to a set of probabilities that sum to one. In discreteRV, a random variable is defined through the use of the make.RV function. make.RV accepts a vector of probabilities and a vector of outcome values, and returns an RV object.

```
make.RV(vals = 1:6, probs = rep("1/6", times = 6))


## random variable with 6 outcomes
##
##   1   2   3   4   5   6
## 1/6 1/6 1/6 1/6 1/6 1/6
```

2.2 Structure

The syntactic structure of the included functions lends itself both to a natural presentation in an introductory probability course, as well as more advanced modeling of discrete random variables. The object is constructed by setting a standard R vector object to the possible values that the random variable can take (the sample space). It is preferred, though not required, that these be encoded as integers, since this allows for expected values, variances, and other measures of distributional tendency to be computed. This vector of outcomes is then named by the respective probability of each outcome. The probability can be encoded as a string, such as "1/6", if this aids in readability, but the string must be coercable to a numeric.

The choice to encode the probabilities in the names of the vector may seem counterintuitive. However, it is this choice which allows for the familiar syntax employed by introductory statistics courses and textbooks to be seamlessly replicated. Consider, for instance, an RV object "X" which we will use to represent a single roll of a fair die.

```
X <- make.RV(1:6, rep("1/6", 6))
X
```

```
## random variable with 6 outcomes
##
##   1   2   3   4   5   6
## 1/6 1/6 1/6 1/6 1/6 1/6
```

Note that although the print method does not illustrate the inherent structure of the object, the probabilities (1/6, for each of the 6 outcomes of the die roll) are actually stored in the names of the object "X".

```
names(X)
```

```
## [1] "1/6" "1/6" "1/6" "1/6" "1/6" "1/6"
```

2.3 Probabilities

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 — X == 1)*.

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)*.

2.4 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 2 gives the first eight outcomes for n = 2, and Table 3 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.

```
## Loading required package:  MASS
```

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 2: 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 3: 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 4

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

2.5 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 100 independent rolls of a die is given in Figure 3.

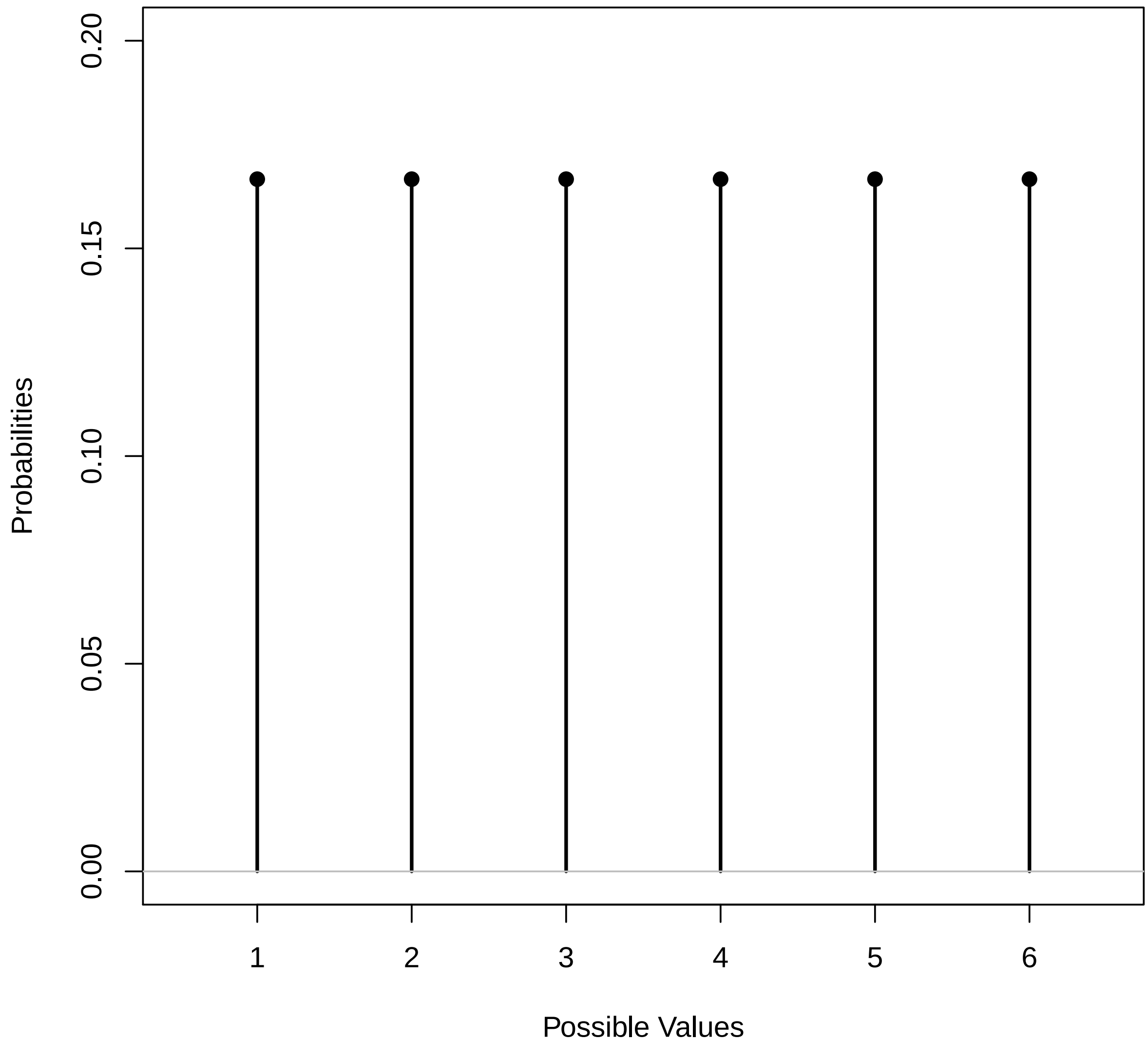

Figure 1: Plot method called on a fair die random variable.

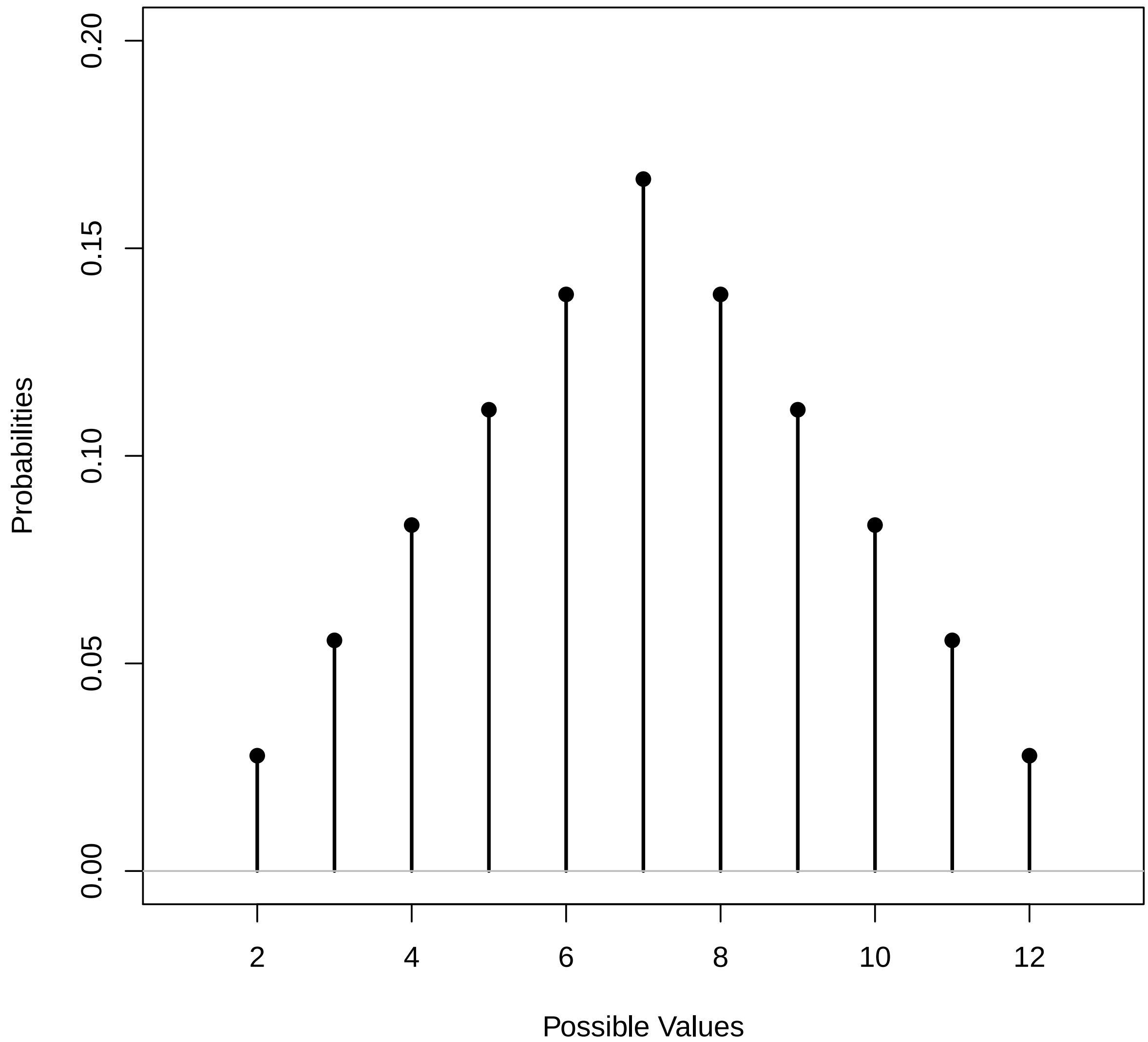

Figure 2: Plot method called on a sum of two fair die random variable.

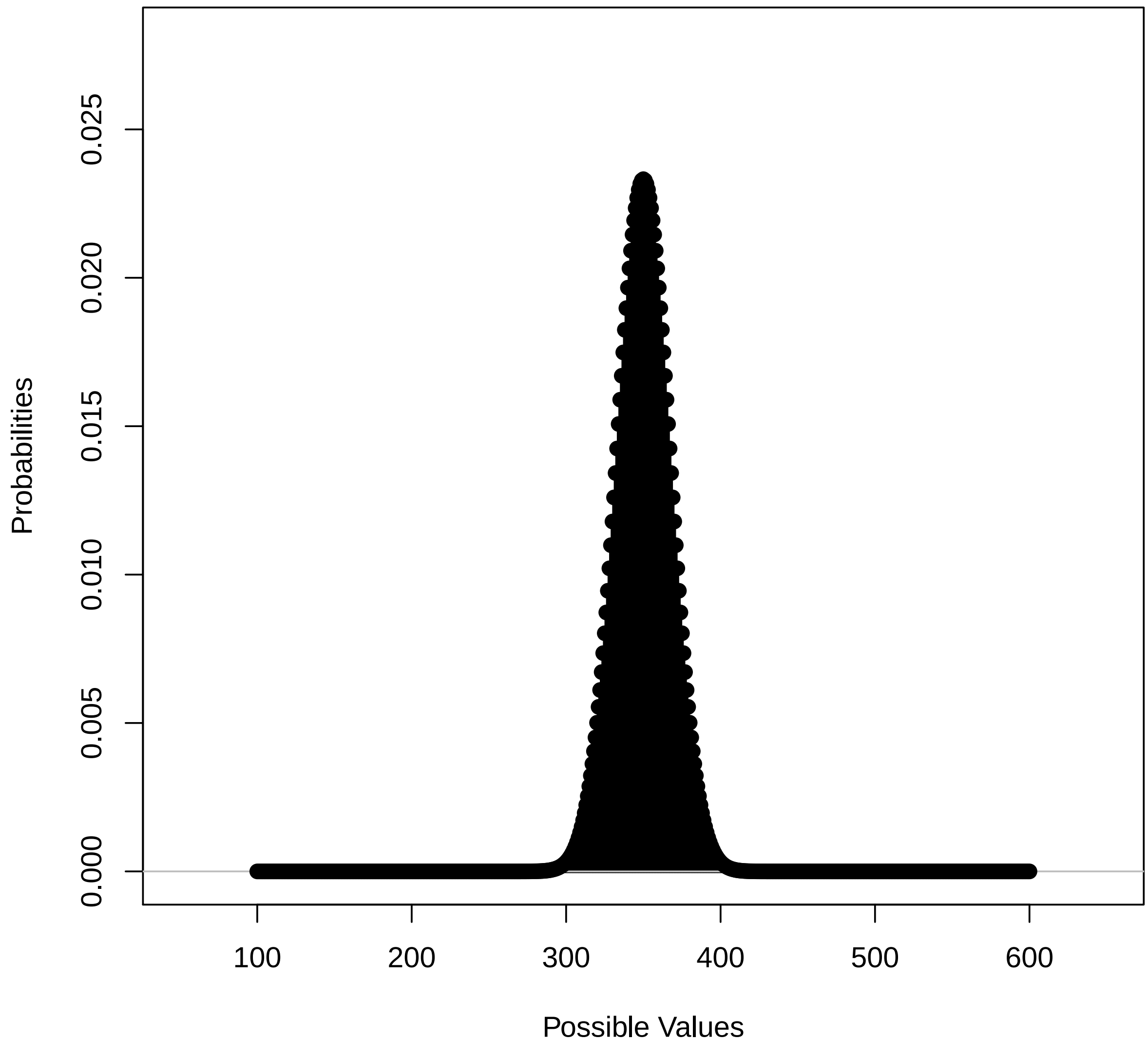

Figure 3: Plot method called on a sum of 100 fair die random variable.

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.

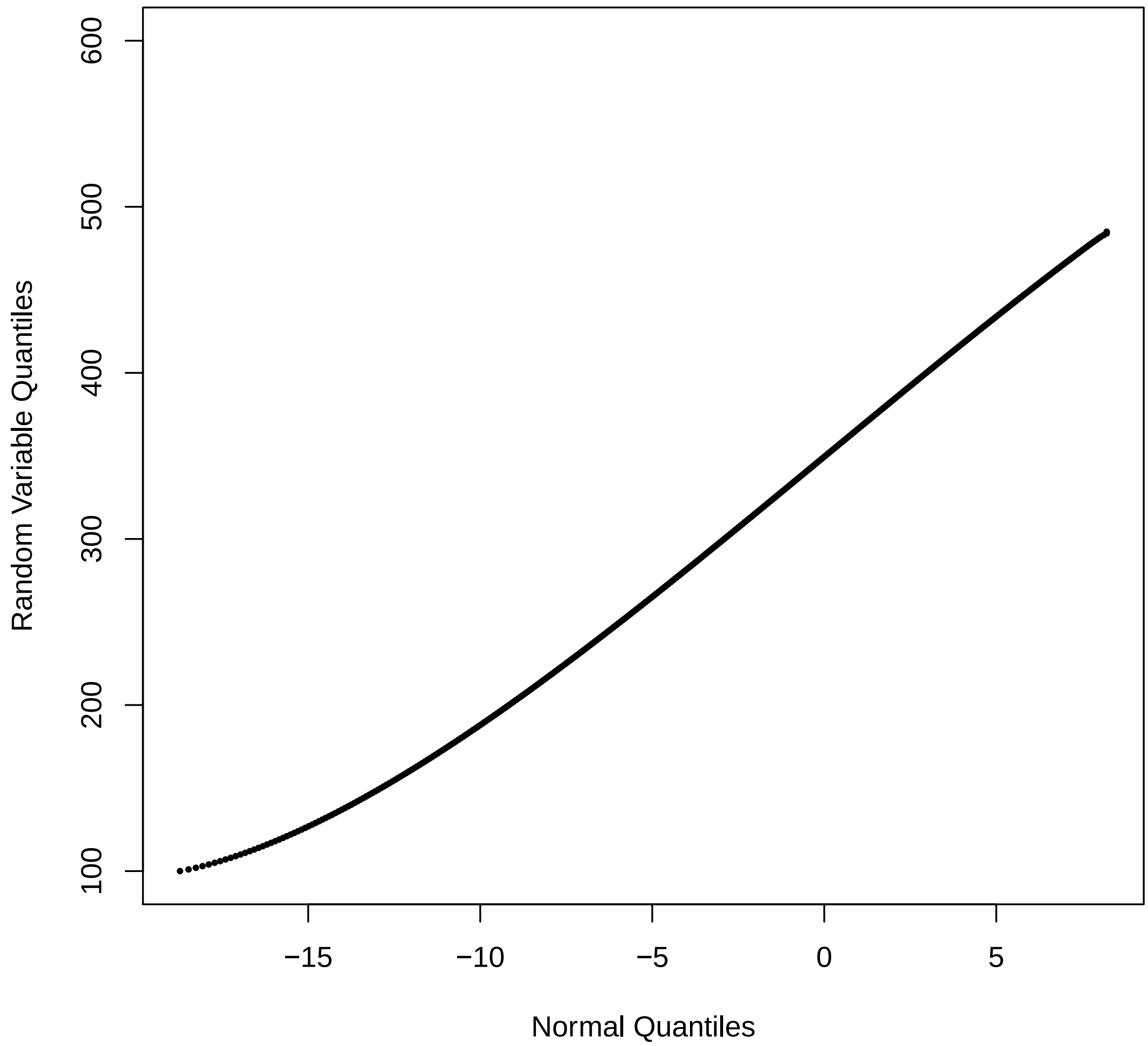

Figure 4: qqnorm method called on a sum of 100 fair die random variable.

3 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 5.

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 5: List of the simulation functions contained in discreteRV.

3.1 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


## 1/6 1/6 1/6 1/6 1/6 1/6 1/6 1/6 1/6 1/6
##   3   5   1   6   3   3   2   6   5   5
## attr(,"RV")
## random variable with 6 outcomes
##
##   1   2   3   4   5   6
## 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] 3 5 1 6 3 3 2 6 5 5
```

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 0.1 0.3 0.0 0.3 0.2
```

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

```
## [1] 0.3
```

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

```
## [1] 0.5
```

4 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.