Lecture 6: Mixture Models

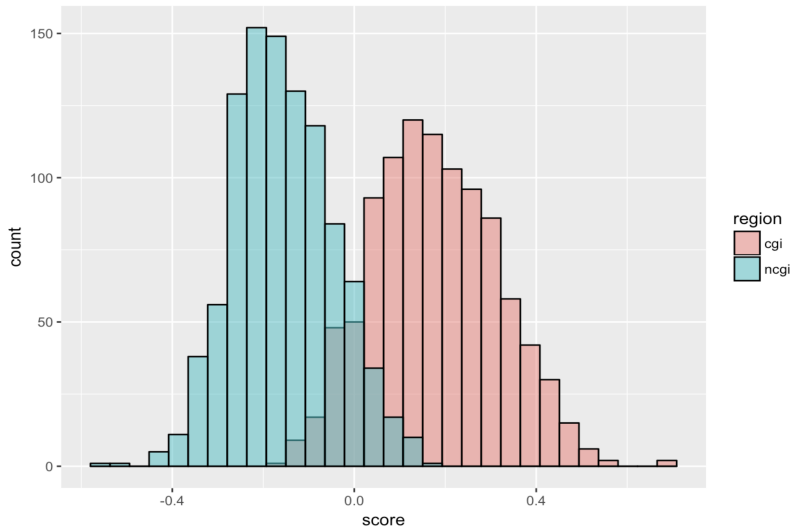
Jing Ma, Statistics, TAMU

25 September, 2019

This lecture

- How to generate finite mixtures.
- Learn the Expectation-Maximization (EM) algorithm.
- Examples of infinite mixtures.
- Data transformations.

Why mixture models?



- Heterogeneity due to hidden variables: some that we want to detect, some that are uninteresting.
- The statistical sample does not come from only one population.
- Types of Mixture Models
  - finite
  - infinite

Simple examples and computer experiments

Suppose we want two equally likely components.

We decompose the generating process into steps:

Flip a fair coin.

- If it comes up heads
  - Generate a random number from a Normal with mean 1 and variance 0.25.
- If it comes up tails
  - Generate a random number from a Normal with mean 3 and variance 0.25.

Fair mixture

This is what the resulting data would look like if we did this 10,000 times.

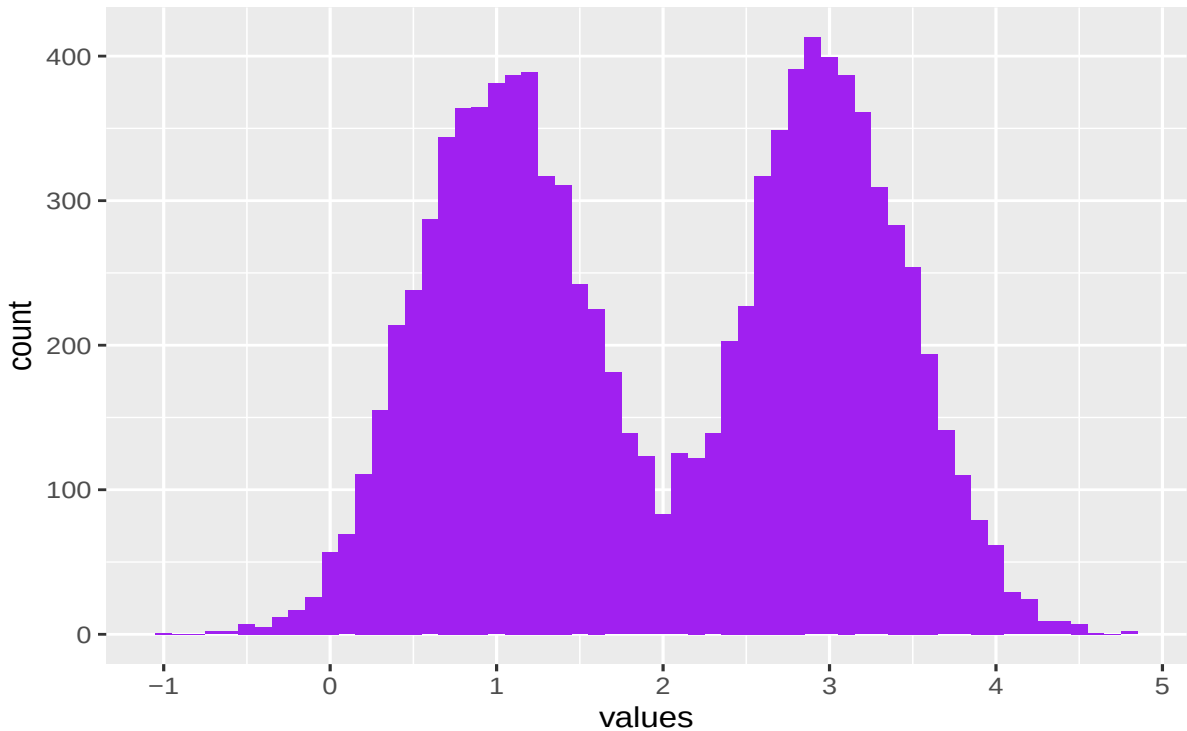


Fig. 1: Histogram of two normals from 10,000 random draws.

Histogram

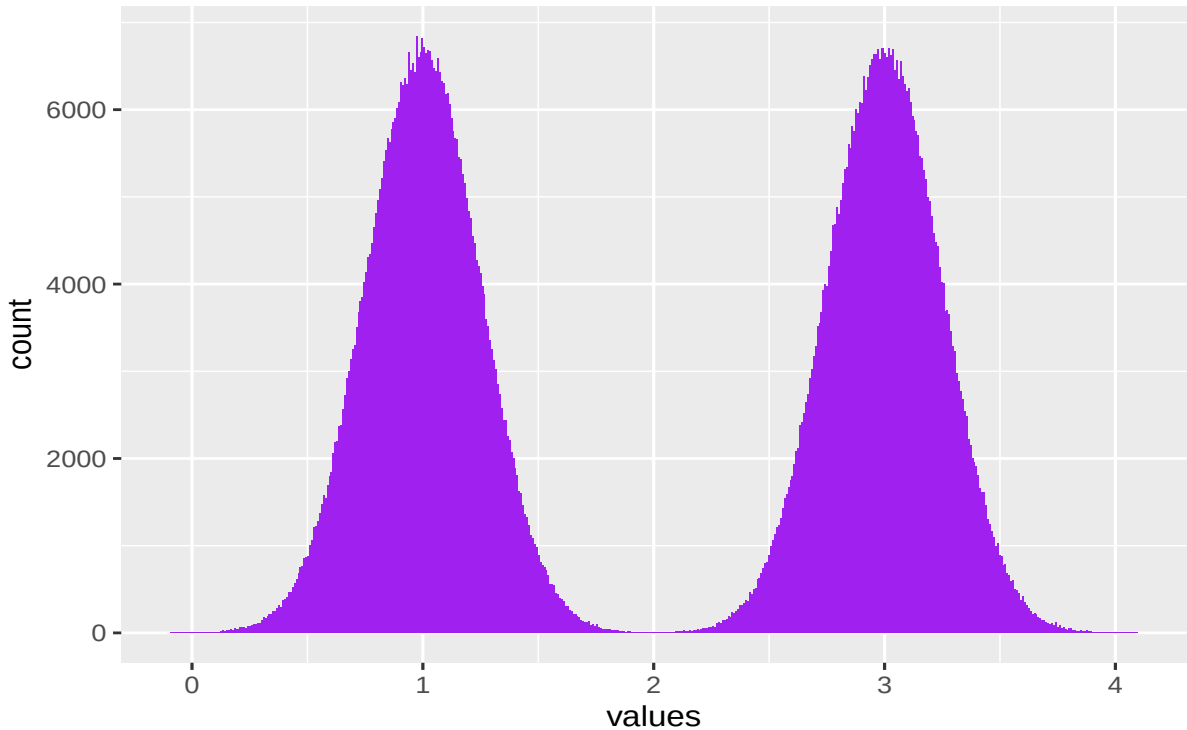


Fig. 2: Histogram of two normals from 1 million random draws.

Question: Comparing Fig. 1 and 2, what do you notice?

Theoretical density

In fact we can write the density (the limiting curve that the histograms tend to look like) as

$$f(x) = \lambda\phi_1(x) + (1 - \lambda)\phi_2(x),$$

where

- $\lambda = 0.5$ is the mixing proportion,
- ϕ_1 is the density of the Normal$(\mu_1 = 1, \sigma^2 = 0.25)$, and
- ϕ_2 is the density of the Normal$(\mu_2 = 3, \sigma^2 = 0.25)$.

Theoretical density

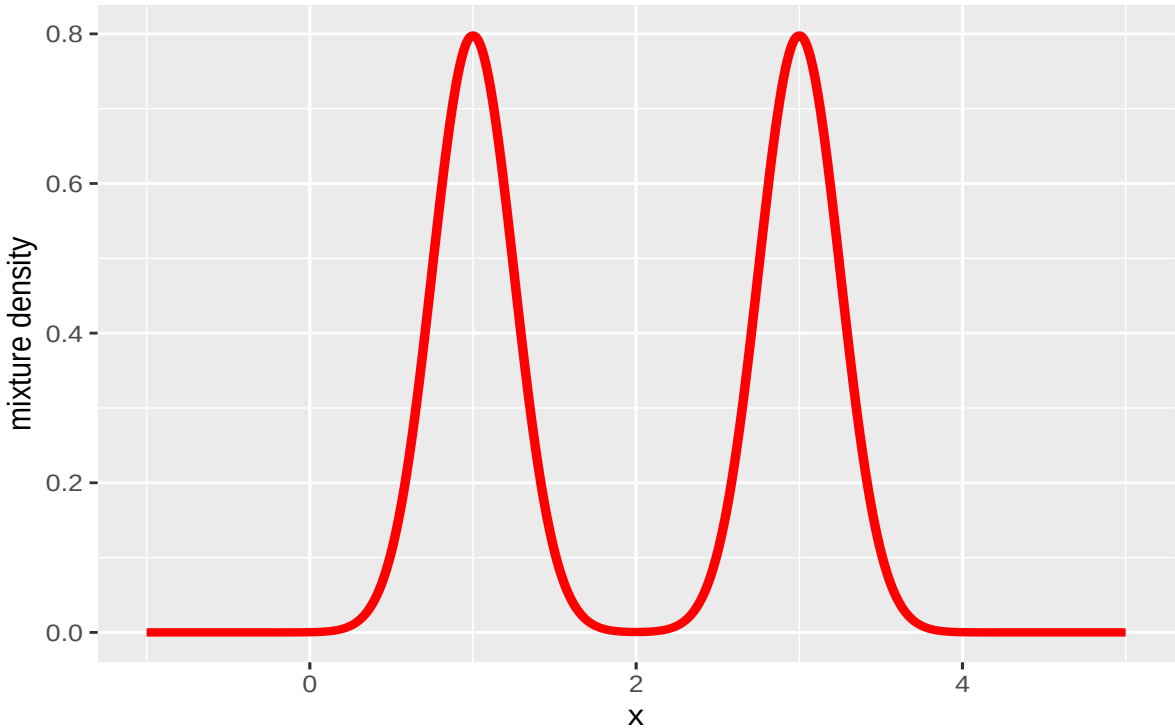


Fig. 3: Theoretical density of the mixture.

Less definite mixtures

- The mixture model in the previous example of course was extremely visible as the two distributions do not overlap
- This can happen if we have two very separate populations, for instance different species of fish whose weights are very different.
- However in many cases the separation is not so clear.

Less definite mixtures

Here is a histogram generated by two Normals with the same variances. Can you guess the two parameters for these two Normals?

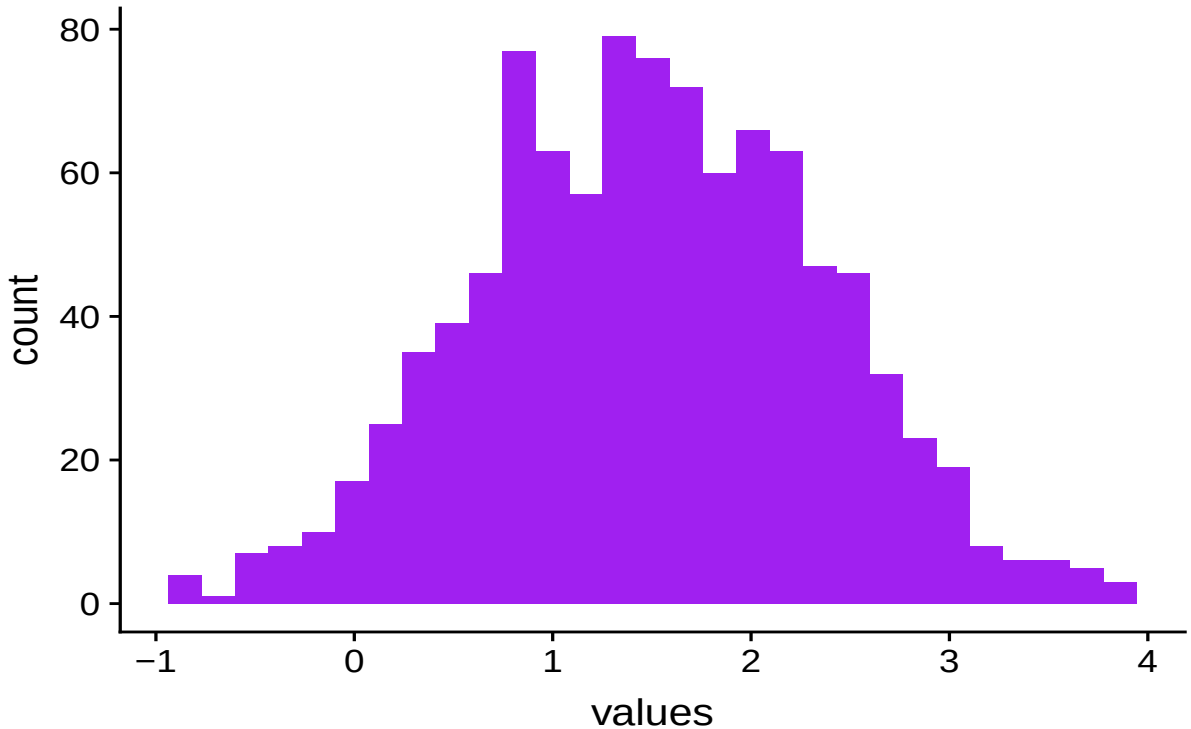


Fig. 4: A mixture of two normals that is harder to recognize.

Less definite mixtures

Actually we know the labels.

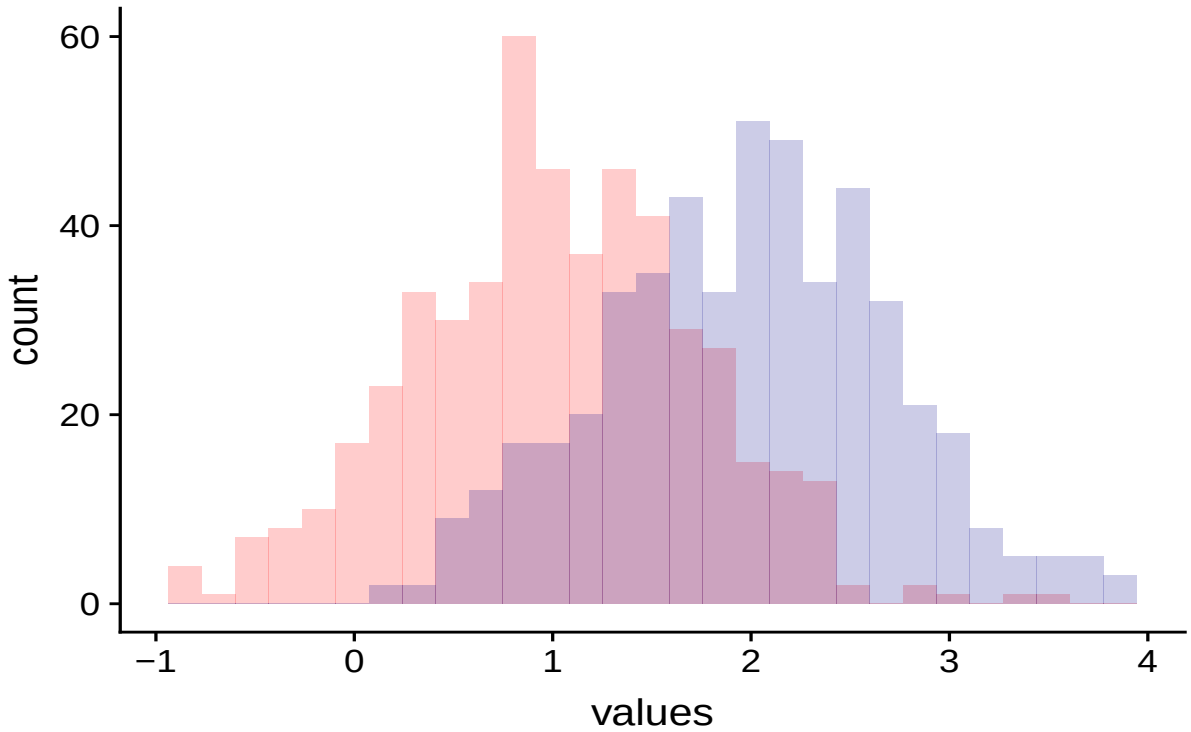


Fig. 5: The mixture with the two components colored in red and blue.

Less definite mixtures

- Some values are shared between the two components, but were counted only once in Fig. 5.
- For each value, if we count its total number of appearance from both components, we recover the mixture (in purple).

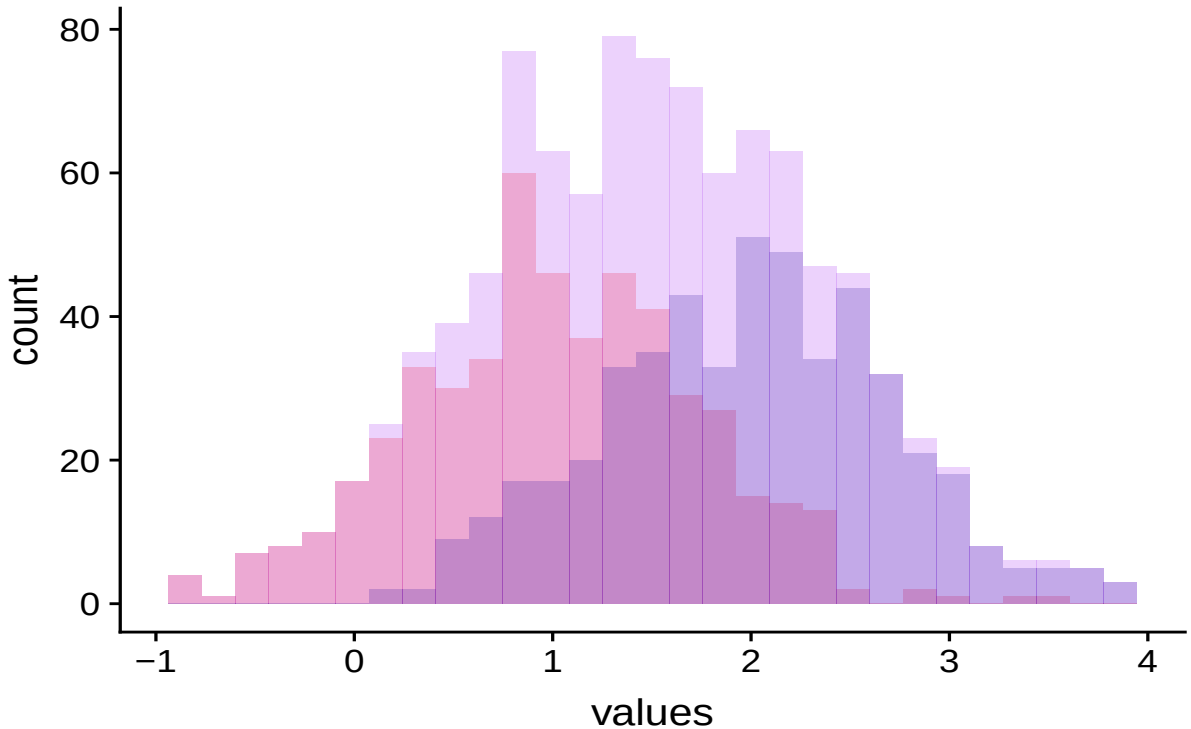


Fig. 6: As in Fig. 5, with stacked bars for the two mixture components.

Latent variable

- In the previous example we knew who had been generated from which component of the mixture.
- Often this information is missing. We call the hidden variable *a latent variable*.
- This is an important example of heterogeneity due to a hidden variable (unmeasured or **latent**).
- We use the EM algorithm to infer the value of the hidden groupings.

Example: mixture of normals

Suppose we have a fair mixture of two normals with parameters

$$\theta = (\mu_1 = ?, \mu_2 = ?, \sigma_1 = 1, \sigma_2 = 1, \lambda = 0.5),$$

where μ_1 and μ_2 are unknown. Let u denote the hidden group labels.

Question: Given observed y, how can we recover the unknown μ_1, μ_2 and class labels u?

Example: mixture of normals

We want to use the maximum likelihood approach.

- The probability $p(y, u \mid \theta)$ serve as a weight or "participation" of observation y in the likelihood function.
- The **marginal likelihood** for the observed y is the sum over all possible values of u of the densities at (y, u):

$$\text{marglike}(\theta; y) = p(y \mid \theta) = \sum_{u=1}^{2} p(y, u \mid \theta).$$

- However, MLE with the **marginal likelihood** is difficult.
- The EM algorithm provides a solution.

The EM algorithm

The **E**xpectation-**M**aximization algorithm alternates between

- pretending we know the labels and estimating the distribution parameters of the components, and
- pretending we know the distribution parameters of the components and estimating the hidden labels (*soft labels*).

Parameter estimation with known labels

- If we know the labels u, we can estimate the means using separate maximum likelihood for each group.
- The overall MLE is obtained by maximizing the **likelihood for the complete data**

$$f(y, u \mid \theta) = \prod_{\{i:\, u_i=1\}} \phi_1(y_i) \prod_{\{i:\, u_i=2\}} \phi_2(y_i), \qquad (1)$$

  or its logarithm.
- The maximization can be split into two independent pieces and solved as if we had two different MLEs to find.

Class assignment with known parameters

- If we know the parameter values, we can replace the "hard" labels u for each observation (it is either in group 1 or 2) by membership probabilities that sum up to 1: e.g.

$$p(u = 1 \mid y, \theta) = 1 - p(u = 2 \mid y, \theta).$$

- This is sometimes called **soft** averaging, and highlights the fact that we create weighted averages where each point's probability serves as a weight, so we don't have to make a hard decision that a point belongs to this or that group.

E-step and M-step

- Suppose the current best guesses for the unknown parameters are $\theta^* = (\mu_1^*, \mu_2^*, \lambda^*)$.
- We use these to compute the so-called **E**xpectation function

$$\begin{aligned} E(\theta \mid \theta^*) &= E_{u|Y,\theta^*}[\log p(y, u \mid \theta^*)] \\ &= \sum_{u=1}^{2} p(u \mid y, \theta^*) \log p(y, u \mid \theta^*). \end{aligned}$$

- The value of θ that maximizes $E(\theta \mid \theta^*)$ is found in what is known as the **M**aximization step:

$$\theta^{\text{new}} = \text{argmax}_\theta E(\theta \mid \theta^*).$$

- These two iterations (**E** and **M**) are repeated until the improvements are small.

The EM algorithm is iterative

- In reality we do not know the u labels, nor do we know the mixture proportions (i.e., λ).
- We have to start with an initial guess for the labels, estimate the parameters and go through several iterations of the algorithm, updating at each step our current best guess of the group labels and the parameters until we see no substantial improvement in our optimizations.

Things to learn from the EM algorithm

1. It shows us how we can tackle a difficult problem with too many unknowns by alternating between solving simpler problems.
2. *Soft* averaging allows an observation to belong to several groups by using probabilities of membership as weights.
3. Good practice to repeat such a procedure from different starting points and check that we always get the same answer.

Animation

Wiki

More than two components

Weighing N=7000 nucleotides obtained from mixtures of deoxyribonucleotide monophosphates (each type has a different weight, measured with the same standard deviation sd=3), could give something like:

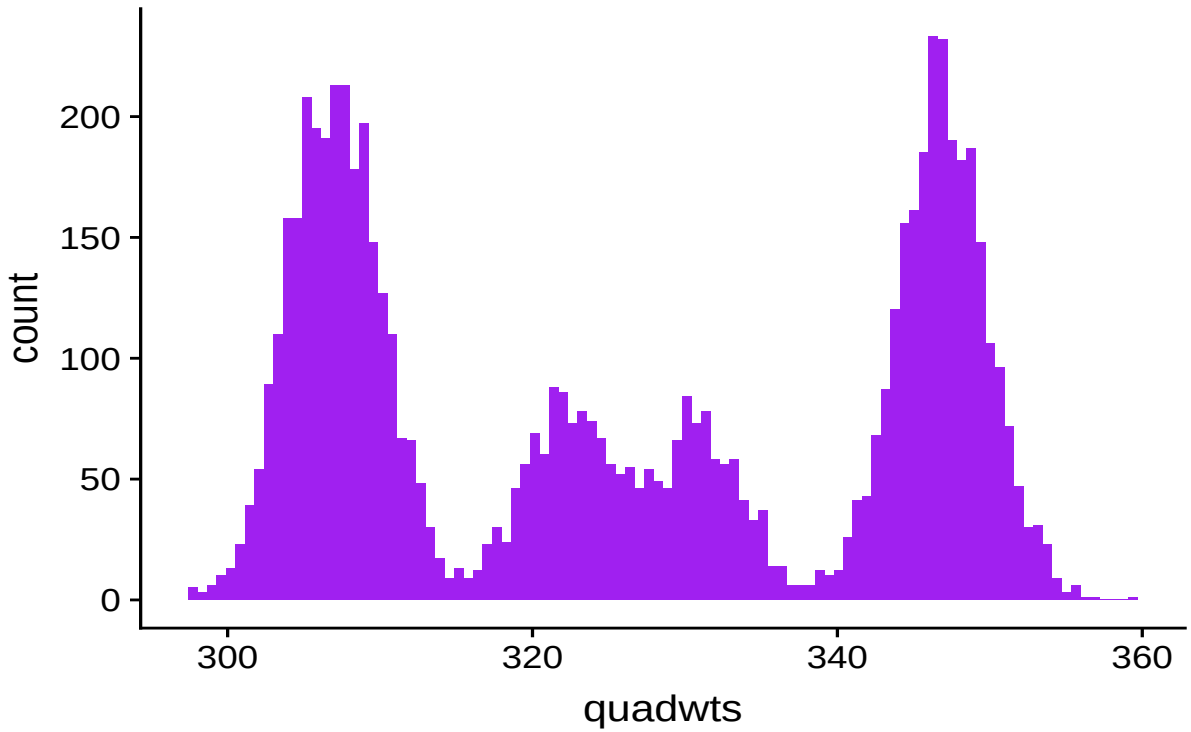


Fig. 7: Simulation of 7,000 nucleotide mass measurements.

Infinite mixtures

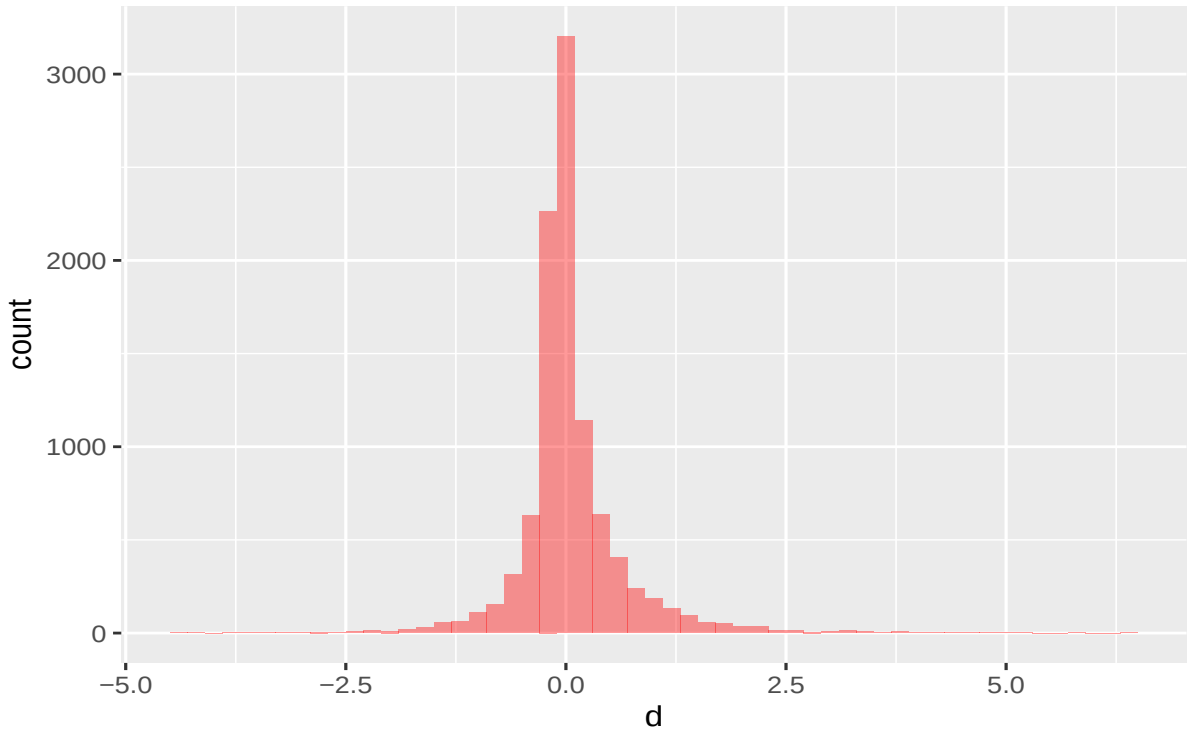
- Mixtures can be useful without having to find who came from which distribution.
- Suppose that instead of flipping a coin, we picked a ball from an urn with the mean and variances written on it. If half the balls were marked $(\mu_1 = 1, \sigma^2 = 0.25)$ and the other half $(\mu_1 = 3, \sigma^2 = 0.25)$, this would actually be equivalent to our original mixture.
- This gives us much more freedom: all the balls could have different parameter-numbers on them and these numbers could actually be drawn at random from a special distribution.
- This is often called **a hierarchical model**.

Infinite mixtures

Hierarchical models can be generated in a two-step process:

1. Generate the numbers on the *parameter balls*.
2. Draw a ball, then draw a random number according to that parametric distribution.

Infinite mixture of Normals



The Gamma-Poisson mixture model

Count data are often messier than simple Poisson and Binomial distributions.

The Gamma-Poisson mixture model is the simplest mixture model for counts and used for

- Overdispersion (in Ecology).
- Different evolutionary mutation rates.
- Abundance data of 16sRNA measured taxa.

The Gamma distribution

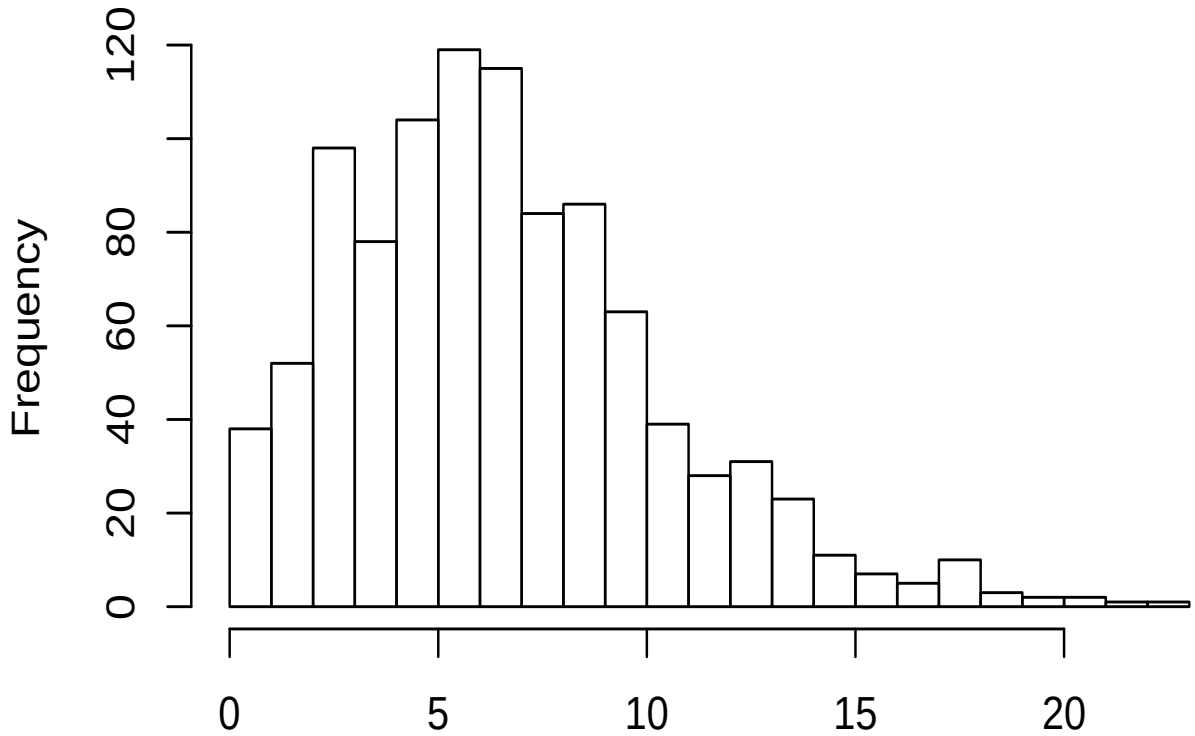
- The Gamma distribution (wiki) is used to model continuous variables with any range of positive values.
- Typical quantities that follow this distribution are waiting times and survival times.

Gamma mixture of Poissons: a hierarchical model

We use a two step process to generate N samples from the Gamma-Poisson mixture:

1. Generate a whole set of Poisson parameters: $\lambda_1, \lambda_2, ...\lambda_N$ from a Gamma distribution.
2. Generate a set of Poisson(λ_i) random variables.

Example of Gamma Poisson mixture



The Negative Binomial distribution

The Gamma-Poisson distribution is also called the **negative binomial**. A negative binomial experiment is a statistical experiment that has the following properties:

- The experiment consists of x repeated trials.
- Each trial can result in just two possible outcomes. We call one of these outcomes a success and the other, a failure.
- The probability of success, denoted by p, is the same on every trial.
- The trials are independent.
- The experiment continues until r successes are observed, where r is specified in advance.

Test your understanding

Consider the following statistical experiment. You flip a coin repeatedly and count the number of times the coin lands on heads. You continue flipping the coin until it has landed 5 times on heads.

This is a negative binomial experiment because:

- The experiment consists of repeated trials. We flip a coin repeatedly until it has landed 5 times on heads.
- Each trial can result in just two possible outcomes - heads or tails.
- The probability of success is constant - 0.5 on every trial.
- The trials are independent.
- The experiment continues until a fixed number of successes have occurred; in this case, 5 heads.

Applications of Gamma-Poisson

- For each taxon i, let u_i be its relative abundance.
- The number of reads for the sample j and taxon i would be

$$K_{ij} \sim \text{ Poisson } (s_j u_i).$$

- Assume that the proportion of taxon i

$$U_i \sim \text{Gamma}(r_i, \frac{p_i}{1-p_i}).$$

- Thus the read counts K_{ij} have a Gamma-Poisson mixture of different Poisson variables.

Variance stabilization

Data transformations affect variances.

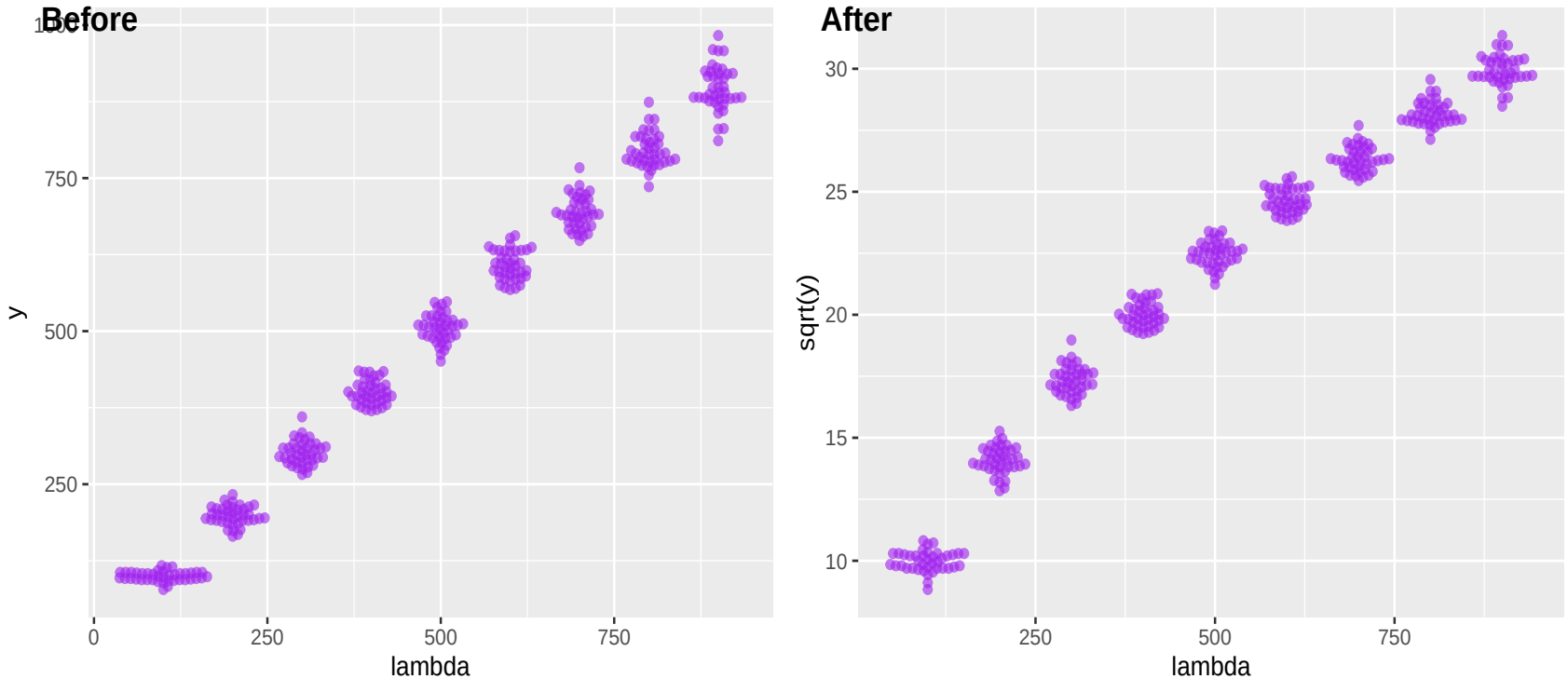
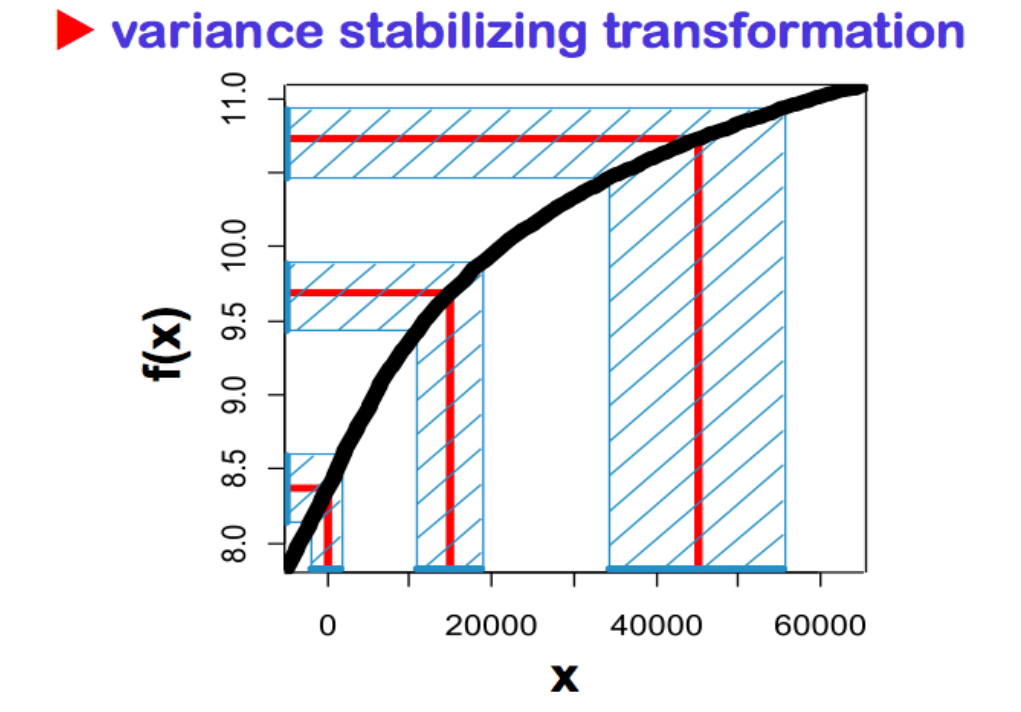


Fig. 8: Poisson distributed measurement data with varying means. Note how the shapes of the beeswarm clouds are different between the two panels.

Variance stabilization

The logarithm transformation



Variance stabilization examples

Suppose X is a random variable with mean μ and variance $v = v(\mu)$. Denote $g(X)$ a transformation function.

- if $v(\mu) = \mu$ (Poisson), we use $g(x) = \sqrt{x}$, the square root transformation.
- If $v(\mu) = c\,\mu^2$, we use $g(x) = \log(x)$, the logarithm transformation.

The logarithm transformation is so popular because

- it acts as a variance stabilizing transformation whenever the data have a constant coefficient of variation, (when the standard deviation is proportional to the mean).

Applications

Variance stabilization can be useful for statistical testing when both the means *and* the variances are different.

Mixture models can often instruct us to use appropriate data transformations.

Examples of variance stabilization:

- the *generalized logarithmic* transformation applied in microarray and RNA-seq normalization
- normalization of next generation reads in microbial ecology and ChIP-Seq analysis.

Summary

Finite mixture models

- Mixture of Normals with different means and variances.
- Decomposing the mixtures using the EM algorithm.

Infinite mixture models

- Gamma-Poisson for read counts.

R session

R packages we will use

```
library(cowplot)
library(dplyr)
library(flexmix)
library(ggbeeswarm)
library(ggplot2)
library(HistData)
library(MASS)
library(mixtools)
library(tibble)
library(vcd)
library(VGAM)
```

Fair mixture

Suppose we want two equally likely components.

Flip a fair coin.

- If it comes up heads
  - Generate a random number from a Normal with mean 1 and variance 0.25.
- If it comes up tails
  - Generate a random number from a Normal with mean 3 and variance 0.25.

How data are generated

```
coinflips = (runif(10000) > 0.5)
# table(coinflips)

oneFlip = function(fl, mean1 = 1, mean2 = 3, sd1 = 0.5, sd2 = 0.5) {
  if (fl) {
   rnorm(1, mean1, sd1)
  } else {
   rnorm(1, mean2, sd2)
  }
}
fairmix = vapply(coinflips, oneFlip, numeric(1))
```

Histogram

```
ggplot(tibble(values = fairmix), aes(x = values)) +
     geom_histogram(fill = "purple", binwidth = 0.1)
```

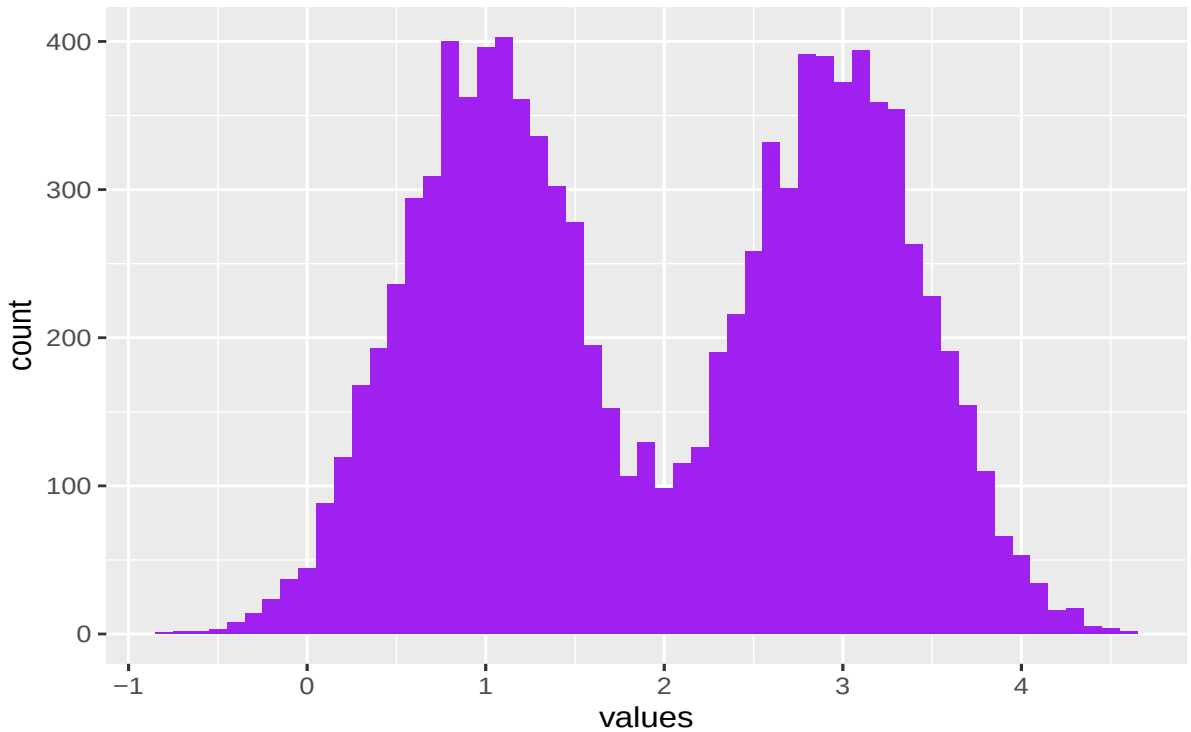


Fig. 9: Histogram of two normals from 10,000 random draws.

Questions

- How many modes are there? Is this surprising?
- Now plot the density using `geom_density()`.
- How is density related to hist?

How data are generated

```
pop1=rnorm(2000000)
pop2=rnorm(1000000, 1, 2)
combined = c(pop1, pop2)
data2plot = data.frame(data=c(combined, pop1, pop2),
                       labels=rep(c("combined", "pop1", "pop2"),
                                  c(3e6, 2e6, 1e6)))
plt= ggplot(data2plot, aes(x=data)) +
  stat_bin(aes(fill=labels), position="identity", binwidth=0.25, alpha=0.5) +
  theme_bw()
```

Question:

- What does this histogram show?
- Plot the densities of pop1 and pop2 on the same axes

Multivariate mixtures

Generate two clusters with 10,000 points each:

```
require(MASS)
Sigma=matrix(c(5,3,3,2),2,2)
ex1=mvrnorm(10000,rep(0,2),Sigma)
Sigma=matrix(c(9,-5,-1,5),2,2)
ex2=mvrnorm(n=10000, rep(3, 2), Sigma)
```

Question:

- Plot both clusters on the same axes, along with 2d contours of their densities.
- You will want to use `geom_point()` and `geom_density2d()`.

The EM algorithm step by step

As an example dataset, we use the values in the file `Myst.rds`. As always, it is a good idea to first visualize the data.

```
yvar = readRDS("data/Myst.rds")$yvar
ggplot(tibble(yvar), aes(x = yvar)) + geom_histogram(binwidth=0.025)
str(yvar)
```

The EM algorithm step by step

- We are going to model these data as a mixture of two normal distributions with unknown means and standard deviations. We'll call the two components A and B.
- We start by randomly assigning a "probability of membership" to each of the values in `yvar` for each of the groups, A and B. These are numbers between 0 and 1, `pA` represents the probability of coming from mixture component A. The complementary probability is `pB`.

```
yvar = readRDS("data/Myst.rds")$yvar
pA = runif(length(yvar))
pB = 1 - pA
```

The EM algorithm step by step

We also need to set up some housekeeping variables:

- `iter` counts over the iterations of the EM algorithm;
- `loglik` stores the current log-likelihood;
- `delta` stores the change in the log-likelihood from the previous iteration to the current one.
- We also define the parameters `tolerance`, `miniter` and `maxiter` of the algorithm.

```
iter = 0
loglik = -Inf
delta = +Inf
tolerance = 1e-3
miniter = 50; maxiter = 1000
```

The EM algorithm step by step

```
while( (delta > tolerance) && (iter <= maxiter) || (iter < miniter) ) {
  lambda = mean(pA)
  muA = weighted.mean(yvar, pA)
  muB = weighted.mean(yvar, pB)
  sdA = sqrt(weighted.mean((yvar - muA)^2, pA))
  sdB = sqrt(weighted.mean((yvar - muB)^2, pB))

  phiA = dnorm(yvar, mean = muA, sd = sdA)
  phiB = dnorm(yvar, mean = muB, sd = sdB)
  pA   = lambda * phiA
  pB   = (1 - lambda) * phiB
  ptot = pA + pB
  pA   = pA / ptot
  pB   = pB / ptot

  loglikOld = loglik
  loglik = sum(log(pA))
  delta = abs(loglikOld - loglik)
  iter = iter + 1
}
param = tibble(group = c("A","B"), mean = c(muA,muB), sd = c(sdA,sdB))
param

iter
```

The EM algorithm step by step

a. Which lines correspond to the E-step, and which to the M-step?
b. What does the M-step do, and what does the E-step do?
c. What is the role of the algorithm arguments `tolerance`, `miniter` and `maxiter`?
d. Why do we need to compute `loglik`?

R implementations of EM algorithm

Several R packages provide EM implementations, including

- mclust
- EMcluster
- EMMIXskew

Choose one and run the EM function several times with different starting values.

Then use the function `normalmixEM` from the mixtools package to compare the outputs.

Output from **mixtools**

```
library("mixtools")
y = c(rnorm(100, mean = -0.2, sd = 0.5),
      rnorm( 50, mean =  0.5, sd =   1))
gm = normalmixEM(y, k = 2, lambda = c(0.5, 0.5),
                 mu = c(-0.01, 0.01), sigma = c(1, 1))
```

```
## number of iterations= 165
```

```
gm$lambda
```

```
## [1] 0.870093 0.129907
```

```
gm$mu
```

```
## [1] -0.1036505  1.6523308
```

```
gm$sigma
```

```
## [1] 0.4999119 0.4946658
```

```
gm$loglik
```

```
## [1] -156.5321
```

Output from **EMCluster**

```
library(EMCluster)
y1 <- matrix(y,ncol=1)
emobj <- init.EM(y1, nclass = 2)
summary(emobj)

ret <- emcluster(y1, emobj, assign.class = TRUE)
summary(ret)
```

The Gamma distribution

```
nr=10000;
set.seed(20130607)
outg=rgamma(nr,shape=2,scale=3)
p=qplot(outg,geom="histogram",binwidth=1,color="purple")
p
```

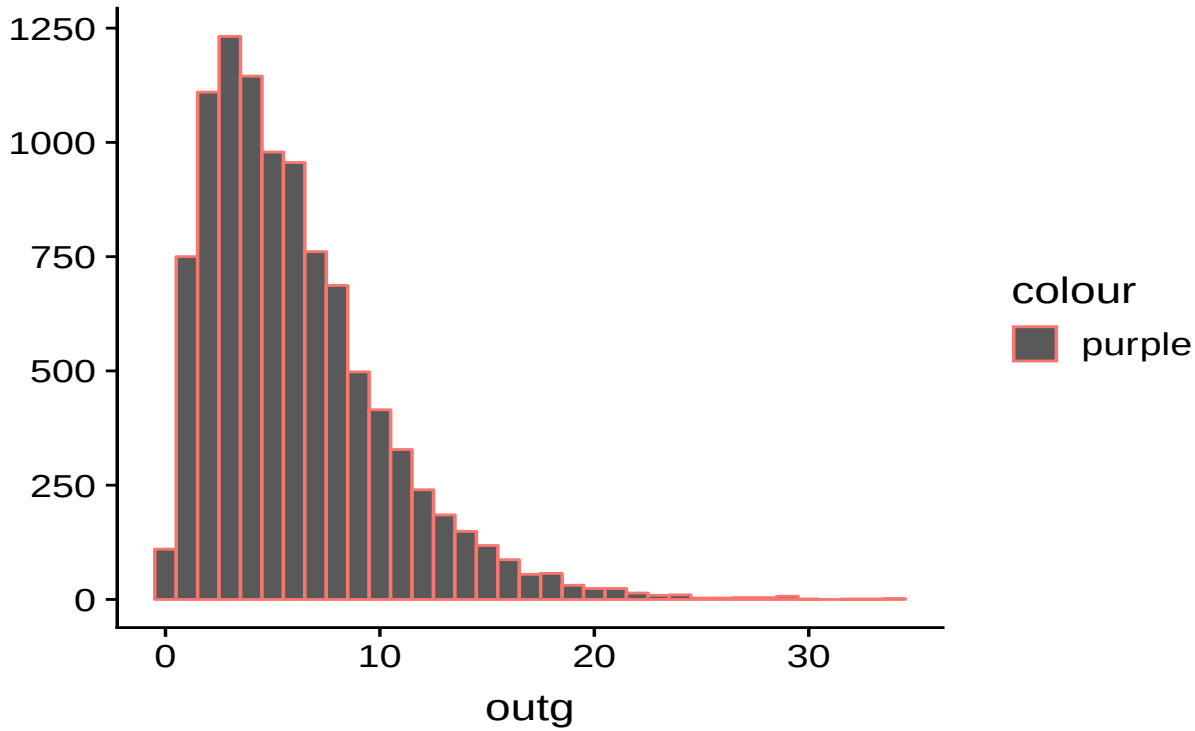


Fig. 10: Histogram of data generated from Gamma(2,3).

The Gamma density

```
require(ggplot2)
pts=seq(0,max(outg),0.5)
outf=dgamma(pts,shape=2,scale=3)
p=qplot(pts,outf,geom="line")
p + theme_bw(10)
```

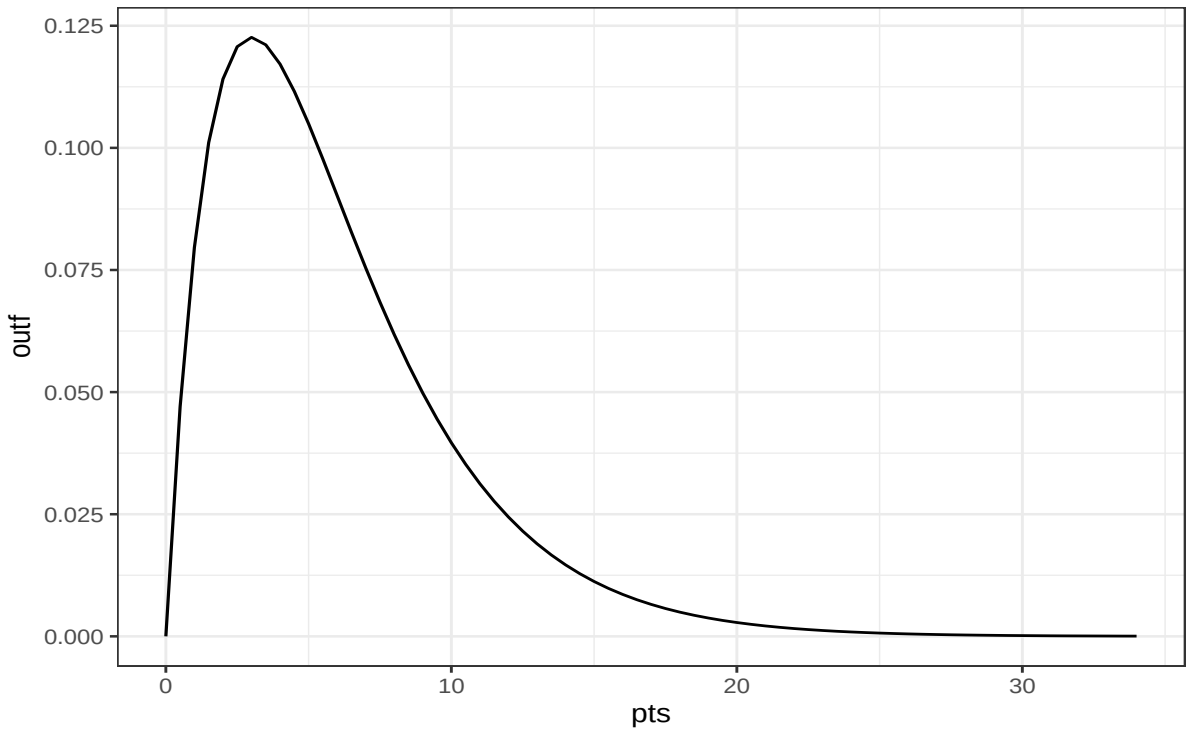


Fig. 11: Theoretical density of Gamma(2,3).

Fitting distributions with R

```
require(MASS)
## avoid spurious accuracy
op = options(digits = 3)
set.seed(123)
x = rgamma(100, shape = 5, rate = 0.1)
####Rate is usually the second or 1/scale
fitdistr(x, "gamma")
```

```
##    shape     rate
##   6.4870   0.1365
##  (0.8946) (0.0196)
```

```
## now do this directly with more control.
fitdistr(x, dgamma, list(shape = 1, rate = 0.1), lower = 0.001)
```

```
##    shape     rate
##   6.4869   0.1365
##  (0.8944) (0.0196)
```

```
# lower: bounds on your parameters
```

Gamma mixture of Poissons: a hierarchical model

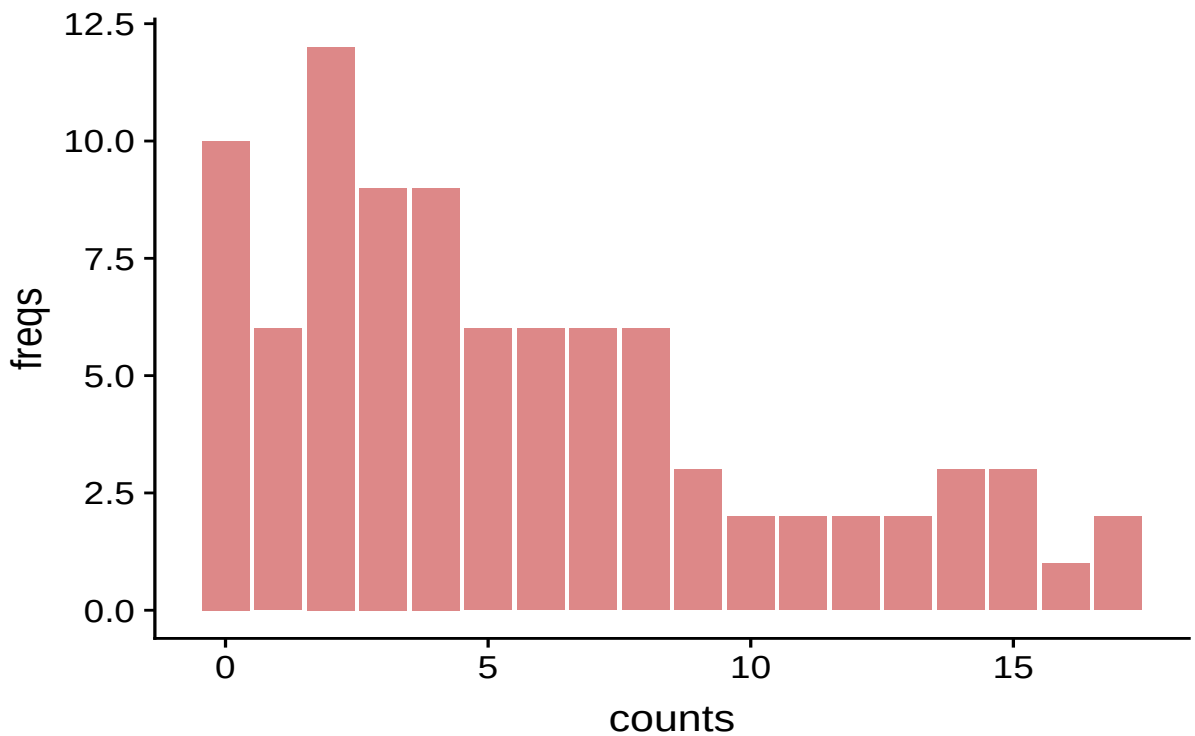
1. Generate a whole set of Poisson parameters: $\lambda_1, \lambda_2, ...\lambda_{90}$ from a Gamma(2,3) distribution.
2. Generate a set of Poisson(λ_i) random variables.

```
set.seed(1001015)
ng=90;
lambdas=rgamma(ng,shape=2,scale=3)
veco=rep(0,ng)
for (j in (1:ng)){
  veco[j]=rpois(1,lambda=lambdas[j])
}
```

Data from Gamma-Poisson

```
veco.df <- data.frame("counts"=veco)
veco.df <- veco.df %>%
  group_by(counts) %>%
  summarise(freqs = n())

ggplot(data=veco.df, aes(x=counts, y=freqs)) +
  geom_bar(stat="identity",fill="#DD8888")
```



Data from negative binomial

```
set.seed(321)
cts=0:11
out=dnbinom(cts,size=4,p=0.5)
dfnb=data.frame(counts=cts,freqs=out)
ggplot(data=dfnb, aes(x=counts, y=freqs)) +
  geom_bar(stat="identity",fill="#DD8888")
```

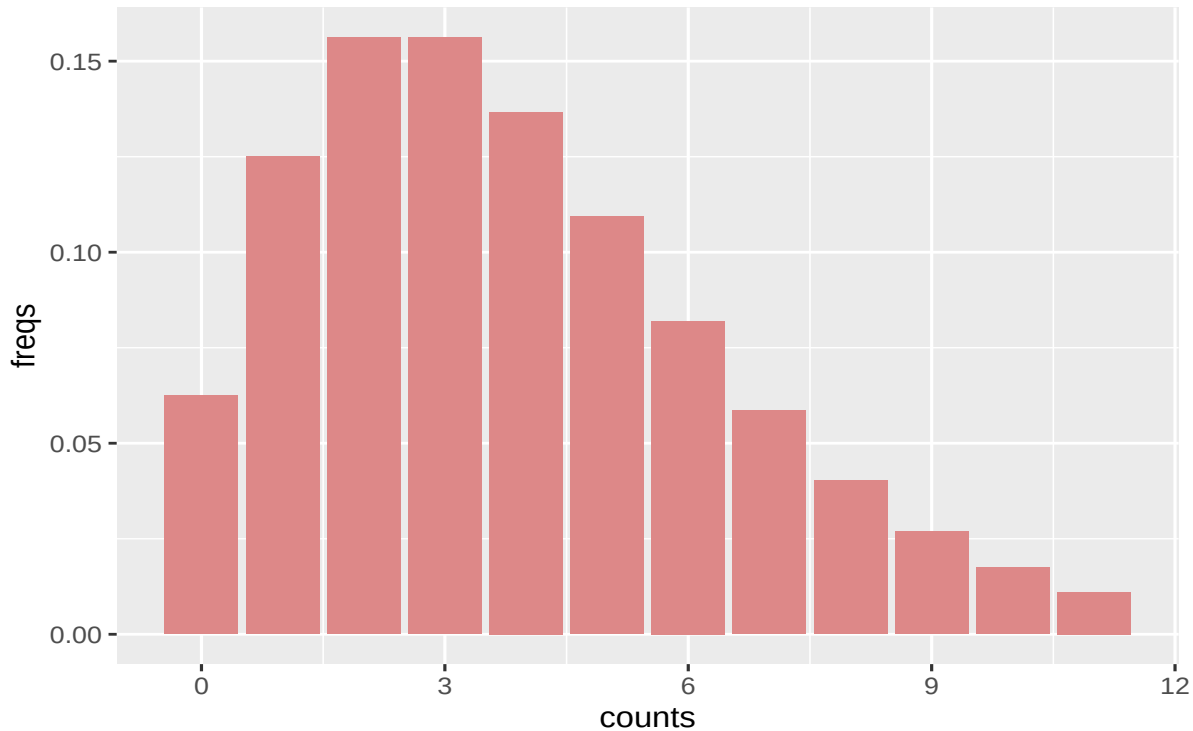


Fig. 12: Barplot from a Negative Binomial with p=0.5, until 4 successes.

Fitting a negative binomial model

```
require(vcd)
goodnb=goodfit(veco,"nbinomial")
summary(goodnb)
```

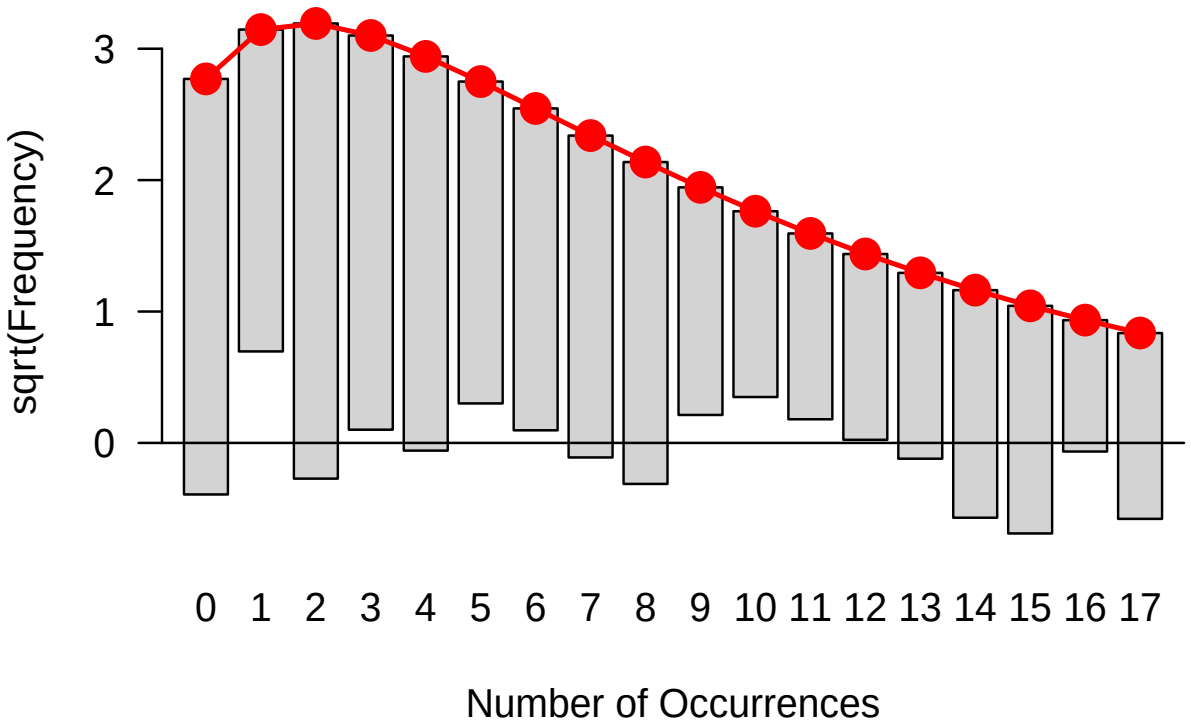
```
##
##   Goodness-of-fit test for nbinomial distribution
##
##                     X^2 df P(> X^2)
## Likelihood Ratio 15.2 15    0.435
```

```
goodnb$par
```

```
## $size
## [1] 1.67
##
## $prob
## [1] 0.23
```

Visual goodness of fit test

```
#table(veco)
plot(goodnb)
```



Fitting a negative binomial model

We can also use the `fitdistr` function to fit a Negative Binomial model.

```
set.seed(123)
x4 = rnbinom(500, size=4, mu = 5)
#theta is the size parameter r
fitdistr(x4, "Negative Binomial")
```

```
##     size      mu
##    4.290   5.062
##  (0.516) (0.149)
```

Note

- Mean of NB(r, p) is $\mu = \frac{r(1-p)}{p}$.