Statistical Modeling

Brandon Kozak

2019-10-09

```r
thetas = seq(0, 1, by = 0.001)
library(here)
library(tidyverse)
library(BiocManager)
library(Biostrings)
library(BSgenome.Celegans.UCSC.ce2)
library(seqLogo)
```

Intro

Key idea here is that we do not know the underlying distributions nor the values of the parameters. Most of what we will do in this chapter is known as statistical inference. In particular, we will use the data itself to make estimations for the parameters we seek.

Goals

- Difference between probability and statistics.
- Use visualization to fit data to probability distributions.
- Use maximum likelihood
- Bayesian estimation
- Evaluate dependencies in binomial and multinomial distributions
- Look at genomic data assembled into tables
- Markov chain models for dependent data
- Concrete applications

2.2 statistical and probabilistic models

In chapter one we dealt with probabilistic models, that is we knew/assumed the distributions of our problems.

In a statistical model, we do not know the distribution and must use the data to obtain parameter estimates.

Again, the idea of generating parameter estimates is called **Statistical inference**.

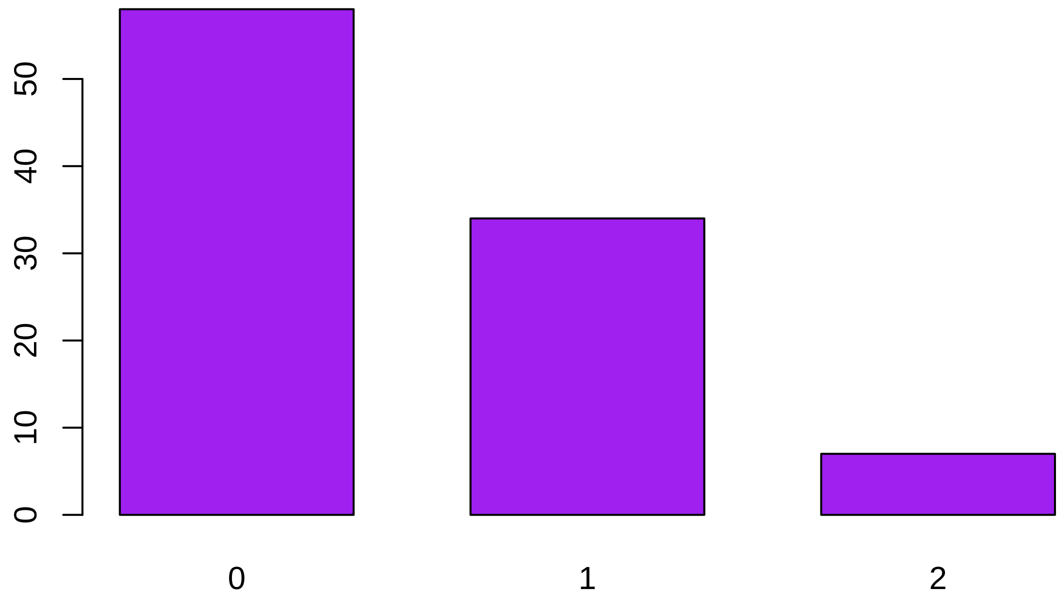
2.3 Our first example

Lets load in the data from the epitope example, minus the outlier.

```r
load(here::here("Data","e100.RData"))
# Remove outlier
e99 = e100[-which.max(e100)]
```

Our first approach is to visualize the raw data:

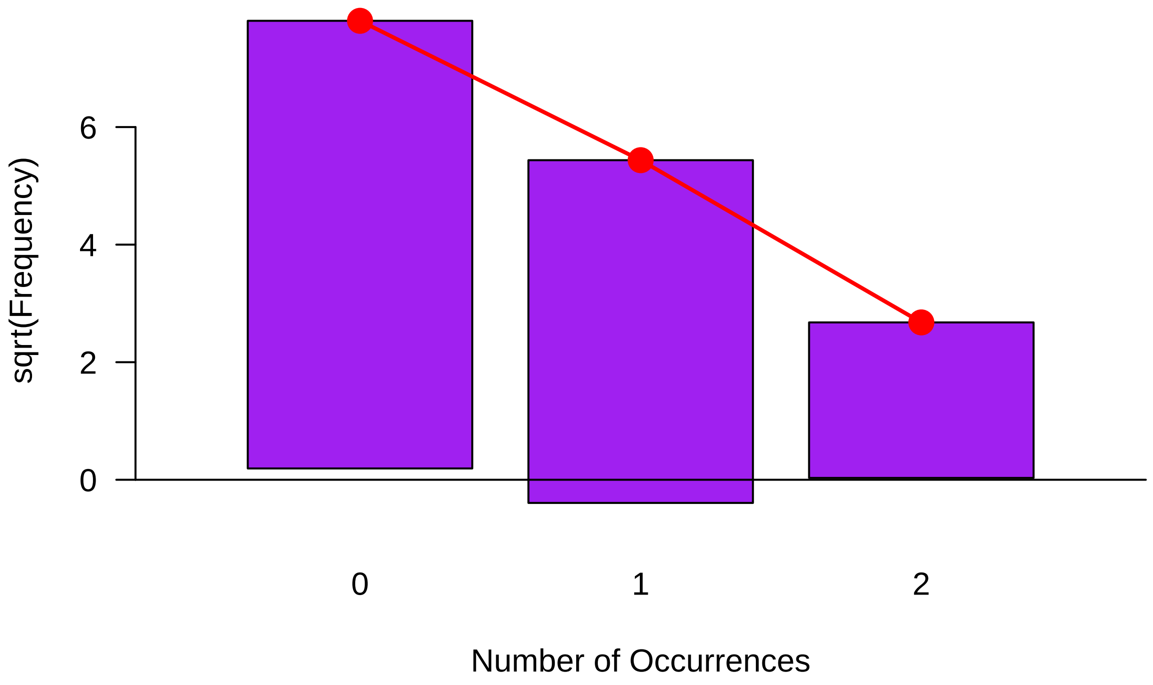
```r
barplot(table(e99), space = .5 , col = "purple")
```



However, we need something to compare this too. One visual goodness-of-fit test is known as the rootogram.

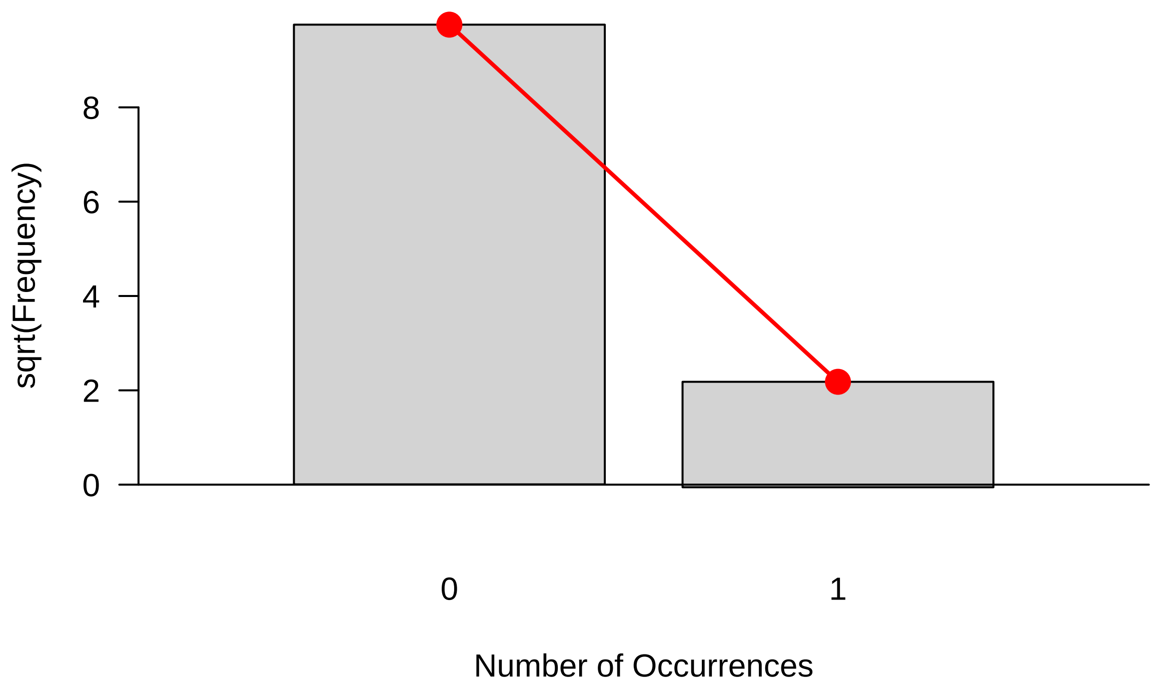
The idea being that for each observed value (bar), it is hanged by a red dot up to the theoretical value. If a sample matches a certain distribution perfectly, then all the bars will line up on the x-axis.

```r
library("vcd")
e99_good_fit <- goodfit(e99, "poisson")
rootogram(e99_good_fit, rect_gp = gpar(fill = "purple"))
```



What would a rootogram look like if the data actually came from a pois dist?

```
rootogram( goodfit( rpois(100,lambda=.05) ,"poisson"))
```



So we have a good idea that our data comes from a pois dist, but we still need it's parameter λ. To do this we use a method called **maximum likelihood estimation** (MLE), where we pick a value $\hat{\lambda}$ that makes the observed data most likely.

To do this, lets return back to the original data

```
table(e100)
```

```
## e100
##  0  1  2  7
## 58 34  7  1
```

We need to answer the question "What value of λ will give us the highest change of generating this data?"

Could try trial and error…

```
table(rpois(100,1))
```

```
##
##  0  1  2  3  4
## 45 33 19  2  1
```

```
table(rpois(100,2.5))
```

```
##
##  0  1  2  3  4  5  6
##  9 20 33 16 10  7  5
```

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

```
##
##  0  1  2  3  4  5  6  7  8
##  3 15 14 25 21 10  8  3  1
```

Or we could do a lil math.

We can find: $P(58\ zeros, 34\ ones, 7\ twos, 1\ seven \mid Data\ follows\ Poisson(\lambda)) = P(0)^{58} \times P(1)^{34} \times P(2)^7 \times P(7)^1$

And then compare for various values of $lambda$

```
prod(dpois(c(0,1,2,7), lambda = .4) ^ (c(58,34,7,1)))
```

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

```
prod(dpois(c(0,1,2,7), lambda = 1) ^ (c(58,34,7,1)))
```

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

```
prod(dpois(c(0,1,2,7), lambda = .55) ^ (c(58,34,7,1)))
```

```
## [1] 1.057087e-44
```

If we want to obtain actual answer, we would use the likelihood function $\prod_{i=1}^{n} f(k_i)$

However, we often take the log of this function (since it is strictly increasing).

Once we do this, out goal is to maximize this function with respect to our parameter.

In R we could write our own function to plot many values of λ as above, or we could just use the function goodfit()

```
e100_good_fit <- goodfit(e100, "poisson")

e100_good_fit$par
```

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

2.3.1

Proof of the above, I've done this before, but I don't think this will be covered in detail for this class.

2.4 Binomial distribution and maximum likelihood

It's often that we have a good feeling that our data comes from a binomial distribution, but we do not know what the probability is.

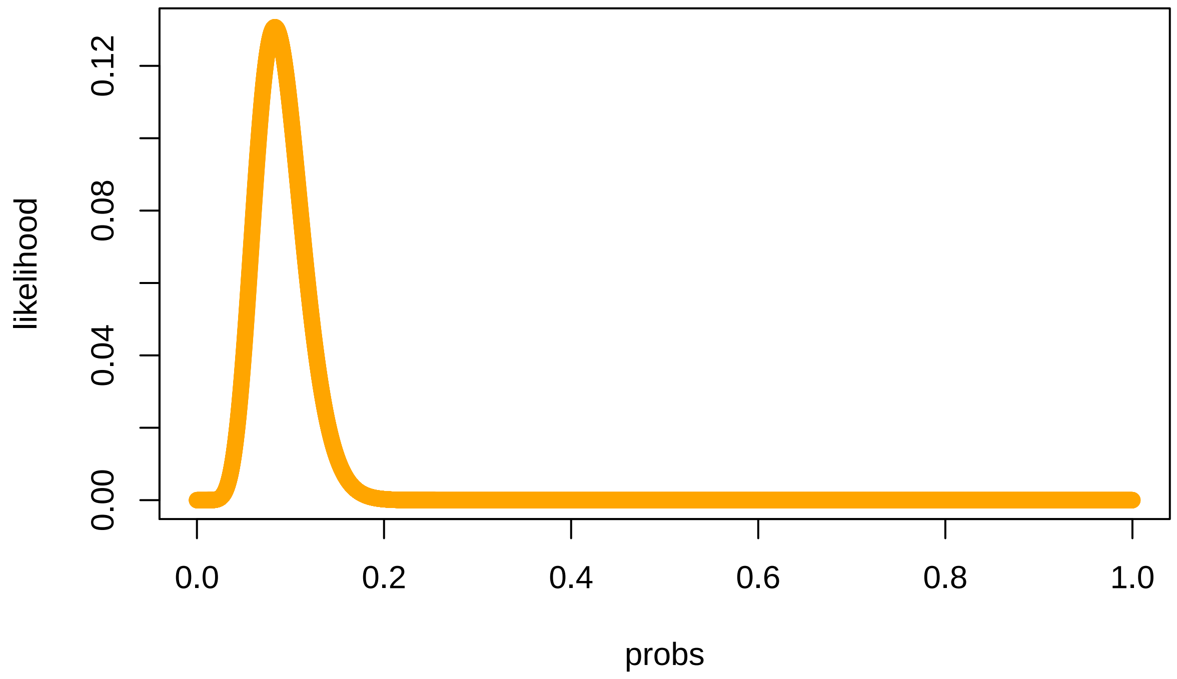
Data we have a sample of 120 male and test them for colorblindness. We say 0 means no colorblindness and 1 means colorblindness.

```
cb = c(rep(0,10),rep(1,110))
```

What is the value of p that would most likely generate this data under a binomial(120,p)?

Our intuition say $\frac{10}{120} = \frac{1}{12}$ and in this case, it so happens to be the MLE, however it might not always be the case the the MLE is as straight forward to obtain.

```
probs = seq(0,1,by=.0001)

likelihood = dbinom(10, prob = probs, size = length(cb))

plot(probs, likelihood, col = "orange")
```



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

```
## [1] 0.0833
```

2.5 Multinomial Data

2.5.1 Intro to DNA data

Four basic molecules of DNA:

A - adenine C - cytosine G - guanine T - thymine

2 groups, purines (A,G), and pyrimidines (C,T)

Nucletide bias

read in DNA data using biostrings

```
library("Biostrings")

staph = readDNAStringSet(here::here("Data", "staphsequence.ffn.txt"), "fasta")
```

Look at first gene

```
staph[1]
```

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

Can look at frequency via letterFrequency()

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

Chi-squared test:

```
chisq.test(staph_freq)
```

```
##
##  Chi-squared test for given probabilities
##
## data:  staph_freq
## X-squared = 184.4, df = 3, p-value < 2.2e-16
```

Could use Monte carol simulation to see if the first 10 genes come from a different mutinomial dist, but we can do it without.

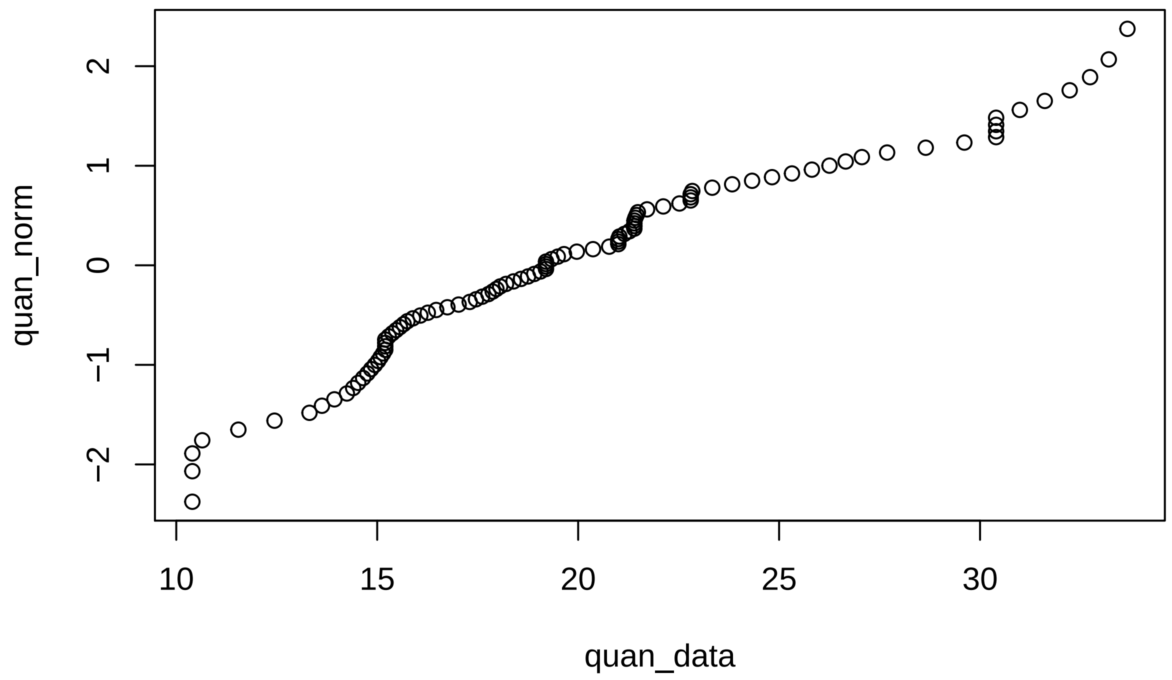
2.6 chi-squared

Quantiles and the quantile-quantile plot

QQ plots are useful for checking distribution assumptions.

Mostly used when we want to verify normality of our data.

```
qs = ppoints(100)

mpg = mtcars$mpg

quan_data = quantile(mtcars$mpg,qs)

quan_norm = quantile(qnorm(qs),qs)

plot(quan_data,quan_norm)
```



2.7 Chargaff's Rule

Chargaff's rule states that DNA from any cell should have a 1:1 ratio of pyrimidine and purine.

Thus %A = %T and %G = %C

2.7.1 Two categorical variables

Pure categorical data can be stored in a table where each cell is the number of observations of that particular combination.

Such a table is called a **contingency table**.

```
#ie.

HairEyeColor[,,"Male"]
```

```
##        Eye
## Hair    Brown Blue Hazel Green
##   Black    32   11    10     3
##   Brown    53   50    25    15
##   Red      10   10     7     7
##   Blond     3   30     5     8
```

```r
# can also do chi-squared test

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

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 Hardy-Weinberg equilibrium

Consider two alleles, M and N.

Suppose that the overall frequency of M in the population is p, so that N is q = 1-p.

HWE looks at the relationship between p and q and if there s independence of the frequency of both alleles in a genotype.

The probabilities of the genotypes are

$P_{MM} = p^2, P_{NN} = q^2, P_{MN} = 2pq$

Let $S = n_{MM} + n_{NN} + n_{MN}$

Then the above follows a $Muilt_N(S, (P_{MM}, P_{NN}, P_{MN}))$

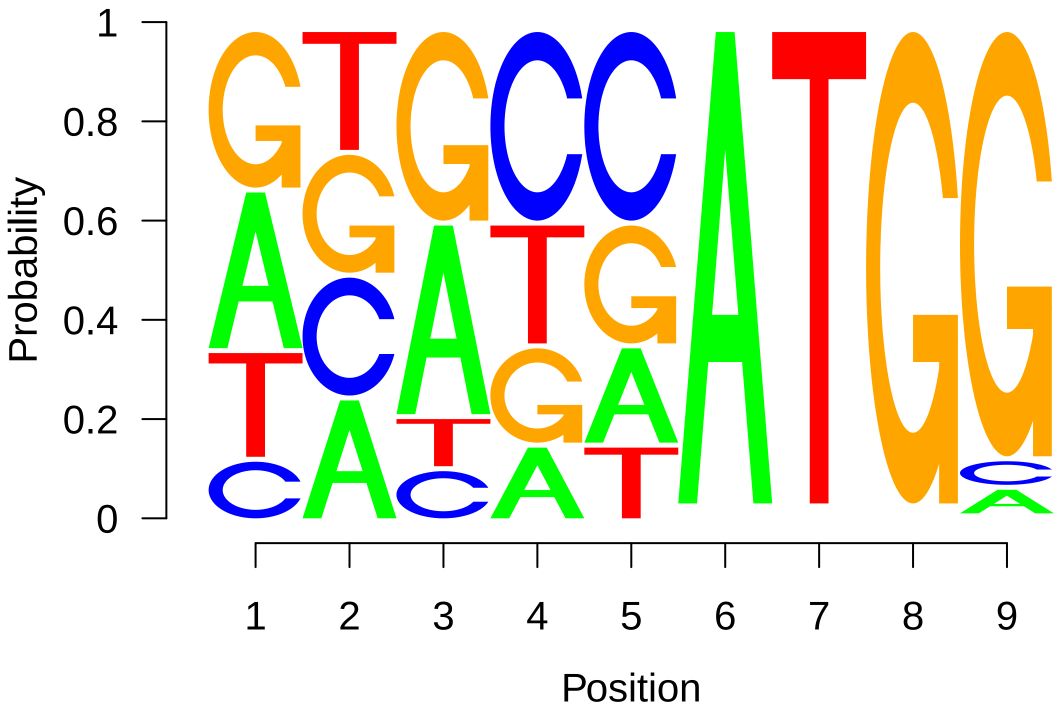
and it can be shown that the MLE of p is:

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

2.7.3 sequence motifs and logos

No idea about this…

```r
load(here::here("Data", "kozak.RData"))

pwm = makePWM(kozak)

seqLogo(pwm, ic.scale = FALSE)
```



2.8 Markov Chains

Used to generate certain probabilities under a system of dependent probabilities.

$P(CA) = P(C)P(A|C)$

Wish the book gave interactive examples, Markov Chains are fun!

2.9 Bayesian Thinking

Main idea is to use previous information, or as my profs like to put it "play god."

Info could be certain restrictions on parameters.

prior and posterior distributions.

Prior probability, written as $P(H)$

Posterior probability, written as $P(H|Data)$

Haplotypes, a example

Haplotype is a collection of alleles that are spatially adjacent

Haplotype frequencies

```r
haplo6 <- read.table(here::here("data","haplotype6.txt"),header = T)

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
```

Simulation Study

Since we want to use prior knowledge, the Bayesian view allows us to assuming our data is being drawn from a entire distribution, rather than focusing on one parameter.

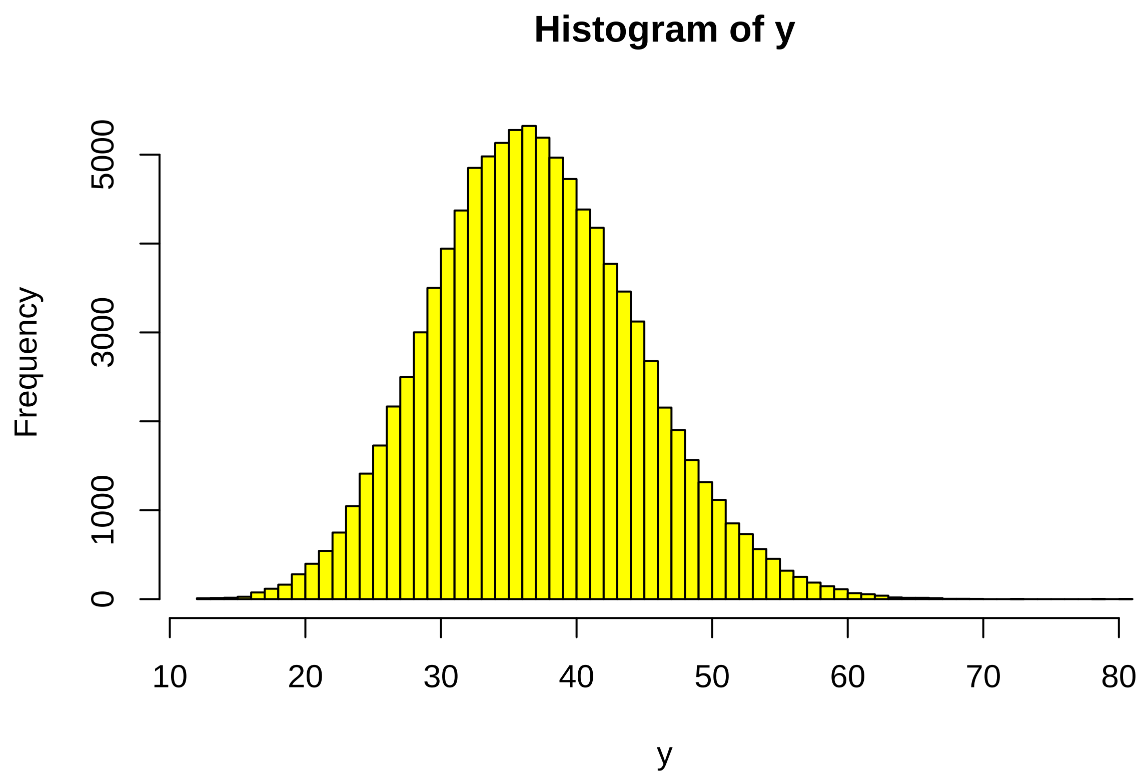
Since we are interested in looking at a proportion, we would like a dist. that has its domain defined on (0,1).

It just so happens that the Beta dist. has this property!

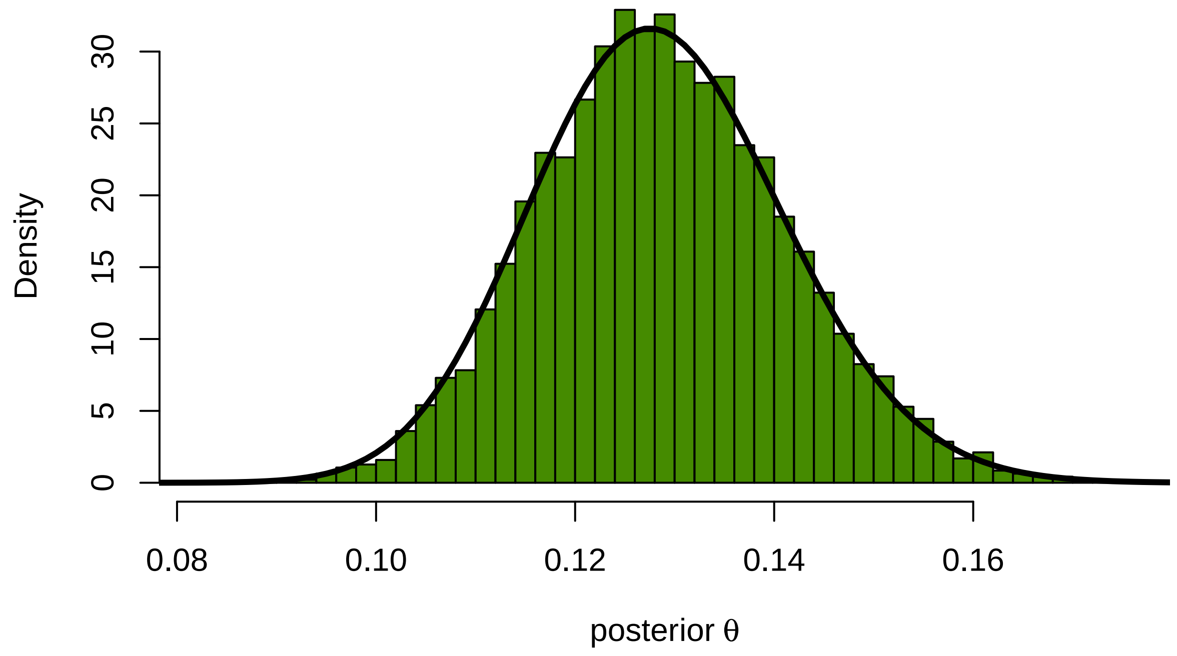
Nice mathematical property as well: if we start with a prior belief on θ that happens to be beta-shaped, then observe a data set of n binomial trials, and then update our belief, the posterior dist. on θ will also have a beta-shape.

Marginal distribution of Y

```r
para = rbeta(100000, 50, 350)

y = vapply(para, function(th){ rbinom(1, prob = th, size = 300)}, numeric(1))

hist(y, breaks = 50, col="yellow")
```



```
thetaPostEmp = para[ 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)
```



2.10 Nucleotide pattern in a genome

Modeling in the case of dependencies

Going back to Markov chains!

Not 100% sure about the data though.

CpG Islands, tells us the start and end points of a island. we are interested to look at the frequencies of nucleotides and of the diagrams 'CG', 'CT', 'CA', 'CC'. Our question is "Are there any dependencies between the nucleotide occurrences and if so, how to model them?"

```r
library("BSgenome.Hsapiens.UCSC.hg19")

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

Here we do some data cleaning, we create an "IRanges" S4 object that only includes chromosome 8.

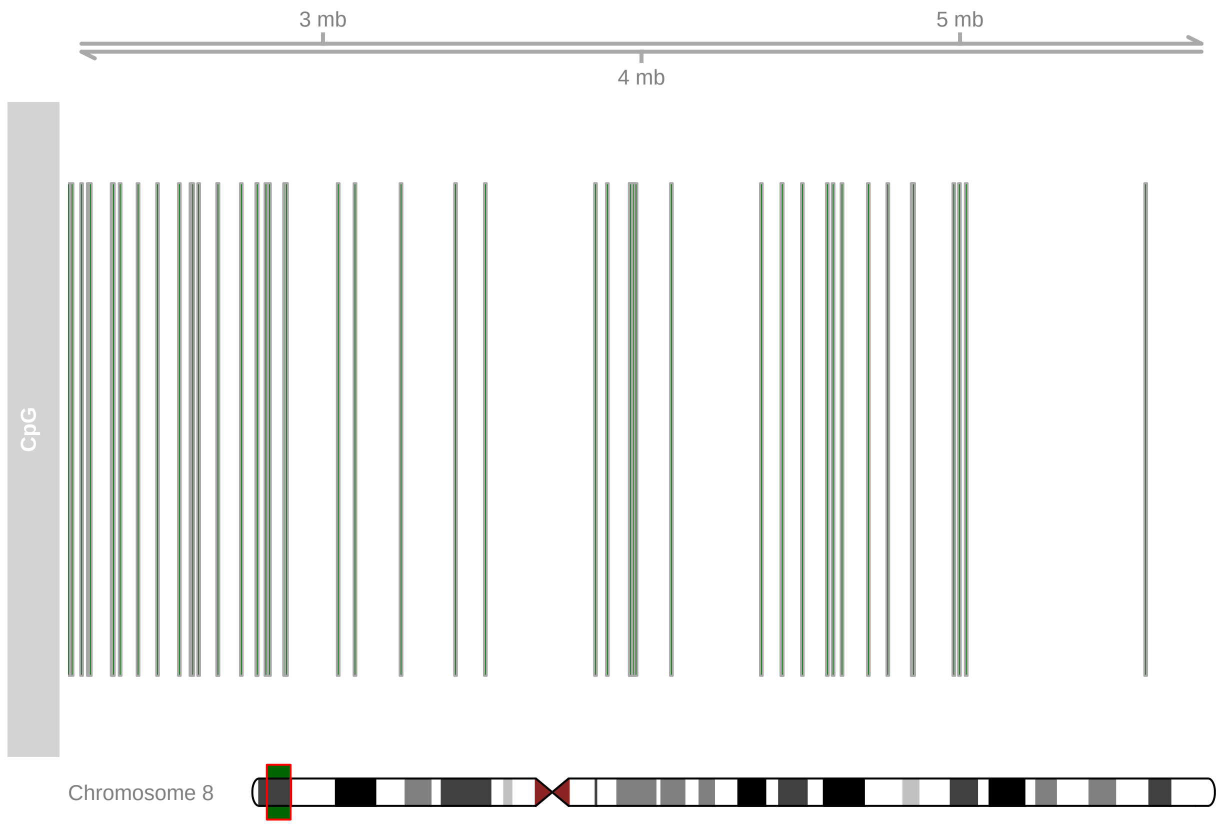
More info: https://bioconductor.org/packages/release/bioc/vignettes/IRanges/inst/doc/IRangesOverview.pdf

```r
# Filter and construct the IRanges object
irCpG = with(dplyr::filter(CpGtab, chr == "chr8"),
         IRanges(start = start, end = end))
```

```
# Add names to columns to introduce biological context
grCpG = GRanges(ranges = irCpG, seqnames = "chr8", strand = "+")
genome(grCpG) = "hg19"
```

A Gviz plot, not quite sure what this tells us.

```
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")
```



Now we create "NonCGIview" objects. They contain the coordinates rather than the sequence itself.

```
# XStringViews objects
CGIview    = Views(unmasked(Hsapiens$chr8), irCpG)
NonCGIview = Views(unmasked(Hsapiens$chr8), gaps(irCpG))
```

We compute transition counts in CpG islands and non-islands using the data.

Very confusing, converting our "NonCGIview" objects to "DNAStringSet", but why??

Then we use the function dinucleotideFrequency() over a call to sapply!?

Then we take the sum of the rows of dinucCpG and dinucNonCpG.

```r
seqCGI      = as(CGIview, "DNAStringSet")
seqNonCGI   = as(NonCGIview, "DNAStringSet")
dinucCpG    = sapply(seqCGI, dinucleotideFrequency)
dinucNonCpG = sapply(seqNonCGI, dinucleotideFrequency)
dinucNonCpG[, 1]
```

```
##  AA  AC  AG  AT  CA  CC  CG  CT  GA  GC  GG  GT  TA  TC  TG  TT
## 389 351 400 436 498 560 112 603 359 336 403 336 330 527 519 485
```

```r
NonICounts = rowSums(dinucNonCpG)
IslCounts  = rowSums(dinucCpG)
```

Now we creates the transition matrix, right now though, it is telling us how many times we went from state "x" to state "y"

ie.) we went from state A to state G 122131 times.

```r
TI  = matrix( IslCounts, ncol = 4, byrow = TRUE)
TnI = matrix(NonICounts, ncol = 4, byrow = TRUE)
dimnames(TI) = dimnames(TnI) =
  list(c("A", "C", "G", "T"), c("A", "C", "G", "T"))
```

Now we divide by row sums to get the probabilities of going to state "y" given that we start in state "x"

```r
MI = TI /rowSums(TI)
MI
```

```
##            A         C         G         T
## A 0.20457773 0.2652333 0.3897678 0.1404212
## C 0.20128250 0.3442381 0.2371595 0.2173200
## G 0.18657245 0.3145299 0.3450223 0.1538754
## T 0.09802105 0.3352314 0.3598984 0.2068492
```

Exercises

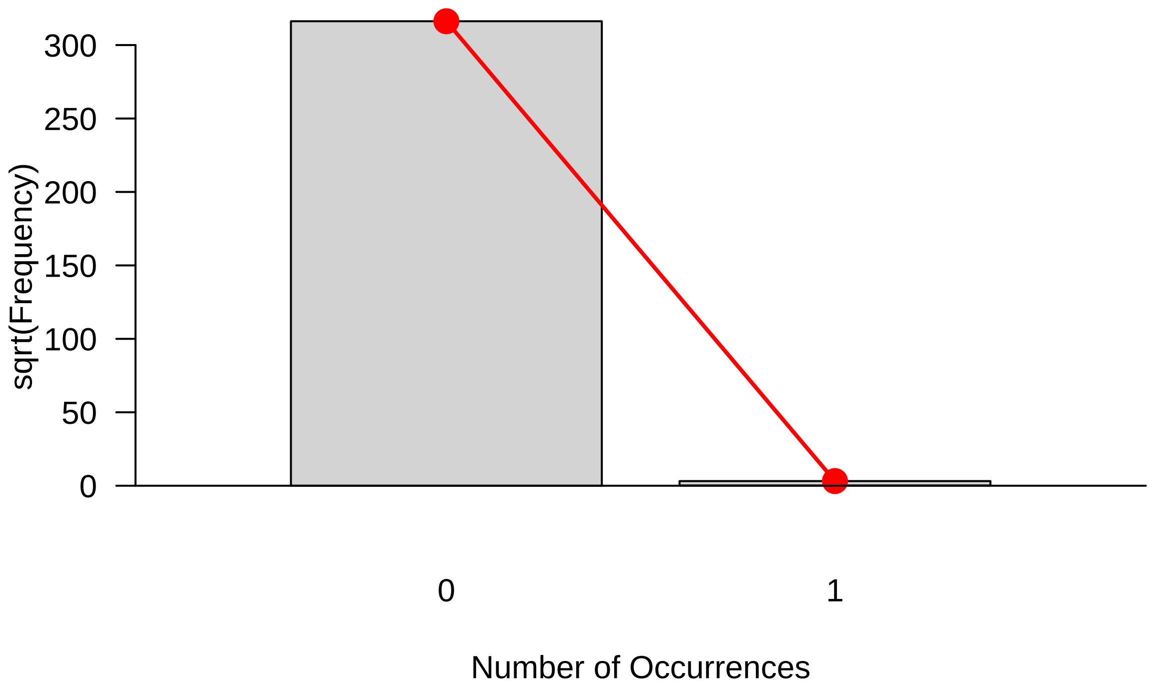
2.1

```r
dist = function(arg=NULL){

return( runif(1)<10^-4 )

}

set.seed(1234)

data =  as.numeric(replicate(100000,dist()))
sum = sum(data)

#Safe bet to say this came from a bin(1000,10^-4)
```

```r
good_fit <- goodfit(data, type="binomial", par=list(prob = 10^-4))
```

```
## Warning in goodfit(data, type = "binomial", par = list(prob = 10^-4)): size
## was not given, taken as maximum count
```

```r
rootogram(good_fit)
```

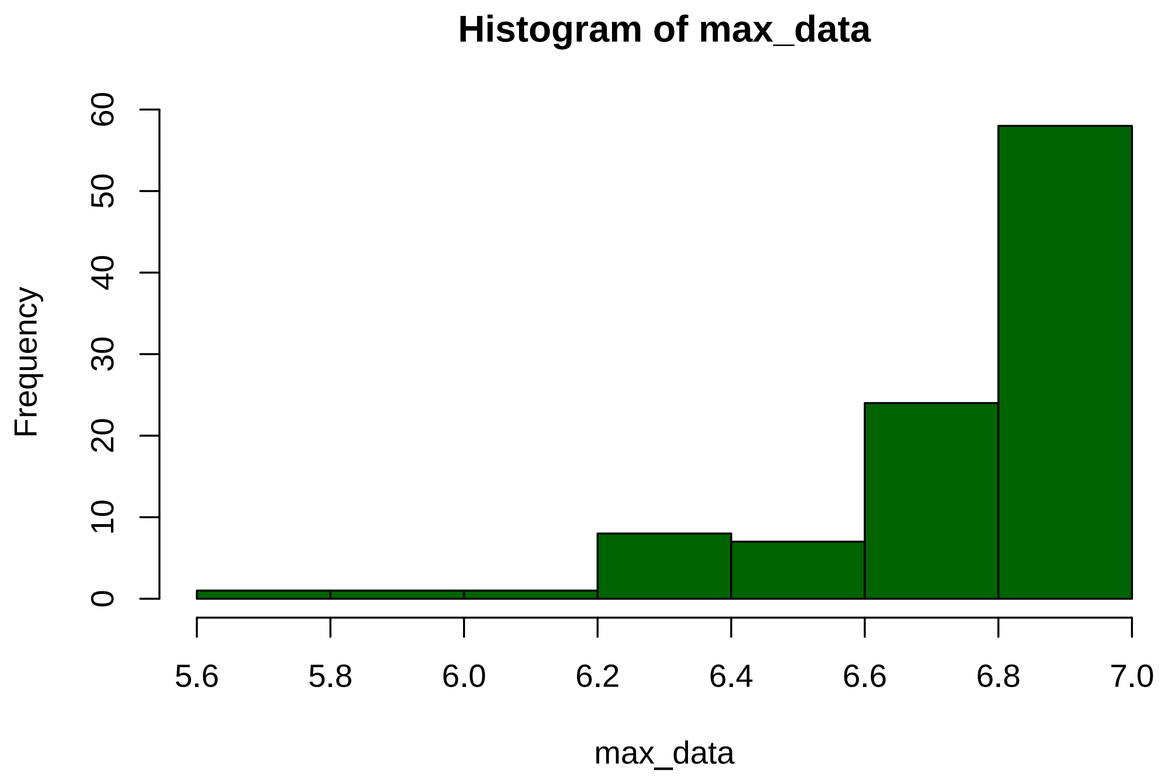


2.2

```r
max_unif = function(n){
  max((runif(n,0,7)))
}
max_unif(25)
```

```
## [1] 6.926803
```

```r
max_data = replicate(100,max_unif(25))

hist(max_data, col="darkgreen")
```



Not really sure what the question is asking here, the whole point of MLE is to estimate a parameter, but since we are generating uniform numbers from 0 to 7, we know that the dist is Unif(a=0, b=7). Certainly if b was unknown, then taking the max would be a good estimate for b (turns out it is a minimal sufficient statistic of b). But the max is NOT a parameter.

2.3

```r
mtb = read.table(here::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
```

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

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

```r
# a.)
# create tables for codon and AmAcid
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
```

```r
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
```

```r
# can also create a data set that just looks at the two
mtb_select <- mtb %>% select(Codon,AmAcid)
mtb_select
```

```
##    Codon AmAcid
## 1    GGG    Gly
## 2    GGA    Gly
## 3    GGT    Gly
## 4    GGC    Gly
## 5    GAG    Glu
## 6    GAA    Glu
## 7    GAT    Asp
## 8    GAC    Asp
## 9    GTG    Val
## 10   GTA    Val
## 11   GTT    Val
## 12   GTC    Val
## 13   GCG    Ala
## 14   GCA    Ala
## 15   GCT    Ala
## 16   GCC    Ala
## 17   AGG    Arg
## 18   AGA    Arg
## 19   AGT    Ser
## 20   AGC    Ser
## 21   AAG    Lys
## 22   AAA    Lys
## 23   AAT    Asn
## 24   AAC    Asn
## 25   ATG    Met
## 26   ATA    Ile
## 27   ATT    Ile
## 28   ATC    Ile
## 29   ACG    Thr
## 30   ACA    Thr
## 31   ACT    Thr
## 32   ACC    Thr
## 33   TGG    Trp
## 34   TGA    End
## 35   TGT    Cys
## 36   TGC    Cys
## 37   TAG    End
## 38   TAA    End
## 39   TAT    Tyr
## 40   TAC    Tyr
## 41   TTG    Leu
## 42   TTA    Leu
## 43   TTT    Phe
## 44   TTC    Phe
## 45   TCG    Ser
## 46   TCA    Ser
## 47   TCT    Ser
## 48   TCC    Ser
## 49   CGG    Arg
## 50   CGA    Arg
## 51   CGT    Arg
## 52   CGC    Arg
## 53   CAG    Gln
## 54   CAA    Gln
## 55   CAT    His
## 56   CAC    His
## 57   CTG    Leu
## 58   CTA    Leu
## 59   CTT    Leu
## 60   CTC    Leu
## 61   CCG    Pro
## 62   CCA    Pro
## 63   CCT    Pro
## 64   CCC    Pro
```

```
# b.) No idea, but lets try to reverse engineer

mtb_mutate <- mtb %>% mutate(test_var = Number/PerThous)



# So we are dividing by a constant? 1344?

#c.) Not sure what they mean by "possible spellings"
```

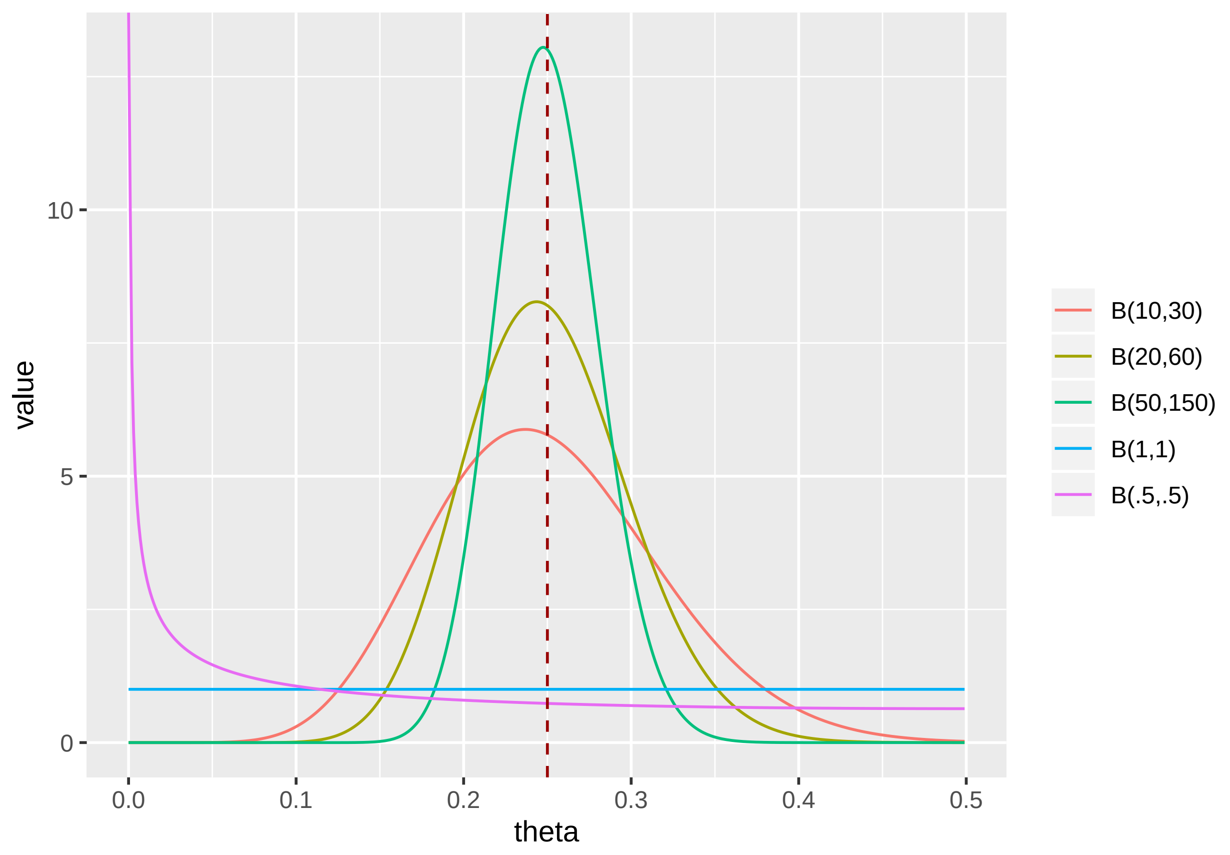
2.4

```r
staph = readDNAStringSet(here::here("Data","staphsequence.ffn.txt"),"fasta")

# a.)
# First 3 sequences in the set.
staph_1_to_3 = staph[1:3,]


# b.)
b = letterFrequencyInSlidingView(staph[[1]], view.width = 100, letters = "GC")

# c.) idk, not sure what they mean by sliding window
c = letterFrequencyInSlidingView(staph[[1]], view.width = 100, letters = "GC")/100


c[1:5]
```

```
## [1] 0.32 0.33 0.34 0.33 0.33
```

2.5

```r
## ----histobeta2, fig.keep = 'high', fig.cap = "Beta distributions with $\\alpha=10,20,50$ and $\\beta



theta = thetas[1:500]
dfbetas = data.frame(theta,
           db1= dbeta(theta,10,30),
           db2 = dbeta(theta, 20, 60),
           db3 = dbeta(theta, 50, 150),
           db4 = dbeta(theta, 1, 1),
           db5 = dbeta(theta, .5, .5))


require(reshape2)
datalong  =  melt(dfbetas, id="theta")
ggplot(datalong) +
geom_line(aes(x = theta,y=value,colour=variable)) +
theme(legend.title=element_blank()) +
geom_vline(aes(xintercept=0.25), colour="#990000", linetype="dashed")+
scale_colour_discrete(name  ="Prior",
                          labels=c("B(10,30)", "B(20,60)","B(50,150)",
                                   "B(1,1)","B(.5,.5)"))
```

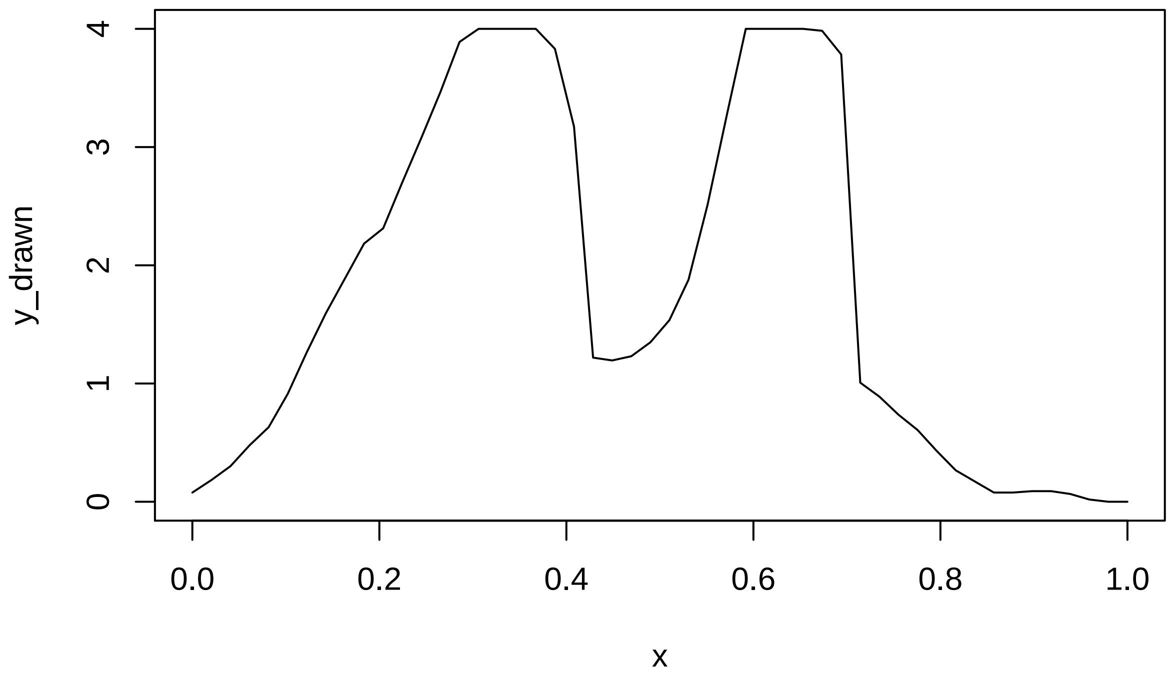


2.6

```
draw = read_csv(here::here("Data","my_drawing.csv"))
```

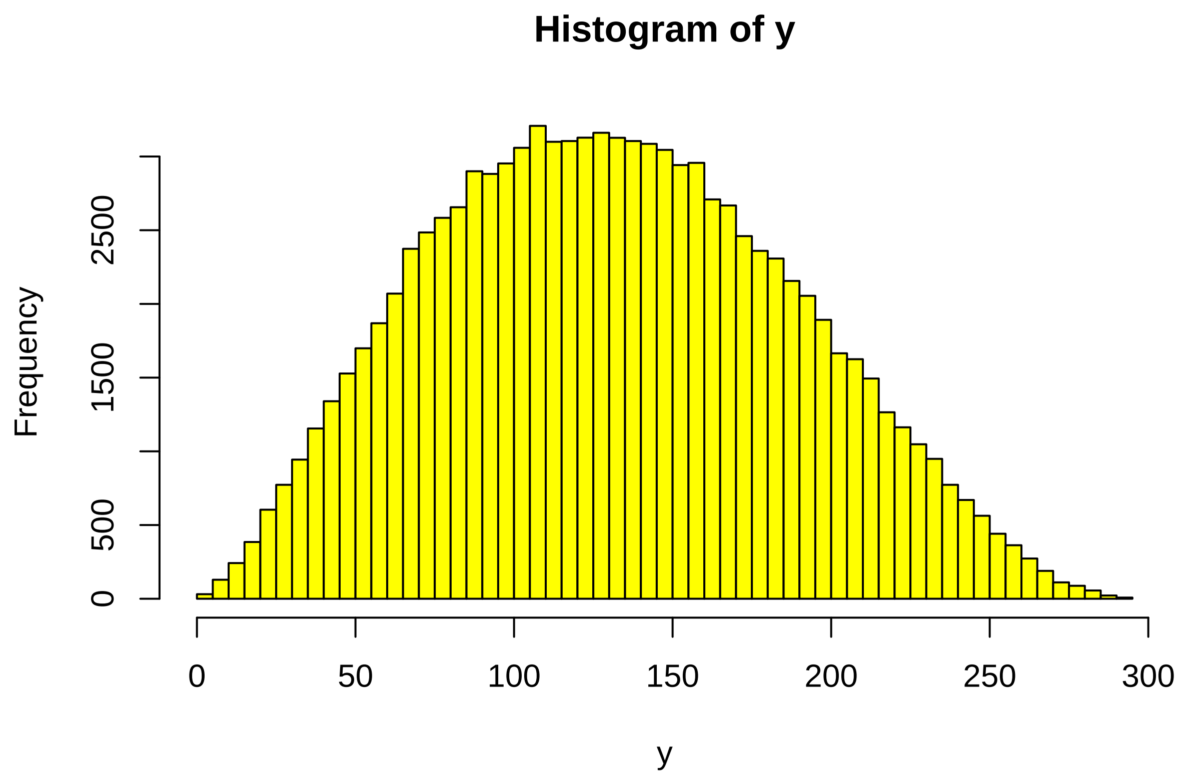
```
## Warning: Missing column names filled in: 'X1' [1]
```

```
plot(draw[,2:3],type="l")
```

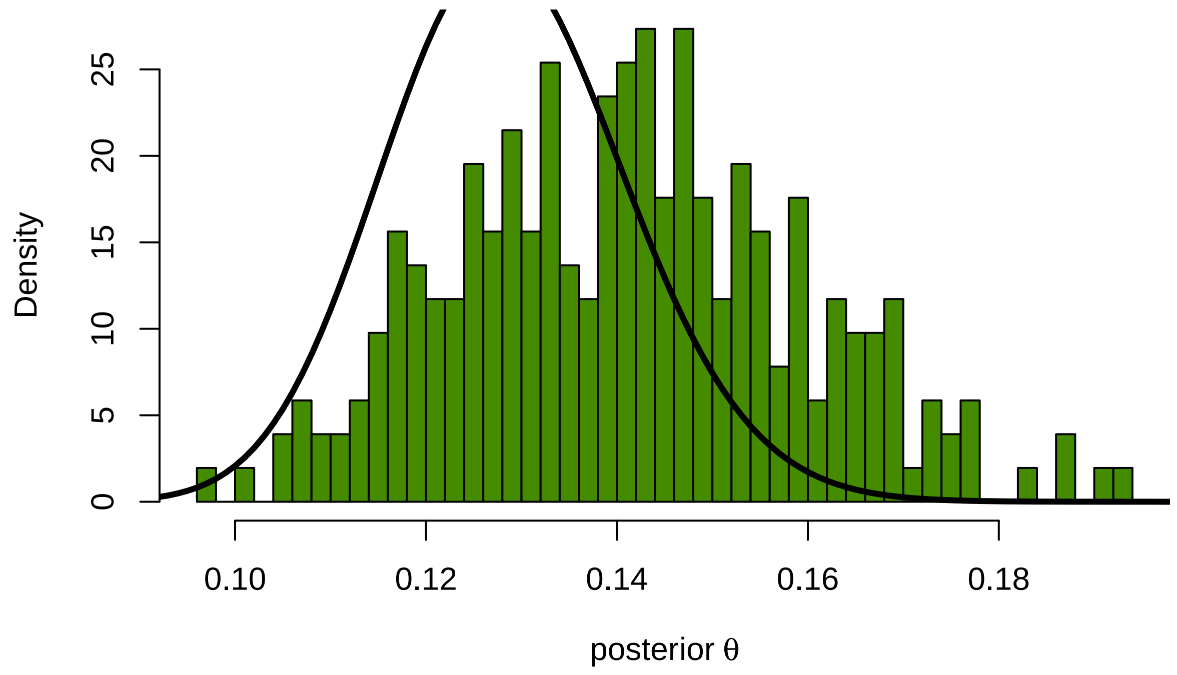


```r
# Best approx is a=2.74, b=3.47
```

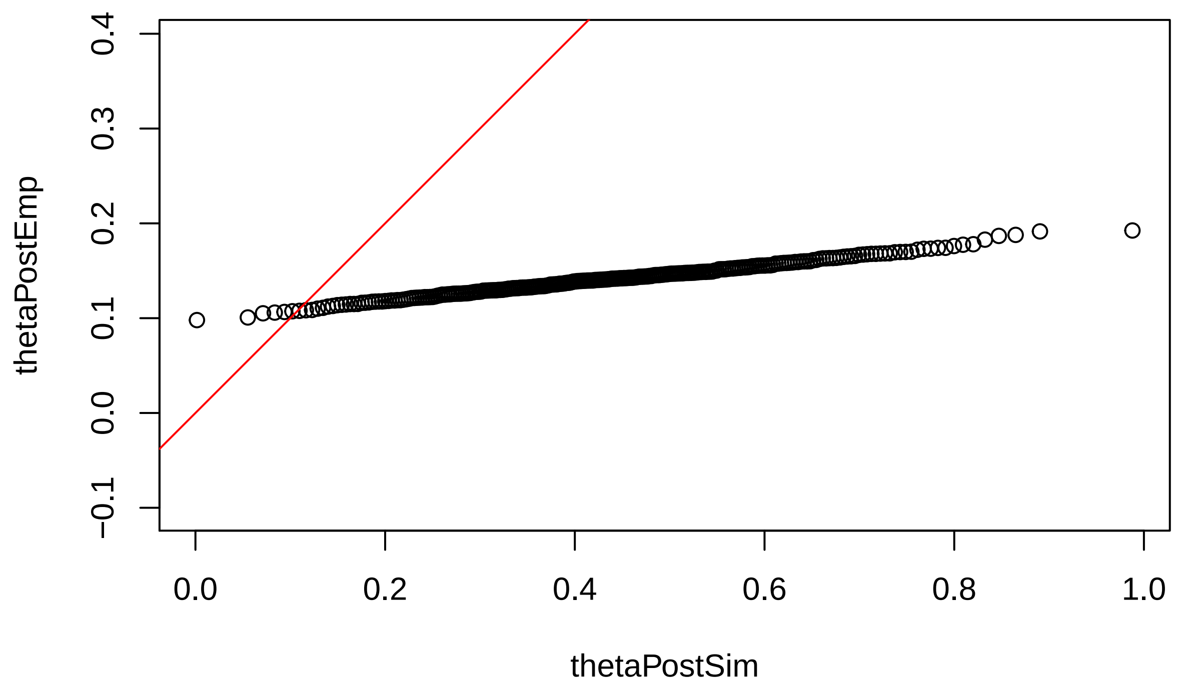
```r
para = rbeta(100000, 2.74, 3.47)

y = vapply(para, function(th){ rbinom(1, prob = th, size = 300)}, numeric(1))

hist(y, breaks = 50, col="yellow")
```



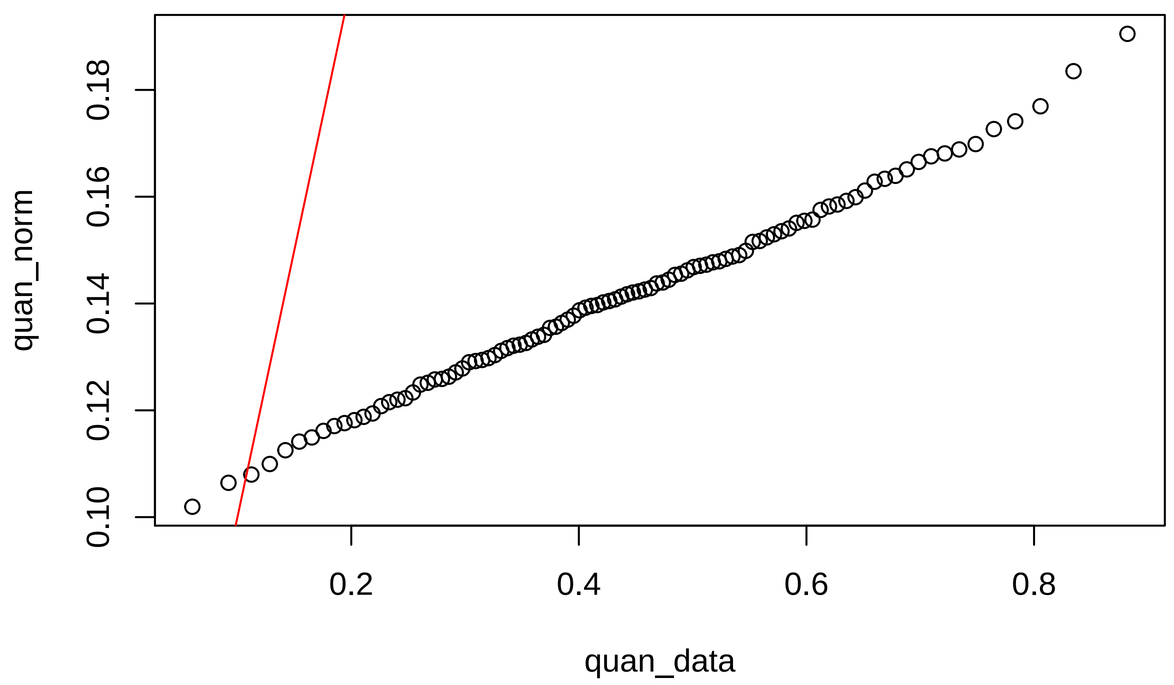
```r
thetaPostEmp = para[ y == 40 ]

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)
```



```
thetaPostSim = rbeta(n=1e6, 2.74, 3.47)


#QQ plot using function
qqplot(thetaPostSim, thetaPostEmp, asp =1)
abline(a =0, b=1, col="red")
```



```
#QQ manual check
quan_data = quantile(thetaPostSim,qs)
quan_norm = quantile(thetaPostEmp,qs)

plot(quan_data,quan_norm)
abline(a =0, b=1, col="red")
```



quan_norm
0.10
0.12
0.14
0.16
0.18
0.2
0.4
0.6
0.8
quan_data