Class 3 Summary

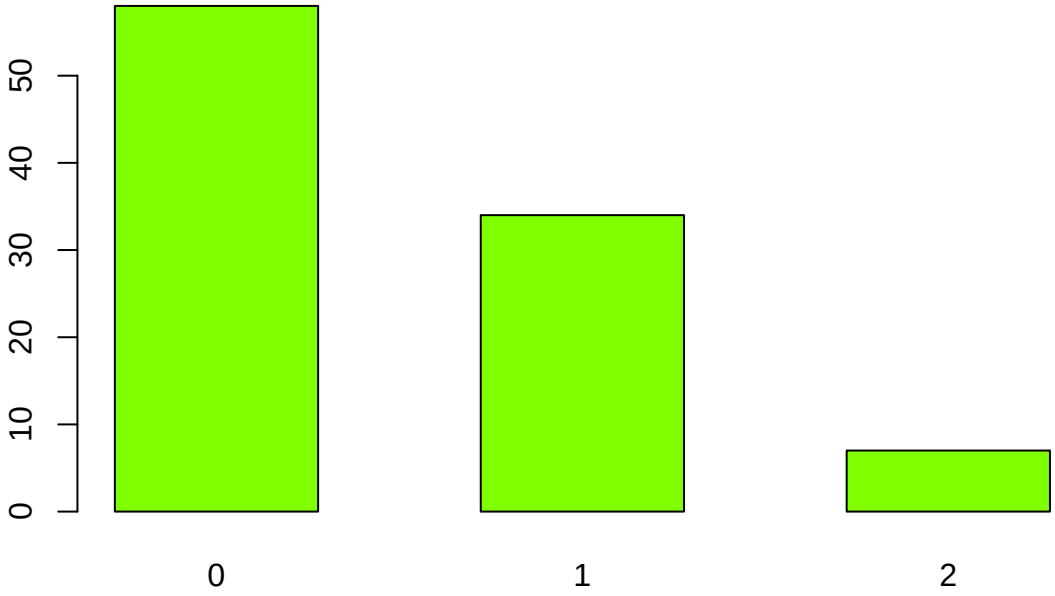
Jenna G. Tichon

07/10/2019

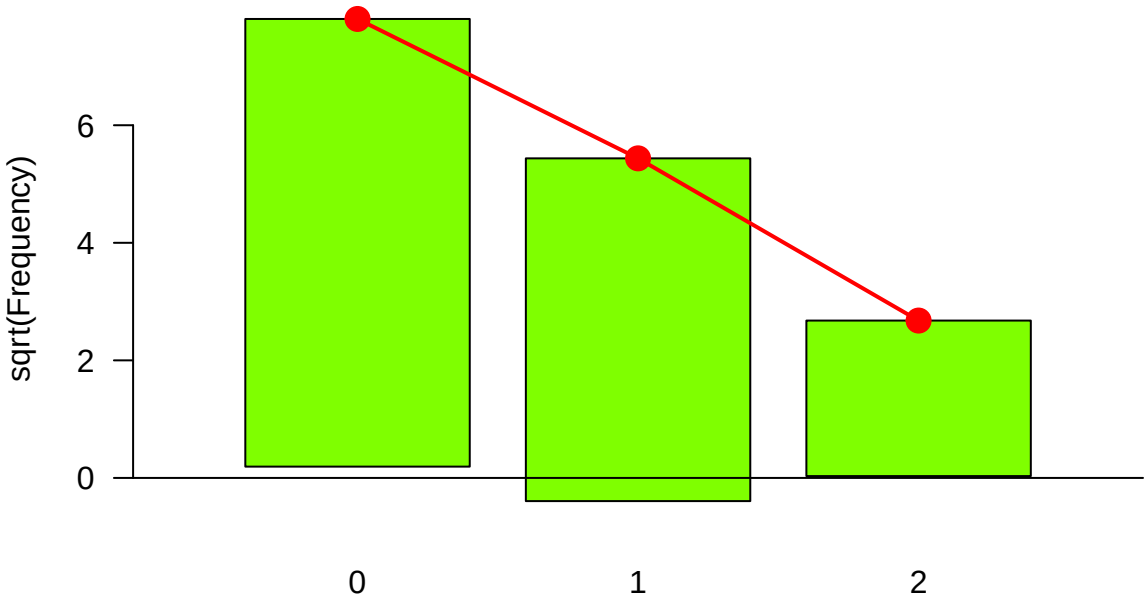
2.3 A simple Example of Statistical Modeling

This is a process for taking real data and trying to decide which distribution we should set it to.

```
load(here("data", "e100.RData"))

#remove outlier to make dataset easier to work with
e99 = e100[-which.max(e100)]

#see picture of distribution to try to decide distribution
barplot(table(e99), space = 0.8, col = "chartreuse")
```




```
#Using vcd library, create theoretical fit of data set to poisson
gf1 = goodfit( e99, "poisson")

#The rootogram shifts the barplot to match theoretical values to show how far off you are
rootogram(gf1, xlab = "", rect_gp = gpar(fill = "chartreuse"))
```

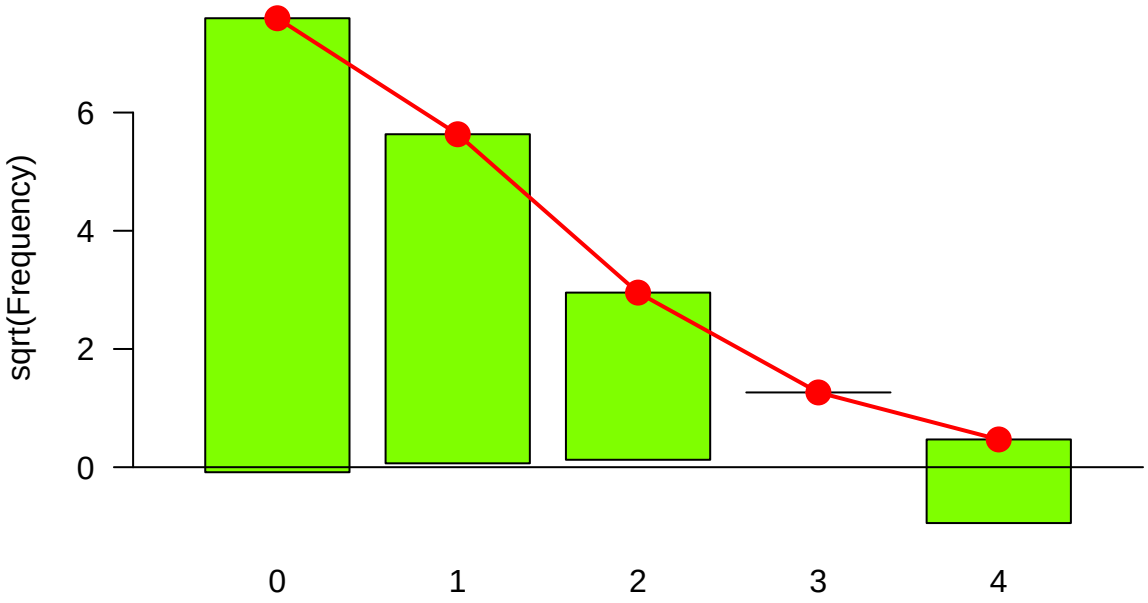


*R tip: goodfit takes in "poisson", "binomial", "nbinomial"

Q2.1

Generate 100 random poissons with $\lambda = 0.5$ to test out rootogram

```
pois.100<-rpois(100,0.5)
gf2 = goodfit(pois.100, "poisson")
rootogram(gf2, xlab="", rect_gp = gpar(fill = "chartreuse"))
```



For the **MLE** we are looking for the most likely parameter based on the observed data.

```
table(e100)
```

```
## e100
```

```
##  0  1  2  7
## 58 34  7  1
```

```
table(rpois(100,3))
```

```
##
##  0  1  2  3  4  5  6  7  8  9
##  4 18 21 20 17 12  4  2  1  1
```

Comparing our dataset to a Poisson 3 obviously shows that 3 would be a bad parameter estimate

Q2.2

Given that we have 58 0's, 34 1's, and 7 2's, what's the probability of that happening given they are Poisson m?

$$P(0)^{58} \times P(1)^{34} \times P(2)^{7} \times P(7)^{1}$$

for $m = 3$ this is:

```
#Side Note This gives individual probabilities
dpois(c(0,1,2,7),lambda = 3)^(c(58, 34, 7, 1))
```

```
## [1] 2.708695e-76 8.396253e-29 2.833371e-05 2.160403e-02
```

```
#the Prod function gives us the product
prod(dpois(c(0,1,2,7),lambda = 3)^(c(58, 34, 7, 1)))
```

```
## [1] 1.392143e-110
```

Which is decidedly super unlikely.

Q2.3

My Function to try different m values and ascertain the likelihood of the data given that they are poisson m.

```
#Function for trying different m's
pois100prob<-function(m){
prod(dpois(c(0,1,2,7),lambda = m)^(c(58, 34, 7, 1)))
}

pois100prob(0)
```

```
## [1] 0
```

```
pois100prob(1)
```

```
## [1] 5.766487e-50
```

```
pois100prob(2)
```

```
## [1] 7.728814e-77
```

```
pois100prob(0.4)
```

```
## [1] 8.5483e-46
```

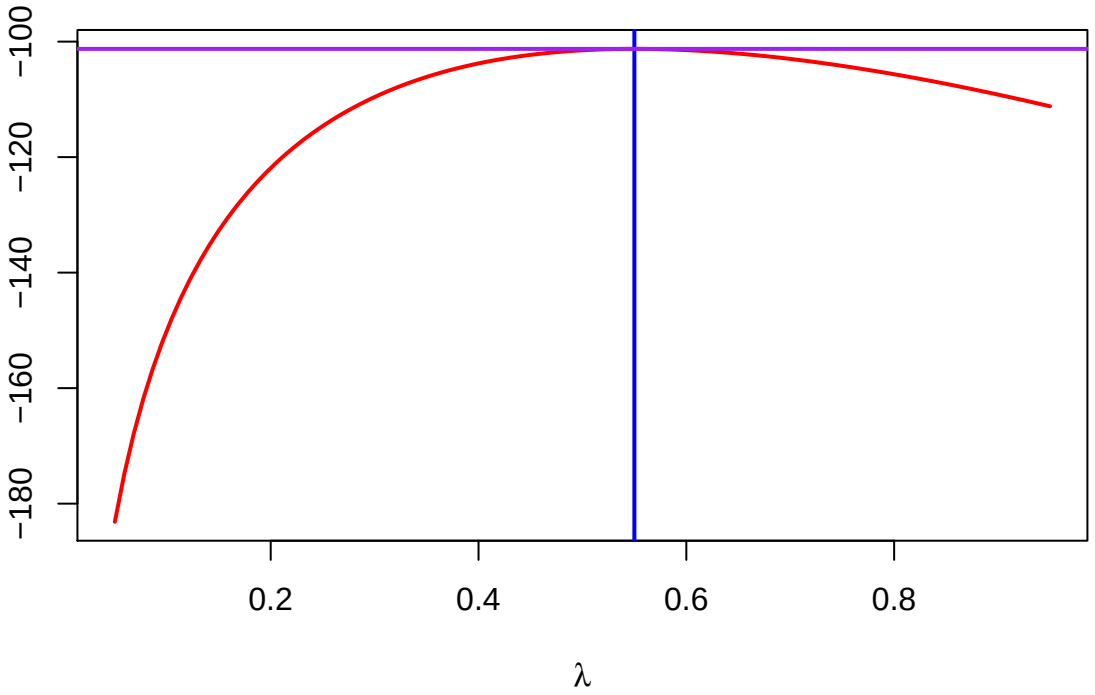
Text's function to find out the log likelihood of various m values to find the max:

```
loglikelihood = function(lambda, data = e100){
sum(log(dpois(data,lambda)))
}
```

Note the sum of the logs of the likelihood is maximized when the product of the likelihoods is.

Use this function to evaluate for a series of lambdas:

```
lambdas = seq(0.05, 0.95, length = 100)
loglik = vapply(lambdas, loglikelihood, numeric(1))
plot(lambdas, loglik, type = "l", col = "red", ylab = "", lwd = 2, xlab = expression(lambda))
m0 = mean(e100)
abline(v = m0, col = "blue", lwd = 2)
abline(h = loglikelihood(m0), col = "purple", lwd = 2)
```




```
m0
```

```
## [1] 0.55
```

> *__R tip__: vapply applies the loglikelihood function to all of the elements of lambdas. numeric(1) tells it that it's returning a single numeric value

Good fit has a shortcut for this:

```
gf = goodfit(e100, "poisson")
names(gf)
```

```
## [1] "observed" "count"    "fitted"   "type"     "method"   "df"
## [7] "par"
```

```
gf$par
```

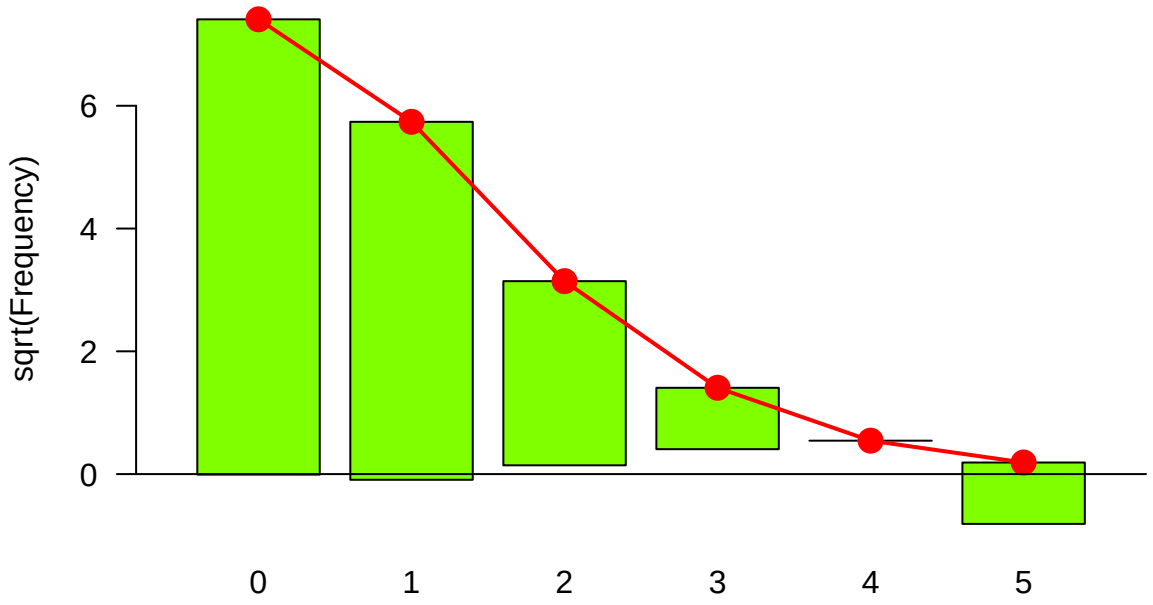
```
## $lambda
## [1] 0.55
```

The outputs are:

- *observed* : observed frequencies
- *count* : corresponding counts
- *fitted* : expected frequencies (maximum likelihood)
- *type* : distribution being fitted
- *method*: fitting method: "ML", "MinChisq", "fixed"
- *df* : degrees of freedom
- *par* : named list of parameter

Redoing the rootogram using 0.55:

```
pois.100<-rpois(100,0.55)
gf2 = goodfit(pois.100, "poisson")
rootogram(gf2, xlab="", rect_gp = gpar(fill = "chartreuse"))
```



Q2.6

Known distributions allow us to not "reinvent the wheel" and reuse methods without rederiving results for each individual data set.

Binomial Distributions and maximum likelihood

Looking at loglikelihood of binomial. Here's an example dataset:

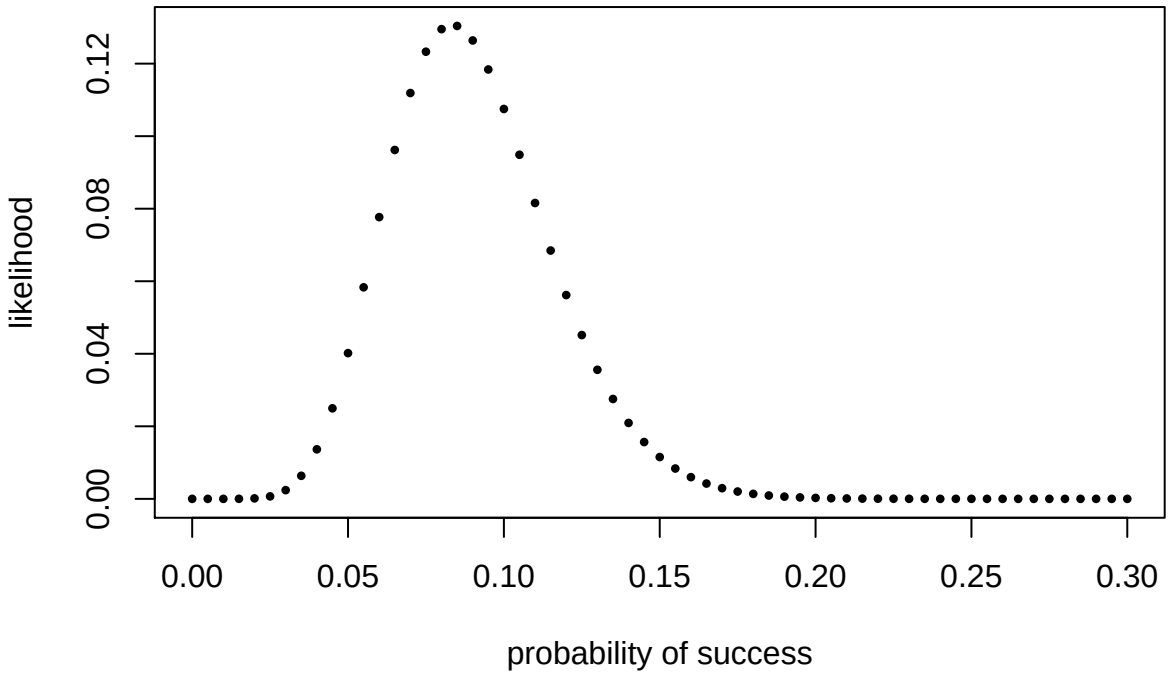
```
cb<-c(rep(0,110), rep(1,10))
table(cb)
```

```
## cb
##   0   1
## 110  10
```

We'd expect the maximum likelihood value to be 10/110=0.0909091

We can test this out using R

```r
probs = seq(0, 0.3, by = 0.005)
likelihood = dbinom(sum(cb), prob = probs, size = length(cb))
plot(probs, likelihood, pch = 16, xlab = "probability of success", ylab = "likelihood", cex=0.6)
```

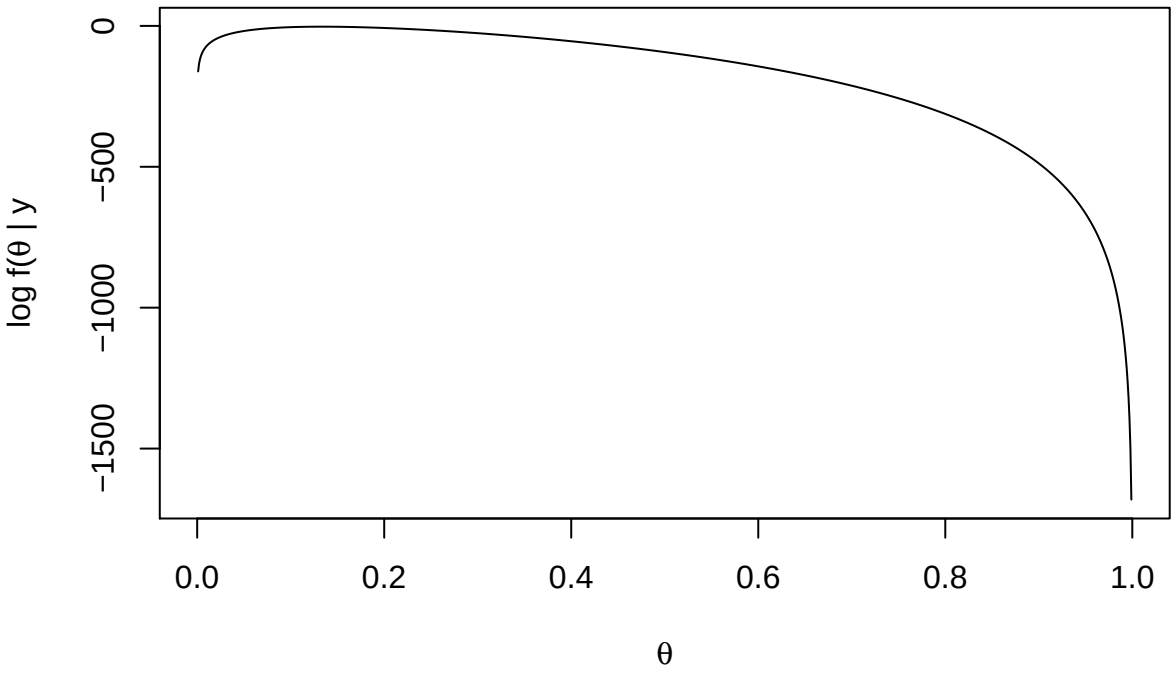



```r
probs[which.max(likelihood)]
```

```
## [1] 0.085
```

We can find the loglikelihood function for binomial as:

```r
loglikelihood.binom = function(theta, n = 300, k = 40){
  115 + k * log(theta) + (n - k) * log(1 - theta)
}

thetas = seq(0, 1, by = 0.001)
plot(thetas, loglikelihood.binom(thetas), xlab = expression(theta),
     ylab = expression(paste("log f(", theta, " | y")), type = "l")
```



*R tip: WHAT DOES EXPRESSION DO?

NB: The diagram is flat near the max. This implies that a Bayesian might suggest that the value of θ is something random in a range of those likely values.

2.5 More boxes: multinomial data

*Bio tip: Four types of moecules in DNA: A - adenine, C - cytosine, G - guanine, T - thymine. A and G are purines and C and T are pyrimidines

Looking at one DNA sequence

```
library("Biostrings")
```

```
## Loading required package: BiocGenerics
```

```
## Loading required package: parallel
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
```

```
## The following objects are masked from 'package:base':
##
```

```
##     anyDuplicated, append, as.data.frame, basename, cbind,
##     colnames, dirname, do.call, duplicated, eval, evalq, Filter,
##     Find, get, grep, grepl, intersect, is.unsorted, lapply, Map,
##     mapply, match, mget, order, paste, pmax, pmax.int, pmin,
##     pmin.int, Position, rank, rbind, Reduce, rownames, sapply,
##     setdiff, sort, table, tapply, union, unique, unsplit, which,
##     which.max, which.min

## Loading required package: S4Vectors

## Warning: package 'S4Vectors' was built under R version 3.6.1

## Loading required package: stats4

##
## Attaching package: 'S4Vectors'

## The following object is masked from 'package:base':
##
##     expand.grid

## Loading required package: IRanges

## Warning: package 'IRanges' was built under R version 3.6.1

##
## Attaching package: 'IRanges'

## The following object is masked from 'package:vcd':
##
##     tile

## Loading required package: XVector

##
## Attaching package: 'Biostrings'

## The following object is masked from 'package:base':
##
##     strsplit
```

```
here("data", "e100.RData")
```

```
## [1] "/Users/Jenna1/Google Drive/ModernStatsModernBioJGT/data/e100.RData"
```

```
staph = readDNAStringSet(here("data","staphsequence.ffn.txt"), "fasta")
staph[1]
```

```
##   A DNAStringSet instance of length 1
##     width seq                                               names
## [1]  1362 ATGTCGGAAAAAGAAATTTGG...AAGAAATAAGAAATGTATAA 1cl|NC_002952.2_c...
```

```
letterFrequency(staph[[1]], letters = "ACGT", OR = 0)
```

```
##   A   C   G   T
## 522 219 229 392
```

*__R tip__: The doublebrackets around the 1 pulls out the entire 1st sequence. Single brakets just gives the whole mess of data because staph is only one element long.

Q 2.9

ALERT!!: Reread this

*Following a similar procedure as in Exercise 1.8, test whether the nucleotides are equally distributed across the four nucleotides for the first gene.

Here are the observed proportions where set an estimate of equal across all genes by averaging the observed proportions across all A, C, G, T (i.e. as if the mucelotides are in consistent proportion across all genes):

```
#Find letter frequency
letterFrq = vapply(staph, letterFrequency, FUN.VALUE = numeric(4),
         letters = "ACGT", OR = 0)
colnames(letterFrq) = paste0("gene", seq(along = staph))
#Compute frequencies in first 10 genes and convert to proportions
tab10 = letterFrq[, 1:10]
computeProportions = function(x) { x/sum(x) }
prop10 = apply(tab10, 2, computeProportions)
round(prop10, digits = 2)
```

```
##   gene1 gene2 gene3 gene4 gene5 gene6 gene7 gene8 gene9 gene10
## A  0.38  0.36  0.35  0.37  0.35  0.33  0.33  0.34  0.38   0.27
## C  0.16  0.16  0.13  0.15  0.15  0.15  0.16  0.16  0.14   0.16
## G  0.17  0.17  0.23  0.19  0.22  0.22  0.20  0.21  0.20   0.20
## T  0.29  0.31  0.30  0.29  0.27  0.30  0.30  0.29  0.28   0.36
```

```
p0 = rowMeans(prop10)
p0
```

```
##         A         C         G         T
## 0.3470531 0.1518313 0.2011442 0.2999714
```

We find the expected probabilities by multiply the mean proportions for each nucleotide with the total count for each gene. i.e. This is the way the observed counts would divide for each gene if the proportion was equal across all genes

```
cs = colSums(tab10)
cs
```

```
##  gene1  gene2  gene3  gene4  gene5  gene6  gene7  gene8  gene9 gene10
##   1362   1134    246   1113   1932   2661    831   1515   1287    696
```

```
expectedtab10 = outer(p0, cs, FUN = "*")
round(expectedtab10)
```

```
##   gene1 gene2 gene3 gene4 gene5 gene6 gene7 gene8 gene9 gene10
## A   473   394    85   386   671   924   288   526   447    242
## C   207   172    37   169   293   404   126   230   195    106
## G   274   228    49   224   389   535   167   305   259    140
## T   409   340    74   334   580   798   249   454   386    209
```

Make a random table with observed column counts if generated from a multinomial with our null proportion spread (i.e. equal across all genes)

```
randomtab10 = sapply(cs, function(s) { rmultinom(1, s, p0) } )
all(colSums(randomtab10) == cs)
```

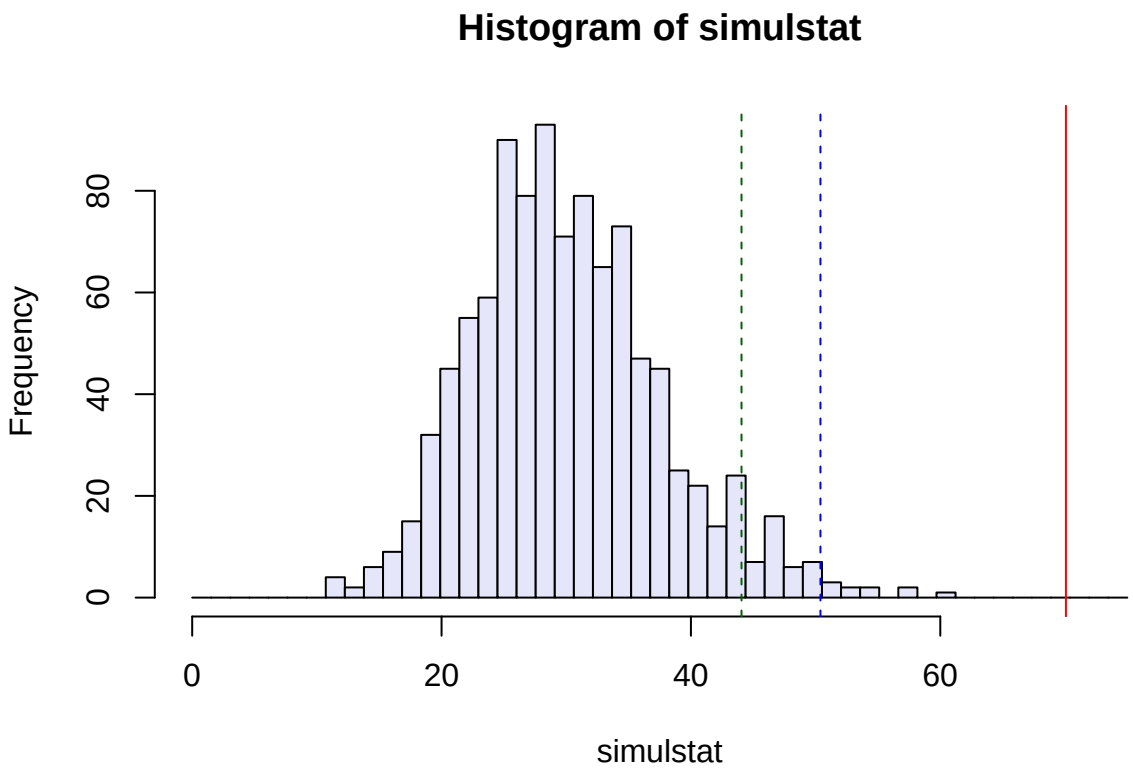
```
## [1] TRUE
```

Repeat this 1000 times to see how often we get a chi-squared value more extreme than we observed.

```
stat = function(obsvd, exptd = 20 * pvec) {
   sum((obsvd - exptd)^2 / exptd)
}
B = 1000
simulstat = replicate(B, {
  randomtab10 = sapply(cs, function(s) { rmultinom(1, s, p0) })
  stat(randomtab10, expectedtab10)
})
S1 = stat(tab10, expectedtab10)
sum(simulstat >= S1)
```

```
## [1] 0
```

```
hist(simulstat, col = "lavender", breaks = seq(0, 75, length.out=50))
abline(v = S1, col = "red")
abline(v = quantile(simulstat, probs = c(0.95, 0.99)),
       col = c("darkgreen", "blue"), lty = 2)
```



It happens 0 times! This is good reason to reject that it's multinomial with equal proportions across all genes.