Implementation of Bayesian mixture models for copy number estimation

Jacob Carey, Steven Cristiano, and Robert Scharpf

October 17, 2016

Contents

1 Introduction

CNPBayes models multi-modal densities via a hierarchical Bayesian Gaussian mixture model. The major application of this model is the estimation of copy number at copy number polymorphic loci (CNPs). Two versions of the mixture model are implemented. A *standard* model, referred to as a *marginal* model, that has one mean and standard deviation for each component, and a *batch* model with batch-specific means and standard deviations. Approximation of the posterior is by Markov Chain Monte Carlo (MCMC) written in C++ using the Rcpp package (Eddelbuettel and François 2011).

For an EM-implementation of Gaussian mixture models for CNPs, see the Bioconductor package CNVtools (Barnes et al. 2008). A Bayesian extension of this model by some of the same authors was developed to automate the analysis of the Welcome Trust Case Control Consortium (WTCCC) genotype data (Cardin et al. 2011) and implemented in the R package CNVCALL (http://niallcardin.com/CNVCALL).

```
library(CNPBayes)
```

2 Implementation

CNPBayes uses several S4 classes to encapsulate key parameters of the MCMC simulation, reducing the need for functions with a large number of arguments and providing an explicit contract for the arguments passed to the C++ back-end. The three core classes are

- `McmcParams`: parameters for the number of burnin simulations - the number of chains to initialize, the number of simulations after burnin, and how often simulated values are to be saved.
- `Hyperparameters`: a virtual class extended by `HyperparametersMarginal` and `HyperparametersBatch` for the marginal and batch mixture model implementations, respectively.

- MixtureModel: a virtual class with slots for data and hyperparameters, as well as a slot for each parameter. The class is extended by MarginalModel and BatchModel for the *marginal* and *batch* implementations, respectively. S4 dispatch on these classes is used to handle MCMC updates that are specific to the marginal or batch models.

3 Approximating the posterior of a mixture

3.1 MarginalModel

3.1.1 McmcParams

The class McmcParams specifies several MCMC options, including the number of saved iterations (iter), length of burn-in (burnin), and the thinning interval (thin). The following code indicates that we will run a burnin of 100 iterations saving the first 1000 iterations after burnin, with no iterations discarded for thinning.

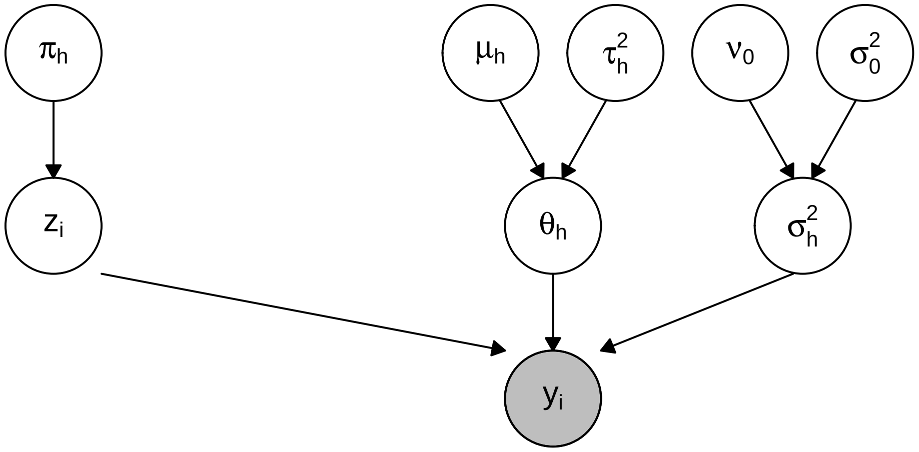
```
mp <- McmcParams(iter=1000, burnin=100, thin=1)
```

If we instead specified thin=10, we would need to run 10,000 MCMC iterations in order to have a chain of length 1000 (every 10th observation is saved).

3.1.2 Hyperparameters

Hyperparameters for the mixture model are specified as an instance of class Hyperparameters. Hyperparameters include:

- k the number of components (or copy number). Defaults to 2.
- mu.0 and tau2.0 priors for $\mu \sim$ N(mu.0, tau2.0), the overall mean of components. Default to 0 and 100 respectively.
- eta.0 and m2.0 priors for $\tau^2 \sim$ Ga(shape=eta.0, rate=m2.0), the overall variance across components. Default to 1 and 0.1 respectively for *marginal* models and 1800 and 1/60 respectively for *batch* models.
- alpha the prior mixture probabilities. Does not have to sum to 1. By default, a noninformative prior of equal mixtures is used.
- beta prior for ν_0. Defaults to 0.1.
- a and b priors for $\sigma_0^2 \sim$ Ga(shape=a, rate=b), the rate parameter for σ^2, the variance for each batch and component.



Constructing an instance of class `Hyperparameters` for a `MarginalModel` can be performed as follows.

```
hypp <- Hyperparameters(type="marginal", k=3)
```

3.1.3 simulateData

CNPBayes allows for the simulation of test data. The number of observations, mixture proportions, means for each component, and standard deviations for each component must be specified.

```
sim.data <- simulateData(N=2500, p=rep(1/3, 3),
                         theta=c(-1, 0, 1),
                         sds=rep(0.1, 3))
```

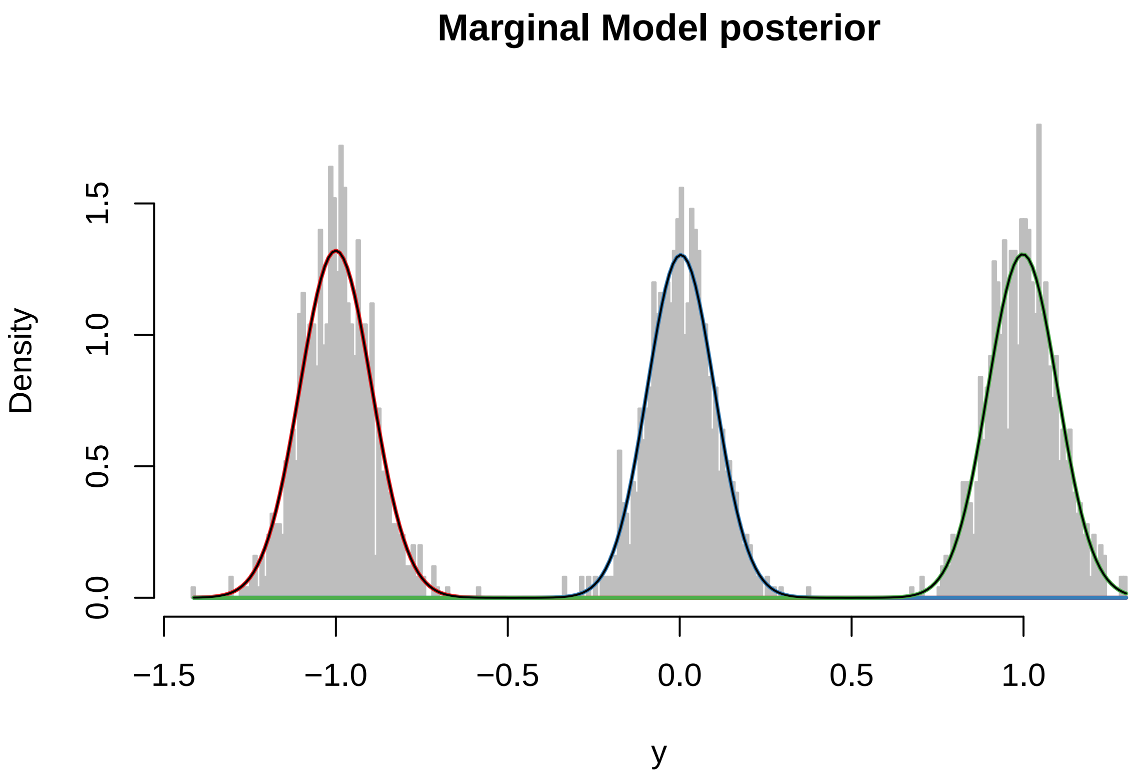
3.1.4 posteriorSimulation

To simulate the posterior distribution, an instance of class `MarginalModel` must first be constructed. Note that when initializing objects of this class, the parameters `data`, `k` (number of *a priori* components), `hypp`, (object of class `Hyperparameters`) and `mcmc.params` (object of class `McmcParams`) should be specified. Default values for `hypp` and `mcmc.params` will be used if not specified. The `y` method should be used to retrieve the data from an object. After construction of a model, the `posteriorSimulation` method should be used.

```
model <- MarginalModel(data=y(sim.data), k=3,
                       hypp=hypp,
                       mcmc.params=mp)

model <- posteriorSimulation(model)
```

The results of a DensityModel call returns an object of class DensityModel. Use the data and a DensityModel for plotting.

```
plot(DensityModel(model), y(sim.data),
     main="Marginal Model posterior")
```

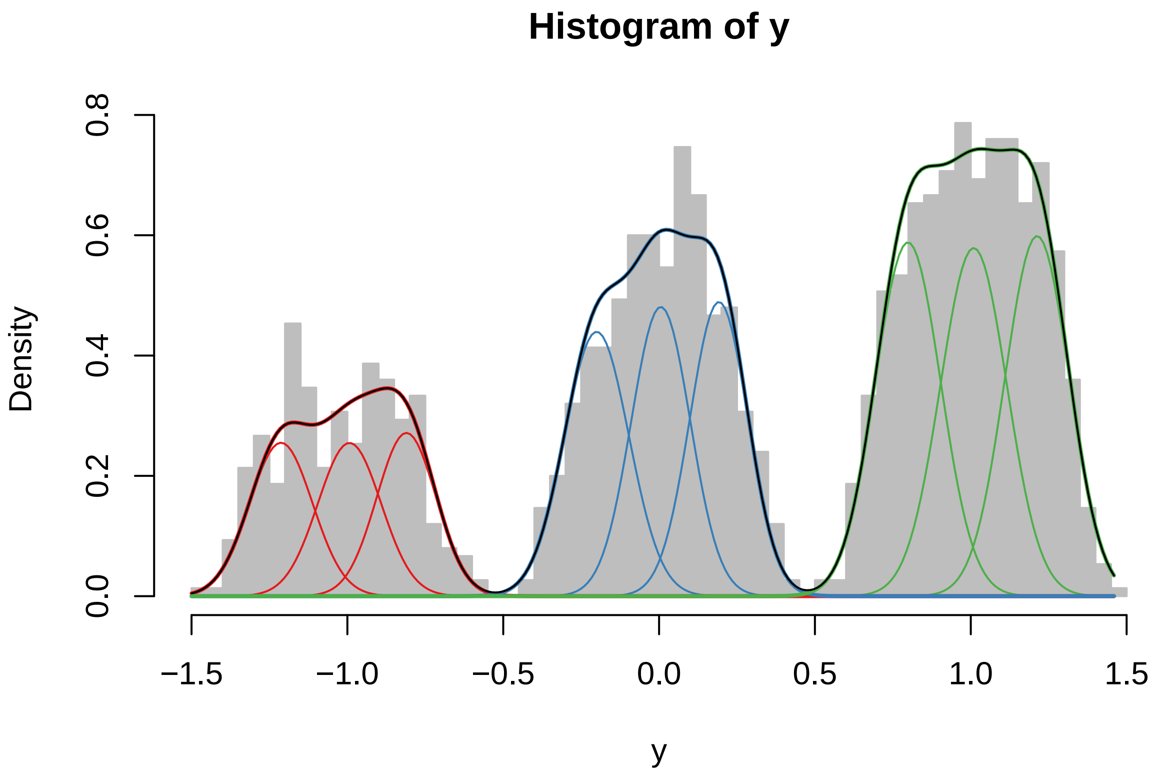


3.2 BatchModel

In general, the construction and posterior simulation of a BatchModel is similar to that of a MarginalModel. The BatchModel is hierarchical over the batches, and thus requires information about the batches. Instances of McmcParams are equivalent between BatchModel and MarginalModel. However, an object of class Hyperparameters is constructed with a type of “batch”. Additionally, simulated batch data is created using simulateBatchData which requires theta and sds to be specifed as $B \times K$ matrices, for K components and B batches. simulateBatchData also requires a batch parameter, labelling the batch of each observation. Finally, BatchModels are constructed using the function BatchModel which operates similarly to MarginalModel but requires a batch parameter, similar to Hyperparameters. plots of BatchModels include batch specific density estimates.

```
## Create McmcParams for batch model
mp <- McmcParams(iter=2000, burnin=1000, thin=1)

## Create Hyperparameters for batch model
hypp <- Hyperparameters(type="batch", k=3)

## simulate batch data
k <- 3
nbatch <- 3
```

```
means <- matrix(c(-1.2, -1.0, -0.8,
                  -0.2, 0, 0.2,
                  0.8, 1, 1.2), nbatch, k, byrow=FALSE)
sds <- matrix(0.1, nbatch, k)
N <- 1500
sim.data <- simulateBatchData(N=N,
                              batch=rep(letters[1:3], length.out=N),
                              theta=means,
                              sds=sds,
                              p=c(1/5, 1/3, 1-1/3-1/5))

# create BatchModel and run posteriorSimulation
model <- BatchModel(data=y(sim.data), k=3,
                    batch=batch(sim.data),
                    hypp=hypp,
                    mcmc.params=mp)

model <- posteriorSimulation(model)
plot(DensityModel(model), y(sim.data),
     breaks=100)
```



While aspects of the `BatchModel` look somewhat Gaussian, there is some evidence of departure from a normal distribution. Due to this skewness, a `MarginalModel` would not capture the distribution well. However, the `BatchModel` does a good job of capturing the shape.

4 K unknown

Generally, the number of components is not known *a priori*. To estimate the number of components *K*, the `marginalLikelihood` function is used. It is important to note that a list of models passed to this function must have converged. For ease of use, one can specify `k` in the posteriorSimulation to use the same data, `Hyperparameters`, and `mcmc.params`, but with a different number of components.

```
mp <- McmcParams(iter=2e3, burnin=1e3)
model <- MarginalModel(data=y(sim.data), k=2, mcmc.params=mp)

mlist <- list(posteriorSimulation(model),
              posteriorSimulation(model, k=3))

x <- marginalLikelihood(mlist)
x
##       SB2       SB3
## -1502.772 -1255.230
```

`marginalLikelihood` uses Chib's Estimator (Chib 1995) to run a reduced Gibbs to estimate the marginal likelihood of each component size. Additionally, `marginalLikelihood` can be used in a similar manner for `BatchModels`.

Alternative methods for selection of *k* are also included. As an example, the `bic` method can be used for calculating the Bayesian Information Criterion.

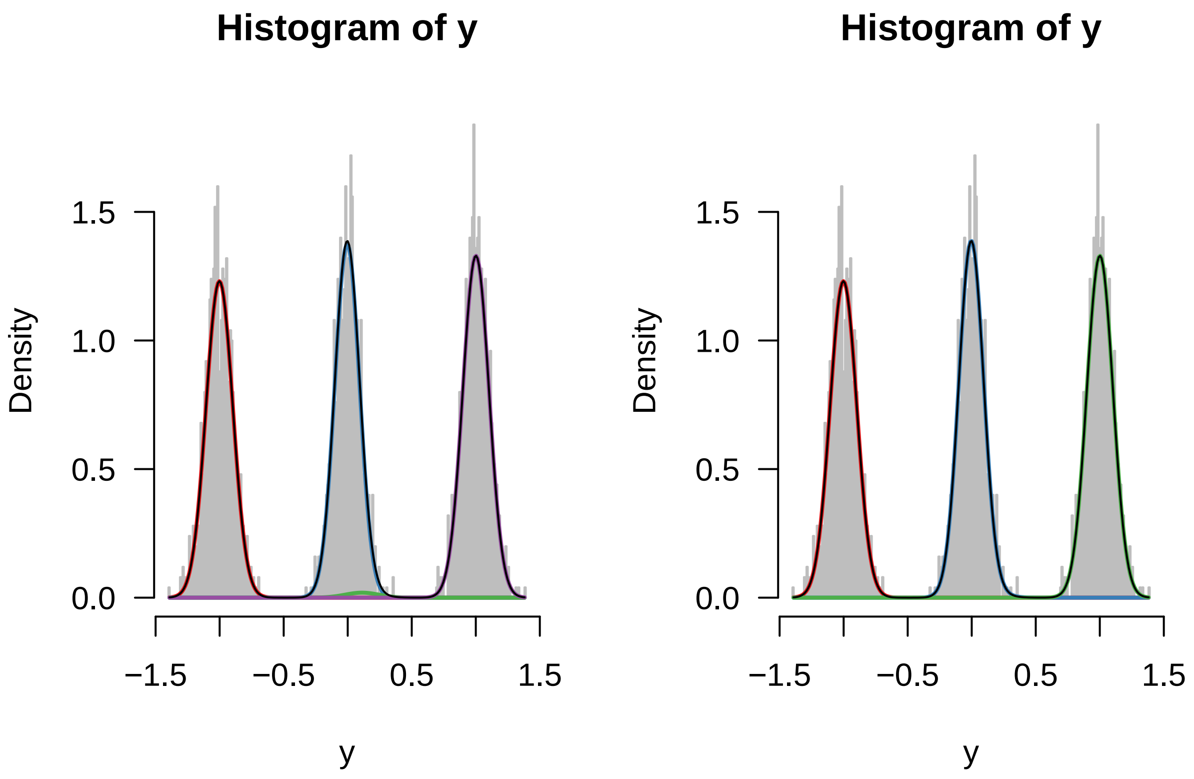
```
## simulate k=2 model
sim.data <- simulateData(N=2500, p=rep(1/3, 3),
                         theta=c(-1, 0, 1),
                         sds=rep(0.1, 3))
hypp1 <- Hyperparameters(k=2)
m1 <- MarginalModel(data=y(sim.data), k=2,
                    hypp=hypp1,
                    mcmc.params=mp)
m1 <- posteriorSimulation(m1)

## simulate k=3 model
hypp2 <- Hyperparameters(k=3)
m2 <- MarginalModel(data=y(sim.data), k=3,
                    hypp=hypp2,
                    mcmc.params=mp)
m2 <- posteriorSimulation(m2)

bic(m1)
## [1] 4231.972
bic(m2)
## [1] 1278.851
```

As the data is simulated from a three component distribution, `m2` should be chosen. Using the Bayesian Information Criterion, the correct model is chosen in this simple example.

Finally, in an overfit model (one with too many components), merging components is supported.

```
mp <- McmcParams(iter=5000, burnin=1000, thin=1)
model <- MarginalModel(data=y(sim.data), k=4,
                       hypp=Hyperparameters(k=4),
                       mcmc.params=mp)
model <- posteriorSimulation(model)
dm <- DensityModel(model)
```

```
modes(dm)
## [1] -1.0021964940  0.0002769548  1.0027504036
dm_merged <- DensityModel(model, merge=TRUE)
k(dm_merged)
## [1] 3

par(mfrow=c(1,2), las=1)
plot(dm, y(sim.data))
plot(dm_merged, y(sim.data))
```



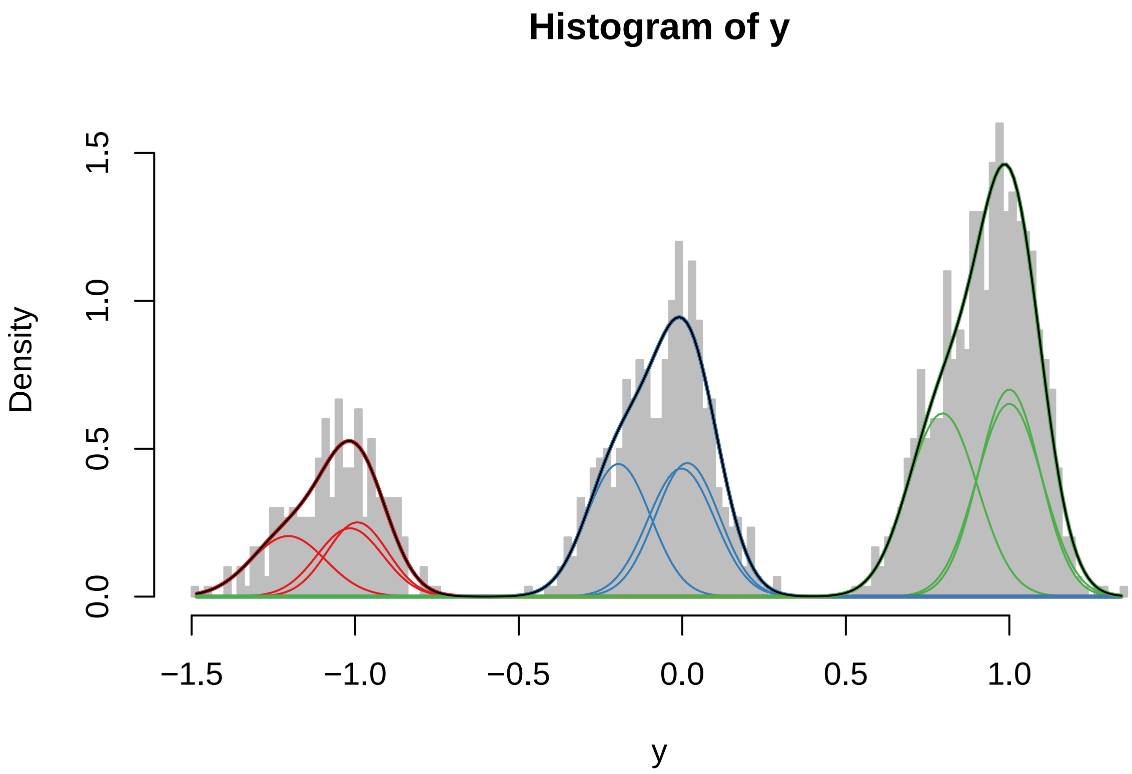
This merging procedure also ensures that there is no underfitting. For example, merging a model with three true components will yield a model with three components.

```
model <- MarginalModel(data=y(sim.data), k=3,
                       hypp=Hyperparameters(k=3),
                       mcmc.params=mp)
model <- posteriorSimulation(model)
dm <- DensityModel(model)
dm_merged <- DensityModel(model, merge=TRUE)
k(dm)
## [1] 3
k(dm_merged)
## [1] 3
```

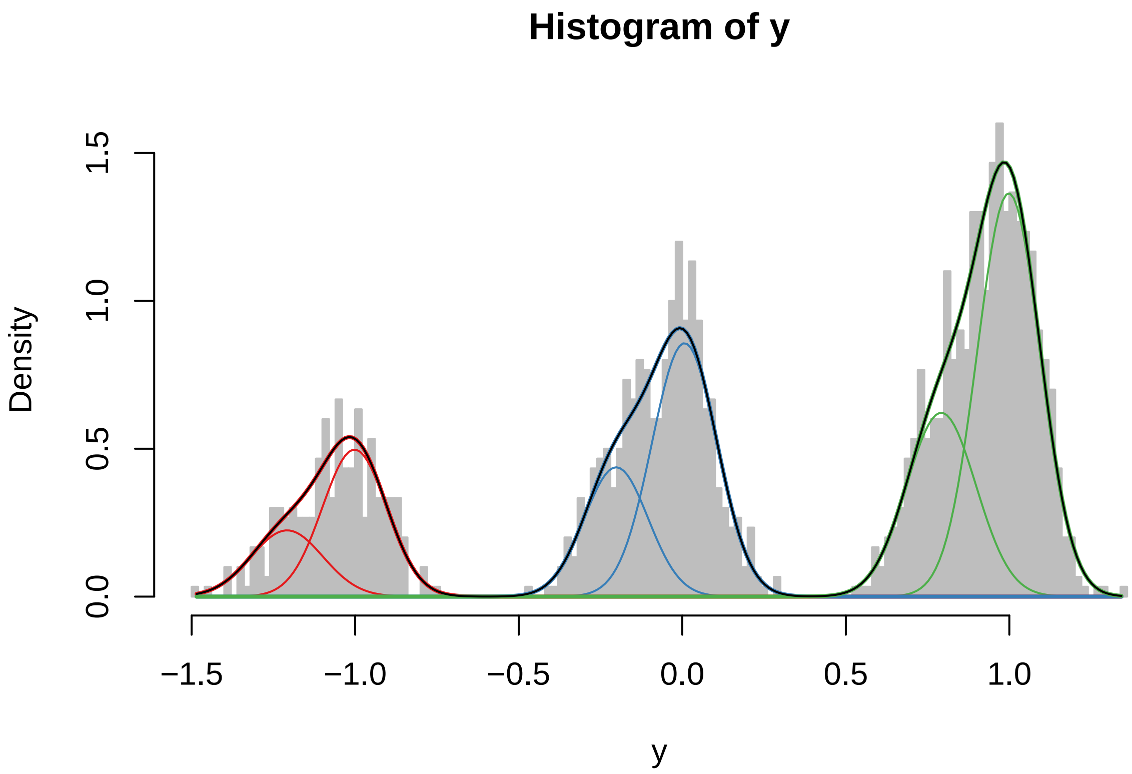
5 Batch Estimation

Differences in the one-dimensional summaries between groups of samples can arise due to technical sources of variation referred to as *batch effects*. For example, date, temperature, machine calibration, and lab technician are all potentially important technical sources of variation (Leek et al. 2010). Batch effects can vary by locus, with some loci more susceptible to technical factors that may affect measurement. The 96 well chemistry plate on which the samples were processed is often a useful surrogate for batch effects since this tends to capture samples that were processed (e.g., PCR amplification) and scanned at approximately the same time. However, in large studies involving potentially hundreds of chemistry plates, it is not computationally feasible to fit fully Bayesian plate-specific mixture models. Because chemistry plates are often processed at similar times and may be comparable in terms of the distribution of a statistic of interest, it may be more useful (and computationally scalable) to *batch* the chemistry plates. Here, we implement a simple two-sample Kolmogorov-Smirnov (KS) test for each pairwise combination of chemistry plates in the function `collapseBatch`. The two-sample KS test is applied recursively until no two combinations of grouped plates can be combined. We illustrate with a trivial example.

```
k <- 3
nbatch <- 3
means <- matrix(c(-1.2, -1.0, -1.0,
                -0.2, 0, 0,
                0.8, 1, 1), nbatch, k, byrow=FALSE)
sds <- matrix(0.1, nbatch, k)
N <- 1500
sim.data <- simulateBatchData(N=N,
                              batch=rep(letters[1:3], length.out=N),
                              theta=means,
                              sds=sds,
                              p=c(1/5, 1/3, 1-1/3-1/5))

plot(sim.data)
```



```
## An object of class 'DensityBatchModel'
##    batch: list of  3 matrices
##    component densities:  list of 3 vectors
##    overall density:  vector of length 250
##    modes:  -1.02, -0.01, 0.99

model <- BatchModel(data=y(sim.data), k=3,
                    batch=batch(sim.data))

model <- BatchModel(data=y(sim.data), k=3,
                    batch=collapseBatch(model))
## .

model <- posteriorSimulation(model)
plot(model)
```



```
## An object of class 'DensityBatchModel'
##    batch: list of  3 matrices
##    component densities:  list of 3 vectors
##    overall density:  vector of length 250
##    modes:  -1.02, -0.01, 0.98
```

As is shown, the model correctly collapses batches two and three, since they are drawn from the same distribution.

References

Barnes, Chris, Vincent Plagnol, Tomas Fitzgerald, Richard Redon, Jonathan Marchini, David Clayton, and Matthew E Hurles. 2008. "A Robust Statistical Method for Case-Control Association Testing with Copy Number Variation." *Nat Genet* 40 (10). Wellcome Trust Sanger Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge, UK.: 1245–52. doi:10.1038/ng.206.

Cardin, Niall, Chris Holmes, Peter Donnelly, and Jonathan Marchini. 2011. "Bayesian Hierarchical Mixture Modeling to Assign Copy Number from a Targeted CNV Array." *Genet. Epidemiol.* doi:10.1002/gepi.20604.

Chib, Siddhartha. 1995. "Marginal Likelihood from the Gibbs Output." *Journal of the American Statistical Association* 90 (432): 1313. doi:10.2307/2291521.

Eddelbuettel, Dirk, and Romain François. 2011. "Rcpp: Seamless R and C++ Integration." *Journal of Statistical Software* 40 (8): 1–18. http://www.jstatsoft.org/v40/i08/.

Leek, Jeffrey T., Robert B. Scharpf, Héctor Corrada Bravo, David Simcha, Benjamin Langmead, W. Evan Johnson, Donald Geman, Keith Baggerly, and Rafael A. Irizarry. 2010. "Tackling the Widespread and Critical Impact of Batch Effects in High-Throughput Data." *Nat Rev Genet* 11 (10): 733–39. doi:10.1038/nrg2825.