Example

Jinlu Liu

15/08/2022

We use the following example (same as Simulation 3 in the main article) to demonstrate the use of functions in the normHDP package. Suppose the dataset we have is named Y_sim, to run a short markov chain, we use a burn-in of $2,000$, thinning of 5 and total iteration of $5,000$:

```
##-- Run on 4 cores
MCMC_sample <- normHDP_mcmc(Y = Y_sim, # Name of Input data
                            J = 3, # Number of components
                            burn_in = 2000,
                            number_iter = 5000,
                            thinning = 5,
                            beta.mean = 0.68, # Mean of capture efficiencies
                            print_Z = TRUE, # Print a summary of Z after each iteration
                            num.cores = 4, # Run the function on 4 cores
                            alpha_mu_2 = 30
                            )
```

If we want to continue to run for more iterations, i.e.: to extend the total number of iterations from $5,000$ to $10,000$:

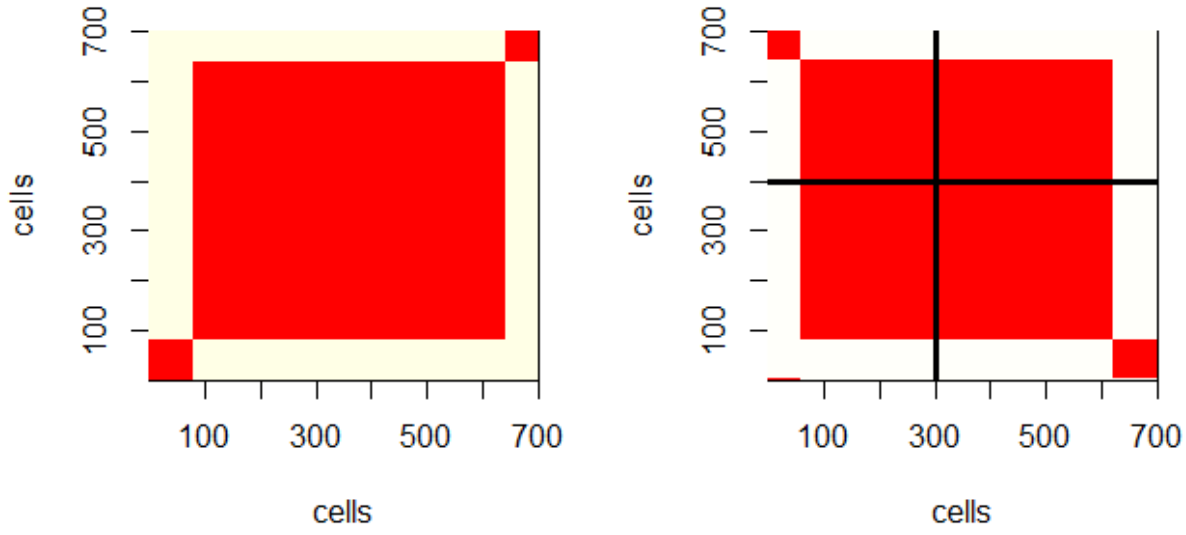
```
MCMC_sample2 <- normHDP_mcmc_2(normHDP_output = MCMC_sample,
                               number_iter = 10000)
```

To obtain the posterior similarity matrix:

```
psm <- similarity_matrix(normHDP_output = MCMC_sample)
```

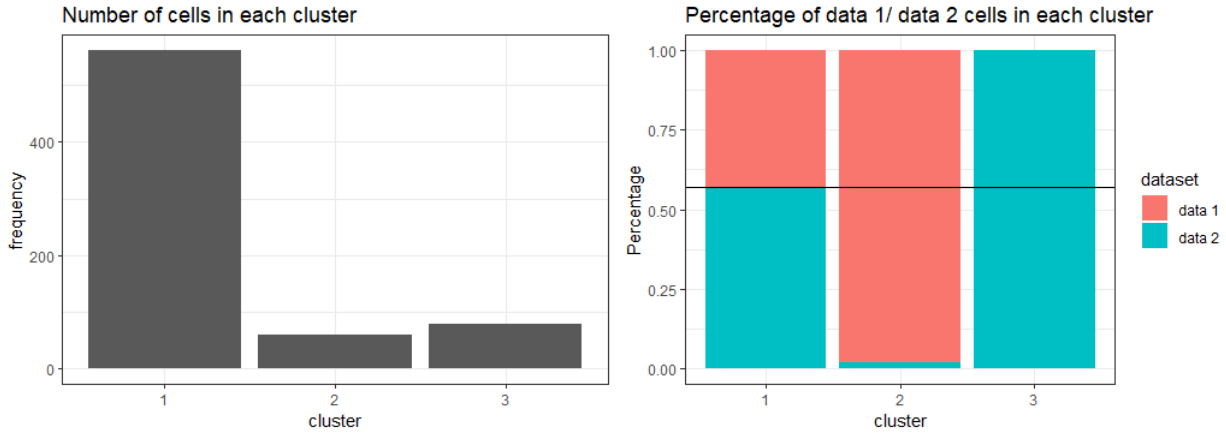
The output from similarity_matrix() contains a list of 2 items: the first item is the posterior similarity matrix between cells from 2 datasets or within the same dataset. The second item is the posterior similarity matrix between cells from all datasets. In plot below, the left sub-plot shows posterior similarity by combining cells from all datasets together; the right sub-plot shows posterior similarity between cells from the same dataset and between (any of the) 2 datasets.

```
plotpsm(psm.ind = psm$psm.within,
        psm.tot = psm$psm.combined,
        method="complete",
        xlab = 'cells',
        ylab = 'cells')
```



Next, we can summarize the posterior estimates of clustering using a single point estimate:

```
# Point estimate to minimize VI
Z_post <- opt.clust(normHDP_output = MCMC_sample)

# Use bar-chart to summarize this point estimate
cluster_summary(Z_point = Z_post)
```



Since we have the label-switching problem for the previous MCMC, we cannot carry out analysis on the unique parameters (mean expressions and dispersion) by simply taking the posterior mean or median, instead, we analyze the posterior estimates of mean expression and dispersion by running a separate MCMC chain for the unique parameters based on the posterior estimates of allocations and capture efficiencies:

```
MCMC_sample_post <- normHDP_mcmc_post(normHDP_output = MCMC_sample,
                                      burn_in = 1000,
                                      thinning = 5,
                                      number_iter = 3000,
                                      Z = Z_post,
                                      Y = Y_sim,
                                      iter_update = 100)

##-- Length of posterior samples
length.mcmc <- length(MCMC_sample_post$mu_star_1_J_output)

##-- Posterior mean of mean expression with fixed allocation
post.mean.mu <- Reduce("+", MCMC_sample_post$mu_star_1_J_output)/length.mcmc

##-- Posterior mean of dispersion with fixed allocation
post.mean.phi <- Reduce("+", MCMC_sample_post$phi_star_1_J_output)/length.mcmc

##-- Posterior mean of Beta
post.mean.beta <- MCMC_sample_post$Beta_posterior_mean
```

Similar to the normHDP_mcmc function(), we can extend the total number of iterations using:

```
MCMC_sample_post2 <- normHDP_mcmc_post2(normHDP_post_output = MCMC_sample_post,
                                        number_iter = 5000)
```

The structure of the output from normHDP_mcmc_post2() and normHDP_mcmc_post() are the same.

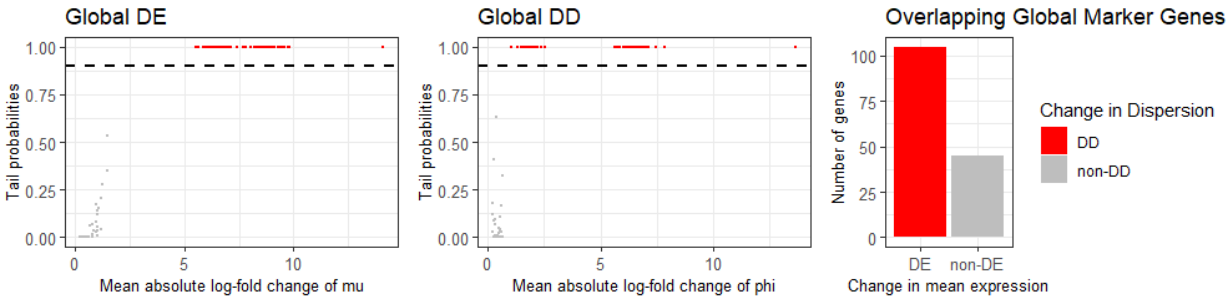
Global Marker genes

To find global marker genes, we use the function global_marker_genes() where we use the output from normHDP_post_output() as an input and pre-specify a set of threshold for determining global marker genes, example below uses 1.5 for both mean expressions and dispersions. We can also pre-specify alpha_M and alpha_D (example below). By default, alpha_M and alpha_D are set to control the expected false discovery rate (EFDR) to 5 percent. The output of the function includes four items:

- marker_DE: a data frame with four variables: the gene index, the absolute log-fold change (μ) of the gene, tail-probability (μ) of the gene, a binary variable to indicate whether the gene is globally differentially expressed (DE).
- marker_DD: a data frame with four variables: the gene index, the absolute log-fold change (ϕ) of the gene, tail-probability (ϕ) of the gene, a binary variable to indicate whether the gene is globally differentially dispersed (DD).
- alpha_M: threshold for the tail probabilities of μ. Genes with tail probabilities greater than alpha_M are classified as global DE genes.
- alpha_D: threshold for the tail probabilities of ϕ. Genes with tail probabilities greater than alpha_D are classified as global DD genes.

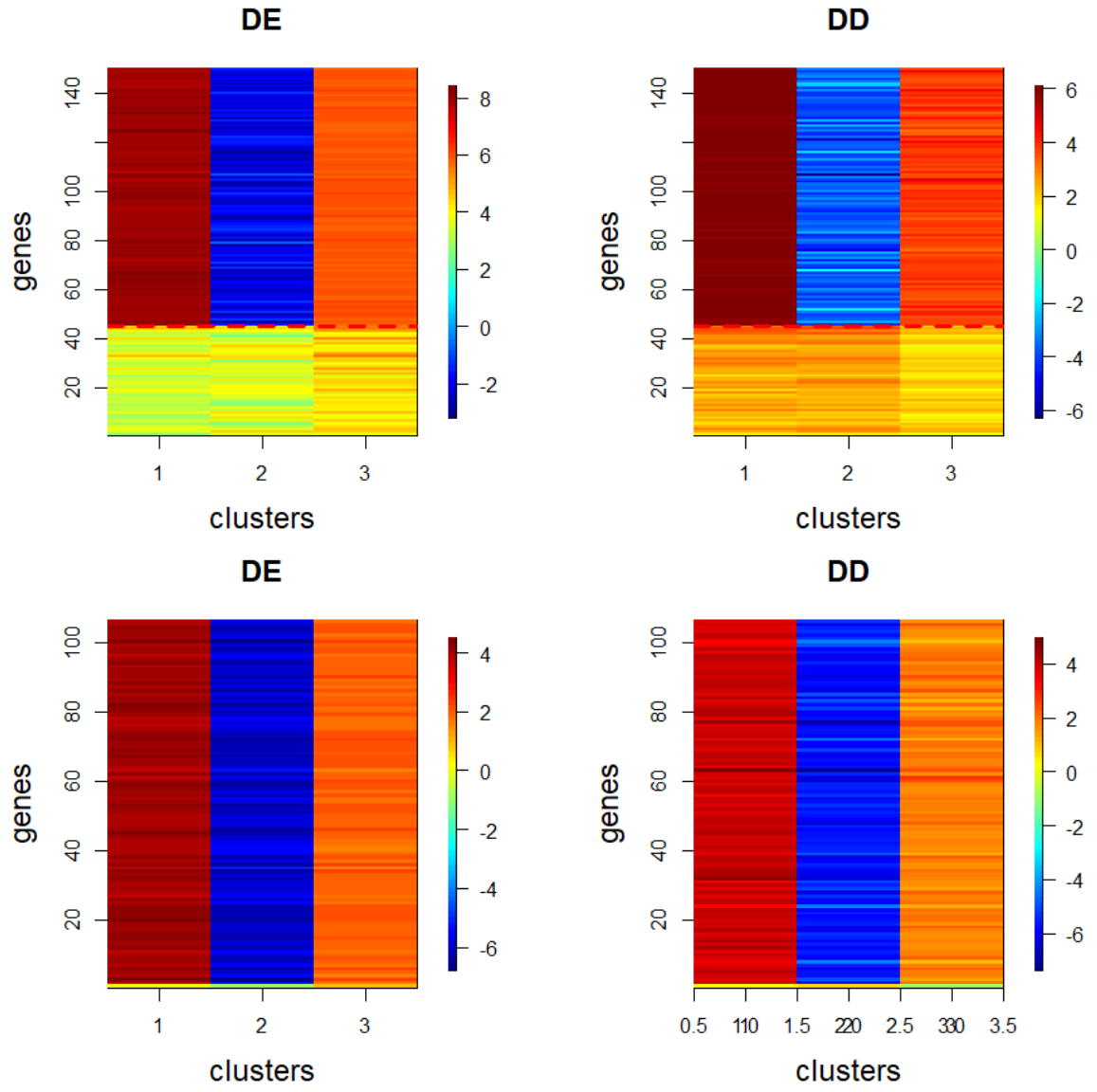
The summary plot contains three sub-figures. The sub-figure on the left compares the tail probability against the mean absolute log-fold change for the mean expressions. The sub-figure in the middle compares the tail probability against the mean absolute log-fold change for dispersion. The sub-figure on the right summarizes the total number of genes with these global characteristics.

```
threshold.example <- list('mu' = 1.5, 'phi' = 1.5)

# Example with given alpha_M and alpha_D
gmg_output_test <- global_marker_genes(normHDP_post_output = MCMC_sample_post,
                                       threshold = threshold.example,
                                       alpha_M = 0.9,
                                       alpha_D = 0.9)

# To have the summary plot
global_marker_genes_plot(gmg_output = gmg_output_test)
```



Graph below shows posterior estimated mean expressions and dispersions with reordered genes (by increasing tail probabilities); for the first plot, genes above the horizontal dashed lines are classified as global marker genes, and vice versa. The second plot only contains global marker genes and standardized mean expressions and dispersions are shown (standardized to have zero mean across all clusters for each gene):

```
global_marker_genes_heatmap(gmg_output = gmg_output_test,
                            normHDP_post_output = MCMC_sample_post)
```



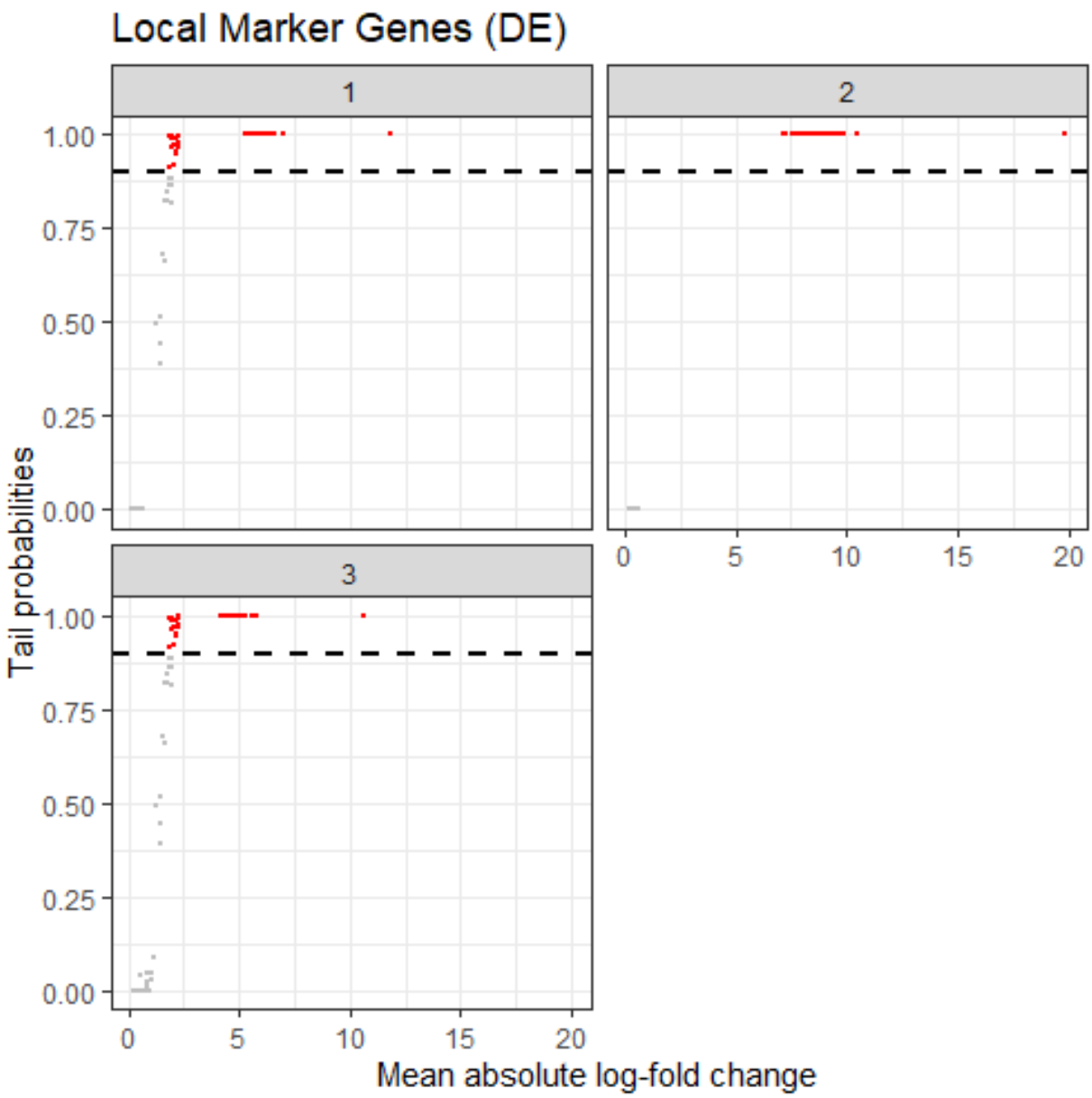
Local marker genes

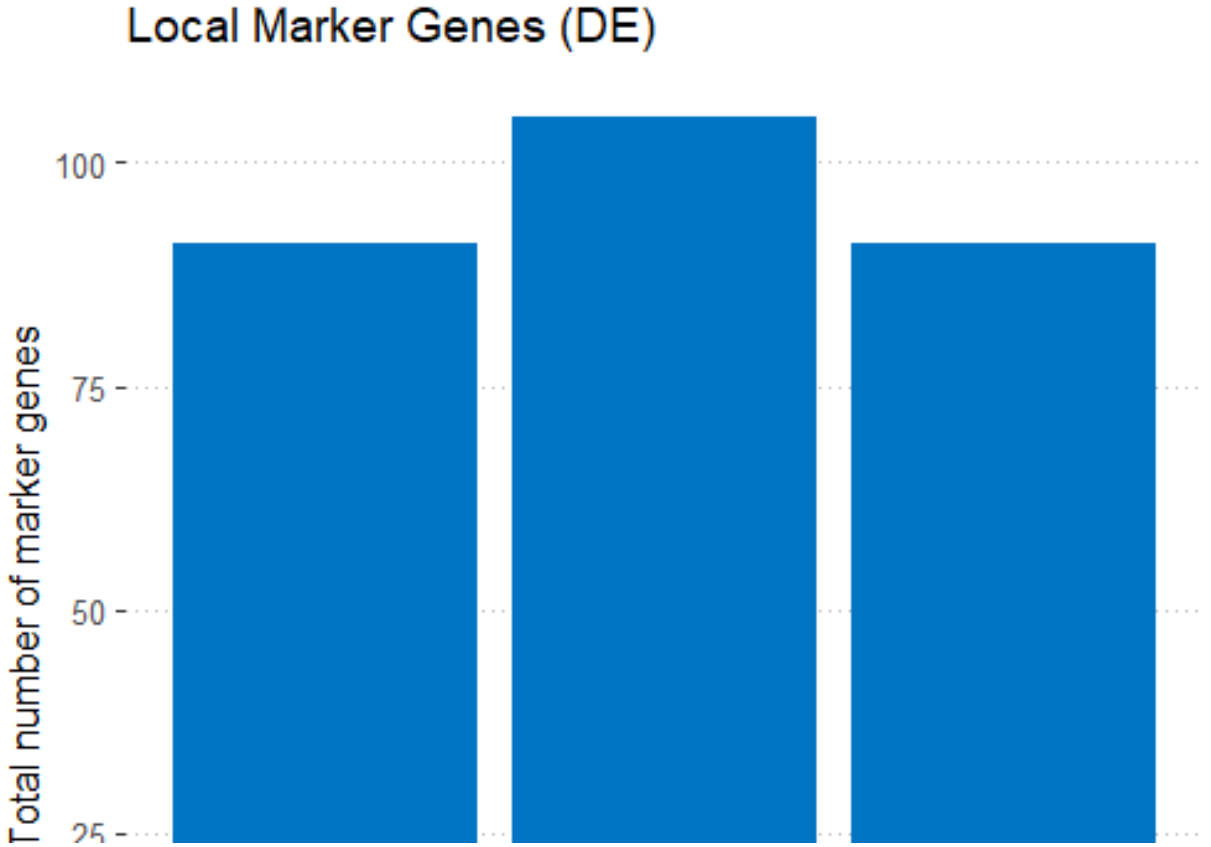
To find local marker genes, we use the function local_marker_genes() where the input structure is the same as for function global_marker_genes(). It also returns 4 items as output, but the structure of marker_DE and marker_DD are slightly different; both of these data frames contain one extra variable to label the cluster. Note that when referring to local marker genes, we need to specify the cluster as well.

```
##-- Using the same set of threshold as before
lmg_output_test <- local_marker_genes(normHDP_post_output = MCMC_sample_post,
                                      threshold = threshold.example,
                                      alpha_M = 0.9,
                                      alpha_D = 0.9)
```

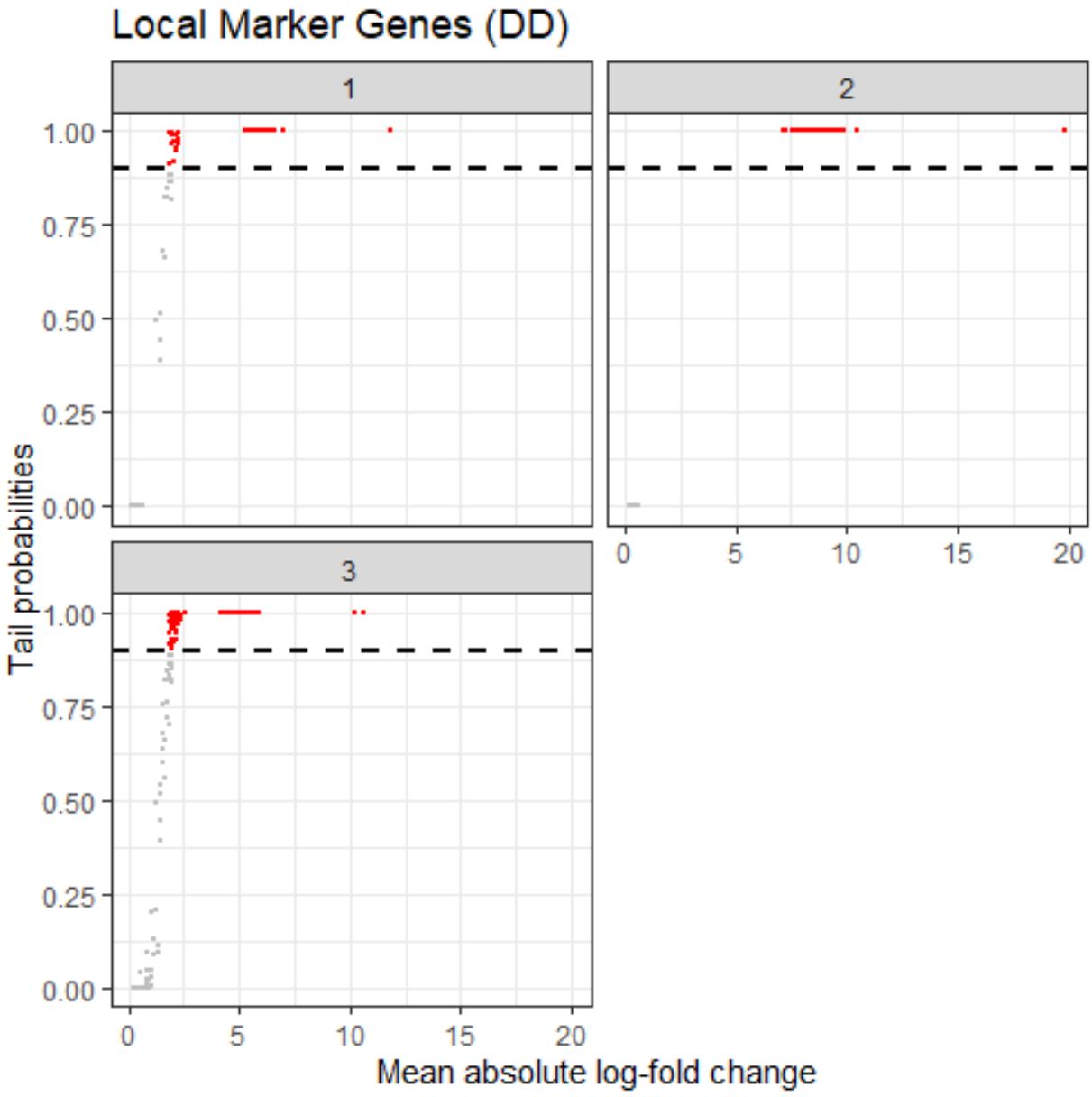
For the plot to show relationships between absolute log-fold change and tail probabilities, and plot to summarize total number of local marker genes for each cluster:

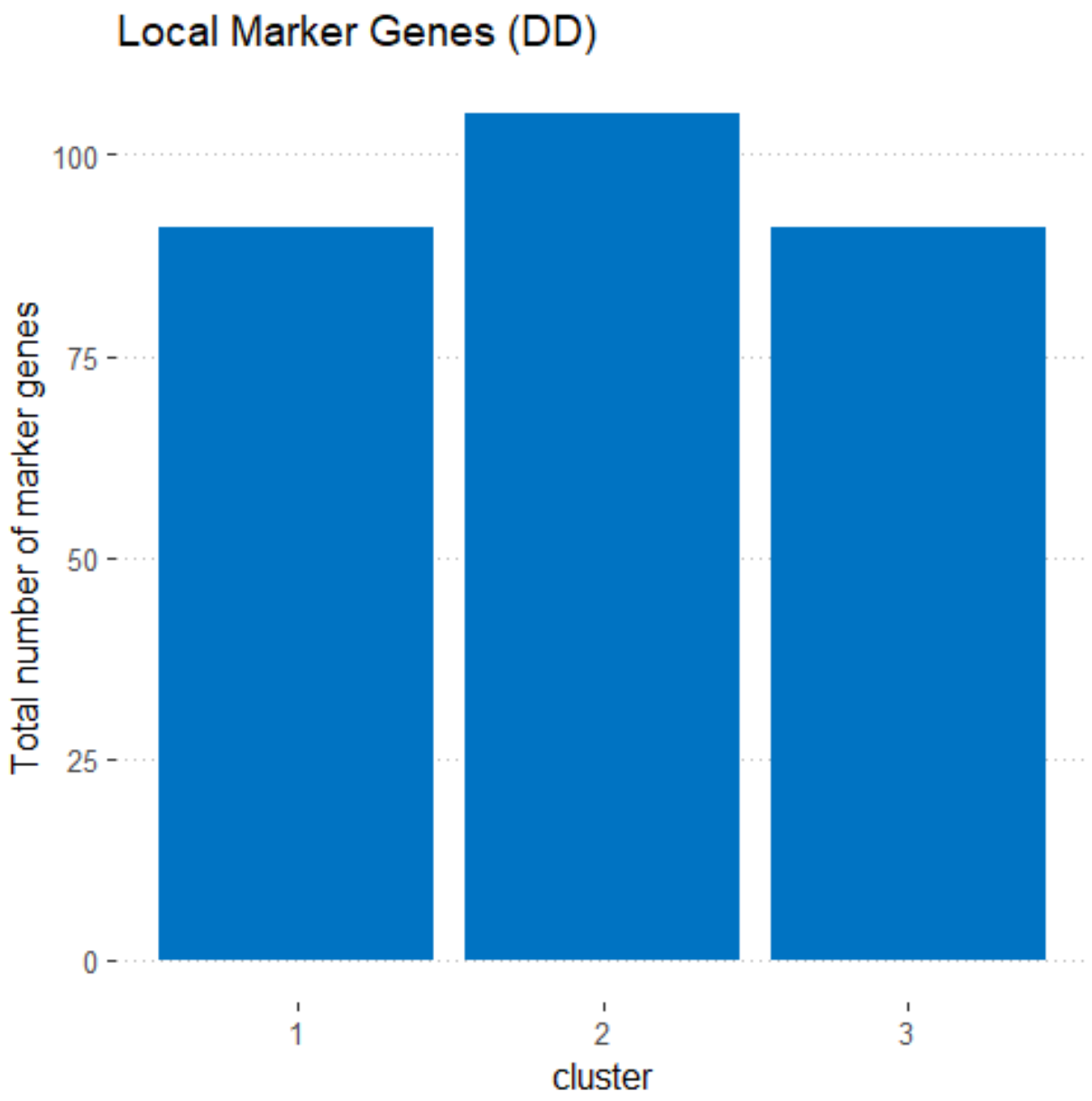
```
local_marker_genes_plot(lmg_output = lmg_output_test)
```


Local Marker Genes (DE)
1
2
3
1.00
0.75
0.50
0.25
0.00
0
5
10
15
20
Tail probabilities
Mean absolute log-fold change


Local Marker Genes (DE)
100
75
50
25
Total number of marker genes

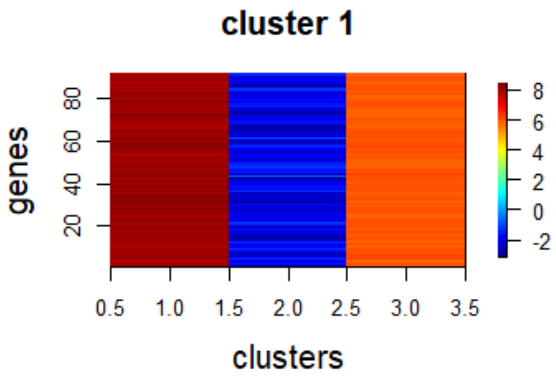

0
1
2
3
cluster

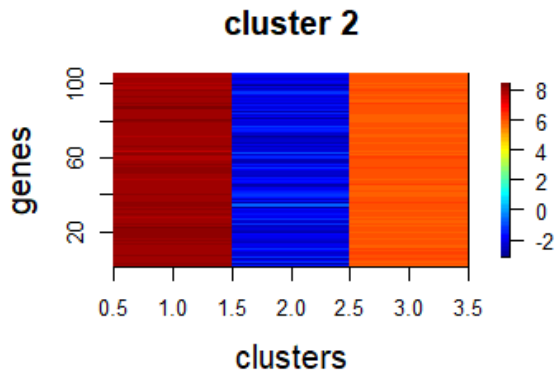

Local Marker Genes (DD)
1
2
3
1.00
0.75
0.50
0.25
0.00
0
5
10
15
20
Tail probabilities
Mean absolute log-fold change

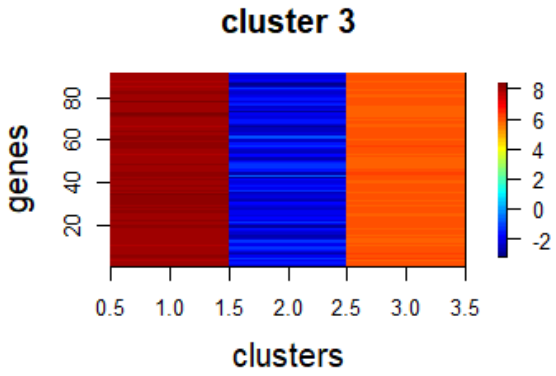


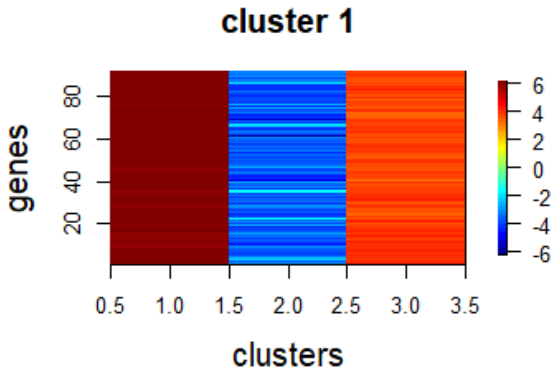
For the heat map:

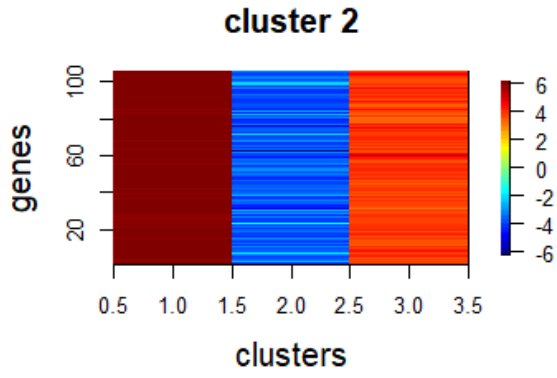
```
local_marker_genes_heatmap(lmg_output = lmg_output_test,
                           normHDP_post_output = MCMC_sample_post)
```

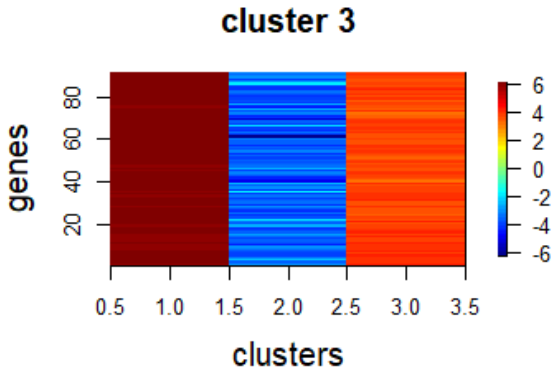












Latent counts and observed counts

We use the latent_counts() function to calculate and reorder latent counts; By default, only cells are reordered unless output from gmg_output() is provided. Specifically, we have a list of 3 items:

- matrix.output: a list of D items, same structure as input Y.
- index.solid: index to plot on the x-axis to separate cells from different clusters.
- index.dashed: index to plot on the x-axis to separate cells from different datasets.

If output from gmg_output() is provided, then the output will be a list with 6 items:

- Y_latent_DE: a list of D items, same structure as input Y. Cells are reordered by clustering and genes are reordered by the tail probabilities corresponding to mean expressions for each dataset.
- Y_latent_DD: a list of D items, same structure as input Y. Cells are reordered by clustering and genes are reordered by the tail probabilities corresponding to dispersions fpr each dataset.
- DE_number: Number of global differentially expressed genes.
- DD_number: Number of global differentially dispersed genes.
- index.solid and index.dashed are also provided.

```
##-- True latent counts
latent_counts_sim <- latent_counts(mu_JG = post.mean.mu,
                                   phi_JG = post.mean.phi,
                                   Z_CD = Z_post,
                                   beta_CD = post.mean.beta,
                                   Y = Y_sim,
                                   gmg_output = gmg_output_test)
```

We can use the observed_counts() function to reorder cells and genes of the observed counts; by default, the function only reorder cells unless output from gmg_output() is provided. Under default, the observed_counts() function outputs a list of 3 items:

- matrix.output: a list of D items, same structure as input Y.
- index.solid: index to plot on the x-axis to separate cells from different clusters.
- index.dashed: index to plot on the x-axis to separate cells from different datasets.

If output from gmg_output() is provided, then the observed_counts() function outputs a list of 6 items:

- Y_DE: a list of D items, same structure as input Y. Cells are reordered by clustering and genes are reordered by the tail probabilities corresponding to mean expressions.
- Y_DD: a list of D items, same structure as input Y. Cells are reordered by clustering and genes are reordered by the tail probabilities corresponding to dispersions.
- DE_number: Number of global differentially expressed genes.
- DD_number: Number of global differentially dispersed genes.
- index.solid and index.dashed are also provided.

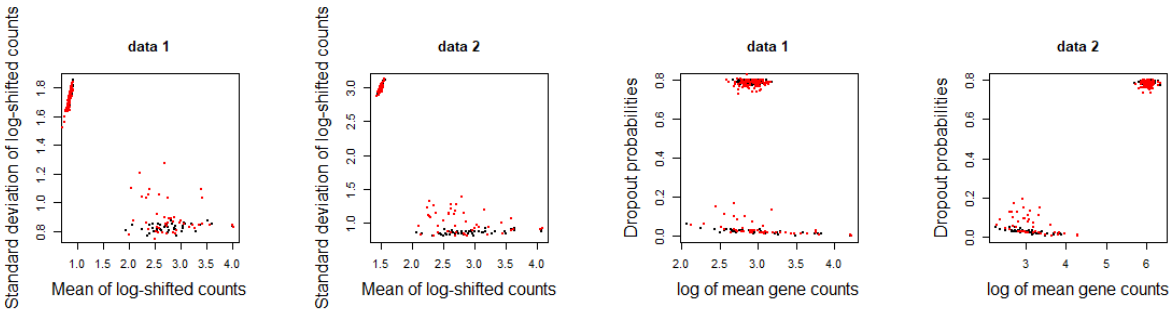
```
##-- Observed counts with cells reordered
observed_counts_sim <- observed_counts(Z_CD = Z_post,
                                       Y = Y_sim,
                                       gmg_output = gmg_output_test)
```

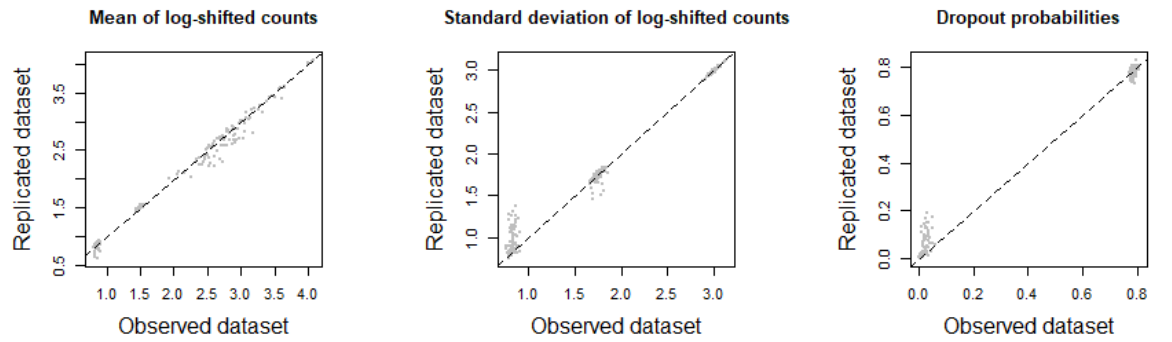
Posterior Predictive Checks

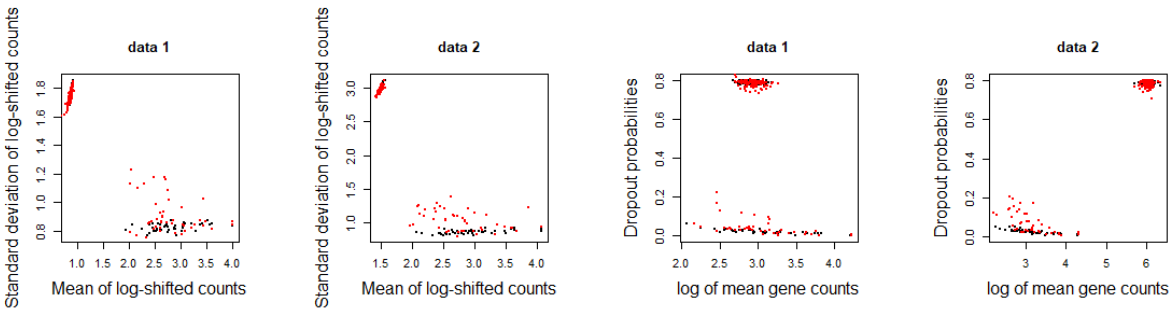
Section below demonstrate the use of posterior predictive checks (ppc). We repeat steps below twice to show variation. For each posterior predictive check below, we compare 2 relationships:

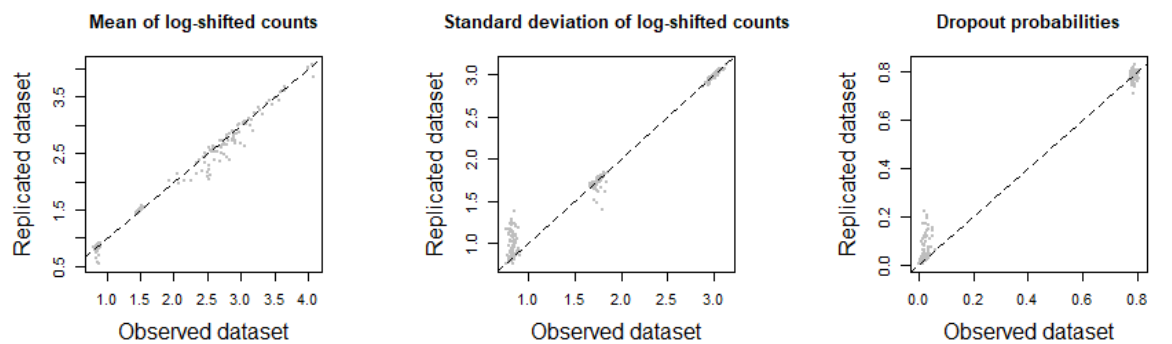
- Relationship between mean of log-shifted counts and standard deviation of log-shifted counts.
- Relationship between log of mean counts and dropout probabilities (the proportion of zero counts for each gene).

```
for(i in 1:2){

  ppc_single(normHDP_output = MCMC_sample,
             Y = Y_sim,
             data.name = c('data 1', 'data 2'))
}
```









In addition, there is also an option to check the recovery of these above relationships with multiple replicated datasets, and to show each relationship for each dataset on one plot. We use function ppc() to obtain multiple replicates; example below uses 30 replicates. The ppc() function outputs 2 items:

- Statistics of the replicated datasets. The index of the replicated datasets are labelled with variable t in the data frame.
- Statistics of the observed datasets.

```
multiple_ppc <- ppc(normHDP_output = MCMC_sample,
                    Y = Y_sim,
                    number_rep = 30)
```

Unlike ppc with single replicate, ppc with multiple replicates ignores the exact statistic for each gene and only consider the general fitted relationship.

```
ppc_plot(ppc_output = multiple_ppc,
         data.name = c('data1', 'data2'))
```