Non-parametric Statistics: Notes 2
Two-sample methods

Gregory Matthews [1]

[1]Department of Mathematics and Statistics
Loyola University Chicago

Spring 2016

Outline

Permutation test

- Let's say we want to test formally if there is a difference between two groups.
- For instance, we could compare a new procedure to an old procedure.

Two sample tests

- How would you do this parametrically?
- We use a t-test with the assumption of normality
- Or you can use asymptotic theory and the Central Limit Theorem (CLT)

Permutation test

```
dat<-data.frame(score=c(37,49,55,57,23,31,46),method=c(rep("N",4),rep("O",3)))
dat
```

```
##   score method
## 1    37      N
## 2    49      N
## 3    55      N
## 4    57      N
## 5    23      O
## 6    31      O
## 7    46      O
```

```
#Computer the observed means
scoreMeans<-tapply(dat$score,dat$method,mean)
scoreMeans
```

```
##        N        O
## 49.50000 33.33333
```

```
obsDiffMeans<-diff(scoreMeans)
#Compute the observed medians
scoreMedians<-tapply(dat$score,dat$method,median)
scoreMedians
```

```
##  N  O
## 52 31
```

```
obsDiffMedians<-diff(scoreMedians)
```

Permutation test

```
library(gtools)
perms<-combinations(7,4)
dim(perms)
```

```
## [1] 35  4
```

```
permDiffs<-rep(NA,dim(perms)[1])
for (i in 1:dim(perms)[1]){
scoresTemp<-c(dat$score[perms[i,]],dat$score[setdiff(c(1:7),perms[i,])] )
datPermuted<-data.frame(score=scoresTemp,method=dat$method)
permDiffs[i]<-diff(tapply(datPermuted$score,datPermuted$method,mean))
}
```

Permutation test

```
#all differences of all permutations.
sort(permDiffs)
```

```
##  [1] -21.4166667 -16.1666667 -14.4166667 -12.6666667 -10.9166667
##  [6] -10.9166667  -9.7500000  -8.0000000  -7.4166667  -6.2500000
## [11]  -6.2500000  -5.6666667  -2.7500000  -2.1666667  -1.5833333
## [16]  -1.0000000  -1.0000000  -0.4166667   0.7500000   2.5000000
## [21]   2.5000000   3.6666667   4.2500000   4.2500000   5.4166667
## [26]   6.0000000   7.1666667   7.7500000   8.9166667   8.9166667
## [31]  12.4166667  13.0000000  14.1666667  17.6666667  19.4166667
```

```
#observed difference of means
obsDiffMeans
```

```
##         0
## -16.16667
```

Permutation test

- How can we use these simulated values to construct a p-value?
- Well, what is a p-value?
- It's the probability of observing something as or more extreme than what was actually observed.
- How many of the observations were AS or MORE extreme than what we actually observed?

Permutation test

```
#One-Sided p-value
sum(permDiffs<=obsDiffMeans)/length(permDiffs)

## [1] 0.05714286

#Two-Sided p-value
(sum(permDiffs<=obsDiffMeans)+sum(permDiffs>=abs(obsDiffMea

## [1] 0.1142857

#http://spark.rstudio.com/ahmed/permutation/
```

Permutation test

- The test we just did was exact in the sense that we looked at all possible permutations.
- However, sometimes the number of permutations is very large and so we simply take a sample of all possibilities.
- Let's look at that.

Permutation test

```
dat<-data.frame(score=c(37,49,55,57,23,31,46),
                method=c(rep("N",4),rep("O",3)))
dat
```

```
##   score method
## 1    37      N
## 2    49      N
## 3    55      N
## 4    57      N
## 5    23      O
## 6    31      O
## 7    46      O
```

```
scoreMeans<-tapply(dat$score,dat$method,mean)
scoreMeans
```

```
##        N        O
## 49.50000 33.33333
```

```
diff(scoreMeans)
```

```
##         O
## -16.16667
```

```
scoreMedians<-tapply(dat$score,dat$method,median)
```

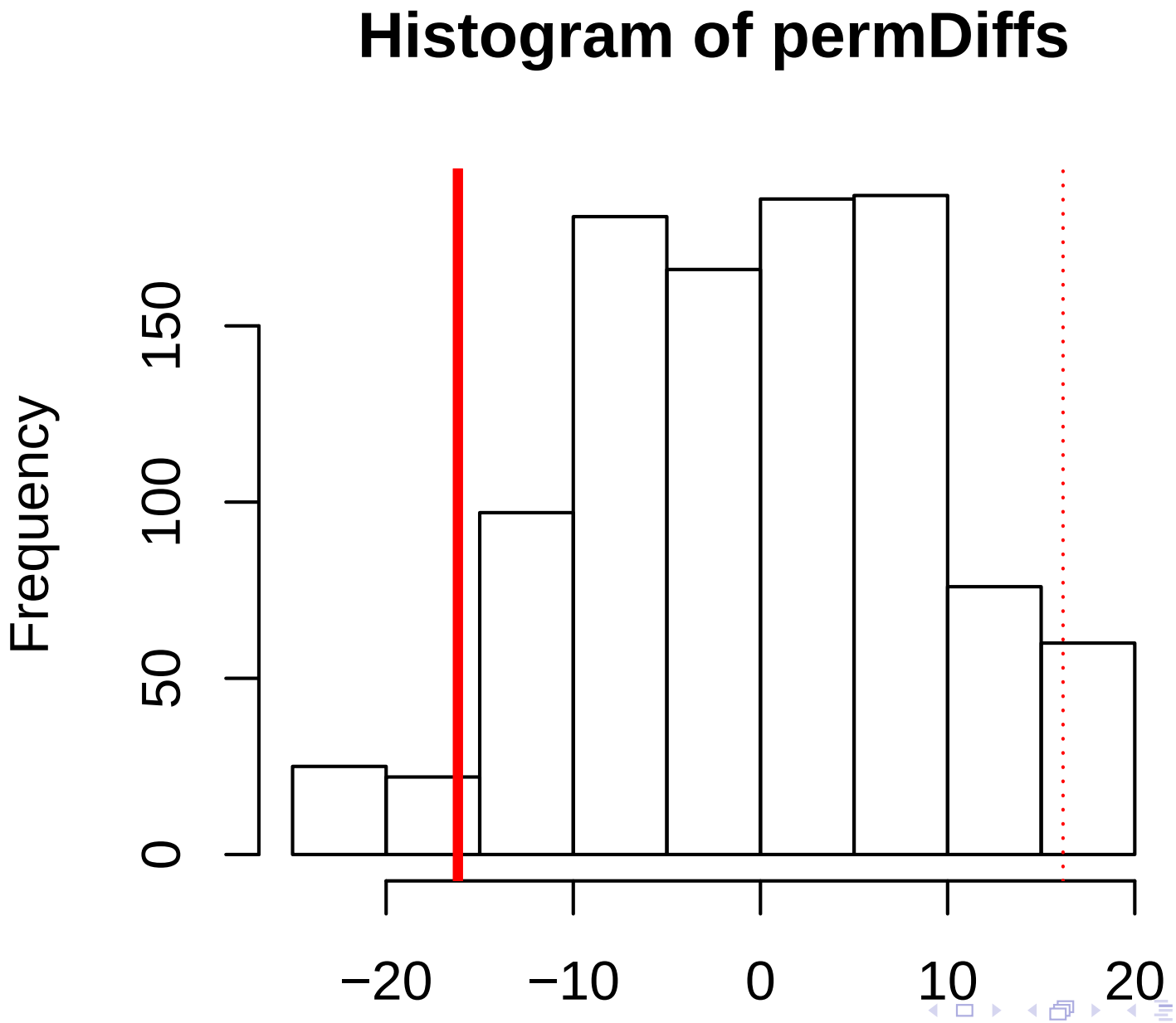
Permutation test

```r
set.seed(1234)
```

```r
nsim<-1000
permDiffs<-rep(NA,nsim)
for (i in 1:nsim){print(i)
datPermuted<-data.frame(score=
  sample(dat$score,replace=FALSE),method=dat$method)
permDiffs[i]<-diff(tapply(datPermuted$score,
  datPermuted$method,mean))
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
```

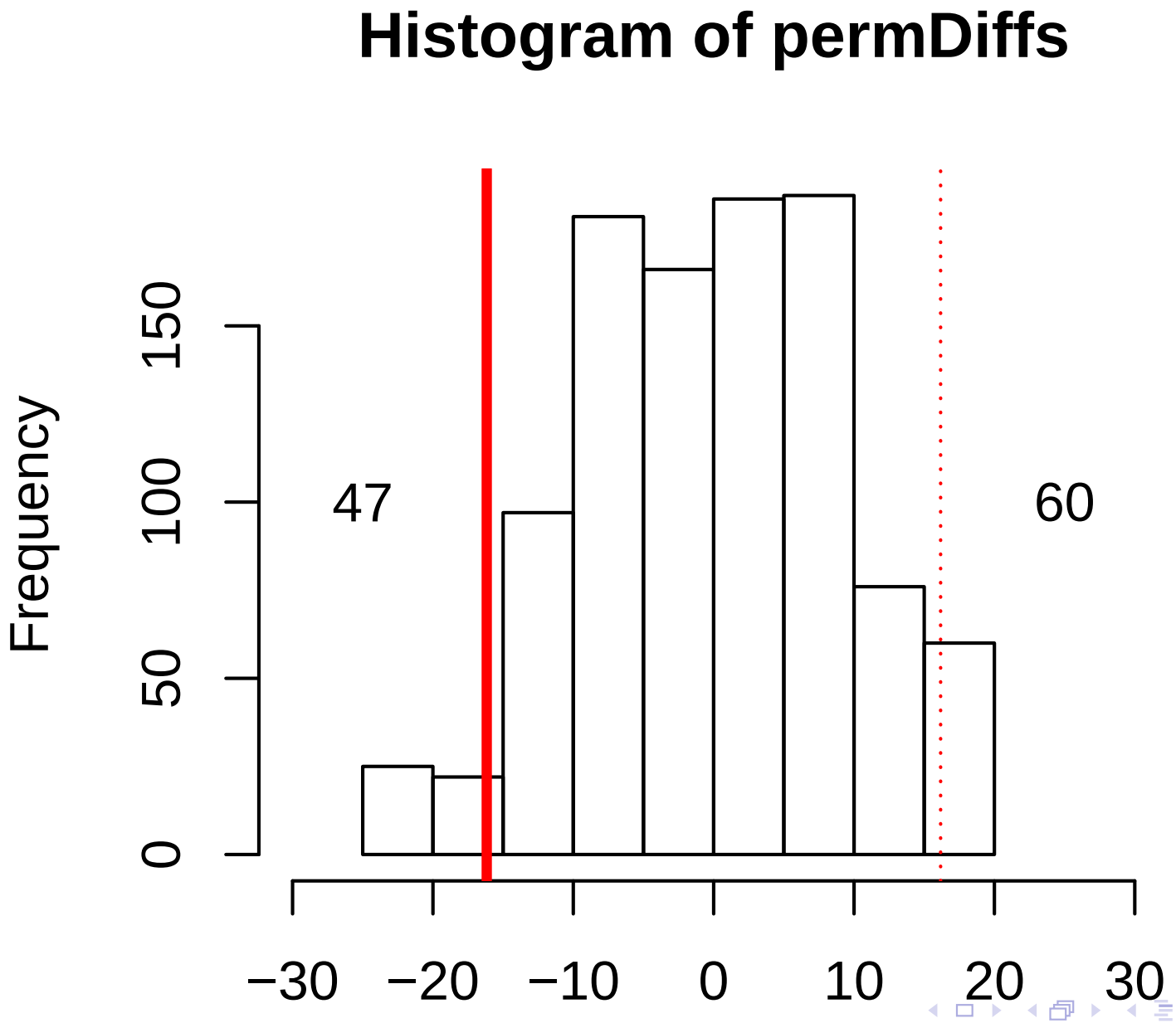
Permutation test



Permutation test

```
lower<-sum(permDiffs<=diff(scoreMeans))
higher<-sum(permDiffs>=abs(diff(scoreMeans)))
#How do we constuct a p-value now?
#Two sided p-value
(lower+higher)/nsim

## [1] 0.107

#One sided lower p-value
lower/nsim

## [1] 0.047

#Note: the two-sided p-value is not the
#one-sided p-value times 2 here.
#Why not?
```

Permutation test



```r
#R package for these permutation tests
library(perm)
permTS(dat$score~dat$method,alternative="two.sided",exact=TRUE)
```

```
##
##  Exact Permutation Test (network algorithm)
##
## data:  dat$score by dat$method
## p-value = 0.1143
## alternative hypothesis: true mean dat$method=N - mean dat$method=O is not equal to 0
## sample estimates:
## mean dat$method=N - mean dat$method=O
##                             16.16667
```

- SAS procedure is “npar1way”.

Wilcoxon Rank Sum and Mann-Whitney U Test

Wilcoxon Rank Sum

- Let's say we have two groups as before.
- Take all of the data in both groups, combine them, and then rank them.
- Add up the ranks from group 1 (or group 2, either will work).
- Call this sum W_1.
- Do a permutation test based on the ranks to find a p-value.

Wilcoxon Rank Sum and Mann-Whitney U Test

Mann-Whitney U Test

Value	Brand	Rank
3.6	Brand 1	1
3.8	Brand 2	2
3.9	Brand 1	3
4.0	Brand 1	4
4.1	Brand 2	5
4.3	Brand 1	6
4.5	Brand 2	7
4.8	Brand 2	8
		$W_2 = 22$

Wilcoxon Rank Sum and Mann-Whitney U Test

Mann-Whitney U Test

	Brand 1	Brand 2	
0	3.6	3.8	1
1	3.9	4.1	3
1	4.0	4.5	4
2	4.3	4.8	4
Sum=4			Sum=12

U=max(4,12)=12

Wilcoxon Rank Sum and Mann-Whitney U Test

Linearly related tests

$$W_2 = \frac{1}{2}n(n+1) + U$$

Check it:

$$22 = \frac{1}{2}4(5) + 12$$

```r
dat<-data.frame(scores=c(3.6,3.9,4.0,4.3,3.8,4.1,4.5,4.8),
                brand=c(1,1,1,1,2,2,2,2))
dat$rank<-rank(dat$scores)
W2<-sum(dat$rank[dat$brand==2])
W2
```

```
## [1] 22
```

```r
set.seed(1234)
```

```r
nsims<-10000
rankSumPerms<-rep(NA,nsims)
for (i in 1:nsims){
  rankSumPerms[i]<- sum(sample(1:8,4,replace=FALSE))
  }
#pvalue
sum(rankSumPerms>=W2)/nsims
```

```
## [1] 0.1739
```

Tests for equality of scale parameters

- Siegel-Tukey
- Ansari-Bradley

Siegel-Tukey

- Arrange observations from smallest to largest.
- Assign 1 to the smallest observation, 2 to the **largest** observation, 3 to the next largest, 4 to the next smallest observtion, etc.
- Perform the Wilcoxon rank-sum test. Smaller ranks are associated with the group with larger variability.
- Problem is why start the ranking with the smallest? Could just as easily start with the largest. (Ansari-Bradley!)

Ansari-Bradley

- Arrange observations from smallest to largest.
- Assign 1 to the smallest and largest observations, 2 to the next smallest and next largest, etc.
- Compute the Wilcoxon rank-sum and compute a p-value. (Can't use a traditional Wilcoxon rank-sum table. But we shouldn't be using tables anyway! It's 2016!)

```r
dat[c(1,2,6,7),]
```

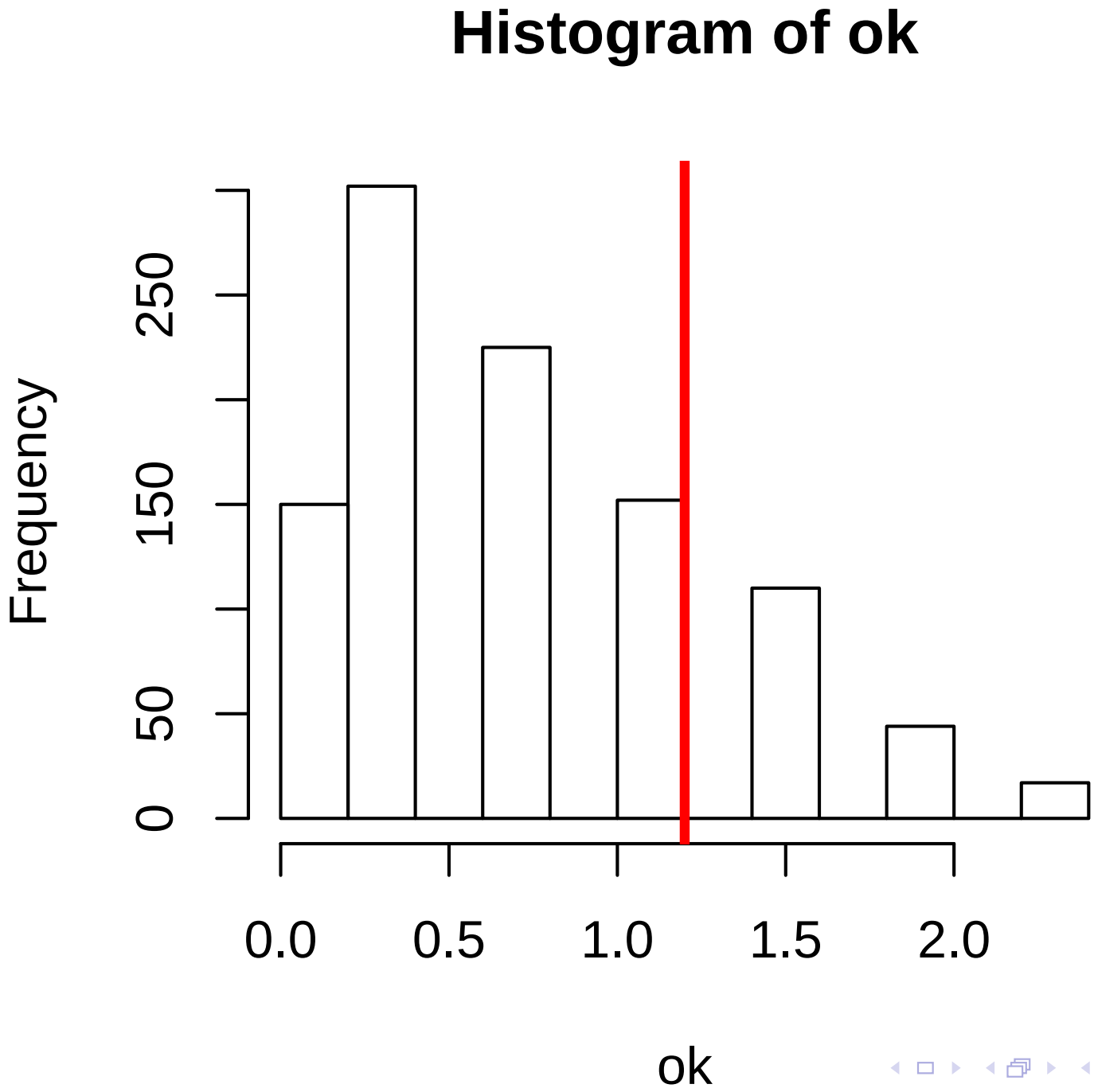
```
##       scores trt
## 1   6.378803   1
## 2  10.832288   1
## 6  12.530279   2
## 7   7.126300   2
```

```r
#Some code for Ansari Bradley
dat<-dat[order(dat$scores),]
dat$ranks<-c(1:(dim(dat)[1]/2),(dim(dat)[1]/2):1)
abs(diff(tapply(dat$ranks,dat$trt,mean)))
```

```
##   2
## 1.2
```

```r
#Now simulate a null distribution.
#Some code for Ansari Bradley
nsim<-1000
ok<-rep(NA,nsim)
for (i in 1:nsim){print(i)
dat<-dat[sample(1:10,10,replace=FALSE),]
dat$ranks<-c(1:(dim(dat)[1]/2),(dim(dat)[1]/2):1)
ok[i]<-abs(diff((tapply(dat$ranks,dat$trt,mean))))
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
```

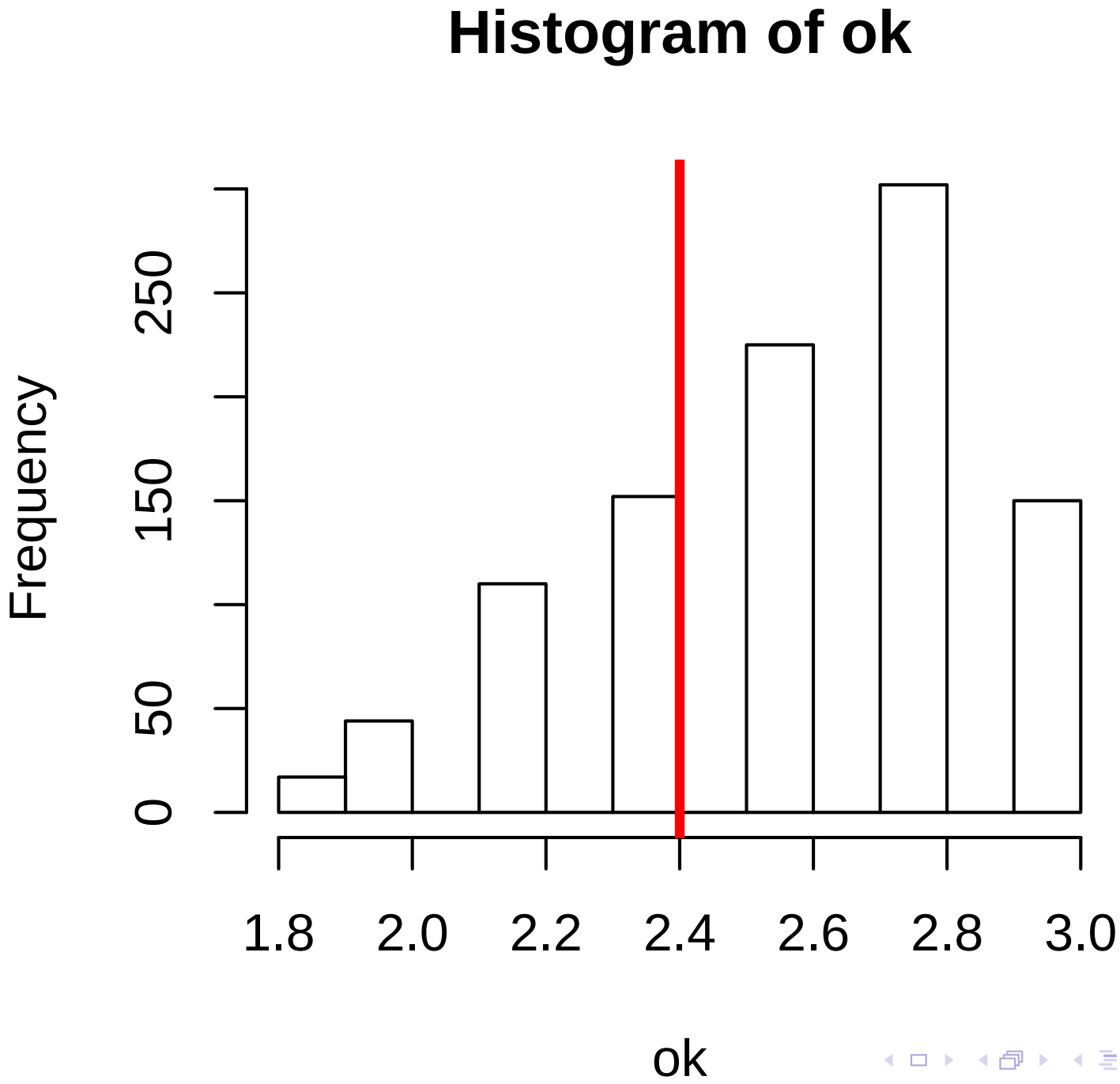

Histogram of ok
Frequency
0
50
150
250
0.0
0.5
1.0
1.5
2.0
ok

```r
#P-value for this test:
sum(ok>=1.2)/nsim

## [1] 0.323
```

```r
#let's try it again, but with a small change
#Some code for Ansari Bradley
dat<-dat[order(dat$scores),]
dat$ranks<-c(1:(dim(dat)[1]/2),(dim(dat)[1]/2):1)
min(tapply(dat$ranks,dat$trt,mean))

## [1] 2.4
```

```r
#Now simulate a null distribution.
#Some code for Ansari Bradley
nsim<-1000
ok<-rep(NA,nsim)
for (i in 1:nsim){print(i)
dat<-dat[sample(1:10,10,replace=FALSE),]
dat$ranks<-c(1:(dim(dat)[1]/2),(dim(dat)[1]/2):1)
ok[i]<-min((tapply(dat$ranks,dat$trt,mean)))
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
```


Histogram of ok
Frequency
0
50
150
250
1.8
2.0
2.2
2.4
2.6
2.8
3.0
ok

```r
#P-value for this test:
sum(ok<=2.4)/nsim

## [1] 0.323
```