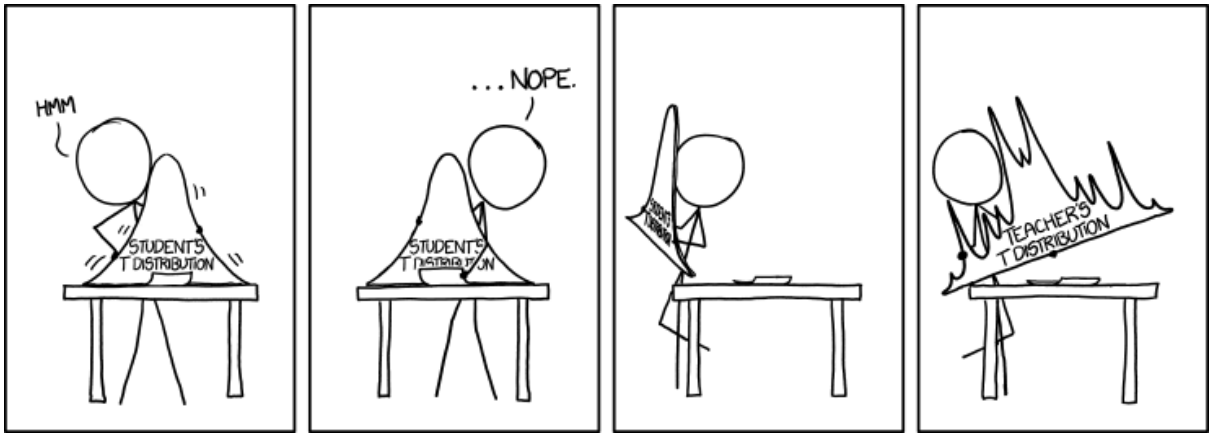
Mixture Models



Susan Holmes (c) June, 2014

Mixture Models and Simulations

Why 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.

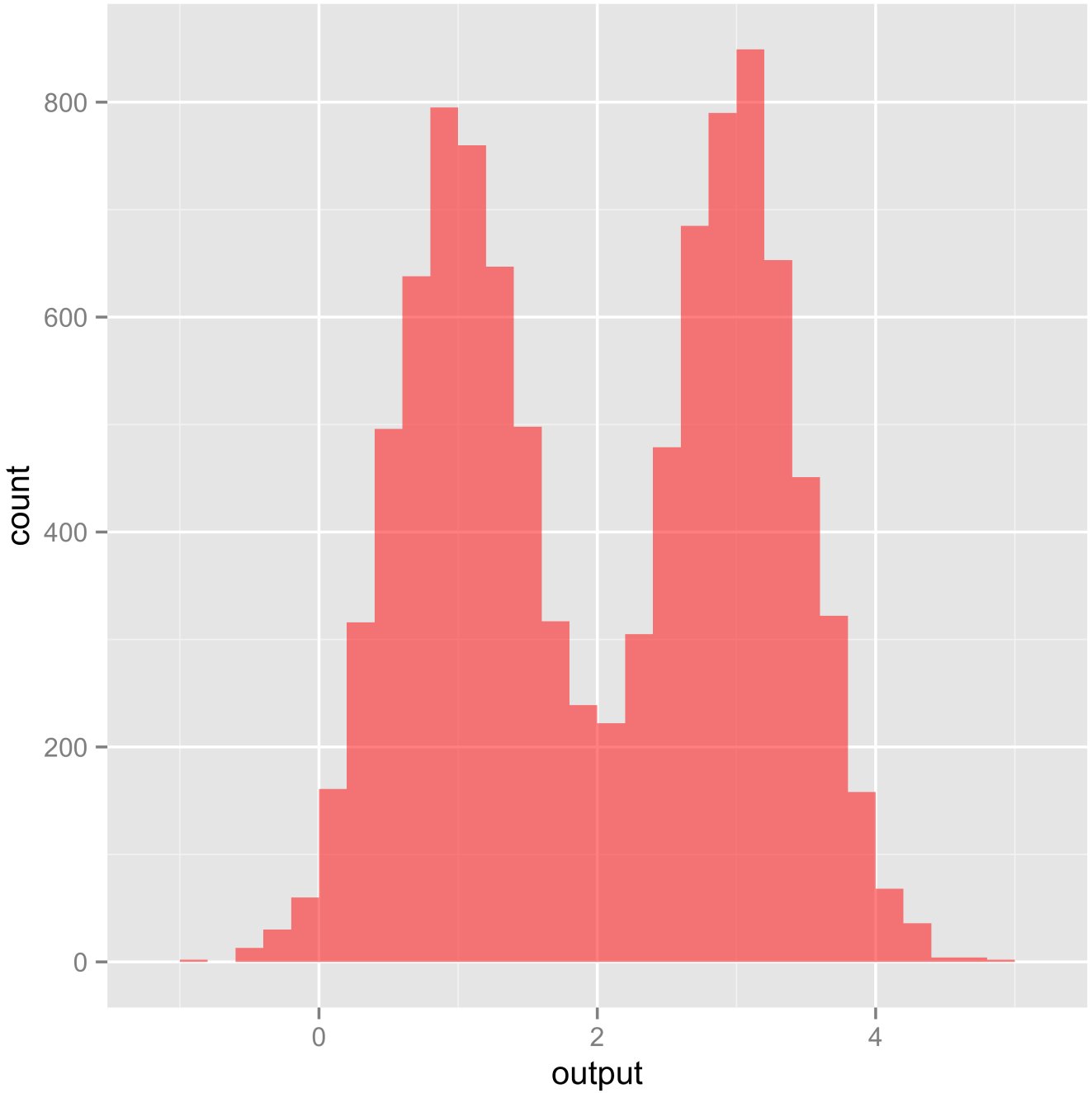
- If it comes up heads
  - *Generate a random number from a Normal with mean 1 and variance 0.25.*
- If it comes up tails
  - *Generate a random number from a Normal with mean 2 and variance 0.25.*

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

```
require(ggplot2)

coinflips=as.numeric(runif(10000)>0.5)

table(coinflips)
```

```
## coinflips

##    0    1

## 4972 5028
```

```
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))
```

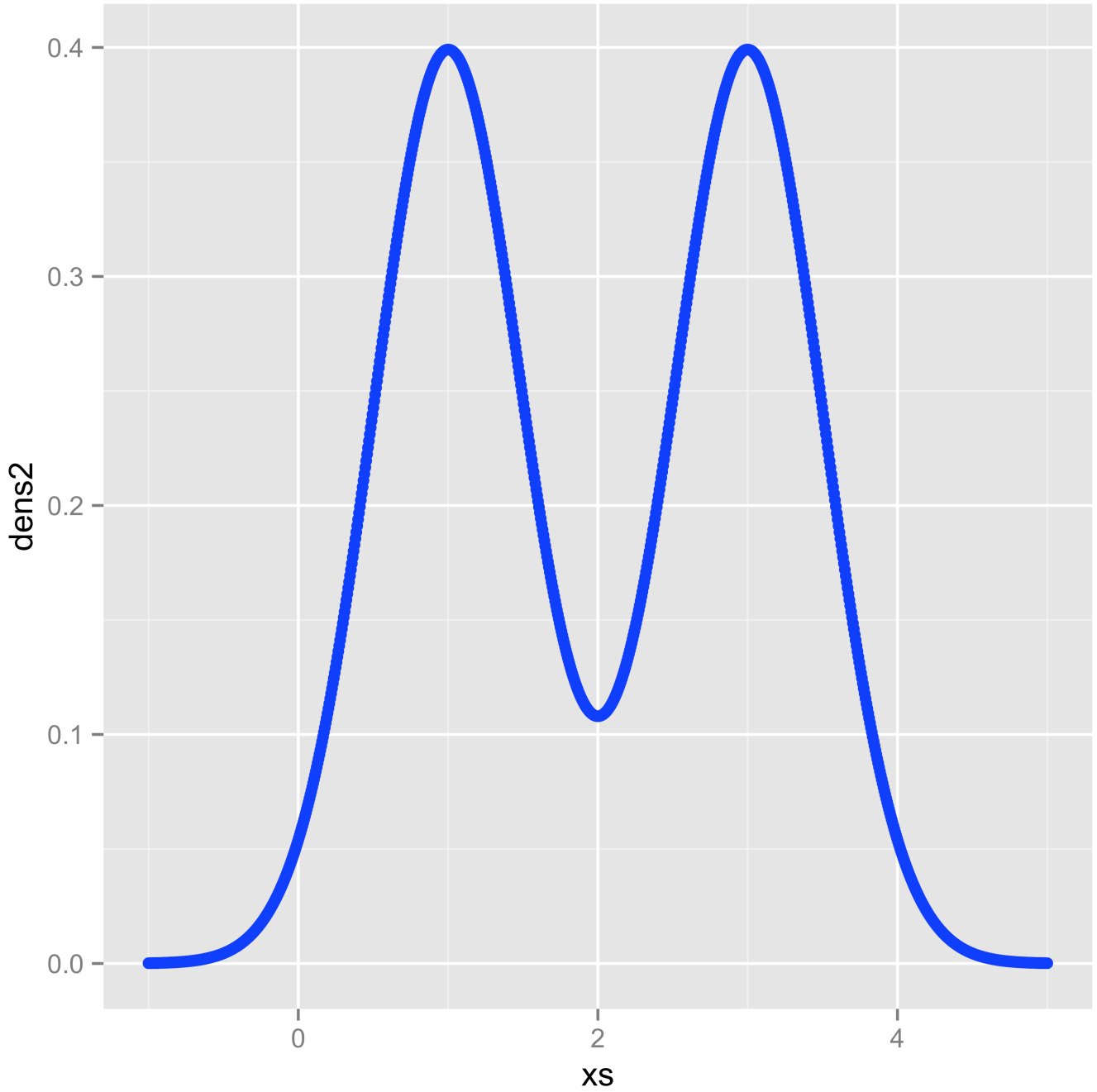


In fact we can write the density (the limiting curve that the histograms tend to look like) as

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

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)
```



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.

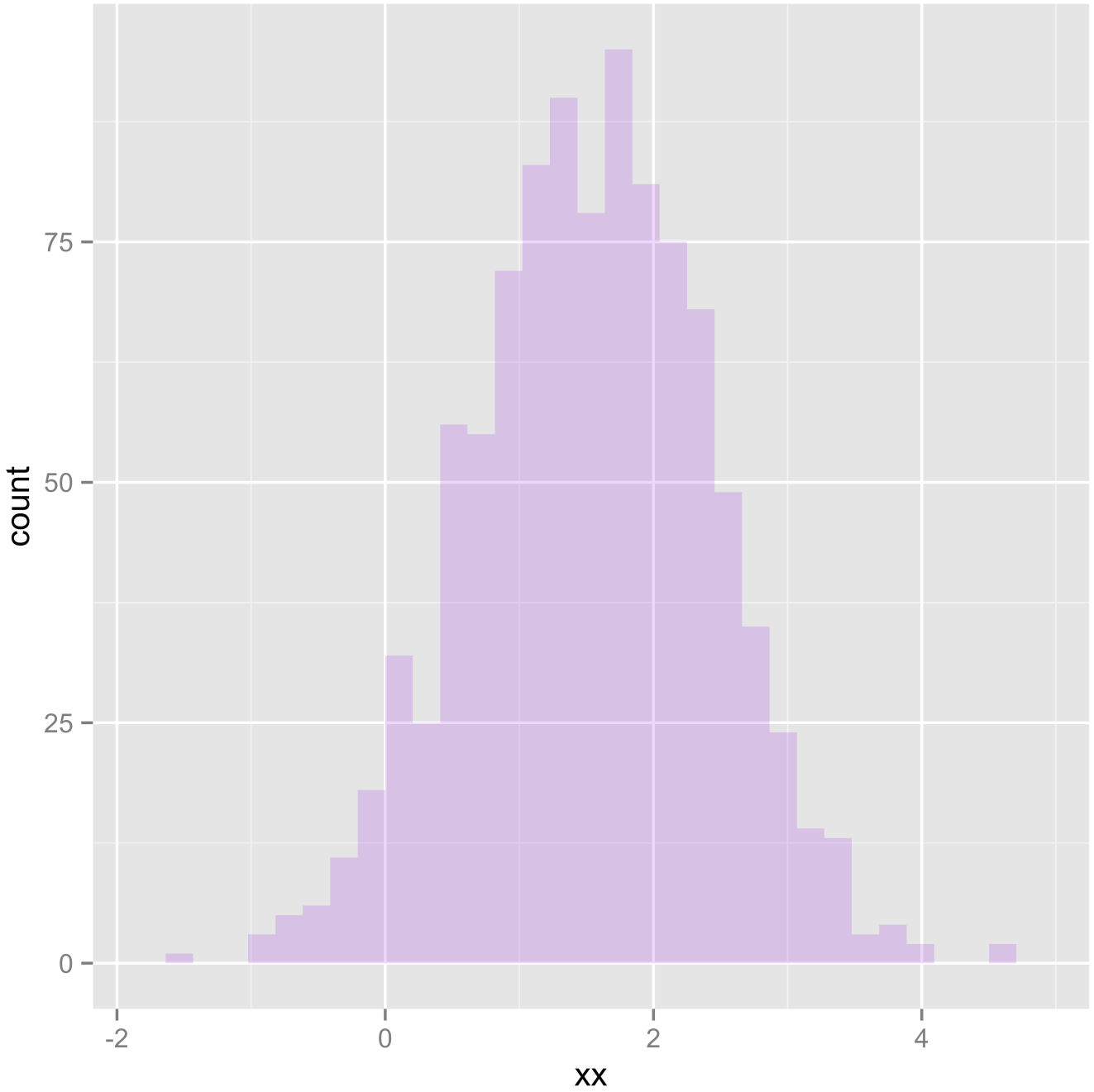
Challenge: Here is a histogram generated by two Normals with the same variances, can you guess the two parameters for these two Normals?

```
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
```

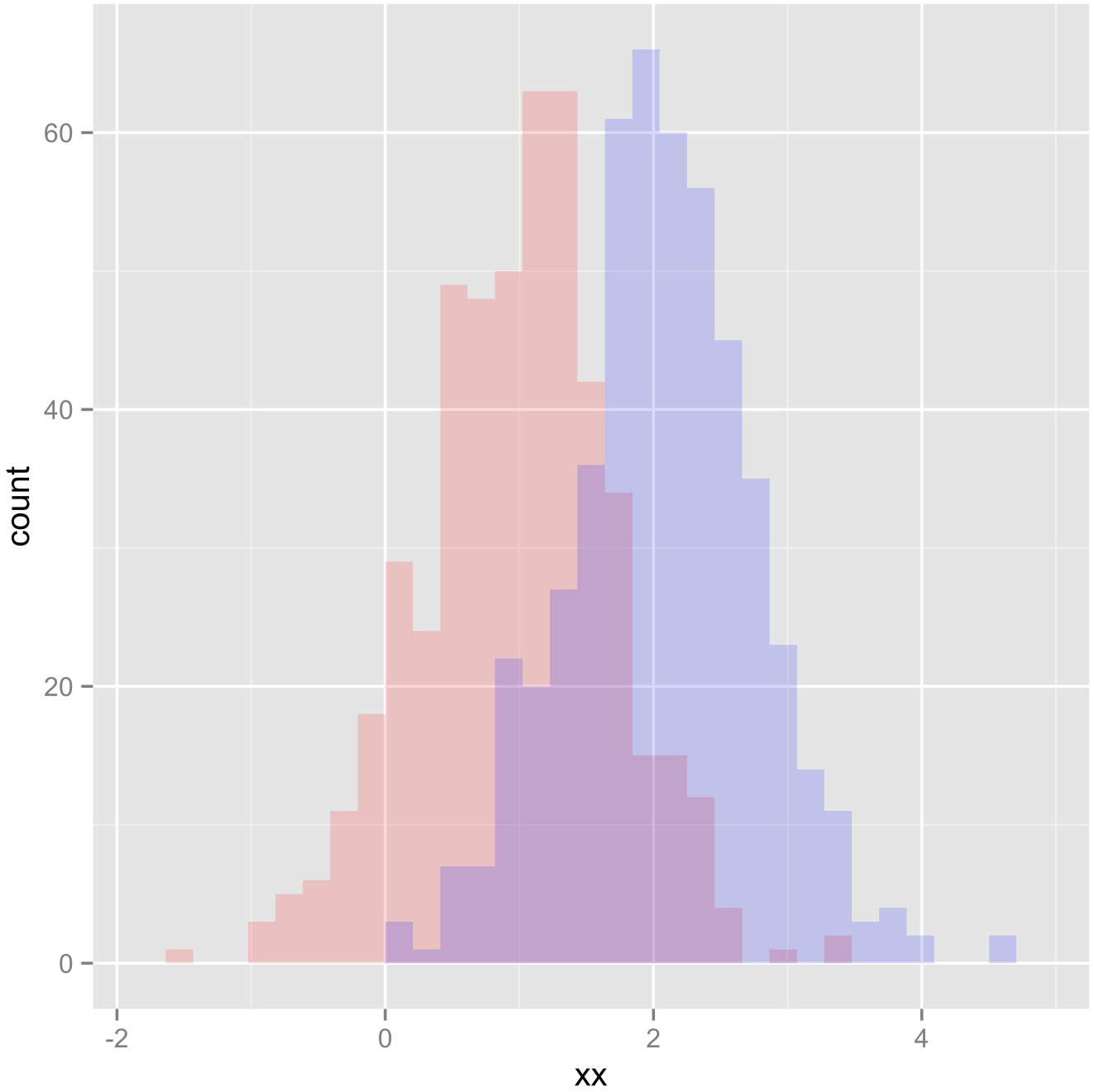
```
dat=data.frame(xx=output,yy=group)

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

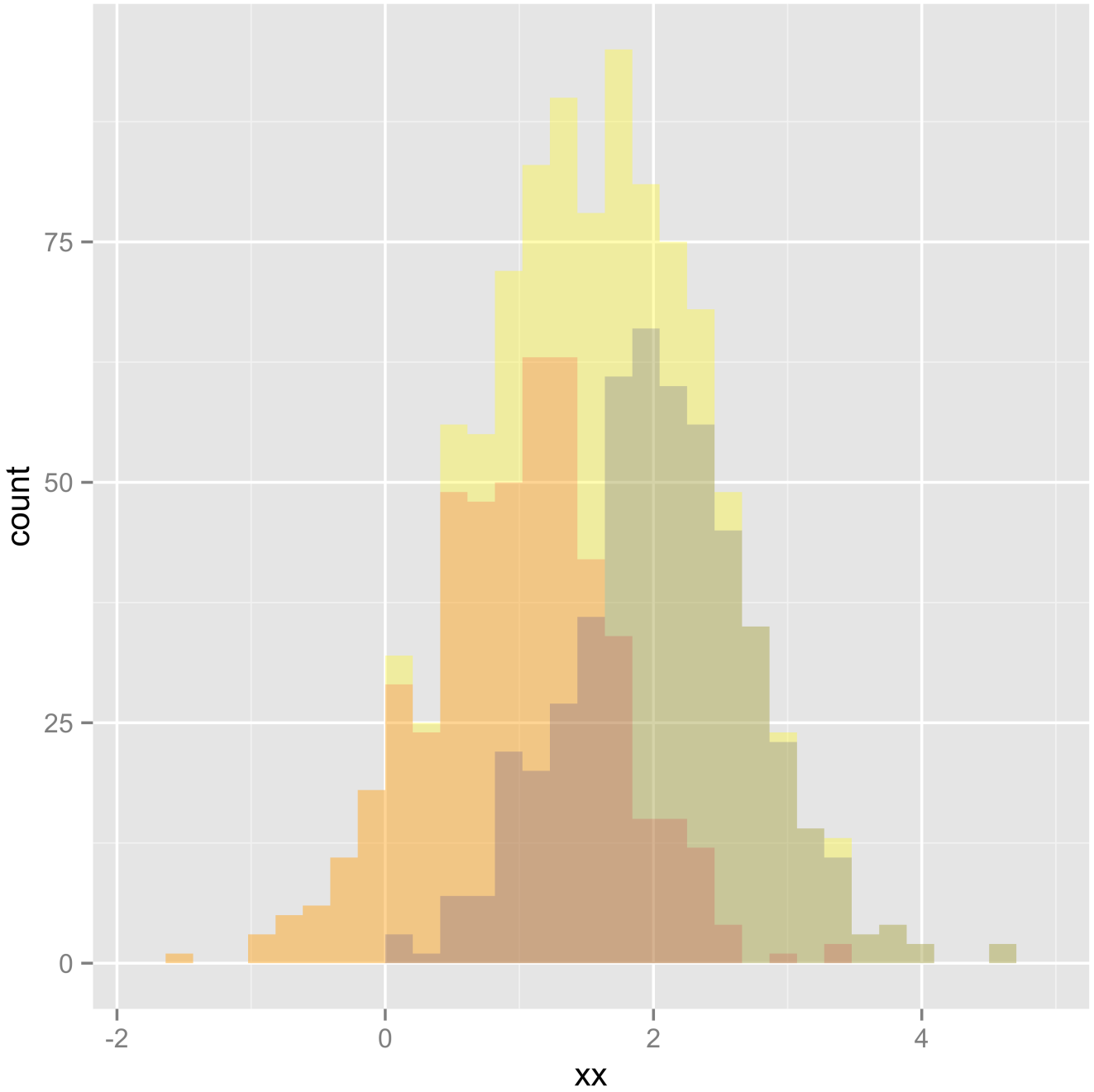
Here is the answer: if we color in red the points that were generated from the heads coin flip and blue the one from tails, we can see that the first normal has a range of about

```
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)
```



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)
```



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

In the case of simple parametric components, we use a method called the EM (Expectation-Maximization) algorithm to infer the value of the hidden variable.

The Expectation-Maximization algorithm is a very popular iterative procedure. We start with observations (Y) and we augment the data with an unobserved (latent) cluster variable U, which says which group each observation came.

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. We will use the notion of bivariate distribution here, we are going to look at what the distribution of `couples` (Y, U) which can be written:

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

Suppose we have a fair mixture of two normals with parameters $\theta = (\mu_1 =?, \mu_2 =?, \sigma_1 = 1, \sigma_2 = 1)$, μ_1 and μ_2 are unknown, we suppose for now, we know that the standard deviations of both distributions is 1.

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

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

Marginal likelihood for the observed y:

$$marglike(\theta; Y) = f(Y|\theta) = \sum_u f(y, u|\theta)$$

Intiatize the parameter θ to any value θ^*

E `expectation' step:

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

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

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 $p(u|y, \theta^*)$**.**

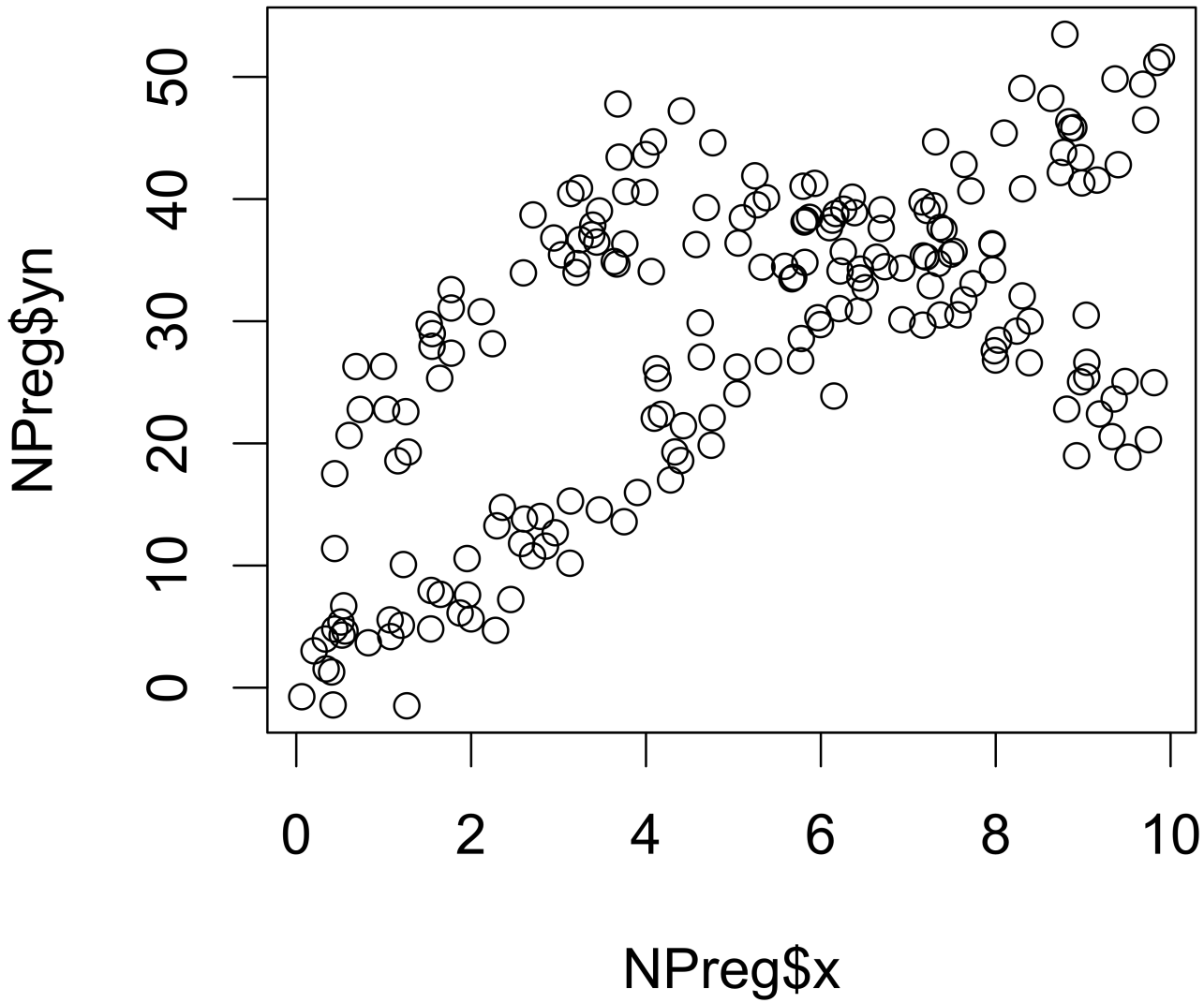
Stop when improvement is negligible.

```
# EM algorithm manually, dat is the data

x <- dat


# initial values for parameters

pi1<-0.5; pi2<-0.5

mu1<--0.01 ;mu2<-0.01

sigma1<-sqrt(0.01) ; sigma2<-sqrt(0.02)

loglik<- rep(NA, 1000)

loglik[1]<-0

loglik[2]<-mysum(pi1*(log(pi1)+log(dnorm(dat,mu1,sigma1))))

          +mysum(pi2*(log(pi2)+log(dnorm(dat,mu2,sigma2))))


mysum <- function(x) {

  sum(x[is.finite(x)])

}

logdnorm <- function(x, mu, sigma) {

  mysum(sapply(x, function(x) {logdmvnorm(x, mu, sigma)}))

}

tau1=0; tau2=0; k=2
```

```
# loop
while(abs(loglik[k]-loglik[k-1]) >= 0.00001) {
  # E step
  tau1=pi1*dnorm(dat,mean=mu1,sd=sigma1)/
      (pi1*dnorm(x,mean=mu1,sd=sigma1)+pi2*dnorm(dat,mean=mu2,sd=sigma2))
  tau2=pi2*dnorm(dat,mean=mu2,sd=sigma2)/
      (pi1*dnorm(x,mean=mu1,sd=sigma1)+pi2*dnorm(dat,mean=mu2,sd=sigma2))
  tau1[is.na(tau1)] = 0.5
  tau2[is.na(tau2)] = 0.5
  # M step
  pi1=mysum(tau1)/length(dat)
  pi2=mysum(tau2)/length(dat)
  mu1=mysum(tau1*x)/mysum(tau1)
  mu2=mysum(tau2*x)/mysum(tau2)
  sigma1=mysum(tau1*(x-mu1)^2)/mysum(tau1)
  sigma2=mysum(tau2*(x-mu2)^2)/mysum(tau2)
  loglik[k+1]=mysum(tau1*(log(pi1)+logdnorm(x,mu1,sigma1)))+
              mysum(tau2*(log(pi2)+logdnorm(x,mu2,sigma2)))
  k=k+1
}

# compare
library(mixtools)
gm=normalmixEM(x,k=2,lambda=c(0.5,0.5),mu=c(-0.01,0.01),sigma=c(0.01,0.02))
gm$lambda
gm$mu
gm$sigma
gm$loglik
```

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.



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 15 iterations
```

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

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

```
##                  Comp.1

## coef.(Intercept) -0.2098

## coef.x            4.8175

## coef.I(x^2)       0.0362

## sigma             3.4760
```

and

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

```
##                 Comp.2
## coef.(Intercept) 14.717
## coef.x            9.847
## coef.I(x^2)      -0.968
## sigma             3.480
```

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 95  5
##   2  5 95
```

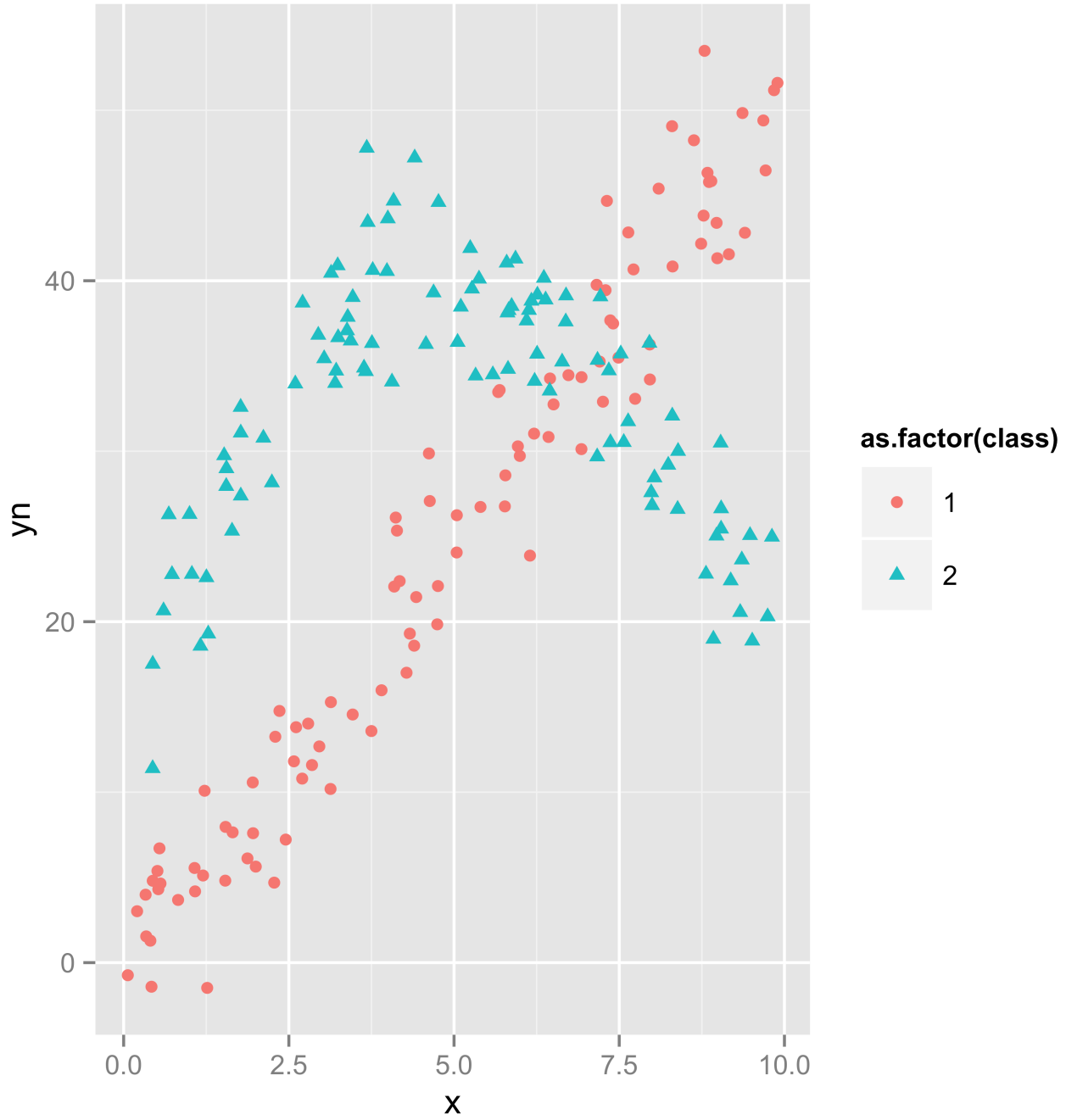
The summary method

```
summary(m1)
```

```
##
## Call:
## flexmix(formula = yn ~ x + I(x^2), data = NPreg, k = 2)
##
##        prior size post>0 ratio
## Comp.1 0.494  100    145 0.690
## Comp.2 0.506  100    141 0.709
##
## 'log Lik.' -643 (df=9)
## AIC: 1303   BIC: 1333
```

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)))
```



Zero inflated models

$$f_{zero} = \pi I_{\{0\}}(y) + (1-\pi) f_{count}(y)$$

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.

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.

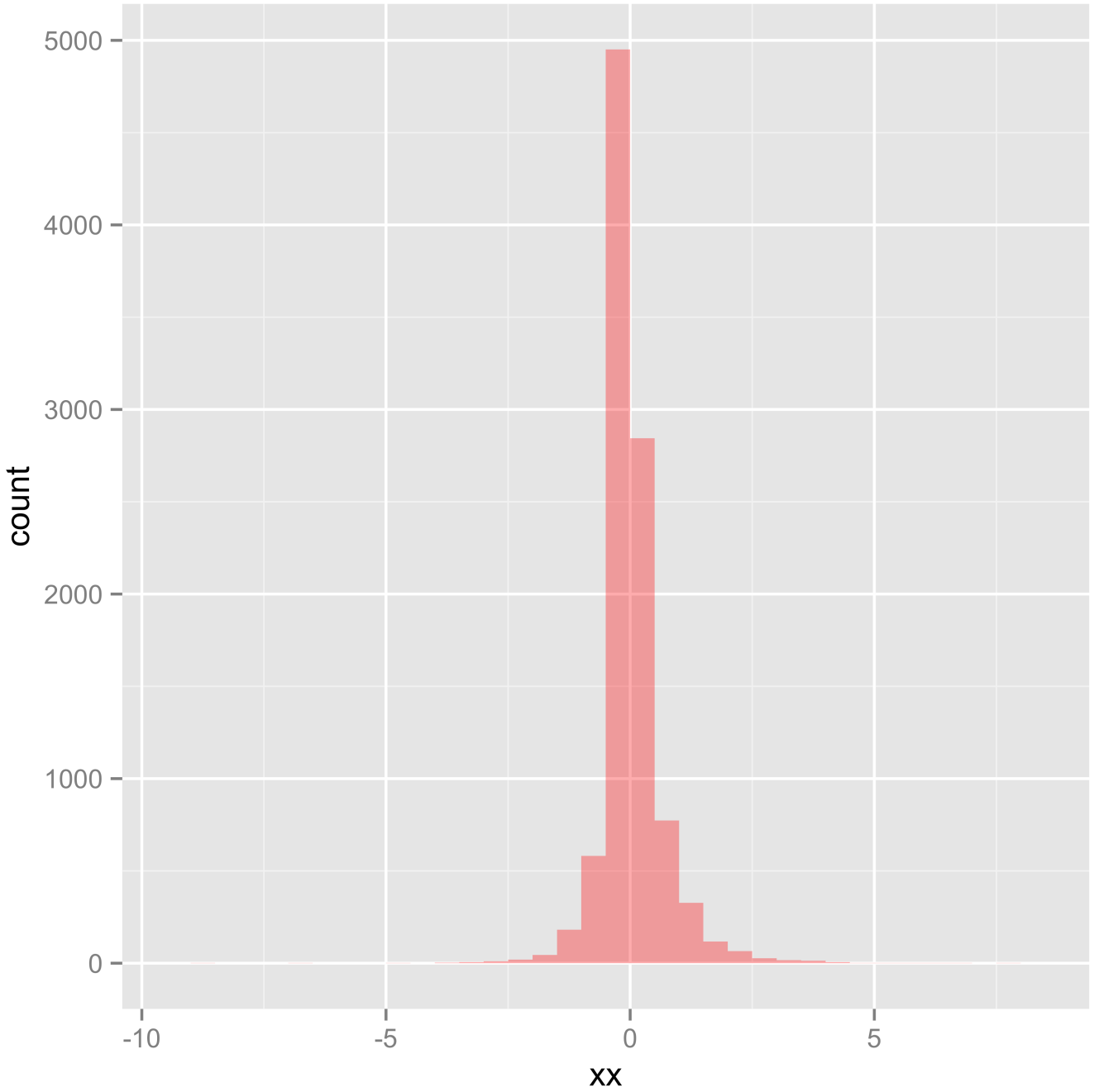
We decomposed the mixture model into a two step process, we extend the description to cases where there could be many more components in the model, suppose that instead of flipping a coin, we picked a ball from an urn with the mean and variances written on it, if half the balls were marked $(\mu_1 = 1, \sigma^2 = 0.1)$ 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.

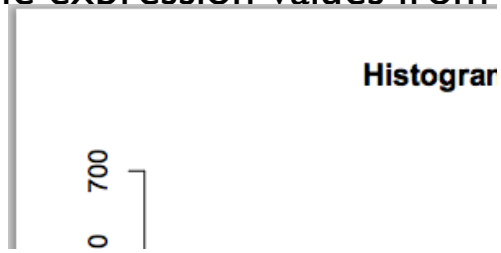
Infinite Mixture of Normals

```
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)
```



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.

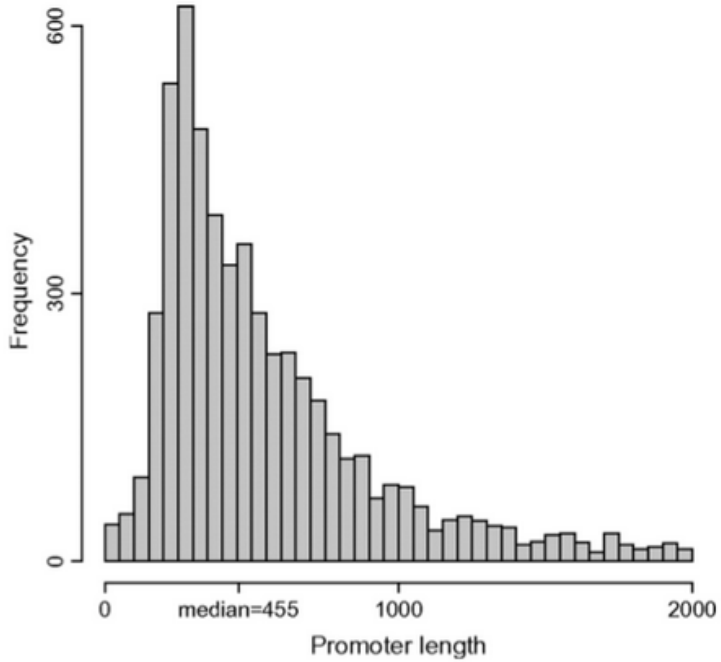
Instead of using the mean and the variance to summarize it, we should use the median and the mean absolute deviation as they have better properties.

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

which is a very nice example of a Laplace distribution.

Promoter Lengths

Histogram of the 5,735 Saccharomyces cerevisiae promoters used in this study.



Kristiansson E et al. Mol Biol Evol 2009;26:1299-1307

MBE MOLECULAR BIOLOGY AND EVOLUTION

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

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

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.

Laplace's First Error Distribution

Useful representation:

$$Y = \sqrt{X} \cdot Z, \qquad X \sim Exp(1), \qquad Z \sim N(0,1)$$

A mixture of Normals whose scale parameters vary from an exponential.

The probe sequences are of different length and so have varying hybridization precisions.

Here is an example of what one can do with such a model:

Example of use: Parametric Bootstrap for Power Simulations

It is very useful when we want to design an experiment to know how well it will detect differences of different sizes, often called the effect size. Fir that, we generate data under various null hypotheses that have as many

features similar to the original data:

- Same error distribution (form and moments), with all the different sources of error incorporated into the study.
- Vary some underlying tuning parameters, sample sizes or number of repetitions, numbers of reads,….

Pretend that the data come from a parametric family F_θ.

Replace in the generation of `new' data, the unknown parameters by $F_{\hat{\theta}}$.

Now we can generate a whole set of simulated data to find the power of our analyses at various levels of sample size and effect size.

Power simulation for Microarrays

Model, after renormalization, and eventual variance stabilization.

$$m_{jk} = m_k \epsilon_k + \sqrt{\epsilon_k} Z_{kj}, \qquad Z_{kj} \sim \mathcal{N}(0, \Sigma)$$

Genes are $k, k = 1....n$. Estimate the parameters, covariance matrix, medians, median absolute deviations. This model is added onto the variance stabilization model.

One may want to change the covariance matrix to adjust for the hypotheses being tested.

Plug these into an `Assymetric Multivariate Laplace generator`.

We can compare the number of genes chosen with the parametric model with a given covariance structure as the one we get with the data.

The Poisson Gamma Mixture Model

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

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

- Overdispersion (in Ecology)
- Simplest Mixture Model for Counts
- Different evolutionary mutation rates
- Throughout Bioinformatics and Bayesian Statistics
- Abundance data

Gamma Distribution: two parameters (shape and scale)

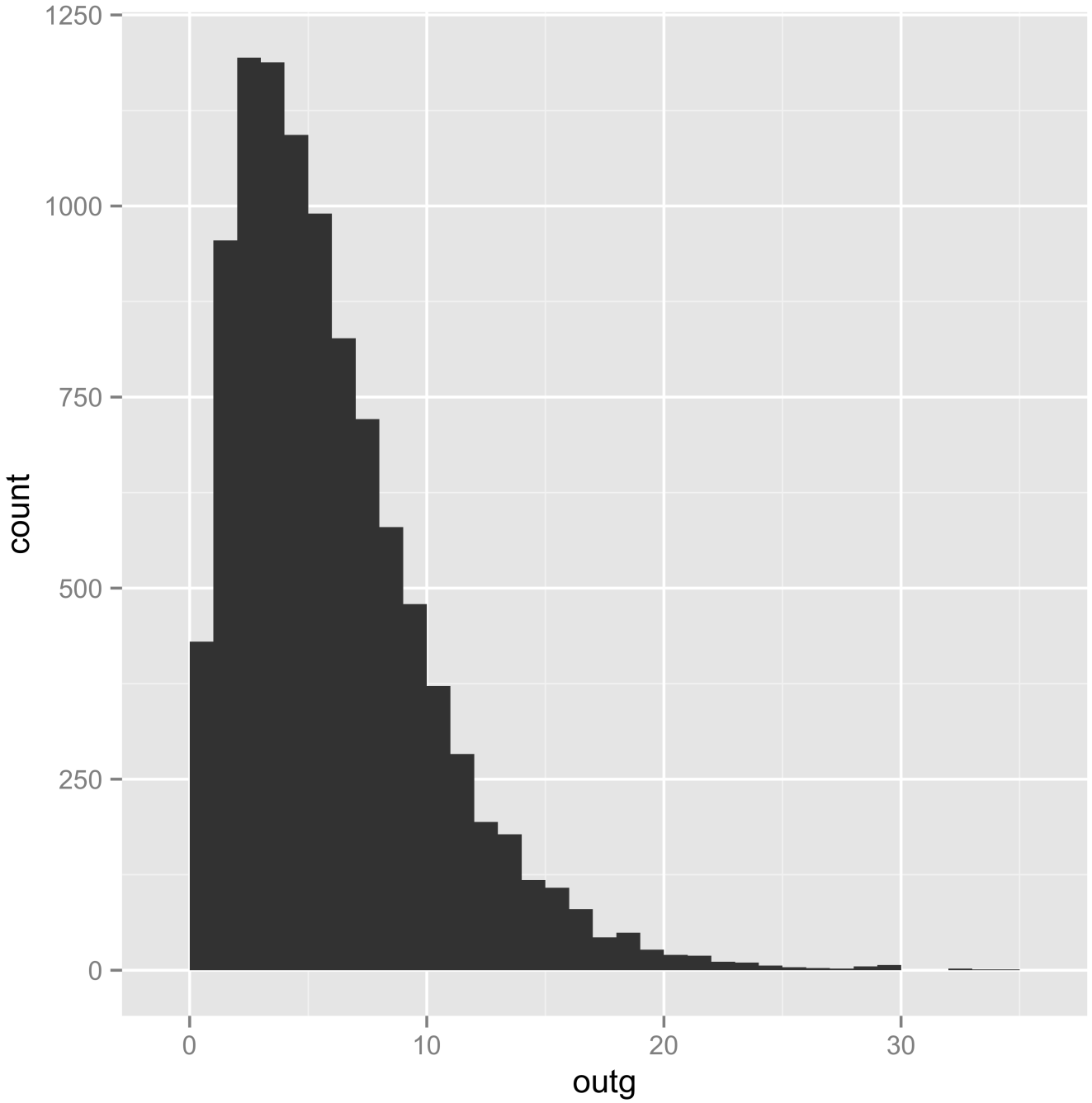
%http://en.wikipedia.org/wiki/Gamma_distribution (http://en.wikipedia.org/wiki/Gamma_distribution) wikigamma (http://en.wikipedia.org/wiki/Gamma_distribution) Like the Beta distribution, the Gamma distribution is used to model certain continuous variables, however the random variables that have a Gamma 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
```



Note on fitting distributions:

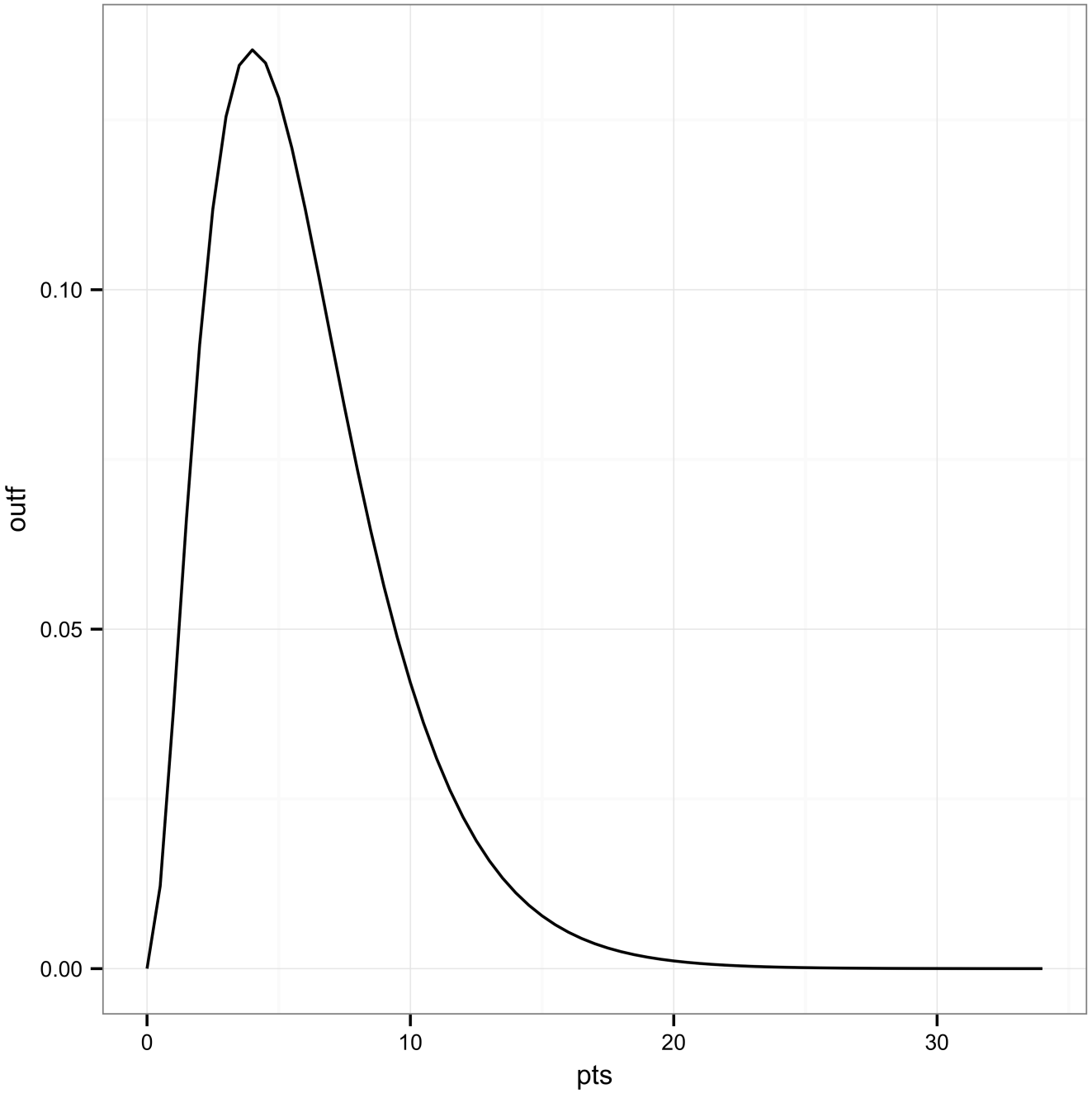
```
require(MASS)

## 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)
```



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")

goodnb
```

```
##
## Observed and fitted values for nbinomial distribution
## with parameters estimated by `ML'
##
##  count observed fitted
##      0       10  7.673
##      1        6  9.895
##      2       12 10.191
##      3        9  9.613
##      4        9  8.652
##      5        6  7.562
##      6        6  6.479
##      7        6  5.471
##      8        6  4.568
##      9        3  3.782
##     10        2  3.109
##     11        2  2.542
##     12        2  2.067
##     13        2  1.675
##     14        3  1.352
##     15        3  1.088
##     16        1  0.873
##     17        2  0.699
```

`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

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

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)
```

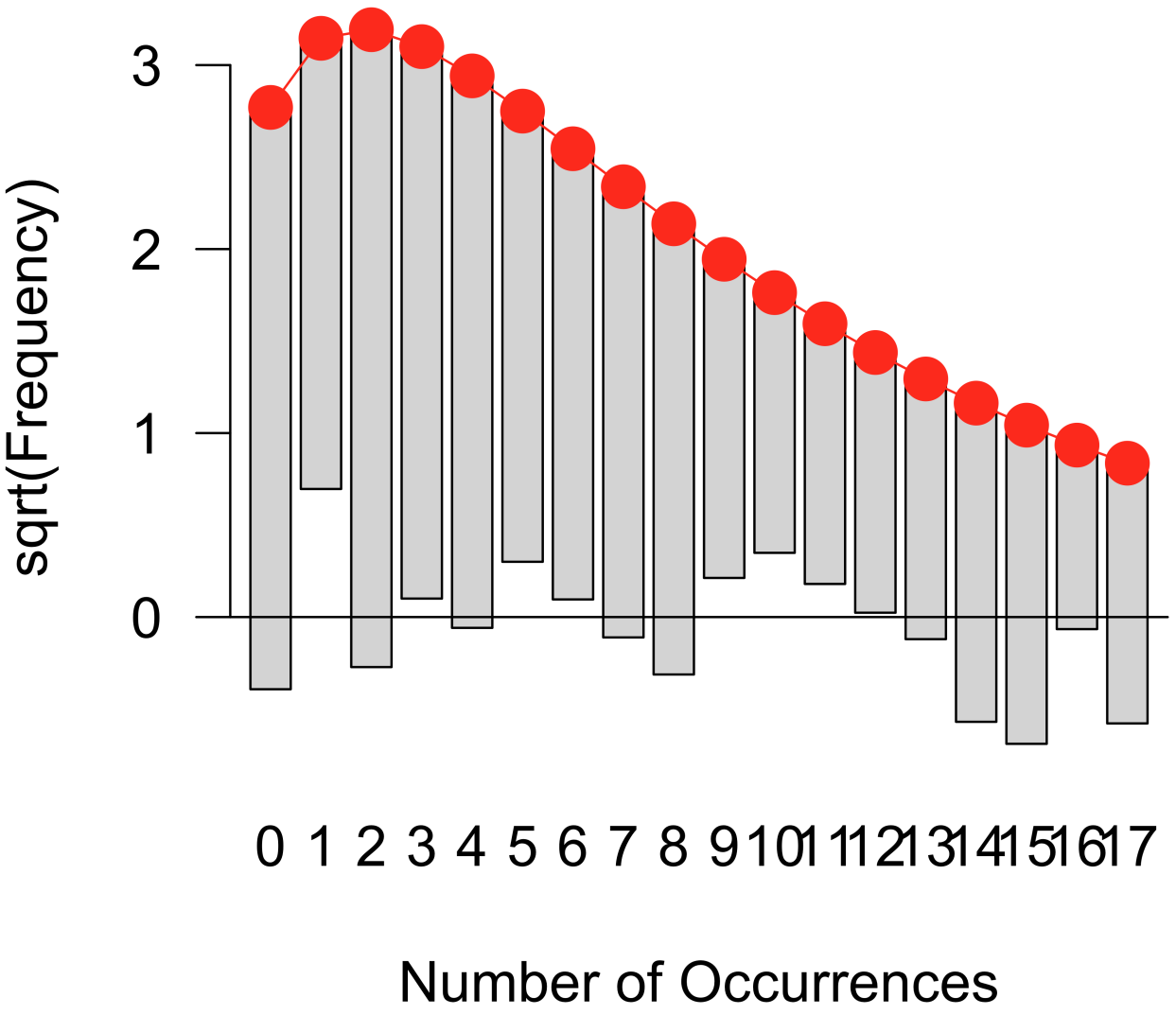
```
##

##   Goodness-of-fit test for nbinomial distribution

##

##                   X^2 df P(> X^2)

## Likelihood Ratio 15.2 15    0.435
```

```
goodnb$par
```

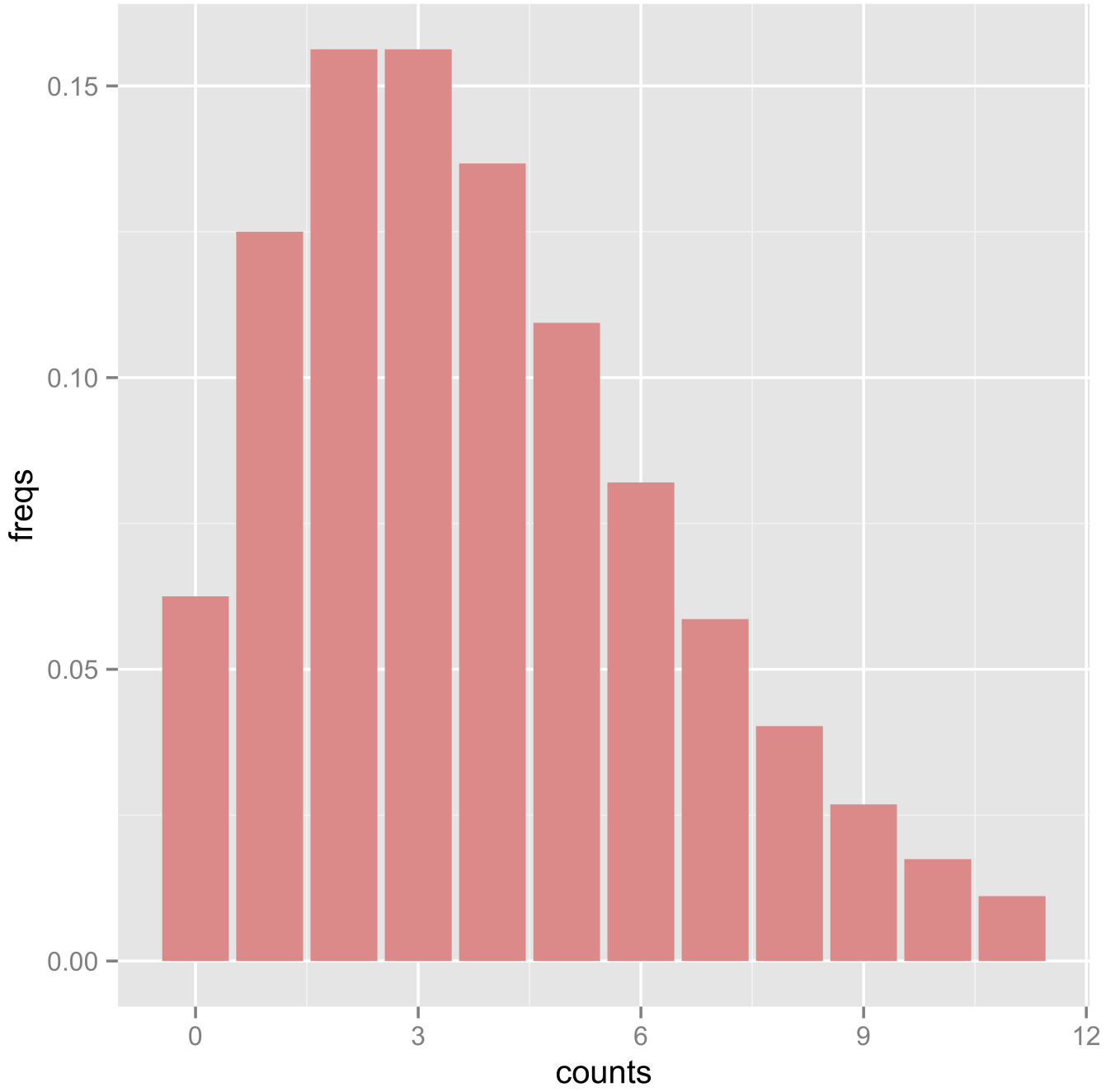
```
## $size

## [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
```


sqrt(Frequency)
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
Number of Occurrences

```
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")
```



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 $\frac{b}{b+1}$.

Example of use: Gamma Exponential Noise Models (used for PCR)

We will see another example of using a Gamma to generate a mixture, but this time for the error model for PCR where the second distribution is exponential with the parameter generated by a Gamma. (See link Gamma-Exponential Noise Models (http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3137271/) useful to model noise in PCR experiments.)

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:

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.

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 .

Overexpressed genes

Mainstay in multiple testing when trying to find relevant genes in microarray and RNA-seq transcriptomic 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.

Small Note: There is mathematical reason….

It is actually not a mystery why mixtures are so ubiquitous, there is even a theorem that says that if the order in which the observations are made doesn't matter, then they are from a mixture….

If we don't know much about the noise but we know that the order in which we collect the data doesn't matter, that is called exchangeability:

$$D(X_1, X_2, X_3, \ldots, X_n) = D(X_{\pi(1)}, X_{\pi(2)}, X_{\pi(3)}, \ldots, X_{\pi(n)})$$

For pi any permutation of size n.

If the random variables $(X_1, X_2, X_3, \ldots, X_n)$ are independent then

$$P(X_1 = x_1, X_2 = x_2, X_3 = x_3, \ldots, X_n = x_n) = \prod_{i=1}^{n} P(X_1 = x_1)P(X_2 = x_2)P(X_3 = x_3) \ldots$$

commutativity of multiplication gives us exchangeability.
The converse is not true, but conditional independence through mixing.
Binary Data Example:

$$p(1, 1, 0, 0, 0, 1, 0, 0) = p(1, 0, 1, 0, 0, 0, 0, 1)$$

Exchangeable but not independent: Polya's urn and the restaurant.
Stefan Lauritzen Lecture
(http://www.stats.ox.ac.uk/~steffen/teaching/grad/definetti.pdf)

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).