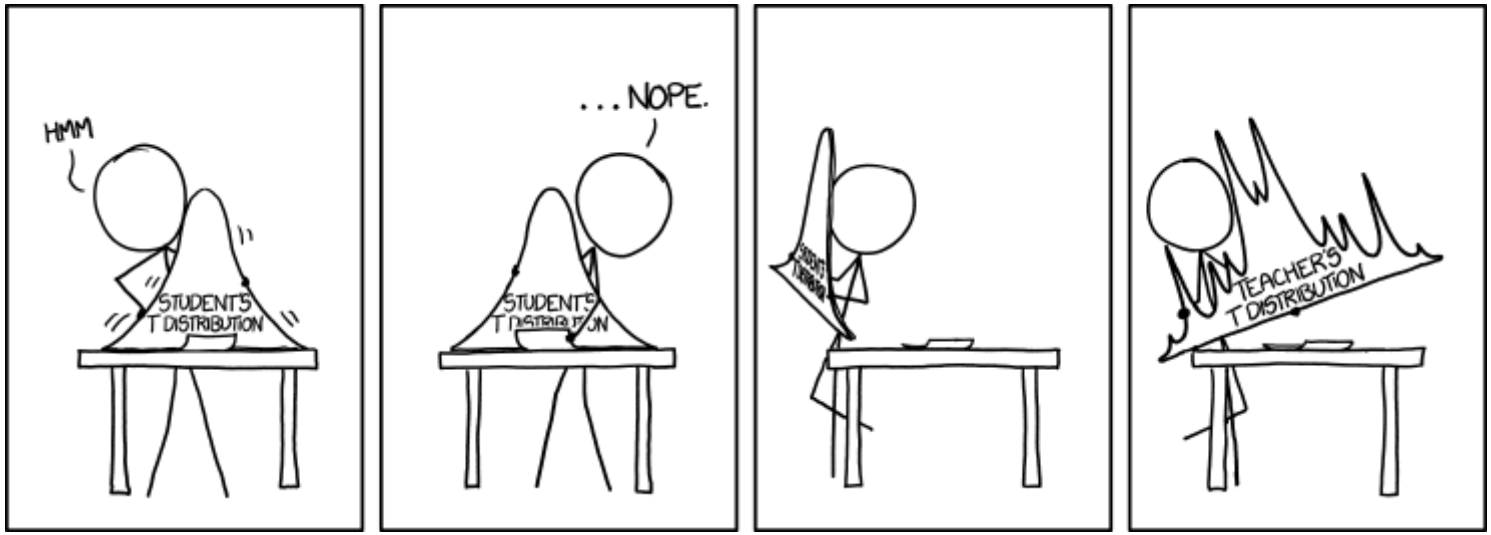
Mixture Models

Susan Holmes and Wolfgang Huber

October 2nd, 2019



Teachers distribution

Mixture Models and Simulations

Why mixture models?

Many sources of variation that can't be thought of as additive.

Types of Mixture Models

There are two types of mixture models we will discuss: finite and infinite

Simple Examples and computer experiments

Suppose we want two equally likely components, we decompose the generating process into steps:

Flip a fair coin.

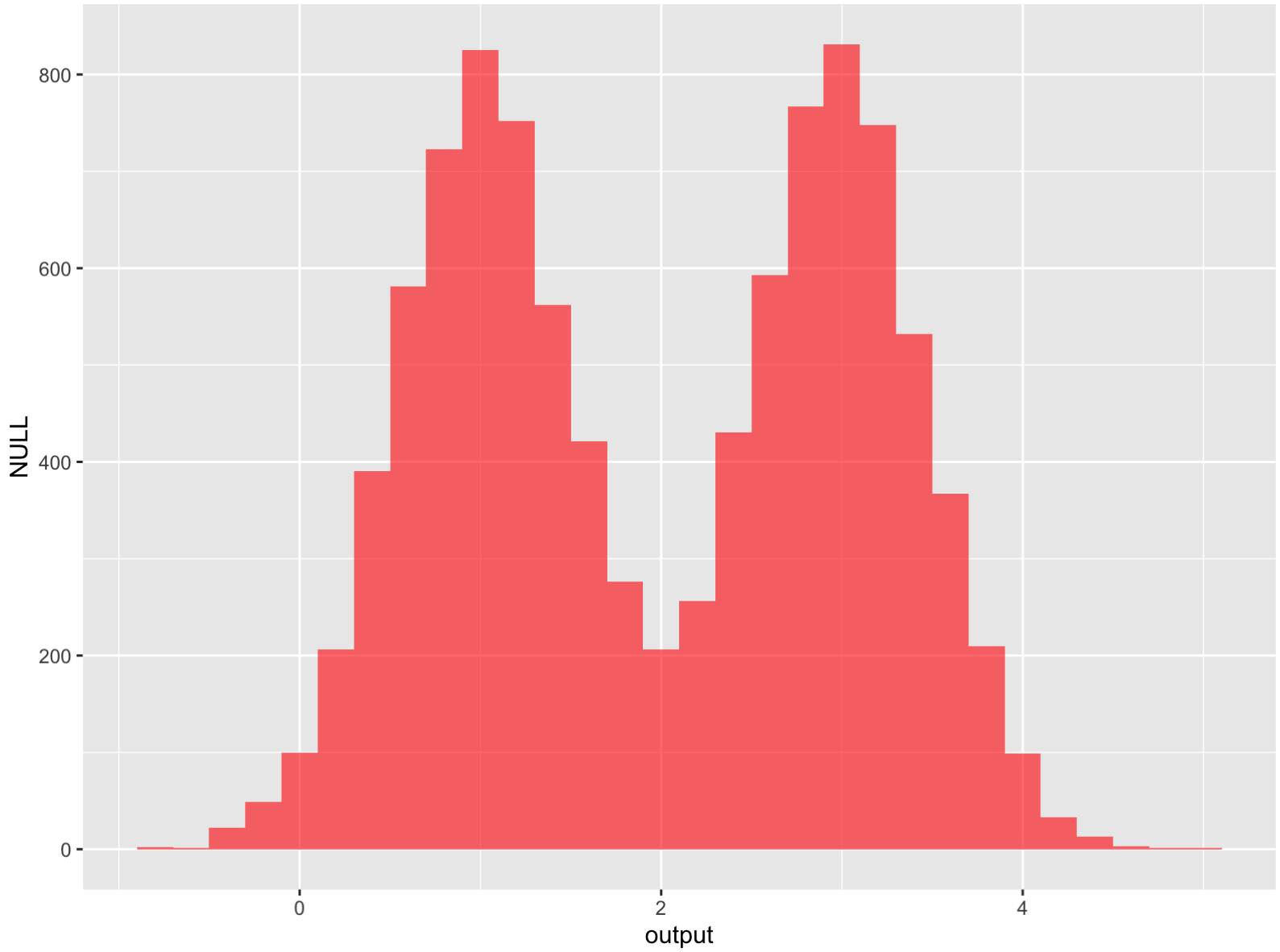
- If it comes up heads
  + Generate a random number from a Normal with mean 1 and variance 0.25.
- If it comes up tails
  + Generate a random number from a Normal with mean 2 and variance 0.25.

Resulting histogram if we did this 10,000 times.

```
require(ggplot2)
coinflips=as.numeric(runif(10000)>0.5)
table(coinflips)
```

```
## coinflips
##    0    1
## 5018 4982
```

```
output=rep(0,10000)
sd1=0.5;sd2=0.5;mean1=1;mean2=3
for (i in 1:10000){
  if (coinflips[i]==0)
  output[i]=rnorm(1,mean1,sd1)
  else
    output[i]=rnorm(1,mean2,sd2)  }
group=coinflips+1
do=data.frame(output)
qplot(output,data=do,geom="histogram",fill=I("red"),binwidth=0.2,alpha=I(0.6))
```



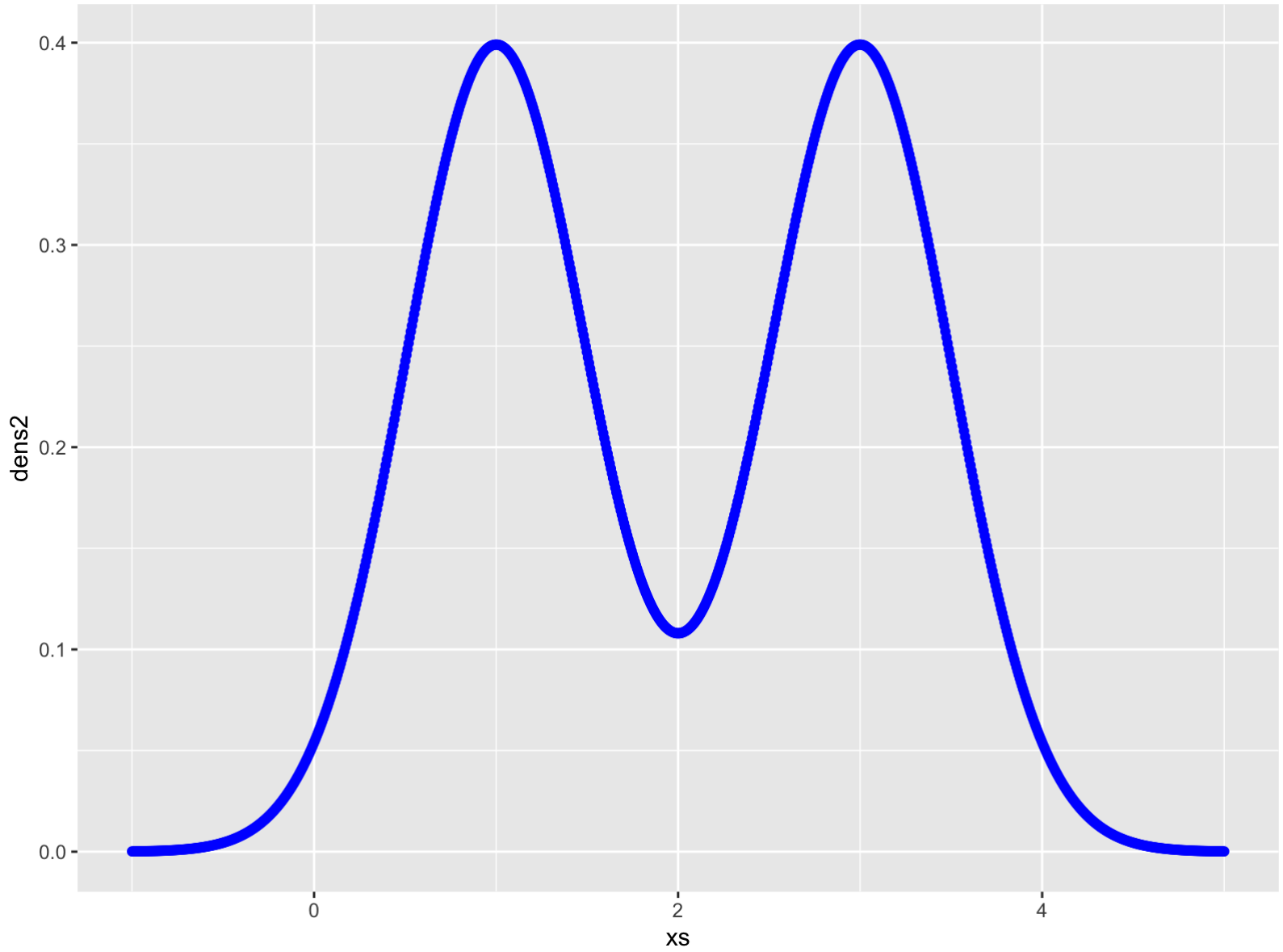
In fact we can write the density (the limiting curve that the histograms tend to look like) as

$$f(x) = \frac{1}{2}\phi_1(x) + \frac{1}{2}\phi_2(x)$$

where ϕ_1 is the density of the Normal($\mu_1 = 1, \sigma^2 = 0.25$) and ϕ_2 is the density of the Normal($\mu_2 = 2, \sigma^2 = 0.25$).

```
xs=seq(-1,5,length=1000)
dens2=0.5*dnorm(xs,mean=1,sd=0.5)+
      0.5*dnorm(xs,mean=3,sd=0.5)
do=data.frame(xs,dens2)
qplot(xs,dens2,type='l',col=I("blue"),data=do)
```

```
## Warning: Ignoring unknown parameters: type
```



In this case of course the mixture model was extremely visible as the two distributions don't overlap, this can happen if we have two very separate populations, for instance different species of fish whose weights are very different. However if many cases the separation is not so clear.

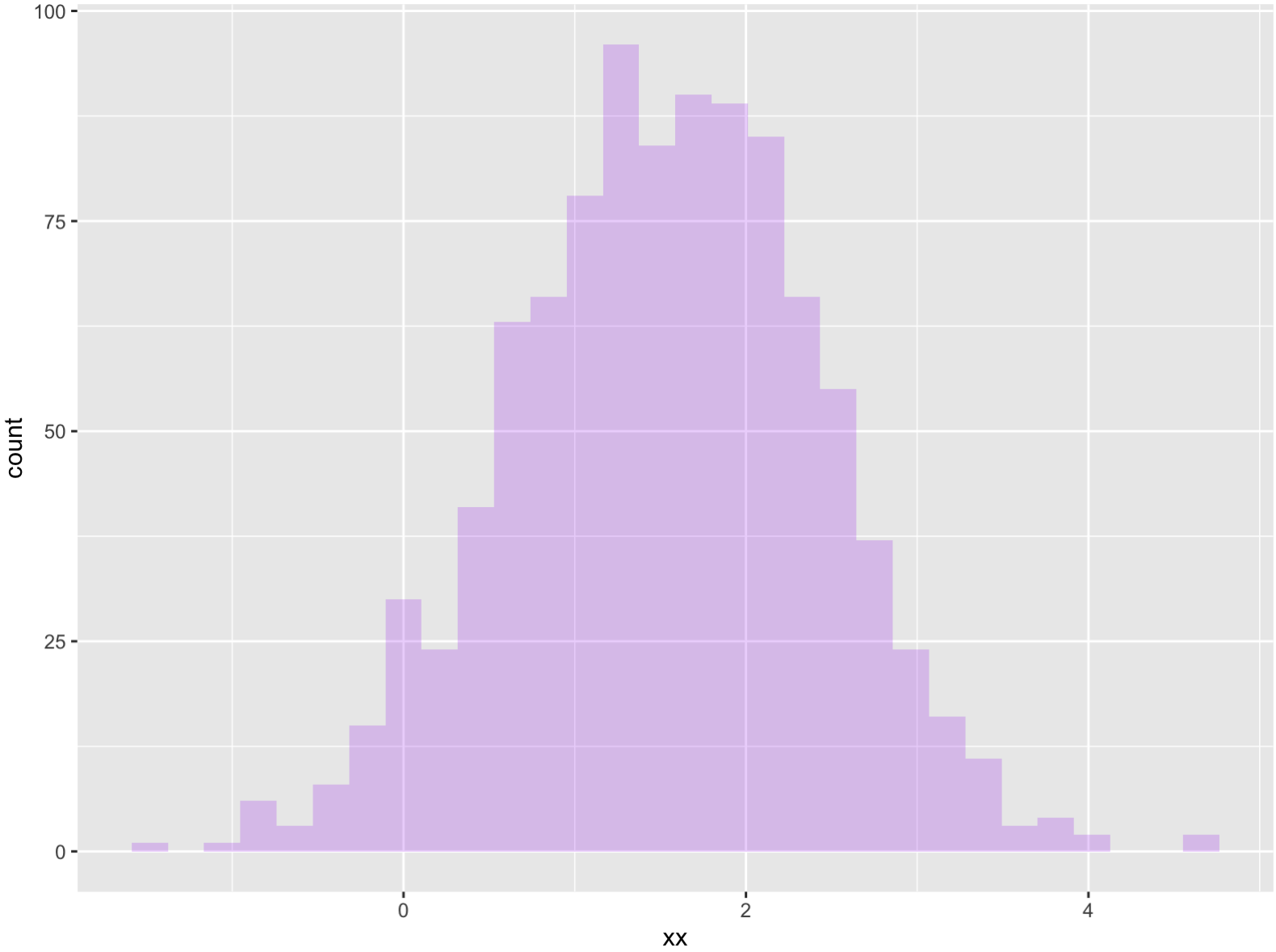
Challenge: Here is a histogram generated by two Normals with the same variances, can you guess the two parameters for these two Normals?

```
require(ggplot2)
set.seed(1233341)
coinflips=as.numeric(runif(1000)>0.5)
table(coinflips)
```

```
## coinflips
##   0   1
## 495 505
```

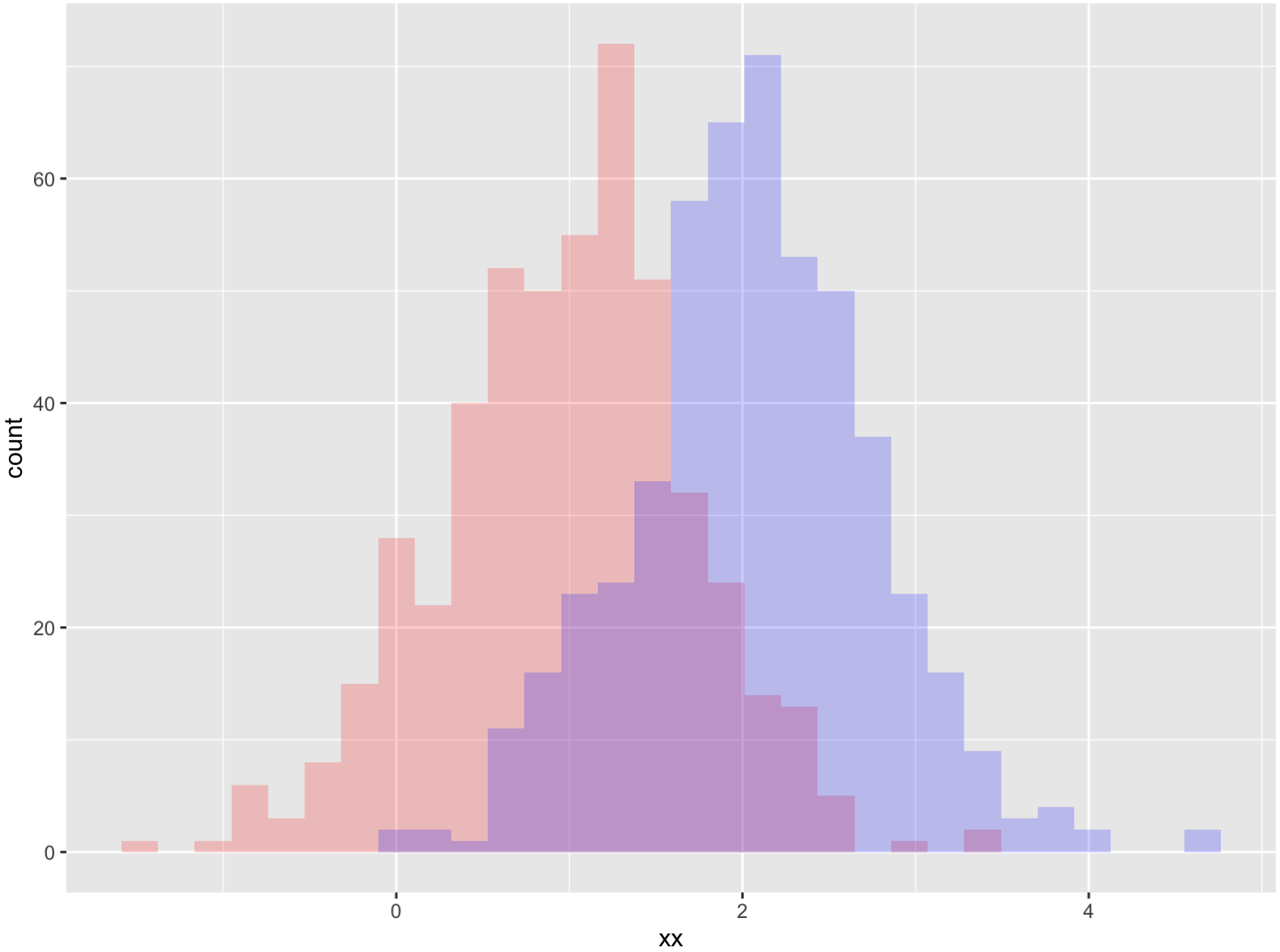
```
output=rep(0,1000)
sd1=sqrt(0.5)
sd2=sqrt(0.5)
mean1=1
mean2=2
for (i in 1:1000){
  if (coinflips[i]==0)
  output[i]=rnorm(1,mean1,sd1)
  else
    output[i]=rnorm(1,mean2,sd2)
}
group=coinflips+1
```

```
dat=data.frame(xx=output,yy=group)
ggplot(dat,aes(x=xx)) +
    geom_histogram(data=dat,fill = "purple", alpha = 0.2)
```



Here is the answer: if we color in red the points that were generated from the heads coin flip and blue the one from tails, we can see that the first normal has a range of about

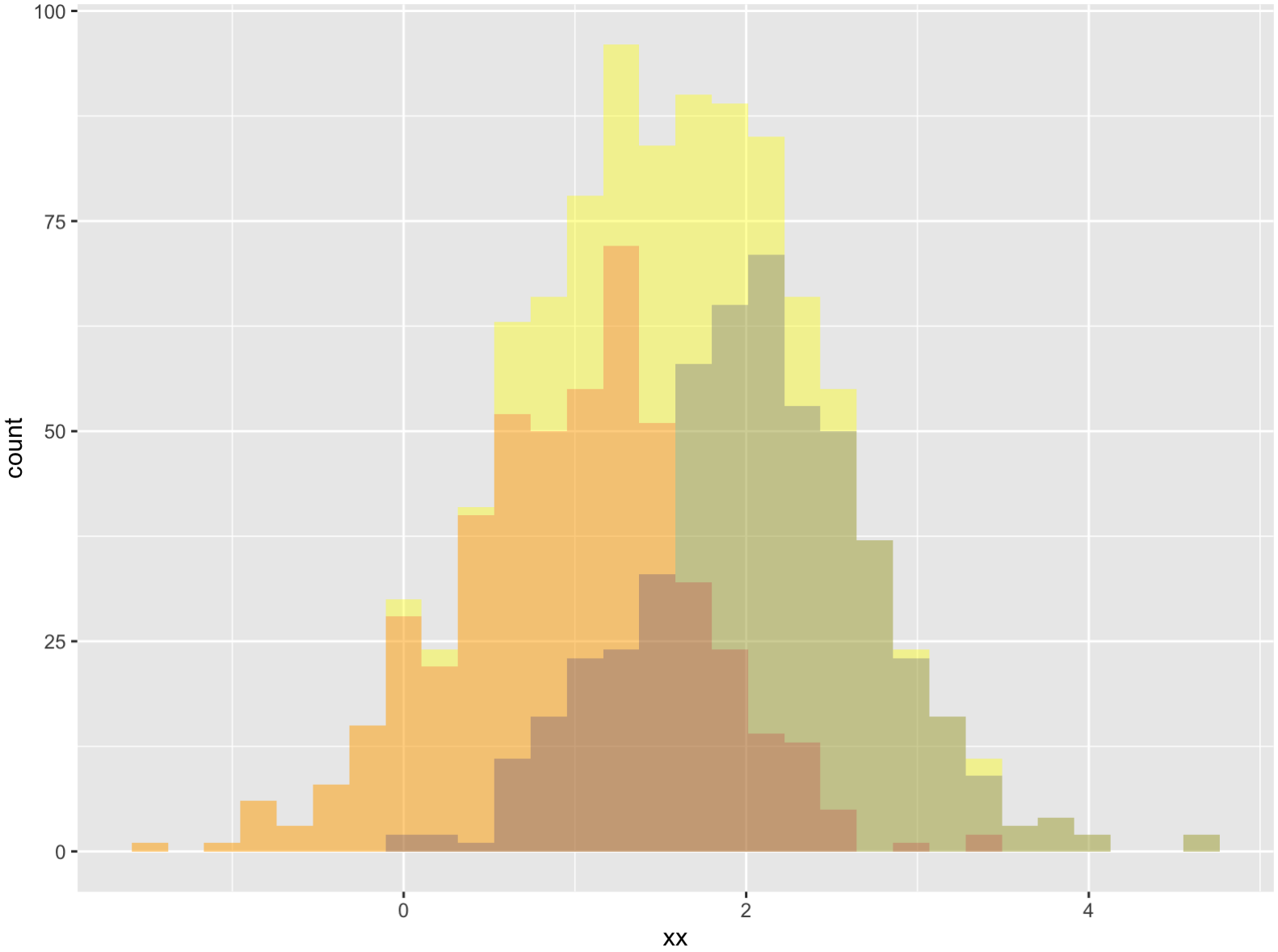
```r
dat <- data.frame(xx=output,yy = group)
ggplot(dat,aes(x=xx)) +
    geom_histogram(data=subset(dat,yy == 1),fill = "red", alpha = 0.2) +
    geom_histogram(data=subset(dat,yy == 2),fill = "blue", alpha = 0.2)
```



Less obvious mixture of two Normals: components colored in red and blue.

The overlapping points are going to be piled up on top of each other in the final histogram, here is an overlayed plot showing the three histograms

```
ggplot(dat,aes(x=xx)) +
    geom_histogram(data=dat,fill = "yellow", alpha = 0.4)+
    geom_histogram(data=subset(dat,yy == 1),fill = "red", alpha = 0.2) +
    geom_histogram(data=subset(dat,yy == 2),fill = "darkblue", alpha = 0.2)
```



Less obvious mixture of two Normals: components colored in orange and green.

Here 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 book MacLachlan, (2004) provides a complete treatment of the subject of finite mixtures.

Discovering the hidden class: EM

First we use a method called the EM (Expectation-Maximization) algorithm to infer the value of the hidden variable.

On Monday we will do clustering which is does not use the same model.

The Expectation-Maximization algorithm is an **alternating** and **iterative** procedure.

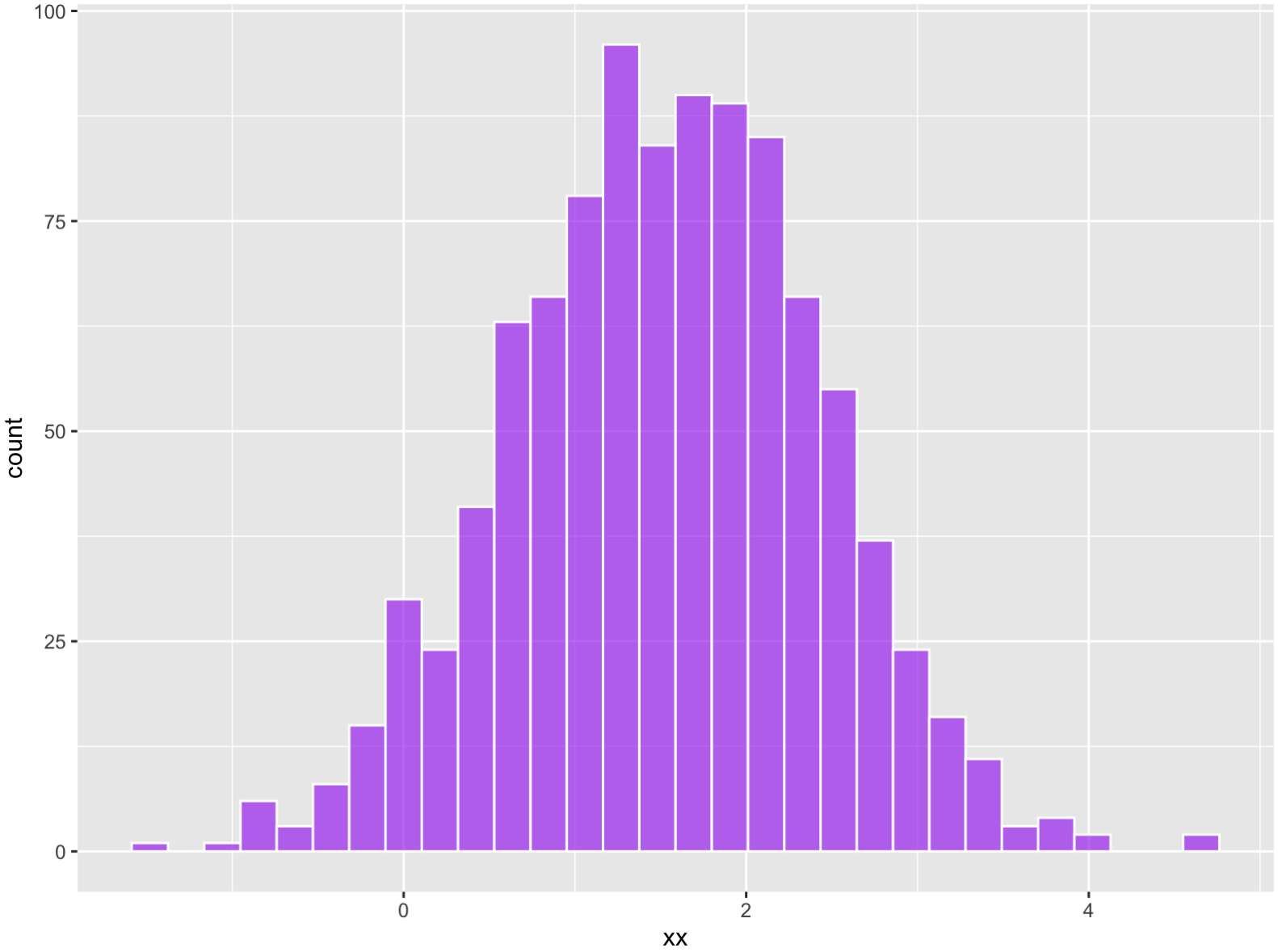
Start with observations $Y = y_1, y_2, \ldots, y_n$ and we **augment** the data with an unobserved (latent) cluster variable U, which says which group each observation came. U=`group`

$$(y_1, u_1), (y_2, u_2), \ldots, (y_n, u_n)$$

We are interested in finding the values of U and the unknown parameters of the underlying densities that make the observed data Y the most likely.

Two normals example

```
ggplot(dat,aes(x=xx)) +
    geom_histogram(data=dat,fill = "purple", col="white", alpha = 0.6)
```




```
dat$xx[1:15]
```

```
##  [1]  1.7530620  1.2659780  1.5606189  1.9925737  1.2565143  0.6081198
##  [7]  2.0529329 -0.3323572  1.1305426  1.2929795  2.1768070  1.4380620
## [13]  2.2939339  1.7879879  2.7537675
```

```
x=dat$xx
```

Bivariate distribution here: distribution of `couples` (Y, U)

$$f(y, u|\theta) = f(u|y, \theta)f(y|\theta)$$

Suppose we have a fair mixture of two normals with parameters $\theta = (\mu_1 =?, \mu_2 =?, \sigma_1 = 0.5, \sigma_2 = 0.5)$, μ_1 and μ_2 are unknown, we suppose for now, we know that the standard deviations of both distributions is 0.5.

- If we knew the labels u we could use maximum likelihood to compute the mu1 and mu2 means of the distributions.
- If we knew the **true** means mu_1 and mu_2, we could assign the u's to the more likely group.

We pretend each of these in turn.

For this bivariate distribution we can define a `complete joint likelihood`, we usually work with its log

$$\text{loglikeli}(\theta) = \log f(y, u|\theta)$$

Marginal likelihood for the observed y:

$$\text{marglike}(\theta; Y) = f(Y|\theta) = \sum_{u} f(y, u|\theta)$$

Intiatize the parameter θ to any value θ^*

For instance 50-50 in each group and mu1= -0.05 and mu2 =+0.05 sigma1=0.5 and sigma2=0.5

```
sum(0.5*dnorm(x,-0.5,0.5)+0.5*dnorm(x,0.5,0.5))
```

```
## [1] 144.0505
```

If each point had probability 0.5 of belonging to each group:

```
sum(0.5*dnorm(x,-0.5,0.5)+0.5*dnorm(x,0.5,0.5))
```

```
## [1] 144.0505
```

```
sum(0.5*dnorm(x,-0.6,0.5)+0.5*dnorm(x,0.9,0.5))
```

```
## [1] 181.9464
```

```
sum(0.5*dnorm(x,-2,0.5)+0.5*dnorm(x,3,0.5))
```

```
## [1] 70.21037
```

However, this is not true, we are going to add in the probabilities of the different u's in the computation of the likelihood.

```
##P1's
dnorm(x,-0.5,0.5)[1:5]
```

```
## [1] 3.109796e-05 1.559878e-03 1.635991e-04 3.202308e-06 1.667423e-03
```

```
##P2's
dnorm(x,0.5,0.5)[1:5]
```

```
## [1] 0.034523319 0.246785475 0.084111155 0.009266551 0.254000512
```

E `expectation' step:

Use group probabilities under the current model giving $p(y, u|\theta^*)$ that are used to compute the expectation

$$\sum_u p(u|y, \theta^*) \log f(\theta, y, u) = E_{u|y,\theta^*} \log f(\theta, y, u) = Q(\theta, \theta^*)$$

M `maximization' step:

Estimate distribution parameters by maximizing the log likelihood $Q(\theta, \theta^*)$\ This gives a new θ^*.

Store cluster probabilities as instance weights $p(u|y, \theta^*)$.

$E^*(\theta)$

$$E^*(\theta) = E_{\theta^*,X}[\log p(u, x|\theta^*)] = \sum_u p(u|x, \theta^*) \log p(u, x|\theta^*).$$

The value of θ that maximizes E^* is found in what is known as the **M** aximization step.

These two iterations (**E** and **M**) are repeated until the improvements are small; this is a numerical indication that we are close to a flattening of the likelihood and so we have reached a local maximum.

We need to use several initial starting points to ensure that we always get the same answer.

Remarks:

The EM algorithm is very instructive:

1. It shows us how we can alternate tackling different unknowns in a problem eventually finding estimates of hidden variables.

2. It provides a first example of *soft* averaging i.e., where we don't decide whether a point belongs to one group or another, but allow it to

participate in several groups by using probabilities of membership as weights providing more nuanced estimates.

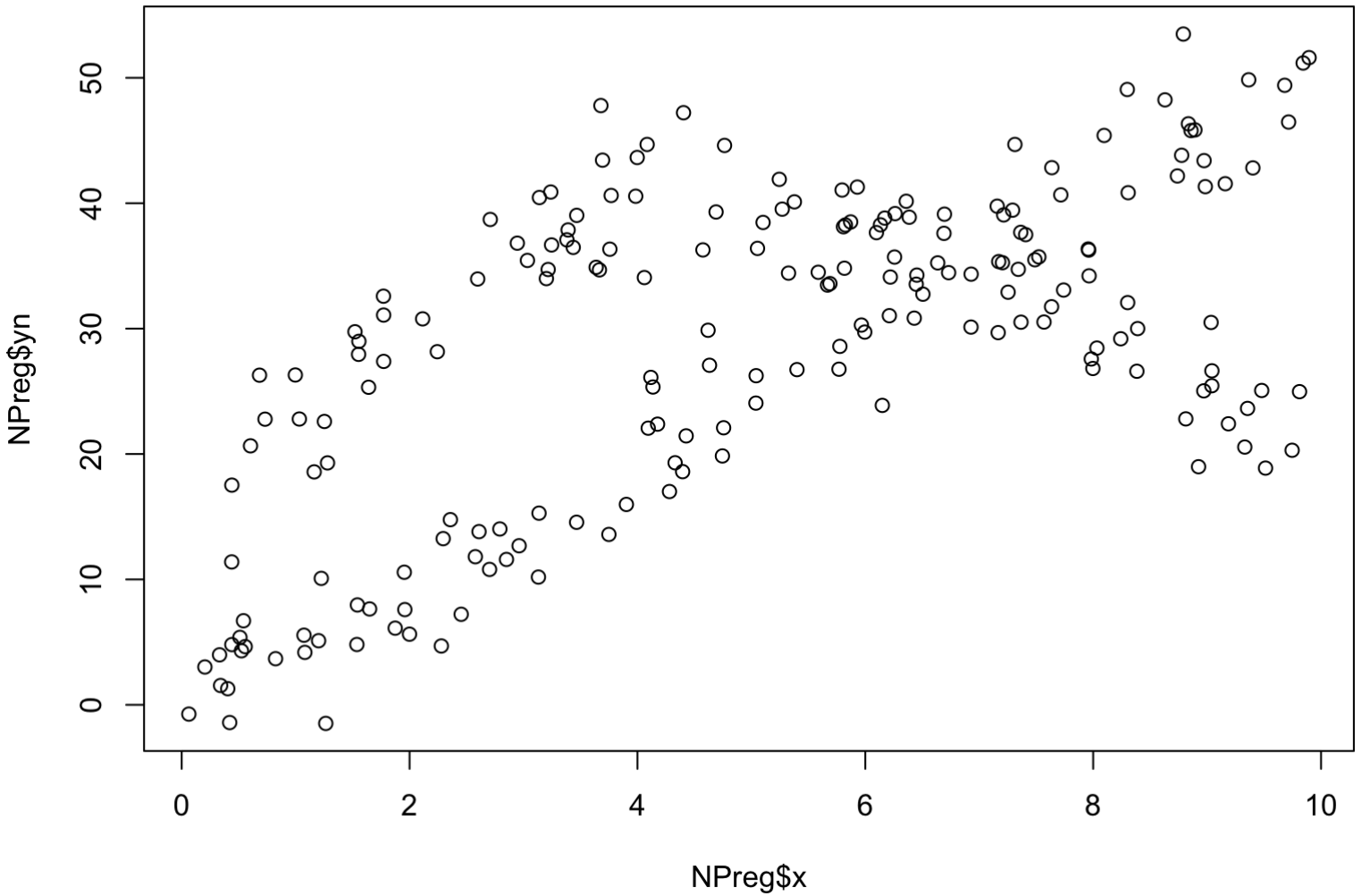
3. The method employed here can be extended to the more general case of **model-averaging** , where more complex models replace the clusters we are dealing with. When we are uncertain which model is correct for the data at hand we can average models with weights given by their likelihoods.

Stop when improvement is negligible.

Mixture Modeling Examples for Regressions

The flexmix package allows to cluster and fit regressions to the data at the same time. The standard M-step `FLXMRglm()` of FlexMix is an interface to R's generalized linear modelling facilities - `glm()` function.

```
require(flexmix)
data(NPreg)
plot(NPreg$x,NPreg$yn)
```



NPreg x and yn data scatterplot

As a simple example we use artificial data with two latent classes of size 100 each:

$$Class\ 1: \quad y = 5x + \epsilon$$
$$Class\ 2: \quad y = 15 + 10x - x^2 + \epsilon$$

with $\epsilon \sim N(0, 9)$ and prior class probabilities $\pi_1 = \pi_2 = 0.5$.

We can fit this model in R using the commands

```
library("flexmix")
data("NPreg")
m1 = flexmix(yn ~ x + I(x^2), data = NPreg, k = 2)
m1
```

```
##
## Call:
## flexmix(formula = yn ~ x + I(x^2), data = NPreg, k = 2)
##
## Cluster sizes:
##   1   2
## 100 100
##
## convergence after 13 iterations
```

and get a first look at the estimated parameters of mixture component~1 by

```
parameters(m1, component = 1)
```

```
##                     Comp.1
## coef.(Intercept) 14.7171315
## coef.x            9.8462869
## coef.I(x^2)      -0.9683139
## sigma             3.4801398
```

and

```
parameters(m1, component = 2)
```

```
##                       Comp.2
## coef.(Intercept) -0.20945380
## coef.x            4.81724681
## coef.I(x^2)       0.03621418
## sigma             3.47590252
```

for component 2. The parameter estimates of both components are close to the true values. A cross-tabulation of true classes and cluster memberships can be obtained by

```
table(NPreg$class, clusters(m1))
```

```
##
##      1  2
##   1  5 95
##   2 95  5
```

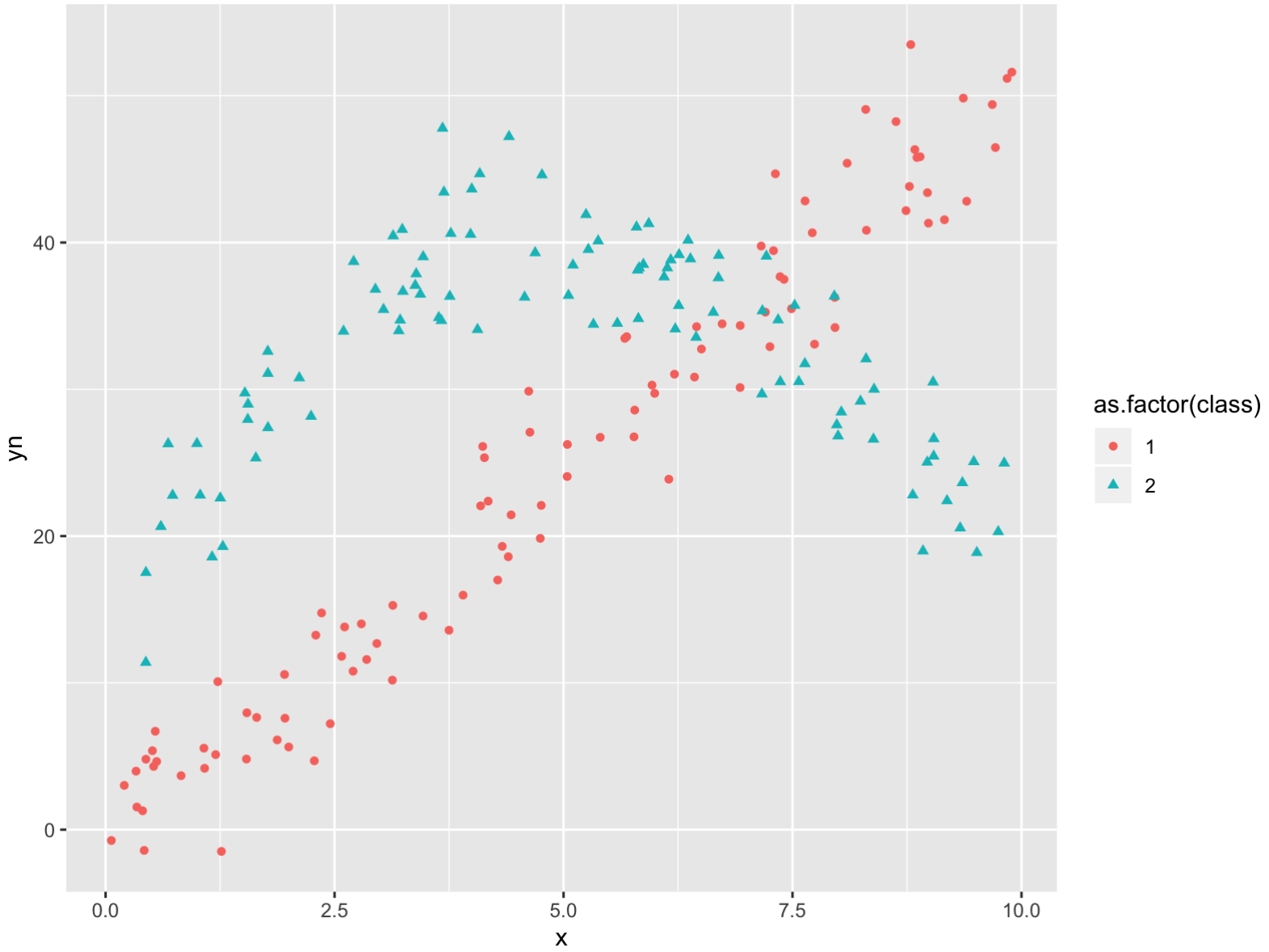
The summary method

```
summary(m1)
```

```
##
## Call:
## flexmix(formula = yn ~ x + I(x^2), data = NPreg, k = 2)
##
##        prior size post>0 ratio
## Comp.1 0.506  100    141 0.709
## Comp.2 0.494  100    145 0.690
##
## 'log Lik.' -642.5452 (df=9)
## AIC: 1303.09   BIC: 1332.775
```

gives the estimated prior probabilities $\hat{\pi}_k$, the number of observations assigned to the corresponding clusters, the number of observations where $p_{nk} > \delta$ (with a default of $\delta = 10^{-4}$), and the ratio of the latter two numbers. For well-seperated components, a large proportion of observations with non-vanishing posteriors p_{nk} should also be assigned to the corresponding cluster, giving a ratio close to 1. For our example data the ratios of both components are approximately 0.7, indicating the overlap of the classes at the cross-section of line and parabola.

```
ggplot(NPreg,aes(x,yn)) +geom_point(aes(colour = as.factor(class),shape=as.factor(class)))
```



Regression example using `flexmix`

Zero inflated models

$$f_{\text{zi}} = \alpha\delta(y) + (1-\alpha)f_{\text{count}}(y) \qquad \text{where } \delta(y) = 1 \text{ and } 0 \text{ elsewhere}$$

There are many examples and functions for **zero inflated** counts.

We will see late how we can try to tease out these clusters and assign a group to many of the observations without knowing the distributions, in the nonparametric setting this is called clustering.

Real Example of zero-inflation

Example: CHip-seq data

Let's consider the example of ChIP-sequencing data. These data are sequences that result from using chromatin immunoprecipitation (ChIP) assays that identify genome-wide DNA binding sites for transcription factors and other proteins.

This enables the mapping of the chromosomal locations of transcription factors, nucleosomes, histone modifications, chromatin remodeling enzymes, chaperones, and polymerases. This mapping the main technology used by the Encyclopedia of DNA Elements (ENCODE) Project. Here we use an example from the package which shows data measured on chromosome 22 from ChIP-seq counts of STAT1 binding and H3K4me3 modification in the GM12878 cell line.

```
library("mosaics")
library("mosaicsExample")
```

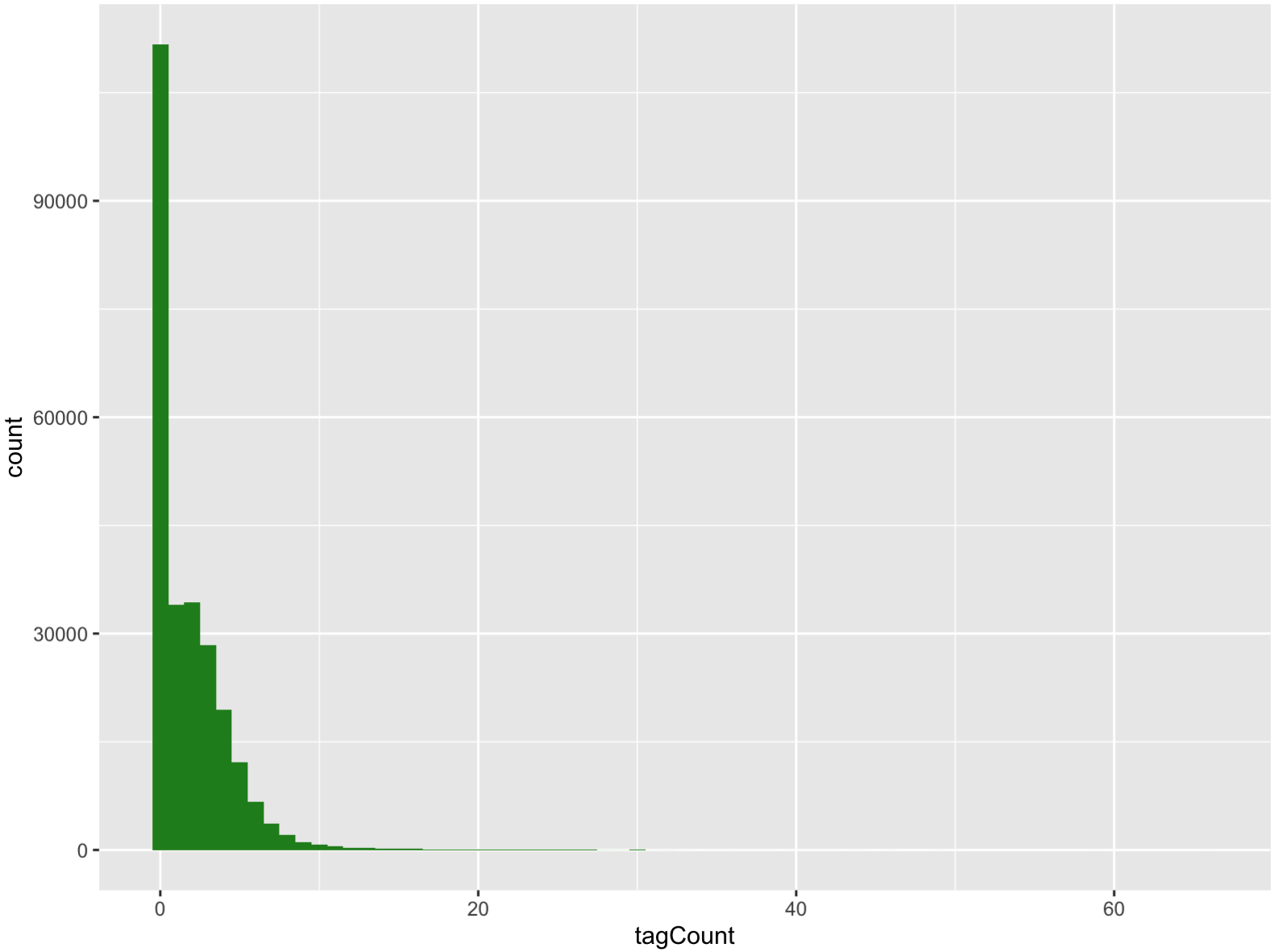
We read in the data as shown in the vignette and transform the BinData object into a simple data.frame (the code for preprocessing the data is not displayed).

```
## Summary: bin-level data (class: BinData)
## ----------------------------------------
## - # of chromosomes in the data: 1
## - total effective tag counts: 462479
##   (sum of ChIP tag counts of all bins)
## - control sample is incorporated
## - mappability score is NOT incorporated
## - GC content score is NOT incorporated
```

```
## - uni-reads are assumed
## ----------------------------------------
```

We can then create a histogram of the data as shown in Figure .

```
bincts = print(binTFBS)
ggplot(bincts,aes(x=tagCount)) +
 geom_histogram(binwidth=1, fill="forestgreen")
```



Over-abundant/over-expressed genes/proteins/taxa

Mainstay in multiple testing when trying to find relevant genes in microarray and RNA-seq or proteiomic studies

$$f_m = p_0 f_u + (1 - p_0) f_e$$

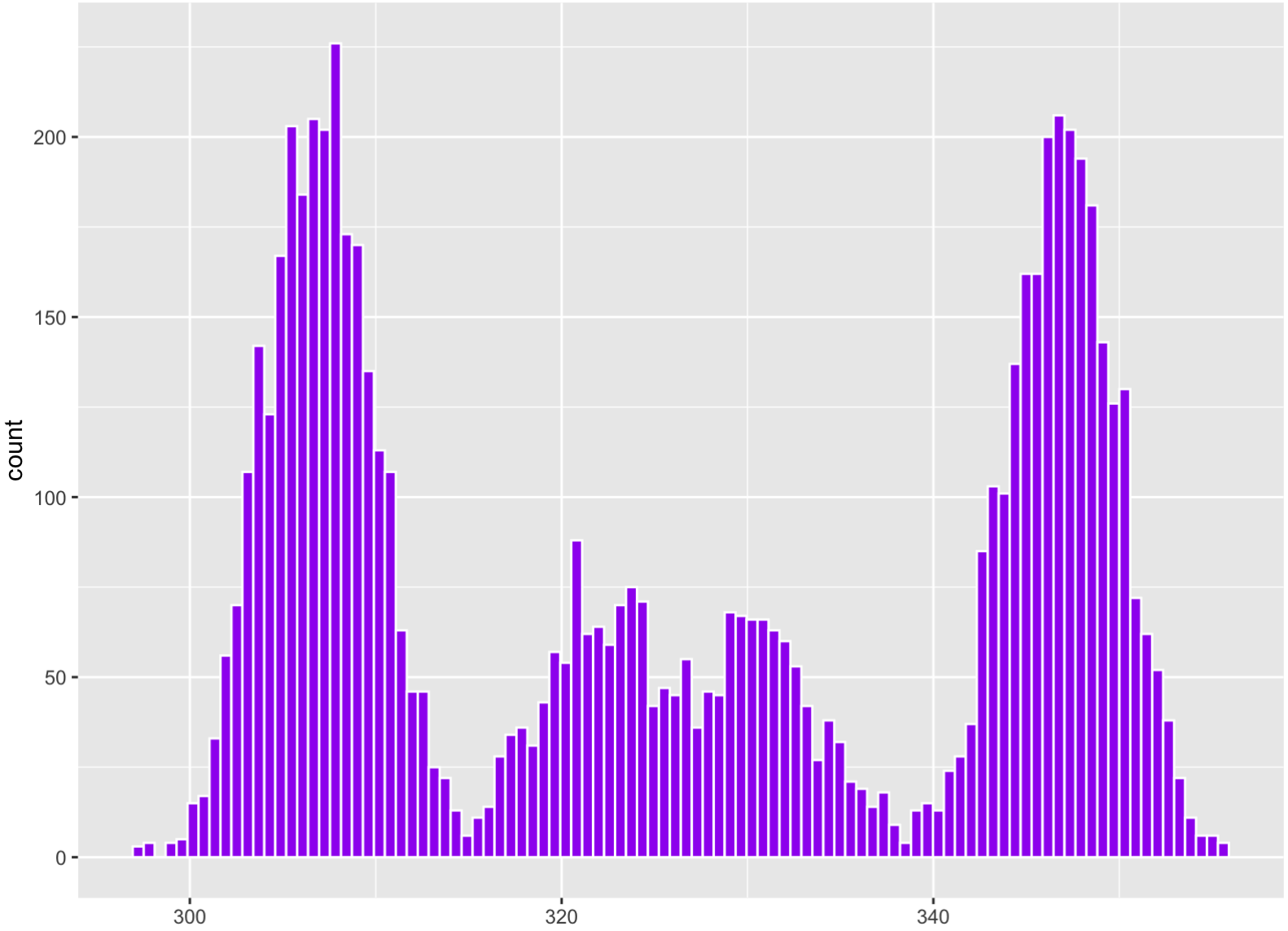
Here there are two distributions, usually not Normals, one for the unexpressed genes (f_u) and one for the expressed genes f_e. An ideal situation is when the histogram is bimodal.

Example RforProteiomics

More than two components

So far we have looked at mixtures of two components. We can extend our description to cases where there may be more. For instance, when weighing N=7,000 nucleotides obtained from mixtures of Deoxyribonucleotide Monophosphates (each type has a different weight, measured with the same standard deviation sd=3), we might observe the histogram such as Figure @ref(fig:nucleotideweights) generated by the code:

```
mA=331;mC=307;mG=347;mT=322; sd=3;
p_C=0.38; p_G=0.36; p_A=0.12; p_T=0.14
pvec=c(p_A,p_C,p_G,p_T); N=7000
nuclt=sample(4,N,replace=TRUE,prob=pvec)
quadwts = rnorm(length(nuclt),
          mean = c(mA, mC, mG, mT)[nuclt],
          sd   = sd)
ggplot(data.frame(quadwts),aes(x=quadwts))+
          geom_histogram(bins=100,col="white",fill="purple") +xlab("")
```



#Special boundary case: n components: the bootstrap

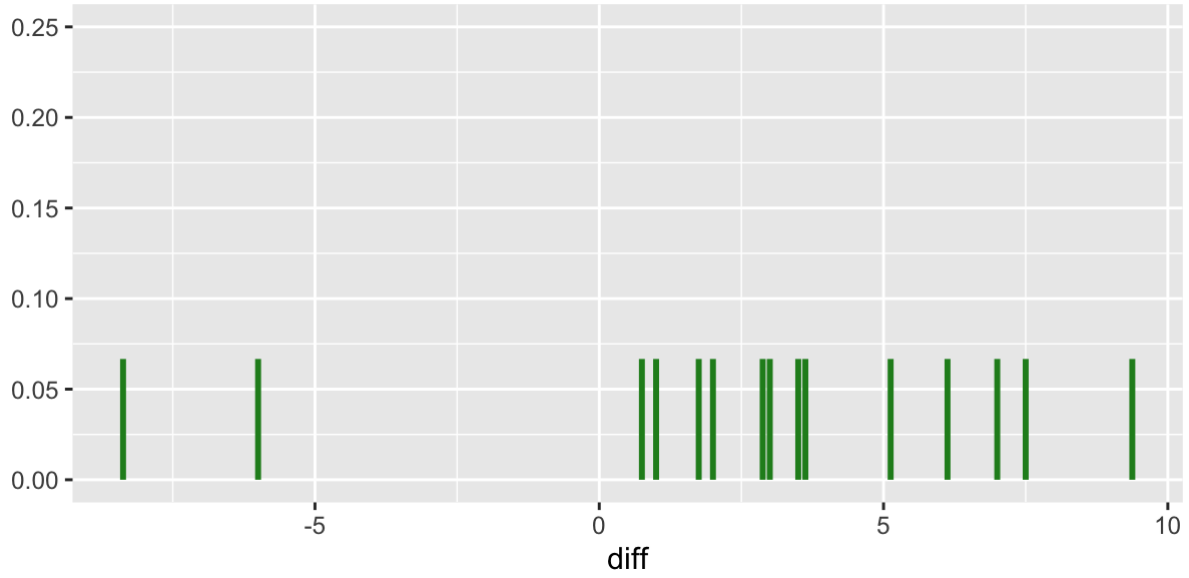
Empirical Distributions and the nonparametric bootstrap

Given a set of measurements, for instance the differences in heights of 15 pairs (15 self hybridized and 15 crossed) of *Zea Mays* plants

```
library("HistData")
ZeaMays$diff
```

```
##  [1]  6.125 -8.375  1.000  2.000  0.750  2.875  3.500  5.125  1.750  3.625
## [11]  7.000  3.000  9.375  7.500 -6.000
```

```
ggplot(data.frame(ZeaMays,y=1/15),
       aes(x=diff, ymax=1/15, ymin=0)) +
  geom_linerange(size=1, col= "forestgreen") +ylim(0,0.25)
```



In Section 3.5 we saw the empirical cumulative distribution function for a sample of size n was written

$$\hat{F}_n(x) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{x \leq x_i}$$

and various ECDF plots were shown in Chapter 3. As we mentioned the empirical cumulative distribution can be easier to understand than the empirical mass function tied to a finite sample:

$$\hat{f}_n(x) = \frac{1}{n} \sum_{i=1}^{n} \delta_{x_i}(x)$$

But we can see now that the sample data can be considered a mixture of at the observed values $x_1, x_2, \ldots, x_n$ as show in Figure @ref(fig:ecdfZea).

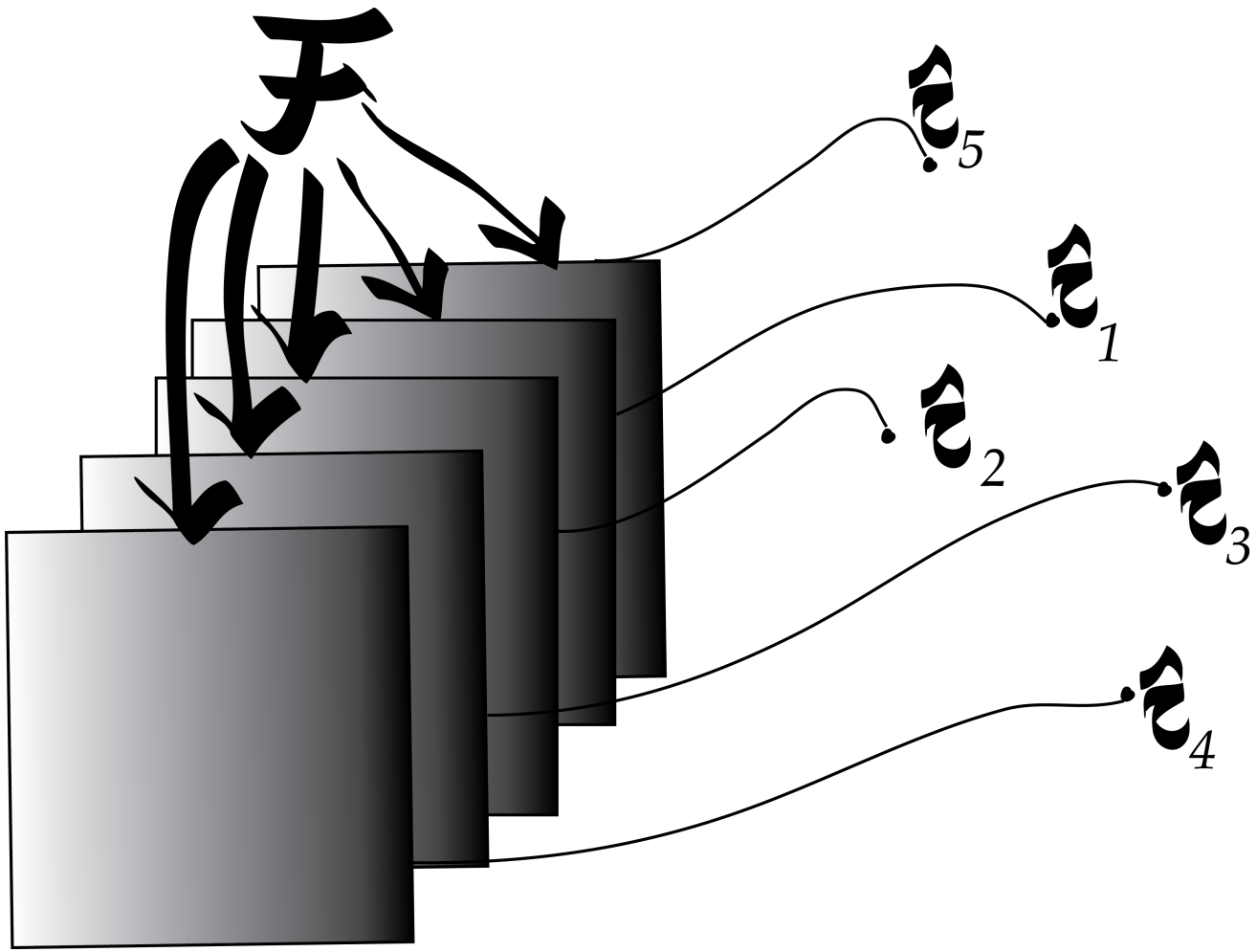


fig:bootpple

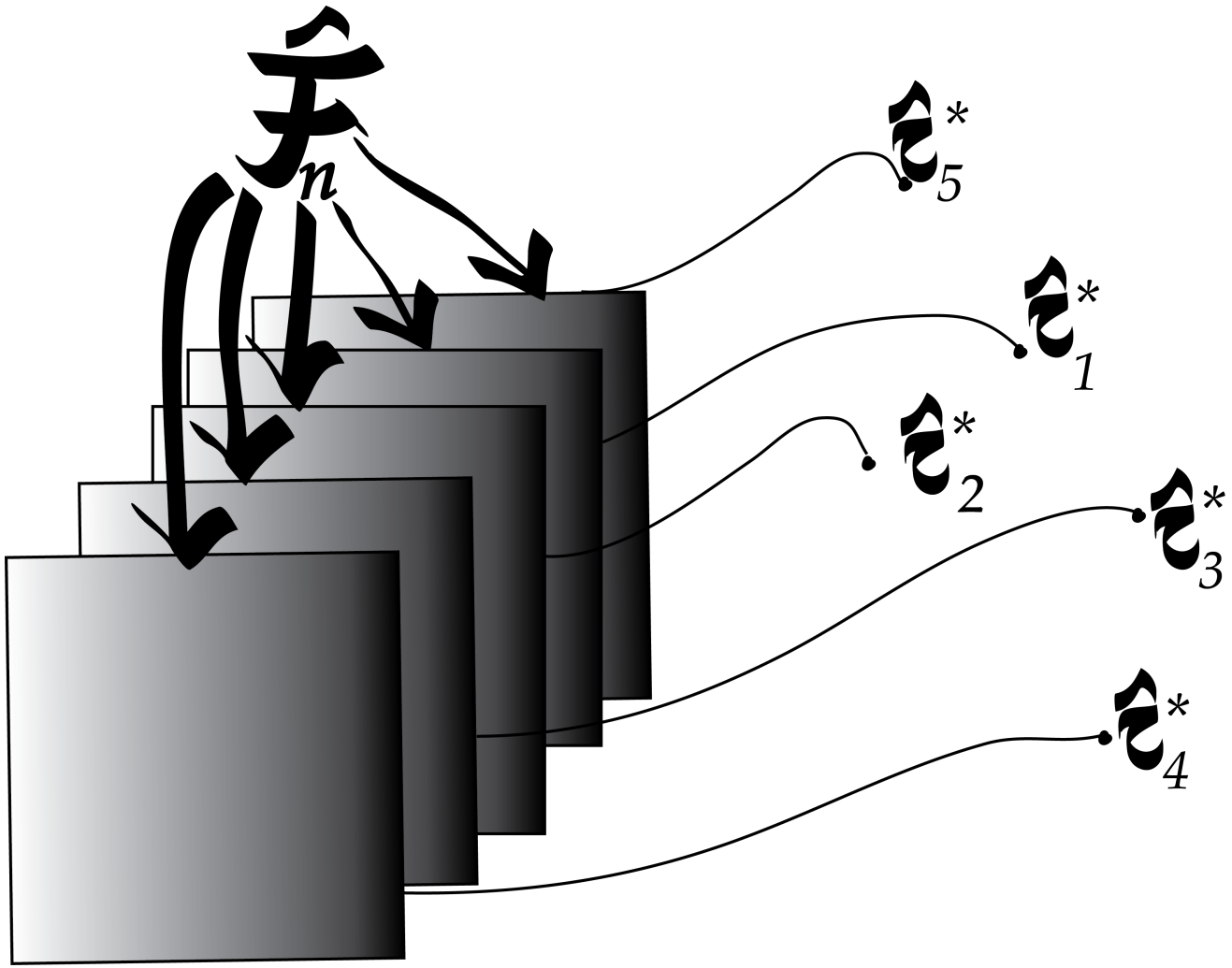
A statistic such as the mean, minimum or median of a sample can be written as a function of the empirical distribution $\bar{x} = mean(\hat{F}_n)$, and for n an odd number, $median = x_{(\frac{n+1}{2})}$.

The true **sampling distribution** of a statistic $\hat{\tau}$ is often hard to know as it requires many different data samples from which to compute the statistic; this is shown in the Figure above.

The **bootstrap** principle approximates the true sampling distribution of $\hat{\tau}$ by creating new samples drawn from the empirical distribution built from the original sample.

We reuse the data as a mixture to create several plausible data sets by taking subsamples and looking at the different statistics $\hat{\tau}^*$ that we compute from the resamples. This is called the **nonparametric bootstrap** resampling approach, see Efron and Tibshirani's 1994 book for a complete reference.

It is a convenient method that generates a simulated sampling distribution for any statistic whose variation we would like to study (we will see several examples of this method, in particular in clustering in Chapter 5).

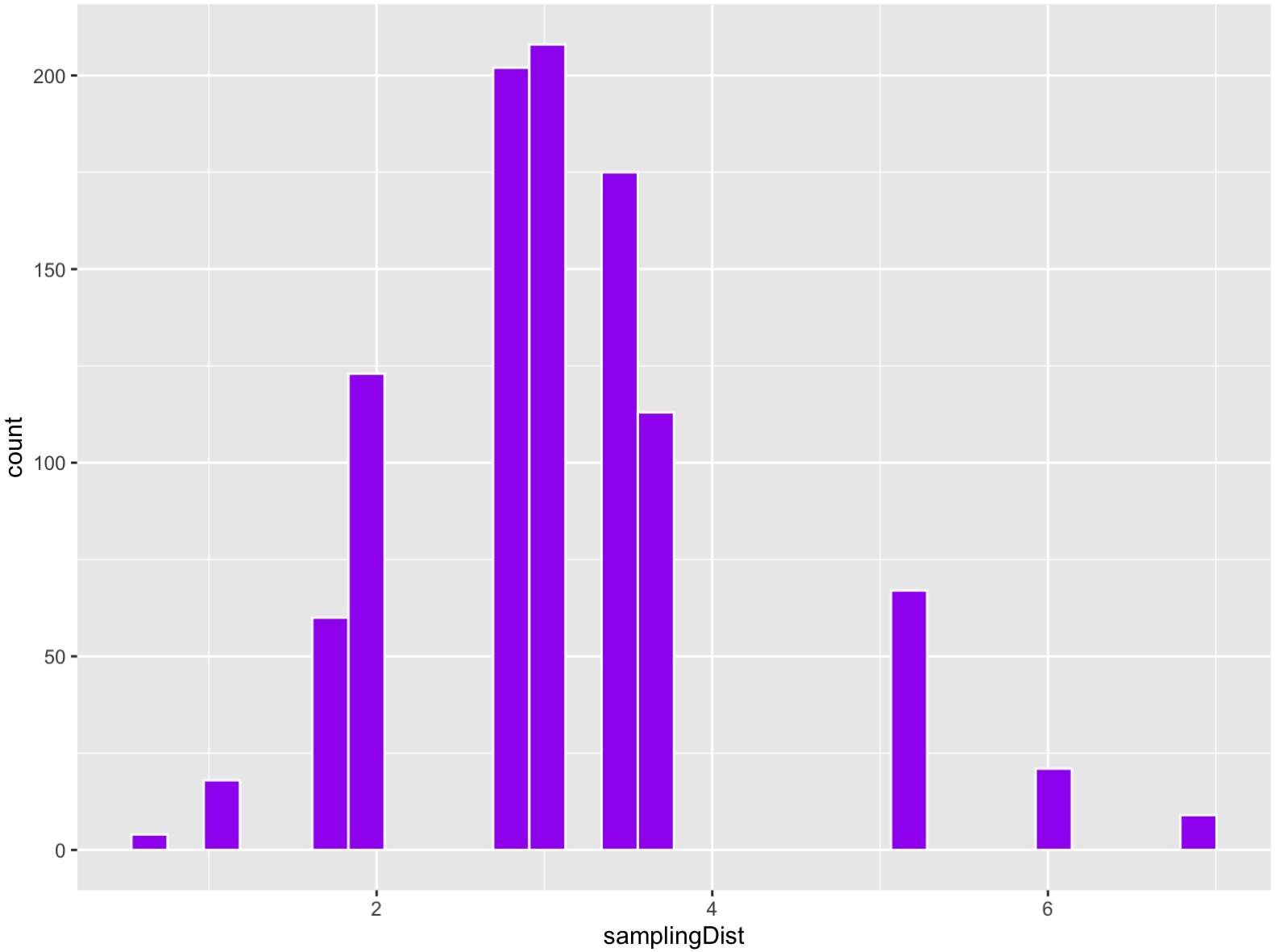


Bootstrap Principle

Let's make a 95% confidence interval for the median of the Zea Mays differences show in Figure @ref(fig:ecdfZ). We use similar simulations to those in the previous sections: Draw $B = 10,000$ samples of size 15 from the 15 values (each their own little component in the 15 part mixture). Then compute the 10,000 medians of each

of these sets of 15 values and look at their distribution: this is called the **sampling distribution** of the median.

```
B = 1000
diff = ZeaMays$diff
samplesB = replicate(B,sample(15,15,replace=T))
samplingDist = apply(samplesB,2,function(x){return(median(diff[x]))})
ggplot(data.frame(samplingDist),aes(x=samplingDist))+
geom_histogram(bins=30,col="white",fill="purple")
```



Why nonparametric?

(Despite their name, nonparametric methods are not methods that do not use parameters, all statistical methods estimate unknown quantities.)

In theoretical statistics, nonparametric methods are those that have infinitely many degrees of freedom or parameters. In practice, we do not wait for infinity; when the number of parameters becomes as large or larger than the amount of data available, we say the method is nonparametric.

The bootstrap uses a mixture with n components, so with a sample of size n, it qualifies as a nonparametric method.

Infinite Mixtures

Sometimes mixtures can be useful without us having to find who came from which distribution, this is especially the case when we have (almost) as many different distributions as observations, let's look at a case where the total distribution can still be studied, even if we don't know where each member came from.

We decomposed the mixture model into a two step process, we extend the description to cases where there could be many more components in the model, 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 $(\mu_1 = 1, \sigma^2 = 0.1)$ and the other half $(\mu_1 = 2, \sigma^2 = 0.1)$ this would actually be equivalent to our original mixture.

However 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 because it is a two step process where one step has to be done before the next: generate the numbers on the `parameter balls', draw a ball, then draw a random number according to that parametric distribution.

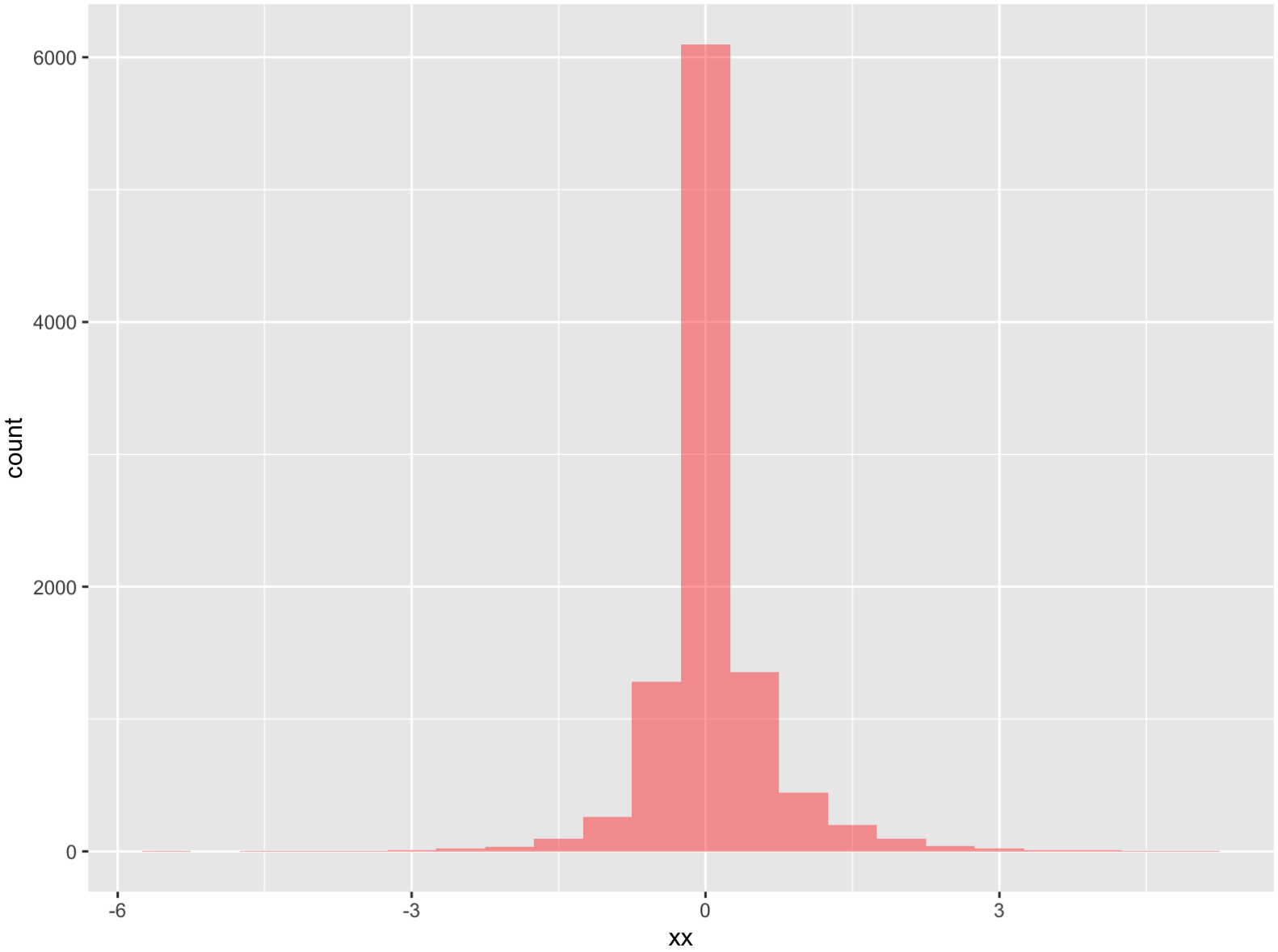
By using this generating mixture we can go as far as generating a new parameter for all the picks from our distribution.

Infinite Mixture of Normals

Laplace

```
theta=-0.1
mu=0.15
sigma=0.43
#Choose some Wis
W=rexp(10000,1)
#Choose some Normals
output=rnorm(10000,theta+mu*W,sigma*W)

dat = data.frame(xx=output)
# hist(output,30)
ggplot(dat,aes(x=xx)) +
    geom_histogram(data=subset(dat),fill = "red", alpha = 0.4,binwidth = 0.5)
```



If we generate observations at two levels:

- **Level 1**

we create a sample of w's from an exponential distribution.

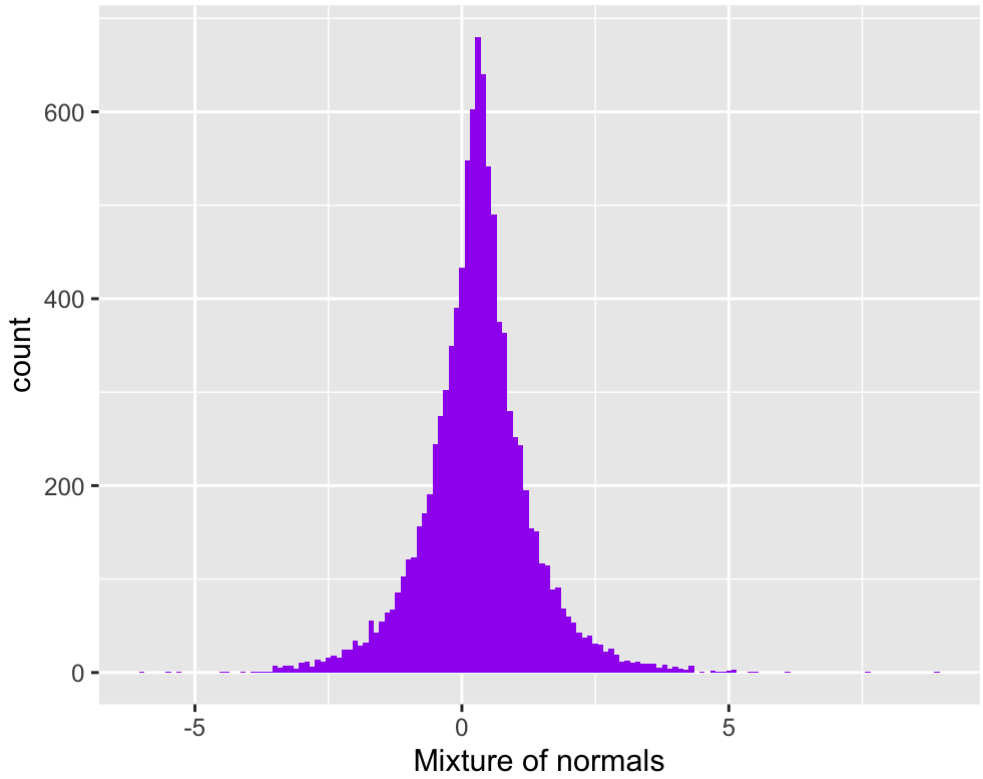
```
w = rexp(10000, rate = 1)
mean(w)
```

```
## [1] 1.001634
```

- **Level 2**

  The ws serve as the (random) variances of normal variables with mean μ generated using `rnorm`.

```
mu    = 0.3
lps = rnorm(length(w), mean=mu, sd=sqrt(w))
ggplot(data.frame(lps), aes(x = lps)) + xlab("Mixture of normals") +
  geom_histogram(fill = "purple", binwidth = 0.1)
```



Data sampled from a Laplace distribution – each pick has their own variance. The variances come from an exponential distribution. This is often called a {} of many normals; each pick has their own variance multiplied by a scaling factor.

A Laplace distribution from an infinite mixture of Normals

This is actually a rather useful mixture, it turns out such a mixture has a name and well known properties, it is called the Laplace distribution of errors.

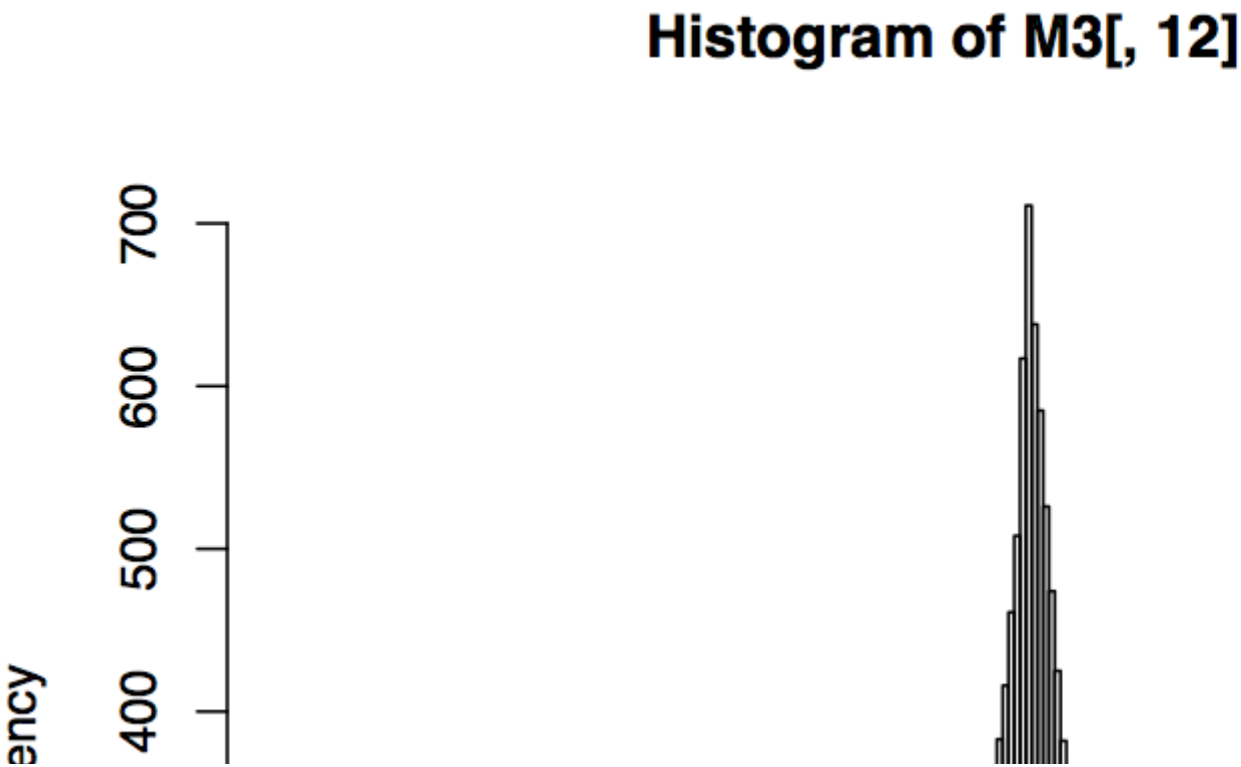
Instead of using the mean and the variance to summarize it, we should use the median and the mean absolute deviation as they have better properties.

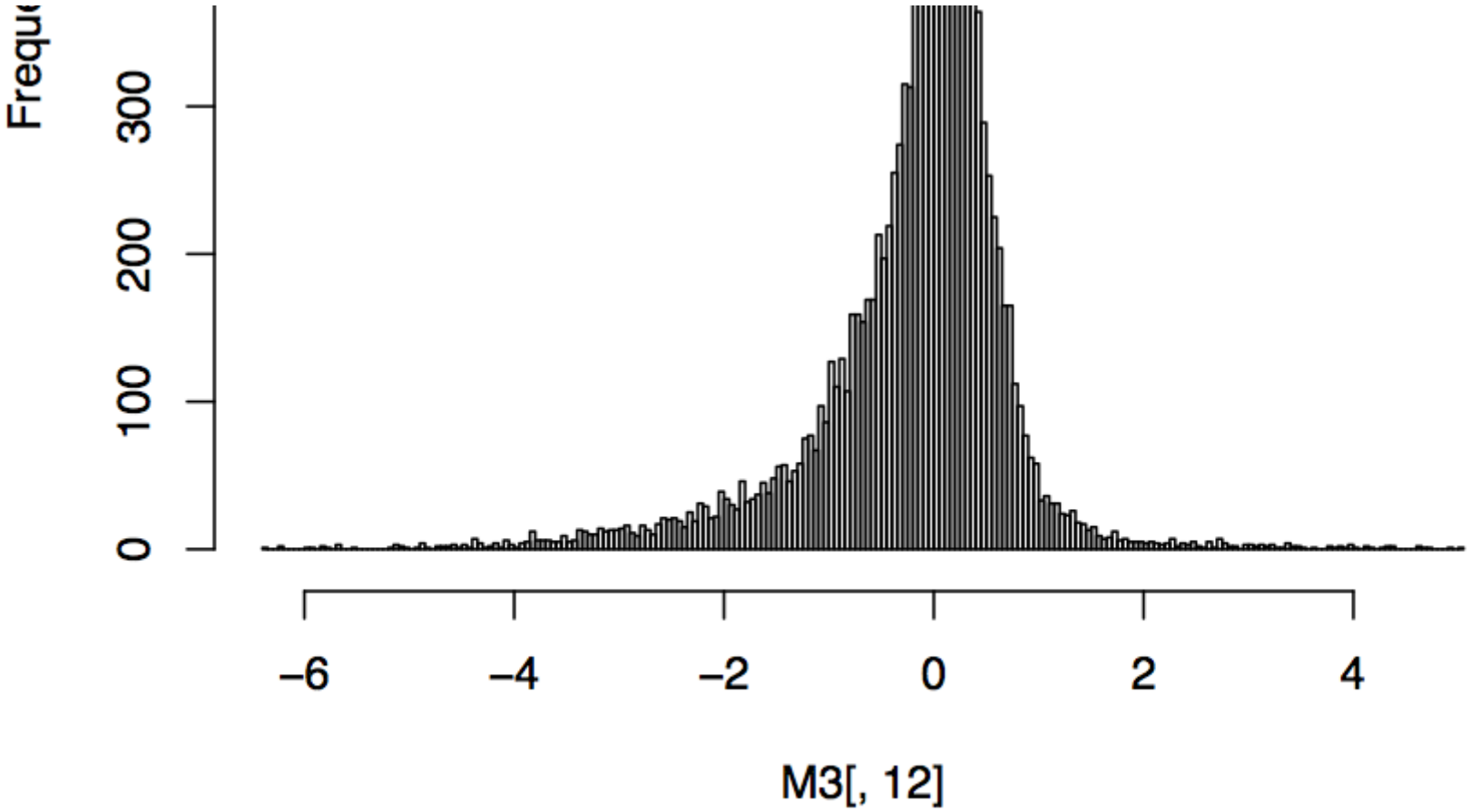
Note:Laplace knew already that the error distribution that has the location parameter the median and the scale parameter the mad has density:

$$f_Y(y) = \frac{1}{2\phi} exp[-|y-\theta|/\phi], \qquad \phi > 0$$

Asymmetric Laplace

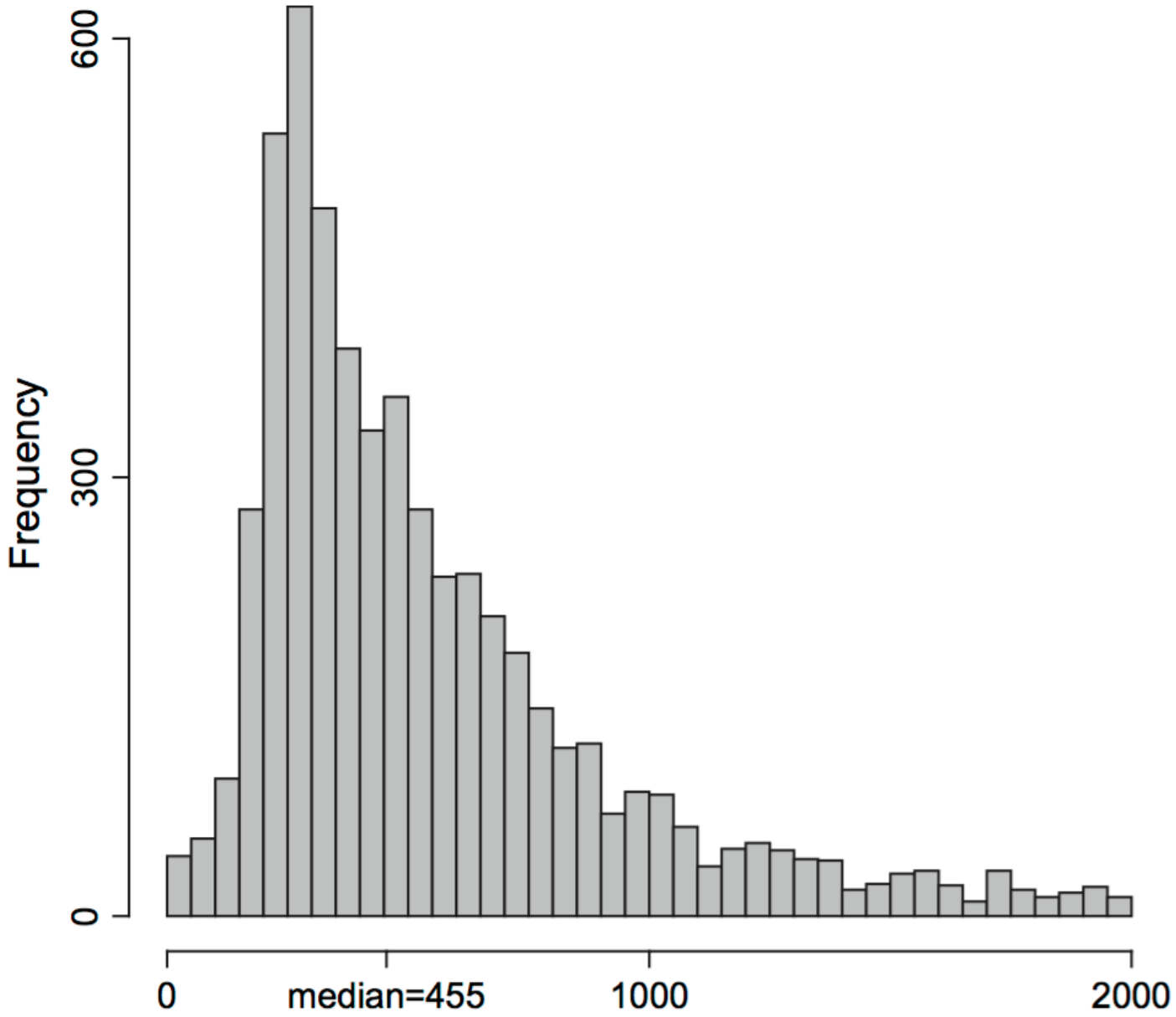
Looking at all the gene expression values from a Microarray, one gets a distribution like this:





Gene Expression for T cells

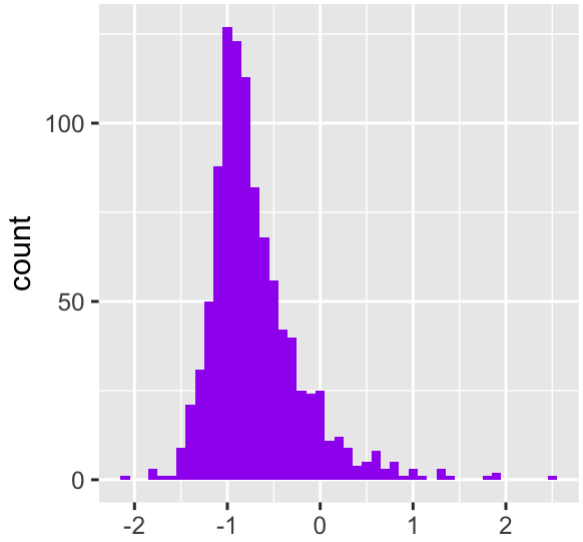
Promoter Lengths



Promoter Lengths

A useful extension to this distribution incorporates this same dependence of the mean on the variance with an extra parameter θ controlling the **location** or center of the distribution. We generate data alps from a hierarchical model with W an exponential variable; the output shown in the Figure below generated as normal $N(\theta + w\mu, \sigma w)$ conditionally on $W = w$ for many w's, randomly generated from the exponential with mean 1 as shown in the code:

```
mu    = 0.3; sigma = 0.43
theta = -1; ws =rexp(B,1)
alps = rnorm(length(ws), theta + mu * w, sigma * sqrt(w))
ggplot(data.frame(alps), aes(x = alps)) + xlab("") +
  geom_histogram(fill = "purple", binwidth = 0.1)
```



Data sampled from an asymmetric Laplace distribution – a scale mixture of many normals whose means and variances are dependent. We write $X \sim \mathcal{AL}(\theta, \mu, \sigma)$

This is a good example where looking at the generative process we can see why the variance and mean are linked.

The expectation and variance of an AL(θ, μ, σ) are

$$E(Y) = \theta + \mu \text{ and } var(Y) = \sigma^2 + \mu^2$$

Note the variance is not independent of the mean unless $\mu = 0$ – the case of the symmetric Laplace Distribution. This is a feature of the distribution that makes it useful.

In practical situations, when looking at gene expression in microarrays, taxa counts in 16sRNA studies, we will see that the variances depend on the mean, we will need models that can acomodate for this problem.

The Gamma–Poisson Mixture Model

Count data are often messier than simple Poisson and Binomial distributions serve as building blocks for more sophisticated models called mixtures.

Three Worlds

What's a Gamma–Poisson mixture model used for?

- Overdispersion (in Ecology)
- Simplest Mixture Model for Counts

- Different evolutionary mutation rates
- Throughout Bioinformatics and Bayesian Statistics
- Abundance data

In ecology, for instance, we might be interested in variations of fish species in all the lakes in a region.

We sample the fish species in each lake to estimate their true abundances, and that could be modeled by a Poisson.

But the true abundances will vary from lake to lake.

The different Poisson rate parameters λ can be modeled as coming from a distribution of rates.

This is a hierarchical model, this type of model will also allow us to add supplementary steps in the hierarchy, for instance we could be interested in many different types of fish, etc...

Gamma Distribution: two parameters (shape and scale)

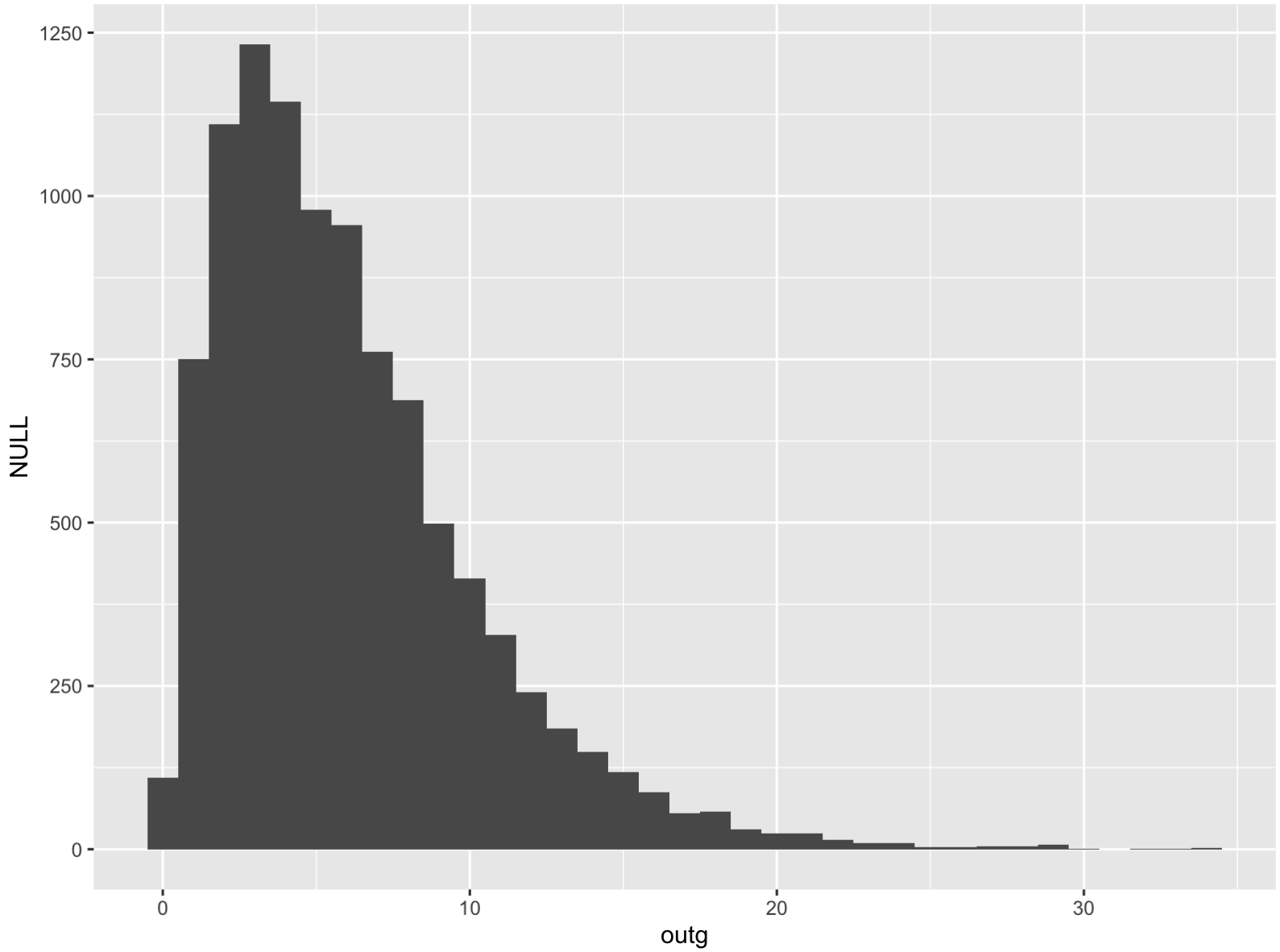
wikigamma Like the Beta distribution, the Gamma distribution is used to model certain continuous variables, however the random variables that have a Gamma distribution can take on any positive values, typical quantities that follow this distribution are waiting times and survival times.

It is often used in Bayesian inference to model the variability of the Poisson or Exponential parameters (conjugate family).

This is not unrelated to why we use it for mixture modeling.

Let's explore it by simulation and examples:

```
require(ggplot2)
nr=10000
set.seed(20130607)
outg=rgamma(nr,shape=2,scale=3)
#
p=qplot(outg,geom="histogram",binwidth=1)
p
```



A histogram of randomly generated Gamma(2,3) generated points.

Note on fitting distributions:

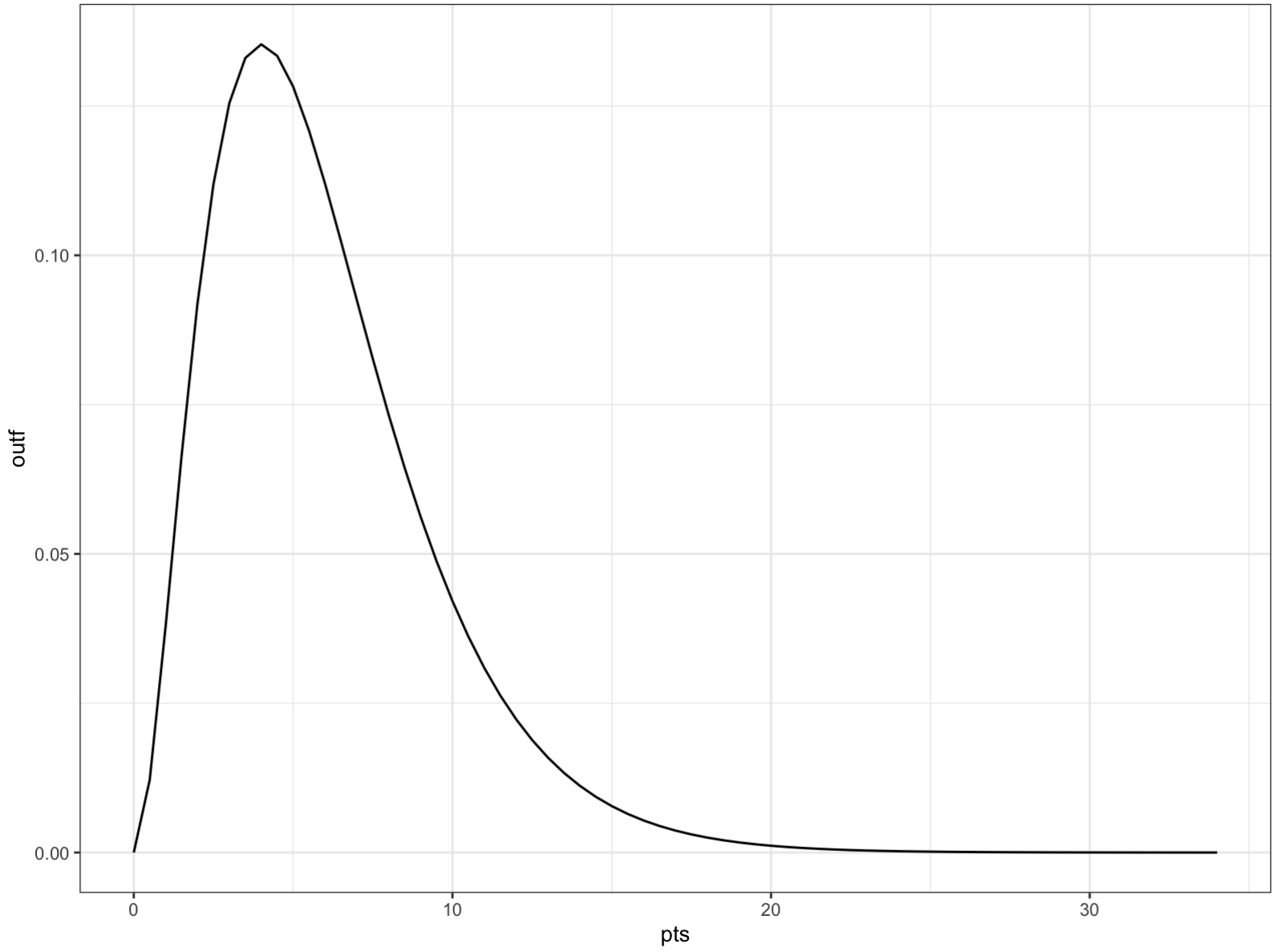
```
require(MASS)
## avoid spurious accuracy
op = options(digits = 3)
set.seed(123)
x = rgamma(100, shape = 5, rate = 0.1)
fitdistr(x, "gamma")
```

```
##    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)
```

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



The theoretical Gamma (2,3) probability density.

We are going to use this type of variability for the variation in our Poisson parameters.

Gamma mixture of Poissons: a hierarchical model

This is a two step process:

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

2. Generate a set of Poisson(λ_i) random variables.

```
ng=90
set.seed(1001015)
lambdas=rgamma(ng,shape=2,scale=3)
####Rate is usually the second it is 1/scale
veco=rep(0,ng)
for (j in (1:ng)){
  veco[j]=rpois(1,lambda=lambdas[j]) }
require(vcd)
goodnb=goodfit(veco,"nbinomial")
```

```
## Warning in densfun(x, parm[1], parm[2], ...): NaNs produced
```

```
goodnb
```

```
##
## Observed and fitted values for nbinomial distribution
## with parameters estimated by `ML'
##
##  count observed fitted pearson residual
##      0       10  7.673           0.8399
##      1        6  9.895          -1.2383
##      2       12 10.191           0.5668
##      3        9  9.613          -0.1977
##      4        9  8.652           0.1184
```

```
##      5         6  7.562          -0.5681
##      6         6  6.479          -0.1881
##      7         6  5.471           0.2264
##      8         6  4.568           0.6698
##      9         3  3.782          -0.4022
##     10         2  3.109          -0.6291
##     11         2  2.542          -0.3397
##     12         2  2.067          -0.0469
##     13         2  1.675           0.2512
##     14         3  1.352           1.4171
##     15         3  1.088           1.8327
##     16         1  0.873           0.1354
##     17         2  0.699          -0.7622
```

`nbinomial` stands for the Negative Binomial and is another distribution for count data. In general it is used to model the number of trials until we obtain a success in a Binomial (p) experiment.

`rnegbin`, `dnegbin`,`pnegbin` are the corresponding functions.

Fitting a Negative Binomial with `fitdistr`:

```
set.seed(123)
x4 = rnegbin(500, mu = 5, theta = 4)
fitdistr(x4, "Negative Binomial")
```

```
##     size      mu
##    4.216   4.945
##   (0.504) (0.147)
```

The Mathematical explanation

The Negative Binomial probability distribution function

$$\text{dnbinom}(k, size = r, \text{prob} = p) = \binom{r+k-1}{k} p^r (1-p)^k$$

This can be interpreted as the probability of waiting to have k failures until the rth success occurs. Success having probability p

Does it have a Negative Binomial distribution?

We can compate the theoretical fit of the Negative Binomial with the data using a rootogram.

```
summary(goodnb)
```

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

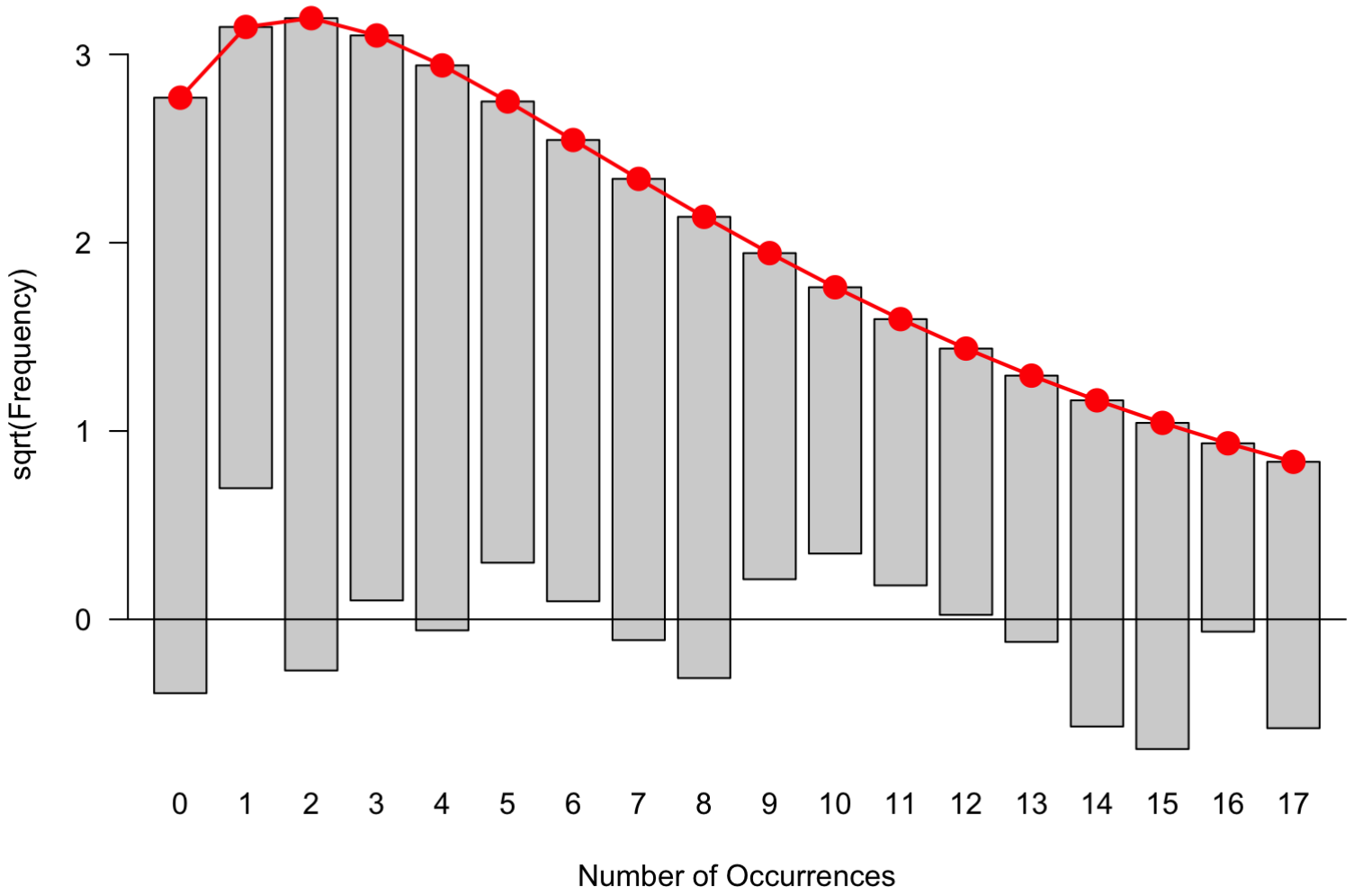
```
goodnb$par
```

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

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

```
table(veco)
```

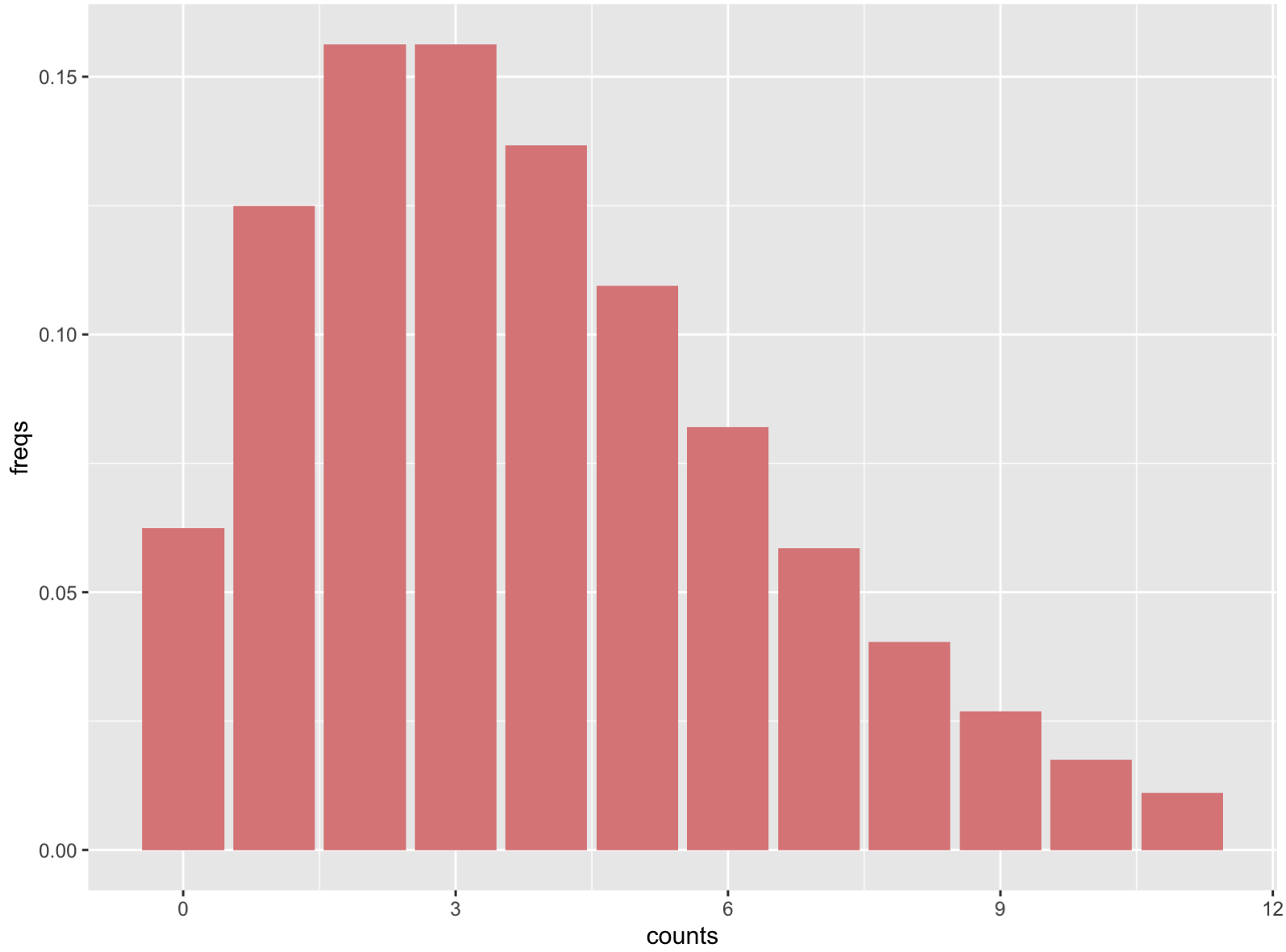
```
## veco
##  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
## 10  6 12  9  9  6  6  6  6  3  2  2  2  2  3  3  1  2
```



##Rootogram showing the theoretical and observed data for NB

```
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")
```



Barplot from a theoretical negative binomial with p=0.5, until 4 successes.

Gamma Mixture of Poissons: the densities

Theoretically taking a mixture of Poisson(μ) variables where $\mu \sim Gamma(\alpha = k, \beta)$.

The final distribution is the result of a two step process: \ - 1. Generate a Gamma (α, β) distributed number, call it z from density

$$dgamma(z, \alpha, \beta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} z^{\alpha-1} e^{-\beta z}$$

\ - 2. Generate a number from the Poisson(z) distribution with parameter z, call it y.

$$dpois(y, \lambda = z) = \frac{z^{y} e^{-z}}{y!}$$

If z only took on integer numbers from 0 to 10 then we would write

$$P(Y = y) = P(Y = y|z = 0)P(z = 0) + P(Y = y|z = 1)P(z = 1)$$

$$\ldots + P(Y = y|z = 10)P(z = 10)$$

It's not quite that simple and we have to write it out as an integral sum instead of a discrete sum.

Gamma-Poisson is Negative Binomial

We call the distribution of Y the marginal and it is given by

$$P(Y=y) = \int dgamma(z,a,b)dpois(y,z)dz = \int \frac{b^a}{\Gamma(a)} z^{a-1} e^{-bz} \frac{z^y e^{-z}}{y!} dz$$

Remembering that $\Gamma(a) = (a-1)!$

$$P(Y=y) = \frac{b^a}{(a-1)!y!} \int z^{y+a-1} e^{-z(b+1)} dz$$

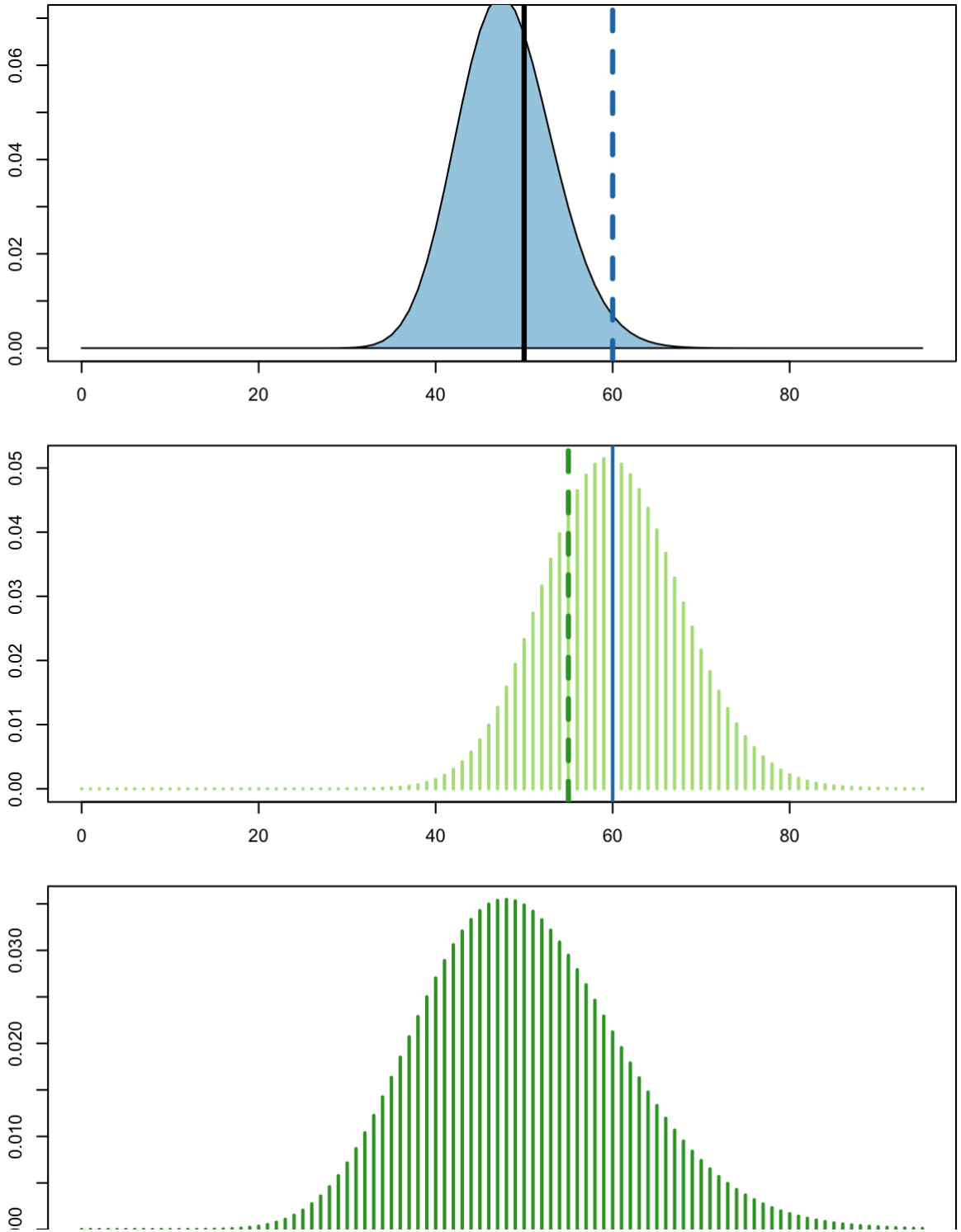
Now we use that the integral

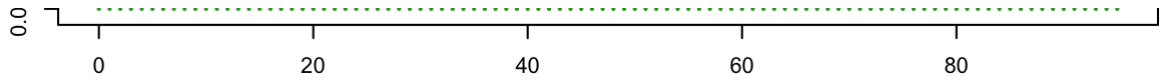
$$\int z^{r-1} e^{-wz} dz = \frac{\Gamma(r)}{w^r}$$

so

$$P(Y=y) = \frac{(y+a-1)!}{(a-1)!y!} \frac{b^a}{(b+1)^a(b+1)^y} = \binom{y+a-1}{y} (\frac{b}{b+1})^a (1-\frac{b}{b+1})^y$$

giving the negative binomial with size parameter a and probability of success $\frac{b}{b+1}$.


0.00
0.02
0.04
0.06
0
20
40
60
80
0.00
0.01
0.02
0.03
0.04
0.05
0
20
40
60
80
0.010
0.020
0.030



Visualization of the hierarchical model that generates the Gamma-Poisson distribution.

The top panel shows the density of a Gamma distribution with mean 50 (vertical black line) and variance 30. Assume that in one particular experimental replicate, the value 60 is realized. This is our latent variable. The observable outcome is distributed according to the Poisson distribution with that rate parameter, shown in the middle panel. In one particular experiment the outcome may be, say, 55, indicated by the dashed green line. Overall, if we repeat these two subsequent random process many times, the outcomes will be distributed as shown in the bottom panel the Gamma-Poisson distribution.

Read Noise Modeling

Negative Binomial :hierarchical mixture for reads

In biological contexts such as RNA-seq and microbial count data the negative binomial distribution arises as a hierarchical mixture of Poisson distributions. This is due to the fact that if we had technical replicates with the same read counts, we would see Poisson variation with a given mean. However, the variation among biological replicates and library size differences both introduce additional sources of variability.

To address this, we take the means of the Poisson variables to be random variables themselves having a Gamma distribution with (hyper)parameters shape r and scale $p/(1-p)$. We first generate a random mean, λ, for the Poisson from the Gamma, and then a random variable, k, from the Poisson(λ). The marginal distribution is:

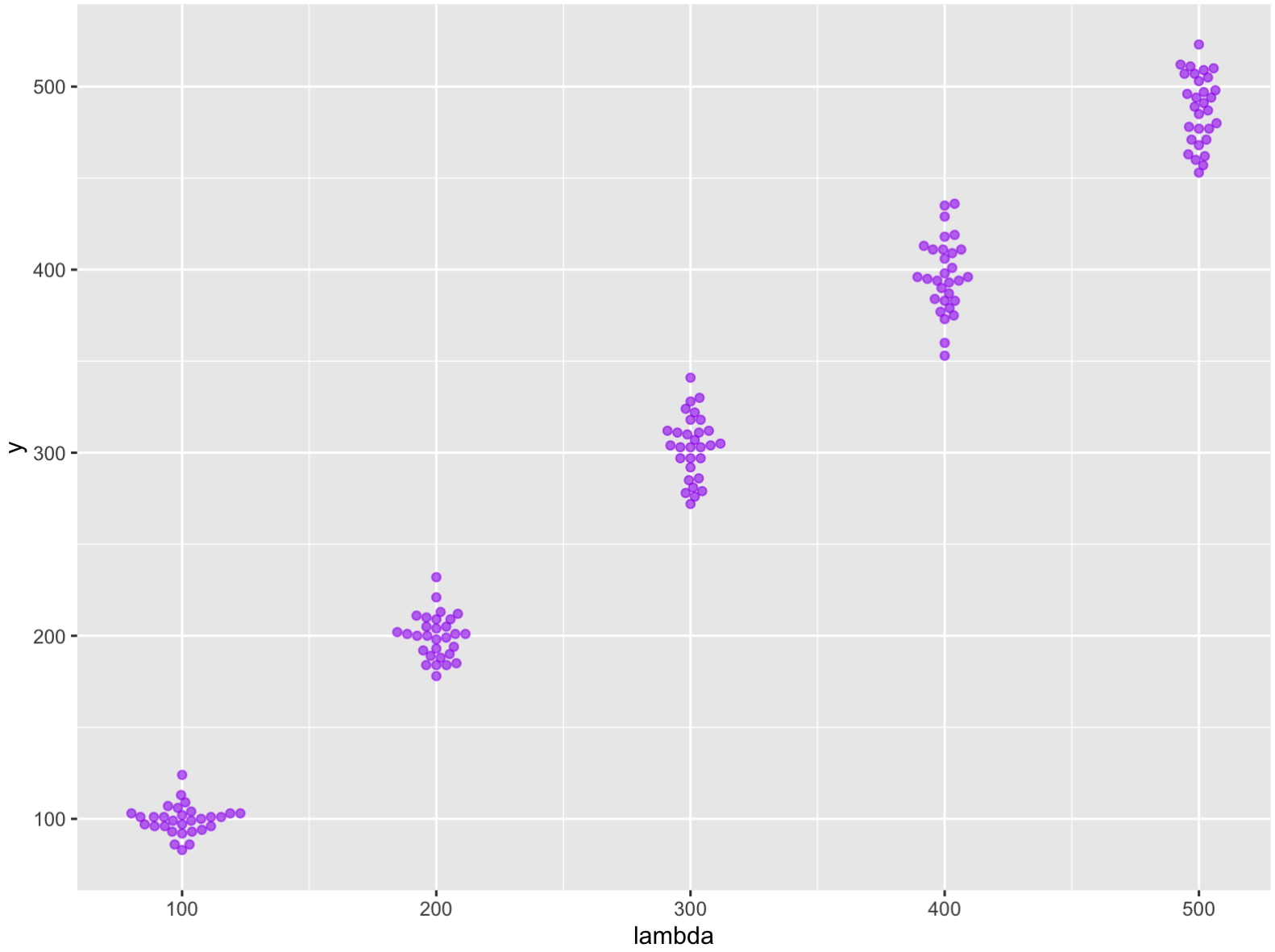
$$
\begin{aligned}
P(X=k) &= \int_0^\infty Po_\lambda(k) \times \gamma_{(r,\frac{p}{1-p})} d\lambda \\
&= \int_0^\infty \frac{\lambda^k}{k!} e^{-\lambda} \times \frac{\lambda^{r-1} e^{-\lambda\frac{1-p}{p}}}{(\frac{p}{1-p})^r \Gamma(r)} d\lambda \\
&= \frac{(1-p)^r}{p^r k! \Gamma(r)} \int_0^\infty \lambda^{r+k-1} e^{-\lambda/p} d\lambda \\
&= \frac{(1-p)^r}{p^r k! \Gamma(r)} p^{r+k} \Gamma(r+k) \\
&= \frac{\Gamma(r+k)}{k! \Gamma(r)} p^k (1-p)^r
\end{aligned}
$$

Variance Stabilization

Take for instance different Poisson variables with mean μ_i. Their variances are all different if the μ_i are different.

However, if the square root transformation is applied to each of the variables, then the transformed variables will have approximately constant variance.

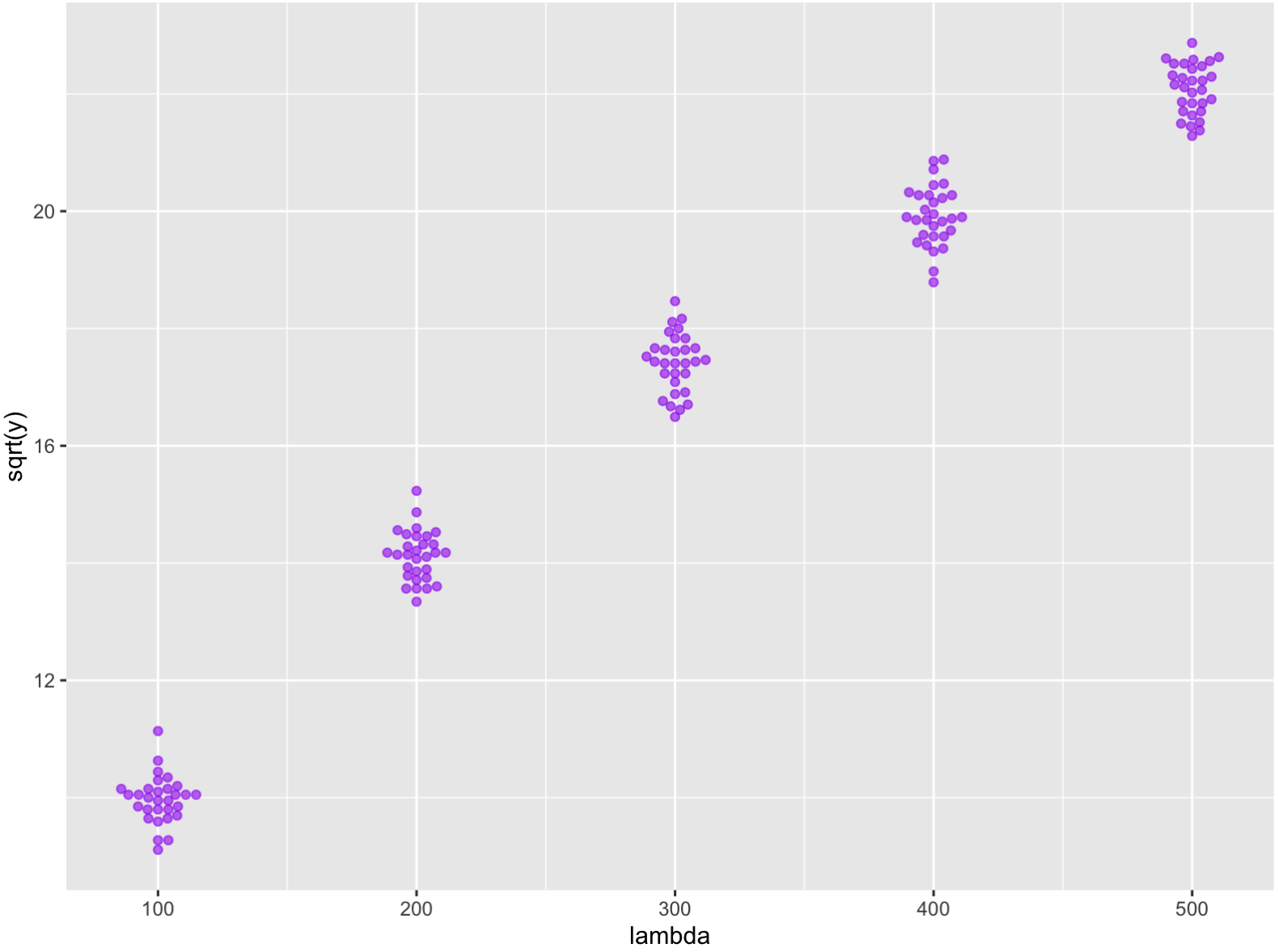
```
library("dplyr")
lambdas = seq(100, 500, by = 100)
simdat = lapply(lambdas, function(l)
  tibble(y = rpois( n = 30, lambda=l),
             lambda = l)) %>% bind_rows
library("ggbeeswarm")
ggplot(simdat, aes( x=lambda, y=y)) +
  geom_beeswarm(alpha = 0.6, color="purple")
```



Series of Poisson

We have equalized the variances at the different levels by taking a square root transformation of the y variable.

```r
ggplot(simdat, aes( x=lambda, y=sqrt(y))) +
  geom_beeswarm(alpha = 0.6, color="purple")
```



After transformation

More generally, choosing a transformation that makes the variance constant is done by using a Taylor series expansion, called the delta method. We will not give the complete development of variance stabilization in the context of mixtures but point the interested reader to the standard texts in Theoretical statistics such as and one of the original articles on variance stabilization.

Anscombe showed that there are several transformations that stabilize the variance of the Negative Binomial depending on the values of the parameters m and r, where $r = \frac{1}{\phi}$, sometimes called the of the Negative Binomial. For large m and constant $m\phi$, the transformation

$$sinh^{-1}\sqrt{(\frac{1}{\phi} - \frac{1}{2})\frac{x + \frac{3}{8}}{\frac{1}{\phi} - \frac{3}{4}}}$$

gives a constant variance around $\frac{1}{4}$. Whereas for m large and $\frac{1}{\phi}$ not substantially increasing, the following simpler transformation is preferable

$$log(x + \frac{1}{2\phi})$$

Modeling read counts

If we have technical replicates with the same number of reads s_j, we expect to see Poisson variation with mean $\mu = s_j u_i$, for each gene or taxa or locus i whose incidence proportion we denote by u_i.

Thus the number of reads for the sample j and taxa i would be

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

We use the notational convention that lower case letters designate fixed or observed values whereas upper case letters designate random variables.

For biological replicates within the same group – such as treatment or control groups or the same environments – the proportions u_i will be variable between samples.

A flexible model that works well for this variability is the Gamma distribution, as it has two parameters and can be adapted to many distributional shapes.

Call the two parameters r_i and $\frac{p_i}{1-p_i}$. So that U_{ij} the proportion of taxa i in sample j is distributed according to Gamma$(r_i, \frac{p_i}{1-p_i})$.

Thus we obtain that the read counts K_{ij} have a Poisson-Gamma mixture of different Poisson variables. As shown above we can use the Negative Binomial with parameters $(m = u_i s_j)$ and ϕ_i as a satisfactory model of the variability.

The counts for the gene/taxa i and sample j in condition c having a Negative Binomial distribution with $m_c = u_{ic}s_j$ and ϕ_{ic} so that the variance is written

$$u_{ic}s_j + \phi_{ic}s_j^2 u_{ic}^2.$$

We can estimate the parameters u_{ic} and ϕ_{ic} from the data.

Random Effects

This application of a hierarchical mixture model is equivalent to the random effects models used in the classical context of analysis of variance.

Applications

Mixtures occur naturally because of heterogeneous data, an experiment may be run by different labs, use differing technologies.

There are often differing binding propensities in different parts of the genome, PCR biases can occur when different operators use different protocols.

The most common problems involve different distributions because both the means *and* the variances are different. This requires variance stabilization to do statistical testing.

Mixture models can often lead us to be able to use data transformations are actually used in what is often known as a *generalized logarithmic* transformation applied in

microarray variance stabilizing transformations and RNA-seq normalization that we will study in depth in chapter 7 and which also proves useful in the normalization of next generation reads in microbial ecology and Chip-SEQ analysis .

Wrapup about Mixture Models

Finite Mixture Models

- Mixture of Normals with different means and variances.
- Mixtures of multivariate Normals with different means and covariance matrices (we'll study next week).
- Decomposing the mixtures using the EM algorithm.

Common Infinite Mixture Models

- Mixtures of Normals (often with a hierarchical model on the variances).
- Beta-Binomial Mixtures (the p in the Binomial is generated according to a Beta(a,b) distribution.
- Gamma-Poisson for read counts.
- Gamma-Exponential for PCR.
- Dirichlet-Multinomial (Birthday problem and the Bayesian setting).