

MSMB, Lecture 2: Statistical Modeling

Susan Holmes and Wolfgang Huber

1 Oct 2019

Statistical Modeling



- Previous lecture: knowledge of a generative model and the values of the *parameters* provided us with probability distributions of the possible data
- This enhanced our decision making – for instance, whether we had really found an epitope.
- In many real situations, neither generative distribution nor parameters are known.
- We need to estimate them from the data we have.

Statistical modeling works from the data *upwards* to a model that *might* plausibly explain the data.

- This lecture will show us some of the distributions that serve as building blocks for statistical model building and inference in such situations.
- The examples in this lecture are all parametric (i.e., the models only have a small number of unknown parameters), the principles do generalize.

Goals for this lecture

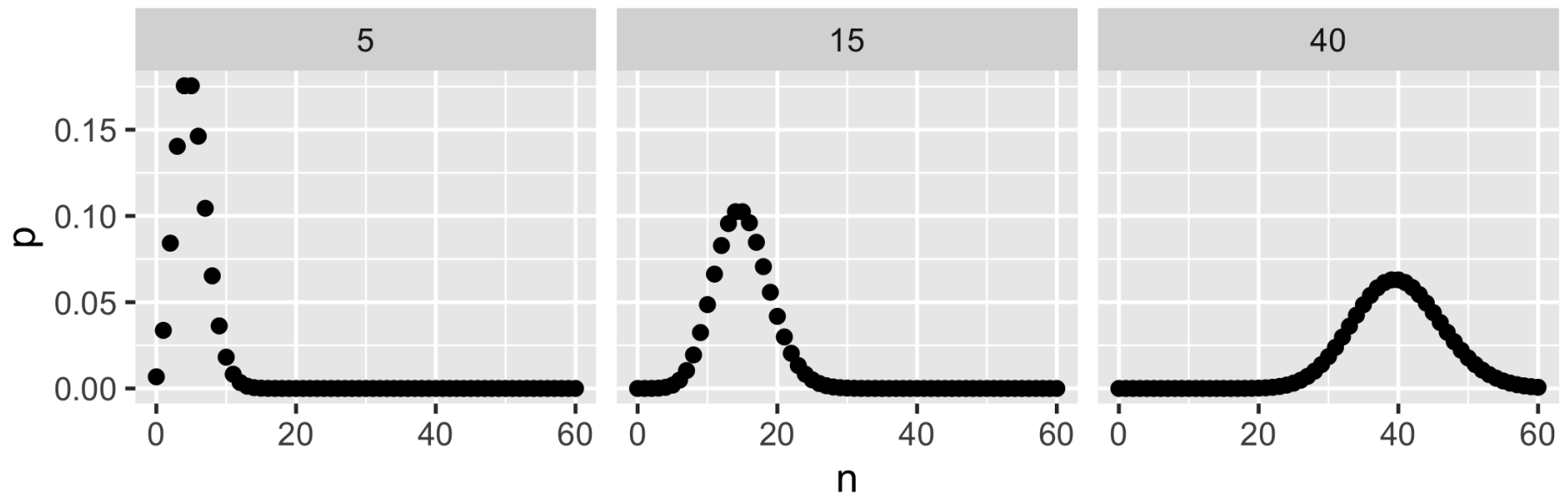
- See the difference between probabilistic modeling and statistics.
- Fit distributions to data histograms.
- See why we have to use *estimators* instead of actual parameters.
- Give examples of estimating procedures such as *maximum likelihood*.
- Show we can use statistical models and estimation for evaluating dependencies in binomial and multinomial distributions.
- Explore a simple model class for (sequential) data: Markov chains.
- Lab: show concrete applications involving genomes and Bioconductor classes dedicated to the manipulation of genomic data as strings or sequences.

Parameters are the key

Examples of parameters

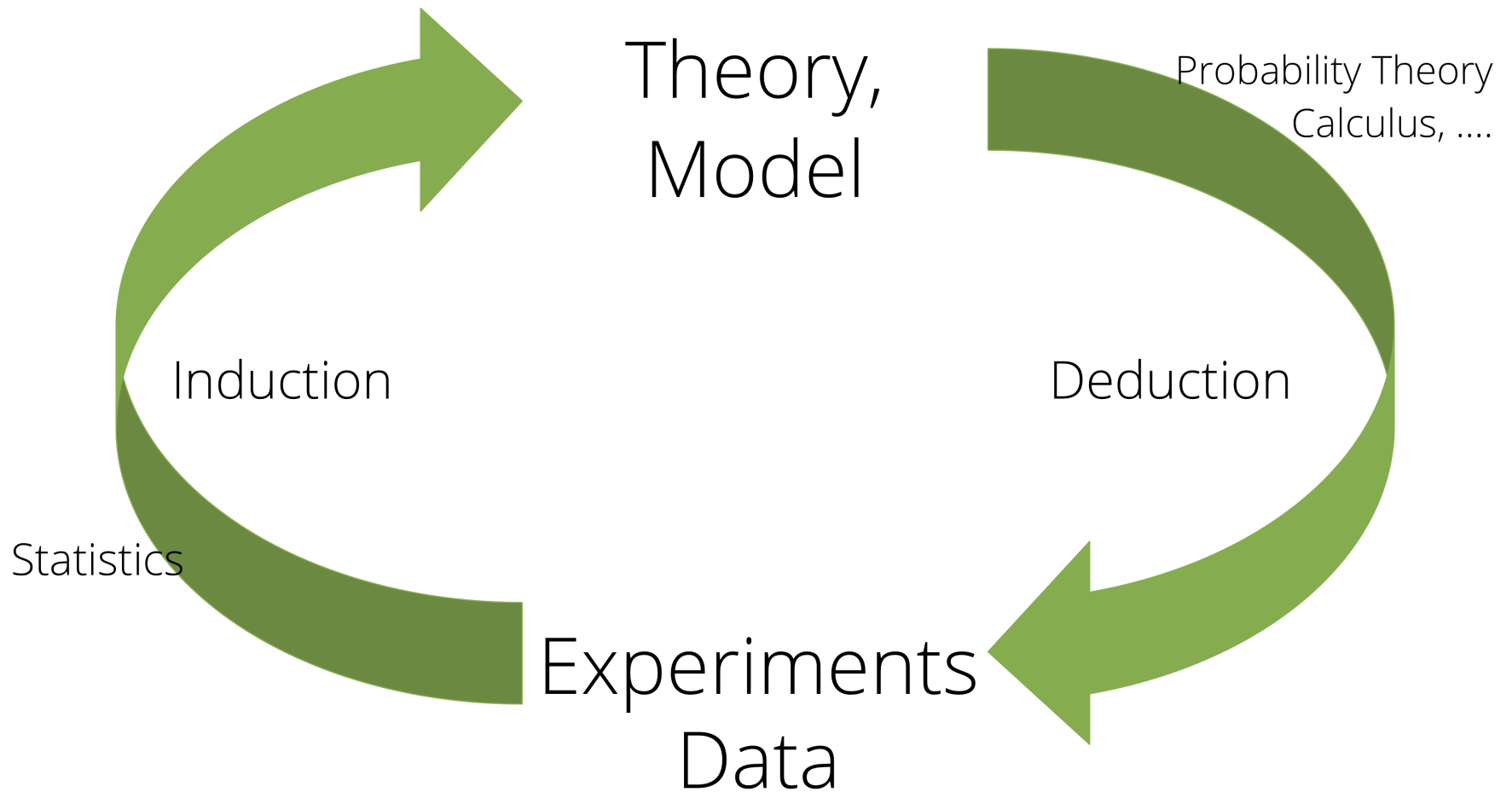
- A single parameter λ defines a Poisson distribution.
- Other distributions also have more than one parameter, e.g., binomial (n, p) , normal (μ, σ) .

```
library("dplyr")
library("ggplot2")
lapply(c(5, 15, 40), function(lambda)
  tibble(n = 0:60, p = dpois(n, lambda), lambda = lambda)) %>%
  bind_rows %>%
  ggplot(aes(x = n, y = p)) + geom_point() + facet_grid(~ lambda)
```



In the Epitope example, knowing the parameter gave us our probability model and enabled us to test a null hypothesis based on the data at hand.

Difference between statistical and probabilistic models



1

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

Now suppose that we know the number of patients and the length of the proteins (these are given by the experimental design). We then observe data, but we suppose now that the distribution itself and the false positive rate is unknown.

We need to go *up* from the data we observe to estimate both a probability model \mathcal{F} (Poisson, normal, binomial, ...) and eventually missing parameter(s) for that model.

An example of simple statistical modeling

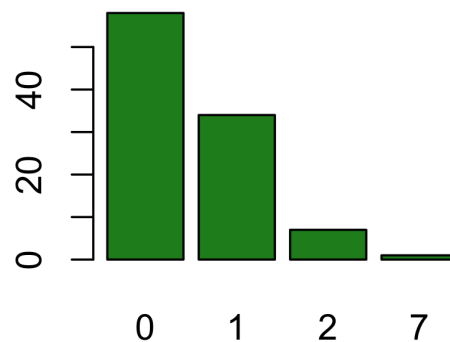
There are two parts to the modeling process:

1. A suitable probability **distribution** for the data generating process
2. A feasible way to **estimate its parameters**

As we saw in Lecture 1, discrete count data can often be modeled by elementary probability distributions such as binomial, multinomial or Poisson distributions. For continuous measurements, the normal distribution is the most elementary building block. Realistic distributions can also be complicated mixtures of these elementary ones (more on this in a later lecture).

Let's revisit the epitope data example from the previous chapter; we again remove the tricky outlier.

```
load(input_dir("data/e100.RData"))  
barplot(table(e100), space = 0.2, col = "forestgreen")
```



```
bad = (e100 > 3)  
library("assertthat")
```



```
assert_that(sum(bad) == 1)
e99 = e100[!bad]
```

Goodness-of-fit: visual and quantitative evaluation

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

One visual diagram is known as the rootogram (Cleveland 1988). It hangs the observed values from the theoretical values (red points). If the counts correspond exactly to their theoretical values, the bottom of the boxes will align exactly with the horizontal axis.

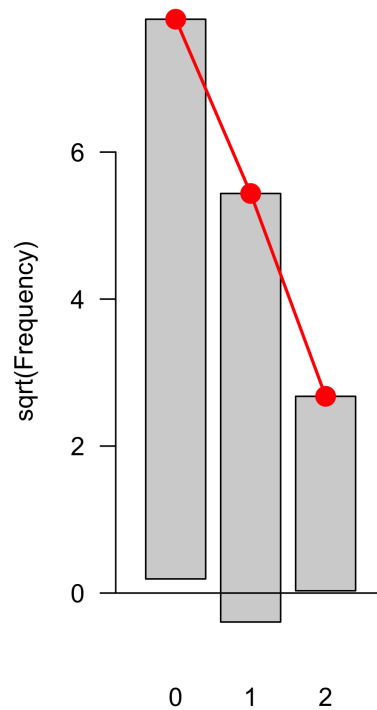
```
library("vcd")
gfl = goodfit(e99, "poisson")
gfl$par
```

```
## $lambda
## [1] 0.4848485
```

gfl

```
##
## Observed and fitted values for poisson distribution
## with parameters estimated by `ML`
##
## count observed      fitted pearson residual
##      0         58 60.963259      -0.3795207
##      1         34 29.557944       0.8170469
##      2          7  7.165562      -0.5078572
```

```
rootogram(gfl, xlab = "")
```



Calibrating our expectations

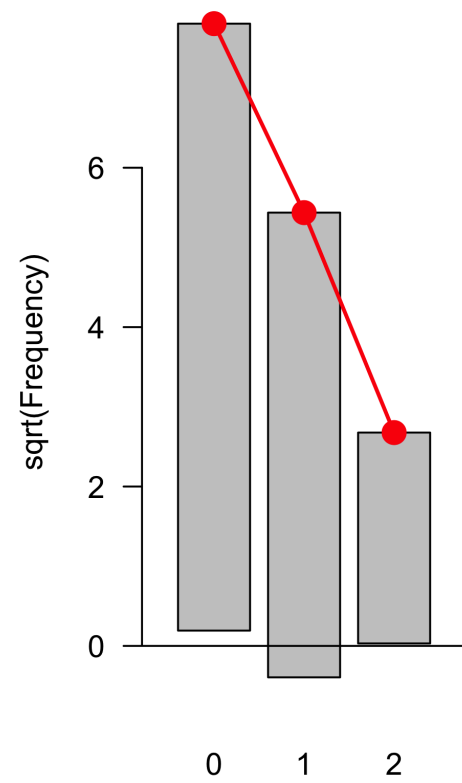
To see what such a plot looks like with data that we know to come from a Poisson distribution, use $\lambda = 0.5$ to generate 100 numbers sampled from a Poisson distribution and draw their rootogram.

```
simdat = rpois(100, lambda = 0.5)
gf2 = goodfit( simdat, "poisson")
```

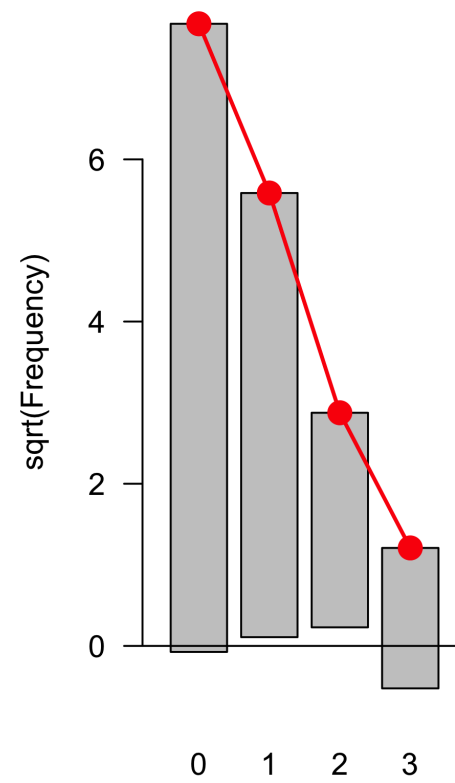
```
rootogram(gf1, main = "gf1", xlab = "")
```

```
rootogram(gf2, main = "gf2", xlab = "")
```

gf1



gf2



How do we know λ ?

Or: how did `goodfit` do it?

A Poisson distribution is completely determined by one parameter: λ (which is also its mean).

If we assume that the data follow a Poisson distribution, how do we estimate the parameter?

We choose the value that makes the observed data the most likely: **maximum likelihood estimation**

Estimators commonly wear a hat: $\hat{\lambda}$

The likelihood function

For each possible value of the parameter λ , we compute the probability of the data. This is the **likelihood function**

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

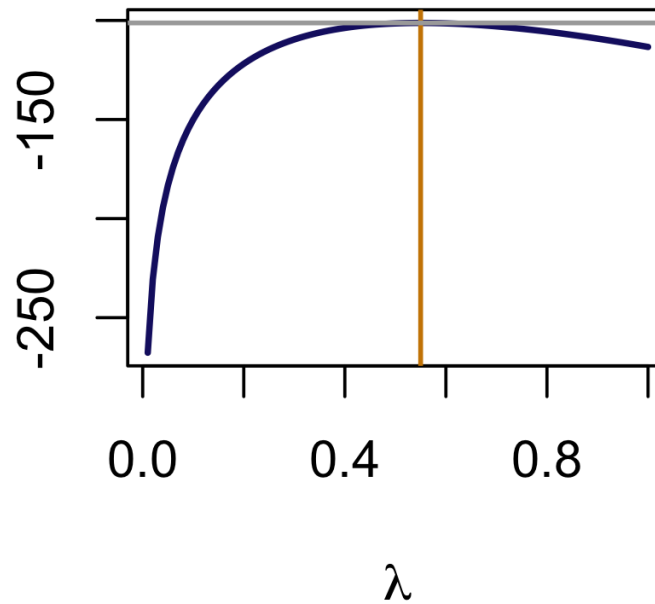
Often, we (equivalently) work with its logarithm. It is as simple as

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

Note that in this function, k is a vector.

Now we can compute the likelihood for a whole series of values from 0.05 to 0.95 by using the following:

```
lambdas = seq(0.01, 1, by = 0.01)  
ll = sapply(lambdas, loglikelihood, k = e100)  
plot(lambdas, ll, type = "l", col = "midnightblue", xlab = expression(lambda), ylab = "", lwd = 2)  
m0 = mean(e100)  
abline(v = m0, col = "orange3", lwd = 1.5)  
abline(h = loglikelihood(m0, k = e100), col = "darkgrey", lwd = 1.5)
```



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

```
gf = goodfit(e100, "poisson")
gf$par
```

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

```
m0
```

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

Classical statistics for classical data

Here is a traditional proof of our computational finding.

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

To find the λ that maximizes this, we compute the derivative in λ and set it to zero. (And verify that the second derivative is positive.)

$$\begin{aligned}\frac{d}{d\lambda} \log L &= -100 + \frac{1}{\lambda} \sum_{i=1}^{100} k_i \stackrel{!}{=} 0 \\ \Leftrightarrow \lambda &= \frac{1}{100} \sum_{i=1}^{100} k_i \quad (\text{the mean})\end{aligned}$$

So what is the value of modeling data with a known distribution and parameter estimates?

- Hypothesis testing: formulate a *null model* for the data (e.g., all observations are from the same distribution regardless group or treatment) and check whether this fits
- Alternatively (but not always) we also have an *alternative model* and can compare which one fits better
- Models are also concise but expressive representation of the data - a summary

Statistical models often combine deterministic and probabilistic parts

Prototypical: the **linear model**

$$y = ax + b + \varepsilon, \quad \varepsilon \sim N(0, s)$$
$$\Leftrightarrow y \sim N(\text{mean} = ax + b, \text{sd} = s)$$

for two continuous variables x and y , where we assume that y (the “response”) follows a distribution determined by x (the “explanatory variable”).

Model fit: the residuals ε should be independent, identically distributed normal

The linear model can be generalized by

- having more complicated deterministic parts
- have a different stochastic part: e.g., for count data, Poisson (\leftarrow RNA-seq and other types of high-throughput sequencing data), or binomial.

Binomial distributions and maximum likelihood

Two parameters:

- number of trials n - usually known
- probability p of seeing a “hit”, or a “1”, ... in a trial - often unknown

Example

Someone took a sample of 225 males and tested them for red-green colorblindness. They coded the data as 0 if the subject was not colorblind and 1 if he was. We can summarize the data `cb` by:

```
table(cb)
```

```
## cb
##   0   1
## 207  18
```

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

$\hat{p} = ?$

In this special case, your intuition may give you an estimate which is the maximum likelihood estimator (MLE).

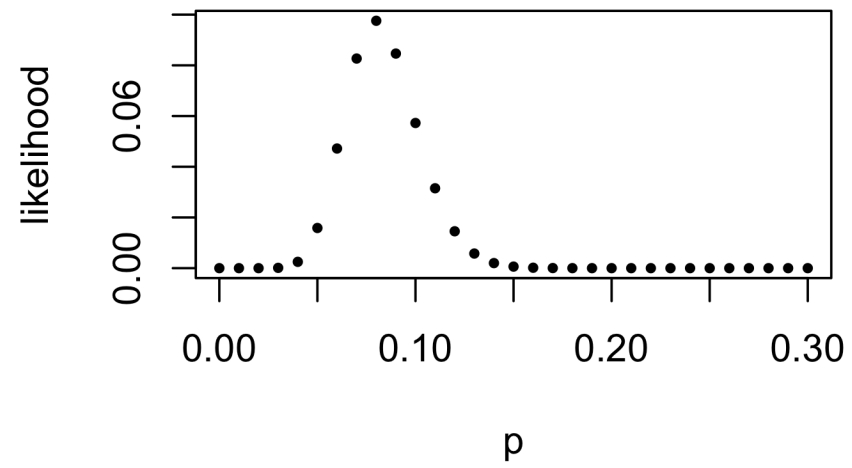
As before in the case of the Poisson, let's compute the likelihood for many possible p , plot it and see where the maximum falls.

```
probs = seq(0, 0.3, by = 0.01)
probs[1:4]
```

```
## [1] 0.00 0.01 0.02 0.03
```

```
likelihood = function(p, x)
  dbinom(sum(x), prob = p, size = length(x))

lv = likelihood(probs, cb)
plot(probs, lv, pch = 16, xlab = "p", ylab = "likelihood", cex = 0.6)
```



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

```
## [1] 0.08
```

Another example

The likelihood and the probability are the same mathematical function, just interpreted differently.

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

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

As the numbers involved are often very large or very small, e.g.,

```
choose(300, 40)
```

```
## [1] 9.793479e+49
```

we use the log likelihood, abbreviated by ℓ to give:

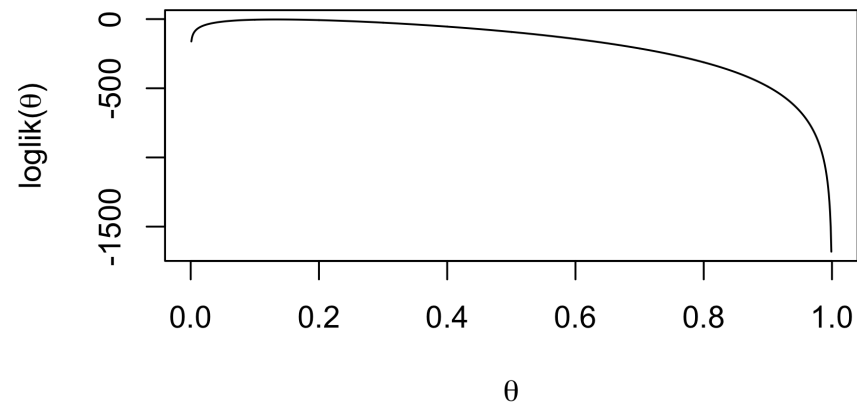
$$\log f(\theta | y) = \ell(\theta) = \log \binom{n}{y} + y \log \theta + (n - y) \log(1 - \theta).$$

Here's a little function we use to calculate it:

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

which we plot for the range of θ (the binomial probability) from 0 to 1.

```
thetas = seq(0, 1, by = 0.001)  
llv = loglikelihood(thetas, n = 300, k = 40)  
plot(thetas, llv,  
     type="l", xlab = expression(theta),  
     ylab = expression(paste("loglik(", theta, ")")), main = "")
```



```
thetas[which.max(llv)]
```

```
## [1] 0.133
```

- The maximum is consistent with intuition ($40/300=0.1333\dots$)
- But we see that many other values of θ are almost equally likely, as the function is quite flat around the maximum.

Multinomial data

Modeling count of DNA base pairs

There are four basic molecules of DNA:

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

The nucleotides are classified into two groups: purines (A and G) and pyrimidines (C and T). The binomial distribution won't work, as we have not two, but four different possible values. We use the multinomial distribution instead. Let's look at an example.

Nucleotide bias

Data from the one strand of DNA for the genes of the bacterium *Staphylococcus aureus* are available in a FASTA file, which we can load as follows.

```
library("Biostrings")
staph = readDNAStringSet(input_dir("data/staphsequence.ffn.txt"), "fasta")
```

Let's print the number of genes and the names of the first two:

```
length(staph)
```

```
## [1] 2650
```

```
names(staph)[1:2]
```

```
## [1] "lc1|NC_002952.2_cdsid_YP_039478.1 [gene=dnaA] [protein=chromosomal replication initiation protein] [protein_id=YP_039478.1] [location=517..1878]"
## [2] "lc1|NC_002952.2_cdsid_YP_039479.1 [gene=dnaN] [protein=DNA polymerase III subunit beta] [protein_id=YP_039479.1] [location=2156..3289]"
```

The first gene:

```
staph[1]
```

```
## A DNAStringSet instance of length 1
## width seq names
```

```
## [1] 1362 ATGTCGGAAAAAGAAATTTGG...AAGAAATAAGAAATGTATAA 1c1|NC_002952.2_c...
```

```
length(staph[[1]])
```

```
## [1] 1362
```

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

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

We hypothesize that selective pressure can modify the nucleotide frequencies in different genes differently. How do we check for that?

Does the nucleotide composition of the first ten genes come from the same multinomial distribution?

We do not have a prior reference, we just want to see whether the nucleotides occur in the same proportions, exemplarily in the first 10 genes. Deviations might deliver a biological insight: different selective pressure on these genes.

```
dimnames(tablf)[[2]] = names(staph) %>% sub("^.*gene=", "", ".") %>% sub("].*", "", ".")
tablfs = tablf[, 1:10]
tablfs
```

```
##      dnaA dnaN SAR0003 recF gyrB gyrA SAR0007 hutH serS SAR0010
## A      522  413      85  411  685  887      275  510  487      191
## C      219  176      31  168  293  395      137  244  180      111
## G      229  193      56  207  423  586      169  316  263      142
## T      392  352      74  327  531  793      250  445  357      252
```

Calculate sums of columns - the total number of bases per gene:

```
bpg = colSums(tablfs)
bpg
```

```
##      dnaA      dnaN SAR0003      recF      gyrB      gyrA SAR0007      hutH      serS
##      1362      1134      246      1113      1932      2661      831      1515      1287
## SAR0010
##         696
```

```
sweep(tablfs, 2, bpg, `/`) %>% round(2)
```

```
##      dnaA dnaN SAR0003 recF gyrB gyrA SAR0007 hutH serS SAR0010
## 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
```

We want to compare the nucleotide proportions in these genes.

As a first step, we compute the average proportion for each nucleotide across the genes.

```
p0 = rowSums(tablfs) / sum(tablfs)
p0
```



```
##      A      C      G      T
## 0.3495343 0.1529310 0.2022384 0.2952962
```

We suppose p_0 is the vector of multinomial probabilities for the four nucleotides, in all the genes. We use a Monte Carlo simulation to test whether the observed departures from the nucleotide frequencies in each gene are within a probable enough range.

We compute the expected counts by taking the outer product of the vector of probabilities p_0 by the total number of bases in each gene:

```
expectedtab = outer(p0, bpg)
round(expectedtab)
```

```
##      dnaA dnaN SAR0003 recF gyrB gyrA SAR0007 hutH serS SAR0010
## A    476  396      86  389  675  930      290  530  450      243
## C    208  173      38  170  295  407      127  232  197      106
## G    275  229      50  225  391  538      168  306  260      141
## T    402  335      73  329  571  786      245  447  380      206
```

A random table with the correct column sums can be created as follows. (This is done according to the null hypothesis that the true proportions are given by p_0 .)

```
randomtab = sapply(bpg, function(s) { rmultinom(1, s, p0) } )
randomtab
```

```
##      dnaA dnaN SAR0003 recF gyrB gyrA SAR0007 hutH serS SAR0010
## [1,]  483  385      92  404  693  934      277  548  454      225
## [2,]  204  168      40  165  275  417      126  213  207      113
## [3,]  300  229      40  210  360  520      190  333  251      145
## [4,]  375  352      74  334  604  790      238  421  375      213
```

How do we measure whether the two tables are “close”? Here we choose to use the following summary statistic:

$$\text{diffstat} = \sum_{ij} \frac{(\text{observed}_{ij} - \text{expected}_{ij})^2}{\text{expected}_{ij}}$$

or as an R function

```
computediffstat = function(o, e) {
  sum((o - e)^2/e)
}
```

```
}  
computediffstat(tablfs, expectedtab)
```

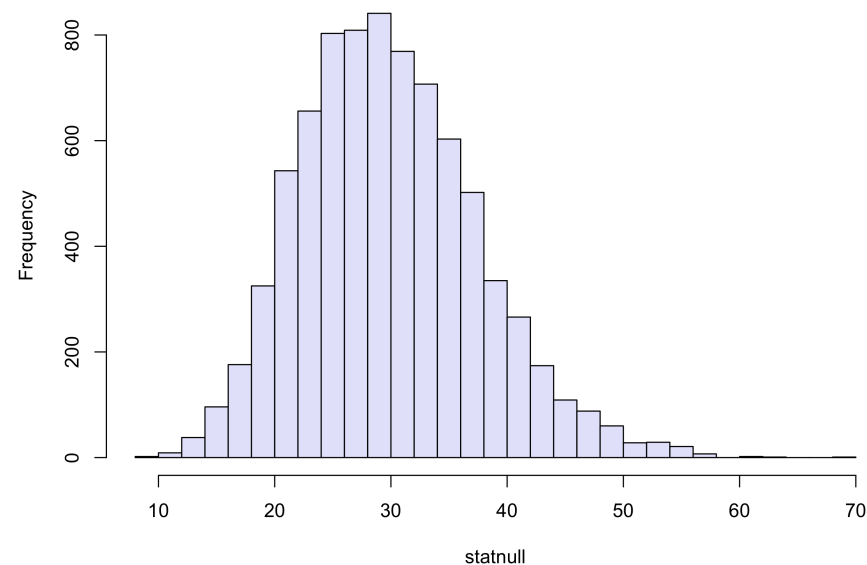
```
## [1] 68.65126
```

```
computediffstat(randomtab, expectedtab)
```

```
## [1] 29.96952
```

Now we do this not only once, but many times. All these statistics constitute the null distribution, as they were generated under the null hypothesis that p_0 is the vector of multinomial proportions.

```
statnull = replicate(8000, {  
  randomtab = sapply(bpg, function(s) { rmultinom(1, s, p0) } )  
  computediffstat(randomtab, expectedtab)  
})  
hist(statnull, col = "lavender", breaks = 40, main = "")
```



```
statdata = computediffstat(tablfs, expectedtab)  
mean(statnull > statdata)
```

```
## [1] 0.000125
```

We see that in fact the probability of seeing a value as large as 68.7 is very small. We can conclude that it is very rare that such a value would occur under the null model (which says that the data come from a common multinomial distribution with p_0 as its probability vector).

Assessing a distribution's fit using the QQ-plot

Statistical theory could have helped us avoid us running these simulations.

It tells us that the values of the `statnull` statistic are well approximated by a distribution called χ^2 (chi-squared) with parameter 27 (3 times 9).

Besides reducing your carbon footprint, using theory also is more accurate when you are estimating small probabilities (\leftarrow Lecture 1). Tail probabilities tend to be hard to compute by Monte Carlo.

We can check how well the theoretical and the simulated distribution match up using another visual goodness-of-fit tool: the QQ-plot.

(NB: A perhaps seemingly more intuitive approach to compare two distributions, whether from two different samples, or one from a sample and one from a theoretical model, would be looking at their histograms and/or densities. But this is clumsy and difficult to read.)

Quantiles

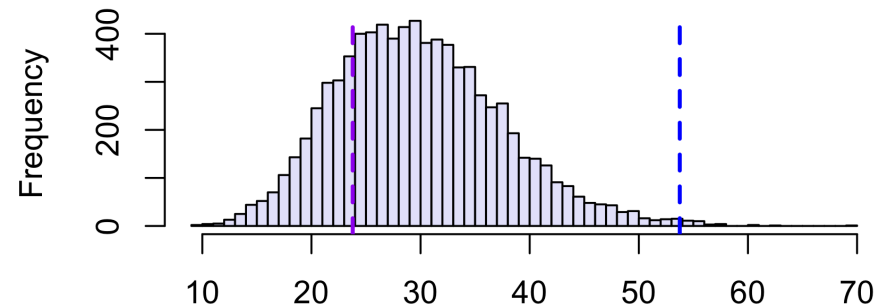
In the previous lecture on the epitope data, we ordered the 100 sample values $x_{(1)}, x_{(2)}, \dots, x_{(100)}$. Then, for instance, any value between the 22nd and the 23rd, will be acceptable as a 0.22-quantile $c_{0.22}$. We can also write this as $x_{(21)} \leq c_{0.22} < x_{(23)}$. $c_{0.22}$ is any number such that 22% of the values are smaller than it:

$$c_{0.22} \text{ is any number such that: } \frac{1}{n} \# \{i : x_i \leq c_{0.22}\} = 0.22$$

we also write

$$\hat{F}_n(c_{0.22}) = c_{0.22},$$

where \hat{F}_n is a function we call the empirical cumulative distribution function (ECDF).



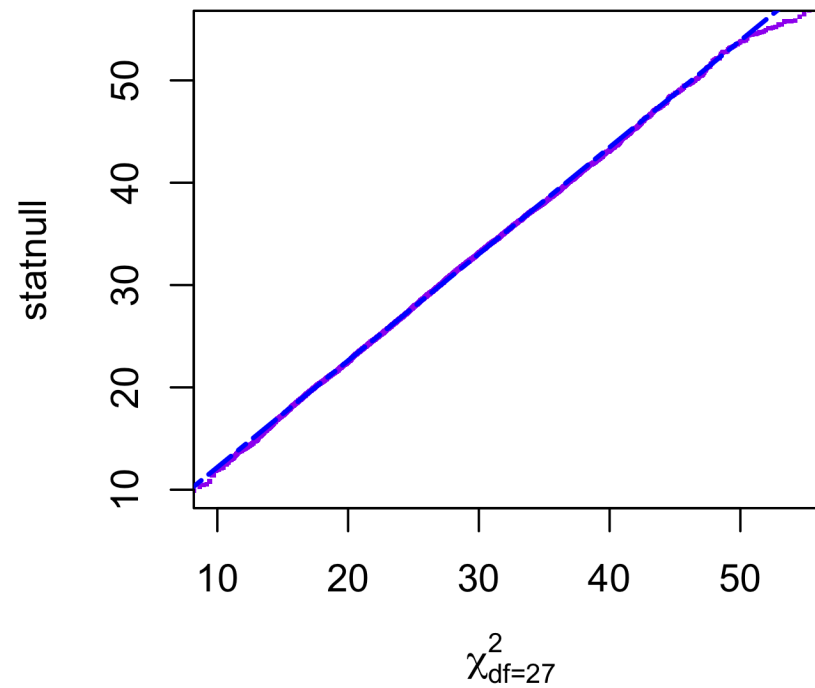
The histogram of the distribution of `statnull` and the quantiles $c_{0.22}$ and $c_{0.995}$

We can now compute the quantiles of the data vector and compare them to the those of the χ^2_{27} distribution.

```
pp = ppoints(length(statnull))
head(pp)
```

```
## [1] 0.0000625 0.0001875 0.0003125 0.0004375 0.0005625 0.0006875
```

```
qqplot(qchisq(pp, df = 27), statnull, pch=".", cex = 2,
       col="purple", xlab = expression(chi["df=27"]^2),
       xlim = c(10, 55), ylim=c(10, 55))
qqline(statnull, dist = function(p) qchisq(p, df = 27),
       col="blue", lty=6, lwd=2)
```



Now that we have established that `statnull` follows a χ^2_{27} distribution, we can use this fact to compute our p-value: the probability that we would observe a value as high as 68.7:

```
1 - pchisq(statdata, df = 27)
```

```
## [1] 1.74323e-05
```

Chargaff's rule

In fact, it was Chargaff who discovered the most important pattern in the nucleotide frequencies. Long before DNA sequencing was available, he used the weight of the molecules.

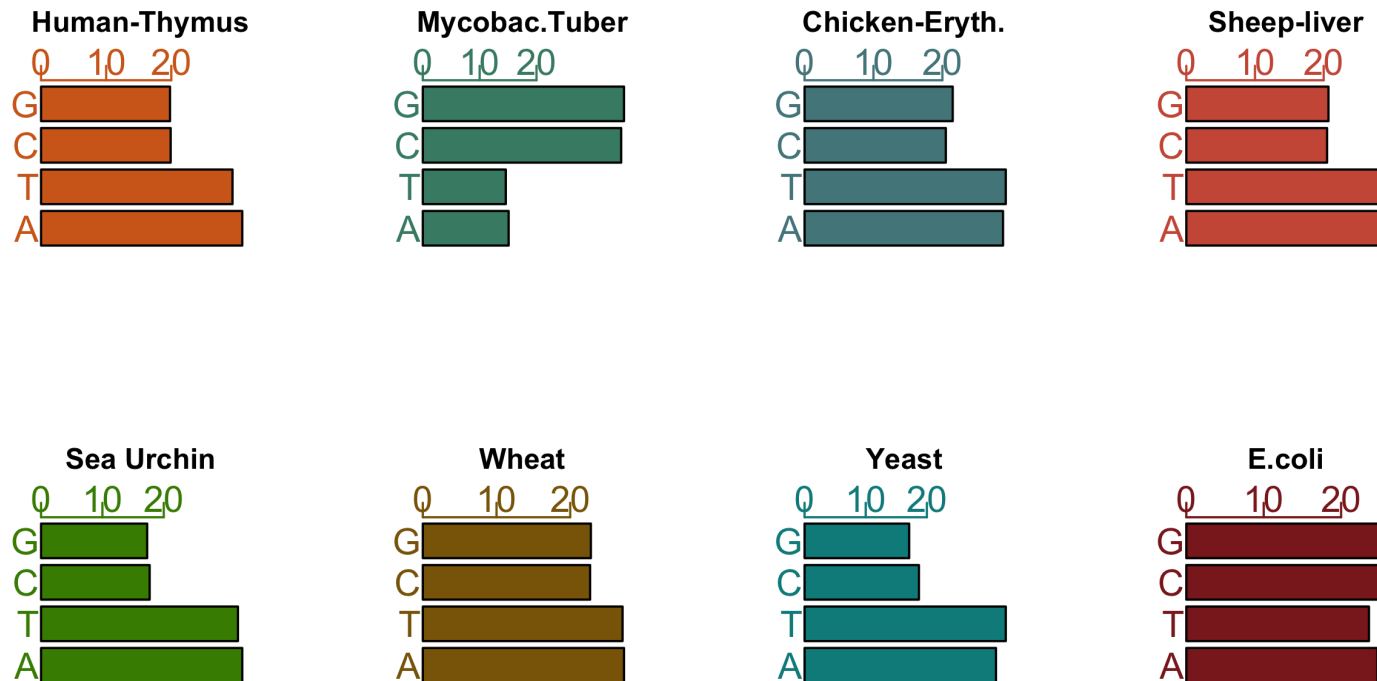
 Erwin Chargaff, 1947

Erwin Chargaff, 1947

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

```
load(input_dir("data/ChargaffTable.RData"))  
ChargaffTable
```

```
##  
## 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
```



Each barplot was made with a different row from the above table. There is certainly a tendency in the nucleotide frequencies.

Do these data seem to come from equally likely multinomial categories?

Chargaff saw the answer to this question and postulated a pattern called *base pairing*, which ensured a perfect match of the (molar) amount of adenine (A) in the DNA of an organism to the amount of thymine (T). This is now called Chargaff's rule. Similarly, whatever the amount of guanine (G), the amount of cytosine (C) is the same. The departure from the balanced multinomial distribution is not subtle.

Testing for significant base pairing

Suppose a first round of data revealed the pattern ($p_C = p_G$ and $p_A = p_T$), and now we want to test this hypothesis with the data above from a *second round* of experiments, summarized by the above table.

Define our test statistic to be

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

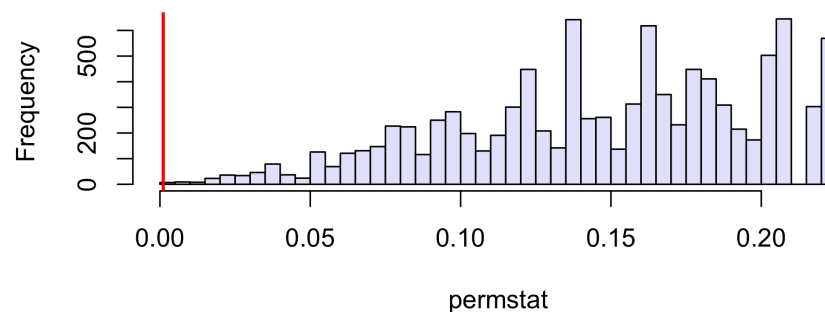
summed over all rows of the table. This value should be the smaller, the more similar the frequencies of paired nucleotides are.

```
nucleotide_frequencies = ChargaffTable / 100
statchargaff = function(x) {
  sum((x[, 2] - x[, 1])^2 + (x[, 4] - x[, 3])^2)
}
statchargaff(nucleotide_frequencies)
```

```
## [1] 0.001108
```

```
permstat = replicate(10000, {
  permuted = t(apply(nucleotide_frequencies, 1, sample))
  statchargaff(permuted)
})
fractionsmallerorequal = mean(permstat <= statchargaff(nucleotide_frequencies))
fractionsmallerorequal
```

```
## [1] 3e-04
```



Question 1: should we think of fractionsmallerorequal as a p-value?

Question 2: if we had come up with the hypothesis that $p_C = p_G$ and $p_A = p_T$ from the same data that we then use to compute `fractionsmallerorequal`, would there be a problem?

Codon usage patterns

A sequence of three nucleotides (a codon) in a coding region of a gene can be transcribed into one of 20 possible amino acids. There are $4^3 = 64$ possible codon sequences, but only 20 amino acids.

The multiplicity (the number of codons that code for the same amino acid) varies from 2 to 6. The different codon-spellings of each amino acid do not occur with equal probabilities. Let's look at the data for the standard laboratory strain of tuberculosis (H37Rv):

```
library("readr")
Myc = read_table(input_dir("data/M_tuberculosis.txt"), col_names = TRUE, comment = "#")
Myc
```

```
## # A tibble: 79 x 4
##   AmAcid Codon Number PerThous
##   <chr>  <chr>   <dbl>   <dbl>
## 1 Gly   GGG     25874    19.2
## 2 Gly   GGA     13306     9.9
## 3 Gly   GGT     25320    18.8
## 4 Gly   GGC     68310    50.8
## 5 ""    <NA>      NA       NA
## 6 Glu   GAG     41103    30.6
## 7 Glu   GAA     21767    16.2
## 8 Asp   GAT     21165    15.8
## 9 Asp   GAC     56687    42.2
## 10 ""   <NA>      NA       NA
## # ... with 69 more rows
```

The codons for the amino acid proline are of the form CC*, which occur with the following frequencies.

```
library("dplyr")
MycPro = filter(Myc, AmAcid == "Pro")
MycPro$Codon
```

```
## [1] "CCG" "CCA" "CCT" "CCC"
```

```
MycPro$Number / sum(MycPro$Number)
```

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

The counts enable us to estimate the theoretical probabilities (or parameters) by taking the maximum likelihood estimates as the proportions with which each occurs.

Exercise: use a χ^2 -distribution with 3 degrees of freedom as the reference distribution to which you compare the χ^2 -statistic computed from the actual numbers of occurrence of the four categories (proline codons) and their expected values.

Working with 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 and Poisson for Yes/No binary boxes and the Multinomial for categorical variables such as A/C/G/T or genotypes such as 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. This concept, called **contingency table**, is useful for many biological data types.

HairEyeColor

```
## , , Sex = 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
##
## , , Sex = 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
```

HairEyeColor is a 3 dimensional array with three dimensions:

- 1 Hair: Black, Brown, Red, Blond
- 2 Eye: Brown, Blue, Hazel, Green
- 3 Sex: Male, Female

dim(HairEyeColor)

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

? 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(input_dir("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 of two independent binomials: one for sex and one for color blindness, with parameters from the respective marginal distributions. Under this model we know all the table cells' probabilities, and we can compare the observed counts to the expected ones. In R, this is done through the `chisq.test` function.

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

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

Concatenating several multinomials: motifs

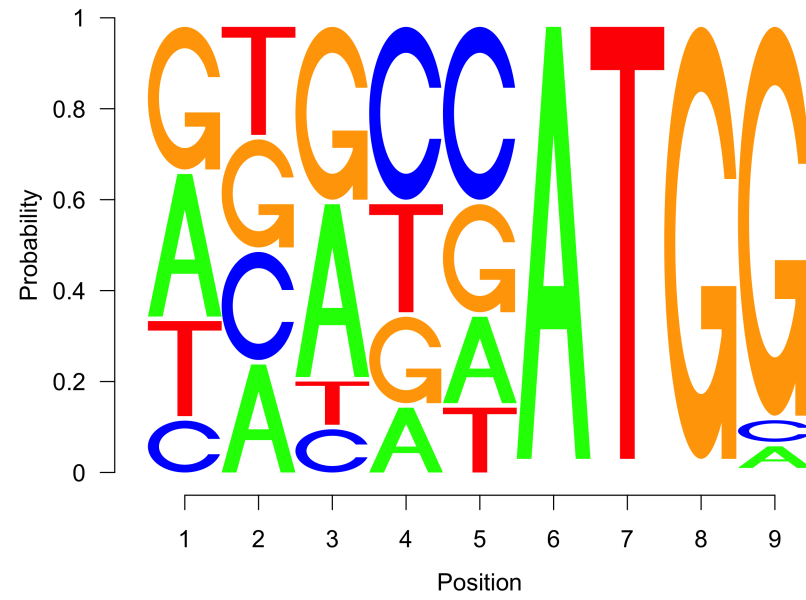
The **Kozak Motif** is a sequence that occurs close to the start codon **ATG** of a coding region. The start codon itself always has a fixed spelling, but in positions 5 to the left of it, there is also a nucleotide pattern that is not fixed, but in which the letters are quite far from being equally likely.

We summarize this by giving the Position-Weight-Matrix(PWM), which provides the multinomial probabilities at every position. This is encoded graphically by what is called a **logo**.

```
library("seqLogo")
load(input_dir("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)
```



Recap

What have we noticed in our study of the frequencies in these 'box' models for categorical variables?

- different 'boxes' in the multinomials we have encountered are rarely of equal size, i.e., equality of the different p_i 's is the exception rather than the rule.
- departures from independence (i.e., correlations) of two categorical variables (sex and colorblindness, hair and eye color, ..) are common.

We will see in a later lecture that we can explore the directions, patterns in and possible explanations for these dependencies by using multivariate decompositions of the contingency tables.

An important special case of dependent categorical variables are sequences (in time and/or space).

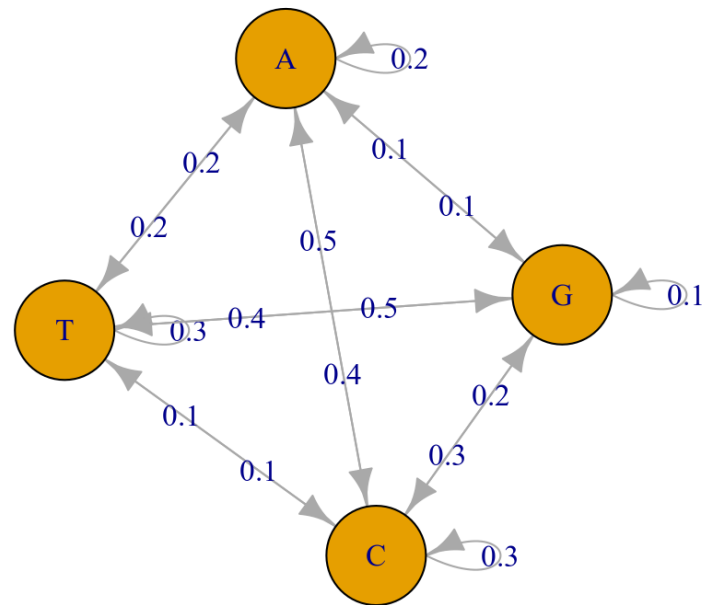
Markov chains

Dependence in ordered observations occurs, for instance, in temporal weather patterns and nucleotides along a sequence: the distribution of the next value depends on the ones before.

Sometimes it is useful to make a simplifying assumption: the distribution of the next value depends only on the one before. If this holds, we call it the *Markov property*. (A different way of stating the assumption is that the current values, or *state*, contains all information that there needed to predict all the future behaviour of the system – cf. Newtonian, Hamiltonian mechanics.)

For DNA sequences, we may see specific succession patterns so that the pairs of nucleotides, called digrams (e.g., 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:

$$P(CA) \neq P(C)P(A).$$



Transition matrix

$$\begin{pmatrix} W_{11} & W_{12} & \dots & W_{1n} \\ W_{21} & W_{22} & \dots & W_{2n} \\ \vdots & \vdots & & \vdots \\ W_{n1} & W_{n2} & \dots & W_{nn} \end{pmatrix}$$

If at time (or sequence position, ...) t the value (or state) is i ($i \in \{1, \dots, n\}$), the probability that at time $t + 1$ the value (state) is j is W_{ij} .


The row sums and the column sums of the matrix W are all one ($\sum_i W_{ij} = \sum_j W_{ij} = 1$), and all $W_{ij} \geq 0$. Such a matrix is also called a stochastic matrix.

How to estimate a transition matrix from data?

Bayesian Thinking

Up to now we have followed a classical approach where probabilities are defined as long term frequencies from observations. If we visit a new place and have had four days of sunshine, we might estimate that the probability of sunshine here is 100% and that of rain 0%. However, this does not take into account any other information that we might already know. For that we need a different approach, which is able to combine prior knowledge as well as the data at hand: this is provided by the Bayesian paradigm.

Bayesian thinking also operates in terms of parameters (i.e. 'knowledge' = parameter values). However, at any point in our chain of reasoning (even prior to seeing the current data), there is a probability distribution on our parameters. Once the current data is seen, we simply update the distribution, which is then called the posterior.

Turtles all the way down

Turtles all the way down

Example: haplotype frequencies

Here's a forensics example using signatures from the Y chromosome called haplotypes. Our roadmap is:

- We'll first look at the motivation and terminology behind the computations involved in haplotype frequency analyses.
- Then we'll revisit the concept of likelihood.
- We'll see how we can think of unknown parameters as being random numbers, modeling our uncertainty about them with a prior distribution.
- Finally, we'll see how to incorporate newly observed data into the distributions and compute a posterior distribution and confidence statements about the parameters.

In this example we want to estimate the proportion of a particular Y-haplotype assembled from the different short tandem repeats (STR).



STR

The combination of values at the special STR locations used for forensics are labeled by the number of repeats at the specific positions. Here is part of an STR haplotype table:

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, We want 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 lets 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 that takes values in the same range of values as our parameter (we also call that the distribution's **support**).

For 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 is:

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

We can see that 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). We also see that not only gene naming can be confusing: here both the name of the distribution family and one of its two parameters is called beta.

Now here comes a nice fact: if we start with a prior belief that our distribution of θ is beta-shaped, observe a dataset of n binomial trials, then update our belief, the posterior will also have a beta distribution, albeit (usually) with new parameters.

This is a provable mathematical property, which is expressed by saying that the binomial and the beta are *conjugate*. We will not prove it, however we illustrate it with a simulation.

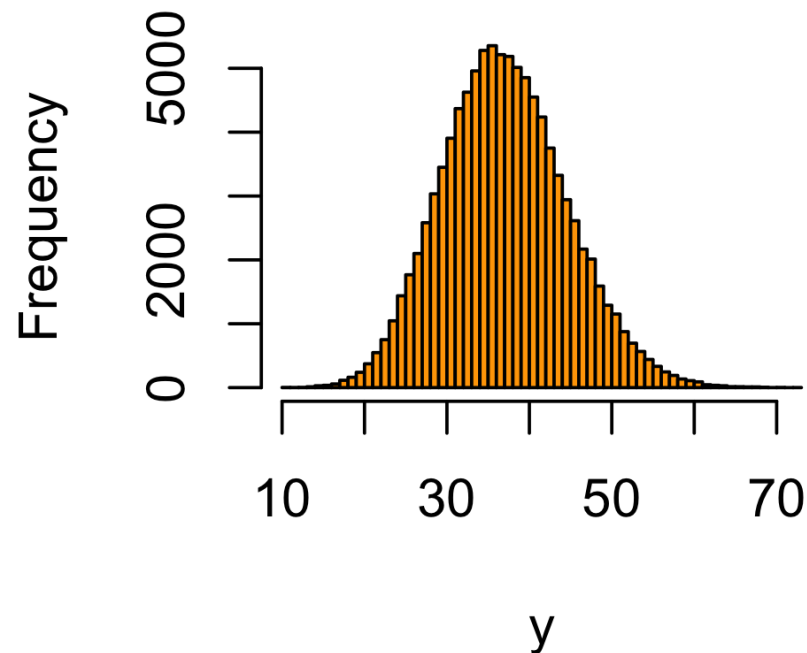
Priors and the marginal distribution

Consider a random variable Y that is binomial distributed with parameter θ . We have seen how the density of Y (or sampled histograms) look when θ takes a given, fixed value. But what happens if θ itself also varies according to some distribution?

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

Let's simulate that. We first generate a random sample of 10000 θ s, and for each of these we then generate a random sample of Y .

```
rtheta = rbeta(100000, 50, 350)
y = sapply(rtheta, function(th) {
  rbinom(1, prob = th, size = 300)
})
hist(y, breaks = 50, col = "orange", main="")
```



Histogram of Y

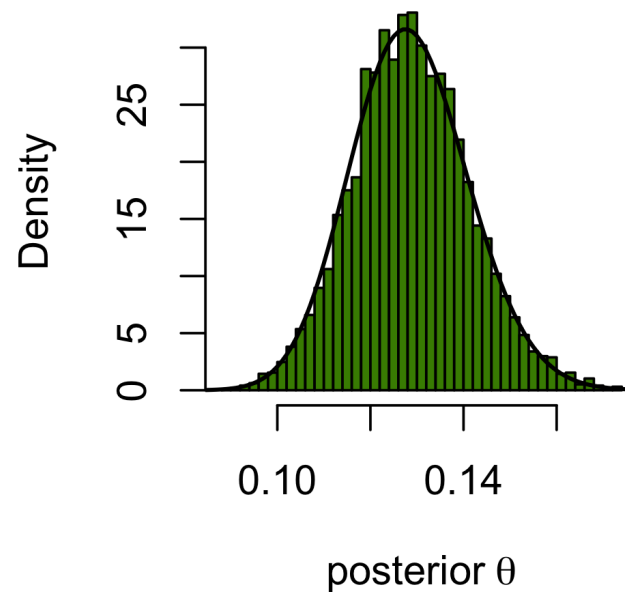
Histogram of all the thetas such that $Y = 40$: the posterior distribution

So let's suppose we observed an experimental data point where Y was 40. How does this change our belief about the distribution of θ ? We compute the posterior distribution of θ by conditioning on those outcomes where Y was 40 (`thetaPostEmp`), and also the one given by theory (`densPostTheory`).

```
thetaPostEmp = rtheta[y == 40]
hist(thetaPostEmp, breaks = 40, col = "chartreuse4",
      probability = TRUE, main = "", xlab = expression("posterior"-theta))
```



```
densPostTheory = dbeta(thetas, 90, 610) # more on this below  
lines(thetas, densPostTheory, type = "l", lwd = (1+sqrt(5))/2)
```



We can check the means of both versions of the posterior distribution and see that they are close to 4 significant digits.

```
mean(thetaPostEmp)
```

```
## [1] 0.1288427
```

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

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

To get the mean of the theoretical density, above we computed the integral $\int_0^1 \theta f(\theta) d\theta$ using numerical integration, i.e., by summing over the integrant. This is not always convenient (or feasible), in particular when our parameters are high-dimensional (e.g., if our model involved not only a single θ parameter, but several thousand as for instance in the case of image analysis). In such cases, and when an analytic solution is also not available, we can resort to Monte Carlo integration (which here is very simple):

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

```
## [1] 0.1285694
```

The posterior distribution is also a beta

Now we have seen that the posterior distribution is also a beta. In our case its parameters 90 and 610 were obtained by adding to the prior parameters 50, 350 the number of observed successes, 40, and observed failures, 260. Thus the general rule is

$$\text{beta}(\alpha_{\text{post}}, \beta_{\text{post}}) = \text{beta}(\alpha_{\text{post}} + y, \beta_{\text{prior}} + (n - y))$$

Now suppose we have a second set of observations

After seeing our first set of data, we 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.

What is now our new best belief about 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 point estimate of θ would be 0.135.

The theoretical (MAP) estimate would be the mode of $\text{beta}(115, 735)$, i.e., $\frac{114}{848} \simeq 0.134$. 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] 0.1353139
```

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

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



The last line of this code uses a Monte Carlo method for finding the MAP from a sample from `rbeta(. , 115, 735)`.

Exercise

Redo these computations replacing our original prior with a softer prior (broader, less peaked), synonymous with less prior information. How much does this change the final result?

As a general rule, the prior rarely changes the posterior distribution substantially except if it is very peaked (i.e., if, at the outset, we were already rather sure of it) or 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 has negligible impact on the final result.

Confidence statements

Sometimes we don't just want a point estimate, but a **credible interval** (a Bayesian equivalent of the **confidence interval**). E.g., we can take the 2.5 and 97.5 percentiles of the posterior distribution:

$$P(L \leq \theta \leq U) = 0.95$$

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

```
##      2.5%      97.5%  
## 0.1131540 0.1590786
```

A more complicated example from RNA-seq analysis

 From Love, Huber and Anders in their article about the `DESeq2` package.

From Love, Huber and Anders in their article about the DESeq2 package.

Shown are the prior (solid black line), the likelihoods (solid lines, scaled to integrate to 1) and the posteriors (dashed lines) for two genes (green and purple). Due to the higher dispersion of the data for the purple gene, its likelihood is wider and less peaked (indicating less information), so 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).

Worked examples using biological sequence data

In Section 2.10, Questions 2.21 - 2.23 of the book, we explore the occurrence of the **Shine-Dalgarno motif** AGGAGGT in the genome of E.coli. This motif helps initiate protein synthesis in bacteria. We look at the frequency at which it occurs per length of genomic sequence, and at distances between its occurrences.

In Section 2.10.1, Questions 2.24 - 2.26, we look at differences between dinucleotide frequencies in regions of the human genome called **CpG islands** and the rest.

Summary of this lecture

This lecture explained how to go from a set of experimental data back to a distribution model and its parameters.

We saw statistical models for experiments with two or more possible (categorical) outcomes: binomial and multinomial.

We saw maximum likelihood and Bayesian estimation procedures.

We saw examples on codon usage and nucleotide pattern discovery.

Further reading: Chapter 2 in the book

Cleveland, William S. 1988. *The Collected Works of John W. Tukey: Graphics 1965-1985*. Vol. 5. CRC Press.