W203 Supplementary Exercise 2

Mohammad Jawad Habib

February 21, 2016

```
require(knitr)
```

```
## Loading required package: knitr
```

1. Write a p-value calculator

1. Write a function in R that does the following: Takes sample data (e.g., as a vector), mean of the null hypothesis, population standard deviation, a Boolean variable indicating whether we want a one-tailed or two-tailed test, and another Boolean variable indicating which tail (left or right) should be taken in case of a one-tailed test. Returns p-values for the required test (Null is "population mean is the one that the function is fed"), as well as a Boolean value showing whether the test passes a conventional 5% level or not.

```
myfun <- function(data, mu, pop.sd, two.tailed=TRUE, left.tail=NULL) {
  # we only want to deal with a numeric vector
  if(!is.numeric(data)) stop("'data' must be a numeric vector \n")

  # we expect 'left.tail' to be TRUE or FALSE for one-tailed test
  if(!two.tailed & is.null(left.tail))
    stop("'left.tail' must be TRUE or FALSE for one tailed test \n")

  # calculate z-score
  z.raw <- (mean(data) - mu) / (pop.sd/(sqrt(length(data))))

  # convert p-value to two-tailed if applicable
  if(two.tailed) {
    p.val <- 2*pnorm(-abs(z.raw), lower.tail = TRUE)
    rejectNull <- ifelse(p.val < 0.025, TRUE, FALSE)
  } else {
    if(left.tail == TRUE) { # left-tailed
      # take the area to the left of z.raw
      p.val <- pnorm(z.raw, lower.tail = TRUE)
    } else { # right-tailed
      # take the area to the right of z.raw
      p.val <- pnorm(z.raw, lower.tail = FALSE)
    }
    rejectNull <- ifelse(p.val < 0.05, TRUE, FALSE)
  }

  list("p.value" = p.val, "rejectNull" = rejectNull)
}
```

Seed the RNG for repeatable results.

```r
set.seed(5000)
```

2. Test p-value calculator

2. Choose a sample size and then use `rnorm(n, mean, sd)` to generate a random sample and test your function.

```r
pop.mean <- 50
pop.sd <- 21
sample.size <- 36
sample.data <- rnorm(sample.size, mean = pop.mean, sd = pop.sd)

mean(sample.data)
```

```
## [1] 49.19029
```

```r
# run a right tailed test
kable(data.frame(myfun(sample.data, mu = pop.mean, pop.sd = pop.sd, two.tailed = FALSE,
                       left.tail = FALSE)))
```

p.value	rejectNull
0.5914767	FALSE

```r
# run a left-tailed test
kable(data.frame(myfun(sample.data, mu = pop.mean, pop.sd = pop.sd, two.tailed = FALSE,
                       left.tail = TRUE)))
```

p.value	rejectNull
0.4085233	FALSE

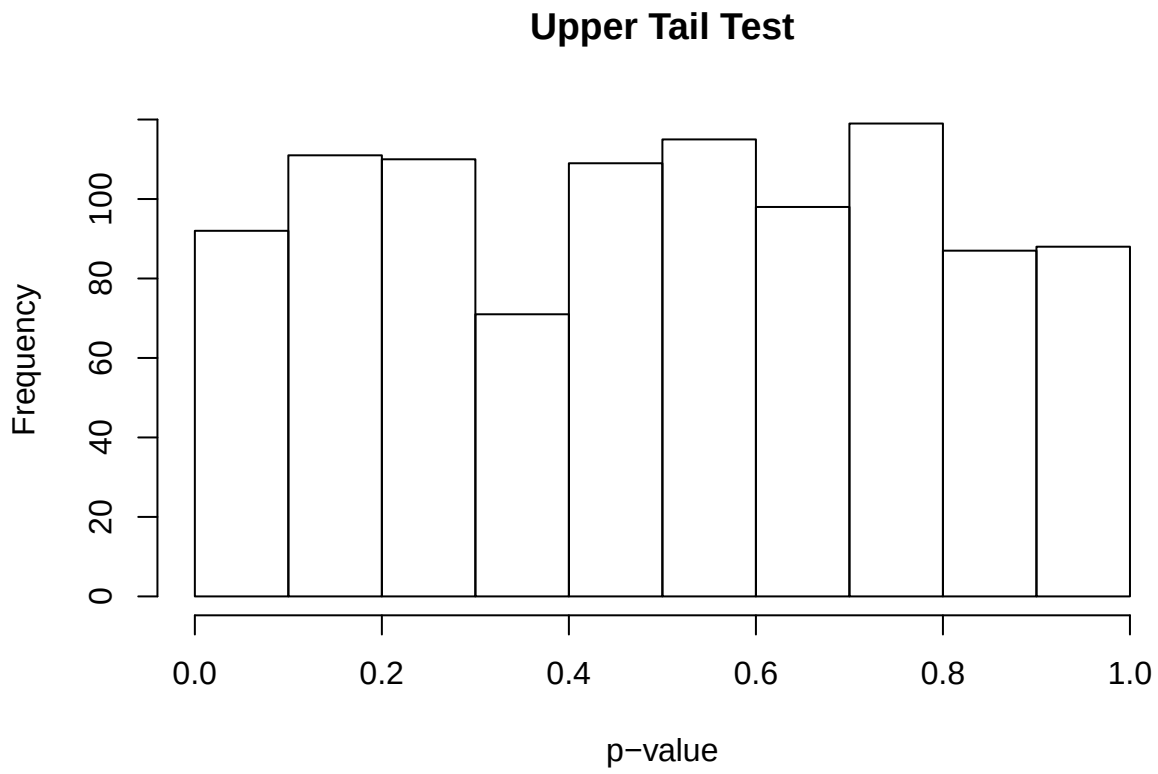
```r
# run a two-tailed test
kable(data.frame(myfun(sample.data, mu = pop.mean, pop.sd = pop.sd, two.tailed = TRUE,
                       left.tail = NULL)))
```

p.value	rejectNull
0.8170466	FALSE

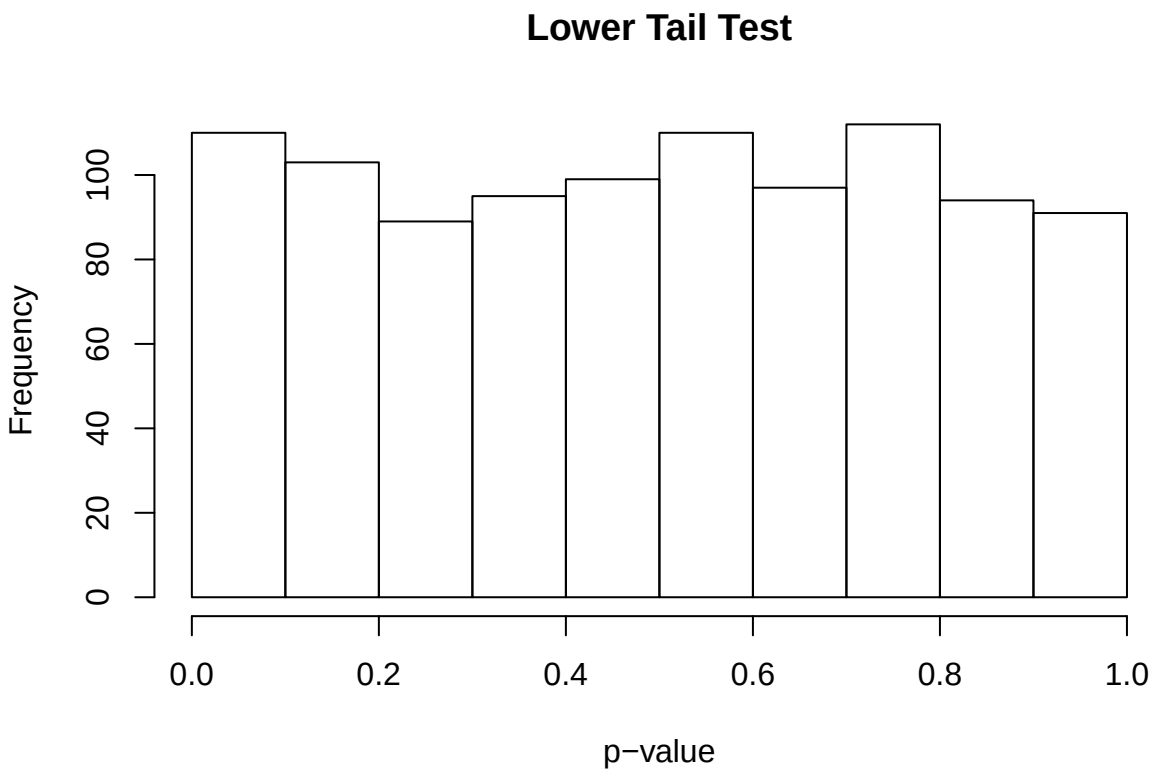
3. Repeat p-value calculator test multiple times

3. Use replicate or sapply (or replicate) to generate a sample and do the test multiple times (say 1000). Plot the histogram of p-values that you are getting when the Null is true. Is your function calculating Type-I errors correctly?

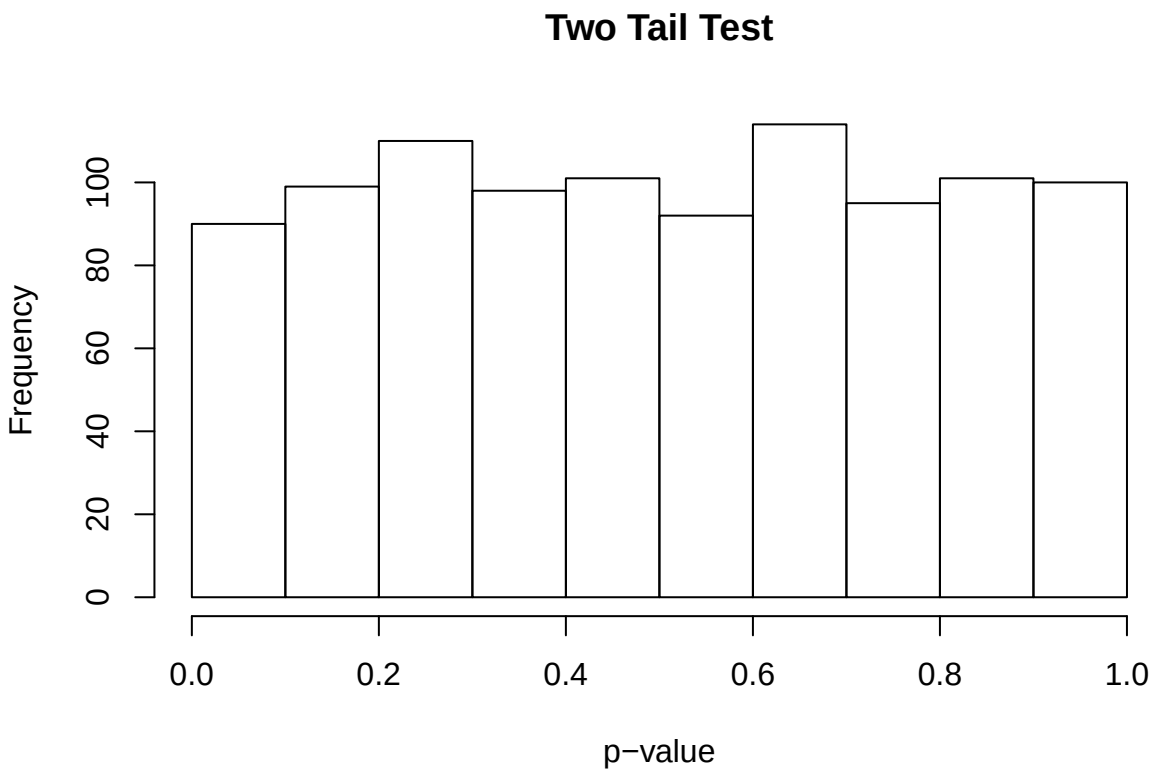
```r
num.trials <- 1000

result.vec.right.tailed <- replicate(num.trials,
                                     myfun(rnorm(sample.size, mean = pop.mean, sd = pop.sd),
                                           mu = pop.mean,
                                           pop.sd = pop.sd,
                                           two.tailed = FALSE,
                                           left.tail = FALSE)$p.value,
                                     simplify = TRUE)
hist(result.vec.right.tailed, main = "Upper Tail Test", xlab = "p-value")
```




```r
result.vec.left.tailed <- replicate(num.trials,
                                    myfun(rnorm(sample.size, mean = pop.mean, sd = pop.sd),
                                          mu = pop.mean,
                                          pop.sd = pop.sd,
                                          two.tailed = FALSE,
                                          left.tail = TRUE)$p.value,
                                    simplify = TRUE)
hist(result.vec.left.tailed, main = "Lower Tail Test", xlab = "p-value")
```




```
result.vec.two.tailed <- replicate(num.trials,
                                   myfun(rnorm(sample.size, mean = pop.mean, sd = pop.sd),
                                         mu = pop.mean,
                                         pop.sd = pop.sd,
                                         two.tailed = TRUE,
                                         left.tail = NULL)$p.value,
                                   simplify = TRUE)
hist(result.vec.two.tailed, main = "Two Tail Test", xlab = "p-value")
```



We can test below whether we are returning `TRUE` or `FALSE` correctly from `myfun`.

```
result.vec.right.tailed <- replicate(num.trials,
                                     myfun(rnorm(sample.size, mean = pop.mean, sd = pop.sd),
                                           mu = pop.mean,
                                           pop.sd = pop.sd,
                                           two.tailed = FALSE,
                                           left.tail = FALSE),
                                     simplify = TRUE)
df <- data.frame(cbind(result.vec.right.tailed[1,], result.vec.right.tailed[2,]))
colnames(df) <- c("p.value", "rejectNull")
kable(head(df))
```

p.value	rejectNull
0.9486921	FALSE
0.9826398	FALSE
0.7462191	FALSE
0.5525445	FALSE
0.9985539	FALSE
0.9829156	FALSE

4. Calculate Type II error

4. Now assume your Null is false. Note: For type-II error calculation, you need a specific assumption about the mean of the population from which the sample is taken. You can assume that this mean is one tenth of one standard deviation above the Null mean. Calculate type-II errors both theoretically and by simulation as in step 3.

Theoretical Type II error calculation

`sample.data` represents our test vector with population mean of `pop.mean` and population sd of `pop.sd`. We assume that `mean(sample.data)` is our null hypothesis in this case.

The questions specifies a mean value (we will call this `rejection.mean`).

Perform a right-tailed test

For a right-tailed test, any values above this `rejection.mean` will cause us to reject the null hypothesis.

```
rejection.mean <- pop.mean + pop.sd/10
rejection.mean
```

```
## [1] 52.1
```

For original population with mean pop.mean, rejection.mean represents a z-score of `z.pop` given below.

```
z.pop <- (rejection.mean - pop.mean) /
  (pop.sd/sqrt(length(sample.data)))
z.pop
```

```
## [1] 0.6
```

Assuming that Null `pop.mean` is False, we take `mean(sample.data)` as our Null hypothesis That is, we use a distribution with `mean(sample.data)`.

For Type I error, we will use the original population distribution where Null is given by `pop.mean`. Type I error is the area to the right of `rejection.mean` on the original population distribution for a right-tailed test.

```
z.type.one.right <- (rejection.mean - pop.mean) /
  (pop.sd/sqrt(length(sample.data)))
z.type.one.right
```

```
## [1] 0.6
```

```
p.type.one.right <- pnorm(z.type.one.right, lower.tail = FALSE)
p.type.one.right
```

```
## [1] 0.2742531
```

Type II error is the area to the left of `rejection.mean` on the alternate distribution with mean equal to `mean(sample.data)`.

```
z.type.two.right <- (rejection.mean - mean(sample.data)) /
  (pop.sd/sqrt(length(sample.data)))
z.type.two.right
```

```
## [1] 0.8313453
```

```
# For Type II, we want the area to the left of z.type.two.right on alternate distribution
p.type.two.right <- pnorm(z.type.two.right, lower.tail = TRUE)
p.type.two.right
```

```
## [1] 0.7971107
```

```
test.power.right <- 1 - p.type.two.right
test.power.right
```

```
## [1] 0.2028893
```

For a right-tailed test, there is a probability given by `p.type.two.right` that we will fail to reject the null hypothesis when we should. This is our Type II error in this case.

Perform a left-tailed test

In a left-tailed test, Type I error is area to the left of `rejection.mean` on the original distribution with mean `pop.mean`.

```
z.type.one.left <- (rejection.mean - mean(sample.data)) /
  (pop.sd/sqrt(length(sample.data)))
z.type.one.left
```

```
## [1] 0.8313453
```

```
p.type.one.left <- pnorm(z.type.one.left, lower.tail = TRUE)
p.type.one.left
```

```
## [1] 0.7971107
```

In a left tailed test, Type II error is given by the area to the right of `rejection.mean` on the alternate distribution with mean equal to `mean(sample.data)`.

```
# z-value in this case is the same as z.type.two.right but let's calculate it anyways
z.type.two.left <- (rejection.mean - mean(sample.data)) /
  (pop.sd/sqrt(length(sample.data)))
z.type.two.left
```

```
## [1] 0.8313453
```

```r
# For Type II, this time we want the area to the right of z.type.two.left on alternate distribution
p.type.two.left <- pnorm(z.type.two.left, lower.tail = FALSE)
p.type.two.left
```

```
## [1] 0.2028893
```

```r
test.power.left <- 1 - p.type.two.left
test.power.left
```

```
## [1] 0.7971107
```

For a left-tailed test, there is a probability given by `p.type.two.left` that we will fail to reject the null hypothesis when we should. This is our Type II error in this case.

Perform a two-tailed test

In a two-tailed test, rejection region is to the left of -alpha/2 and to the right of alpha/2 on the original distribution.

```r
z.original <- (rejection.mean - pop.mean) / (pop.sd/(sqrt(length(sample.data))))
z.original
```

```
## [1] 0.6
```

```r
# Type I error is the same as the value given below
p.type.one.twotailed <- pnorm(z.original, lower.tail=FALSE)
p.type.one.twotailed
```

```
## [1] 0.2742531
```

```r
alpha.by.two <- p.type.one.twotailed/2
alpha.by.two
```

```
## [1] 0.1371266
```

```r
z.alpha.by.two.pos <- qnorm(alpha.by.two, lower.tail = FALSE)
z.alpha.by.two.pos
```

```
## [1] 1.09332
```

```r
rejection.mean.twotailed.pos <- pop.mean +
  z.alpha.by.two.pos*(pop.sd/sqrt(length(sample.data)))
rejection.mean.twotailed.pos
```

```
## [1] 53.82662
```

```
z.alpha.by.two.neg <- -z.alpha.by.two.pos

rejection.mean.twotailed.neg <- pop.mean +
  z.alpha.by.two.neg*(pop.sd/sqrt(length(sample.data)))
rejection.mean.twotailed.neg
```

```
## [1] 46.17338
```

To calculate Type II error, we will first get the z-scores for our `rejection.mean.twotailed.pos` and `rejection.mean.twotailed.neg` on the alternate distribution. Then we will use `pnorm` to get the regions to the right and left of the positive and negative z-scores on the alternate distribution. We will then calculate Type II error by subtracting those two values from 1.

```
# For the positive rejection.mean.two.tailed
z.twotailed.positive <- (rejection.mean.twotailed.pos - mean(sample.data)) /
  (pop.sd/sqrt(length(sample.data)))
z.twotailed.positive
```

```
## [1] 1.324666
```

```
p.twotailed.positive <- pnorm(z.twotailed.positive, lower.tail = FALSE)
p.twotailed.positive
```

```
## [1] 0.09264102
```

```
z.twotailed.negative <- (rejection.mean.twotailed.neg - mean(sample.data)) /
  (pop.sd/sqrt(length(sample.data)))
z.twotailed.negative
```

```
## [1] -0.8619752
```

```
p.twotailed.negative <- pnorm(z.twotailed.negative, lower.tail = TRUE)
p.twotailed.negative
```

```
## [1] 0.1943506
```

Finally, we can calculate our Type II error for a two-tailed test using the alternative distribution and the power of test.

```
p.type.two.twotailed <- 1 - p.twotailed.positive - p.twotailed.negative
p.type.two.twotailed
```

```
## [1] 0.7130084
```

```
test.power.twotailed <- 1 - p.type.two.twotailed
test.power.twotailed
```

```
## [1] 0.2869916
```

Numeric Type II error calculation

Let's hack together a crude function that let's us calculate Type II error. When calling this function leave the default value of `sample.size` for a vector. You only need to specify `sample.size` when your `test.data` consists of a single value. This is so we can pass a randomly changing vector of `test.data` to this function using `replicate`.

```
TypeTwoError <- function(test.data, null.mean, null.sd, alpha,
                         sample.size = NULL, tail.type = "Two") {
  # return(a)

  if(is.null(sample.size))
    sample.size <- length(test.data)

  alternate.mean <- mean(test.data)

  std.err <- null.sd / sqrt(sample.size)

  tail.type <- casefold(tail.type, upper = TRUE)

  switch(tail.type,
         TWO = {
           type.one.vec <- c(alpha/2, 1-alpha/2)
           bounds.vec <- qnorm(type.one.vec, null.mean, std.err, lower.tail = TRUE)
           p.values.vec <- pnorm(bounds.vec,
                                 alternate.mean,
                                 std.err,
                                 lower.tail = TRUE)
           type.two.error <- diff(p.values.vec)
         },
         LEFT = {
           lower.bound <- qnorm(alpha, null.mean, std.err, lower.tail = TRUE)
           type.two.error <- pnorm(lower.bound,
                                   alternate.mean,
                                   std.err,
                                   lower.tail = FALSE)
           },
         RIGHT = {
           upper.bound <- qnorm(alpha, null.mean, std.err, lower.tail = FALSE)
           type.two.error <- pnorm(upper.bound,
                                   alternate.mean,
                                   std.err,
                                   lower.tail = FALSE)

         })
  list("tail.type" = tail.type, "type.two.error" = type.two.error)
}
```

Let's test our function by using our `sample.data`.

```
# Our function taks an alpha value so we convert rejection mean to that.
alpha <- pnorm((rejection.mean - pop.mean) / (pop.sd / sqrt(length(sample.data))),
               lower.tail = FALSE)
```

```
# Let's use a for-loop and hope the R gods won't smite us
# At least it reduce code bloat
res <- data.frame()
for(case in c("Right", "Left", "Two")) {
  res <- append(res, TypeTwoError(test.data = sample.data,
                                  null.mean = pop.mean,
                                  null.sd = pop.sd,
                                  alpha = alpha,
                                  tail.type = case))
}
res <- data.frame(res, check.names = FALSE)
kable(res)
```

tail.type	type.two.error	tail.type	type.two.error	tail.type	type.two.error
RIGHT	0.2028893	LEFT	0.6438074	TWO	0.7130084

Now that we have our function `TypeTwoError` working, let's assume we had a larger sample.

```
res <- data.frame()
for(case in c("Right", "Left", "Two")) {
  res <- append(res, TypeTwoError(test.data = sample.data,
                                  null.mean = pop.mean,
                                  null.sd = pop.sd,
                                  alpha = alpha,
                                  sample.size = 200,
                                  tail.type = case))
}
res <- data.frame(res, check.names = FALSE)
kable(res)
```

tail.type	type.two.error	tail.type	type.two.error	tail.type	type.two.error
RIGHT	0.1260453	LEFT	0.5218168	TWO	0.6575182

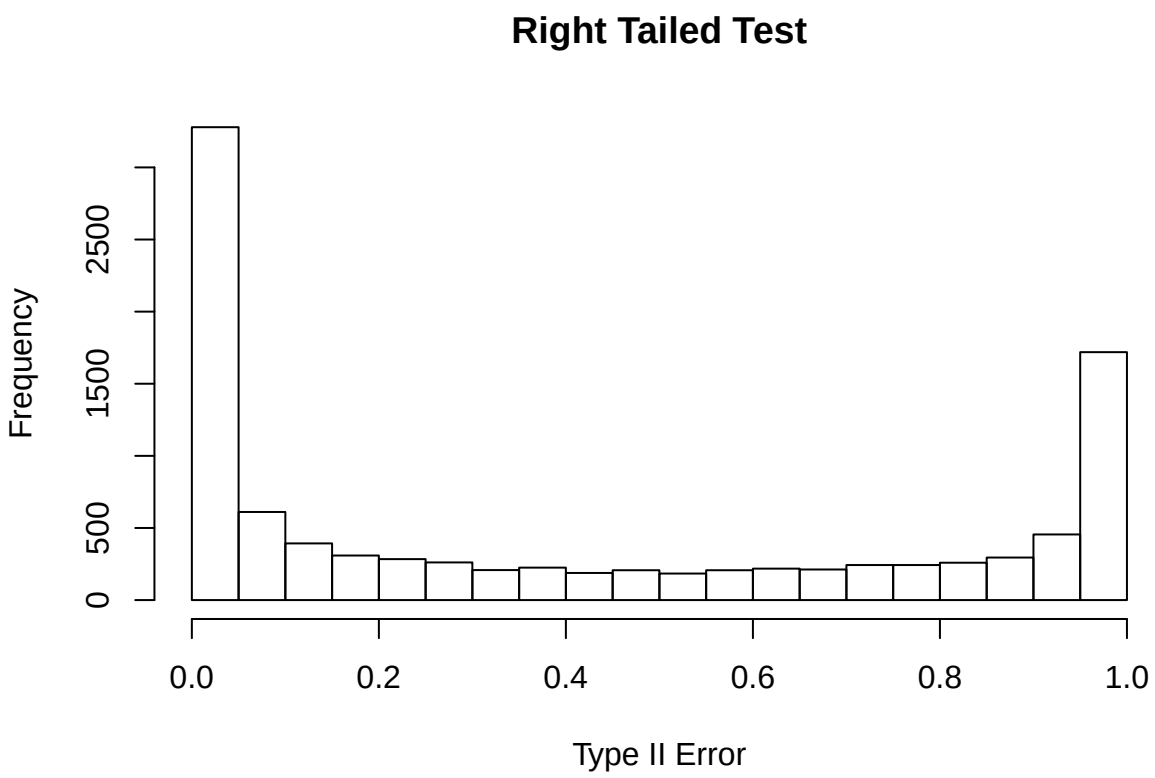
As shown in the results above, Type II error decreases as sample size increases. Consequently, power of test increases as Type II error decreases.

Repeat numeric Type II calculation multiple times

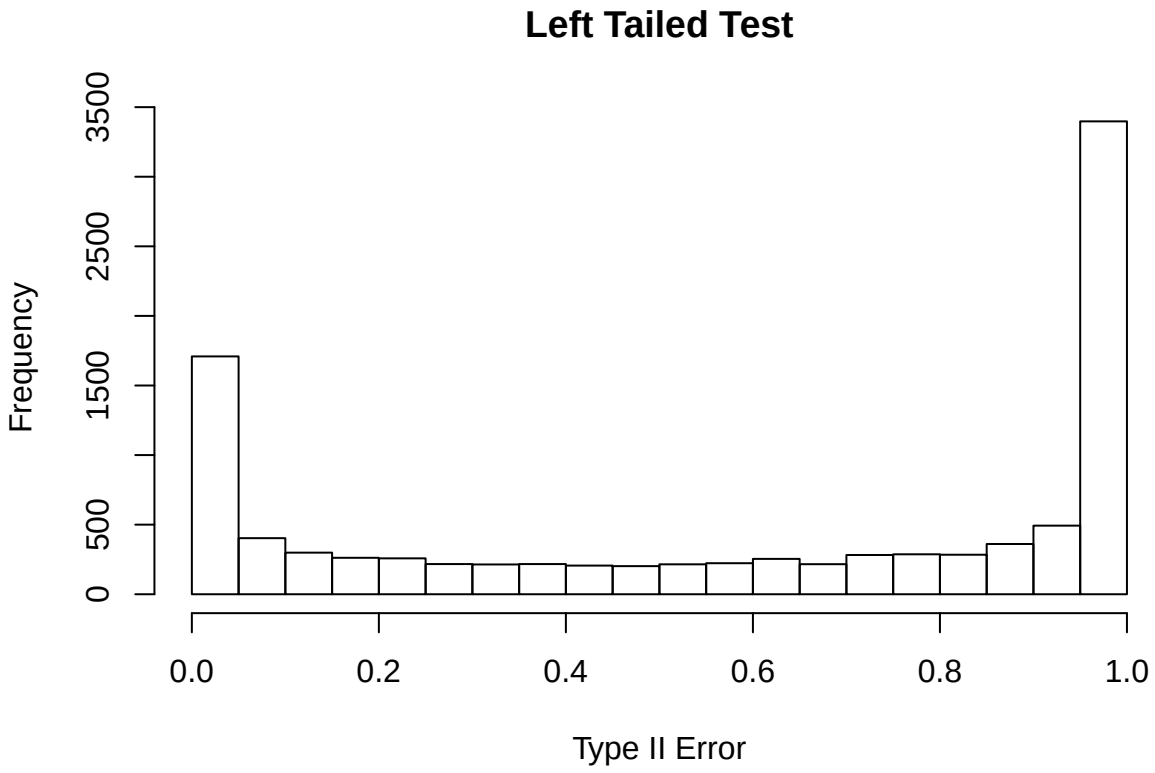
Finally, let's repeat the Type II error calculation a large number of times.

```
num.trials <- 10000

for(case in c("Right", "Left", "Two")) {
 results.vec <- replicate(num.trials,
                          TypeTwoError(test.data = rnorm(sample.size, mean = pop.mean,
                                                         sd = pop.sd),
                                       null.mean = pop.mean,
                                       null.sd = pop.sd,
```

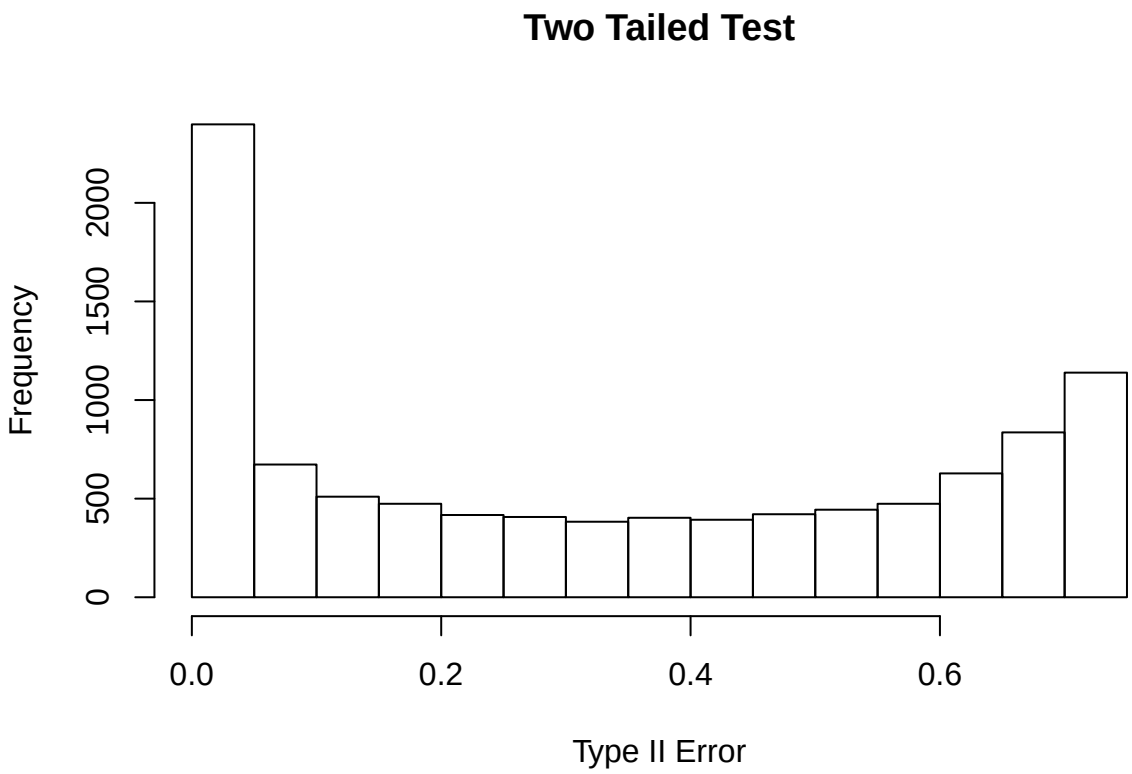
```
                                        alpha = alpha,
                                        sample.size = 200,
                                        tail.type = case)$type.two.error,
                            simplify = TRUE)
 hist(results.vec, main = paste0(case, " Tailed Test"), xlab = "Type II Error")
 print(summary(results.vec))
}
```




```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.01409 0.27300 0.40960 0.84500 1.00000
```




```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.1654  0.7178  0.5931  0.9882  1.0000
```




```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.05643 0.31480 0.33550 0.60910 0.72570
```