Syllabus

Calendar

Instructor Insights

Readings

Class Slides

Assignments

Exams

Studio Resources

Related Resources

R Tutorial 1B: Random Numbers

In order to run simulations of experiments with random outcomes we make use of R's ability to generate random numbers. More precisely it generates a 'pseudo-random' number, which behaves in many ways like a truly random number.

The function sample(x, k, replace = TRUE)

# sample(x,k) generates a random permutation of k objects from the vector x. That is, all k choices are different

```
# start with a vector x with 5 elements.&nbsp;
> x = 1:5
> x
[1] 1 2 3 4 5
```

```
# randomly sample 3 of the elements of x.&nbsp;
> sample(x,3)
[1] 2 5 1
> sample(x,3)
[1] 3 1 4
> sample(x,3)
[1] 1 2 5
# Note every element is chosen at most once.
```

```
# randomly sampling 5 elements of x is a permutation of all 5 elements.&nbsp;
> sample(x,5)
[1] 1 3 2 5 4
> sample(x,5)
[1] 2 3 1 4 5
```

```
# You can't pick more than 5 different elements from a set of 5 things, so R gives an error.&nbsp;
> sample(x,6)
Error in sample.int(length(x), size, replace, prob) :&nbsp;&nbsp;
 cannot take a sample larger than the population when 'replace = FALSE'
```

Allowing repeated elements
# Sometimes you want to allow repeats. For example when we roll a die repeatedly we expect to see numbers repeating. We can think of this as picking a random element from a set and then putting it back, i.e. replacing it, so it can be drawn again.

# In R we do this by setting the optional argument replace=TRUE

```
# Note that we get can get repeated values&nbsp;
> sample(x,3,replace=TRUE)
[1] 3 1 4
> sample(x,3,replace=TRUE)
[1] 4 5 4
> sample(x,3,replace=TRUE)
[1] 2 1 5
> sample(x,5,replace=TRUE)
[1] 5 1 5 1 4
> sample(x,5,replace=TRUE)
[1] 3 5 5 2 1
> sample(x,5,replace=TRUE)
[1] 4 2 3 2 1
```

```
# Now there is no problem asking for more than 5 things from a set of 5 elements.&nbsp;
> sample(x,12,replace=TRUE)
[1] 1 1 2 1 4 2 3 4 1 2 4 5
> sample(x,12,replace=TRUE)
[1] 3 2 5 1 4 2 4 1 4 5 3 1
```

# To generate an m x n array of random values we can use the function sample function followed by the matrix function.

# Let's simulate rolling a die.
# We use 1:6 to make vector (1,2,3,4,5,6).

Feedback

```
# Generate a 3 x 4 array of random dice rolls.&nbsp;
> y = sample(1:6, 12, replace=TRUE)
> matrix(y,nrow=3,ncol=4)
     [,1] [,2] [,3] [,4]
[1,]    1    3    6    1
[2,]    2    2    3    6
[3,]    2    1    4    3
```

```
# Or we could make it a 2 x 6 array.
> matrix(y,nrow=2,ncol=6)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    2    6    4    6
[2,]    2    3    1    3    1    3
```

Simulation

# First let's simulate rolling 1 die 3 times and checking if one of the rolls was a 6.

```
# First we use sample to generate 3 samples from 1:6
> x = sample(1:6,3, replace=TRUE)
> x
[1] 6 4 1
```

```
# To check if any of the 3 rolls are 6 we use the following command. It returns a vector of TRUE or
FALSE depending on whether that entry of x is 6 or not. Note the use of the double equal sign. We
can't use a single equal sign because that would mean 'set the value of x to 6'. Compare the result
with the value of x above.
> x == 6
[1] TRUE FALSE FALSE
```

```
#  We can also see which elements of x are less than 6
> x < 6
[1] FALSE TRUE TRUE
```

# Now let's roll the die 5000 times and see what fraction of the rolls give 6. We expect that about 1/6 of them will be.

```
# Simulate 1000 rolls
> x = sample(1:6,1000, replace=TRUE)
```

```
# x == 6 gives a vector of TRUE or FALSE. R is clever, when we sum the vector: each TRUE counts as 1
and each FALSE counts as 0. So the sum is the number of TRUE's. In this case that means the number
of 6's, which happens to be 168.
> sum(x == 6)
[1] 168
```

```
# Divide by 1000 to get the fraction of 6's.
> sum(x == 6)/1000
[1] 0.168
```

```
# Compare that with the theoretical value of 1/6.
> 1/6
[1] 0.1666667
# Not bad!
```

# Now let's estimate the probability of getting at least one 6 in 4 rolls.

# Goal: estimate the probability of getting at least one 6 in 4 rolls.
# Experiment: roll 1 die 4 times and check for a 6.
# Repeat the experiment 10 times and see what fraction of times this happens.

```
# So you can see all the commands at once, we'll show them all and then explain them later. For
commenting, we'll put a command number before each '>'
1. > x = matrix(sample(1:6, 4*10, replace=TRUE), nrow=4, ncol=10)
2. > x
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    5    6    1    1    3    6    1    4    6     1
[2,]    3    6    6    4    3    2    2    1    6     5
[3,]    2    3    5    2    5    3    4    6    2     5
[4,]    2    6    6    5    5    3    1    4    5     3
3. > y = (x==6)
4. > y
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10]
[1,] FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE
[2,] FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
[3,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
[4,] FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
5. > z = colSums(y)
6. > z
[1] 0 3 2 0 0 1 0 1 2 0
7. > z > 0
[1] FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE
8. > sum(z > 0)
[1] 5
9. > mean(z > 0)
[1] 0.5
```

# Command 1: Generate a 4 by 10 random array. Each column represents one experimental trial of 4 rolls. The 10 columns represent the 10 trials.

# Command 2: Display x

# Command 3: See which of the rolls are 6's. Note the double equals to check which rolls were 6. The result y is an array of TRUE or FALSE. You can check that it picks out the 6's.

# Command 4: Display y

# Command 5: Sum each of the columns. The result is then number of 6's in each
trial (column).

# Command 6: Display z

# Command 7: If the column sum is greater than 0 there was at least one TRUE in the column, that is at least one 6. This command returns a vector of TRUE or FALSE representing whether or not the experiment yielded a 6.

# command 8: sum the vector in command 7 to get the number of trials that yielded a 6. We see that 5 out of 10 trials did.

# command 9: The mean function is just the sum divided by the number of trials. This is just 5/10 = .5. Half the trials yielded a 10.

```
# Let's repeat this but with 1000 trials instead of 10. Without all the comments it's pretty
short.
> x = matrix(sample(1:6, 4*1000, replace=TRUE), nrow=4, ncol=1000)
> y = (x==6)
> z = colSums(y)
> sum(z > 0)
[1] 525
> mean(z>0)
[1] 0.525
```

Our estimate of the probability of at least one 6 in 4 rolls is .525. This is pretty close to the theoretical value of .518.

```
The dim() function
# We can always check that x has 4 rows and 1000 columns using the dim() function.
> dim(x)
[1]    4 1000
```

One more simulation

```
# Goal: estimate the probability of getting a sum
of 7 when rolling two dice.
1. > ntrials = 10000
2. > x = matrix(sample(1:6, 2*ntrials, replace=TRUE), nrow=2, ncol=ntrials)
3. > y = colSums(x)
4. > mean(y == 7)
[1] 0.1658
```

# Command 1: We assign the number of trials to the variable ntrials. Writing code like this makes it easier to modify later. If we want to change the number of trials to 7 we just have to change this one line of code.
# Command 2: we create 10000 columns with 2 rows. That is, we run 10000 experiments of rolling 2 dice.

# Command 3: we sum each of the columns, i.e., we sum the two dice.
# Command 4: we find the fraction of sums that are 7.

# Note, this is essentially the exact answer of 1/6.

Exercise: Try to estimate the probability of two sixes when rolling two dice.