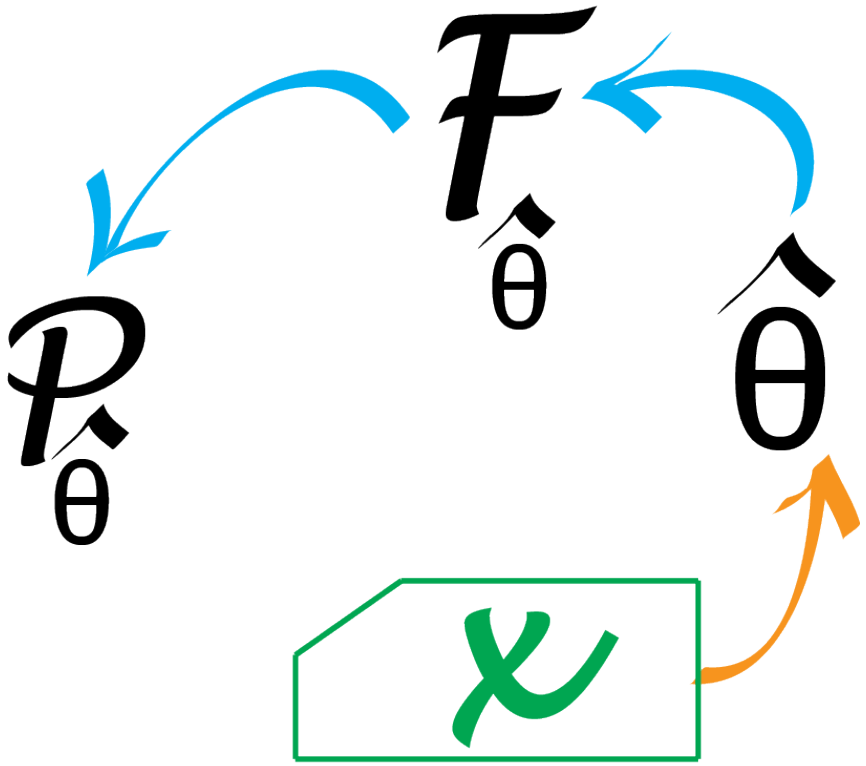
Lecture 2: Statistical Models

Susan Holmes

June 25, 2019

Statistical Modeling



In the previous lecture, the knowledge of both the generative model and the values of the parameters provided us with probabilities we could use for decision making – for instance, whether we had really found an epitope.

In many real situations, neither the generative model nor the parameters are known

- We estimate them using the data we have collected.
- Statistical modeling works from the data *upwards* to a model that *might* plausibly explain the data. This upward–reasonning step is called statistical **inference**.

This lecture will show us some of the distributions and estimation mechanisms that serve as building blocks for inference.

We *estimate* the parameters, denoted by Greek letters with what we call hats on them;

Goals for this chapter

In this lecture we will:

- See that there is a difference between two subjects that are often confused: “Probability” and “Statistics”.
- Fit data to probability distributions using histograms and other visualization tricks.
- Have a first encounter with an estimating procedure known as **maximum likelihood** through a simulation experiement.
- Make inferences from data for which we have prior information using the Bayesian paradigm (involves new distributions with specially tailored properties).

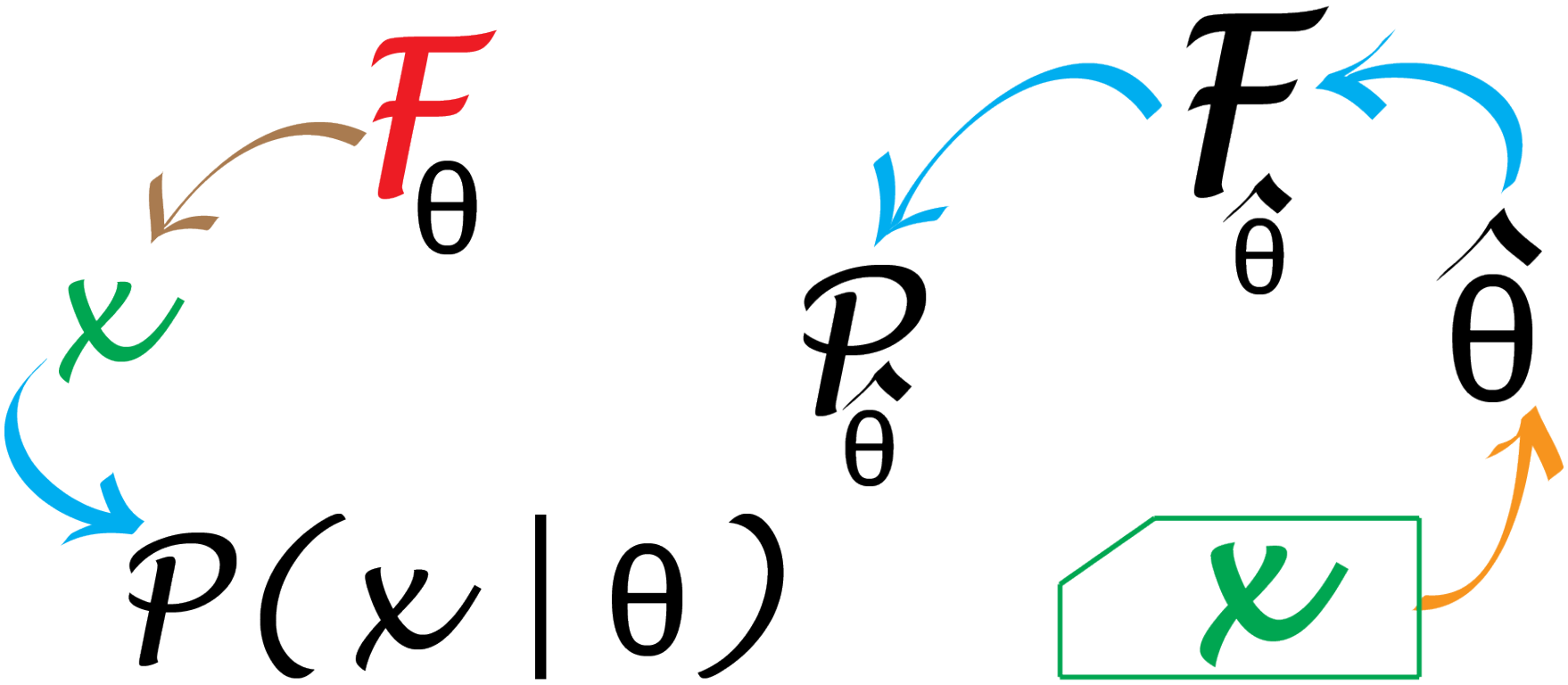
- Use statistical models and estimation to evaluate dependencies in binomial and multinomial distributions.
- Analyse some historically interesting genomic data assembled into tables.
- Make Markov chain models for **dependent** data.
- Do a few concrete applications counting motifs in whole genomes and manipulate special Bioconductor classes dedicated to genomic data.

Parameters are the key.

- Poisson(λ).
- Binomial ($\theta = (n, p)$)
- Multinomial ($\theta = (n, p_1, p_2, p_3, \ldots, p_{m-1})$)
- Beta(α, β)
- Chisquare ($\nu = df$)
- Gamma(α, β)
- **Dirichlet ($\alpha_m, \alpha_m, \ldots, \alpha_m$)**

The difference between statistical and probabilistic models

A probabilistic analysis is possible when we know a good generative model for the randomness in the data, *and* we are provided with the parameters' actual values.



In the epitope example, knowing that false positives occured as Poisson(0.01) per position, the number of patients assayed and the length of the protein ensured that there were *no unknown parameters*.

In such a case, we can use mathematical **deduction** to compute the probability of an event.

In the epitope examples, we used the Poisson probability as our **null model** with the given parameter $\lambda = 0.5$.

We were able to conclude through mathematical deduction that the chances of seeing a maximum value of 7 or larger was smaller than 10^{-4} and thus that in fact the observed data were highly unlikely under that model (or "null hypothesis").

Now suppose that we know the number of patients and the length of the proteins (these are given by the experimental design) but not the distribution itself and the false positive rate.

Once we observe data, we need to go *up* from the data to estimate both a probability model F (Poisson, normal, binomial) and eventually the missing parameter(s) for that model. This is the type of statistical **inference** we will explain in this chapter.

A simple example of statistical modeling

Start with the data

There are two parts to the modeling procedure.
- First we need a reasonable probability *distribution* to model the data generation process.
- Discrete count data may be modeled by simple probability distributions such as binomial, multinomial or Poisson distributions.

The normal distribution, or bell shaped curve, is often a good model for continuous measurements. Distributions can also be more complicated mixtures of these elementary ones (more on this in the lecture on Mixtures)).

Let's revisit the epitope example from the previous chapter, starting without the tricky outlier.

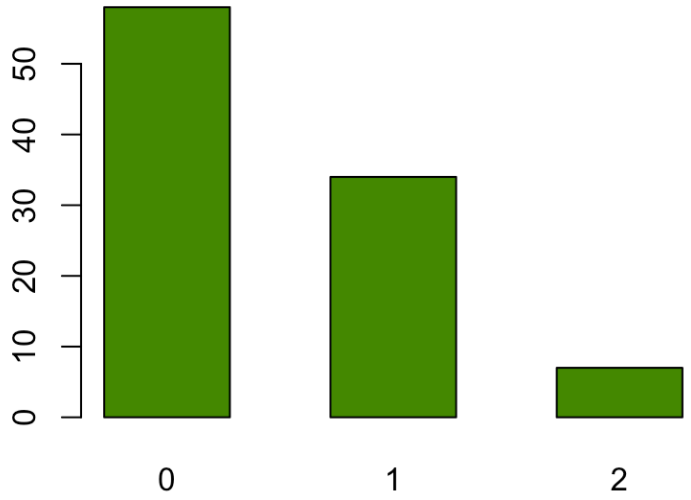
```
load("~/Books/CUBook/data/e100.RData")
e99 = e100[-which.max(e100)]
```

Goodness-of-fit : visual evaluation

Our first step is to find a fit from candidate distributions; this requires consulting graphical and quantitative goodness-of-fit plots.

For discrete data, we can plot a barplot of frequencies (for continuous data, we would look at the histogram).

```
barplot(table(e99), space = 0.8, col = "chartreuse4")
```

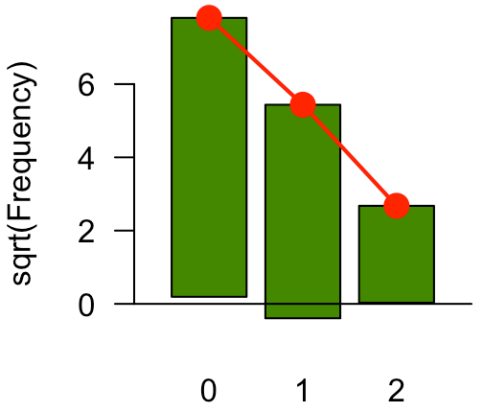


The observed distribution of the epitope data without the outlier.

However, it is hard to decide which theoretical distribution fits the data best without using a comparison. One visual **goodness-of-fit** diagram is known as the **rootogram**.

It hangs the bars with the observed counts from the theoretical red points. If the counts correspond exactly to their theoretical values, the bottom of the boxes will align exactly with the horizontal axis.

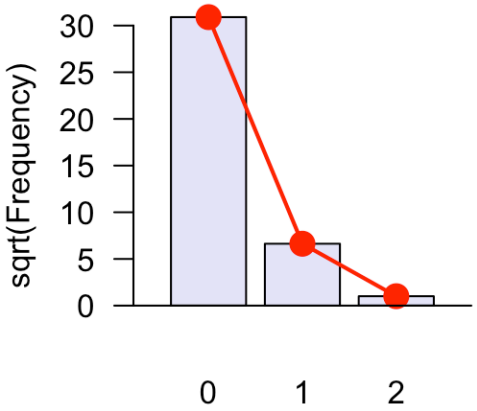
```
library("vcd")
gf1 = goodfit( e99, "poisson")
rootogram(gf1, xlab = "", rect_gp = gpar(fill = "chartreuse4"))
```



Rootogram showing the square root of the theoretical values as red dots and the square root of the observed frequencies as drop down rectangles. (We'll see a bit below how the `goodfit` function decided which λ to use.)

To calibrate what such a plot looks like with a known Poisson variable, use `rpois` with λ = 0.05 to generate 100 Poisson distributed numbers and draw their rootogram.

```
simp = rpois(1000, lambda = 0.05)
gf2 = goodfit(simp, "poisson")
rootogram(gf2, xlab = "", rect_gp = gpar(fill = "lavender"))
```



We see that the rootogram for `e99` seems to fit the Poisson model reasonably well. But remember, to make this happen we removed the outlier.

The Poisson is completely determined by one parameter, often called the Poisson mean λ.

We estimate the Poisson parameter from the data.

“The parameter is called the Poisson mean because it is the mean of the theoretical distribution *and*, as it turns out, is estimated by the sample mean. This overloading of the word is confusing to everyone.”

The most common way of estimating λ is to choose the value $\hat{\lambda}$ that makes the observed data the most likely. This is called the **maximum likelihood estimator** often abbreviated **MLE**.

We will illustrate this idea in the next section.

Although we took out the extreme observation before taking a guess at the probability distribution, we are going to return to the data with it for the rest of our analysis.

In practice we would not know whether there is an outlier, and which data point(s) it is / they are.

The effect of leaving it in is to make our estimate of the mean higher.

In turn, this would make it more likely that we'd observe a value of 7 under the null model, resulting in a larger p-value.

So, if the resulting p-value is small even with the outlier included, we are assured that our analysis is up to something real.

We call such a tactic being **conservative**: we err on the side of caution, of not detecting something.

Estimating the parameter of the Poisson distribution

What value for the Poisson mean makes the data the most probable? In a first step, we tally the outcomes.

```
table(e100)
```

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

Then we are going to try out different values for the Poisson mean and see which one gives the best fit to our data. If the mean λ of the Poisson distribution were 3, the counts would look something like this:

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

```
##
##  0  1  2  3  4  5  6  7  9
##  5 13 27 20 14 13  5  2  1
```

This has many more 2's and 3's than we see in our data. So we see that $\lambda = 3$ is unlikely to have produced our data, as the counts do not match up so well.

So we could try out many possible values and proceed by brute force.

```
table(rpois(100, 2))
```

```
##
##  0  1  2  3  4  5
## 13 21 34 20  9  3
```

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

```
##
##  0  1  2  3
## 35 30 22 13
```

```
table(rpois(100, 0.75))
```

```
##
##  0  1  2  3
## 46 37 13  4
```

We can use a little mathematics to see which value maximizes the probability of observing our data.

Let's calculate the probability of seeing the data if the value of the Poisson parameter is m.

Since we suppose the data derive from independent draws, this probability is simply the product of individual probabilities:

$$P(58 \times 0, 34 \times 1, 7 \times 2, \text{one } 7 \mid \text{data are Poisson}(m)) = P(0)^{58} \times P(1)^{34} \times P(2)^{7} \times P(7)$$

For $m = 3$ we can compute this

```
prod(dpois(c(0, 1, 2, 7), lambda = 3) ^ (c(58, 34, 7, 1)))
```

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

This probability is the **likelihood function** of λ, given the data, and we write it

Here L stands for likelihood and $f(k) = e^{-\lambda}\frac{\lambda^k}{k!}$, the Poisson probability we saw earlier.

$$L(\lambda,\ \ x = (k_1, k_2, k_3, \dots)) = \prod_{i=1}^{100} f(k_i)$$

Instead of working with multiplications of a hundred small numbers, it is convenient to take the logarithm.

We compute the likelihood for many different values of the Poisson parameter.

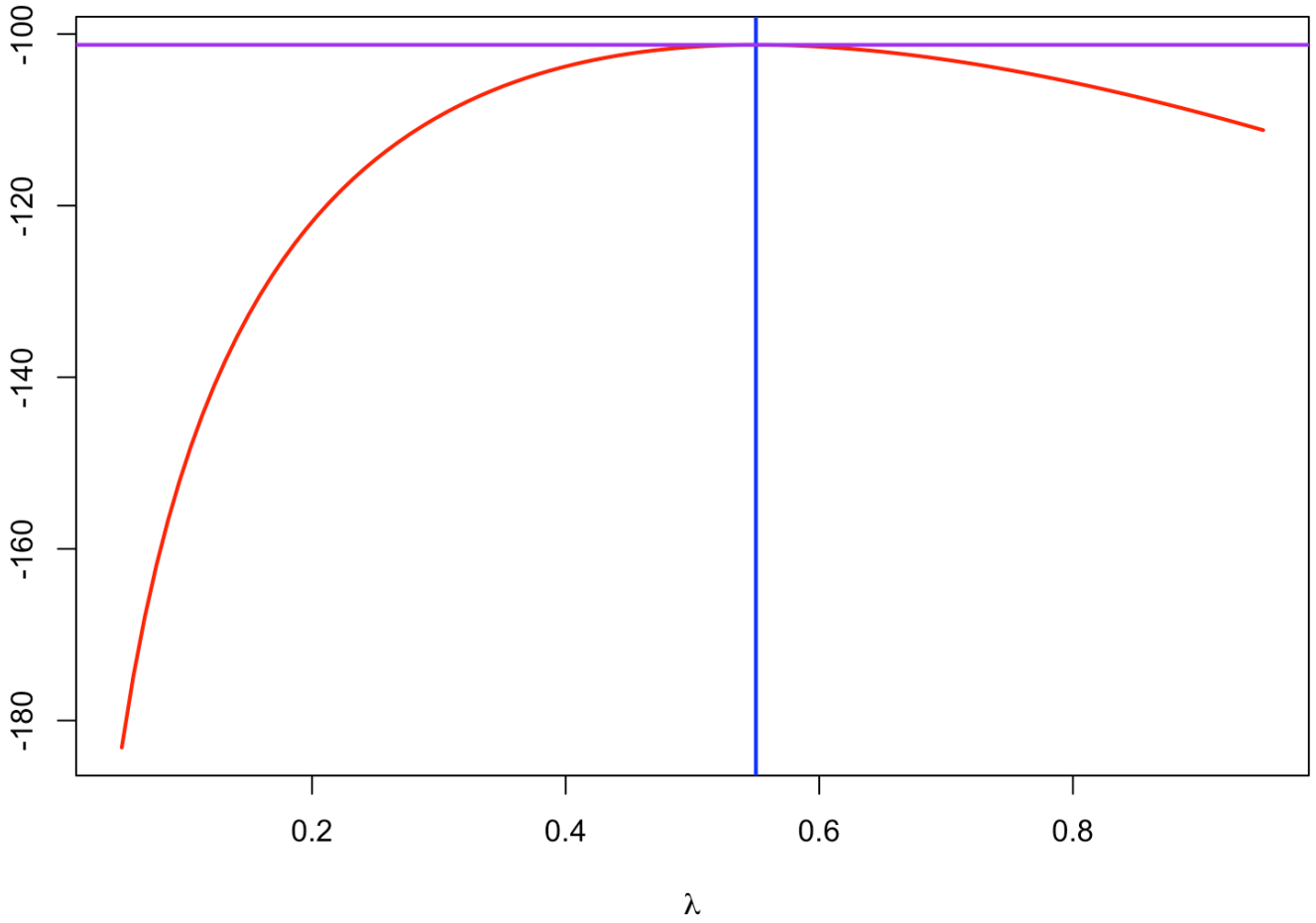
To do this we need to write a small function that computes the probability of the data for different values.

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

Now we can compute the likelihood for a whole series of `lambda` values from 0.05 to 0.95.

```
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
```



The red curve is the log-likelihood function. The vertical line shows the value of `m` (the mean) and the horizontal line the log-likelihood of `m`. It looks like `m` maximizes the likelihood.

In fact there is a shortcut: the function `goodfit`.

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

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

```
gf$par
```

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

The output of `goodfit` is a composite object called a list.

One of its components is called `par` and contains the value(s) of the fitted parameter(s) for the distribution studied.

In this case it's only one number, the estimate of λ.

Classical statistics for classical data

Here is a formal proof of our computational finding that the mean maximizes the (log-)likelihood.

$$\begin{aligned}\log L(\lambda, x) &= \sum_{i=1}^{100} -\lambda + k_i \log \lambda - \log(k_i!) \\ &= -100\lambda + \log \lambda \left(\sum_{i=1}^{100} k_i \right) + \text{const.}\end{aligned}$$

We use the catch-all "const." for terms that do not depend on λ (although they do depend on x, i.e., on the k_i). To find the λ that maximizes this, we compute the derivative in λ and set it to zero.

$$\begin{aligned}\frac{d}{d\lambda} \log L &= -100 + \frac{1}{\lambda} \sum_{i=1}^{100} k_i \stackrel{?}{=} 0 \\ \lambda &= \frac{1}{100} \sum_{i=1}^{100} k_i = \bar{k}\end{aligned}$$

First steps of a *statistical approach*, starting `from the ground up' (from the data) to infer the model parameter(s): this is statistical *estimation* a parameter from data.

Another important component will be choosing which family of distributions our data come from, that part is done by evaluating the *goodness of fit*.

In the classical *statistical testing* framework, we consider one single model, that we call the *null model*, for the data.

The null model formulates an “uninteresting” baseline.

We then test whether there is something more interesting going on by computing the probability that the data are compatible with that model.

Models are concise but expressive representations of the data generating process.

For the Poisson for instance, knowing one number allows us to know everything about the distribution, including, as we saw earlier, the probabilities of extreme or rare events.

Another useful direction is **regression**.

We may be interested in knowing how our count-based response variable (e.g., the result of counting sequencing reads) depends on a continuous covariate, say, temperature or nutrient concentration.

You may already have encountered linear regression, where our model is that the response variable y depends on the covariate x via the equation $y \sim ax + b + e$, with parameters a and b (that we need to estimate), and with residuals e whose probability model is a normal distribution (whose variance we usually also need to estimate).

For count data the same type of regression model is possible, although the probability distribution for the residuals then needs to be non-normal.

In that case we use the **generalized linear models** framework.

Binomial distributions and maximum likelihood

In a binomial distribution there are two parameters: the number of trials n, which is typically known, and the probability p of seeing a 1 in a trial. This probability is often unknown.

An example

Suppose we take a sample of $n = 120$ males and test them for red-green colorblindness. We can code the data as 0 if the subject is not colorblind and 1 if he is. We summarize the data by the table:

```
table(cb)
```

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

Which value of p is the most likely given these data?

$\hat{p} = \frac{1}{12}$.

However, be careful: sometimes ML estimates are harder to guess and to compute, as well as being much less intuitive.

In this special case, your intuition may give you the estimate $\hat{p} = \frac{1}{12}$, which turns out to be the maximum likelihood estimate.

We put a hat over the letter to remind us that this is not (necessarily) the underlying true value, but an estimate we make from the data.

```
probs  =  seq(0, 0.3, by = 0.005)
likelihood = dbinom(sum(cb), prob = probs, size = length(cb))
probs[which.max(likelihood)]
```

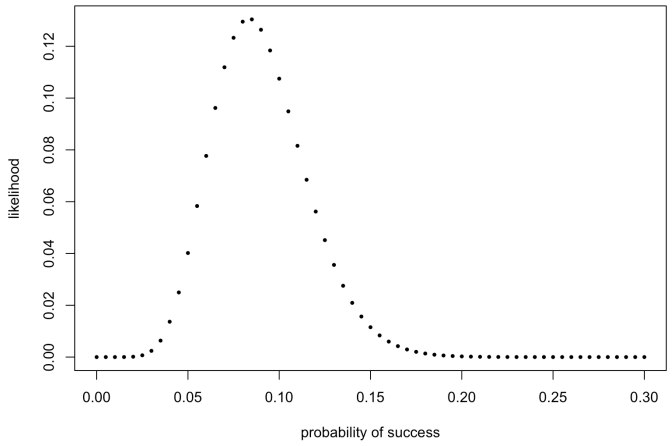
```
## [1] 0.085
```

Plot of the likelihood as a function of the probabilities.

The likelihood is a function on $[0, 1]$; here we have zoomed in, as the likelihood is practically zero for larger values of p.

```
plot(probs, likelihood, pch = 16, xlab = "probability of success",
       ylab = "likelihood", cex=0.6)
probs[which.max(likelihood)]
```

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



Likelihood for the binomial distribution

One can come up with different criteria than ML, which lead to other estimators: they all carry hats. We'll see other examples in the lecture on Mixtures.

The likelihood and the probability are the same mathematical function,interpreted in different ways

- in one case, it tells us how probable it is to see a particular set of values of the data, given the parameters;
- in the other case, we consider the data as fixed, and ask for the particular parameter value that makes the data more likely.

Suppose $n = 300$, and we observe $y = 40$ successes. Then, for the binomial distribution:

$$f(\theta \mid n, y) = f(y \mid n, \theta) = \binom{n}{y} \theta^y \, (1 - \theta)^{(n-y)}.$$

As $\binom{n}{y}$ is very large, we use the logarithm of the likelihood to give:

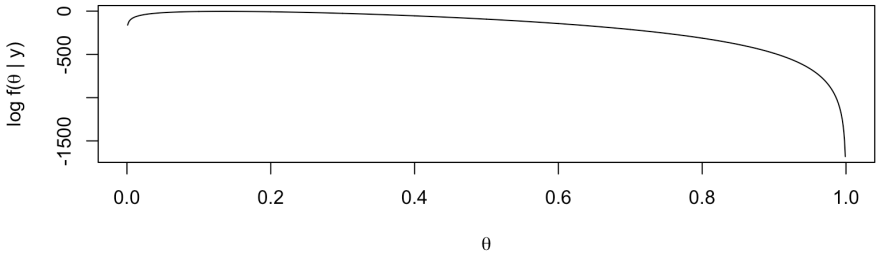
$$\log f(\theta|y) = 115 + 40 \log(\theta) + (300 - 40) \log(1 - \theta).$$

Here's a function we use to calculate it:

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

which we plot for the range of θ from 0 to 1 .

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



Plot of the log likelihood function for $n = 300$ and $y = 40$.

The maximum lies at 40/300 = 0.1333… , consistent with intuition, but we see that other values of θ are almost equally likely.

Bayesian methods allow us to use a range of values for θ.

Modeling sequential dependencies: Markov chains

If we want to predict tomorrow's weather, a reasonably good guess is that it will most likely be the same as today's weather, in addition we may state the probabilties for various kinds of possible changes.

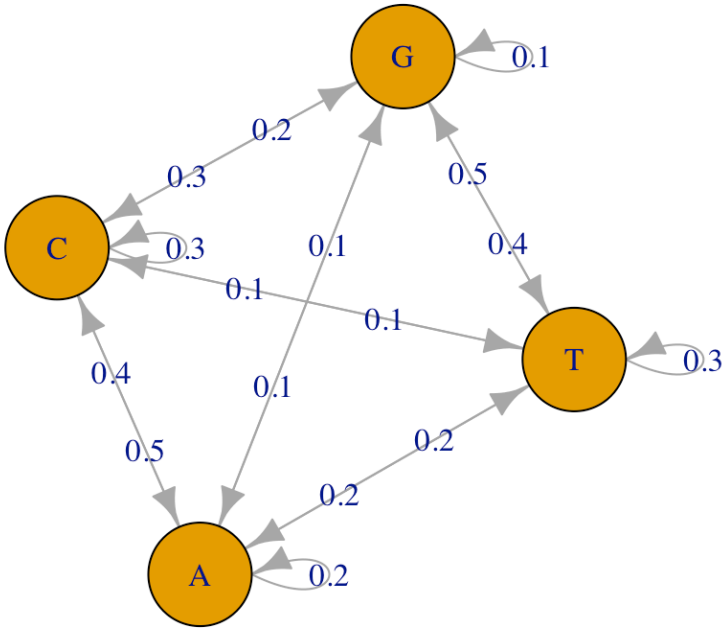
In DNA, we may see specific succession of patterns so that pairs of nucleotides, called digrams, say, [CG, CA, CC] and [CT] are not equally frequent. For instance, in parts of the genome we see more frequent instances of [CA] than we would expect under independence:

$$P(\mathtt{CA}) \neq P(\mathtt{C})\,P(\mathtt{A}).$$

We model this dependency in the sequence as a **Markov chain**:

$$P(\mathtt{CA}) = P(\mathtt{NCA}) = P(\mathtt{NNCA}) = P(\ldots\mathtt{CA}) = P(\mathtt{C})\,P(\mathtt{A}|\mathtt{C})$$

where `N` stands for any nucleotide, and $P(\mathtt{A}|\mathtt{C})$ stands for "the probability of `A`, given that the preceding base is a `C`". Figure below shows a schematic representation of such transitions on a graph.



Visualisation of a 4-state Markov chain. The probability of each possible digram (e.,g., CA) is given by the weight of the edge between the corresponding nodes. So for instance, the probability of CA is given by the edge C→ A. We'll see in the Images lecture how to use **R** packages to draw these type of network graphs.

Bayesian Thinking

“Turtles all the way down.”

Bayesian modeling of the uncertainty of the parameter of a distribution is done by using a random variable whose distribution may depend on parameters whose uncertainty can be modeled as a random variable; these are called hierarchical models.

Classical (frequentist) approach where the parameters of our distributions, i.e., the probabilities of the possible different outcomes, represent long term frequencies.

The parameters are –at least conceptually– definite, knowable, fixed numbers.

We may not know them, so we estimate them from the data at hand.

However, such an approach does not take into account any information that we might already know.

For that we need a different approach, in which we use probability distributions to express our knowledge about the parameters, and use data to *update* this knowledge, for instance by shifting those distributions or making them more narrow; this is provided by the Bayesian paradigm.

The Bayesian paradigm is a practical approach where *prior* and *posterior* distributions are used as models of our knowledge *before* and *after* collecting some data and making an observation. It is particularly useful for integrating or combining information from different sources.

Suppose we have a certain hypothesis, call it H, true ??

We can formalize our prior knowledge about H in the form of a **prior** probability, written $P(H)$.

After we see the data, we have the **posterior** probability: $P(H \mid D)$, The probability of H given that we saw D.

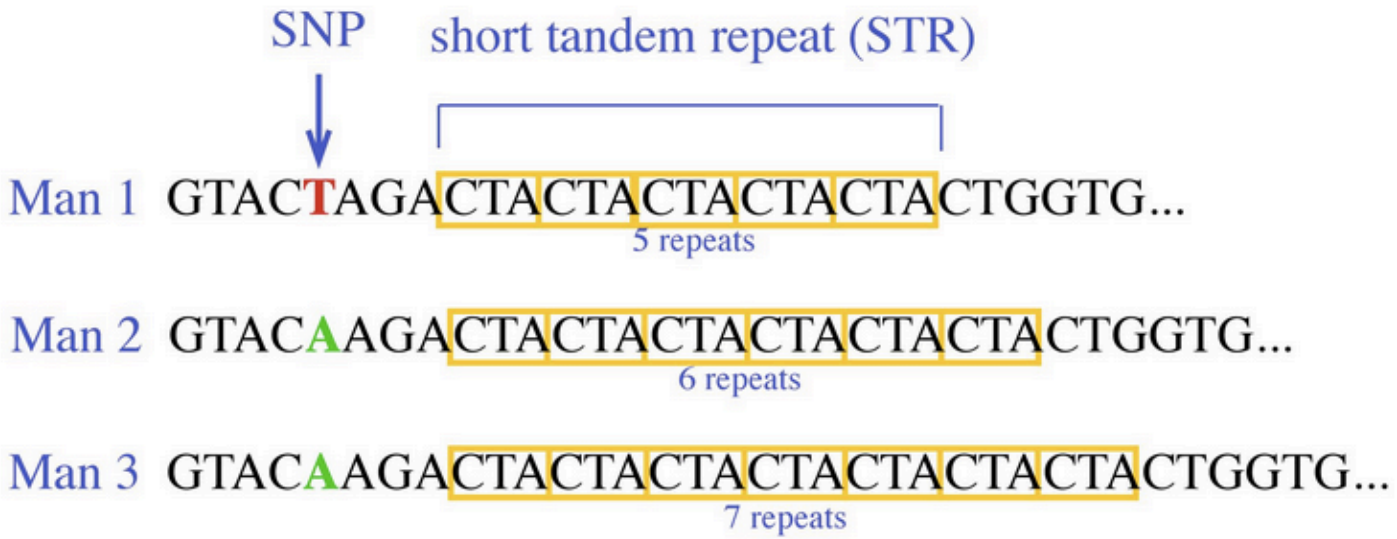
This may be higher or lower than $P(H)$, depending on what the data D were.

Haplotypes

We study a forensics example using combined signatures from the Y chromosome called haplotypes.

A haplotype is a collection of alleles (DNA sequence variants) that are spatially adjacent on a chromosome, are usually inherited together (recombination tends not to disconnect them), and thus are genetically linked. In this case we are looking at linked variants on the Y chromosome.

First we'll look at the motivation behind haplotype frequency analyses, then we'll revisit a little the idea of likelihood. After this, we'll explain how we can think of unknown parameters as being random numbers themselves, modeling their uncertainty with a prior distribution. Then we will see how to incorporate new data observed into the probability distributions and compute posterior confidence statements about the parameters.


SNP
short tandem repeat (STR)
Man 1 GTACTAGACTACTACTACTACTACTGGTG...
5 repeats
Man 2 GTACAAGACTACTACTACTACTACTACTGGTG...
6 repeats
Man 3 GTACAAGACTACTACTACTACTACTACTACTGGTG...
7 repeats

Example: haplotype frequencies

We want to estimate the proportion of a particular Y-haplotye that consists of a set of different short tandem repeats (STR). The combination of STR numbers at the specific STR locations used for DNA forensics are labeled by the number of repeats at the specific positions. The US Y STR database can be accessed at the http://www.usystrdatabase.org. Here is a short excerpt of an STR haplotype table:

```
haplo6=read.table("~/Books/CUBook/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
```

This says that the haplotype H1 has 14 repeats at position DYS19, 12 repeats at position DXYS156Y . . .. The haplotypes created through the use of these Y-STR profiles are shared between men in the same patriarchal lineages. For these reasons it is possible that two different men share the same profile.

We need to find the underlying proportion θ of the haplotype of interest in the population of interest. We are going to consider the occurrence of a haplotype as a `success' in a binomial distribution using collected observations.

Simulation study of the Bayesian paradigm for the binomial

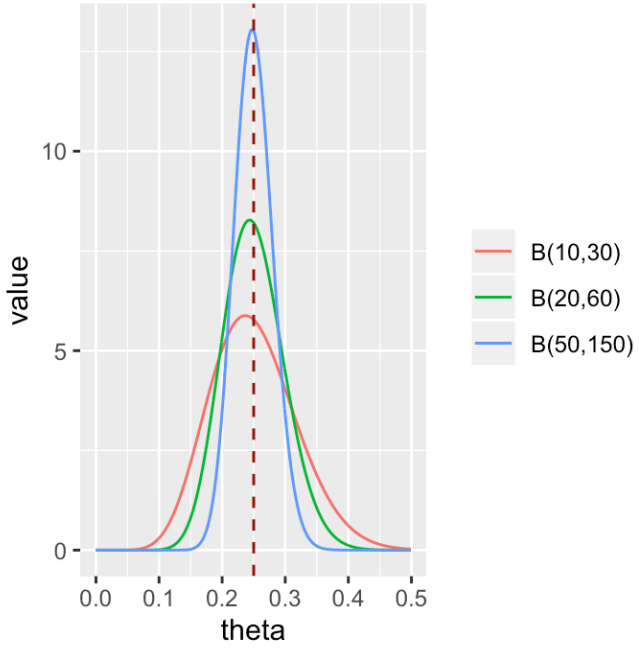
Instead of assuming that our parameter θ has one single value, the Bayesian world view allows us to see it as a draw from a statistical distribution. The distribution expresses our belief about the possible values of the parameter θ. In principle, we can use any distribution that we like whose possible values are permissible for θ. When we are looking at a parameter that expresses a proportion or a probability, and which takes its values between 0 and 1, it is convenient to use the *beta distribution*.

Its density formula is written

$$f_{\alpha,\beta}(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\mathrm{B}(\alpha,\beta)} \quad \text{where} \quad \mathrm{B}(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$

We can see how this function depends on two parameters α and β, making it a very flexible family of distributions (so it can 'fit' a lot different situations).

It has a nice mathematical property: if we start with a prior belief on θ that is beta-shaped, observe a dataset of n binomial trials, then update our belief, the posterior distribution on θ will also have a beta distribution, albeit with updated parameters. This is a mathematical fact, we will not prove it, however we demonstrate it by simulation.



Beta distributions with $\alpha = 10, 20, 50$ and $\beta = 30, 60, 150$ used as a {prior} for a probability of success. These three distributions have the same mean ($\frac{\alpha}{\alpha+\beta}$), but different concentrations around the mean.

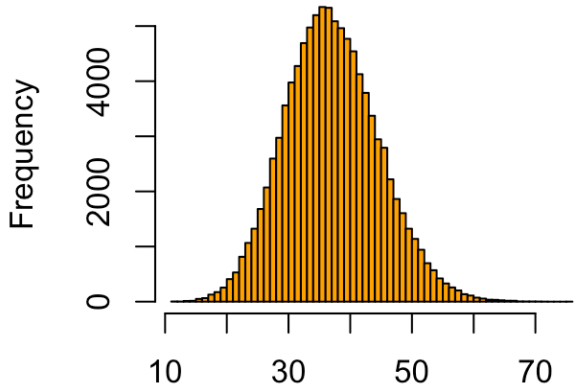
The distribution of Y

For a given choice of θ, we know what the distribution of Y is. But what is the distribution of Y if θ itself also varies according to some distribution ?

We call this the **marginal distribution** of Y.

First we generate a random sample of 10000 θs. For each of these θs, we then generate a random sample of Y.

```
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 = "")
```

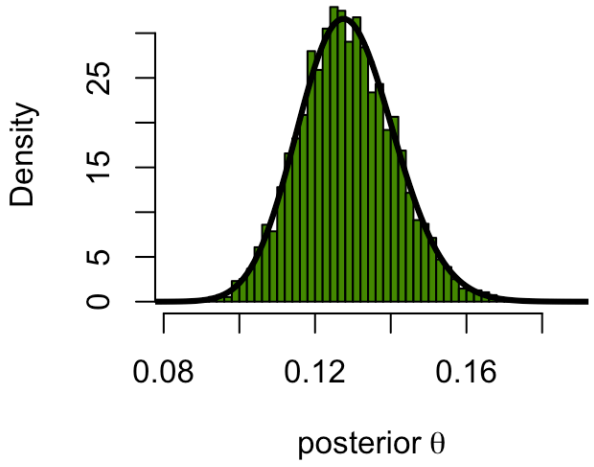


Marginal Distribution of Y.

Histogram of all the thetas such that $Y = 40$: the posterior distribution

So let's now compute the posterior distribution of θ by conditioning on those outcomes where Y was 40. We compare it to the theoretical posterior, `densPostTheory`, of which more below.

```
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)
```



Only choosing the values of the distribution with $Y = 40$ gives the posterior

distribution of θ. The histogram (green) shows the simulated values for the posteriror distribution, the line the theoretical density of a beta distribution with the theoretical posterior parameters.

We can also check the means of both distributions computed above and see that they are close to 4 significant digits.

```
mean(thetaPostEmp)
```

```
## [1] 0.1284284
```

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

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

To approximate the mean of the theoretical density `densPostTheory`, we have computed the integral $\int_0^1 \theta f(\theta)\, d\theta$ using numerical integration, i.e., the `sum` over the integrant.

This is not always convenient (or feasible).

Thus, let's see how we could use **Monte Carlo integration** instead.

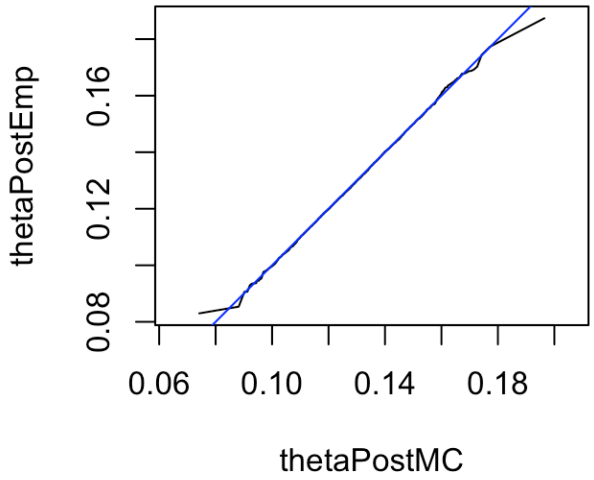
This is similar to the code above, where we used numerical integration to compute the posterior mean from `thetaPostEmp` by calling R's `mean` function.

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

```
## [1] 0.1285677
```

We can check the concordance between our Monte Carlo sample `thetaPostMC` and our sample `thetaPostEmp` using a **quantile-quantile plot** (**QQ-plot**).

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



QQ-plot of our Monte Carlo sample `thetaPostMC` from the theoretical distribution and our simulation sample `thetaPostEmp`. We could also similarly compare either of these two distributions to the theoretical distribution function `pbeta(., 90, 610)`. If the curve lies on the line $y = x$ this indicates a good agreement. There are some random differences at the tails.

Posterior distribution is also a beta

Now we have seen that the posterior distribution is also a beta. In our case its parameters $\alpha = 90$ and $\beta = 610$ were obtained by summing the prior parameters $\alpha = 50, \beta = 350$ with the observed successes $y = 40$ and the observed failures $n - y = 260$, thus obtaining the posterior

$$\text{beta}(90,\ 610) = \text{beta}(\alpha + y, \beta + (n - y)).$$

We can use it to give the best estimate we can for θ with its uncertainty given by the posterior distribution.

Suppose we had a second series of data

After seeing our previous data, we now have a new prior, $\text{beta}(90, 610)$.

- Now we collect a new set of data with $n = 150$ observations and $y = 25$ successes, thus 125 failures.
- Now what would we take to be our best guess at θ?

Using the same reasoning as before, the new posterior will be: $\text{beta}(90 + 25 = 115,\ 610 + 125 = 735)$. The mean of this distribution is $\frac{115}{115+735} = \frac{115}{850} \simeq 0.135$, thus one estimate of θ would be 0.135.

The theoretical **maximum a posteriori** (MAP) estimate would be the mode of $\text{beta}(115, 735)$, ie $\frac{114}{848} \simeq 0.134$. Let's check this numerically.

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

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

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

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

```
## [1] 0.1353109
```

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

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

As a general rule, the prior rarely changes the posterior distribution substantially except if it is very peaked. This would be the case if, at the outset, we were already rather sure of what to expect. Another case when the prior has an influence is if there is very little data.

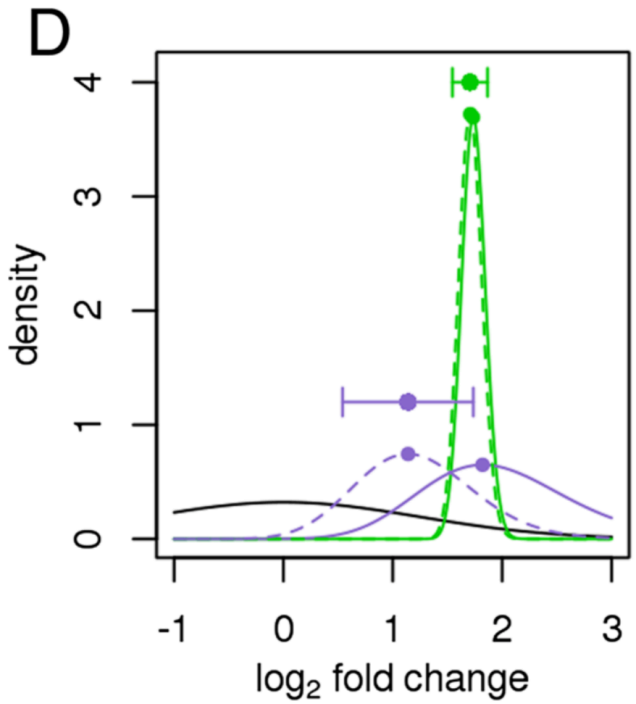
The best situation to be in is to have enough data to swamp the prior so that its choice doesn't have much impact on the final result.

Confidence Statements for the proportion parameter

Now it is time to conclude about where the proportion actually lies given the data. One summary is a posterior credibility interval, which is a Bayesian analog of the confidence interval. We can take the 2.5 and 97.5-th percentiles of the posterior distribution: $P(L \leq \theta \leq U) = 0.95$ using R.

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

```
##      2.5%     97.5%
## 0.1131596 0.1590770
```



An example from Love et al. shows plots of the likelihoods (solid lines, scaled to integrate to 1) and the posteriors (dashed lines) for the green and purple genes and of the prior (solid black line): due to the higher dispersion of the purple gene, its likelihood is wider and less peaked (indicating less information), and the prior has more influence on its posterior than for the green gene. The stronger curvature of the green posterior at its maximum translates to a smaller reported standard error for the MAP logarithmic fold change (LFC) estimate (horizontal error bar).

Intermezzo: quantiles and the quantile-quantile plot

In the previous lecture, we ordered the 100 sample values $x_{(1)}, x_{(2)}, \ldots, x_{(100)}$. Say we want the 22nd percentile. We can take any value between the 22nd and the 23rd value, i.e., any value that fulfills $x_{(22)} \leq c_{0.22} < x_{(23)}$ is acceptable as a 0.22 **quantile** ($c_{0.22}$). In other words, $c_{0.22}$ is defined by

$$\frac{\#x_i' s \leq c_{0.22}}{n} = 0.22.$$

We'll introduce the **empirical cumulative distribution** function (**ECDF**) $\hat{F}$, and we'll see that our definition of $c_{0.22}$ can also be written as $\hat{F}_n(c_{0.22}) = 0.22$. Our histogram of the distribution of `simulstat`, the quantiles $c_{0.95}$ and $c_{0.99}$ are also shown.

Do you know another name for the 0.5 quantile?

In the above definition, we were a little vague on how the quantile is defined in general, i.e., not just for 0.22. How is the quantile computed for any number between 0 and 1, including ones that are not multiples of $1/n$?

Example

Now that we have an idea what quantiles are, we can do the quantile-quantile plot. We plot the quantiles of the `simulstat` values, which we simulated under the null hypothesis, as we did yesterday, against the theoretical null distribution χ^2_{30}.

```
library("Biostrings")
staph = readDNAStringSet("/Users/susan/Books/CUBook/data/staphsequence.ffn.txt", "fasta")
staph[1]
```

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

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

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

```
letterFrq = vapply(staph, letterFrequency, FUN.VALUE = numeric(4),
         letters = "ACGT", OR = 0)
colnames(letterFrq) = paste0("gene", seq(along = staph))
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
```

```
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
```

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

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

```r
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
```

```r
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)


## ----quantiles3, results = "hide"----------------------------------------
qs = ppoints(100)
quantile(simulstat, qs)
```

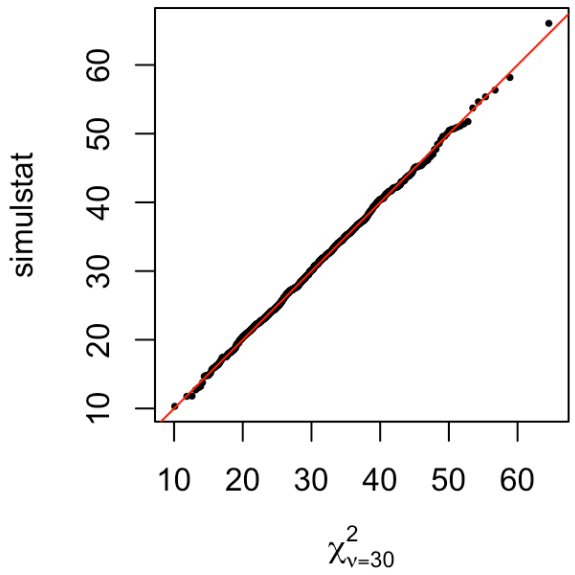
```
##      0.5%      1.5%      2.5%      3.5%      4.5%      5.5%      6.5%      7.5%
## 13.20500 15.89946 16.99220 17.54334 18.18538 18.62506 19.42654 20.00401
##      8.5%      9.5%     10.5%     11.5%     12.5%     13.5%     14.5%     15.5%
## 20.45738 20.80610 21.05267 21.36324 21.64697 21.99116 22.24713 22.41011
##     16.5%     17.5%     18.5%     19.5%     20.5%     21.5%     22.5%     23.5%
## 22.65597 22.82566 22.98341 23.19774 23.45116 23.65964 23.90069 24.10968
##     24.5%     25.5%     26.5%     27.5%     28.5%     29.5%     30.5%     31.5%
## 24.27877 24.39464 24.59680 24.84055 25.00760 25.32015 25.47057 25.83695
##     32.5%     33.5%     34.5%     35.5%     36.5%     37.5%     38.5%     39.5%
## 26.13995 26.35890 26.59286 26.82501 26.98051 27.17708 27.26425 27.38923
##     40.5%     41.5%     42.5%     43.5%     44.5%     45.5%     46.5%     47.5%
## 27.52349 27.62541 27.72985 27.92958 28.18695 28.44794 28.64189 28.79023
##     48.5%     49.5%     50.5%     51.5%     52.5%     53.5%     54.5%     55.5%
## 29.00234 29.23597 29.44457 29.62480 29.97438 30.07458 30.31322 30.64879
##     56.5%     57.5%     58.5%     59.5%     60.5%     61.5%     62.5%     63.5%
## 30.79298 30.96474 31.17992 31.51293 31.63883 31.85473 32.00046 32.18669
##     64.5%     65.5%     66.5%     67.5%     68.5%     69.5%     70.5%     71.5%
## 32.44111 32.60020 32.75601 32.92959 33.24610 33.59389 33.79435 34.07003
##     72.5%     73.5%     74.5%     75.5%     76.5%     77.5%     78.5%     79.5%
```

```
## 34.27596 34.44904 34.68285 35.05178 35.30848 35.47194 35.73732 36.05698
##    80.5%    81.5%    82.5%    83.5%    84.5%    85.5%    86.5%    87.5%
## 36.34514 36.76013 36.96392 37.28811 37.57727 38.05504 38.74359 39.31005
##    88.5%    89.5%    90.5%    91.5%    92.5%    93.5%    94.5%    95.5%
## 39.85189 40.38479 40.65731 41.40253 41.87741 42.24915 43.06681 43.86416
##    96.5%    97.5%    98.5%    99.5%
## 45.17572 46.16617 49.60478 53.73402
```

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

```
##     0.5%     1.5%     2.5%     3.5%     4.5%     5.5%     6.5%     7.5%
## 14.74308 16.23869 17.16382 17.87179 18.45879 18.96730 19.42030 19.83175
##     8.5%     9.5%    10.5%    11.5%    12.5%    13.5%    14.5%    15.5%
## 20.21086 20.56401 20.89588 21.20996 21.50896 21.79501 22.06984 22.33486
##    16.5%    17.5%    18.5%    19.5%    20.5%    21.5%    22.5%    23.5%
## 22.59125 22.83999 23.08192 23.31776 23.54813 23.77359 23.99461 24.21163
##    24.5%    25.5%    26.5%    27.5%    28.5%    29.5%    30.5%    31.5%
## 24.42502 24.63512 24.84224 25.04668 25.24867 25.44846 25.64627 25.84230
##    32.5%    33.5%    34.5%    35.5%    36.5%    37.5%    38.5%    39.5%
## 26.03673 26.22975 26.42152 26.61219 26.80192 26.99084 27.17910 27.36683
##    40.5%    41.5%    42.5%    43.5%    44.5%    45.5%    46.5%    47.5%
## 27.55414 27.74118 27.92804 28.11486 28.30175 28.48883 28.67621 28.86400
##    48.5%    49.5%    50.5%    51.5%    52.5%    53.5%    54.5%    55.5%
## 29.05231 29.24127 29.43100 29.62161 29.81322 30.00595 30.19994 30.39530
##    56.5%    57.5%    58.5%    59.5%    60.5%    61.5%    62.5%    63.5%
## 30.59219 30.79073 30.99107 31.19337 31.39779 31.60450 31.81367 32.02550
##    64.5%    65.5%    66.5%    67.5%    68.5%    69.5%    70.5%    71.5%
## 32.24019 32.45796 32.67904 32.90367 33.13213 33.36471 33.60172 33.84351
##    72.5%    73.5%    74.5%    75.5%    76.5%    77.5%    78.5%    79.5%
## 34.09046 34.34299 34.60155 34.86665 35.13886 35.41883 35.70725 36.00496
##    80.5%    81.5%    82.5%    83.5%    84.5%    85.5%    86.5%    87.5%
## 36.31286 36.63203 36.96368 37.30925 37.67041 38.04916 38.44789 38.86951
##    88.5%    89.5%    90.5%    91.5%    92.5%    93.5%    94.5%    95.5%
## 39.31761 39.79669 40.31253 40.87266 41.48728 42.17057 42.94329 43.83752
##    96.5%    97.5%    98.5%    99.5%
## 44.90723 46.25504 48.12235 51.45778
```

```r
B=2000
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")
```



Our simulated statistic's distribution compared to χ^2_{30} using a QQ-plot, which shows the theoretical **quantiles** for the χ^2_{30} distribution on the horizontal axis and the sampled ones on the vertical axis.

Chargaff's Rule

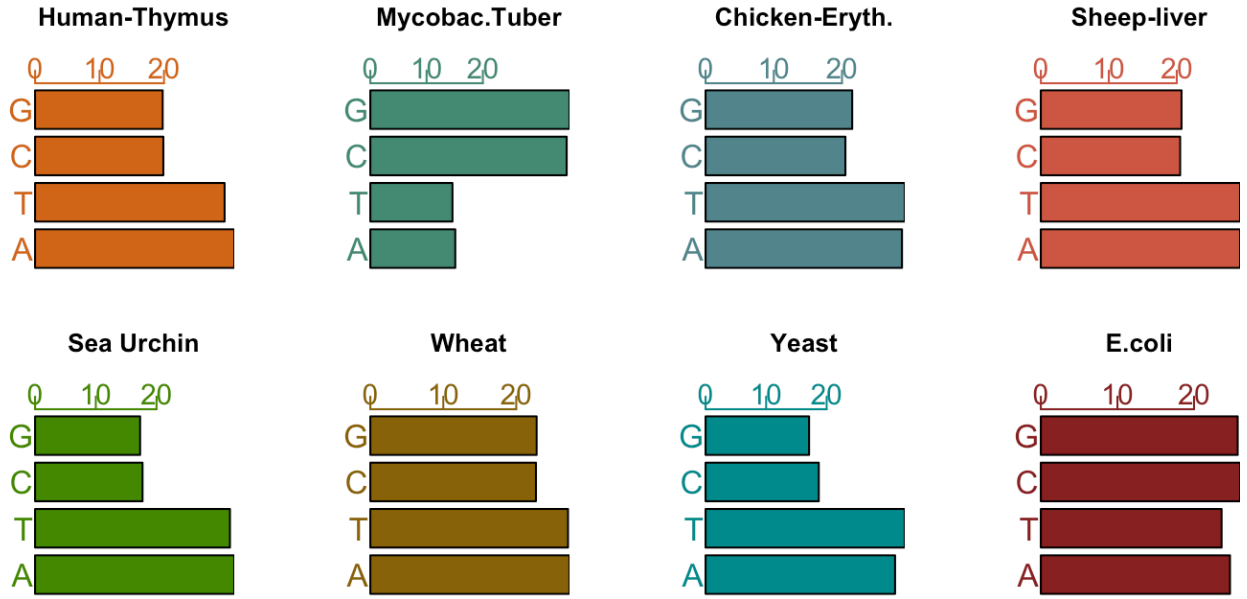
The most important pattern in the nucleotide frequencies was discovered by Chargaff

Before DNA sequencing was available, using the weight of the molecules, he asked whether the nucleotides occurred at equal frequencies. He called this the tetranucleotide hypothesis: $p_{\mathrm{A}} = p_{\mathrm{C}} = p_{\mathrm{G}} = p_{\mathrm{T}}$.

Unfortunately, Chargaff only published the *percentages* of the mass present in different organisms for each of the nucleotides, not the measurements themselves.

```
load("~/Books/CUBook/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
```



Barplots for the different rows in `ChargaffTable`. Can you spot the pattern?

- Do these data seem to come from equally likely multinomial categories?
- Can you suggest an alternative pattern?
- Can you do a quantitative analysis of the pattern, perhaps inspired by the simulations above?

Chargaff *saw* the answer to this question and postulated a pattern called *base pairing*, which ensured a perfect match of the amount of adenine (A) in the DNA of an organism to the amount of thymine (T). Similarly, whatever the amount of guanine (G), the amount of cytosine (C) would be the same. This is now called Chargaff's rule. On the other hand, the amount of C/G in an organism could be quite different from that of A/T, with no obvious pattern across organisms. Based on Chargaff's rule, we might define a statistic

$$(p_{\mathrm{C}} - p_{\mathrm{G}})^2 + (p_{\mathrm{A}} - p_{\mathrm{T}})^2,$$

summed over all rows of the table.

We are going to look at a comparison between the data and what would occur if the nucleotides were 'exchangeable', in the sense that the probabilities observed in each row were in no particular order, so that there were no special relationship between the proportions of As and Ts, or between those of Cs and Gs.

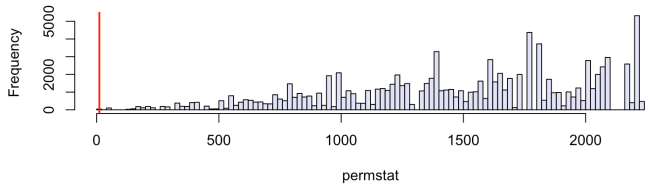
Histogram of our statistic `statChf` computed from simulations using per-row permutations of the columns. The value it yields for the observed data is shown by the red line.

```
statChf = function(x){
  sum((x[, "C"] - x[, "G"])^2 + (x[, "A"] - x[, "T"])^2)
}
chfstat = statChf(ChargaffTable)
permstat = replicate(100000, {
     permuted = t(apply(ChargaffTable, 1, sample))
     colnames(permuted) = colnames(ChargaffTable)
     statChf(permuted)
})
pChf = mean(permstat <= chfstat)
pChf
```

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

Histogram of our statistic `statChf` computed from simulations using per-row permutations of the columns. The value it yields for the observed data is shown by the red line.

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



The histogram shows that it is quite rare to have a value as small as the observed 11.1, where the red line is drawn. The probability of observing a value as small or smaller is `pChf` = 1.310^{-4}.

Thus the data strongly support Chargaff's insight.

When computing `pChf`, we only looked at the values in the null distribution smaller than the observed value. Why did we do this in a one-sided way here?

Two categorical variables

Up to now, we have visited cases where the data are taken from a sample that can be classified into different boxes: the binomial for Yes/No binary boxes and the multinomial distribution for categorical variables such as A, C, G, T or different genotypes such aa, aA, AA. However it might be that we measure two (or more) categorical variables on a set of subjects, for instance eye color and hair color. We can then cross-tabulate the counts for every combination of eye and hair color. We obtain a table of counts called a **contingency table**. This concept is very useful for many biological data types.

```
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
```

Explore the `HairEyeColor` object in R. What data type, shape and dimensions does it have?

It is a numeric array with three dimensions:

```
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"
```

```
? HairEyeColor
```

Color blindness and sex

Deuteranopia is a form of red-green color blindness due to the fact that medium wavelength sensitive cones (green) are missing. A deuteranope can only distinguish 2 to 3 different hues, whereas somebody with normal vision sees 7 different hues. A survey for this type of color blindness in human subjects produced a two-way table crossing color blindness and sex.

```
load("~/Books/CUBook/data/Deuteranopia.RData")
Deuteranopia
```

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

How do we test whether there is a relationship between sex and the occurrence of color blindness? We postulate the null model with two independent binomials: one for sex and one for color blindness. Under this model we can estimate all the cells' multinomial probabilities, and we can compare the observed counts to the expected ones. This is done through the `chisq.test` function in R.

```
chisq.test(Deuteranopia)
```

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

The small p value tells us that we should expect to see such a table with only a very small probability under the null model – i.e., if the fractions of deuteranopic color blind among women and men were the same.

We'll see another test for this type of data called Fisher's exact test (also known as the hypergeometric test).

This test is widely used for testing the over-representations of certain types of genes in a list of significantly expressed ones.

A special multinomial: Hardy-Weinberg equilibrium

A special multinomial with three possible levels created by combining two alleles M and N. Overall frequency of allele M in the population is p, so that of N is $q = 1 - p$.

The Hardy-Weinberg model looks at the relationship between p and q if there is independence of the frequency of both alleles in a genotype, the so-called **Hardy-Weinberg equilibrium** (HWE).

This would be the case if there is random mating in a large population with equal distribution of the alleles among sexes. The probabilities of the three genotypes are then as follows:

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

We only observe the frequencies $(n_{\mathrm{MM}},\ n_{\mathrm{MN}},\ n_{\mathrm{NN}})$ for the genotypes MM, MN, NN and the total number $S = n_{\mathrm{MM}} + n_{\mathrm{MN}} + n_{\mathrm{NN}}$. We can write the likelihood, i.e., the probability of the observed data when the probabilities of the categories are given by @ref(eq:HWE), using the multinomial formula

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

and the log-likelihood under HWE

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

The value of p that maximizes the loglikelihood is

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

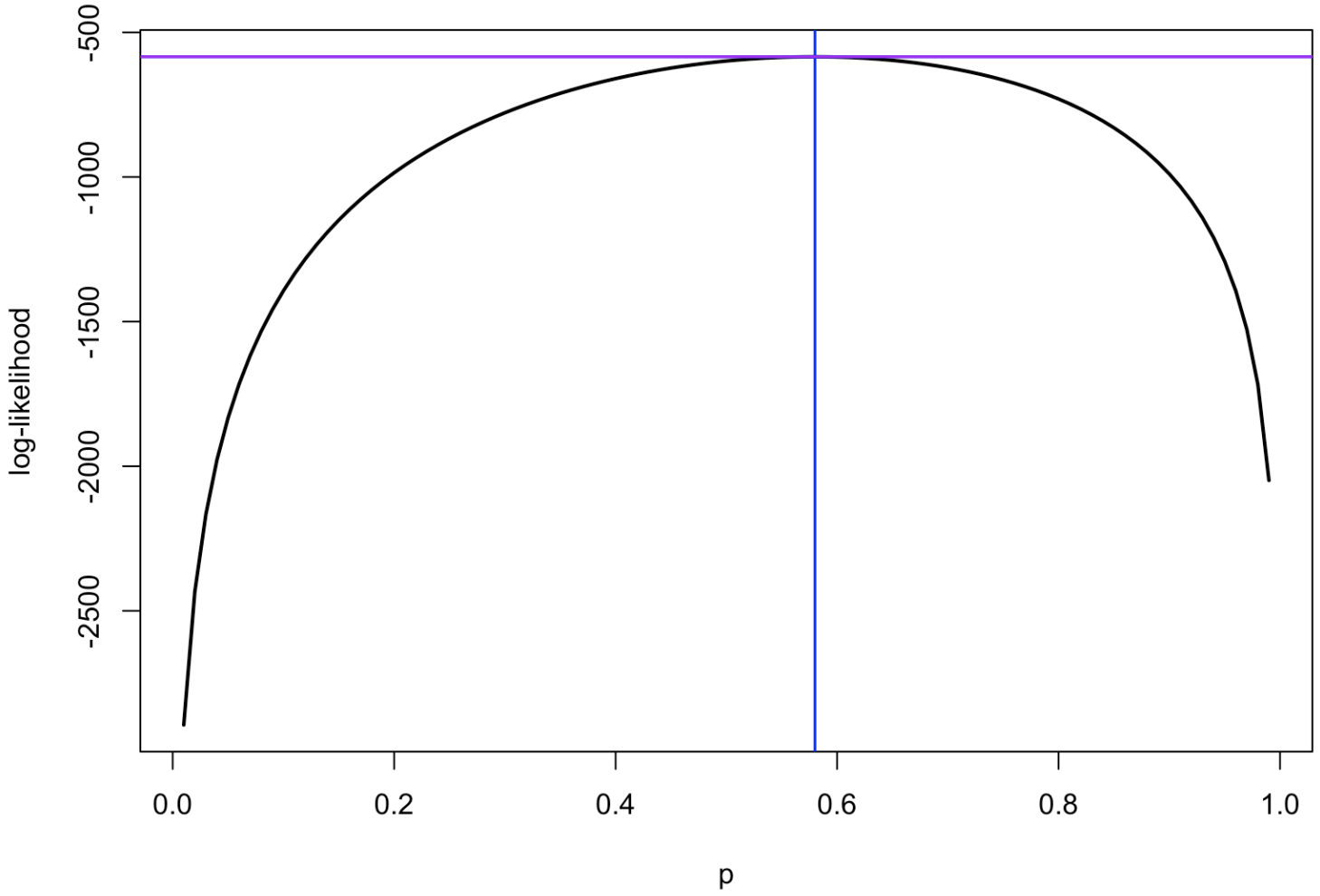
Given the data $(n_{\mathrm{MM}},\ n_{\mathrm{MN}},\ n_{\mathrm{NN}})$, the log-likelihood L is a function of only one parameter, p. The Figure shows this log-likelihood function for different values of p for the 216th row of the Mourant data, computed in the following code.

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

```
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")
```



Plot of the log-likelihood for the data.

The maximum likelihood estimate for the probabilities in the multinomial is also obtained by using the observed frequencies as in the binomial case, however the estimates have to account for the relationships between the three probabilities. We can compute $\hat{p}_{\mathrm{MM}}, \hat{p}_{\mathrm{MN}}$ and $\hat{p}_{\mathrm{NN}}$ using the `af` function from the **HardyWeinberg** package.

```
phat  =  af(c(nMM, nMN, nNN))
phat
```

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

```
pMM   =  phat^2
qhat  =  1 - phat
```

The expected values under Hardy-Weinberg equilibrium are then

```
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
```

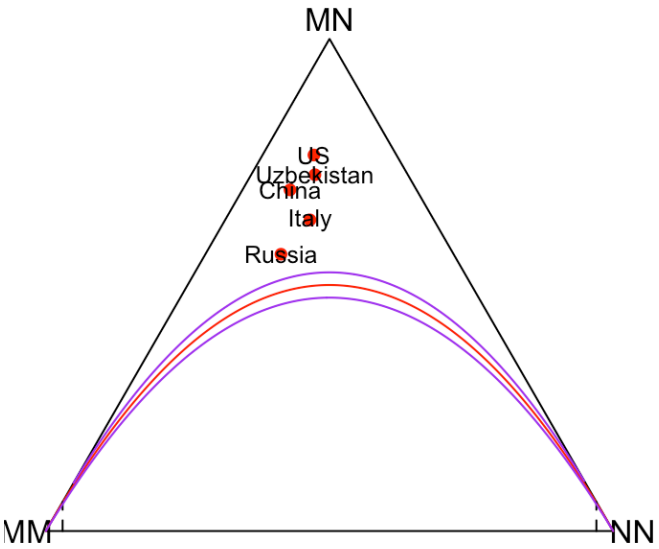
A visual evaluation of the goodness-of-fit of Hardy-Weinberg was designed by de Finetti

It places every sample at a point whose coordinates are given by the proportions of each of the different alleles.

Visual comparison to the Hardy-Weinberg equilibrium

We use the `HWTernaryPlot` functon to display the data and compare it to Hardy-Weinberg equilibrium graphically.

```
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)
```



This **de Finetti plot** shows the points as barycenters of the three genotypes using the frequencies as weights on each of the corners of the triangle. The Hardy-Weinberg model is the red curve, the acceptance region is between the two purple lines. We see

that the US is the furthest from being in HW equilibrium.

Make the ternary plot as in the code above, then add the other data points to it, what do you notice? You could back up your discussion using the `HWChisq` function.

```
HWTernaryPlot(genotypeFrequencies[-pops, ], alpha = 0.0001,
   newframe = FALSE, cex = 0.5)
```

```
newgf = round(genotypeFrequencies / 50)
HWTernaryPlot(newgf[pops, ],
        markerlab = Mourant$Country[pops],
        alpha = 0.0001, curvecols = c("red", rep("purple", 4)),
        mcex = 0.75, vertex.cex = 1)
```

Concatenating several multinomials: sequence motifs and logos

The Kozak Motif occurs close to the start codon ATG of a coding region.

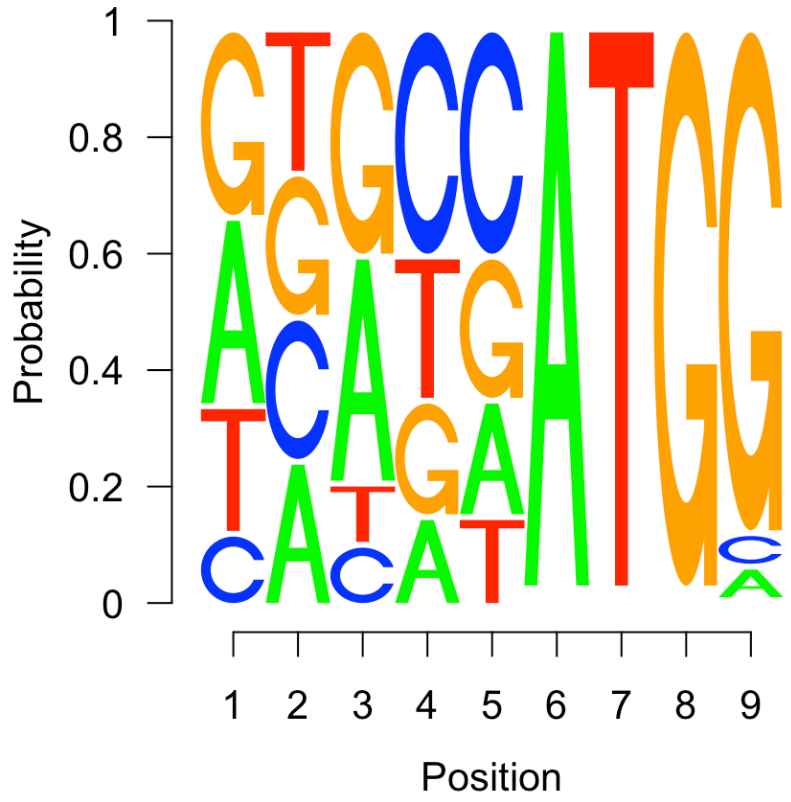
The start codon itself always has a fixed spelling but in positions 5 to the left of it, there is a nucleotide pattern in which the letters are quite far from being equally likely.

We summarize this by giving the **position weight matrix** (PWM) or **position-specific scoring matrix** (PSSM), which provides the multinomial probabilities at every position. This is encoded graphically by the **sequence logo** .

```
library("seqLogo")
load("~/Books/CUBook/data/kozak.RData")
kozak
```

```
##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## A 0.33 0.25  0.4 0.15 0.20    1    0    0 0.05
## C 0.12 0.25  0.1 0.40 0.40    0    0    0 0.05
## G 0.33 0.25  0.4 0.20 0.25    0    0    1 0.90
## T 0.22 0.25  0.1 0.25 0.15    0    1    0 0.00
```

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



Here is a diagram called a sequence logo for the position dependent multinomial used to model the Kozak motif. It codifies the amount of variation in each of the positions on a log scale. The large letters represent positions where there is no uncertainty about which nucleotide occurs.

Over the last sections, we've seen how the different "boxes" in the multinomial distributions we have encountered very rarely have equal probabilities. In other words, the parameters $p_1, p_2, \ldots$ are often different, depending on what is being modeled.

Examples of multinomials with unequal frequencies include the twenty different amino acids, blood types and hair color.

If we have multiple categorical variables, we have seen that they are rarely independent (sex and colorblindness, hair and eye color, ...). We will see later that we can explore the patterns in these dependencies by using multivariate decompositions of the contingency tables. Here, we'll look at an important special case of dependencies between categorical variables: those that occur along a sequence (or "chain") of categorical variables, e.g., over time or along a biopolymer.

Example: occurrence of a nucleotide pattern in a genome

This case study of the distributions of the distances between instances of a specific motif in genome sequences will also allow us to explore specific genomic sequence manipulations in Bioconductor.

The **Biostrings** package provides tools for working with sequence data. The essential data structures, or *classes* as they known in **R**, are *DNAString* and *DNAStringSet*. These enable us to work with one or multiple DNA sequences efficiently ⊕▢The **Biostrings** package also contains additional classes for representing amino acid and general biological strings..

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

The first line prints genetic code information, the second one returns IUPAC nucleotide ambiguity codes. The third line lists all the vignettes available in the **Biostrings** package, the fourth display one particular vignette.

```
GENETIC_CODE
IUPAC_CODE_MAP
vignette(package = "Biostrings")
vignette("BiostringsQuickOverview", package = "Biostrings")
```

This last command will open a list in your browser window from which you can access the documentation. The **BSgenome** package provides access to many genomes, and you can access the names of the data packages that contain the whole genome sequences by typing

```
library("BSgenome")
ag = available.genomes()
length(ag)
```

```
## [1] 93
```

```
ag[1:2]
```

```
## [1] "BSgenome.Alyrata.JGI.v1"
## [2] "BSgenome.Amellifera.BeeBase.assembly4"
```

We are going to explore the occurrence of the `AGGAGGT` motif in the geonome of E.coli. We use the genome sequence of one particular strain, **Escherichia coli** str `K12 substr.DH10B`, whose NCBI accession number is NC_010473.

```
library("BSgenome.Ecoli.NCBI.20080805")
Ecoli
shineDalgarno = "AGGAGGT"
ecoli = Ecoli$NC_010473
```

We can count the pattern's occurrence in windows of width 50000 using the `countPattern` function.

```
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
```

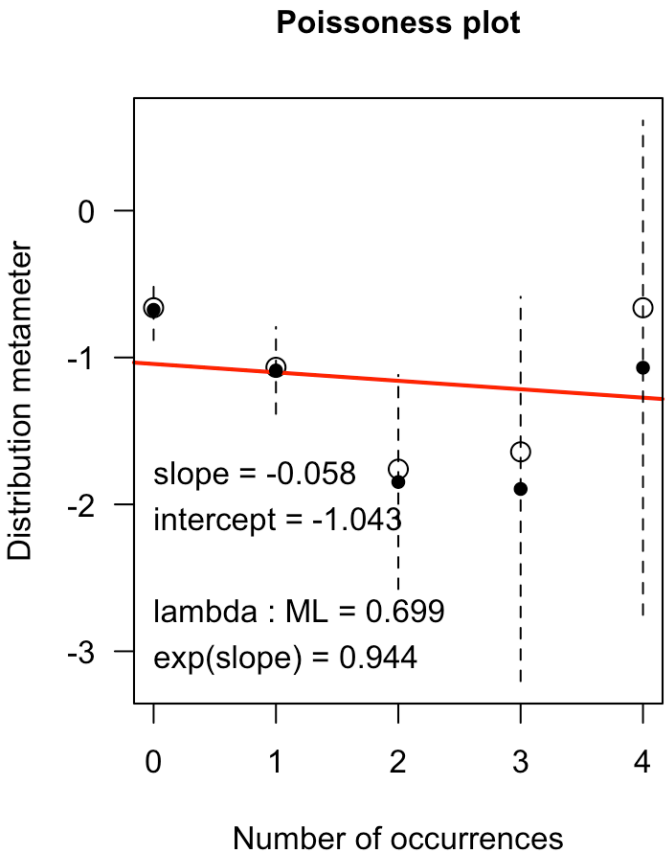
Fitting the data

The Poisson is a good candidate, as a quantitative and graphical evaluation for these data shows.

```
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")
```



Evaluation of a Poisson model for motif counts along the sequence \texttt{Ecoli$NC_010473}.

We can inspect the matches using the `matchPattern` function.

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

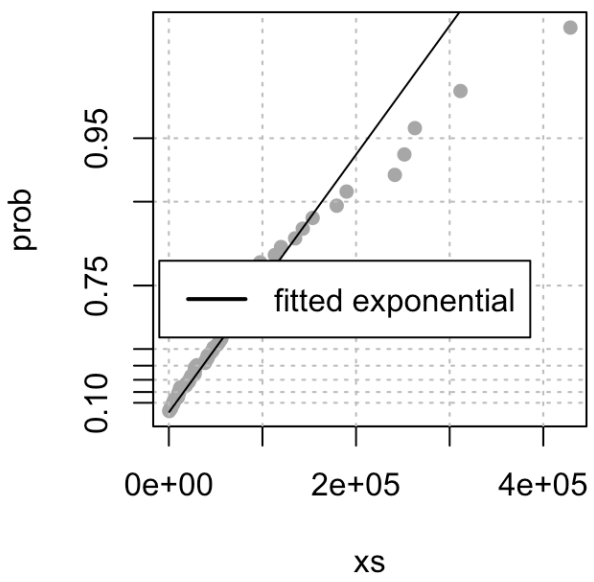
You can type `sdMatches` in the R command line to obtain a summary of this object. It contains the locations of all 1 pattern matches, represented as a set of so-called *views* on the original sequence. Now what are the distances between them?

```
betweenmotifs = gaps(sdMatches)
```

So these are in fact the 66 complementary regions. Now let's find a model for the distribution of the gap sizes between motifs. If the motifs occur at random locations, we expect the gap lengths to follow an exponential distribution.

The code below assesses this assumption. If the exponential distribution is a good fit, the points should lie roughly on a straight line. The exponential distribution has one parameter, the rate, and the line with slope corresponding to an estimate from the data is also shown.

```
library("Renext")
expplot(width(betweenmotifs), rate = 1/mean(width(betweenmotifs)),
        labels = "fitted exponential")
```



Evaluation of fit to the exponential distribution of the gaps between the motifs.

There appears to be a slight deviation from the fitted line in the figure at the right tail of the distibution, i.e., for the largest values.

Modeling in the case of dependencies

Nucleotide sequences are often dependent: the probability of seing a certain nucleotide at a given position tends to depend on the surrounding sequence. Here we are going to put into practice dependency modeling using a **Markov chain**. We are going to look at regions of chromosome 8 of the human genome and try to discover differences between regions called CpG islands and the rest.

We use data that tell us where the start and end points of the islands are in the genome and look at the frequencies of nucleotides and of the digrams 'CG', 'CT', 'CA', 'CC'. So we can ask whether there are dependencies between the nucleotide occurrences and if so, how to model them.

```
library("BSgenome.Hsapiens.UCSC.hg19")
chr8  =  Hsapiens$chr8
CpGtab = read.table("~/Books/CUBook/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
```

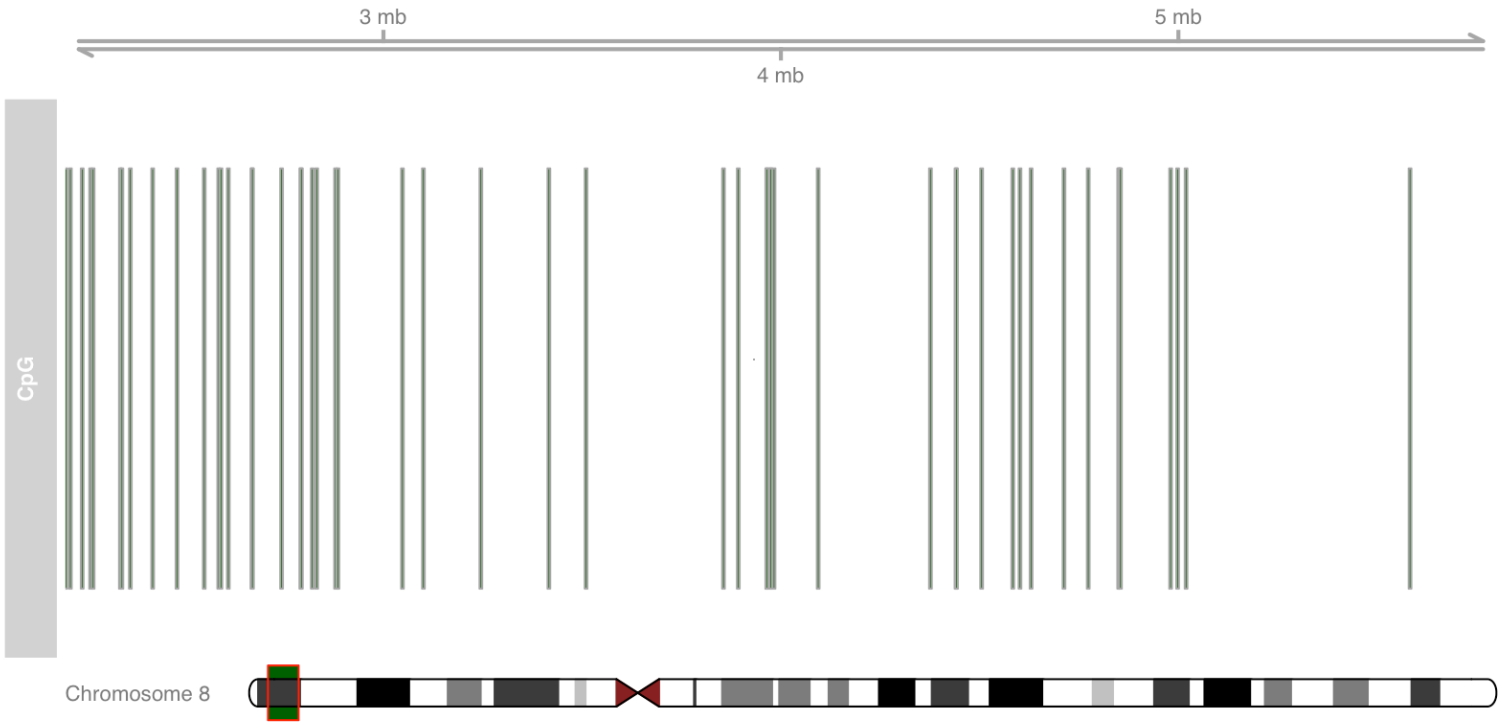
```
irCpG = with(dplyr::filter(CpGtab, chr == "chr8"),
         IRanges(start = start, end = end))
```

In the line above, we subset (`filter`) the data frame `CpGtab` to only chromosome 8, and then we create an *IRanges* object whose start and end positions are defined by the equally named columns of the data frame. In the `IRanges` function call (which constructs the object from its arguments), the first `start` is the argument name of the function, the second `start` refers to the column in the data frame obtained as an output from `filter`; and similarly for `end`. *IRanges* is a general container for mathematical intervals. We create the biological context with the next line.

```
grCpG = GRanges(ranges = irCpG, seqnames = "chr8", strand = "+")
genome(grCpG) = "hg19"
```

Now let's visualize; see the output :

```
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")
```



Gviz plot of CpG locations in a selected region of chromosome 8.

We now define so-called views on the chromosome sequence that correspond to the CpG islands, `irCpG`, and to the regions in between (`gaps(irCpG)`). The resulting objects `CGIview` and `NonCGIview` only contain the coordinates, not the sequences themselves (these stay in the big object `Hsapiens$chr8`), so they are fairly lightweight in terms of storage.

```
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.

```
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
```

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

For a four state Markov chain as we have, we define the transition matrix as a matrix where the rows are the `from` state and the columns are the `to` state.

```
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"))
```

We use the counts of numbers of transitions of each type to compute frequencies and put them into two matrices.

The transition probabilities are probabilities so the rows need to sum to 1.

```
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
```

```
MN = TnI / rowSums(TnI)
MN
```

```
##           A         C          G         T
## A 0.3351380 0.1680007 0.23080886 0.2660524
## C 0.3641054 0.2464366 0.04177094 0.3476871
## G 0.2976696 0.2029017 0.24655406 0.2528746
## T 0.2265813 0.1972407 0.24117528 0.3350027
```

Are the transitions different in the different rows? This would mean that, for instance, $P(\mathtt{A} \mid \mathtt{C}) \neq P(\mathtt{A} \mid \mathtt{T})$?

The transitions are different. For instance, the transitions from C to A and T to A for in the islands (MI) transition matrix seem very different (0.201 versus 0.098).

Are the relative frequencies of the different nucleotides different in CpG islands compared to elsewhere ?

```
freqIsl=alphabetFrequency(seqCGI,baseOnly=TRUE,collapse=TRUE)[1:4]
freqIsl / sum(freqIsl)
```

```
##         A         C         G         T 
## 0.1781693 0.3201109 0.3206298 0.1810901
```

```
freqNon=alphabetFrequency(seqNonCGI,baseOnly=TRUE,collapse=TRUE)[1:4]
freqNon / sum(freqNon)
```

```
##         A         C         G         T 
## 0.3008292 0.1993832 0.1993737 0.3004139
```

This shows an inverse pattern: in the CpG islands, C and G have frequencies around 0.32, whereas in the non-CpG islands, we have A and T that have frequencies around 0.30.

How can we use these differences to decide whether a given sequence comes from a CpG island?

Use a χ^2-squared statistic to compare the frequencies between the observed and `freqIsl` and `freqNon` frequencies. For shorter sequences, this may not be sensitive enough, and a more sensitive approach is given below.

Given a sequence for which we do not know whether it is in a CpG island or not, we can ask what is the probability it belongs to a CpG island compared to somewhere else. We compute a score based on what is called the odds ratio. Let's do an example: suppose our sequence x is ACGTTATACTACG, and we want to decide whether it comes from a CpG island or not.

If we model the sequence as a first order Markov chain we can write, supposing that the sequence comes from a CpG island:

$$\begin{aligned} P_{\mathrm{i}}(x = \mathtt{ACGTTATACTACG}) = {} & P_{\mathrm{i}}(\mathtt{A})\, P_{\mathrm{i}}(\mathtt{AC})\, P_{\mathrm{i}}(\mathtt{CG})\, P_{\mathrm{i}}(\mathtt{GT})\, P_{\mathrm{i}}(\mathtt{TT}) \times \\ & P_{\mathrm{i}}(\mathtt{TA})\, P_{\mathrm{i}}(\mathtt{AT})\, P_{\mathrm{i}}(\mathtt{TA})\, P_{\mathrm{i}}(\mathtt{AC})\, P_{\mathrm{i}}(\mathtt{CG}). \end{aligned}$$

We are going to compare this probability to the probability for non-islands. As we saw above, these probabilities tend to be quite different. We will take their ratio and see if it is larger or smaller than 1. These probabilties are going to be products of many small terms and become very small. We can work around this by taking logarithms.

$$\log \frac{P(x \mid \text{island})}{P(x \mid \text{non-island})} =$$

$$\log\left(\frac{P_{\mathrm{i}}(\mathtt{A})\, P_{\mathrm{i}}(\mathtt{A} \to \mathtt{C})\, P_{\mathrm{i}}(\mathtt{C} \to \mathtt{G})\, P_{\mathrm{i}}(\mathtt{G} \to \mathtt{T})\, P_{\mathrm{i}}(\mathtt{T} \to \mathtt{T})\, P_{\mathrm{i}}(\mathtt{T} \to \mathtt{A})}{P_{\mathrm{n}}(\mathtt{A})\, P_{\mathrm{n}}(\mathtt{A} \to \mathtt{C})\, P_{\mathrm{n}}(\mathtt{C} \to \mathtt{G})\, P_{\mathrm{n}}(\mathtt{G} \to \mathtt{T})\, P_{\mathrm{n}}(\mathtt{T} \to \mathtt{T})\, P_{\mathrm{n}}(\mathtt{T} \to \mathtt{A})} \times \right.$$
$$\left. \frac{P_{\mathrm{i}}(\mathtt{A} \to \mathtt{T})\, P_{\mathrm{i}}(\mathtt{T} \to \mathtt{A})\, P_{\mathrm{i}}(\mathtt{A} \to \mathtt{C})\, P_{\mathrm{i}}(\mathtt{C} \to \mathtt{G})}{P_{\mathrm{n}}(\mathtt{A} \to \mathtt{T})\, P_{\mathrm{n}}(\mathtt{T} \to \mathtt{A})\, P_{\mathrm{n}}(\mathtt{A} \to \mathtt{C})\, P_{\mathrm{n}}(\mathtt{C} \to \mathtt{G})} \right)$$

This is the **log-likelihood ratio** score. To speed up the calculation, we compute the log-ratios $\log(P_{\mathrm{i}}(\mathtt{A})/P_{\mathrm{n}}(\mathtt{A})), \ldots, \log(P_{\mathrm{i}}(\mathtt{T} \to \mathtt{A})/P_{\mathrm{n}}(\mathtt{T} \to \mathtt{A}))$ once and for all and then

sum up the relevant ones to obtain our score.

```
alpha = log((freqIsl/sum(freqIsl)) / (freqNon/sum(freqNon)))
beta  = log(MI / MN)
```

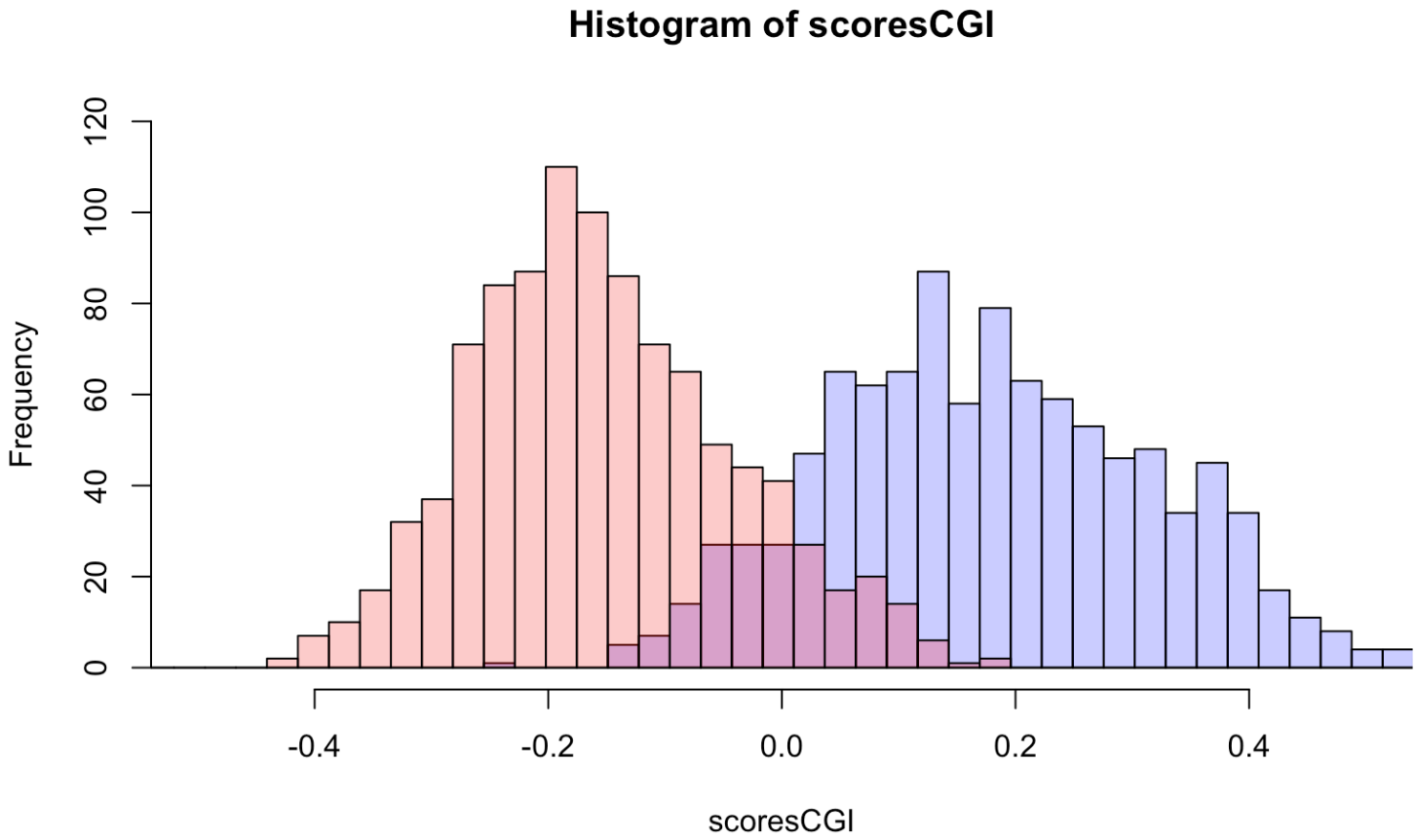
```
x = "ACGTTATACTACG"
scorefun = function(x) {
  s = unlist(strsplit(x, ""))
  score = alpha[s[1]]
  if (length(s) >= 2)
    for (j in 2:length(s))
      score = score + beta[s[j-1], s[j]]
  score
}
scorefun(x)
```

```
##          A
## -0.2824623
```

In the code below, we pick sequences of length `len = 100` out of the 2855 sequences in the `seqCGI` object, and then out of the 2854 sequences in the `seqNonCGI` object (each of them is a *DNAStringSet*). In the first three lines of the `generateRandomScores` function, we drop sequences that contain any letters other than A, C, T, G; such as "." (a character used for undefined nucleotides). Among the remaining sequences, we sample with probabilities proportional to their length minus `len` and then pick subsequences of length `len` out of them. The start points of the subsequences are sampled uniformly, with the constraint that the subsequences have to fit in.

```
generateRandomScores = function(s, len = 100, B = 1000) {
  alphFreq = alphabetFrequency(s)
  isGoodSeq = rowSums(alphFreq[, 5:ncol(alphFreq)]) == 0
  s = s[isGoodSeq]
  slen = sapply(s, length)
  prob = pmax(slen - len, 0)
  prob = prob / sum(prob)
  idx  = sample(length(s), B, replace = TRUE, prob = prob)
  ssmp = s[idx]
  start = sapply(ssmp, function(x) sample(length(x) - len, 1))
  scores = sapply(seq_len(B), function(i)
    scorefun(as.character(ssmp[[i]][start[i]+(1:len)]))
  )
  scores / len
}
scoresCGI    = generateRandomScores(seqCGI)
scoresNonCGI = generateRandomScores(seqNonCGI)
```

```
br = seq(-0.6, 0.7, length.out = 50)
h1 = hist(scoresCGI,    breaks = br, plot = FALSE)
h2 = hist(scoresNonCGI, breaks = br, plot = FALSE)
plot(h1, col = rgb(0, 0, 1, 1/4), xlim = c(-0.5, 0.5), ylim=c(0,120))
plot(h2, col = rgb(1, 0, 0, 1/4), add = TRUE)
```



Island and non-island scores as generated by the function `generateRandomScores`. This is the first instance of a **mixture** we encounter. We will revisit them in the Mixtures lecture

We can consider these our *training data*: from data for which we know the types, we can see whether our score is useful for discriminating.

Summary of this lecture

Basic yoga of statistics: how to go from the data back to the possible generating distributions.

Estimate the parameters that define these distributions.

Statistical models We showed some specific statistical models for experiments with categorical outcomes (binomial and multinomial).

Goodness of fit Different visualizations show whether our data could be fit by a fair four box multinomial model.

We encountered the chi-square statistic and saw how to compare simulation and theory using a qq-plot.

Estimation We explained maximum likelihood and Bayesian estimation procedures.

Prior and posterior distributions When assessing data of a type that has been been previously studied, such as haplotypes, it can be beneficial to compute the posterior distribution of the data. This incorporates uncertainty in the decision making, by a simple computation.

The choice of the prior has little effect on the result as long as there is sufficient data.

CpG islands and Markov chains We saw how dependencies along DNA sequences can be modeled by Markov chain transitions. We used this to build scores based on

likelihood ratios that enable us to see whether long DNA sequences come from CpG islands or not. When we made the histogram of scores, we see in the Figure a noticeable feature: it seemed to be made of two pieces.

This **bimodality** was our first encounter with mixtures, they are the subject of lecture on Mixtures.

This is the first instance of building a model on some training data: sequences which we knew were in CpG islands, that we could use later to classify new data. We will develop a much more complete way of doing this in the lecture on supervised methods.