DiffDist vignette

DiffDist uses a hierarchical model based on the negative binomial distribution to perform tests of differential expression, dispersion and distribution for RNA-seq data. An adaptive Markov chain Monte Carlo (MCMC) algorithm is used to obtain posterior samples for each parameter of interest, and these samples are used to test for differences between groups in either the mean or dispersion parameters separately, or an overall test of differential distribution – a difference in either parameter for a given gene.

The main function to run the MCMC algorithm is `ln_hmm_adapt_3_chains()`. The output is a list containing `mcmc.list` objects (from the `coda` package) for each parameter from the hierarchical model. Below is an example showing how to use the output to perform tests of differential expression, dispersion and distribution. Data for the example is generated using the `compcodeR` package. DiffDist does not perform normalisation. Normalised counts must be provided. In this example, we generate normalisation factors using the TMM method incorporated in edgeR, and use those to generate normalised counts.

```
library(DiffDist)
library(compcodeR)
library(edgeR)
```

```
dat <-  generateSyntheticData('data', n.vars=100, samples.per.cond=5, n.diffexp=10)
# 100 genes, 5 samples per group, differential expression for first 10 genes

libsizes <- colSums(dat@count.matrix) # library sizes
nf <- calcNormFactors(dat@count.matrix, method="TMM") # generate normalisation factors
els <- nf * libsizes # effective library sizes
sf <- els / exp(mean(log(libsizes))) # generate 'size factors' from effective library sizes
norm.counts <- t(t(dat@count.matrix) / sf) # divide raw counts by size factors

res <- ln_hmm_adapt_3_chains(counts=norm.counts,
                             groups=dat@sample.annotations$condition)
#> initialising...
#> optimising proposal distributions...
#> running mcmc...
```

`res` is a list containing `mcmc.list` objects with posterior samples for each parameter:

```
names(res)
#>  [1] "chain.length"                "adaptive.runs"
#>  [3] "mean.proposal.scales0"       "mean.proposal.scales1"
#>  [5] "mean.proposal.scales2"       "disp.proposal.scales0"
#>  [7] "disp.proposal.scales1"       "disp.proposal.scales2"
#>  [9] "mean.prior.scale.proposal.sd" "disp.prior.scale.proposal.sd"
#> [11] "means0"                      "means1"
#> [13] "means2"                      "disps0"
#> [15] "disps1"                      "disps2"
#> [17] "mean.prior.location"         "disp.prior.location"
#> [19] "mean.prior.scale"            "disp.prior.scale"
#> [21] "indicators"                  "proportion"
```

For differential expression and dispersion tests, we are mainly interested in `means1` and `means2`, and `disps1` and `disps2`, respectively. These are the posterior samples for each parameter in each group. The tests are based on a posterior sample of log-transformed differences in each parameter between groups – i.e. a posterior sample of the log fold change (LFC) for each parameter:

```
mean.diff <- unname(log2(as.matrix(res$means1)) - log2(as.matrix(res$means2)))
disp.diff <- unname(log2(as.matrix(res$disps1)) - log2(as.matrix(res$disps2)))
```

Given a posterior sample of LFC, `hpd.pval()` computes a tail probability of differential expression or dispersion. By default `hpd.pval()` tests for any difference between groups, but the parameter `m` can be used to test for a difference at a minimum LFC:

```
p.mean <- apply(mean.diff, 2, hpd.pval) # Tests for differential expression
p.disp <- apply(disp.diff, 2, hpd.pval) # Tests for differential dispersion
p.mean.lfc1 <- apply(mean.diff, 2, hpd.pval, m=1)
# Tests for differential expression at minimum LFC 1
p.disp.lfc1 <- apply(disp.diff, 2, hpd.pval, m=1)
# Tests for differential dispersion at minimum LFC 1
```

It has been found in practice that applying false discovery rate control procedures on (1 – posterior tail probabilities) works well to set a threshold for declaring a gene as differentially expressed or dispersed.

The differential distribution test is based on a mixture model consisting of two components: one where a given gene is not differentially distributed (mean and dispersion are both the same in both groups), and one where it is differentially distributed (mean and/or dispersion differ between groups). An indicator variable is used to denote which mixture component each gene belongs to, and a posterior sample of the indicator variables is taken as a posterior sample of the probability of differential distribution for each gene.

```
prob <- unname(colMeans(as.matrix(res$indicators)))
```

Two alternative methods for setting a posterior probability threshold for declaring differential distribution may be considered: the Bayesian false discovery rate (BFDR), or what we term the posterior threshold method. The posterior threshold method takes advantage of the posterior sample of the proportion of differentially distributed genes that the model provides. An estimate of the proportion of differentially distributed genes is obtained from this sample, and a threshold probability is set to ensure that the corresponding number of genes is declared as differentially distributed.

```
# BFDR method:
bfdr.HMM <- bfdr(prob)

# Posterior threshold method:
proportion <- mean(as.matrix(res$proportion))
threshold <- sort(prob, decreasing=T)[round(nrow(dat@count.matrix) * proportion)]
# Genes with a posterior probability > threshold are declared as differentially
# distributed.
```