Worksheet 10 Solutions

Keith Uzelmann

27/11/2021

Question 1

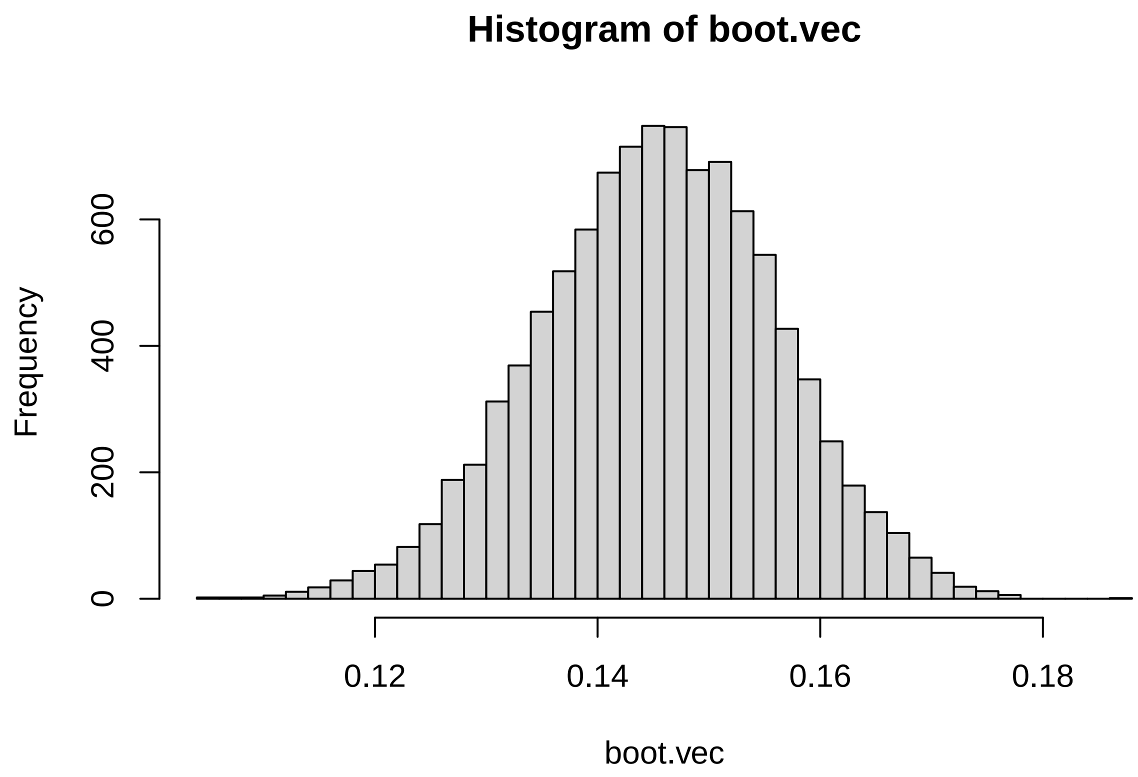
Part A First, we obtain a bootstrap distribution for S.

```r
Response_Times <- read.csv("~/R_Datasets/Response_Times.csv")
data = Response_Times$x

B = 10000
boot.vec = c()
for(b in 1:B)
{
  x.boot = sample(data, length(data), replace = TRUE)
  boot.vec[b] = sd(x.boot)
}
```

Our approximation of the sampling distribution of S is

```r
hist(boot.vec, breaks = 50)
```



Our approximation of the bias of S is

```
boot.bias = mean(boot.vec) - sd(data)
boot.bias
```

```
## [1] -0.0005922634
```

Thus, our bias-corrected estimate of σ is

```
sd(data) - boot.bias
```

```
## [1] 0.1465882
```

Our estimate of the variance of S is

```
var(boot.vec)
```

```
## [1] 0.0001137079
```

And our 95% confidence interval for σ is

```
quantile(boot.vec, c(0.025, 0.975))
```

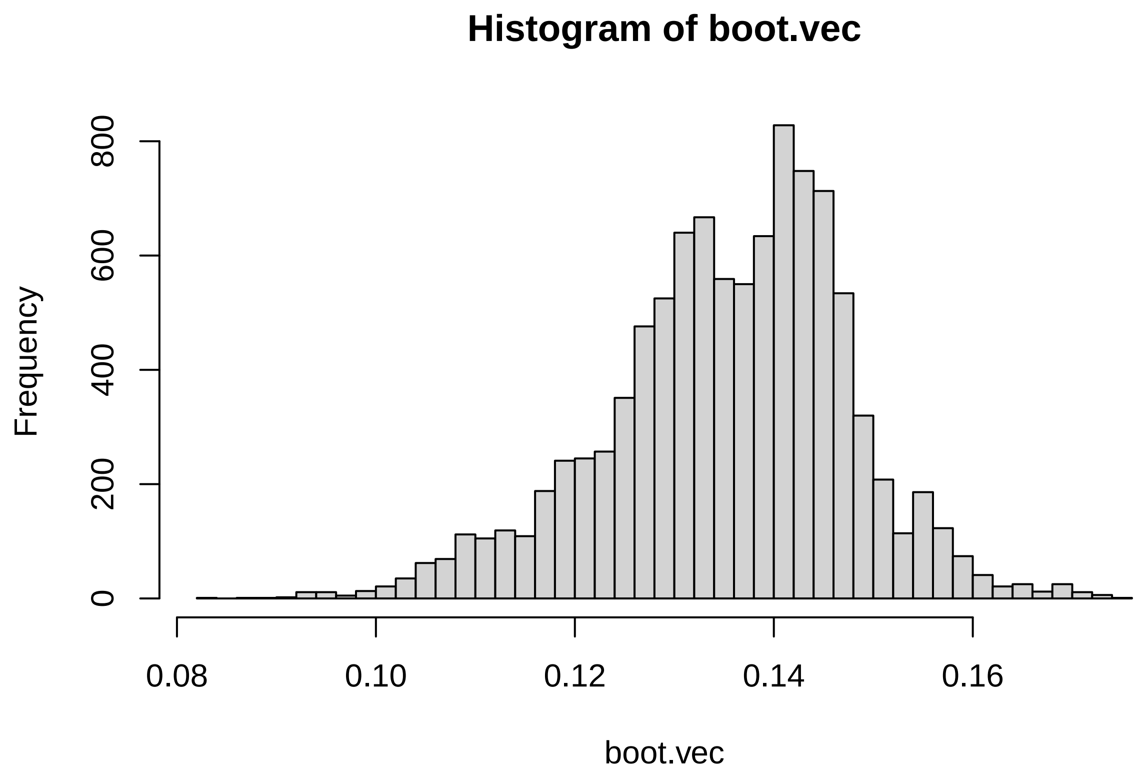
```
##      2.5%     97.5%
## 0.1240384 0.1659491
```

Part B First, we obtain a bootstrap distribution for H.

```r
B = 10000
boot.vec = c()
for(b in 1:B)
{
  x.boot = sample(data, length(data), replace = TRUE)
  boot.vec[b] = IQR(x.boot)
}
```

Our approximation of the sampling distribution of H is

```r
hist(boot.vec, breaks = 50)
```



Our approximation of the bias of H is

```r
boot.bias = mean(boot.vec) - IQR(data)
boot.bias
```

```
## [1] -0.005360125
```

Thus, our bias-corrected estimate of the population IQR is

```
IQR(data) - boot.bias
```

```
## [1] 0.1463601
```

Our estimate of the variance of H is

```
var(boot.vec)
```

```
## [1] 0.0001479169
```

And our 95% confidence interval for σ is

```
quantile(boot.vec, c(0.025, 0.975))
```

```
##      2.5%     97.5%
## 0.1084937 0.1577500
```

Question 2

Part A First, we obtain a bootstrap distribution for $\tilde{X}$.

```
set.seed(2)
data = rexp(40, 0.5)

B = 10000
boot.vec = c()
for(b in 1:B)
{
  x.boot = sample(data, length(data), replace = TRUE)
  boot.vec[b] = median(x.boot)
}

boot.bias = mean(boot.vec) - median(data)
boot.bias
```

```
## [1] 0.0263861
```

Thus, our bias-corrected estimate of θ is

```
estimator.corrected = median(data) - boot.bias
estimator.corrected
```

```
## [1] 1.457583
```

And our 95% confidence interval for θ is

```
quantile(boot.vec, c(0.025, 0.975))
```

```
##     2.5%    97.5%
## 1.192142 2.156603
```

Part B The average bias is

```
theta.est = c()
for(i in 1:100000)
{
  x = rexp(40, 0.5)
  theta.est[i] = median(x)
}

average.bias = mean(theta.est) - log(2)/0.5
average.bias
```

```
## [1] 0.02261892
```

Part C The bias of our corrected estimator is

```
bias.corrected = estimator.corrected - log(2)/0.5
bias.corrected
```

```
## [1] 0.07128869
```

The bias of our uncorrected estimator is

```
bias.uncorrected = median(data) - log(2)/0.5
bias.uncorrected
```

```
## [1] 0.09767479
```

So we have removed some of the bias.

Part D We want to calculate the bias-corrected estimator for 1000 different samples, and look at the bias of that estimator.

```
estimator.corrected = c()
for(i in 1:1000)
{
  data = rexp(40, 0.5)
  boot.vec = c()
  for(b in 1:10000)
  {
    x.boot = sample(data, length(data), replace = TRUE)
    boot.vec[b] = median(x.boot)
  }
  boot.bias = mean(boot.vec) - median(data)
  estimator.corrected[i] = median(data) - boot.bias
}

mean(estimator.corrected) - log(2)/0.5
```

```
## [1] 0.005605133
```

Recall that the bias of our uncorrected estimator is

```
average.bias
```

```
## [1] 0.02261892
```

So this procedure does remove bias, on average.

Question 3

Part A We are testing the hypotheses

$$H_0 : \mu_M = \mu_F \quad \text{vs} \quad H_a : \mu_M \neq \mu_F$$

We will use the unequal-variances t-statistic.

Below we use permutation to determine the distribution of the test statistic.

```
React_85 <- read.csv("~/R_Datasets/React_85.csv")
x.og = React_85$Reaction_time[React_85$Gender == "Male"]
y.og = React_85$Reaction_time[React_85$Gender == "Female"]
xy.og = c(x.og, y.og)
nx = length(x.og)
ny = length(y.og)
t.cal = (mean(x.og) - mean(y.og))/sqrt(var(x.og)/nx + var(y.og)/ny)

t.stat = c()
for(i in 1:10000)
{
  xy.perm = sample(xy.og, nx + ny, replace = FALSE)
  x.perm = xy.perm[1:nx]
  y.perm = xy.perm[(nx + 1):(nx + ny)]
  t.stat[i] = (mean(x.perm) - mean(y.perm))/sqrt(var(x.perm)/nx + var(y.perm)/ny)
}
```

The resultant P-value is…

```
mean(abs(t.stat) > abs(t.cal))
```

```
## [1] 0.2832
```

This P-value is above 0.05, so we reject H_0. I.e., our evidence in support of the claim that men and women have different reaction times is insignificant.

Part B We will repeat the above test, but with other test statistics.

First, we will try with an equal-variances t-statistic.

```
s.pooled = sqrt(((nx - 1)*var(x.og) + (ny - 1)*var(y.og))/(nx + ny - 2))
t.cal = (mean(x.og) - mean(y.og))/sqrt(s.pooled^2 / nx + s.pooled^2 / ny)

t.stat = c()
for(i in 1:10000)
{
  xy.perm = sample(xy.og, nx + ny, replace = FALSE)
  x.perm = xy.perm[1:nx]
  y.perm = xy.perm[(nx + 1):(nx + ny)]
  s.pooled = sqrt(((nx - 1)*var(x.perm) + (ny - 1)*var(y.perm))/(nx + ny - 2))
  t.stat[i] = (mean(x.perm) - mean(y.perm))/sqrt(s.pooled^2 / nx + s.pooled^2 / ny)
}
```

The resultant P-value is...

```
mean(abs(t.stat) > abs(t.cal))
```

```
## [1] 0.2853
```

This P-value is almost identical.

Next, we will try with a simple difference of means.

Below we use permutation to determine the distribution of the test statistic.

```
t.cal = mean(x.og) - mean(y.og)

t.stat = c()
for(i in 1:10000)
{
  xy.perm = sample(xy.og, nx + ny, replace = FALSE)
  x.perm = xy.perm[1:nx]
  y.perm = xy.perm[(nx + 1):(nx + ny)]
  t.stat[i] = mean(x.perm) - mean(y.perm)
}
```

The resultant P-value is...

```
mean(abs(t.stat) > abs(t.cal))
```

```
## [1] 0.282
```

Again, this P-value is almost identical.

(a) Since we will be repeating this procedure many times, it will be easiest if we write a general function to handle these test. Then, we can simply provide alternative parameters to the function instead of rewriting the code each time. This function will take in the sample sizes, the means, and the standard deviation as parameters, and return a P-value for a right-tailed test.

```r
Ptest.right = function(n, m, my.sigma, mu1, mu2, I = 10000)
{
  x1 = rnorm(n, mu1, my.sigma)
  x2 = rnorm(m, mu2, my.sigma)
  s.pooled.true = sqrt(((n - 1)*var(x1) + (m - 1)*var(x2))/(n + m - 2))
  tstat.true = (mean(x1) - mean(x2))/(s.pooled.true*sqrt(1/n + 1/m))
  tstat.vec = c()
  for(i in 1:I)
  {
    x.shuffled = sample(c(x1, x2), n + m, replace = FALSE)
    x1.shuffled = x.shuffled[(1:n)]
    x2.shuffled = x.shuffled[-(1:n)]
    s.pooled.shuffled = sqrt(((n - 1)*var(x1.shuffled) + (m - 1)*var(x2.shuffled))/(n + m - 2))
    tstat.vec[i] = (mean(x1.shuffled) - mean(x2.shuffled))/(s.pooled.shuffled*sqrt(1/n + 1/m))
  }
  return(mean(tstat.vec > tstat.true))
}
```

Let's run this test once. I will set the seed number to 1 so that the test results are reproducible.

```r
set.seed(1)
Ptest.right(15, 15, 4, 0, 0)
```

```
## [1] 0.4544
```

The P-value is 0.4544, so we fail to reject H_0.

(b) We can just run our function from before with different parameters.

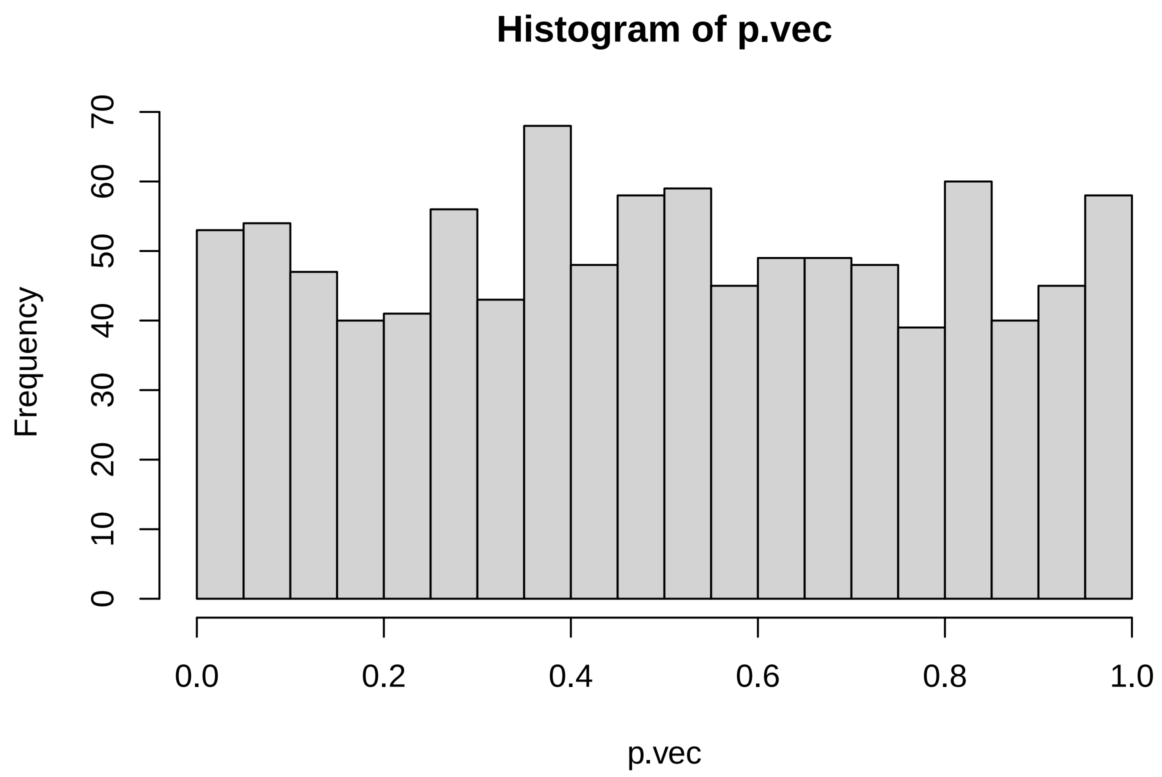
```r
set.seed(1)
Ptest.right(15, 15, 4, 1, 0)
```

```
## [1] 0.2069
```

The P-value is 0.2069, so we fail to reject H_0. Note that this is a Type II error.

(c) Since we are looping this 1000 times, there's no longer a need to fix a seed. However, this will run quite slow if we loop 10000 times per test, so we will change I to 1000.

```r
p.vec = c()
for(i in 1:1000)
{
  p.vec[i] = Ptest.right(15, 15, 4, 0, 0, I = 1000)
}
hist(p.vec, breaks = 20)
```



Yes, this distribution looks to be uniform.

(d) First we run the test 1000 times.

```
p.vec = c()
for(i in 1:1000)
{
  p.vec[i] = Ptest.right(15, 15, 4, 1, 0, I = 1000)
}
```

This leads a rejection of H_0 the following amount:

```
mean(p.vec < 0.05)
```

```
## [1] 0.169
```

This is the power of the test (for this effect size).

(e) Let's change I to 500, just so things run faster.

```
p.vec25 = c()
p.vec50 = c()
p.vec100 = c()
p.vec200 = c()
```

```r
for(i in 1:1000)
{
  p.vec25[i] = Ptest.right(25, 25, 4, 1, 0, I = 500)
  p.vec50[i] = Ptest.right(50, 50, 4, 1, 0, I = 500)
  p.vec100[i] = Ptest.right(100, 100, 4, 1, 0, I = 500)
  p.vec200[i] = Ptest.right(200, 200, 4, 1, 0, I = 500)
}
```

The powers for each sample size (at this effect size) are

```r
mean(p.vec25 < 0.05)
```

```
## [1] 0.196
```

```r
mean(p.vec50 < 0.05)
```

```
## [1] 0.303
```

```r
mean(p.vec100 < 0.05)
```

```
## [1] 0.555
```

```r
mean(p.vec200 < 0.05)
```

```
## [1] 0.803
```

It seems like $n = m = 200$ is around the minimum size to get a power of at least 80%.