Tying Hypothesis Tests, Power, and More Together with Functions

General Homework Notes

- Read the entire question.
- Show your answers. Don't give me the code and assume it shows the answers (e.g., plots, output, etc)
- Please put spaces between blocks of code. One giant chunk of code with no line breaks is unreadable.
- Read and follow the Google or Hadley Wickham's style guide!
- Name your variables well so *YOU* don't confuse them

Review

- Statistical Distributions give us probabilities of discrete or ranges of values
- P-Values allow us to ask $P(x \leq Data|H)$
- We set alpha (often 0.05) to guard against mistakenly rejecting a null
- We obtain power of a test by simulating data and seeing the % of times we fail to reject a null

Today

1. Brief List Review
2. Functions
3. Z-Tests
4. Integrating Power, Z-Tests, and Functions

What is a list?

A list is an object with a key-value combination. Each slot in a list has a unique key and can contain anything.

```
newList <- list(a=1, b=rnorm(3))
```

```
newList$a

# [1] 1

newList$b

# [1]  1.2559 -0.4227 -1.0621
```

What is a list?

You can reference the name of an element in a list many ways

```
newList[["a"]]

# [1] 1

newList[[1]]

# [1] 1
```

```
newList

# $a
# [1] 1
#
# $b
# [1]  1.2559 -0.4227 -1.0621
```

What is a list?

Lists can even contain lists - it can get a little silly.

```
newList$foo<-list(bar = 13)

newList$foo$bar

# [1] 13
```

Functions!

What is a function?

Functions take some object(s) and use it to give us either a new object or perform an action.

```
sum(1:10)


# [1] 55
```

What is inside of that function

Functions take some object(s) and use it to give us either a new object or perform an action.

```
sum


# function (..., na.rm = FALSE)  .Primitive("sum")
```

What is inside of that function

`function(`*arguments*`)` **Code Block**

Example: addOne

```
addOne <- function(x) x+1
```

```
addOne(3)

# [1] 4
```

Default Values

```
addOne <- function(x = 0) x+1
```

```
addOne()

# [1] 1
```

More Hygenic Code: Code Blocks

```
addOne <- function(x = 0){
  x+1
}
```

Note that the last output is returned to the user.

More Hygenic Code: Return

```
addOne <- function(x = 0){
  return(x+1)
}
```

Exercise: Two Functions

1. Write a squaring function (i.e., square(3) = 9)
2. Write an add function that returns the sum of two numbers. If no numbers are supplied, it returns 0. If only one is supplied, it returns that number.

Exercise: Two Functions

```r
square <- function(x) x*x
```

```r
add <- function(x=0, y=0){
  return(x+y)
}
```

Functions for Repetitive Tasks With a Lot of Code

```r
sumFun <- function(aVec){
  #start with 0
  out <- 0

  #loop over the vector, adding
  #each element together
  for(i in aVec){
    out <- out + i
  }

  #return the result
  return(out)
}
```

... - the Garbage Collector

Don't you just hate how you need to make a vector for sum?
`sum(c(4,5,6,1,2,3))`

```
sumNoC <- function(...){
  #convert ... into a vector
  avec <- c(...)

  #NOW sum the vector
  sum(avec)
}
```

This may seem trivial, but it's a nice way to pass arguments between functions.

Exercise: Cummulative Vectors

Write a function that returns a list with the cummulative sum, product, and mean of a vector. Allow it to pass arguments to other functions (e.g., mean takes arguments to deal with NAs).

Exercise: Cummulative Vectors

```r
cumSumProdMean <- function(aVec, ...) {
  #get our sum and product vectors ready
  s <- rep(NA, length(aVec))
  s[1] <- aVec[1]
  m <- p <- s

  #now loop!
  for(i in 2:length(aVec)){
    s[i] <- s[i-1] + aVec[i]
    p[i] <- p[i-1] * aVec[i]
    m[i] <- mean(aVec[1:i], ...)
  }

  #return the results in a list
  return(list(sums = s, prod = p, mean=m))
}
```

Exercise: Cummulative Vectors

```r
cumSumProdMean(1:10)

# $sums
#  [1]  1  3  6 10 15 21 28 36 45 55
#
# $prod
#  [1]       1       2       6      24     120     720    5040
#  [8]   40320  362880 3628800
#
# $mean
#  [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5
```

Exercise: Cummulative Vectors

```
cumSumProdMean(c(1:5, NA, 7:10))

# $sums
#  [1]  1  3  6 10 15 NA NA NA NA NA
#
# $prod
#  [1]   1   2   6  24 120  NA  NA  NA  NA  NA
#
# $mean
#  [1] 1.0 1.5 2.0 2.5 3.0  NA  NA  NA  NA  NA
```

Exercise: Cummulative Vectors

```
cumSumProdMean(c(1:5, NA, 7:10), na.rm=T)

# $sums
#  [1]  1  3  6 10 15 NA NA NA NA NA
#
# $prod
#  [1]   1   2   6  24 120  NA  NA  NA  NA  NA
#
# $mean
#  [1] 1.000 1.500 2.000 2.500 3.000 3.000 3.667 4.286 4.875
# [10] 5.444
```

Functions in Action for Probability!

Is a mean different from 0?

Recall last week that we calculated p-values assuming we knew a population's standard deviation.

Often we want to know if a sample mean is different from 0. We know that an estimated mean from a large sample size is normally distributed, so...

Enter the Z-Test

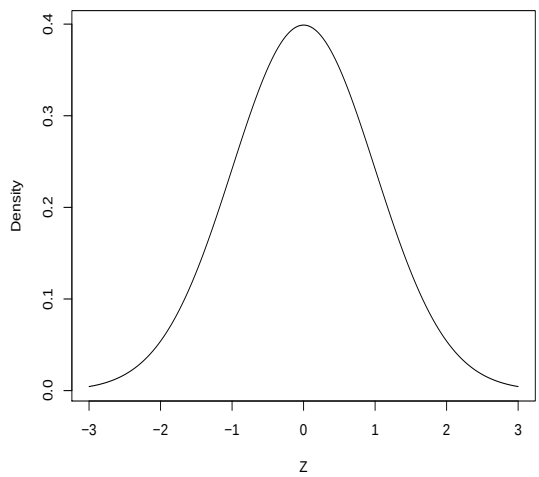
$$Z = \frac{\bar{Y} - \mu}{\sigma_{\bar{Y}}}$$

The Z-score compares a sample mean to an assumed population mean.

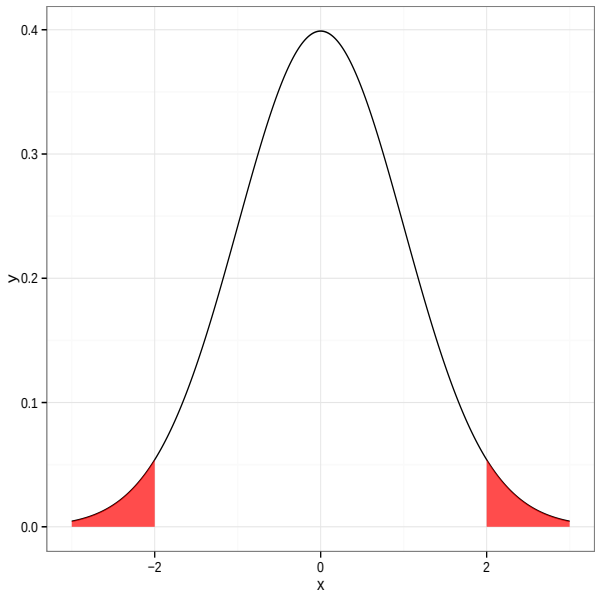
We call it a Z-Score because we correct by the standard error of the mean to compare to a standard deviation of the sample mean (SE).

Standard Normal (Z) Distribution

$$\text{Z-score} = \frac{Y_i - \bar{Y}}{\sigma}$$



Z-Test



Does Z fall into these tails?

A Simple P(z) Function

```
pz <- function(sample, mu = 0, tails=2) {
  #calculate z
  z <- (mean(sample) - mu)/(sd(sample)/sqrt(length(sample)))

  #return the two-tailed answer
  return(tails * pnorm(abs(z), lower.tail = F))
}
```

```
set.seed(697)
pz(rnorm(5000))

# [1] 0.6087

pz(rnorm(5, mean=1))

# [1] 0.0703
```

Exercise: Power of the P(z)

Write a function that both plots and gives you the power of the Z-Test for a variety of sample sizes assuming a given effect size (new mean)

- Assume a SD of 1
- Alpha, Number of sims per calculation, etc, can vary if you want
- Break this big task into smalller functions, when you can
- Challenge: Accomodate variation in both effect size and sample size.

A Power Wrapper Function

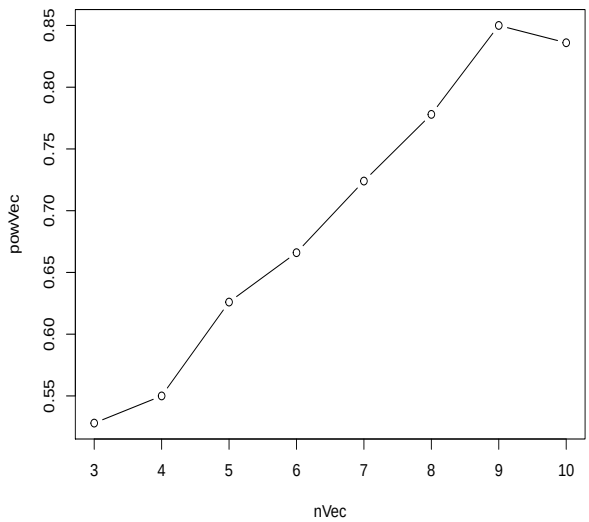
```
pzPowerSample <- function(nVec, effect=0,
                          n.sims=500, alpha=0.05, ...){

  #Create a vector for the output we're going to get
  powVec <- numeric(length(nVec))

  #loop over each sample size
  for(i in 1:length(nVec)){

    #What's this? Another new function!
    powVec[i] <- pzPower(nVec[i], effect, n.sims, alpha)
  }

  #make the plot & return output
  plot(nVec, powVec, ...)

  powVec
}
```

Power One Piece at a Time

```r
pzPower <- function(n, effect=0, n.sims=500, alpha=0.05){

  #a vector of p-values
  p <- numeric(n.sims)

  #just your run of the mill power by
  #simulation for one choice of n and effect
  for(i in 1:n.sims){
    samp <- rnorm(n, effect)

    #Hey - it's our pz function!
    p[i] <- pz(samp)
  }

  #calculate power and return it
  1-sum(p > alpha)/n.sims

}
```

The Power of Flexible Functions and P-Values

```r
pzPowerSample(3:10, effect=1, type="b")
```




```
# [1] 0.528 0.550 0.626 0.666 0.724 0.778 0.850 0.836
```