Class 3 Summary

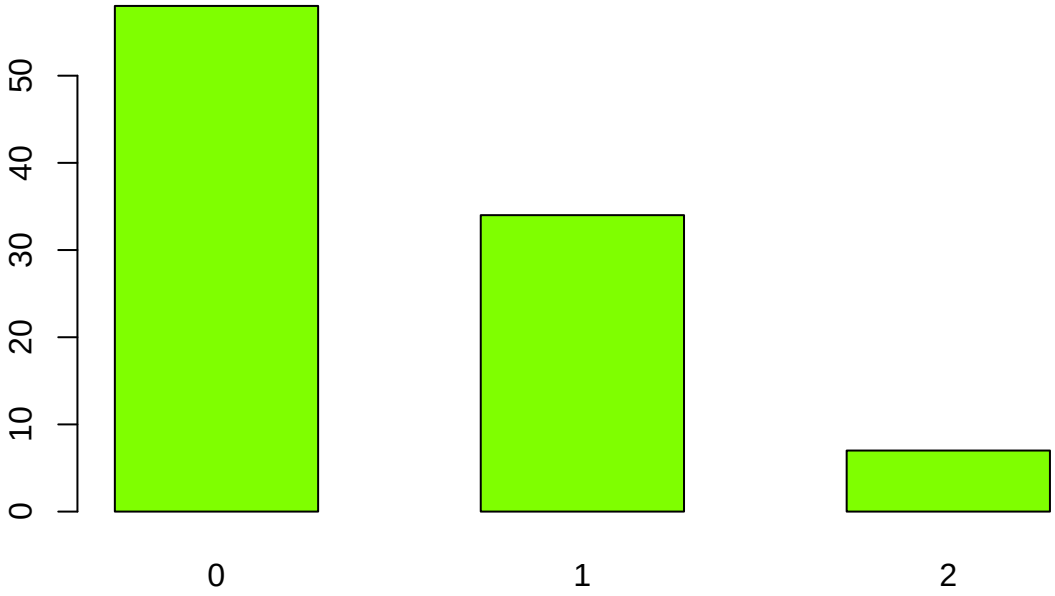
Jenna G. Tichon

07/10/2019

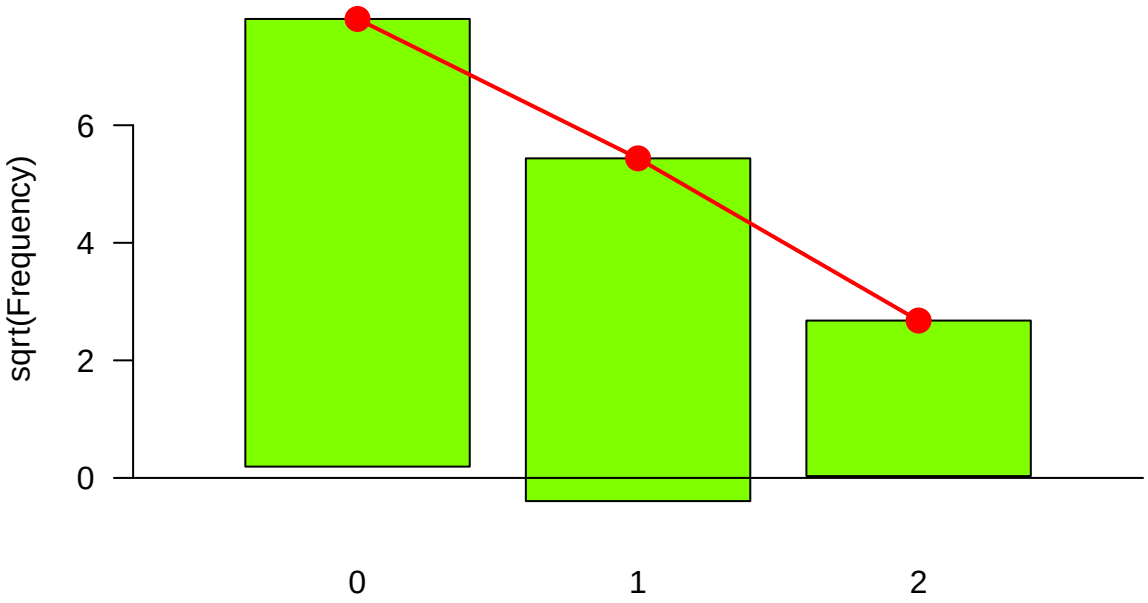
2.3 A simple Example of Statistical Modeling

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"))
```

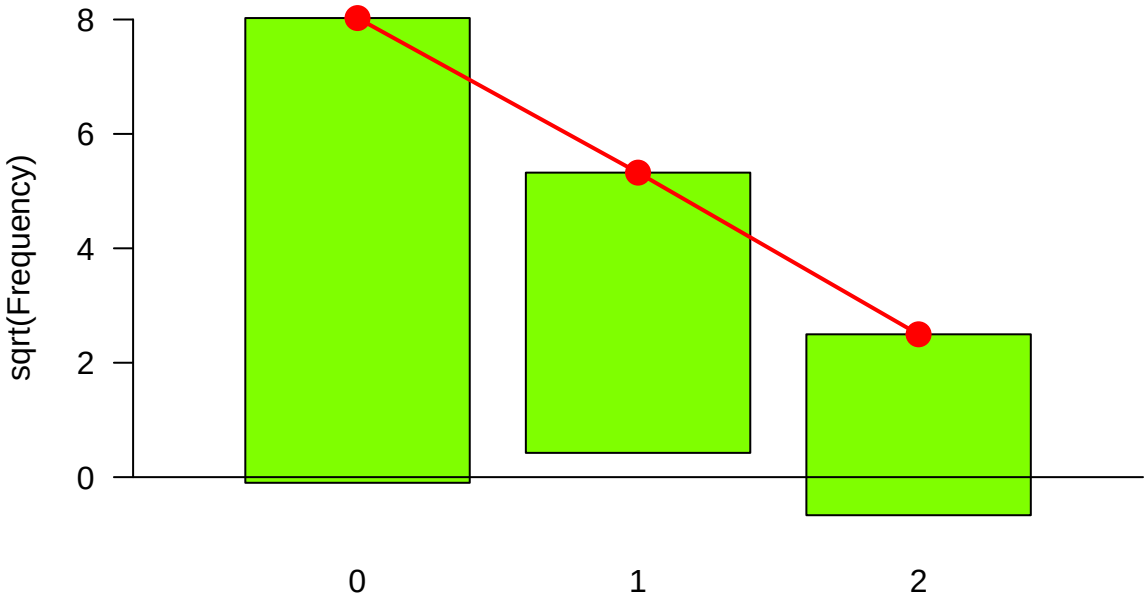


*R tip: `goodfit` takes in "poisson", "binomial", "nbinomial"

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
##  3 16 19 19 24 11  5  3
```

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
```

<<<<<<< HEAD

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
```

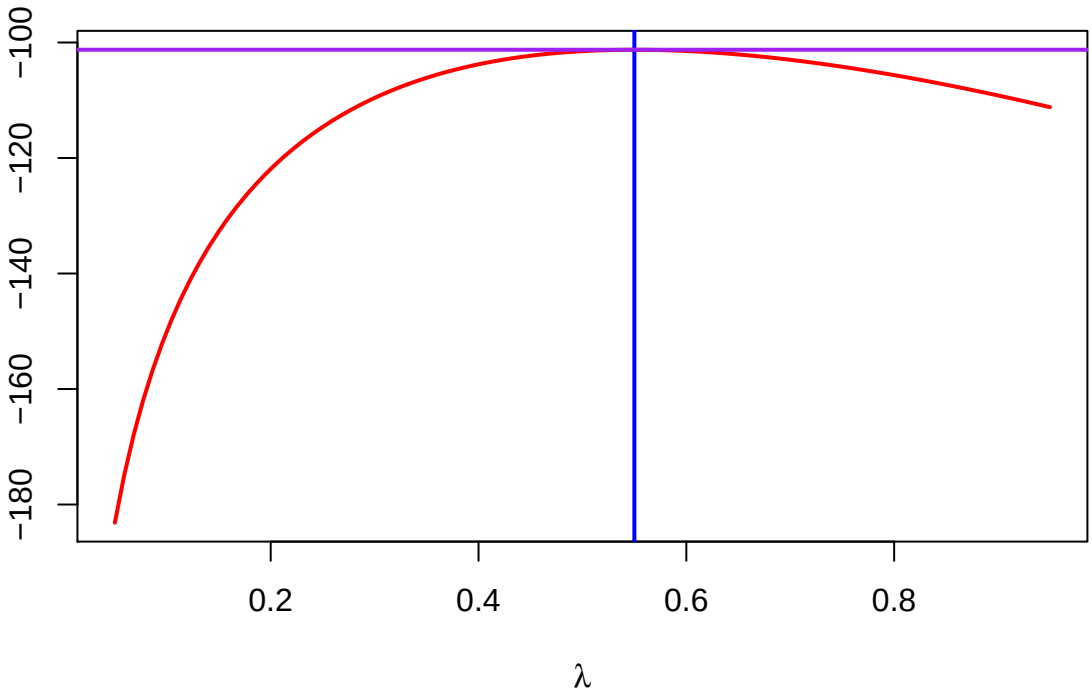
Text's function to find out the log likelihood of various m values to find the max:

```
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
```

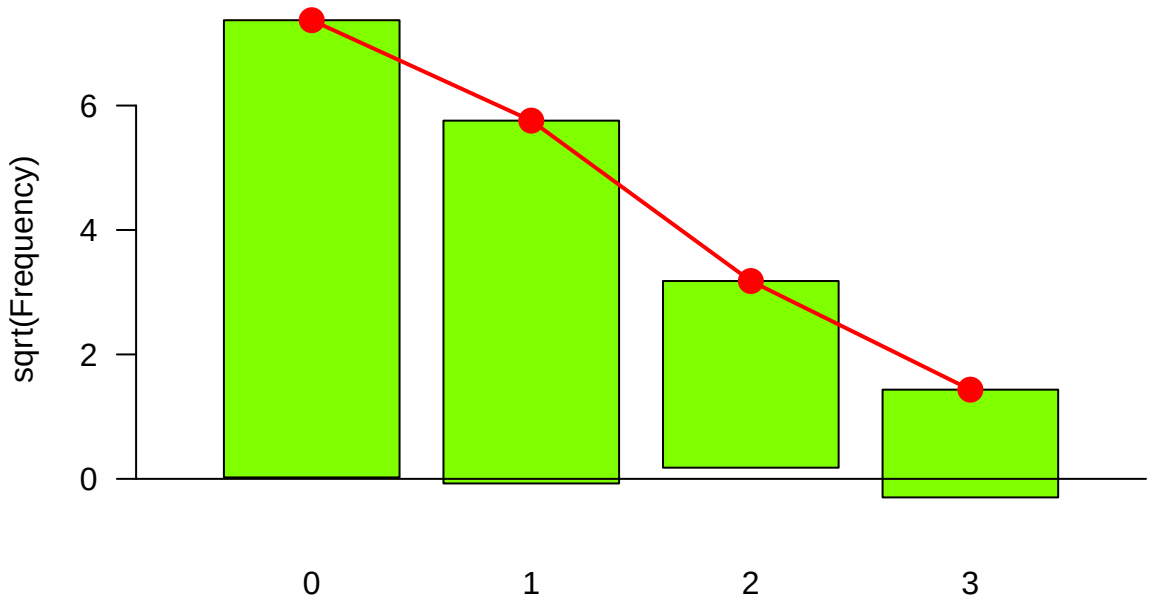
```
## [1] 0.55
```

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:

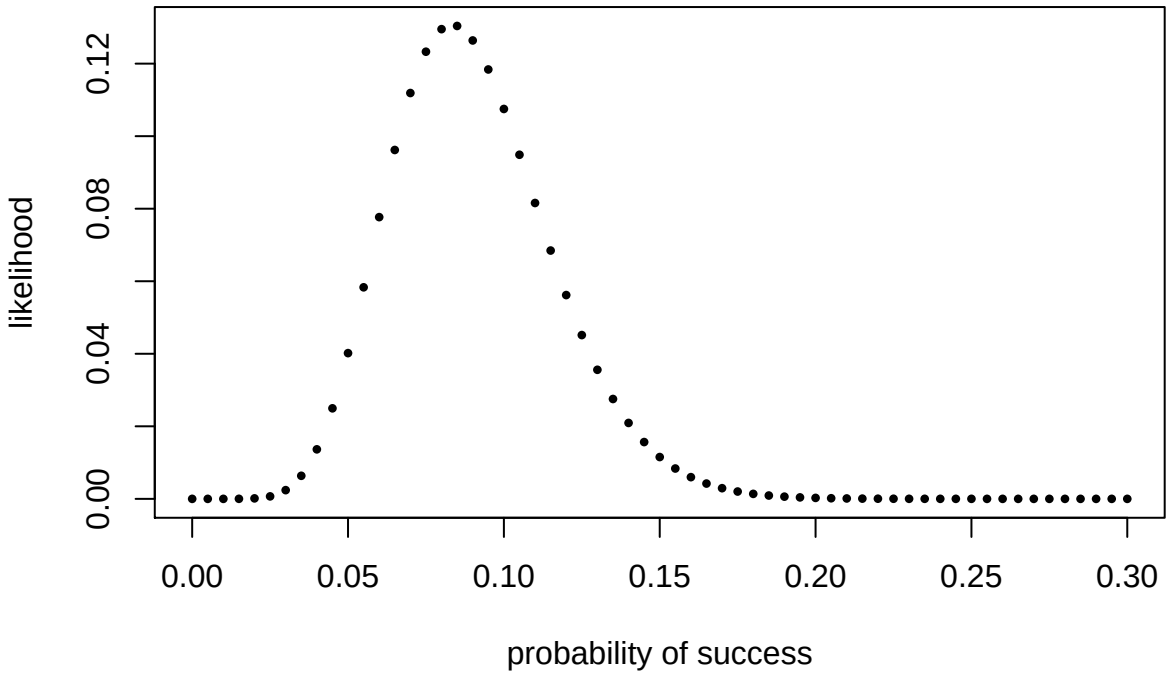
```
cb<-c(rep(0,110), rep(1,10))
table(cb)
```

```
## 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)
```

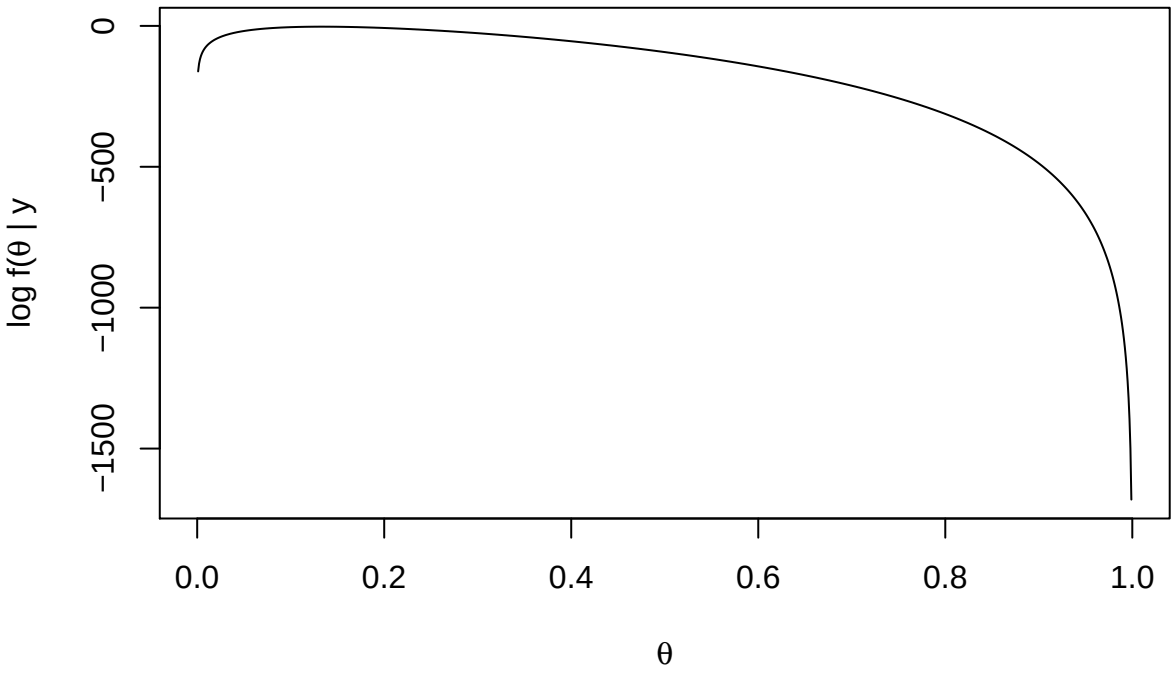



```r
probs[which.max(likelihood)]
```

```
## [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

## Loading required package: stats4

##
## Attaching package: 'S4Vectors'

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

## Loading required package: IRanges

##
## 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)
```

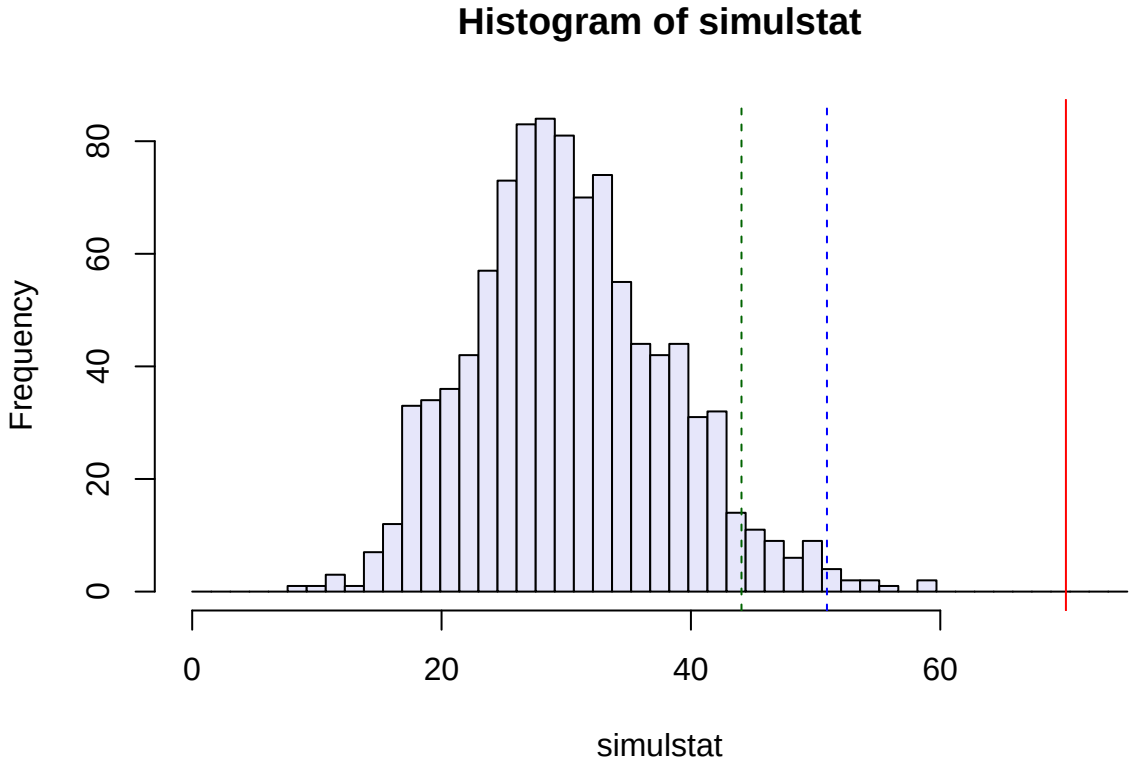
```
## [1] TRUE
```

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.

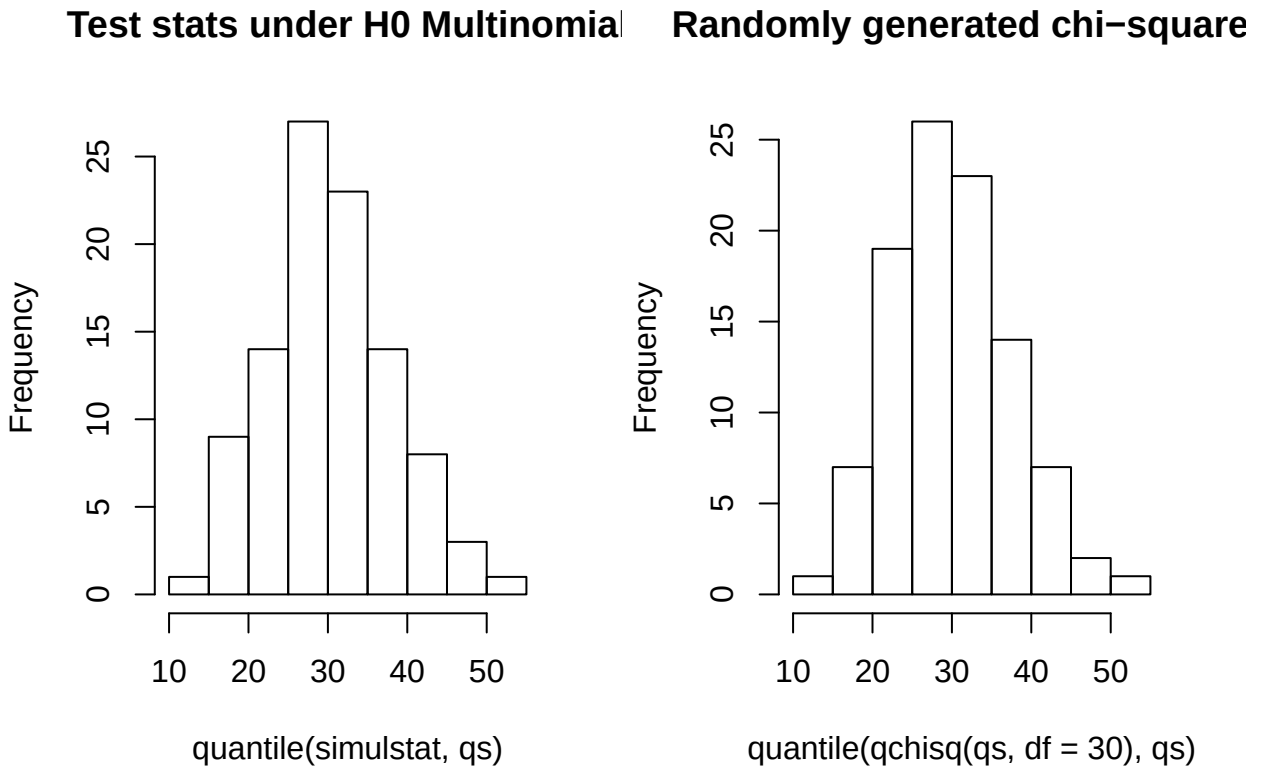
2.6 The χ^2 distribution

2.6.1 Intermezzo: quantiles and the quantile-quantile plot

Q 2.10

Compare quantiles of actual chi-squared test statistics vs our randomly generated chi-squared values with 30 ($= 10 \times (4-1)$) df.

```
qs = ppoints(100)
par(mfrow=c(1,2))
hist(quantile(simulstat, qs), main = "Test stats under H0 Multinomial")
hist(quantile(qchisq(qs, df = 30), qs), main = "Randomly generated chi-squared")
```

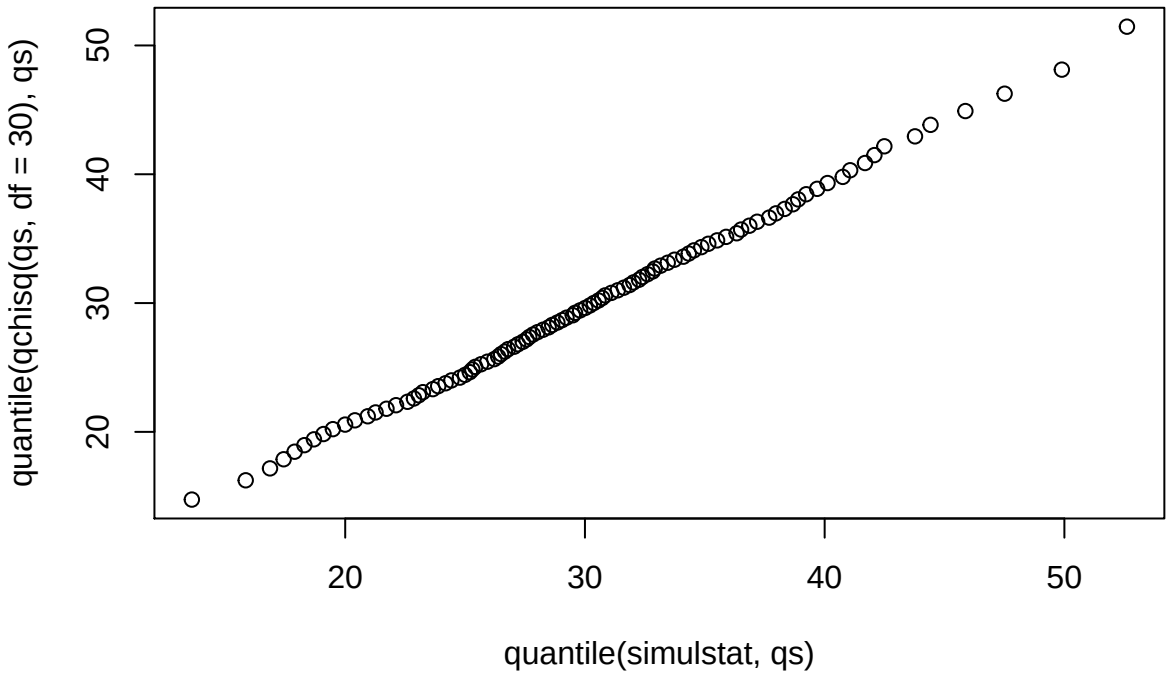



```
dev.off()
```

```
## null device
##           1
```

QQ-Plot comparing the two distributions:

```
qqplot(quantile(simulstat, qs),quantile(qchisq(qs, df = 30), qs))
```



This justifies that the test statistic for whether it follows the distribution is $\chi^2_{(30)}$.

Q 2.11

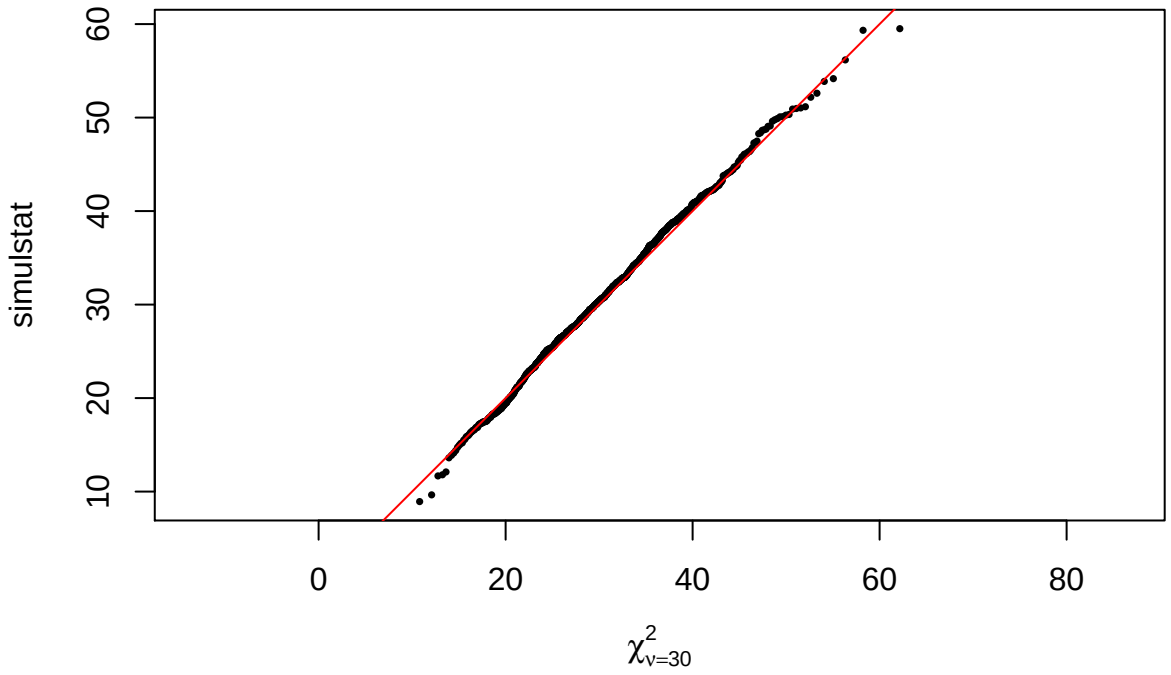
Median / Second quartile

Q 2.12

Weighted average of order statistics is how R computes quantiles

Text's qq-plot:

```
qqplot(qchisq(ppoints(B), df = 30), simulstat, main = "",
       xlab = expression(chi[nu==30]^2), asp = 1, cex = 0.5, pch = 16)
abline(a = 0, b = 1, col = "red")
```

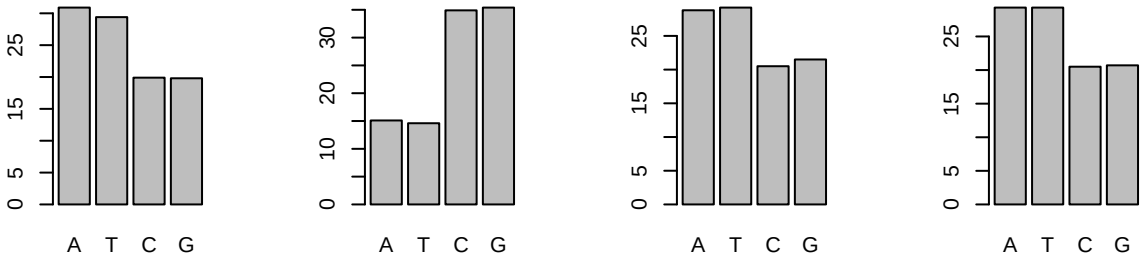


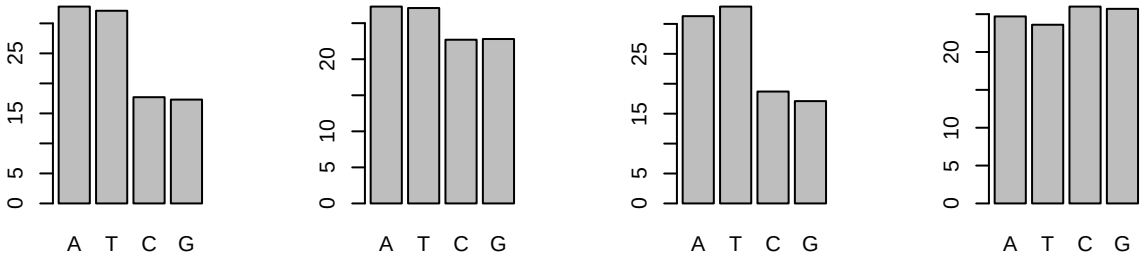
2.7 Chargaff's Rule

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

```
##                   A    T    C    G
## Human-Thymus   30.9 29.4 19.9 19.8
## Mycobac.Tuber  15.1 14.6 34.9 35.4
## Chicken-Eryth. 28.8 29.2 20.5 21.5
## Sheep-liver    29.3 29.3 20.5 20.7
## Sea Urchin     32.8 32.1 17.7 17.3
## Wheat          27.3 27.1 22.7 22.8
## Yeast          31.3 32.9 18.7 17.1
## E.coli         24.7 23.6 26.0 25.7
```

```
par(mfrow=c(2,4))
barplot(ChargaffTable[1,])
barplot(ChargaffTable[2,])
barplot(ChargaffTable[3,])
barplot(ChargaffTable[4,])
barplot(ChargaffTable[5,])
barplot(ChargaffTable[6,])
barplot(ChargaffTable[7,])
barplot(ChargaffTable[8,])
```






```
dev.off()
```

```
## null device
##           1
```

They don't appear to come from the same distribution. 1,3,5,6,7 look the same-ish, 2 looks different and 8 might be it's own or grouped with 2.

> *__Bio tip__: Chargaff's rule says A and T amounts will be similar while C and G amounts will be similar within one organism but not necessarily between organisms.

We might make a test statistic

$$(p_C - p_G)^2 + (p_A + pT)^2$$

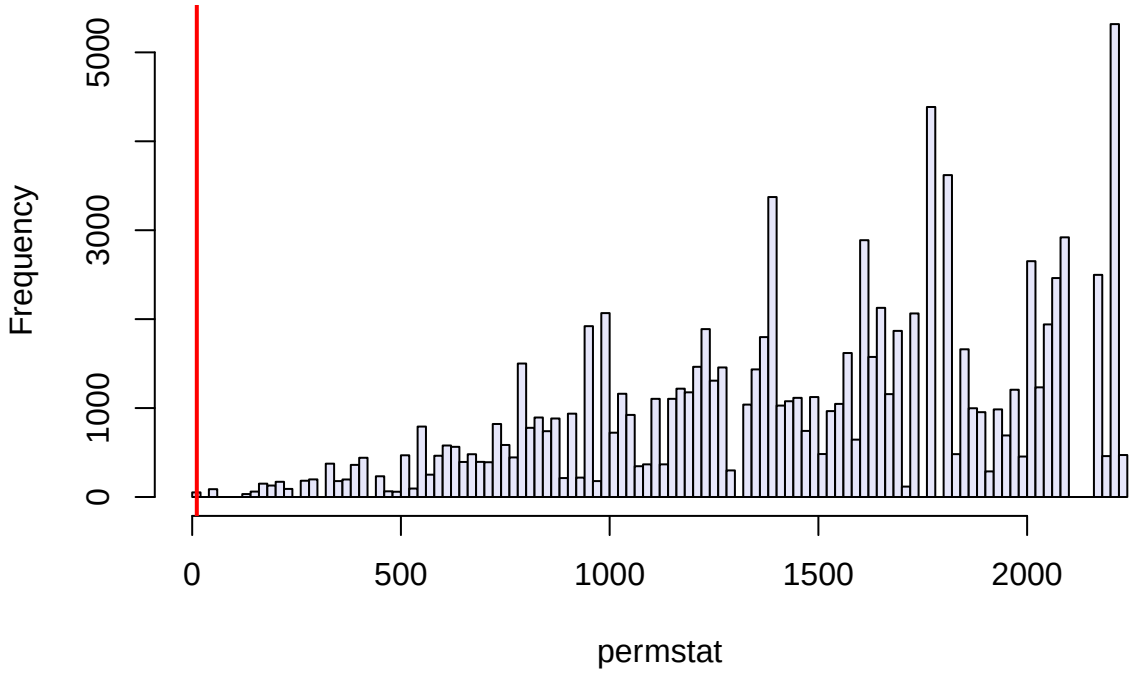
because this would zero under a null hypothesis that A/T are the same and C/G are the same.

We can test this as a permutation test:

```
#Calculate proposed test stat
statChf = function(x){
  sum((x[, "C"] - x[, "G"])^2 + (x[, "A"] - x[, "T"])^2)
}
chfstat = statChf(ChargaffTable)

#Calculate test stats under a permutation test of relabeling all rows as if there was no pattern
permstat = replicate(100000, {
  permuted = t(apply(ChargaffTable, 1, sample))
  colnames(permuted) = colnames(ChargaffTable)
  statChf(permuted)
})

#p-value
pChf = mean(permstat <= chfstat)
pChf
```

```
## [1] 0.00013
```

```
hist(permstat, breaks = 100, main = "", col = "lavender")
abline(v = chfstat, lwd = 2, col = "red")
```



We only consider lower values because we would never consider a high test statistic as indicative that A/T and C/G were similar.

2.7.1 Two categorical Variables

```
HairEyeColor[,,"Female"]
```

```
##        Eye
## Hair    Brown Blue Hazel Green
##   Black    36    9     5     2
##   Brown    66   34    29    14
##   Red      16    7     7     7
##   Blond     4   64     5     8
```

```
dim(HairEyeColor)
```

```
## [1] 4 4 2
```

```
str(HairEyeColor)
```

```
##  'table' num [1:4, 1:4, 1:2] 32 53 10 3 11 50 10 30 10 25 ...
##  - attr(*, "dimnames")=List of 3
##   ..$ Hair: chr [1:4] "Black" "Brown" "Red" "Blond"
##   ..$ Eye : chr [1:4] "Brown" "Blue" "Hazel" "Green"
##   ..$ Sex : chr [1:2] "Male" "Female"
```

This is a built in dataset with dimensions $4 \times 4 \times 4$

```
load(here("Data", "Deuteranopia.RData"))
Deuteranopia
```

```
##           Men Women
## Deute      19     2
## NonDeute 1981  1998
```

```
#Chi-square Test for Independence between and occurrence of color blindness
chisq.test(Deuteranopia)
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  Deuteranopia
## X-squared = 12.255, df = 1, p-value = 0.0004641
```

2.7.2 A special multinomial: Hardy-Weinberg equilibrium

*__Bio tip__: Suppose two Alleles M and N. M has overall frequency p and N has $q = 1 - p$. If mating happens at random with independence between frequency of allele genotype then we get the **Hardy-Weingberg equilibrium (HWE)**

$$p_{MM} = p^2, \qquad p_{NN} = q^2, \qquad p_{MN} = 2pq$$

If they occur with frequencies n_{ij},

$$p(n_{MM}, n_{MN}, n_{NN}|p) = \binom{S}{n_{MM}, n_{MN}, n_{NN}} (p^2)^{n_{MM}} \times (2pq)^{n_{MN}} \times (q^2)^{n_{NN}}$$

$$L(p) = n_{MM} log(p^2) + n_{MN} log(2pq) + n_{NN} log(q^2)$$

which is maximized at

$$p = \frac{n_{MM} + n_{MN}/2}{S}$$

Loglikelihood for a dataset of allelles from Tahiti

```
library("HardyWeinberg")
```

```
## Loading required package: mice
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'mice'
```

```
## The following objects are masked from 'package:IRanges':
##
##     cbind, rbind
```

```
## The following objects are masked from 'package:S4Vectors':
##
##     cbind, rbind
```

```
## The following objects are masked from 'package:BiocGenerics':
##
##     cbind, rbind
```

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

```
## Loading required package: Rsolnp
```

```
data("Mourant")
Mourant[214:216,]
```

```
##     Population    Country Total  MM  MN  NN
## 214    Oceania Micronesia   962 228 436 298
## 215    Oceania Micronesia   678  36 229 413
## 216    Oceania     Tahiti   580 188 296  96
```

```
nMM = Mourant$MM[216]
nMN = Mourant$MN[216]
nNN = Mourant$NN[216]
loglik = function(p, q = 1 - p) {
  2 * nMM * log(p) + nMN * log(2*p*q) + 2 * nNN * log(q)
}
xv = seq(0.01, 0.99, by = 0.01)
yv = loglik(xv)
png(here("Class3","loglikelihoodtahiti.png"))
plot(x = xv, y = yv, type = "l", lwd = 2,
     xlab = "p", ylab = "log-likelihood")
imax = which.max(yv)
abline(v = xv[imax], h = yv[imax], lwd = 1.5, col = "blue")
abline(h = yv[imax], lwd = 1.5, col = "purple")
```

Expected values of proportions under Herdy-Weinberg equilibrium

```
#af is a function from HardyWeinberg package to calculate predicted proportions
phat  =  af(c(nMM, nMN, nNN))
phat
```

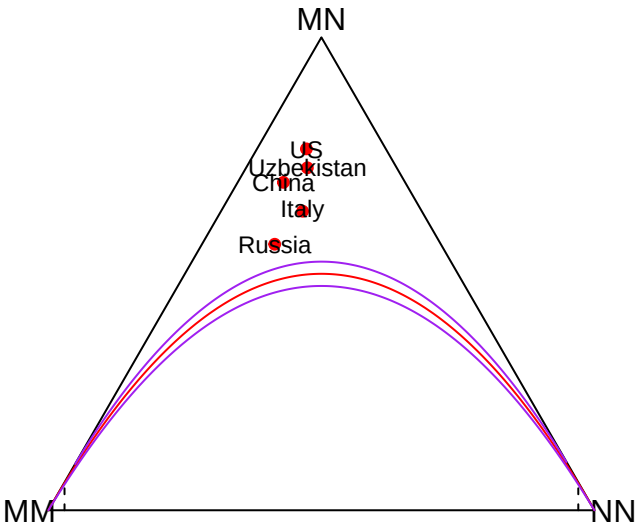
```
## [1] 0.5793103
```

```
pMM   =  phat^2
qhat  =  1 - phat
pHW = c(MM = phat^2, MN = 2*phat*qhat, NN = qhat^2)
sum(c(nMM, nMN, nNN)) * pHW
```

```
##       MM       MN       NN
## 194.6483 282.7034 102.6483
```

This graph shows confidence intervals for the Hardy-Weinberg Equilibrium

```
pops = c(1, 69, 128, 148, 192)
genotypeFrequencies = as.matrix(Mourant[, c("MM", "MN", "NN")])
HWTernaryPlot(genotypeFrequencies[pops, ],
        markerlab = Mourant$Country[pops],
        alpha = 0.0001, curvecols = c("red", rep("purple", 4)),
        mcex = 0.75, vertex.cex = 1)
```



2.7.3

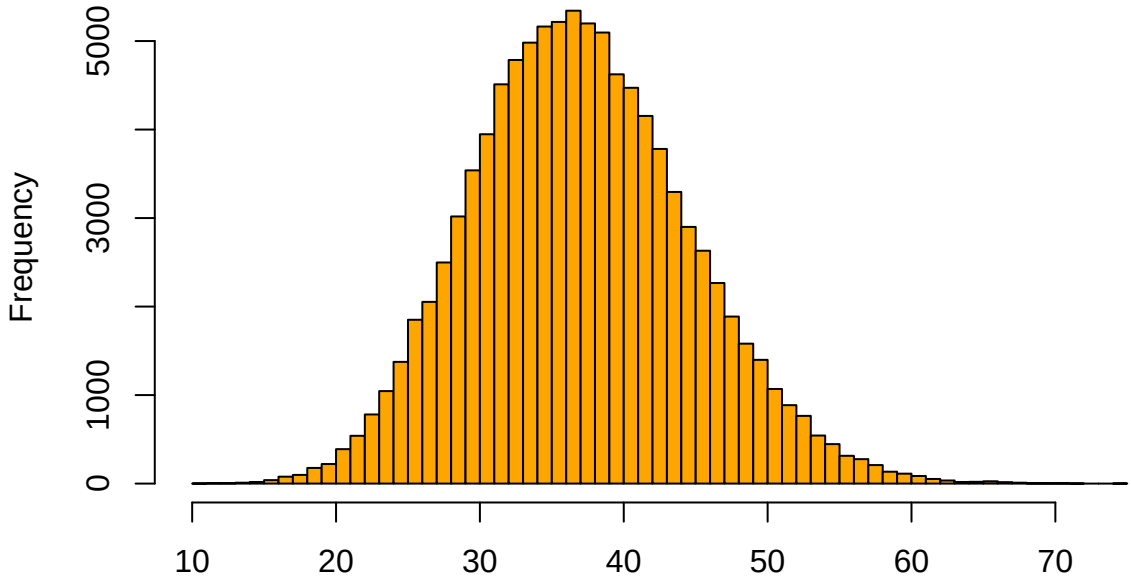
ALERT!!: Can't get this to run seqLogo package

2.9.1

```
haplo6=read.table(here("Data","haplotype6.txt"), header = TRUE)
haplo6
```

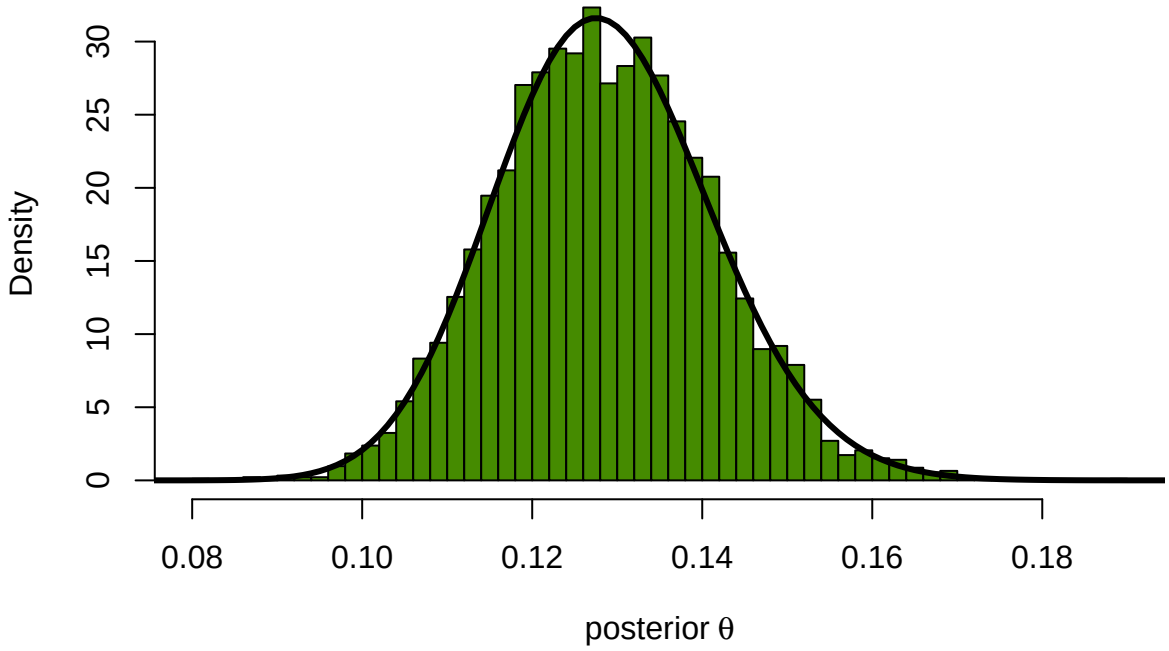
```
##   Individual DYS19 DXYS156Y DYS389m DYS389n DYS389p
## 1         H1    14       12       4      12       3
## 2         H3    15       13       4      13       3
## 3         H4    15       11       5      11       3
## 4         H5    17       13       4      11       3
## 5         H7    13       12       5      12       3
## 6         H8    16       11       5      12       3
```

```
#histogram of y's that are binomial but the probabilities are generated using beta's
rtheta = rbeta(100000, 50, 350)
y = vapply(rtheta, function(th) {
  rbinom(1, prob = th, size = 300)
}, numeric(1))
hist(y, breaks = 50, col = "orange", main = "", xlab = "")
```



ALERT!!: Why 90 610?

```
#all theta's that had y==40 with the theoretical density overtop
thetaPostEmp = rtheta[ y == 40 ]
hist(thetaPostEmp, breaks = 40, col = "chartreuse4", main = "",
  probability = TRUE, xlab = expression("posterior"~theta))
densPostTheory  =  dbeta(thetas, 90, 610)
lines(thetas, densPostTheory, type="l", lwd = 3)
```



Show that this agrees.

```
mean(thetaPostEmp)
```

```
## [1] 0.1286256
```

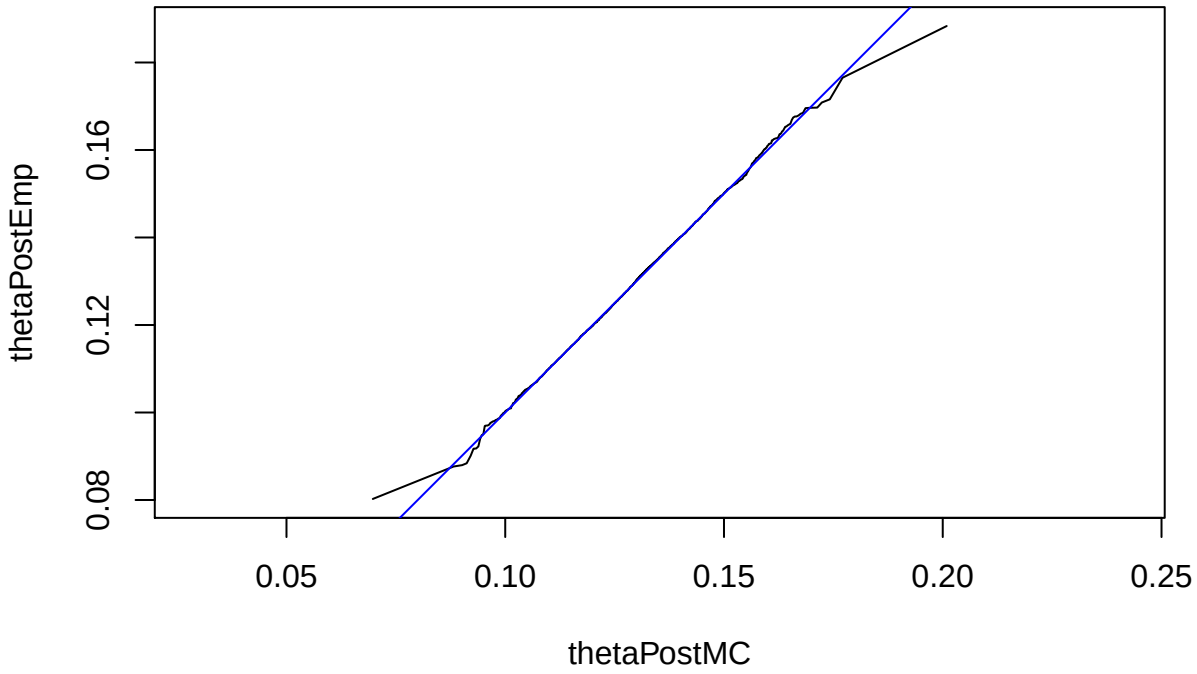
```r
dtheta = thetas[2]-thetas[1]
sum(thetas * densPostTheory * dtheta)
```

```
## [1] 0.1285714
```

```r
thetaPostMC = rbeta(n = 1e6, 90, 610)
mean(thetaPostMC)
```

```
## [1] 0.1285585
```

```r
qqplot(thetaPostMC, thetaPostEmp, type = "l", asp = 1)
abline(a = 0, b = 1, col = "blue")
```




```r
densPost2 = dbeta(thetas, 115, 735)
mcPost2   = rbeta(1e6, 115, 735)

sum(thetas * densPost2 * dtheta)  # mean, by numeric integration
```

```
## [1] 0.1352941
```

```r
mean(mcPost2)                     # mean, by MC
```

```
## [1] 0.1352915
```

```r
thetas[which.max(densPost2)]      # MAP estimate
```

```
## [1] 0.134
```

Q2.20

ALERT!!: WHAT DO HERE?

Posterior Credibility Interval

```
quantile(mcPost2, c(0.025, 0.975))
```

```
##      2.5%     97.5%
## 0.1131193 0.1590720
```

2.10 Example: occurrence of a nucelotide pattern in a genome

```
library("Biostrings")
library("BSgenome.Ecoli.NCBI.20080805")
```

```
## Loading required package: BSgenome
```

```
## Loading required package: GenomeInfoDb
```

```
## Loading required package: GenomicRanges
```

```
## Loading required package: rtracklayer
```

```
Ecoli
```

```
## E. coli genome:
## # organism: Escherichia coli (E. coli)
## # provider: NCBI
## # provider version: 2008/08/05
## # release date: NA
## # release name: NA
## # 13 sequences:
## #   NC_008253 NC_008563 NC_010468 NC_004431 NC_009801 NC_009800 NC_002655
## #   NC_002695 NC_010498 NC_007946 NC_010473 NC_000913 AC_000091
## # (use 'seqnames()' to see all the sequence names, use the '$' or '[['
## # operator to access a given sequence)
```

```
shineDalgarno = "AGGAGGT"
ecoli = Ecoli$NC_010473
```

Count occurrences of AGGAGGT in windows of width 50000

```
window = 50000
starts = seq(1, length(ecoli) - window, by = window)
ends   = starts + window - 1
numMatches = vapply(seq_along(starts), function(i) {
  countPattern(shineDalgarno, ecoli[starts[i]:ends[i]],
               max.mismatch = 0)
  }, numeric(1))
table(numMatches)
```

```
## numMatches
##  0  1  2  3  4
## 48 32  8  3  2
```

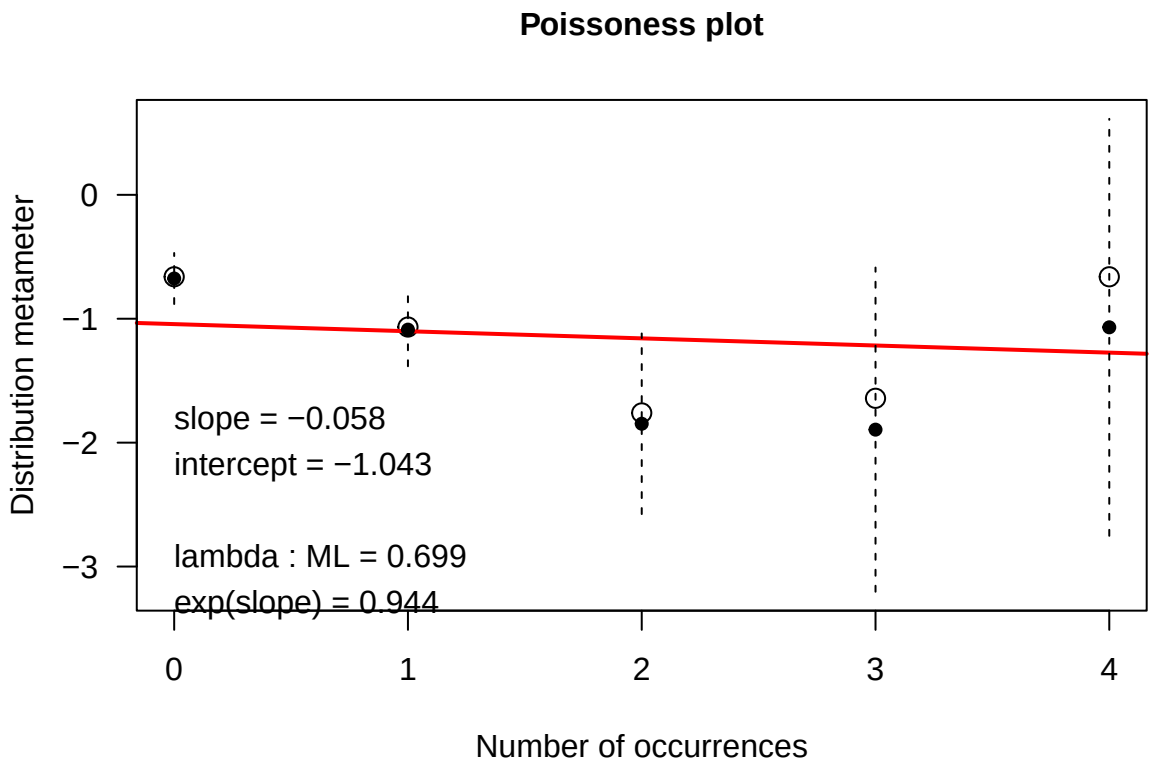
Check to see if this follows a Poisson distribution

```
library("vcd")
gf = goodfit(numMatches, "poisson")
summary(gf)
```

```
##
##   Goodness-of-fit test for poisson distribution
```

```
##
##                         X^2 df  P(> X^2)
## Likelihood Ratio 4.134932  3 0.2472577
```

```
distplot(numMatches, type = "poisson")
```




```
#Inspect matches
sdMatches = matchPattern(shineDalgarno, ecoli, max.mismatch = 0)
sdMatches
```

```
##   Views on a 4686137-letter DNAString subject
## subject: AGCTTTTCATTCTGACTGCAACGGGCAATATG...CAAATAAAAAACGCCTTAGTAAGTATTTTTC
## views:
##         start     end width
##   [1]   56593   56599     7 [AGGAGGT]
##   [2]  199644  199650     7 [AGGAGGT]
##   [3]  202176  202182     7 [AGGAGGT]
##   [4]  214433  214439     7 [AGGAGGT]
##   [5]  217429  217435     7 [AGGAGGT]
##   ...     ...     ...   ... ...
## [61] 4438786 4438792     7 [AGGAGGT]
## [62] 4498085 4498091     7 [AGGAGGT]
## [63] 4536658 4536664     7 [AGGAGGT]
## [64] 4546821 4546827     7 [AGGAGGT]
## [65] 4611626 4611632     7 [AGGAGGT]
```

```
#Distance between matches
betweenmotifs = gaps(sdMatches)
betweenmotifs
```

```
##   Views on a 4686137-letter DNAString subject
## subject: AGCTTTTCATTCTGACTGCAACGGGCAATATG...CAAATAAAAAACGCCTTAGTAAGTATTTTTC
```

```
## views:
##        start     end  width
##  [1]       1   56592  56592 [AGCTTTTCATTCTGACTGCAA...AGGTGTCAGAACCCGGCAGAC]
##  [2]   56600  199643 143044 [AAAGCTACCGTTATCCAGAAT...GAGAGCGCCTGCTTTGCACGC]
##  [3]  199651  202175   2525 [CTGCGGTTCGATCCCGCATAG...GGCTAATCCTGGTCGGACATC]
##  [4]  202183  214432  12250 [TAGTGCAATGGCATAAGCCAG...ATCGTGTTATCGCCAGGCTTT]
##  [5]  214440  217428   2989 [TAATAACATGGGCAGGATAAG...AACGAAAAGCCCCTTACTTGT]
##  ...     ...     ...    ... ...
## [62] 4438793 4498084  59292 [GGATTTAATCACGGTAACATT...AGTCATTGCATCGTCAACTTC]
## [63] 4498092 4536657  38566 [AGATCCGGTTGCGGCAGCAAG...TAAATTTGAACTCCAAATACC]
## [64] 4536665 4546820  10156 [GGAATTAAAGAATGCGATGGA...AGTTATACTTTGTATAACTTA]
## [65] 4546828 4611625  64798 [GCAGATGCGTATTACCATAAA...GCGCGCGCATCGCCGAGTGCA]
## [66] 4611633 4686137  74505 [CGAGATCGCGGCAAAAACTCA...ACGCCTTAGTAAGTATTTTTC]
```

If motifs appear at random, we expect the gaps between to be exponential.

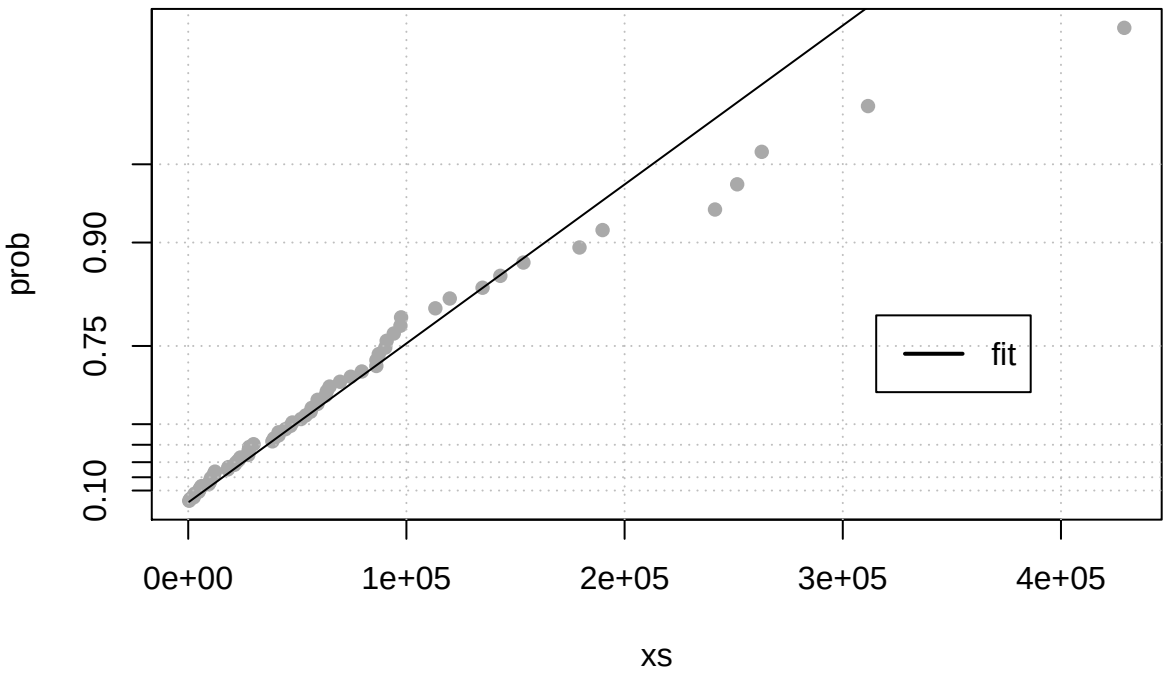
```
library("Renext")
```

```
## Loading required package: evd
```

```
##
## Attaching package: 'evd'
```

```
## The following object is masked from 'package:lattice':
##
##     qq
```

```
expplot(width(betweenmotifs), rate = 1/mean(width(betweenmotifs)),
        labels = "fit")
```



2.10.1 Modeling in the case of dependencies

```
library("BSgenome.Hsapiens.UCSC.hg19")
chr8  =  Hsapiens$chr8
```

```
CpGtab = read.table(here("Data","model-based-cpg-islands-hg19.txt"),
                    header = TRUE)
nrow(CpGtab)
```

```
## [1] 65699
```

```
head(CpGtab)
```

```
##     chr  start    end length CpGcount GCcontent pctGC obsExp
## 1 chr10  93098  93818    721       32       403 0.559  0.572
## 2 chr10  94002  94165    164       12        97 0.591  0.841
## 3 chr10  94527  95302    776       65       538 0.693  0.702
## 4 chr10 119652 120193    542       53       369 0.681  0.866
## 5 chr10 122133 122621    489       51       339 0.693  0.880
## 6 chr10 180265 180720    456       32       256 0.561  0.893
```

```
irCpG = with(dplyr::filter(CpGtab, chr == "chr8"),
         IRanges(start = start, end = end))
grCpG = GRanges(ranges = irCpG, seqnames = "chr8", strand = "+")
genome(grCpG) = "hg19"
```

ALERT!!: And this is where it all goes sideways because I can't get Gviz to work. I was able to download it off the BioConductor site but then it won't appear in the packages and it gives errors if I try to load it.

```
{r} # library("Gviz") # ideo = IdeogramTrack(genome = "hg19",
chromosome = "chr8") # plotTracks( #   list(GenomeAxisTrack(),
#     AnnotationTrack(grCpG, name = "CpG"), ideo), #     from
= 2200000, to = 5800000, #     shape = "box", fill = "#006400",
stacking = "dense") #
```

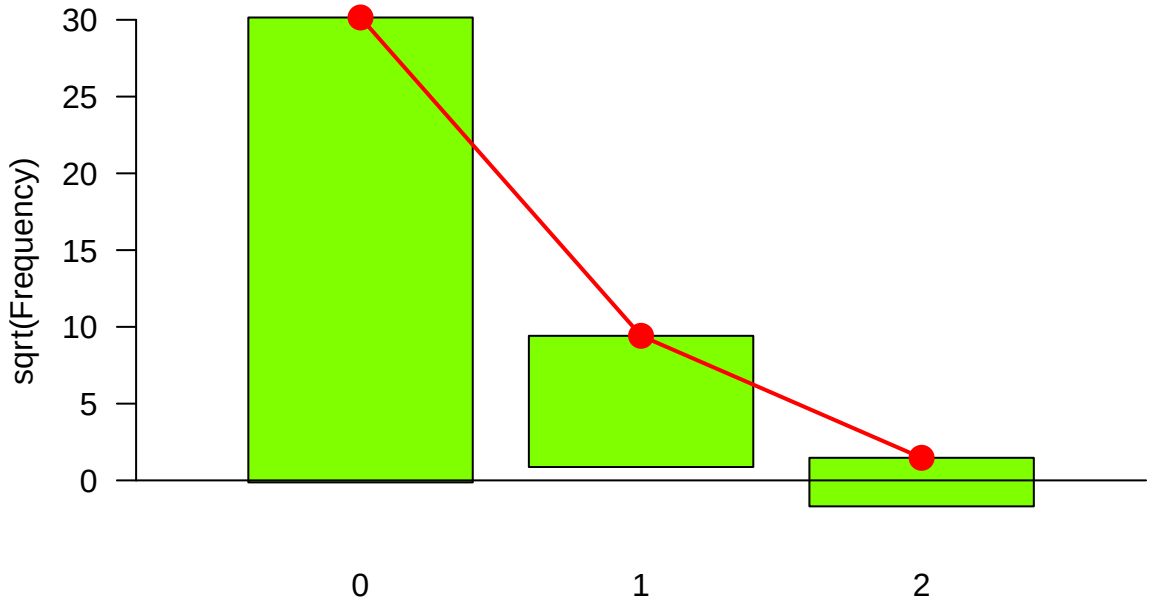
2.13 EXercises

2.1

```
m<-rbinom(1000,1000,0.0001)
gfm = goodfit(m, "binomial")
```

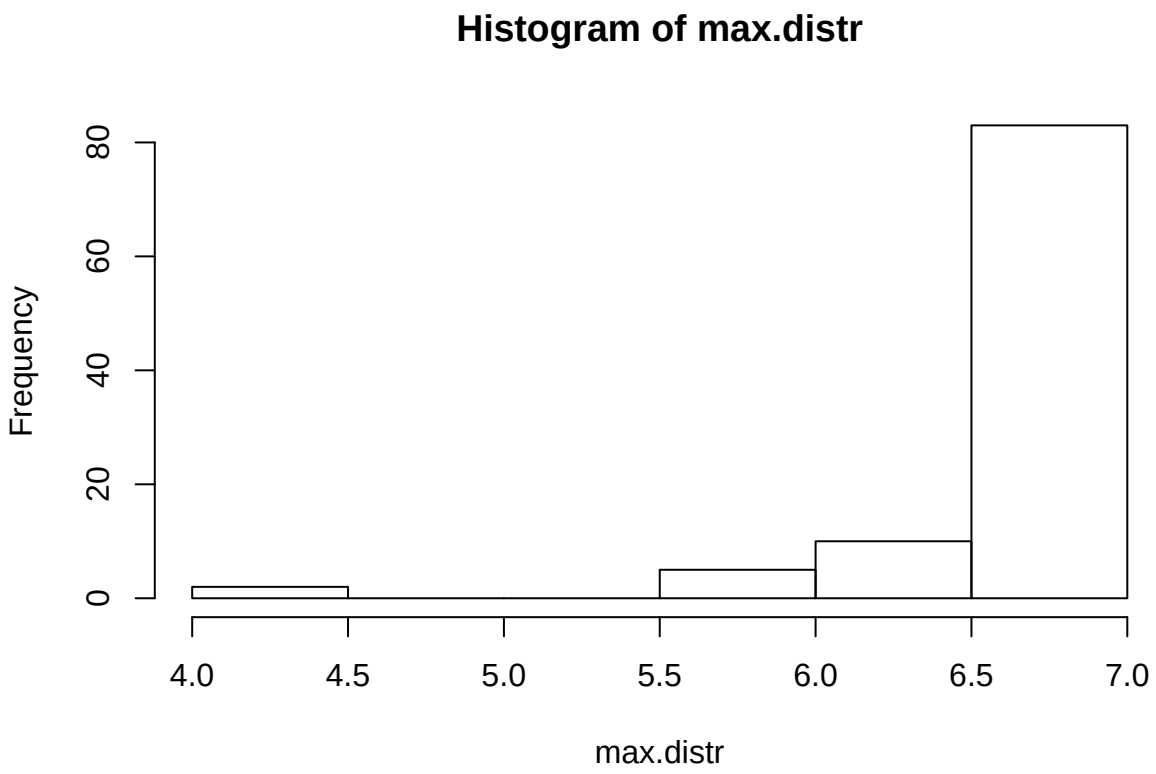
```
## Warning in goodfit(m, "binomial"): size was not given, taken as maximum
## count
```

```
rootogram(gfm, xlab = "", rect_gp = gpar(fill = "chartreuse"))
```



2.2

```
max.unif<-function(n){
  max(runif(n,0,7))
}

max.distr<-apply(as.matrix(seq(1:100)),1,max.unif)
hist(max.distr)
```




```
likelihoodunif<-function(theta,n=25){
  theta^n
}

thetas<-seq(from = 0, to = 7, by=0.001)
likelihood<-apply(as.matrix(thetas),1,likelihoodunif)
max.likelihood<-thetas[which.max(likelihood)]
max.likelihood
```

```
## [1] 7
```

The likelihood function is:

$$P(\theta|x_1, x_2, ..., x_n) = P(X_1 < \theta)P(X_2 < \theta) \times ... \times P(X_n < \theta) = \left(\frac{\theta}{7}\right)^n$$

As this is an increasing function in θ on the range (0,7), it is maximized at 7. So $\hat{\theta} = 7$.

2.3

```
mtb = read.table(here("Data","M_tuberculosis.txt"), header = TRUE)
head(mtb, n = 4)
```

```
##   AmAcid Codon Number PerThous
## 1    Gly   GGG  25874    19.25
## 2    Gly   GGA  13306     9.90
## 3    Gly   GGT  25320    18.84
## 4    Gly   GGC  68310    50.82
```

```
pro  =  mtb[ mtb$AmAcid == "Pro", "Number"]
pro/sum(pro)
```

```
## [1] 0.54302025 0.10532985 0.05859765 0.29305225
```

```
#a
table(mtb$AmAcid)
```

```
##
## Ala Arg Asn Asp Cys End Gln Glu Gly His Ile Leu Lys Met Phe Pro Ser Thr
##   4   6   2   2   2   3   2   2   4   2   3   6   2   1   2   4   6   4
## Trp Tyr Val
##   1   2   4
```

```
table(mtb$Codon)
```

```
##
## AAA AAC AAG AAT ACA ACC ACG ACT AGA AGC AGG AGT ATA ATC ATG ATT CAA CAC
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
## CAG CAT CCA CCC CCG CCT CGA CGC CGG CGT CTA CTC CTG CTT GAA GAC GAG GAT
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
## GCA GCC GCG GCT GGA GGC GGG GGT GTA GTC GTG GTT TAA TAC TAG TAT TCA TCC
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
## TCG TCT TGA TGC TGG TGT TTA TTC TTG TTT
##   1   1   1   1   1   1   1   1   1   1
```

(b) Percentage of total number multiplied by 1000

(c)

```
chisq<-function(x,p=rep(1/length(x))){
  ((x-1000*p)/(1000*p))^2
}
stats<-apply(as.matrix(mtb$PerThous),1,chisq)
maxstat<-as.character(mtb[which.max(stats),1])
maxstat
```

```
## [1] "End"
```

2.4

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

```
##   1362-letter "DNAString" instance
## seq: ATGTCGGAAAAAGAAATTTGGGAAAAAGTGCTTG...GAGAATCTTGAAAAAGAAATAAGAAATGTATAA
```

```
staph[[2]]
```

```
##   1134-letter "DNAString" instance
## seq: ATGATGGAATTCACTATTAAAAGAGATTATTTTA...ACGCAATTAATTTTACCAATCAGAACTTACTAA
```

```
staph[[3]]
```

```
##   246-letter "DNAString" instance
## seq: GTGATTATTTTGGTTCAAGAAGTTGTAGTAGAAG...TCTTTCTTAATCATTCATCAAGGTGAACAATGA
```

(b)

table(numMatches) ### 2.5