Lab 9 Part II
Permutational non-parametric tests

There is a clever alternative to sums-of-squares based ANOVA that compares groups with distances measures, which does not require any assumptions about distributions. Conceptually, this is very similar to ANOVA. In ANOVA, the F-value is the ratio of variance between groups (signal) to variance within groups (noise). The bigger the signal to noise ratio, the larger the F-value, which can be converted to a p-value.

This also works with distances: Instead of an F-value, I can calculate a Delta-value (or an equivalent statistic that goes by some other name), which is the ratio of distances between observations between groups (signal) to distances within groups (noise). This gives me a signal to noise ratio that is my Delta value (equivalent to an F value). Next, I need to compare this to an expected distribution of that value (like an F-distribution).

The problem is that we don't know how that distribution looks like for non-normal data. So, the permutational ANOVA generates this distribution empirically, by randomly shuffling the class variables within the dataset. Now that the class variable has been randomized relative to the actual observations, you calculate a new randomly expected Delta-value. You randomize (permutate) your class variable again, and calculate another Delta-value.

Repeat 10,000 times and you get a Delta distribution specific to your type of data that you can use to get the p-value to indicate significance between groups. This is simply the position (percentile) of your original Delta-value on the distribution of 10,000 values that you generated. Hence, we assume nothing about the shape of the distribution.

9.6 Permutational Analysis of Variance

First, we have to install the lmPerm package:

```
install.packages("lmPerm")
library(lmPerm)
```

For this exercise use the original lentil dataset with two farms. Import and check if the data importedcorrectly as usual. Then, run a regular ANOVA for reference:

```
dat1=read.csv("lentils.csv")

head(dat1)

anova(lm(YIELD~FARM*VARIETY,data=dat1))
```

Next, let's run the permutational ANOVA as an alternative, which is included in the package lmPerm. Because the data is normally distributed, and both ANOVA and permutational ANOVA are equally powerful, there should be no big difference in the results:

```
out1=aovp(YIELD~FARM*VARIETY, seqs=T, data=dat1)
out1
```

```
TukeyHSD(out1)
```

In this case, it does not make sense to do a full set of pairwise comparisons. Given that you have a significant interaction term, and know that the story is different at the two farms, it is more sensible to subset the dataset for Farm1 and Farm2, and then run a single-factor permutational ANOVA and Tukey's HSD for the varieties at each farm separately.

Note that we used the option `seqs=T` , which calculates sequential sum of squares, just like under the default setting of a regular ANOVA implemented with the lm() function, which is a good choice for balanced designs.

The number of iterations can be controlled with the maxIter command (default is 5000):

```
anova(aovp(YIELD~FARM*VARIETY, seqs=T, maxIter=10000, data=dat1))
```

9.7 Permutational T-Test

By sub-setting the data to just two treatments, we can of course implement a permutational T-test as well. The first two lines subset the data. Then we carry out a regular T-test for reference, and the permutational version that does not make any assumptions about normality or homogeneity of variances for comparison.

```
ab1=dat1[dat1$FARM=="Farm1"& dat1$VARIETY!="C",]
head(ab1)
```

Standard T-Test:

```
t.test(ab1$YIELD~ab1$VARIETY)
```

Permutational T-Test

```
TukeyHSD(aovp(ab1$YIELD~ab1$VARIETY, seqs=T))
```

If you wish to follow-up on an ANOVA with a specific set of pairwise comparisons (the ones you care about) and then make a manual adjustment for multiple inference, you can run such permutational T-tests on various sub-sets of your data.

Enter the resulting p-values in a vector and make an adjustment for the appropriate experiment-wise α-level with the p.adjust function, which we covered earlier. Below, I assume that you want to make six comparisons, and I entered the first p-value obtained from the permutational T-test above for A-B in bold.

```
p.adjust(c(4.11e-05, 0.001, 0.001, 0.001, 0.001, 0.001), n=6,
method="holm")
```