

# R for Statistics 571

## Binomial Distribution and Proportions

Bret Larget

September 23, 2010

### 1 Bar Graphs

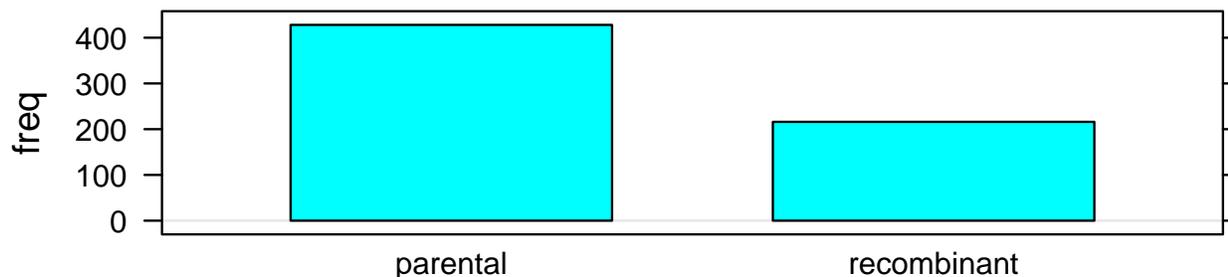
We will create bar graphs in R using `barchart()` in the `lattice` package. Begin by loading the `lattice` package.

```
> library(lattice)
```

The first argument to `barchart()` is a formula of the form `y ~ x` where `y` is an array with the frequencies and `x` is an array with the names of the groups. By default, `barchart()` may not extend bars down to 0. (Why, Deepayan, why?!?) To remedy this poor and misleading default behavior, `barchart()` should always include an additional argument `origin=0`. Note the general behavior of functions in R: if arguments are not named, their position in the list of arguments determines which argument it is interpreted as. But, if the argument is named, it can appear anywhere in the list. Here is a simple bar graph for the fruit fly recombination data.

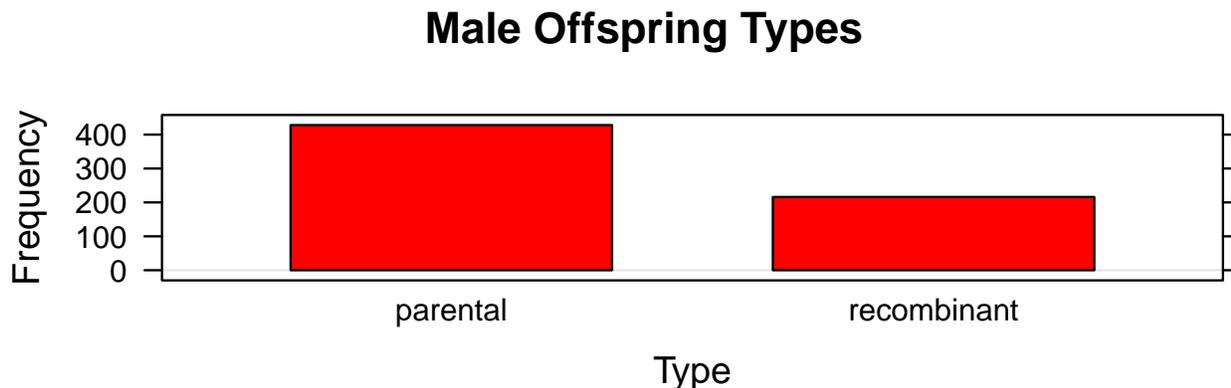
In addition, the commands shown here include the function `plot()` wrapped around the command to make plots. This is not necessary (but not harmful) when typing `lattice` plotting commands directly into R, but is needed for documents like this one where R reads commands in from a file.

```
> freq = c(216, 428)
> type = c("recombinant", "parental")
> plot(barchart(freq ~ type, origin = 0))
```



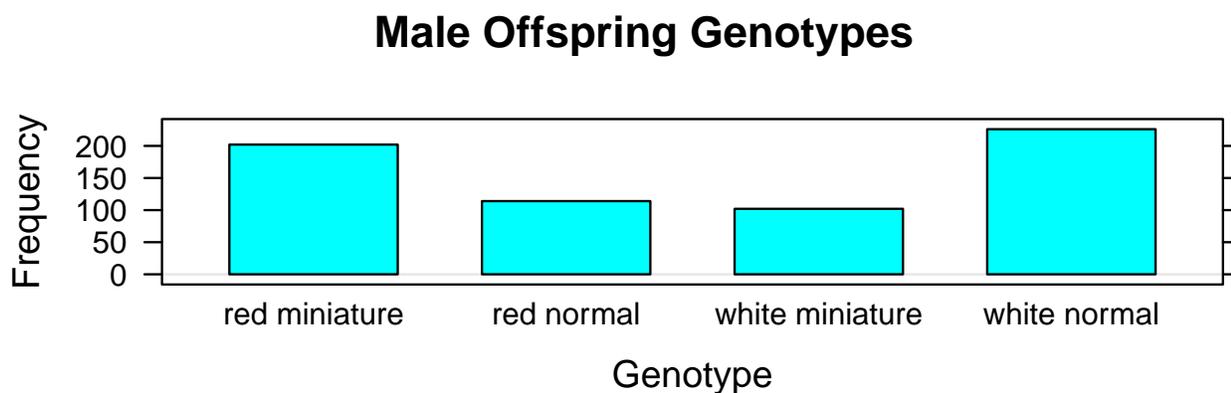
This plot can be customized by adding additional arguments to specify, for example, labels on the x- and y-axes (with arguments `xlab` and `ylab`), a title (with argument `main`), and color (with argument `col`).

```
> plot(barchart(freq ~ type, origin = 0, xlab = "Type",
+   ylab = "Frequency", main = "Male Offspring Types",
+   col = "red"))
```



The next example is very similar, but uses a data frame to hold the frequencies and multiple factors. Note the use of `paste()` to paste together the level names for color and wing and `factor()` to create a new factor with the four combinations of color and wing.

```
> frequency = c(226, 202, 114, 102)
> color = c("white", "red", "red", "white")
> wing = c("normal", "miniature", "normal",
+   "miniature")
> phenotype = c("parental", "parental", "recombinant",
+   "recombinant")
> my.data = data.frame(frequency = frequency,
+   color = color, wing = wing, phenotype = phenotype)
> plot(barchart(frequency ~ factor(paste(color,
+   wing))), data = my.data, origin = 0, xlab = "Genotype",
+   ylab = "Frequency", main = "Male Offspring Genotypes"))
```



The documentation on `barchart()` is available by typing `?barchart` at the prompt. Like most `lattice` documentation, it is incomplete (any mention of the argument `origin`?) and assumes more familiarity with R than beginners typically possess.

## 2 Binomial Probabilities

The two primary functions for computing binomial probabilities are `dbinom()` which computes probabilities at individual values and `pbinom()` which computes sums of binomial probabilities. The function `sum()` may be used in conjunction with `dbinom()` also to compute sums of binomial probabilities. Specifically, `dbinom(k,n,p)` calculates  $P(X = k)$  when  $X \sim \text{Binomial}(n, p)$  and `pbinom(k,n,p)` calculates  $P(X \leq k)$ .

The second homework set had a couple questions with  $X \sim \text{Binomial}(5, 0.7)$ . Here are several example calculations.

Find  $P(X = 4)$ .

```
> dbinom(4, 5, 0.7)
```

```
[1] 0.36015
```

Find  $P(X \leq 4)$  using both `dbinom()` and `pbinom()`.

```
> sum(dbinom(0:4, 5, 0.7))
```

```
[1] 0.83193
```

```
> pbinom(4, 5, 0.7)
```

```
[1] 0.83193
```

Find  $P(X \geq 3)$  using both `dbinom()` and `pbinom()`. Note that  $P(X \geq 3) = 1 - P(X < 3) = 1 - P(X \leq 2)$ .

```
> sum(dbinom(3:5, 5, 0.7))
```

```
[1] 0.83692
```

```
> 1 - pbinom(2, 5, 0.7)
```

```
[1] 0.83692
```

The entire distribution can be found by giving more than one value for the first argument.

```
> dbinom(0:5, 5, 0.7)
```

```
[1] 0.00243 0.02835 0.13230 0.30870 0.36015 0.16807
```

The second and third argument can also be replaced by sequences. For example, here is  $P(X = 0)$  for  $n = 1, 2, 4, 8, 16, 32$  and  $p = 0.5$  and  $P(X = 5)$  for  $n = 10$  and  $p = 0.1, 0.2, \dots, 0.9$ .

```
> dbinom(0, 2^(0:5), 0.5)
```

```
[1] 5.000000e-01 2.500000e-01 6.250000e-02 3.906250e-03
```

```
[5] 1.525879e-05 2.328306e-10
```

```
> dbinom(5, 10, (1:9)/10)
```

```
[1] 0.001488035 0.026424115 0.102919345 0.200658125
```

```
[5] 0.246093750 0.200658125 0.102919345 0.026424115
```

```
[9] 0.001488035
```

### 3 Binomial Random Variables

The function `rbinom()` generates pseudo-random numbers. This series of commands will generate 10,000 different binomial random variables, each with  $n = 8$  and  $p = 0.2$ . The `table()` function will display how many outcomes of each value were sampled.

```
> s = rbinom(10000, 8, 0.2)
> table(s)
```

s	0	1	2	3	4	5	6	7
	1715	3352	2920	1467	448	86	11	1

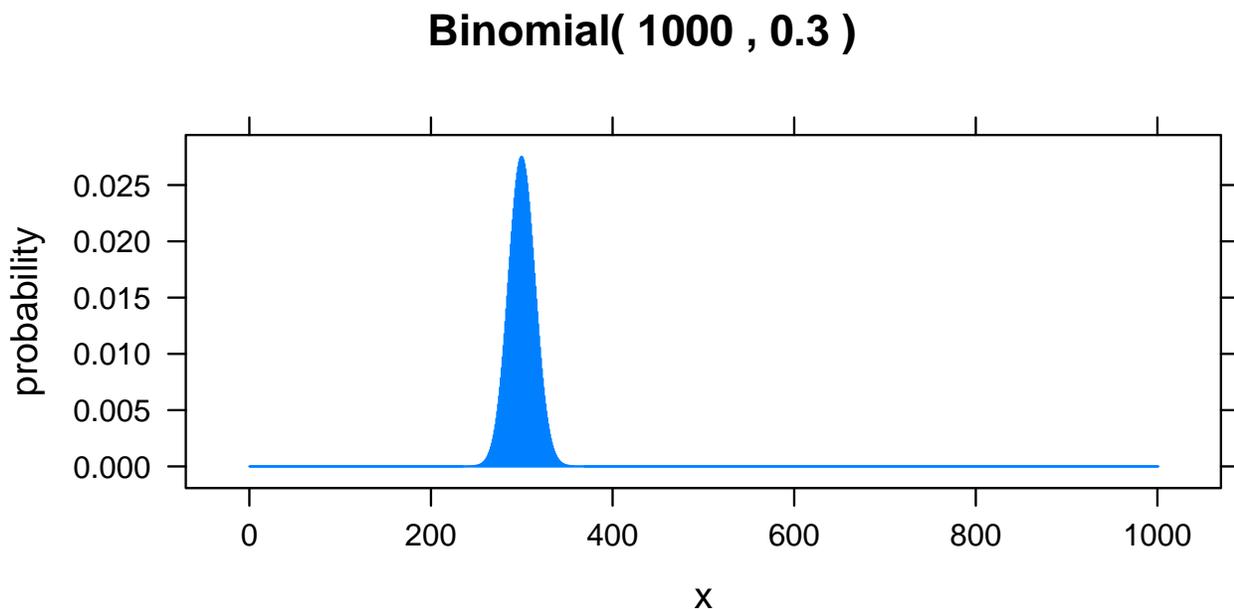
### 4 Graphing the Binomial Distribution

To graph the binomial distribution, we will write a function to do the job. Once the function is written, you can put it into a file, use `source()` to read it into R when desired, and then have access to it. It is also possible to construct the graph separately each time, but writing and saving a function is a good way to reduce repetitive typing of complicated series of commands. The function `gbinom()` is defined in the file `gbinom.R`. It includes quite a few complicated features. We will describe the function writing in more detail with a simpler example.

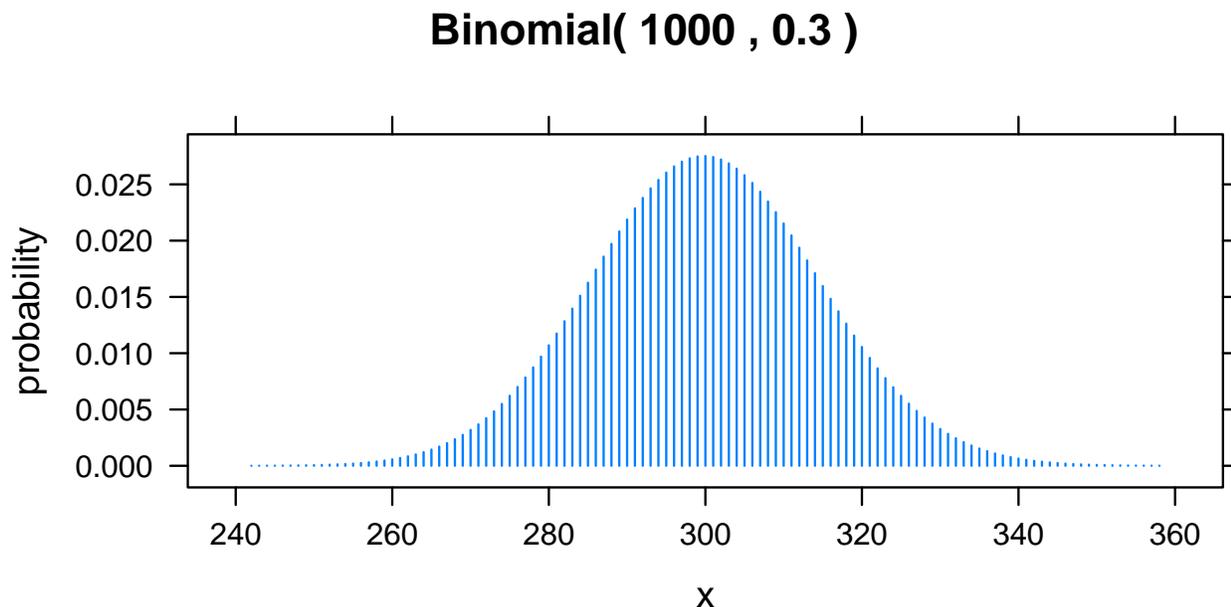
We create a function in R using `function()`. We will use `xyplot()` from `lattice` to do the work. We can write the function so that by default, it shows the entire distribution, but includes an argument `scale` which will cause the graph to be drawn from the 4 SDs below to 4SDs above the mean.

The following two graphs show the effects of setting `scale`.

```
> source("gbinom.R")
> plot(gbinom(1000, 0.3))
```



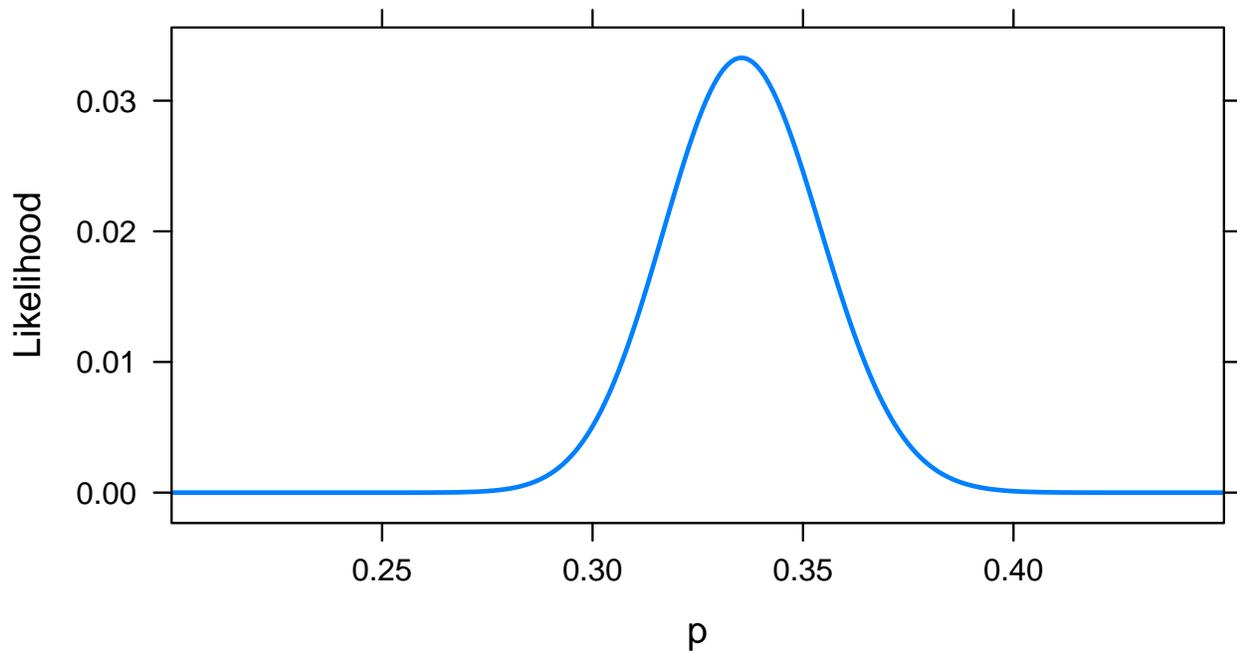
```
> plot(gbinom(1000, 0.3, scale = T))
```



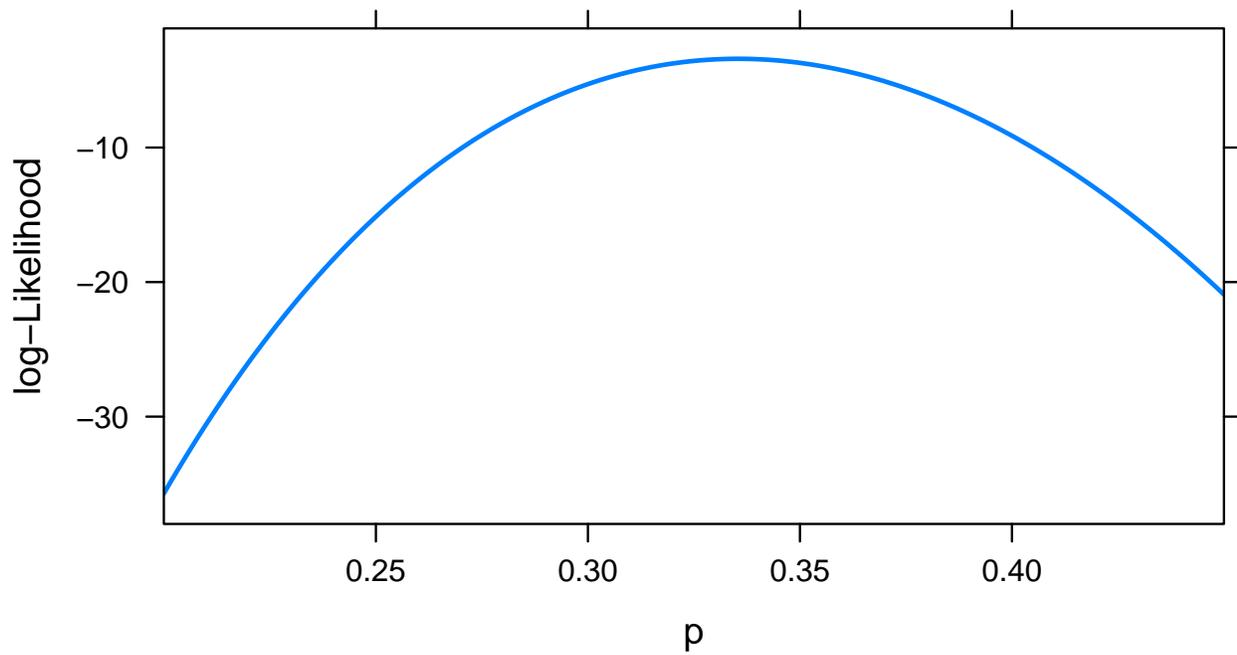
## 5 Graphing the Likelihood

The following example shows how to use `xyplot()` with argument `type="l"` to graph a line representing the likelihood and then log-likelihood from the fruit fly data in class. We use `seq()` to create an array of the values for which we want to compute the likelihood and `dbinom()` for the values themselves. The first two arguments to `seq()` are the start and end values and the third argument is the difference between points. Here we evaluate  $p$  from 0.2 to 0.45 in steps of size 0.001. Optional arguments to `xyplot()` include `lwd=2` to make the line width twice as large as normal, `xlim=c(0.2,0.45)` to force the limits of the x-axis to span these values, and `ylab` to specify a label for the y-axis. In the second plot, `log()` is the natural logarithm.

```
> p = seq(0.2, 0.45, 0.001)
> y = dbinom(216, 644, p)
> plot(xyplot(y ~ p, type = "l", ylab = "Likelihood",
+           lwd = 2, xlim = c(0.2, 0.45)))
```



```
> plot(xyplot(log(y) ~ p, type = "l", ylab = "log-Likelihood",  
+         lwd = 2, xlim = c(0.2, 0.45)))
```



## 6 Confidence Intervals

The text describes a method for building confidence intervals that uses a small adjustment to the sample proportion. The calculations are simple enough, but we can write a function in R to do all the steps for us. Here, we select the name of the function to be `p.ci` and the two arguments to be `x` and `n`, the summary statistics from the sample. Here is code to create a function to do the work, and then an application of it using data from lecture. The function `return()` specifies the value that is returned by the function. By default, it is whatever object is created last, but an explicit use of `return()` make reading the function easier. Note that `c()` is a common utility function used to concatenate (combine) different objects into one. In this case, `c()` concatenates expressions for the lower and upper endpoints of the confidence interval.

```
> p.ci = function(x, n) {
+   p.prime = (x + 2)/(n + 4)
+   margin.error = 1.96 * sqrt(p.prime * (1 -
+     p.prime)/(n + 4))
+   return(c(p.prime - margin.error, p.prime +
+     margin.error))
+ }
> p.ci(212, 644)
```

```
[1] 0.2940355 0.3664583
```

## 7 Hypothesis Tests

The method for hypothesis tests we have explored so far is the binomial test. R's version of this test is `binom.test()`, the first argument of which is the count `x` and the second of which is the total sample size `n`. Additional arguments are `p` which is the null proportion (default value is 0.5) and `alternative` which must be one of "two.sided" (the default), "less", and "greater". As with any argument in an R function which specifies a list of possible values, you need only type enough characters to distinguish it for other options (but for code clarity, typing the whole thing is helpful). For one-sided tests, the calculated p-value is the sum of binomial probabilities less than `x` for `alternative="less"` and the sum of binomial probabilities greater than `x` for `alternative="greater"`. For `alternative="less"`, `binom.test()` computes the p-value as the sum of all probabilities with outcomes less than or equal to the probability of `x` under the null hypothesis. This behavior is consistent with a likelihood-based approach to inference, but differs from the method presented in lecture in which *at least as extreme as* is interpreted as *at least as far from the mean as* and not *at most as probable as*. For the specific example in lecture, this results in a difference, but most of the time, the calculations will be the same.

Here, for example, is the result if we changed the problem and there had been 60 left-handed offspring from 270 total and the null hypothesis  $p = 0.25$ . Using the method in lecture,  $X \sim \text{Binomial}(270, 0.25)$  so  $E(X) = 67.5$  and outcomes at least as extreme as 60 are those 60 and below or  $67.5 + 7.5 = 75$  or higher. The p-value is

```
> sum(dbinom(c(0:60, 75:270), 270, 0.25))
```

```
[1] 0.3251164
```

using the *distance-from-the-mean* definition of extreme and

```
> binom.test(60, 270, p = 0.25, alternative = "two.sided")
```

Exact binomial test

data: 60 and 270

number of successes = 60, number of trials =

270, p-value = 0.3251

alternative hypothesis: true probability of success is not equal to 0.25

95 percent confidence interval:

0.1740749 0.2765797

sample estimates:

probability of success

0.2222222