CRISPhieRmix Manual

Timothy Daley

8/2/2018

Contents

Contents

1 Introduction to CRISPhieRmix

CRISPhieRmix is an R package for analysing large CRISPR interference and activation (CRISPRi/a) screens. CRISPhieRmix uses a hierarchical mixture model approach to identify genes that are unlikely to follow the null distribution. CRISPhieRmix assumes that genes follow a mixture of null genes (genes with no effect) and non-null genes (genes with a significant effect). All guides from null genes follow a common null distribution. The guides for the non-null genes, on the other hand, are assumed to follow a mixture distribution, where some guides are ineffective (possibly with little or no change in the target gene expression) and follow the null distribution and some guides have an effect and follow an alternative distribution.

CRISPhieRmix can be used with out without negative control guides. We find that negative control guides help to better model the null distribution, as in our experience the null distribution tends to have long tails, and this helps to control the false discovery rate. A critical assumption is that the negative control guides accurcately reflect the null distribution. This assumption can be violated in some screens, and we will discuss this later in depth.

If you have any issues with software, please create an issue in the github page https://github.com/timydaley/CRISPhieRmix. If you have any questions, please email me at tdaley@stanford.edu.

2 Input parameters for CRISPhieRmix

2.1 Input

- **x** log2 fold changes of guides targeting genes (required)
- **geneIds** gene ids corresponding to x (required)
- **negCtrl** log2 fold changes of negative control guides; if negCtrl is not included then CRISPhieRmix uses a normal hierachical mixture model
- **max_iter** maximum number of iterations for EM algorithm, default = 100
- **tol** tolerance for convergence of EM algorithm, default = 1e-10
- **pq** initial value of p·q, default = 0.1
- **mu** initial value of mu for the interesting genes, default = -4
- **sigma** initial value of sigma for the interesting genes, default = 1
- **nMesh** the number of points to use in numerical integration of posterior probabilities, default = 100
- **BIMODAL** boolean variable for BIMODAL mode to fit both positive and negative sides, used for cases such as Jost et al. 2017 where both sides are of interest.
- **VERBOSE** boolean variable for VERBOSE mode, default = FALSE
- **PLOT** boolean variable to produce plots, default = FALSE

2.2 Return

- **mixFit** a list containing the mixture fit for **x**
- **genes** a vector of genes from **geneIds**, in the same order as the following return values
- **locfdr** a vector of local false discovery rates, the posterior probablity a gene is null in the same order as **genes**
- **FDR** a vector of global false discovery rates in the same order as **genes**
- **genePosteriors** a vector of posterior probabilities that the genes are non-null (equal to 1 - **locfdr**), in the same order as **genes**; when **BIMODAL** is set to TRUE then both **negGenePosteriors** and **posGenePosteriors** are returned that give the posterior probability that a gene is negative and positive, respectively

3 Examples

3.1 A CRISPRi dropout screen

Gilbert et al. 2014 performed genome wide screens for ricin resistance and susceptibility. In table 2, sheet 1 the authors provide the results of the CRISPRi screen at the sgRNA level. This includes the growth phenotype gamma and the ricin phenotype rho. They were interested in the ricin phenotype, but we will look at the growth phenotype to identify genes that lead to decreased growth and can thus be considered essential. In the CRISPhieRmix paper we used log2 fold changes computed by DESeq2 for a fairer comparison of algorithms (since other algorithms such as MAGeCK work directly on the counts), but here we will use the scores computed by Gilbert et al to show the flexibility of the CRISPhieRmix approach.

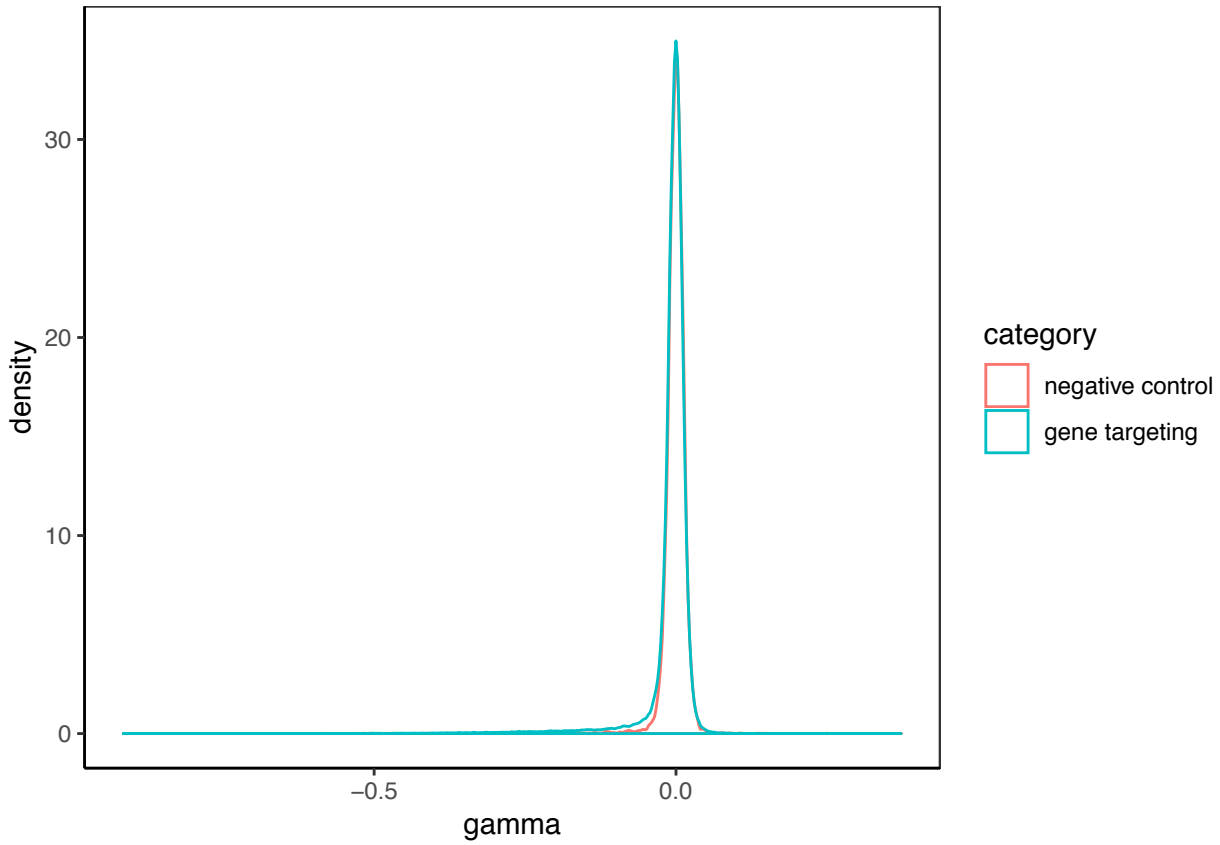
```
Gilbert2014Table2CRISPRi = read.table(file = "Gilbert2014Table2CRISPRi.txt", sep = "\t",
                                      header = TRUE)
head(Gilbert2014Table2CRISPRi)
```

```
##   gene sgRNA.ID Transcripts.targeted   Protospacer.sequence
## 1 A1BG   A1BG-1                  all  GCAAGAGAAAGACCACGAGCA
## 2 A1BG  A1BG-10                  all  GCGGGAACAGGAGCCTTACGG
## 3 A1BG   A1BG-2                  all  GTCTGCAGCAATGAGGCCCCA
## 4 A1BG   A1BG-3                  all   GCAGCCATATGTGAGTGCAG
## 5 A1BG   A1BG-4                  all GACATGATGGTCGCGCTCACTC
## 6 A1BG   A1BG-5                  all  GAATGGTGGGCCCAGGCCGGG
##   Growth.phenotype..gamma. CTx.DTA.phenotype..rho.
## 1             -0.008127700             0.011324965
## 2             -0.006738800             0.011671726
## 3             -0.009591072            -0.028942999
## 4             -0.017039814             0.046149024
## 5             -0.004868127             0.004907662
## 6             -0.012730560            -0.046096108
```

```
# identify the negative control guides
head(sort(table(Gilbert2014Table2CRISPRi$gene), decreasing = TRUE))
```

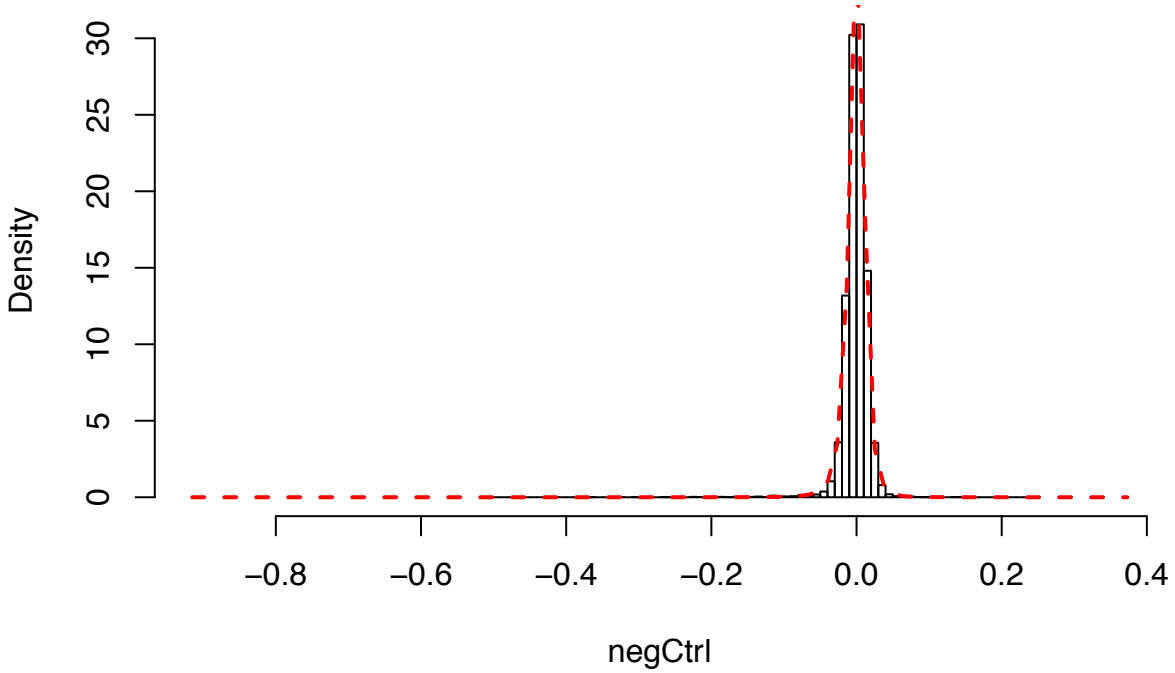
```
##
## negative_control             HLA            NKX2         C1QTNF9B
##            11219             193              85               45
##            STON1            SSPO
##               45              44
```

```
geneTargetingGuides = which(Gilbert2014Table2CRISPRi$gene != "negative_control")
negCtrlGuides = which(Gilbert2014Table2CRISPRi$gene == "negative_control")
gamma = Gilbert2014Table2CRISPRi$Growth.phenotype..gamma.[geneTargetingGuides]
negCtrl = Gilbert2014Table2CRISPRi$Growth.phenotype..gamma.[negCtrlGuides]
geneIds = Gilbert2014Table2CRISPRi$gene[geneTargetingGuides]
# need to remove the negative_control factor
geneIds = factor(geneIds, levels = unique(geneIds))
x = data.frame(gamma = c(gamma, negCtrl),
               category = c(rep("gene targeting", times = length(gamma)),
                            rep("negative control", times = length(negCtrl))))
x$category = factor(x$category, levels = c("negative control", "gene targeting"))
library(ggplot2)
ggplot(x, aes(x = gamma, colour = category)) + geom_density() + theme_bw() +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(), axis.line = element_lin
```

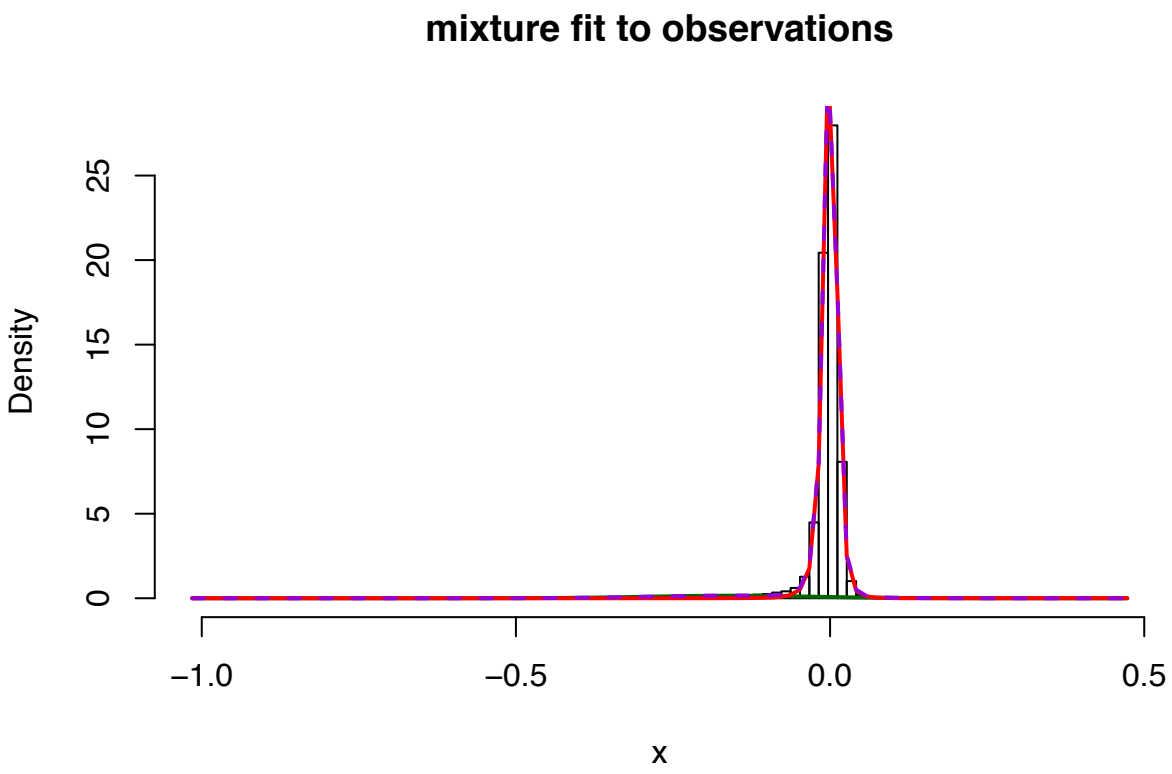


We see from the figure above that most of the gene targeting guides follow the same distribution as the negative control guides, but with a longer tail on the negative end. This is the signal due to cells dying as a result of the gene effects. We can now apply CRISPhieRmix to this data to identify genes that led to decreased growth or cell death.

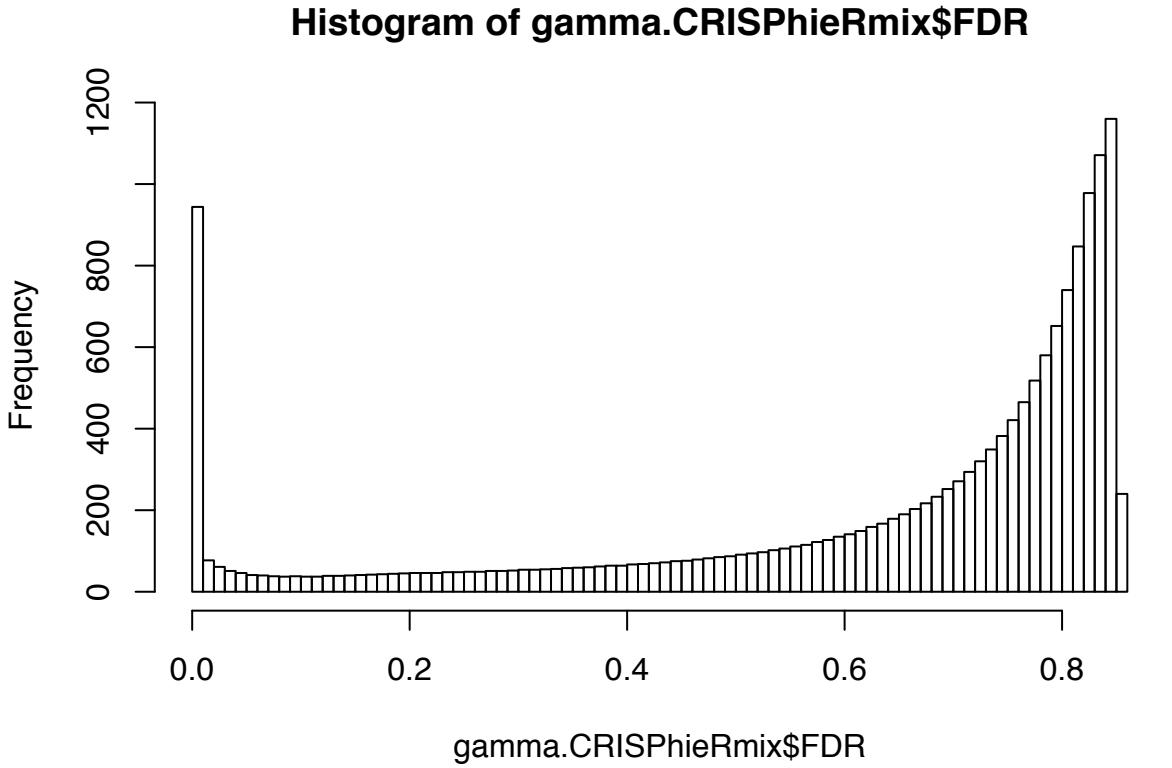
```
library(CRISPhieRmix)
gamma.CRISPhieRmix = CRISPhieRmix(x = gamma, geneIds = geneIds, negCtrl = negCtrl,
                                  mu = -0.2, sigma = 0.1, VERBOSE = TRUE, PLOT = TRUE)
```




```
## fit negative control distributions
## 2 groups
## EM converged
## mu =  -0.1575749
## sigma =  0.1315886
## pq =  0.04938287
```




```
hist(gamma.CRISPhieRmix$FDR, breaks = 100)
```




```
sum(gamma.CRISPhieRmix$FDR < 0.1)
```

```
## [1] 1373
```

The peak near zero indicates the genes that are likely to be essential. Let's look at how much these genes overlap the gold standard list of core constitutive genes and non-essential genes of Hart et al. 2014.

```
ConstitutiveCoreEssentialGenes = scan("ConstitutiveCoreEssentialGenes.txt", what = character())
length(ConstitutiveCoreEssentialGenes)
```

```
## [1] 217
```

```
length(intersect(ConstitutiveCoreEssentialGenes,
                 gamma.CRISPhieRmix$genes[which(gamma.CRISPhieRmix$FDR < 0.1)]))
```

```
## [1] 170
```

```
NonEssentialGenes = scan("NonEssentialGenes.txt", what = character())
length(NonEssentialGenes)
```

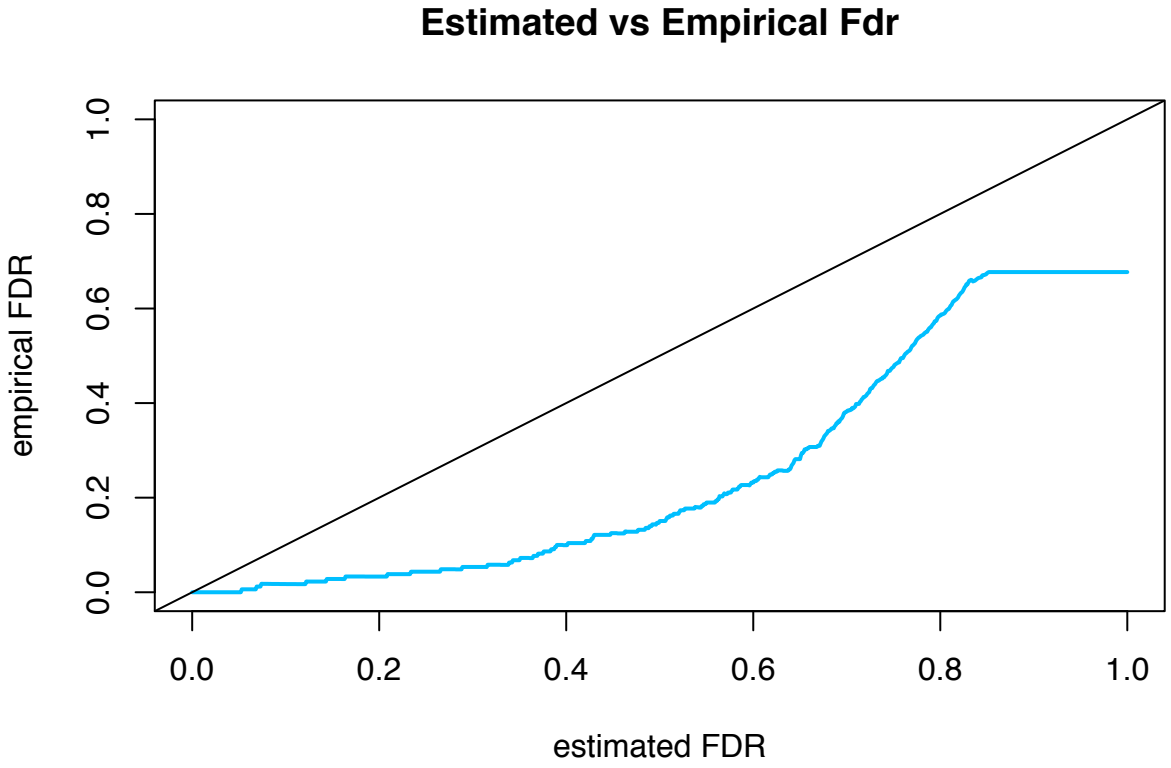
```
## [1] 927
```

```
length(intersect(NonEssentialGenes, gamma.CRISPhieRmix$genes[which(gamma.CRISPhieRmix$FDR < 0.1)]))
```

```
## [1] 3
```

From this we can estimate the empirical false discovery rate at FDR level 0.1 as $3/1373 \approx 0.002$ and an empirical true positive rate as $170/217 \approx 0.78$. The empirical vs estimated FDR curve is plotted below.

```
EssentialGenes = data.frame(gene = factor(c(sapply(ConstitutiveCoreEssentialGenes, toString),
                                            sapply(NonEssentialGenes, toString))),
                            essential = c(rep(1, times = length(ConstitutiveCoreEssentialGenes)),
                                          rep(0, times = length(NonEssentialGenes))))
EssentialGenes = EssentialGenes[which(EssentialGenes$gene %in% gamma.CRISPhieRmix$genes), ]
gamma.CRISPhieRmixEssential = data.frame(genes = gamma.CRISPhieRmix$genes, FDR = gamma.CRISPhieRmix$FDR)
gamma.CRISPhieRmixEssential = gamma.CRISPhieRmixEssential[which(gamma.CRISPhieRmixEssential$genes %in% E
gamma.CRISPhieRmixEssential = gamma.CRISPhieRmixEssential[match(EssentialGenes$gene, gamma.CRISPhieRmixE
fdr.curve <- function(thresh, fdrs, baseline){
  w = which(fdrs < thresh)
  if(length(w) > 0){
    return(sum(1 - baseline[w])/length(w))
  }
  else{
    return(NA)
  }
}
s = seq(from = 0, to = 1, length = 1001)
gamma.CRISPhieRmixFdrCurve = sapply(s, function(t) fdr.curve(t, gamma.CRISPhieRmixEssential$FDR,
                                                             EssentialGenes$essential))
plot(c(0, s[!is.na(gamma.CRISPhieRmixFdrCurve)]), c(0, gamma.CRISPhieRmixFdrCurve[!is.na(gamma.CRISPhieR
     ylab = "empirical FDR", main = "Estimated vs Empirical Fdr", xlim = c(0, 1), ylim = c(0, 1),
     lwd = 2, col = "deepskyblue")
abline(0, 1)
```

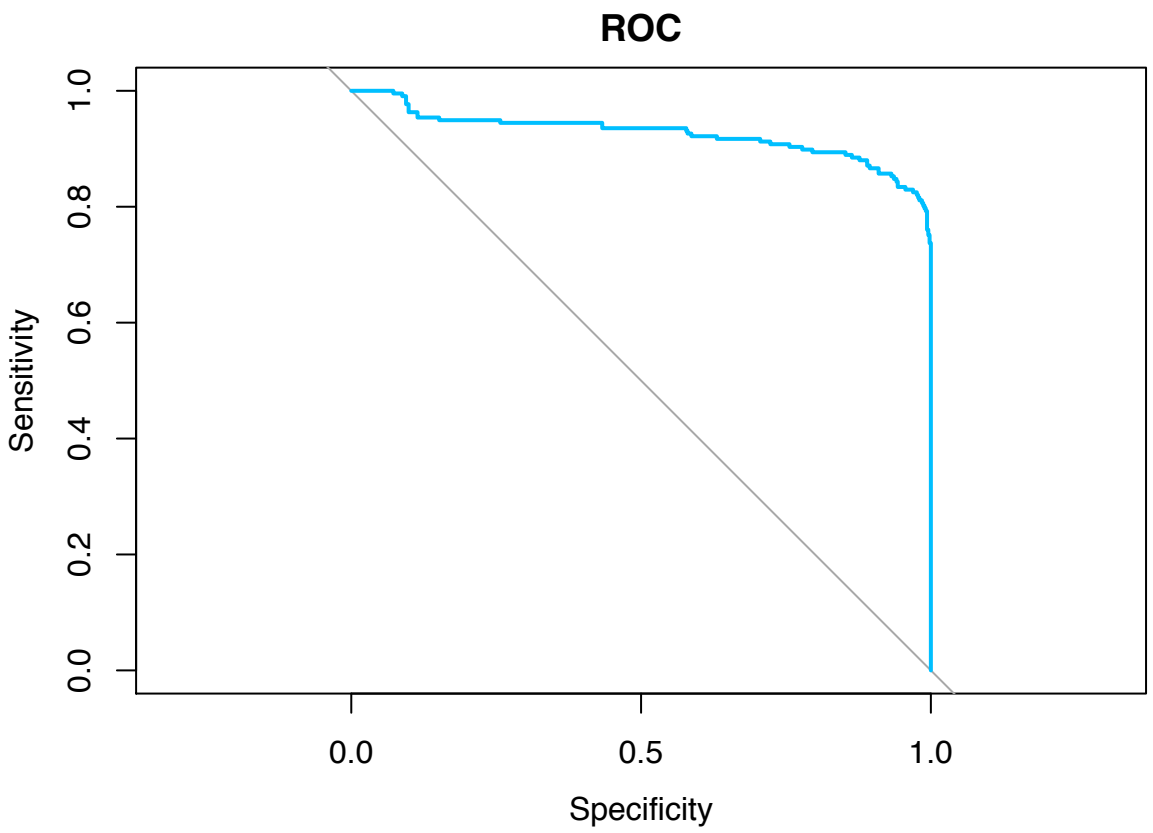


The receiver operator curve based on the above annotated genes is given below.

```
gamma.CRISPhieRmixROC = pROC::roc(EssentialGenes$essential,
                                  gamma.CRISPhieRmixEssential$FDR, auc = TRUE)
gamma.CRISPhieRmixROC
```

```
##
## Call:
## roc.default(response = EssentialGenes$essential, predictor = gamma.CRISPhieRmixEssential$FDR,     au
##
## Data: gamma.CRISPhieRmixEssential$FDR in 455 controls (EssentialGenes$essential 0) > 217 cases (Esse
## Area under the curve: 0.9259
```

```
plot(gamma.CRISPhieRmixROC, col = "deepskyblue", lwd = 2, xlim = c(0, 1), ylim = c(0, 1), main = "ROC")
```


ROC
Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0
Specificity

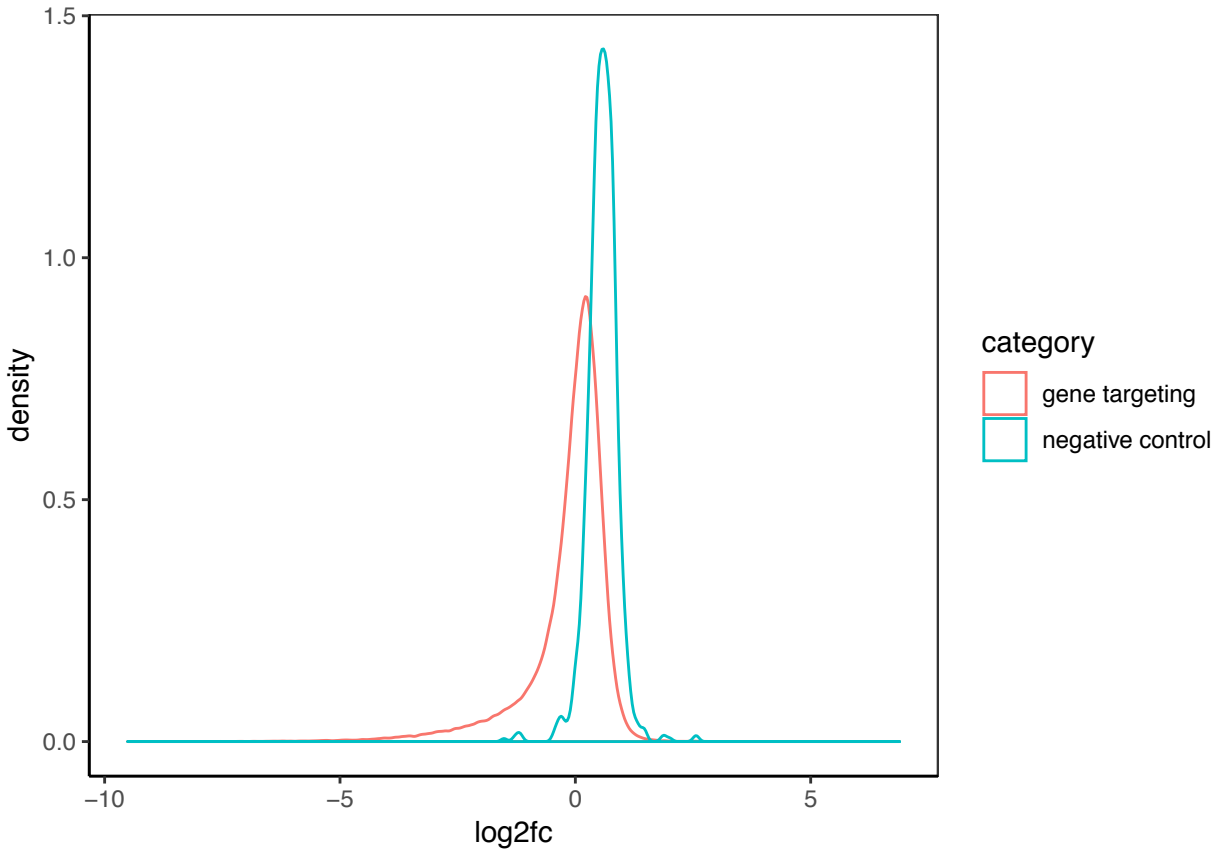
3.2 Checking the negative control distribution

The data shown here is taken from Wang et al, 2015. The authors performed a genome-wide screen for essential genes.

```
Wang2015counts = read.table(file = "aac7041_SM_Table_S2.txt", sep = "\t", header = TRUE)
Wang2015counts = Wang2015counts[-which(rowSums(Wang2015counts[ ,-c(1)]) == 0), ]
which.negCtrl = which(startsWith(sapply(Wang2015counts$sgRNA, toString), "CTRL"))
geneIds = sapply(Wang2015counts$sgRNA[-which.negCtrl],
                 function(g) unlist(strsplit(toString(g), split = "_"))[1])
geneIds = sapply(geneIds, function(g) substring(g, first = 3))
geneIds = factor(geneIds, levels = unique(geneIds))

counts = Wang2015counts[ , -c(1)]
colData = data.frame(cellType = sapply(colnames(counts),
                                       function(x) unlist(strsplit(toString(x), split = ".",
                                                                   fixed = TRUE))[1]),
                     condition = factor(rep(c(0, 1), times = 5)))
rownames(colData) = colnames(counts)
Wang2015DESeq = DESeq2::DESeqDataSetFromMatrix(countData = counts,
                                               colData = colData, design = ~ condition)
Wang2015DESeq = DESeq2::DESeq(Wang2015DESeq)
```

```
## estimating size factors

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing
```

```
Wang2015DESeq = DESeq2::results(Wang2015DESeq)
log2fc = Wang2015DESeq$log2FoldChange
log2fc.negCtrl = log2fc[which.negCtrl]
log2fc.geneTargeting = log2fc[-which.negCtrl]
library(ggplot2)
x = data.frame(log2fc = log2fc, category = c(rep("negative control",
                                                 times = length(which.negCtrl)),
                                             rep("gene targeting", times = length(log2fc.geneTargeting))
ggplot(x, aes(x = log2fc, colour = category)) + geom_density() + theme_bw() +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(), axis.line = element_lin
```



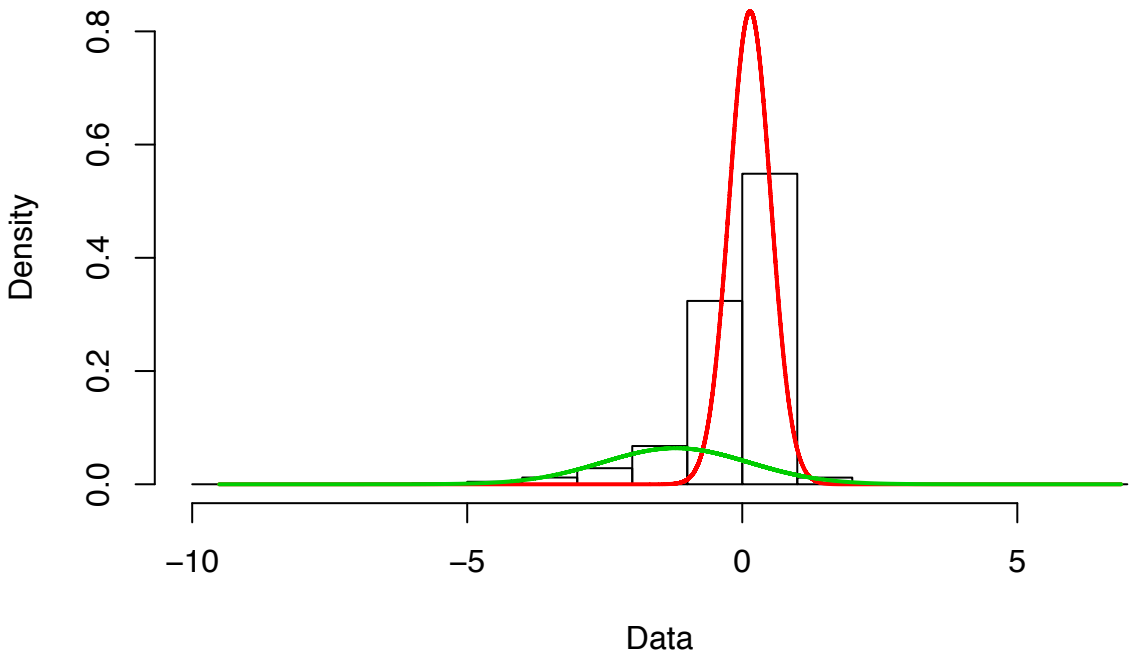
As we can see, the distribution of the negative control guides do not line up with the central peak of the gene targeting guides. This indicates that the distribution of the negative control guides does not reflect the null genes. In this case, using the negative control guides to estimate the null will likely lead to very incorrect inferences and is not suggested. It's unclear why the negative control guides do not look like the central peak. If this is the case in your experiment, it is worth investigating why.

3.3 Using a normal hierarchical mixture to rank genes without negative control guides

We'll look at ranking the genes without negative control guides for this data.

```
Wang2015NormalMix = CRISPhieRmix(x = log2fc.geneTargeting, geneIds = geneIds, PLOT = TRUE, VERBOSE = TRU
```

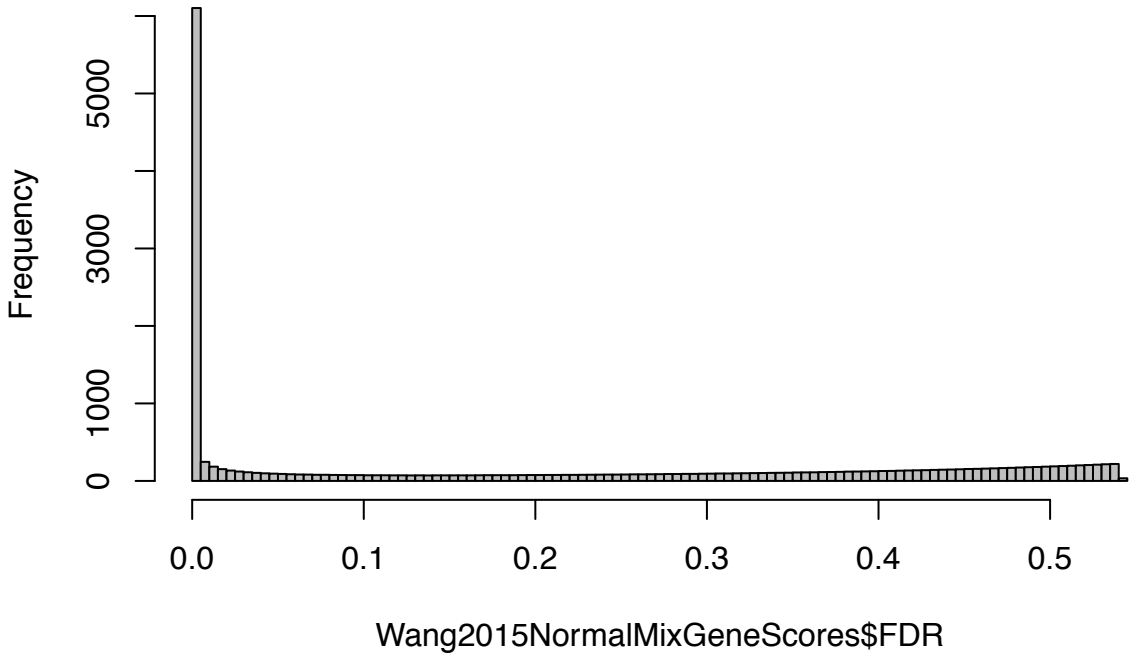
```
## no negative controls provided, fitting hierarchical normal model
## number of iterations= 52
```




```
Wang2015NormalMixGeneScores = data.frame(genes = Wang2015NormalMix$genes, FDR = Wang2015NormalMix$FDR)
head(Wang2015NormalMixGeneScores[order(Wang2015NormalMixGeneScores$FDR, decreasing = FALSE), ], 20)
```

```
##        genes FDR
## 7       AAAS   0
## 18      AAMP   0
## 22      AARS   0
## 23     AARS2   0
## 26  AASDHPPT   0
## 28      AATF   0
## 49     ABCB7   0
## 51     ABCB9   0
## 67     ABCE1   0
## 68     ABCF1   0
## 78    ABHD11   0
## 86   ABHD16B   0
## 90     ABHD2   0
## 100     ABL1   0
## 102   ABLIM1   0
## 108     ABT1   0
## 117    ACAD8   0
## 130    ACBD3   0
## 137      ACD   0
## 138      ACE   0
```

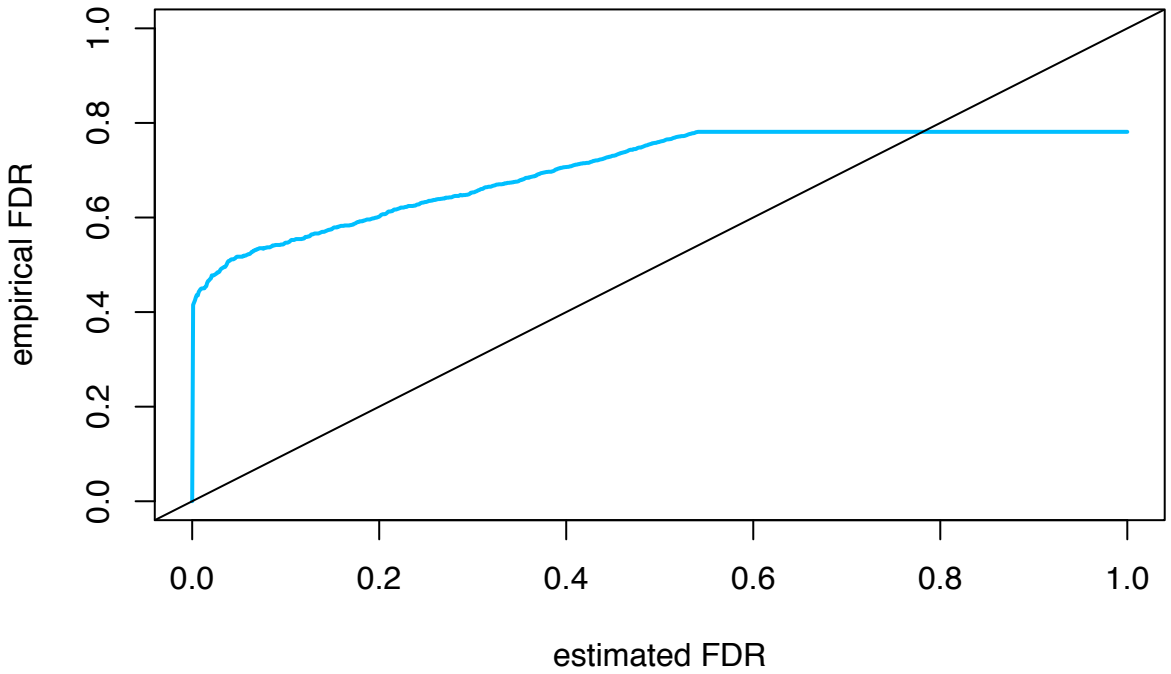
```
hist(Wang2015NormalMixGeneScores$FDR, breaks = 100, col = "grey")
```




```
sum(Wang2015NormalMixGeneScores$FDR == 0)
```

```
## [1] 2777
```

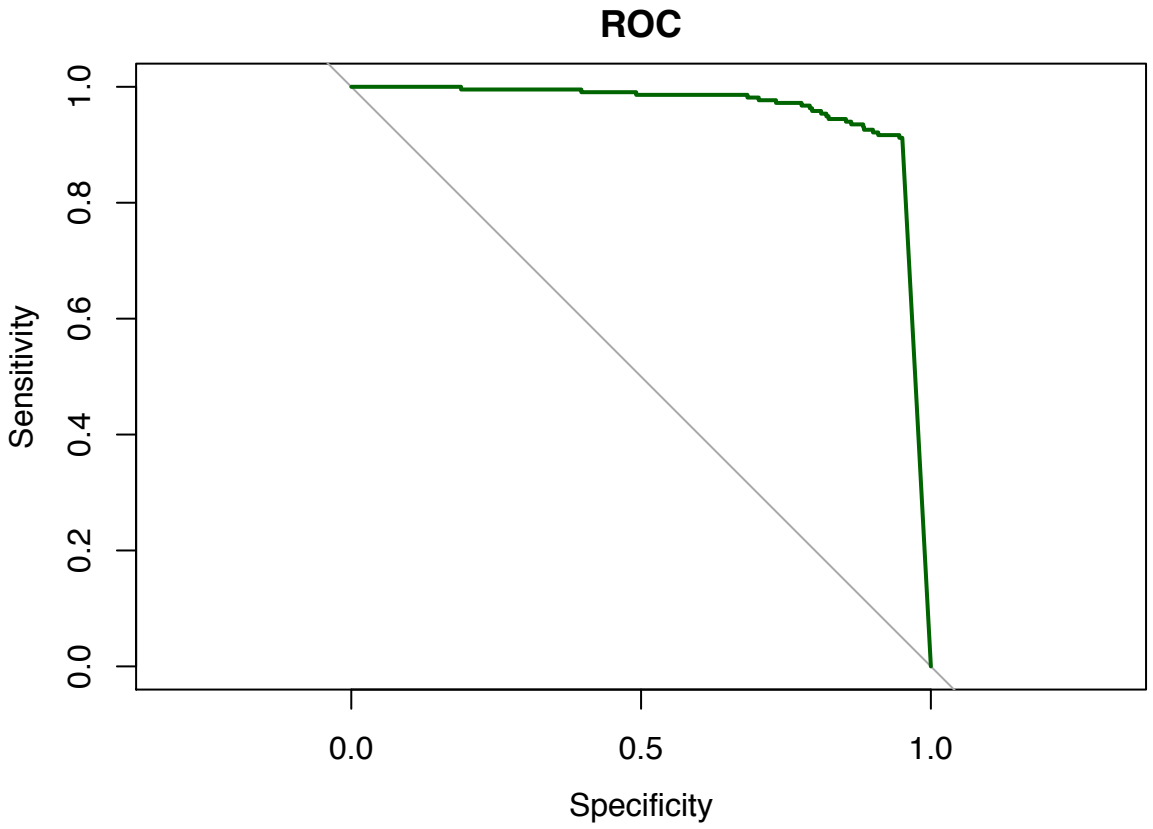
```
EssentialGenes = data.frame(gene = factor(c(sapply(ConstitutiveCoreEssentialGenes, toString),
                                            sapply(NonEssentialGenes, toString))),
                            essential = c(rep(1, times = length(ConstitutiveCoreEssentialGenes)),
                                          rep(0, times = length(NonEssentialGenes))))
EssentialGenes = EssentialGenes[which(EssentialGenes$gene %in% Wang2015NormalMixGeneScores$genes), ]
Wang2015NormalMixGeneScoresEssential = Wang2015NormalMixGeneScores[which(Wang2015NormalMixGeneScores$ge
Wang2015NormalMixGeneScoresEssential = Wang2015NormalMixGeneScoresEssential[match(EssentialGenes$gene,

Wang2015NormalMixGeneScoresEssentialFdrCurve = sapply(s, function(t) fdr.curve(t,
                                                                                Wang2015NormalMixGeneScoresEssentia
                                                                               EssentialGenes$essential))
plot(c(0, s[!is.na(Wang2015NormalMixGeneScoresEssentialFdrCurve)]), c(0, Wang2015NormalMixGeneScoresEss
     ylab = "empirical FDR", main = "Estimated vs Empirical Fdr", xlim = c(0, 1), ylim = c(0, 1),
     lwd = 2, col = "deepskyblue")
abline(0, 1)
```




```
Wang2015NormalMixGeneScoresEssentialROC = pROC::roc(EssentialGenes$essential,
                                    Wang2015NormalMixGeneScoresEssential$FDR, auc = TRUE)
Wang2015NormalMixGeneScoresEssentialROC
```

```
##
## Call:
## roc.default(response = EssentialGenes$essential, predictor = Wang2015NormalMixGeneScoresEssential$FDR
##
## Data: Wang2015NormalMixGeneScoresEssential$FDR in 771 controls (EssentialGenes$essential 0) > 216 cas
## Area under the curve: 0.9555
```

```
plot(Wang2015NormalMixGeneScoresEssentialROC, col = "darkgreen", lwd = 2, xlim = c(0, 1), ylim = c(0, 1
```



As we can see, the normal hierarchical mixture model is calling way too many genes as significant, but does a good job at ranking the essential genes ahead of the non-essential genes. We see that it calls way too many genes are essential at any level of FDR. This will be problematic for most applications, as we can't test all the genes called positive.

One strategy to use a broader tailed that is estimated from the central peak, as discussed in Efron, 2012. The idea is to use a semi-parametric distribution to estimate the null distribution from the central peak. We have found that this does not well approximate the negative control distribution because of the long tails, but it does a better job than the normal distribution. We have also found that for some data it is very difficult to get a approximation of the central peak, so we have not implemented this in the package. We show it here for the interest of the reader.

3.4 Estimating gene effect sizes after filtering

One issue with CRISPhieRmix is that we only rank genes and compute false discovery rates. Unlike MAGeCK MLE, CRISPhieRmix is currently not able to compute gene effect sizes. The major problem is the identifiability issue. With off-targets and gene to gene variation in guide efficiency, it is difficult to consistently estimate gene effect sizes. But if the genes are first chosen to be likely non-null and then gene effect sizes are estimated, then it is relatively easy to estimate gene effect sizes. Below I will show one way of how using Rstan and setting informative priors from the full analysis, though there are other ways.

```
# redefine geneIds
geneIds = Gilbert2014Table2CRISPRi$gene[geneTargetingGuides]
# need to remove the negative_control factor
geneIds = factor(geneIds, levels = unique(geneIds))

gamma.CRISPhieRmix$mixFit$mu
```

```
## [1] -0.1575749
```

```
gamma.CRISPhieRmix$mixFit$sigma
```

```
## [1] 0.1315886
```

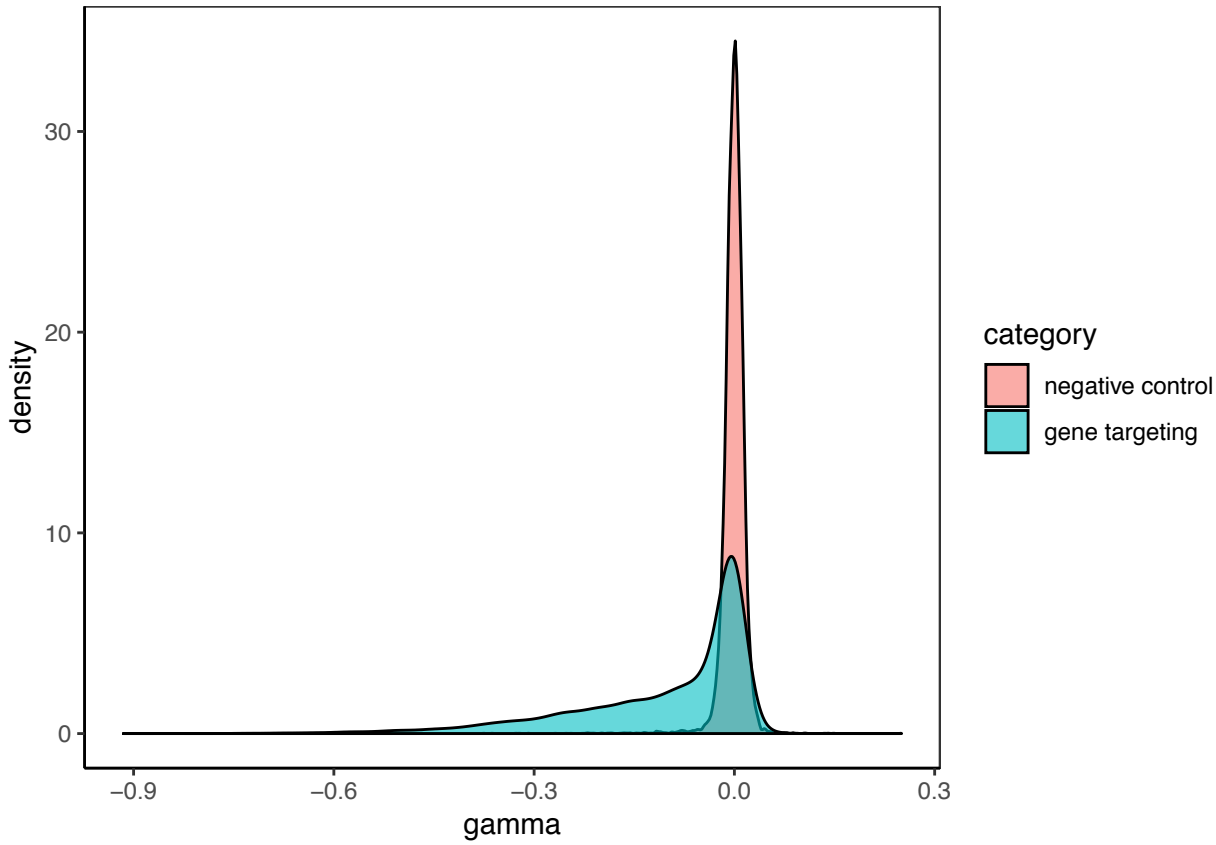
```
gamma.CRISPhieRmix$mixFit$pq
```

```
## [1] 0.04938287
```

```
topGenes = gamma.CRISPhieRmix$genes[which(gamma.CRISPhieRmix$FDR < 0.01)]
length(topGenes)/length(gamma.CRISPhieRmix$genes)
```

```
## [1] 0.05940469
```

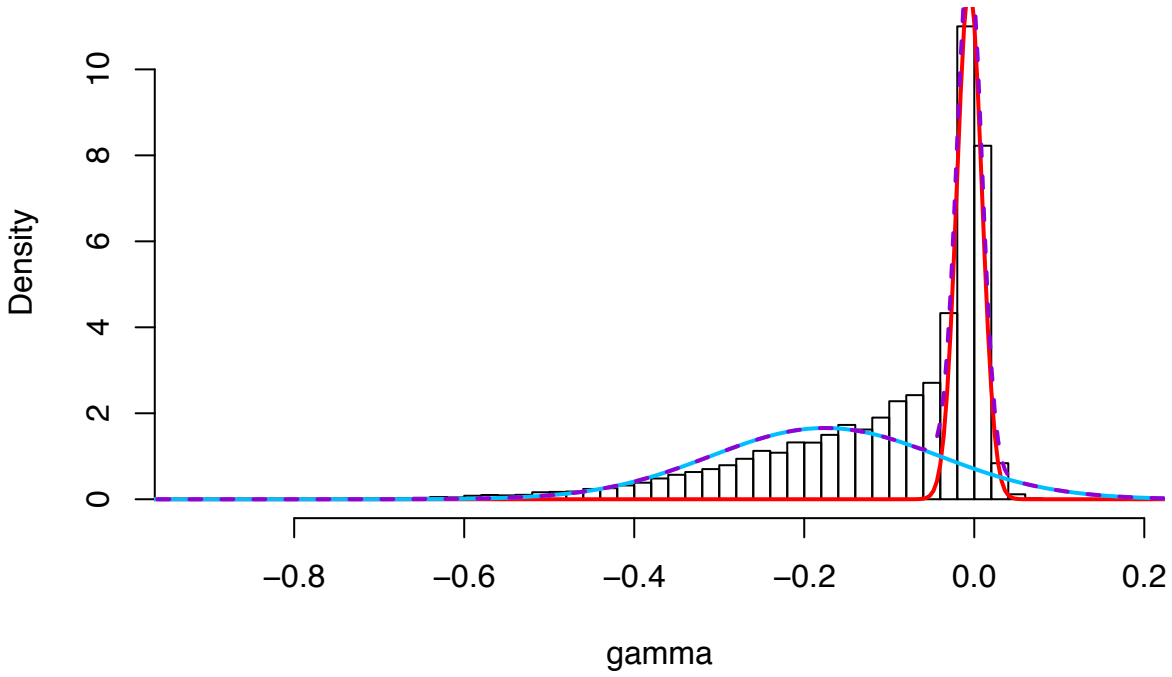
```
x = data.frame(gamma = c(gamma[which(geneIds %in% topGenes)], negCtrl),
               category = c(rep("gene targeting", times = sum(geneIds %in% topGenes)),
                            rep("negative control", times = length(negCtrl))))
x$category = factor(x$category, levels = c("negative control", "gene targeting"))
ggplot(x, aes(x = gamma, fill = category)) + geom_density(alpha = 0.6) + theme_bw() +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(), axis.line = element_lin
```




```
topGenesGammaMixFit = mixtools::normalmixEM(gamma[which(geneIds %in% topGenes)], k = 2, mu = c(0, -0.15)
```

```
## number of iterations= 55
```

```
hist(gamma[which(geneIds %in% topGenes)], breaks = 50, probability = TRUE, xlab = "gamma", main = "Histo
s = seq(from = -2, to = 2, length = 1001)
lines(s, topGenesGammaMixFit$lambda[1]*dnorm(s, mean = topGenesGammaMixFit$mu[1],
                                             sd = topGenesGammaMixFit$sigma[1]),
      col = "red", lwd = 2)
lines(s, topGenesGammaMixFit$lambda[2]*dnorm(s, mean = topGenesGammaMixFit$mu[2],
                                             sd = topGenesGammaMixFit$sigma[2]),
      col = "deepskyblue", lwd = 2)
lines(s, topGenesGammaMixFit$lambda[1]*dnorm(s, mean = topGenesGammaMixFit$mu[1],
                                             sd = topGenesGammaMixFit$sigma[1]) +
        topGenesGammaMixFit$lambda[2]*dnorm(s, mean = topGenesGammaMixFit$mu[2],
                                            sd = topGenesGammaMixFit$sigma[2]),
      col = "darkviolet", lwd = 2, lty = 2)
```



Using a normal distribution as the null distribution likely overestimates the gene effects. The skew-t can be used by storing the probabilities in a look-up table, but this is much slower. For this example, we will use a normal distribution for the null. We're going to assume the following hierarchical model with informative priors and fit it in stan.

$$\begin{aligned}
\gamma_i &\sim (1-\pi_{g_i})\mathcal{N}(0, 0.015^2) + \pi_{g_i}\mathcal{N}(\mu_{g_i}, \sigma_1^2);\\
\mu_g &\sim \mathcal{N}(-0.175, \sigma_2^2);\\
\pi_g &\sim \text{Uniform}(0,1);\\
\sigma_1, \sigma_2 &\sim N_+(0, 0.5^2).
\end{aligned}$$

```
model <-'
data {
  int<lower=1> n_sgRNAs;
  int<lower=1> n_genes;
  real x[n_sgRNAs]; // gamma
  int<lower=0, upper=n_genes> gene_ids[n_sgRNAs];
}
parameters {
  real mu_g[n_genes];  // gene effect sizes
  real<lower=0, upper=1> pi_g[n_genes]; // mixing parameters
  real<lower=0> sigma1; // guide-level sd
  real<lower=0> sigma2; // gene-level sd
}
model{
  sigma1 ~ normal(0, 0.5);
  sigma2 ~ normal(0, 0.5);
  mu_g ~ normal(-0.175, sigma2);
  pi_g ~ beta(1, 1);
  for(i in 1:n_sgRNAs){
    target += log_mix(pi_g[gene_ids[i]],
```

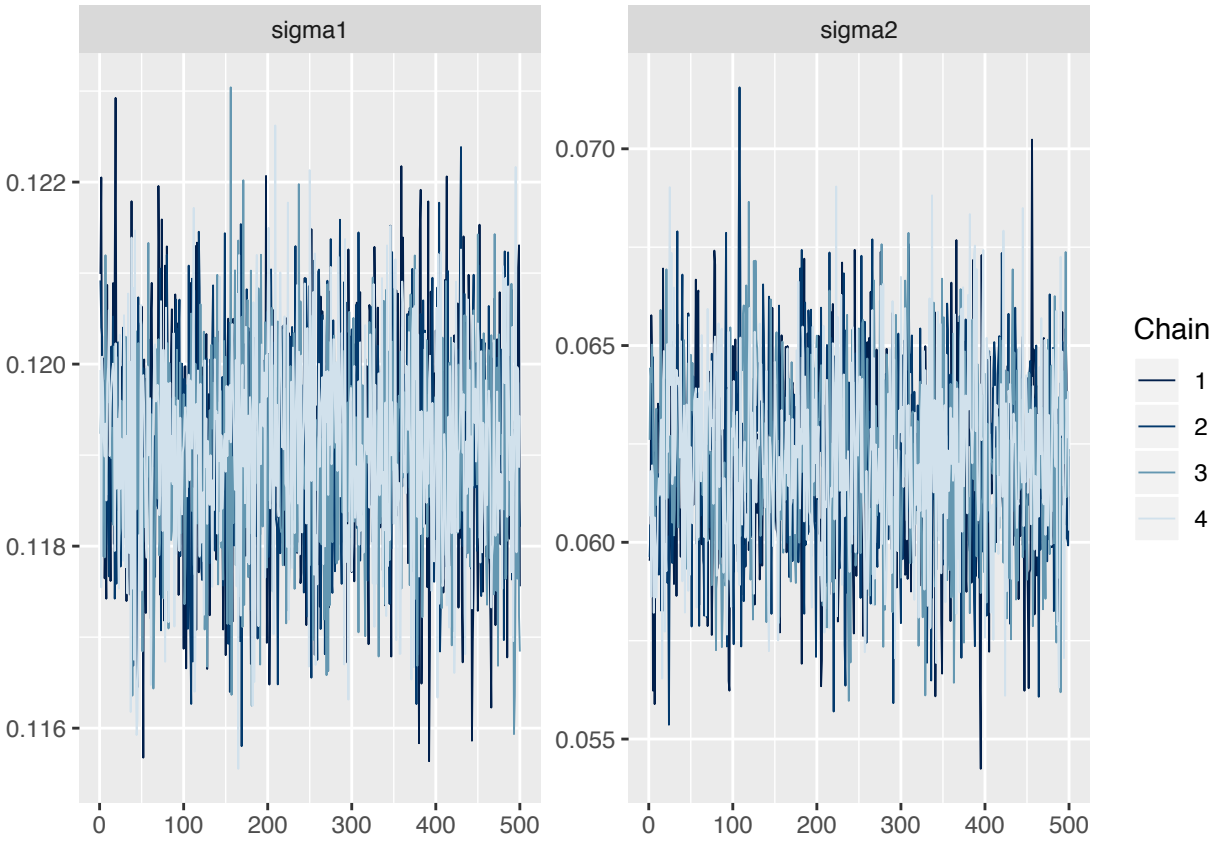
```
                    normal_lpdf(x[i] | mu_g[gene_ids[i]], sigma1),
                    normal_lpdf(x[i] | 0, 0.015));
  }
}
'
library(rstan)
options(mc.cores = 2)
gene_ids = geneIds[which(geneIds %in% topGenes)]
gene_ids = factor(gene_ids, levels = unique(gene_ids))
sgRNAdata = list(n_sgRNAs = length(gene_ids),
                 n_genes = length(levels(gene_ids)),
                 x = gamma[which(geneIds %in% topGenes)],
                 gene_ids = as.numeric(gene_ids))
gammaHierMixFit = stan(model_code = model, data = sgRNAdata, chains = 4, iter = 1000);
```

```
## Warning: There were 46 transitions after warmup that exceeded the maximum treedepth. Increase max_tr
## http://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
gammaHierMixFit.stan.summary = summary(gammaHierMixFit, probs = c(0.05, 0.5, 0.95))$summary
```

```
# trace plot for sigmas
bayesplot::mcmc_trace(as.array(gammaHierMixFit), pars = c("sigma1", "sigma2"))
```

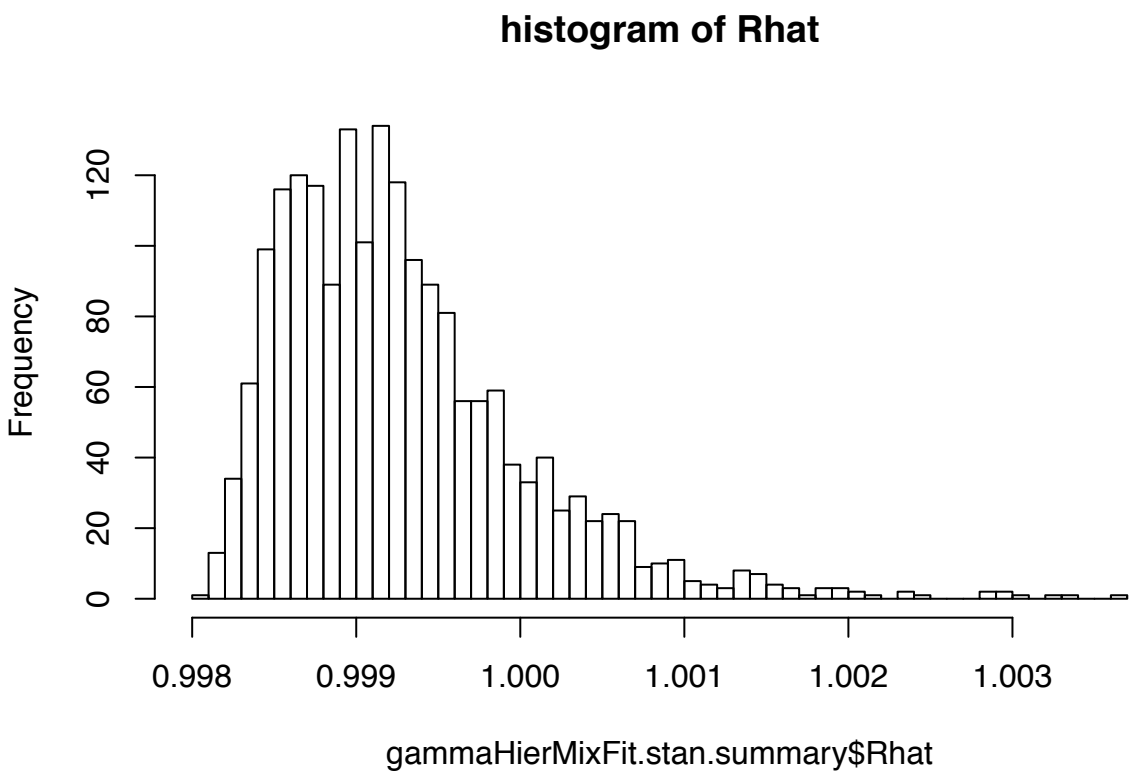



```
gammaHierMixFit.stan.summary =
  data.frame(parameter = as.factor(c(paste0("mu[", levels(gene_ids), "]"),
                                     paste0("pi[", levels(gene_ids), "]"),
```

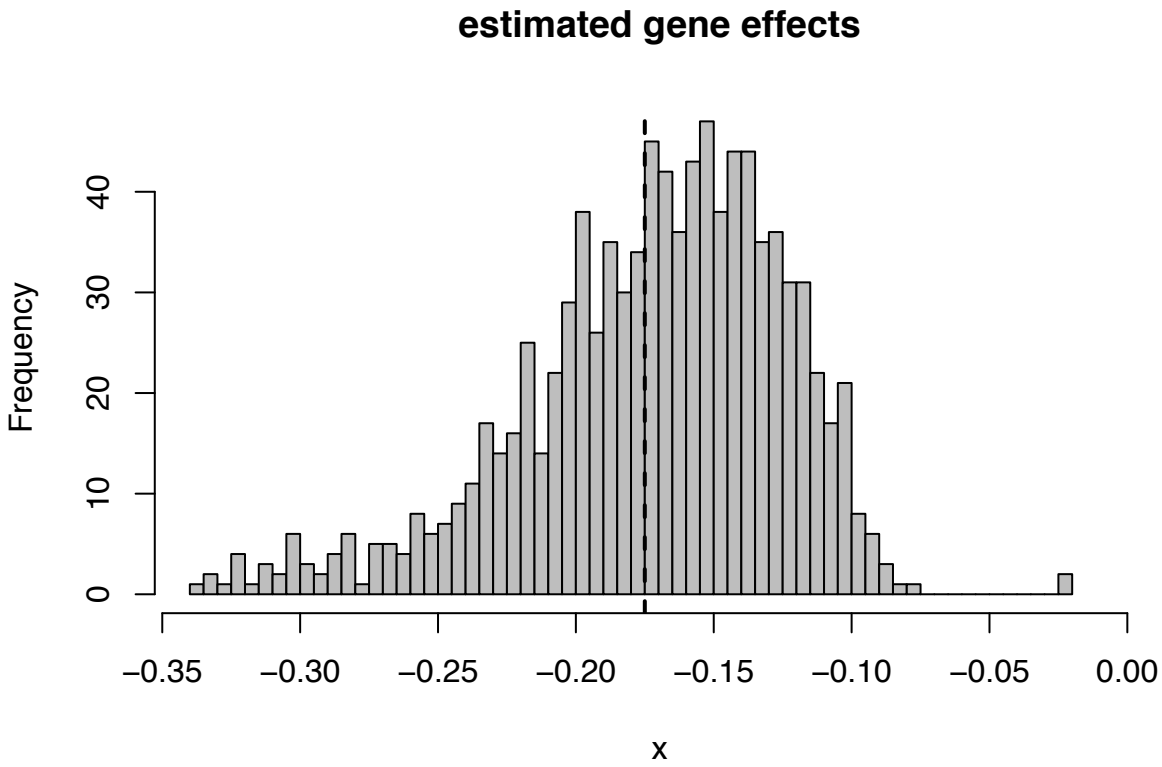
```
                                    "sigma1", "sigma2", "lp__")), gammaHierMixFit.stan.summary)
rownames(gammaHierMixFit.stan.summary) = c()
tail(gammaHierMixFit.stan.summary, 3)
```

```
##      parameter         mean      se_mean           sd          X5.
## 1889    sigma1 1.190777e-01 2.551200e-05  0.001140931 1.171315e-01
## 1890    sigma2 6.202982e-02 5.731784e-05  0.002284655 5.826072e-02
## 1891      lp__ 1.366123e+04 1.388832e+00 35.850207884 1.360002e+04
##              X50.         X95.     n_eff     Rhat
## 1889 1.190947e-01 1.209164e-01 2000.0000 0.999373
## 1890 6.199475e-02 6.580674e-02 1588.7728 1.000609
## 1891 1.366357e+04 1.371656e+04  666.3221 1.002904
```

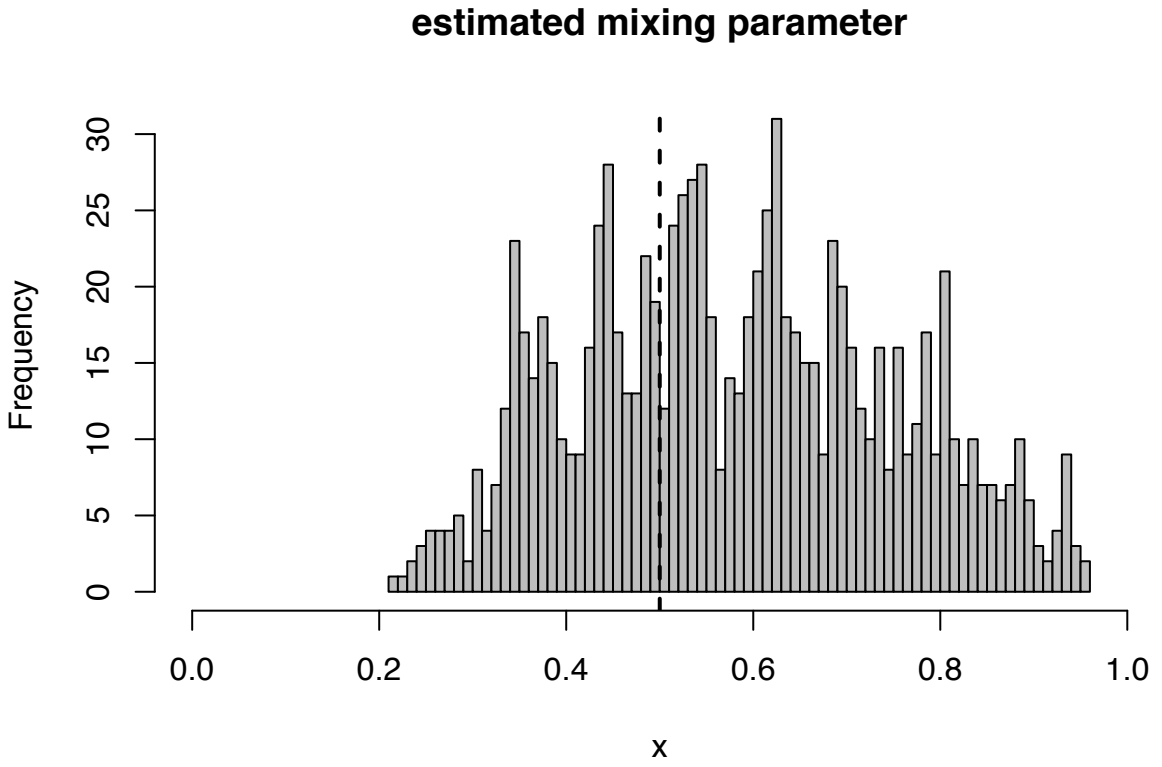
```
hist(gammaHierMixFit.stan.summary$Rhat, breaks = 50, main = "histogram of Rhat")
```




```
gammaIndices = grep("mu", gammaHierMixFit.stan.summary$parameter)
x = gammaHierMixFit.stan.summary$X50.[gammaIndices]
hist(x, breaks = 100, xlim = c(min(x), 0), col = "grey", main = "estimated gene effects")
abline(v = -0.175, lwd = 2, lty = 2)
```



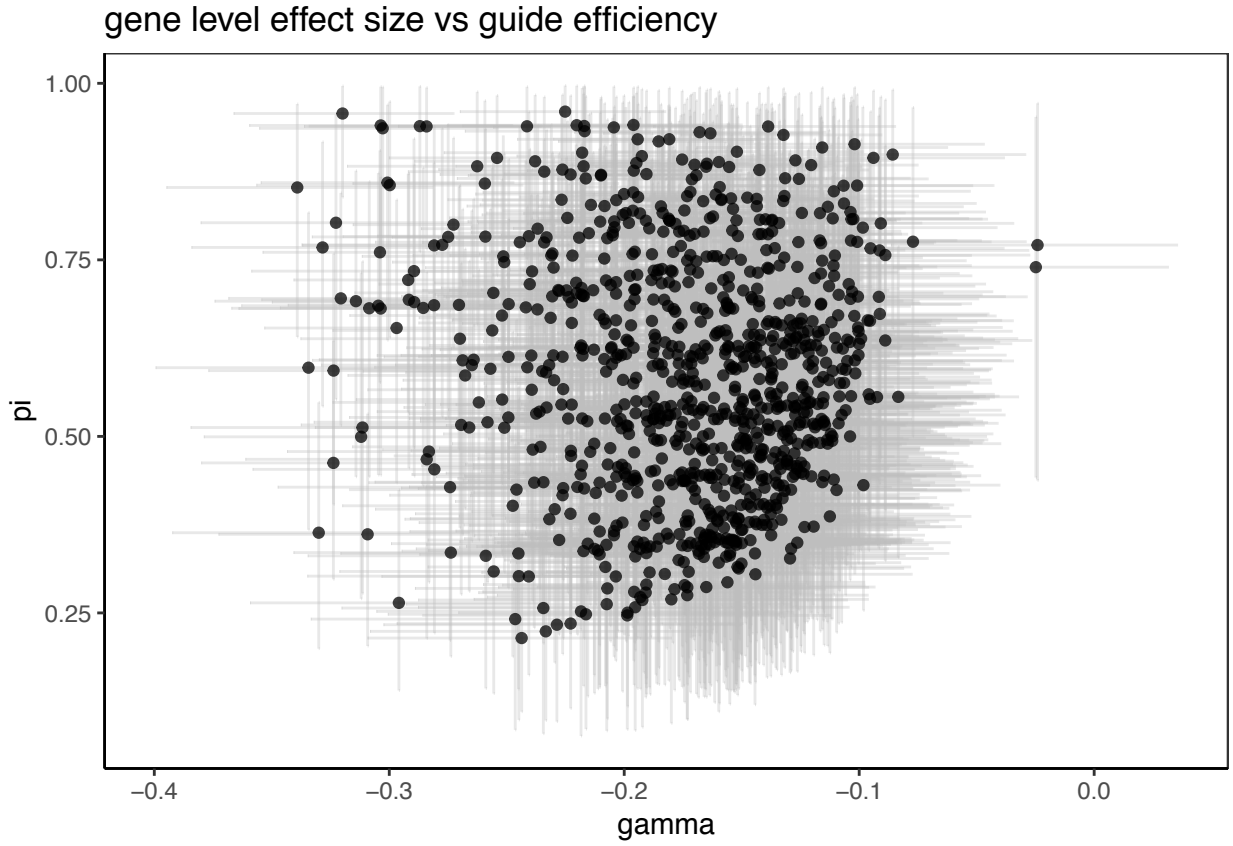

```
mixingIndices = grep("pi", gammaHierMixFit.stan.summary$parameter)
x = gammaHierMixFit.stan.summary$X50.[mixingIndices]
hist(x, breaks = 100, xlim = c(0, 1), col = "grey", main = "estimated mixing parameter")
abline(v = 0.5, lwd = 2, lty = 2)
```



```
y = data.frame(genes = levels(gene_ids),
               geneGamma = gammaHierMixFit.stan.summary$X50.[gammaIndices],
               geneGammaLowerCI = gammaHierMixFit.stan.summary$X5.[gammaIndices],
               geneGammaUpperCI = gammaHierMixFit.stan.summary$X95.[gammaIndices],
               geneMixing = gammaHierMixFit.stan.summary$X50.[mixingIndices],
               geneMixingLowerCI = gammaHierMixFit.stan.summary$X5.[mixingIndices],
               geneMixingUpperCI = gammaHierMixFit.stan.summary$X95.[mixingIndices])
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.4.4
```

```
ggplot(y, aes(x = geneGamma, y = geneMixing))  +
  geom_errorbar(aes(ymin = geneMixingLowerCI, ymax = geneMixingUpperCI), colour = "grey", alpha = 0.35)
  geom_errorbarh(aes(xmin = geneGammaLowerCI, xmax = geneGammaUpperCI), colour = "grey", alpha = 0.35) +
  theme_bw() + theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(), axis.line =
  geom_point(alpha = 0.75) + xlab("gamma") + ylab("pi") + ggtitle("gene level effect size vs guide effi
```



The trace plots and Rhat's near 1 means the model is likely converged. We see that when the gene effect size is far away from zero, mixing parameters can vary a lot. On the other hand, when the gene effect size is near zero it is more difficult to find genes with low mixing parameters. This is likely because genes with small effects and low mixing parameters are chosen by CRISPhieRmix to less likely be null.