

# STAT 457 , Fall 2010

## Lab # 7

### Looping Functions and Simulation

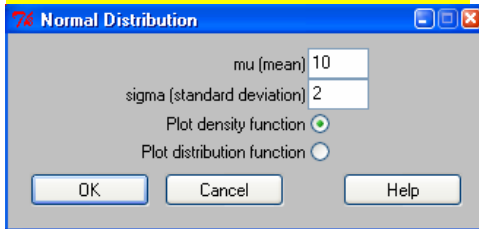
Topics Covered:

- R Commander—a brief formal introduction
- R Looping (i.e., control) functions `for`, `if`, `while`, `break`, and `ifelse`
- Simulation in R

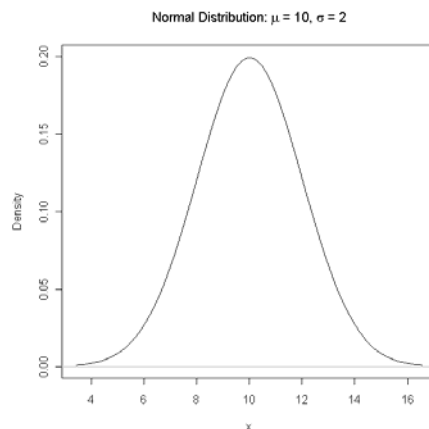
=====

Open up R Commander. We will want to do this lab in R Commander today, just to introduce you to this “add-on” to R, which might sometimes come in handy. I use R Commander for two essential handy purposes: [1] for doing simulations and [2] for reading in/editing data. The “point and click” features of graphing are sometimes handy, also. We will do a couple of these handy things now.

[1] In R Commander click on **DISTRIBUTIONS**  
**CONTINUOUS DISTRIBUTIONS** **NORMAL DISTRIBUTION**  
**PLOT NORMAL DISTRIBUTION** and you will get the dialog box shown below.



Put the values shown in the box, that is, a mean of 10 and std. dev. of 2, and we want to plot the density function, then **OK** . Now, go to the R Gui graphics screen and copy/paste the picture shown below to

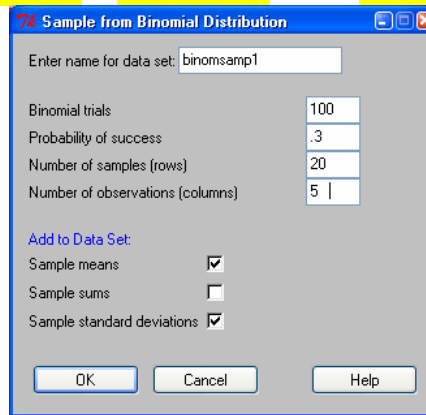


your WORD document.

If you now return to R Commander, you will see printed out the commands it automatically generated to produce your plot. Neat, huh?

[2] We can do a quick “point and click” sampling of various distributions in R Commander, also. In R Commander, click on **DISTRIBUTIONS**

**DISCRETE DISTRIBUTIONS BINOMIAL SAMPLE FROM BINOMIAL** to



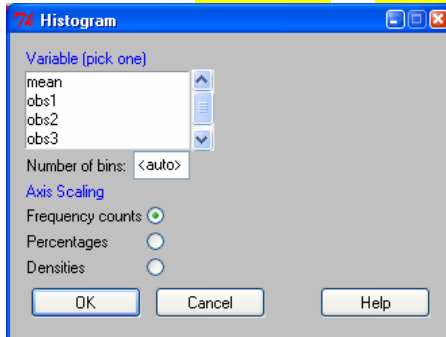
get the following screen.

Put the values shown into the screen, including the title binomsamp1 and click **OK** . Now click **VIEW DATA SET** to get the below

	obs1	obs2	obs3	obs4	obs5	mean	sd
sample1	39	38	33	32	26	33.6	5.224940
sample2	31	30	26	23	34	28.8	4.324350
sample3	24	29	35	29	29	29.2	3.898718
sample4	27	33	31	24	25	28.0	3.872983
sample5	38	25	33	31	24	30.2	5.805170
sample6	25	23	24	34	27	26.6	4.393177
sample7	30	32	27	20	28	27.4	4.560702
sample8	36	28	29	29	31	30.6	3.209361
sample9	35	25	32	33	34	31.8	3.962323
sample10	23	33	31	32	23	28.4	4.979960
sample11	34	29	26	26	32	29.4	3.577709
sample12	23	33	28	31	38	30.6	5.594640
sample13	31	32	30	37	38	33.6	3.646917
sample14	40	36	29	36	28	33.8	5.118594
sample15	31	31	32	30	37	32.2	2.774887
sample16	32	22	22	31	28	27.0	4.795832
sample17	30	31	30	32	28	30.2	1.483240
sample18	35	23	29	28	29	28.8	4.266146
sample19	26	33	40	28	29	31.2	5.540758
sample20	31	33	37	30	26	31.4	4.037326

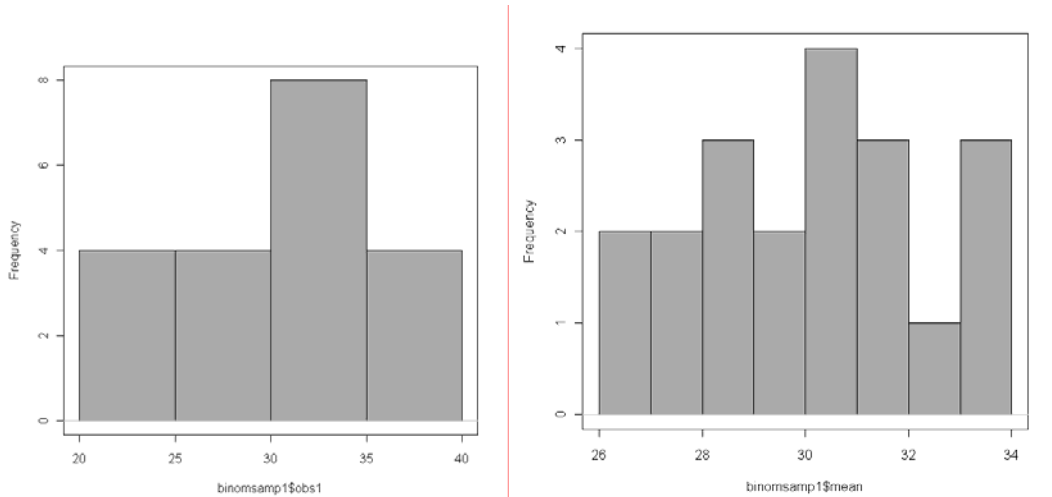
shown. Notice that the table, called binomsamp1, has 20 rows and 5 columns, and a 6<sup>th</sup> column with the mean of each row, and a 7<sup>th</sup> column with the std. dev. of each row, according to the boxes we clicked in the dialog box. Neat, huh?

Now click on **GRAPHS HISTOGRAM** to get the screen shown below.



Click on obs1 and copy/paste the histogram to your WORD document. Go back and get a histogram of the mean for your WORD document, also. They should look similar (but not exactly) like those

shown below. It is not exact because we are taking a sample, and my sample may be a bit different from your sample.



Again, notice the commands generated in R by R Commander “automatically” by your clicking on the buttons. R Commander seems to be an add-on to R by some programmer to help us use a bit of point-and-click to our R.

=====

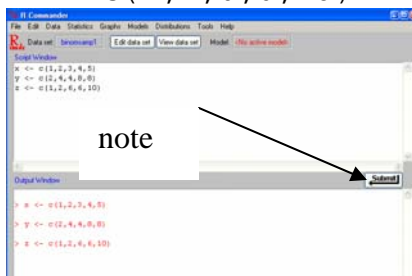
We occasionally to frequently want to have loops in our R “programs” in order to test logical statements for truth or falsehood, as well as to make up vectors of test results, among other uses.

The most common control command is the `if` statement. It has the “framework” ***if(logical condition) result***

Logical conditions are statements which are either TRUE or FALSE. The result is some R command which will be executed if the logical condition is true.

In R Commander, type in the following 3 vectors, x, y, and z in the top half screen. To execute these commands, highlight all 3, then click on **SUBMIT**, to get output on the bottom half of the screen. Your result should be similar to the one shown below.

```
x <- c(1, 2, 3, 4, 5)
y <- c(2, 4, 4, 8, 8)
z <- c(1, 2, 6, 6, 10)
```



Now type the `if` statement below, and notice how the `if TRUE` condition is executed.

```
if (x[1]<3) print("x value is less than 3")
```

Now type the statements below, and notice how the `if TRUE` condition is ignored. The next statement is executed, however.

```
if (x[1]>3) print ("x is greater than 3")
print("x must be less than or = 3")
```

Type in the below “greater than, equal to” conditional statement.

```
if(x[3]>=3) print ("x value is greater than or equal to 3")
```

Now type in the “compound AND” conditional—the AND is a `&` symbol

```
if(x[2]<3 & z[3]<=7)
print ("x is less than 3 and z is less than or equal to 7")
```

Type this `FALSE` “compound AND” conditional.

```
if(x[2]>3 & z[3]<=7)
print("x is less than 3 and z is less than or equal to 7")
print("at least one of the conditions above must be FALSE")
```

Now type the following “compound AND”—note how the equal sign in a conditional is a “double equal”.

```
if (x[2]==2 & z[3]<=7)
print("x = 2 and z is less than or = 7")
```

Below is the “compound OR”—the OR is a `|` symbol. Type this in.

```
if(x[4]=4 | z[3]=8)
print("x = 4 or z = 8 or both are true")
```

**3** Copy/paste all of your input and output from the above statements into your WORD document.

=====

Next, let us look at the `for` statement. The template for the `for` statement is:

```
for(i in num1:num2) {
    ...put program steps in here...
    ...
    ...}
}
```

I use `i` for all of my indexing, but you can use any letter you want. `num1` and `num2` are numbers, and they go sequentially, from `num1` to `num2`, for every iteration of the “for loop”. Next comes the curly braces, `{` and `}`, where the program you want to loop is placed between the curly braces. So, for example, the program

```
for(i in 1:5) {
    print(x[i])
    print("now go update i and print another value of x
out")
}
```

Notice that I usually indent the “program” between the curly braces about 5 spaces. This is a style thing I do that you don’t have to do if you don’t want to.

**[4]** Type the `for` loop above and the output to your WORD document.

**[5]** Below is code which combines `if` with `for` loops. Type this in Commander, execute it, and copy/paste output to WORD.

```
for(i in 1:4) {
  if(x[i]>2) print("x is greater than 2 for this
iteration")
  if(x[i]<=2) print("x is less than or equal to 2 for
this iteration")
}
```

=====

The final control statement we want to look at today is the `while` statement. It is an alternative to the `if` statement, as demonstrated in the little program loop below.

**[6]** Execute this code in Commander, and copy/paste the input and output to WORD.

```
i <- 1
while(x[i] <=3) {
print("this x value is less than or equal to 3")
i <- i + 1
}
```

=====

U of Montana has a lady Griz basketball player who is a 72% free throw shooter, a 45% 2 point shooter, and a 34% 3 point shooter. On average she gets 8 free throw attempts, 15-2 point shots and 7-3 point shots per game. Simulate her shooting game to estimate how many points, on average, she should make per game. Run a simulation of 24 games, since this is how many games she will run in a season.

**[7]** Download and run the script `ladygriz.R` . Include the output from 5 simulations of 24 games (i.e., run the script 5 times and copy/paste output from each of the 5 results in WORD). The script is reproduced below.

```
# Script to simulate 24 basketball games
# of a Lady Griz player

games <- 24
freethrow <- .72; twopoint <- .45; threepoint <- .34
attempt1 <- 8; attempt2 <- 15; attempt3 <- 7
points <- 0
totpoints <- c() # this makes room in R for a vector
totpoints

for(i in 1:games) {
```

```

    point1 <- rbinom(1, size=1, prob=freethrow)
    point2 <- rbinom(1, size=1, prob=twopoint)
    point3 <- rbinom(1, size=1, prob=threepoint)
    totpoints[i] <- point1*attempt1 + 2*point2*attempt2
      + 3*point3*attempt3
  }
  totpoints
  print("points per game is")
  sum(totpoints)/games

# another way to express result
cat("average points per game =", mean(totpoints), "\n")

```

**8** On average, how many points will our Lady Griz be expected to score per game this season, based upon your simulations?

=====

**EXTRA CREDIT:**

Change the number of games to 35, the shooting percentage to 82% for free throws, 52% for 2 point shots, and 40% for 3 point shots. Find the average points made per game.