Module 8 Lab

This lab examines permutation tests in R. We use the O-ring example from *The Statistical Sleuth*, section 4.3 (and Module 8, Lecture 1) to walk through running a permutation test with simulations, rather than with exact combinatoric methods. For more information on the O-ring dataset, run `?ex2223`; be sure to install and load the *Sleuth3* package first.

Next, we conduct permutation test with a pre-written function in the *coin* package.

Permutation Test - O-rings

The O-rings, in combination with cold temperatures, were thought to be responsible for the *Challenger* space shuttle explosion. Therefore, we look at "O-ring incidents" divided into those occurring below 18 degrees Celcius, and those occurring above 18 degrees Celcius. The O-ring data is available in the *Sleuth3* package, but we will create it from scratch in the format of slide 9 of Module 8, Lecture 1.

```
cold <- c(1,1,1,3) # number of incidencts below 18 degrees C
warm <- c(rep(0, 17), 1, 1, 2) # number of incidencts above 18 degrees C
```

Notice in defining "`warm`" we use `rep(0, 17)` to get 17 repeated zeroes, which is less cumbersome than writing out 17 zeroes! Now we have our data. Our null hypothesis, as in lecture, is that both groups come from the same distribution. Here, instead of using the difference in group means as the test statistic, let's use the t-statistic.

$$t = \frac{\bar{x}_1 - \bar{y}_2}{\sqrt{s_1^2/n_1 + s_2^2/n_2}}$$

For our data, we can calculate this as follows.

```
T <- (mean(cold) - mean(warm))/sqrt(var(cold)/4 + var(warm)/20) # t-statistic
T
```

```
## [1] 2.531636
```

See sections 2.3.1 and 2.3.2 in *The Statistical Sleuth* for a review of the t-statistic. Now we need write a function to permute the groups and calculate the new test statistic.

To randomly permute the groups, we will use the `sample()` function. Let's see how it works before we write a permutation function. The implementation below samples four elements from the integers 1 to 24, without replacement. This means we cannot sample the same number twice.

```
sample(1:24, size = 4, replace = FALSE)
```

```
## [1] 11  7 13 24
```

Now to our permutation function. The first line below puts all the data points in one vector. Then `perm_o_ring` randomly reassigns the groups and calculates the t-statistic.

```
O_ring_data <- c(cold, warm) # Put all data in one vector
perm_o_ring <- function(){
  cold_indices <- sample(1:24, size = 4, replace = FALSE) # randomly select 4 integers from 1 to 24
  cold_perm <- O_ring_data[cold_indices] # Assign "cold" group
  warm_perm <- O_ring_data[-cold_indices] # Assign remaining elements to "warm" group
  (mean(cold_perm) - mean(warm_perm))/sqrt(var(cold_perm)/4 + var(warm_perm)/20)
}
perm_o_ring()
```

```
## [1] -2.703274
```

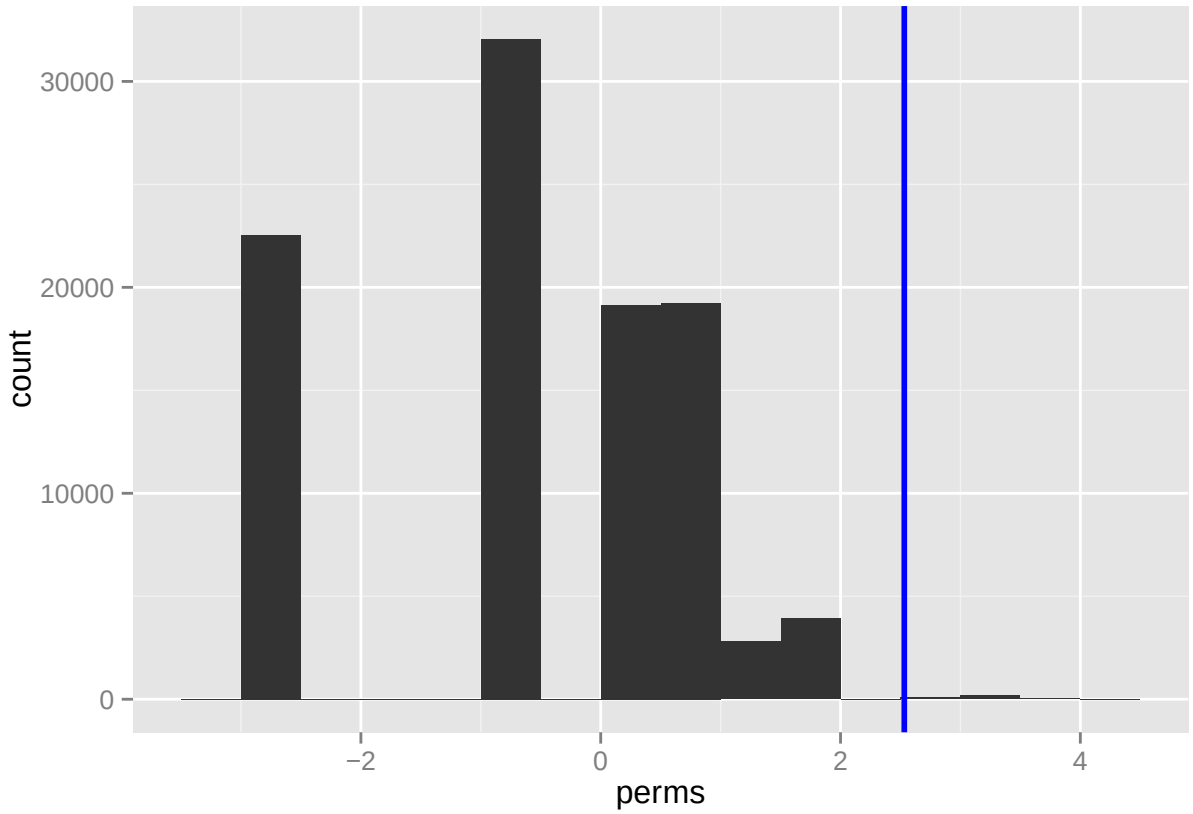
The function `sample()` randomly selects four integers from 1 to 24 to use as indices. Then we name the elements in `O_ring_data` at those indices the cold group; and the other elements–denoted by the subsetting operation `O_ring_data[-cold_indices]`–the "warm" group. The next line calculates the t-statistic. The final line runs the new function.

Now we are ready to conduct a permutation test. We want to randomly permute the groups and calculate a test statistic many times to simulate the null distribution. Then we can see how extreme the observed test statistic is compared to that null distribution.

```
set.seed(1986) # Challenger exploded in 1986
perms <- replicate(100000, perm_o_ring()) # randomly permute, calculate t-stat 100,000 times
mean(perms > T) # Proportion of t-stats more extreme than observed
```

```
## [1] 0.00214
```

```
library(ggplot2)
qplot(perms, binwidth = .5) + # Histogram of randomly assigned group t-stats
  geom_vline(xintercept = T, color = "blue", size = 1) # vertical line = observed t-stat
```



The first line uses our function `perm_o_ring()` to randomly reassign the groups and calculate the test statistic—100,000 times. This gives the simulated null distribution. The value returned from `mean(perms > T)` gives the proportion of outcomes with a test statistic more extreme than the one we observed; this is our permutation test p-value. You can see in the histogram that very few observations lie to the right of the blue line denoting the observed t-statistic.

O-ring Permutation Test - Technical Summary

For these 24 launches, there is very strong evidence to suggest that the standardized mean number of O-ring incidents is greater at temperatures below 18 degrees Celcius than above. A one-sided permutation test, using a t-statistic, gives a p-value of 0.00214. If the null hypothesis is true, the probability of an outcome as extreme or more extreme than what was observed is 0.00214.

O-ring Permutation Test - Coin Package

Now we run a permutation test with a pre-written function in the `coin` package. We can use the same data from before with one small addition.

```
O_ring_data
```

```
##  [1] 1 1 1 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2
```

```
Groups <- as.factor(c(rep(0,4), rep(1, 20))) # Create group label vector
str(Groups)
```

```
##  Factor w/ 2 levels "0","1": 1 1 1 1 2 2 2 2 2 2 ...
```

The first line returns the data, as a reminder that all observations are included in `O_ring_data`; the first four are "below 18 degrees" observations, the remaining 20 "above 18 degrees" observations. The function we are going to use in the `coin` package, called `oneway_test()`, requires group labels for the data. Therefore, we define `Groups` as a factor vector. This way, `oneway_test()` recognizes the group numbers as labels, not numeric values; `str(Groups)` demonstrates this.

We are ready to run the test.

```
library(coin) # load coin package
oneway_test(O_ring_data ~ Groups, distribution = "exact")
```

```
##
##  Exact 2-Sample Permutation Test
##
## data:  O_ring_data by Groups (0, 1)
## Z = 3.0604, p-value = 0.009881
## alternative hypothesis: true mu is not equal to 0
```

The output gives the name of the test, and the data used. The third line provides the observed test statistic, albeit a slightly different statistic than before. The third line also provides the p-value, but now it is a two-sided p-value. The final line gives the alternative hypothesis. Notice that because we use `distribution = "exact"` we get the same result as the combinatoric calculation from Lecture 1, Module 8. Use `?oneway_test` to learn more about the function.

For another option, see `permTS()` in the `perm` package, which also has an argument for running an exact permutation test.