

Condition-specific detection with SpeCond

Florence Cavalli

January 4, 2026

Contents

1	The detection process in a few words	2
2	Quick start	2
2.1	SpeCond function	2
2.2	Result visualisation	4
2.2.1	HTML page displaying the full detection result	4
2.2.2	HTML result pages for each gene	5
3	Parameters	5
3.1	Description	5
3.2	How to change the parameter values	6
3.3	The effect of the parameters on the detections	8
4	More details	8
4.1	Advanced usage of SpeCond	8
4.2	Stepwise analysis	10
5	Output	11
5.1	R Objects	11
5.2	Text files	14
5.3	Visualisation, HTML pages	15
5.3.1	getFullHtmlSpeCondResult	15
5.3.2	getGeneHtmlPage	15
6	Changing the parameter values to improve the detection results	16
7	Advice	19
8	References	20

Introduction

This vignette presents the `SpeCond` package, an R package which performs gene expression data analysis to detect condition-specific genes. Such genes are significantly up- or down-regulated in a small number of different conditions. Conditions can be environmental conditions, tissues, organs or any other sources that you wish to compare in terms of gene expression.

Condition-specific genes are essentially outliers in an expression pattern. In order to detect such outliers, `SpeCond` fits a mixture of normal distributions to the expression values. In addition to the main function `SpeCond`, this package includes other methods such as `writeSpeCondResult`, `getFullHtmlSpeCondResult` and `getGeneHtmlPage` which produce text files and HTML reports.

1 The detection process in a few words

The main steps of the procedure are as follows: (i) model the expression data with a mixture of normal distribution(s), (ii) identify the null distribution as well as candidate outlier observations, (iii) compute p-values of the expression values using the null distribution and adjust for multiple comparisons using the Benjamini and Yekutiely (or a user-defined) method (iv) identify significant condition-specific expression values of the gene using the adjusted p-values.

2 Quick start

2.1 `SpeCond` function

The function `SpeCond` enables the user to perform the full analysis. This function is called with the following arguments:

- *expressionMatrix*: an `ExpressionSet` object or a matrix of expression values (in log2); columns are the conditions, rows are genes (or probe sets)
- *param.detection*: a vector containing the parameters for both steps of the procedure, see section "Parameters"
- *multitest.correction.method*: the multitest correction method; the default is "BY", see `p.adjust` for the possible values
- *prefix.file*: a prefix added to the histogram file (if produced). It will be used to link to the result HTML pages generated by other functions using the result object of this function (if no other prefix value is implemented). The default is "A". It is useful to change the prefix when you perform a new analysis with different parameters, as you may want to compare the results
- *print.hist.pv*: a logical (TRUE/FALSE) value indicating whether a histogram of p-values is to be printed; the default is FALSE
- *condition.factor*: this argument can be used if *expressionMatrix* is an `ExpressionSet` object; a factor object of length equal to the number of columns (samples) of the `ExpressionSet` object specifying which sample(s) belong to which condition (condition.factor levels); can be extracted from the `pData`
- *condition.method*: this argument can be used if *expressionMatrix* is an `ExpressionSet` object; the method (mean, median or max) to summarise the samples by conditions (defined by the *condition.factor* vector)

Example:

Note: For this vignette we will work in a temporary directory (using `tempdir()` as below), to limit the size of the source files. However, for your own use you should ignore it, by default the `.RData` and files are generated and saved in the current directory.

```
> d=tempdir()
> oldir=getwd()
> setwd(d)
```

Loading the library and the example dataset; an ExpressionSet and a matrix expression value:

```
> library(SpeCond)
> data(expSetSpeCondExample)
> expSetSpeCondExample
```

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 220 features, 64 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: S_1 S_2 ... S_64 (64 total)
  varLabels: Tissue Exp
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation:
```

```
> class(expSetSpeCondExample)

[1] "ExpressionSet"
attr(,"package")
[1] "Biobase"

> data(expressionSpeCondExample)
> Mexp=expressionSpeCondExample
> class(Mexp)
```

```
[1] "matrix" "array"

> dim(Mexp)

[1] 220 32
```

Perform the analysis with default parameters:

Using an expression matrix as input:

```
> generalResult=SpeCond(Mexp, param.detection=NULL, multitest.correction.method="BY",
+ prefix.file="E", print.hist.pv=TRUE, fit1=NULL, fit2=NULL, specificOutlierStep1=1)

[1] "Step1"
[1] "Step1, fitting"
[1] "start: get null distributions"
[1] "end: get null distributions"
[1] "start: specific detection from p-values"
[1] "end: specific detection from p-values"
[1] "Step2"
[1] "Step2, fitting"
[1] "start: get null distributions"
[1] "end: get null distributions"
[1] "start: specific detection from p-values"
[1] "end: specific detection from p-values"
```

```
> names(generalResult)
```

```
[1] "prefix.file"          "fit1"                  "fit2"
[4] "specificOutliersStep1" "specificResult"
```

```
> specificResult=generalResult$specificResult
```

Using an ExpressionSet object as input:

Specifying the *condition.factor* and the *condition.method* arguments to extract correctly the expression values according to the conditions, see `getMatrixFromExpressionSet` for details.

```
> generalResult=SpeCond(expSetSpeCondExample, param.detection=NULL,
+   multitest.correction.method="BY", prefix.file="E", print.hist.pv=TRUE, fit1=NULL,
+   fit2=NULL, specificOutlierStep1=NULL, condition.factor=expSetSpeCondExample$Tissue,
+   condition.method="mean")
```

Retrieve the expression matrix used by the analysis with `getMatrixFromExpressionSet` and the same arguments (*condition.factor* and *condition.method*) used in the `SpeCond` function. This is necessary for the visualisation functions.

```
> MexpS=getMatrixFromExpressionSet(expSetSpeCondExample,
+   condition.factor=expSetSpeCondExample$Tissue, condition.method="mean")
```

The **generalResult** and **specificResult** objects generated above are described in the Output section.

2.2 Result visualisation

Two functions `getFullHtmlSpeCondResult` and `getGeneHtmlPage` allow the user to visualise the results.

2.2.1 HTML page displaying the full detection result

A general result HTML page generated by the `getFullHtmlSpeCondResult` function gives an overall view of the condition specific behaviour of genes present in the dataset. This page mainly contains a plot of the number of specific genes by condition, a specific profile heatmap and the result table with the specific genes and their specific condition(s).

The function `getFullHtmlSpeCondResult` is called with the following arguments:

- *SpeCondResult*: the result object of the `SpeCond` functions
- *param.detection*: the parameter matrix used in the analysis (by `SpeCond`)
- *page.name*: the name of the result HTML page. The default is "SpeCond_result"
- *page.title*: the title of the result HTML page. The default is "Condition-specific analysis results"

Further arguments are described in the paragraph "Visualisation, HTML pages".

```
> getFullHtmlSpeCondResult(SpeCondResult=generalResult, param.detection=
+   specificResult$param.detection, page.name="Example_SpeCond_results",
+   page.title="Tissue specific results", sort.condition="all", heatmap.profile=TRUE,
+   heatmap.expression=FALSE, heatmap.unique.profile=FALSE, expressionMatrix=Mexp)

[1] "The following files are created in the directory:"
[1] "/tmp/RtmpZcmwGr/E_General_Result"
[1] "E_barplot_nb_tissue_nb_genes.png"
[1] "E_nb_specific_gene_in_condition.png"
[1] "E_profile_heatmap.png" "E_profile_heatmap.pdf"
[1] "E_result_specific_probeset.txt"
```

2.2.2 HTML result pages for each gene

A result HTML page for each gene can be produced by the function `getGeneHtmlPage`. In addition, this function creates an index to allow you to navigate easily between the results pages for each gene. Each gene result page contains the parameter set used, the expression profile plot and its density curve. An additional plot displays the normal density functions from the normal mixture model as well as the null distribution curve determined by `SpeCond`. Finally, it presents the condition(s) in which the gene is specific with the associated p-value.

The function `getGeneHtmlPage` is called with the following arguments:

- *expressionMatrix*: the matrix of expression values initially used
- *specificResult*: the R object result of the `getSpecificProbeset` function
- *name.index.html*: the name of the HTML index, the default is "index.html"
- *prefix.file*: a prefix added to the generated file(s) and *outdir* directory name to linked to the index file. The default is NULL, the *prefix.file* attribute of *specificResult* object is used
- *outdir*: the name of the directory in which the generated files will be created. The default is "Single_result_pages"
- *gene.html*: a vector of gene names for which you want to create HTML pages, same as the row names of the *expressionMatrix* object, the default is NULL (the values of the *gene.html.ids* argument will be used)
- *gene.html.ids*: a vector of integers corresponding to the row numbers in the *expressionMatrix* object for the genes for which you want to create HTML pages. The default is the first 10 rows (or the number of rows of the *expressionMatrix* if lower to 10).

Remark: If both *gene.html* and *gene.html.ids* are set to NULL, the gene HTML pages for every gene in *expressionMatrix* will be generated. It is possible to use *gene.html* or *gene.html.ids* to select a set of genes. For a given set of genes, you may wish to compare the results contained in two or more **specificResult** objects, which were derived using different parameter sets. For this reason, it is useful to change the prefix (by setting the variable **prefix.file**) as well as the name of the index file (by setting the variable **name.index.html**).

```
> genePageInfo=getGeneHtmlPage(Mexp, specificResult, name.index.html=
+   "index_example_SpeCond_Results.html", gene.html.ids=c(1:20))
[1] "The gene html page(s) will be created in the E_Single_result_pages directory"
```

3 Parameters

3.1 Description

`SpeCond` uses two sets of parameters that can be tuned intuitively by the user. Two parameters, *lambda* and *beta*, are involved in the implementation of the normal mixture model. Additionally, four parameters are used for the determination of the null distribution: the percentage threshold (*per*), the median difference (*md*), the minimum log-likelihood (*mlk*) and the minimum standard deviation ratio (*rsd*).

To be detected as an outlier, a normal component of the mixed distribution must meet several criteria. The difference between its median and that of the principal normal component (distribution clustering most of the expression values) must be greater than *md*, and the percentage of expression values attributed to this normal must be smaller than *per*. Lastly, the component in question and the principal component must be well-separated, such that the minimum of the absolute log-likelihood ratio of the expression values under the two components is larger than *mlk*, or the component in question must be much more spread-out, such that the ratio of their standard deviations is less than *rsd* (see Figure 1). A p-value threshold is set to define a condition as specific for a given gene. Adjusted p-values are used.

- *lambda*: influences the choice of models by affecting the selection of one, two or three normal distributions, thus introducing some weight on the effect of number of parameters to be defined by the clustering model. The default is 1, the model uses the Bayesian Information Criterion (BIC) value taking into account the log-likelihood value
- *beta*: influences the prior applied during the determination of the variance of the normal distributions. It is necessary in the first fitting step to allow the model to capture isolated outliers. It is set by default to 0 in step 2
- *per*, percentage threshold: this is the maximum percentage of conditions in which a gene can be defined as specific. As *per* increases, more expression values will be detected as outliers, so each gene can have a larger number of specific conditions associated with it.
- *md*, median difference: this is the minimum distance between the median values of two normal components of the mixed distribution, which allows identification of one component as an outlier (i.e. possibly not part of the null distribution). Low median distances are excluded to filter out noise that is common in biological samples
- *mlk*, minimum log-likelihood: allows the identification of clusters of conditions that are well separated from the other(s) in the model. If the minimum log-likelihood of a set of expression values in a normal component exceeds *mlk*, then this component is a possible outlier (i.e. not part of the null distribution)
- *rsd*, minimum of standard deviation ratio: allows the identification of clusters of conditions that are extremely spread out compared to the distribution clustering of most expression values. If the ratio of standard deviation between the component and the principal normal component is below *rsd*, then this component is a possible outlier (i.e. is not part of the null distribution)
- *pv*, p-value threshold to detect a condition as specific for a given gene

Figure 1 presents the different detection cases. As *mlk* increases and *rsd* decreases, it is less likely that a mixture component will be defined as an outlier.

3.2 How to change the parameter values

The parameter's values are stored in a matrix (*param.detection*). The procedure includes two steps, which are described in the following section. As a consequence each parameter can be changed to improve the detection. The first row called "Step1" and the second row "Step2" of *param.detection* contain the parameters for the first and the second step of the procedure, respectively.

To obtain the default parameters, use the function `getDefaultParameter`.

```
> param.detection=getDefaultParameter()
> param.detection
```

	beta	lambda	per	md	mlk	rsd	pv
Step 1	6	1	0.1	0.75	5	0.1	0.05
Step 2	0	1	0.3	0.75	25	0.1	0.05

The function `createParameterMatrix` enables the user to create a new parameter matrix or to change value(s) from the default or a given parameter matrix. The different arguments allow to change the values in the *param.detection* send as argument. If *param.detection* is not specified (i.e. NULL), the default parameters will be used and then changed according to the other arguments. The different arguments are: *param.detection*, *beta.1*, *beta.2*, *lambda.1*, *lambda.2*, *per.1*, *per.2*, *md.1*, *md.2*, *mlk.1*, *mlk.2*, *rsd.1*, *rsd.2*, *pv.1* and *pv.2*.

```
> param.detection2=createParameterMatrix(mlk.1=10)
> param.detection2
```



Figure 1: Determination of the null distribution: The three different conditions evaluated in order to consider a normal component as part of the null distribution. (I) The median of the values from each component must have a difference larger than *md*. If this first condition is fulfilled, the procedure tests the following conditions: The normal component will not be part of the null distribution if: (II) The normal component is small and well separated i.e. the minimum of the absolute log-likelihood ratio of the expression values under the two components is larger than *mlk*, (III) The normal component is small and largely spread out i.e. the standard deviation ratio is smaller than *rsd*.

	beta	lambda	per	md	mlk	rsd	pv
Step 1	6	1	0.1	0.75	10	0.1	0.05
Step 2	0	1	0.3	0.75	25	0.1	0.05

```
> param.detection2B=createParameterMatrix(param.detection=param.detection2,
+   mlk.1=15, rsd.2=0.2)
> param.detection2B
```

	beta	lambda	per	md	mlk	rsd	pv
Step 1	6	1	0.1	0.75	15	0.1	0.05
Step 2	0	1	0.3	0.75	25	0.2	0.05

Remark: We strongly advice the following rules:

- beta.2=0
- md.1=md.2
- per.1<=per.2
- pv.1=pv.2

3.3 The effect of the parameters on the detections

To give you a sense of the importance and the effect of the parameter values, we have created a simulated dataset. As example, we analyse this dataset with the default parameters then improve the specific detection modifying two parameters. This detailed analysis is present in the “Changing the parameter values to improve the detection results” section.

4 More details

4.1 Advanced usage of SpeCond

The procedure performed by `SpeCond` integrates two steps (presented in Figure 2) that can be processed separately using the functions presented in the following section. Additionally, after running the `SpeCond` function for the first time, it is possible to test other parameters to obtain the null distribution. In this case, it is not necessary to re-process the fitting step so the results of the first run can be used as an input for the second run. For this purpose, three extra `SpeCond` parameters allow to perform only the latest parts of the procedure (depending of which arguments are still set to NULL).

- *fit1*: the result of the first fitting from the function `fitPrior` or `generalResult$fit1`, the default is NULL
- *fit2*: the result of the second fitting from the function `fitNoPriorWithExclusion` or `generalResult$fit2`, the default is NULL
- *specificOutlierStep1*: the result of the first detection step from the function `getSelectiveOutliers` or `generalResult$specificOutlierStep1`, the default is NULL

Example:

Change the detection parameters for the first step and apply these to the previously computed fitting:

```
> param.detection2=createParameterMatrix(param.detection, mlk.1=10)
> param.detection2
```

	beta	lambda	per	md	mlk	rsd	pv
Step 1	6	1	0.1	0.75	10	0.1	0.05
Step 2	0	1	0.3	0.75	25	0.1	0.05


```
> generalResult2=SpeCond(Mexp, param.detection=param.detection2,
+ multitest.correction.method="BY", prefix.file="E2",
+ print.hist.pv=TRUE, fit1=generalResult$fit1, fit2=NULL,
+ specificOutlierStep1=NULL)
```

```
[1] "Step1"
[1] "start: get null distributions"
[1] "end: get null distributions"
[1] "start: specific detection from p-values"
[1] "end: specific detection from p-values"
[1] "Step2"
[1] "Step2, fitting"
[1] "start: get null distributions"
[1] "end: get null distributions"
[1] "start: specific detection from p-values"
[1] "end: specific detection from p-values"
```

Change the parameters for the second step and apply these to the result of the first step and the second fitting computed previously:

```
> param.detection3=createParameterMatrix(param.detection, rsd.2=0.2, per.2=0.2)
> param.detection3
```

	beta	lambda	per	md	mlk	rsd	pv
Step 1	6	1	0.1	0.75	5	0.1	0.05
Step 2	0	1	0.2	0.75	25	0.2	0.05

```
> generalResult3=SpeCond(Mexp, param.detection=param.detection3,
+ multitest.correction.method="BY", prefix.file="E3", print.hist.pv=TRUE,
+ fit1=generalResult$fit1, fit2=generalResult$fit2,
+ specificOutlierStep1=generalResult$specificOutliersStep1)
```

```
[1] "Step1"
[1] "Use specificOutlierStep1 as result of step1"
[1] "Step2"
[1] "start: get null distributions"
[1] "end: get null distributions"
[1] "start: specific detection from p-values"
[1] "end: specific detection from p-values"
```

<p>Step 1: Detect “single” outliers</p> <ul style="list-style-type: none"> - fitting with $\beta_1 \neq 0$ To be able to fit a normal distribution with one or two values as consequence to catch “single” outliers - Detection: $\text{per}_1 = 0.1$ Must be small representing 1 to 3 conditions to be able to detect and test the specificity (using the others parameters) of the normal components containing the “single” outliers <p>Step 2: Detect the specific-genes</p> <ul style="list-style-type: none"> - fitting with $\beta_2 = 0$ The “single” outliers detected in Step1 are ignored - Detection: $\text{per}_2 = 0.3$ Performs the final detection here 30% of the conditions can be detected as specific for a particular gene

Figure 2: Procedure Steps: Short description of the two step procedure indicating key points in each steps

4.2 Stepwise analysis

As the method works with two steps it can be very useful to process step by step, saving the object(s) and enabling to re-run only the last detection function for example with a new set of parameters.

Here, we described the procedure step by step:

Use the function `getMatrixFromExpressionSet` if your dataset is an `ExpressionSet` to obtain the **Mexp** matrix as presented in the “Quick Start” paragraph.

Step1: get the mixture distributions fitting the expression data with a prior on the variance to catch the single outliers. Then detected the single outliers using the first row (“Step1”) of the *param.detection* matrix:

```
> param.detection

      beta lambda per   md mlk rsd   pv
Step 1    6      1 0.1 0.75   5 0.1 0.05
Step 2    0      1 0.3 0.75  25 0.1 0.05

> fit1=fitPrior(Mexp, param.detection=param.detection)

[1] "Step1, fitting"
```

Or specifying only *lambda* and *beta* values:

```
> fit1=fitPrior(Mexp, lambda=param.detection[1,"lambda"], beta=param.detection[1,"beta"])
```

Detect the single outliers and store them in **specificOutlierStep1**.

```
> specificOutlierStep1=getSpecificOutliersStep1(Mexp, fit=fit1$fit1,param.detection,
+ multitest.correction.method="BY", prefix.file="run1_Step1",
+ print.hist.pv=FALSE)

[1] "start: get null distributions"
[1] "end: get null distributions"
[1] "start: specific detection from p-values"
[1] "end: specific detection from p-values"
```

Step2: get the second fitting ignoring the values of outliers detected in step 1, then perform the specific detection using the second row ("Step2") of the param.detection matrix:

```
> fit2=fitNoPriorWithExclusion(Mexp, specificOutlierStep1=specificOutlierStep1,
+   param.detection=param.detection)

[1] "Step2, fitting"
```

Or specifying only *lambda* and *beta* values:

```
> fit2=fitNoPriorWithExclusion(Mexp, specificOutlierStep1=specificOutlierStep1,
+   lambda=param.detection[2, "lambda"], beta=param.detection[2, "beta"])
```

Detect the condition specific for each gene and store them in **specificResult**.

```
> specificResult=getSpecificResult(Mexp, fit=fit2,
+   specificOutlierStep1=specificOutlierStep1, param.detection,
+   multitest.correction.method="BY", prefix.file="run1_Step2",
+   print.hist.pv=FALSE)

[1] "start: get null distributions"
[1] "end: get null distributions"
[1] "start: specific detection from p-values"
[1] "end: specific detection from p-values"
```

5 Output

The SpeCond package can produce three different types of outputs: R objects, text files and HTML pages.

5.1 R Objects

The result of SpeCond the **generalResult** object is of class **sp_list**. It is a large list containing all the parameters and results of the analysis. The five attributes of **generalResult** correspond to the output of the functions presented in the Stepwise analysis paragraph. So the object **specificResult** obtained by the `getSpecificResult` function is a class **sp_list** as well and corresponds to the fifth attributes of the **generalResult** of the SpeCond function. The later is a list of seven attributes containing all the parameters and results of the detection.

Remark: For all result objects the order of genes and conditions from the input expression value matrix is preserved.

The **generalResult** attributes are:

- *prefix.file*: the prefix used for this analysis. It will be used by default in the function `getFullHtmlSpeCondResult` and `getGeneHtmlPage`
- *fit1*: a list of genes as first attributes and for each gene a list of three attributes:
 - *G*: number of normal components fitting the data
 - *NorMixParam*: the parameters of each normal component of the mixture distribution fitting the expression values of the gene: proportion, mean and standard deviation
 - *classification*: the normal component of the mixture distribution, to which the expression value is attributed
 - Remark: if *evaluate.lamda.beta* is TRUE in `fitPrior`, the result object will be a list of two arguments:
 - * *fit1*: same list as presented above with *G*, *NorMix_param* and *classification* attributes for each gene

- * *G.lambda.beta.effect*: matrix presenting the number of times the values of G (number of normal components for a particular gene) have changed between *lambda=0* and the *lambda.1* value and between *beta=0* and the *beta.1* value
- *fit2*: same list as *fit1* described above with attributes: *G_initial*, *G,NorMixParam* and *classification* for each gene and an additional attribute *specificOutlierStep1*:
 - *specificOutlierStep1*: the condition(s) for which the expression value of the gene is detected as outlier in the first step of the procedure. If NULL, no expression value has been detected in the first step. The second fitting ignores these expression values
- *specificOutliersStep1*: a list of all genes with the condition(s) (i.e. column id(s)), in which the gene has been detected as specific, NULL if not
- *specificResult*: described below

The **specificResult** object is of class **sp_list**. This list containing 8 attributes

- *prefix.file*: the prefix used for this analysis. It will be used by default in the function `getGeneHtmlPage`
- *param.detection*: the parameters used for the two steps
- *fit*: the fit object of the second step (same as *fit2*)
- *L.specific.result*: full detection results (it will be used by the `getFullHtmlSpeCondResult` function). This list contains 7 attributes:
 - *M.specific.all*: matrix of 0: not selective, 1: selective up-regulated, -1: selective down-regulated; same dimensions as the input matrix of expression values
 - *M.specific*: same as *M.specific.all* but reduced to the specific genes. NULL if no gene has been detected as specific
 - *M.specific.sum.row*: Number of conditions in which the gene is specific (-1 and 1 values)
 - *M.specific.sum.column*: Number of specific genes by conditions
 - *L.pv*: list of all genes with a matrix of conditions and the corresponding p-values (if the gene is specific)
 - *specific*: vector of size the number of genes with only the values "Not specific" or "Specific" according to the specificity of the gene (used by the HTML display)
 - *L.condition.specific.id*: list of the specific genes with a vector of column numbers (condition ids), for which the gene is specific
- *L.null*: a list containing a vector of 1 and 0 representing the null distribution. The length of the vector for each gene corresponds to the number of normal components fitting the gene expression value. The list is sorted as the gene in the input matrix of expression values
- *L.mlk*: a list of vectors containing the minimum log-likelihood computed between normal distributions. NULL if the mixture model of the gene is composed of only one component or if the proportion of all components is superior to the *per.2* parameter
- *L.rsd*: a list of vectors containing the standard deviation ratios computed between normal distributions. NULL if the mixture model of the gene has only one component
- *identic.row.ids*: row number(s) from the initial input matrix, which contain identical values for all conditions. These rows are not considered in the analysis

Example:

```
> names(generalResult)
```

```

[1] "prefix.file"          "fit1"          "fit2"
[4] "specificOutliersStep1" "specificResult"

> specificResult=generalResult$specificResult
> names(specificResult)

[1] "prefix.file"          "fit"           "param.detection"
[4] "L.specific.result"   "L.null"        "L.mlk"
[7] "L.rsd"               "identic.row.ids"

> names(specificResult$L.specific.result)

[1] "M.specific.all"       "M.specific"
[3] "M.specific.sum.row"   "M.specific.sum.column"
[5] "L.pv"                "specific"
[7] "L.condition.specific.id"

> dim(specificResult$L.specific.result$M.specific.all)

[1] 220 32

> dim(specificResult$L.specific.result$M.specific)

[1] 23 32

> specificResult

An object of class "sp_list"
Only the main values are presented here see show.sp_list() function for
a comprehensive view of this object
$prefix.file
[1] "E"
$param.detection
      beta lambda per   md mlk rsd   pv
Step 1    6      1 0.1 0.75   5 0.1 0.05
Step 2    0      1 0.3 0.75  25 0.1 0.05
$L.specific.result$M.specific
[1] "M.specific"
      Spinal_cord Fetal_brain Adrenal_cortex Pituitary Whole_brain
204501_at          0          0          1          0          0
204507_s_at          0          0          0          0          0
204508_s_at          0          0          0          0          0
204529_s_at          0          0          0          0          0
204532_x_at          0          0          0          0          0
18 more rows and 27 more columns ...

$L.specific.result$M.specific.sum.row
      200606_at 200607_s_at 200608_s_at   204501_at 204507_s_at
              0           0           0             2             1
215 more elements ...

$L.specific.result$M.specific.sum.column
      Spinal_cord Fetal_brain Adrenal_cortex Pituitary
M_specific_positive_sum          4           6           1           4
M_specific_negative_sum          0           0           0           0
Both                          4           6           1           4

```

	Whole_brain
M_specific_positive_sum	5
M_specific_negative_sum	0
Both	5
27 more columns ...	

Number of genes evaluated: 220

Number of genes specific: 23

Range of the number of conditions for which a gene have been detected as specific: 0

Number of genes specific by number of specific conditions

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	[, 6]	[, 7]
# conditions	"1"	"2"	"5"	"6"	"7"	"8"	"sum"
# genes	"9"	"7"	"4"	"1"	"1"	"1"	"23"

5.2 Text files

Three functions enable to write the results to text files.

- `writeSpeCondResult`: To write the three following text files:
 - The table of genes detected as specific and in which condition they are specific (0: not specific, 1: specific up-regulated, -1: specific down-regulated). The file name is set by the parameter *file.name.profile*, the default name is "specific_profile.txt".
 - The list of the specific genes. The file name is set by the parameter *file.specific.gene*, the default name is: "list_specific_probeset.txt".
 - The table of the unique specific profiles detected. The file name is set by the parameter *file.name.unique.profile*, the default name is: "specific_unique_profile.txt".
- `writeUniqueProfileSpecificResult`: To write a text file with the unique specific profiles among the conditions. If *full.list.gene* is TRUE, the last column corresponds to the gene's names which have the profile described in the row.
- `writeGeneResult`: To write a text file containing the list of all genes from the input expression values matrix, whether they have been detected as condition specific or not (S/N), for how many conditions in total, in how many conditions as up-regulated, in how many conditions as down-regulated, in which conditions as up-regulated and down-regulated. The argument *gene.names* can be used to select a subset of genes (the default is NULL).

All these functions use the function `getProfile` to obtain the profiles of the specific genes, see documentation and example below

```
> L.specific.result.profile=getProfile(specificResult$L.specific.result$M.specific)
> writeSpeCondResult(specificResult$L.specific.result, file.name.profile=
+ "Example_specific_profile.txt", file.specific.gene="Example_list_specific_gene.txt",
+ file.name.unique.profile="Example_specific_unique_profile.txt")
[1] "write file: Example_specific_profile.txt"
[1] "write file: Example_list_specific_gene.txt"
[1] "write file: Example_specific_unique_profile.txt"
> writeUniqueProfileSpecificResult(L.specific.result=specificResult$L.specific.result,
+ file.name.unique.profile="Example_specific_unique_profile.txt", full.list.gene=FALSE)
[1] "write file: Example_specific_unique_profile.txt"
> writeGeneResult(specificResult$L.specific.result, file.name.result.gene=
+ "Example_gene_gummary_result.txt", gene.names=row.names(Mexp)[1:10])
[1] "write file: Example_gene_gummary_result.txt"
```

5.3 Visualisation, HTML pages

The HTML page functions `getFullHtmlSpeCondResult` and `getGeneHtmlPage` have been presented above. Here we present all the possible arguments of these functions enabling to personalise the result HTML pages.

5.3.1 `getFullHtmlSpeCondResult`

Some other arguments can be used in the `getFullHtmlSpeCondResult`; notably the arguments *prefix.file* and *outdir* enable to properly save the different result pages.

- *SpeCondResult*: the result object of **sp_list** class result of the `SpeCond` functions
- *L.specific.result*: list of results present in the **specificResult sp_list** class object, see `SpeCond` or `getSpecificResult` functions
- *param.detection*: the parameter matrix used in the analysis (by `SpeCond`)
- *page.name*: the name of the result HTML page. The default is "SpeCond_result"
- *page.title*: the title of the result HTML page. The default is "Condition-specific analysis results"
- *prefix.file*: a prefix added to the generated file(s) and the *outdir* directory name to link them to the full result HTML page. The default is NULL. It is useful to change the prefix when you create a new result page. As you may want to get results with different parameter sets and plots so using a different *SpeCondResult* or *L.specific.result* objects
- *outdir*: the name of the directory in which the generated files will be created. The default is "General_result"
- *sort.condition*: the way to sort the conditions in the barplot that presents the number of specific genes per condition. Values are "positive", "negative" or "all" determines that the conditions are sorted by the number of specific genes detected as up-regulated, down-regulated or both respectively
- *gene.page.info*: the result of the `getGeneHtmlPage` function. Enables the creation of links between this full result page and the single result pages created by the previous function. The default is "NULL"; no links are created
- *heatmap.profile*: a logical (TRUE/FALSE) whether to print or not a heatmap showing the specific profile of the genes, the default is FALSE
- *heatmap.expression*: a logical (TRUE/FALSE) whether to print or not a heatmap showing the expression of the genes, the default is FALSE
- *heatmap.unique.profile*: a logical (TRUE/FALSE) whether to print or not a heatmap showing the unique specific profile, the default is FALSE
- *expressionMatrix*: Must not be NULL if *heatmap.expression*=TRUE, must be the same as the input expression matrix, the default is NULL

Remark: One can use either *SpeCondResult* or *L.specific.result* as the *L.specific.result* is included in the *SpeCondResult* object. If *prefix.file* is NULL the *prefix.file* attribute of *SpeCondResult* is used.

5.3.2 `getGeneHtmlPage`

The function `getGeneHtmlPage` creates one HTML page by genes and the corresponding plots. As a consequence a large amount of files can be generated. To keep track of the result pages linked to different parameter sets, it is important to use the *prefix.file* arguments. The main index HTML page will be created in the current directory whereas all the other gene HTML pages will be created in the *outdir* directory. Here are all the `getGeneHtmlPage` arguments, more importantly the arguments *prefix.file* and *outdir* allow to configure the classification of your results.

- *expressionMatrix*: the matrix of expression values initially used
- *specificResult*: the R object result of the `getSpecificProbeset` function
- *name.index.html*: the name of the HTML index, the default is "index.html"
- *prefix.file*: a prefix added to the generated file(s) and *outdir* directory name to linked to the index file. The default is NULL, the *prefix.file* attribute of the *specificResult* is used
- *outdir*: The name of the directory in which the generated files will be created. The default is "Single_result_pages"
- *gene.html*: a vector of gene names, same as the row names of the *expressionMatrix* object, the default is NULL
- *gene.html.ids*: a vector of integers corresponding to the row numbers in the *expressionMatrix* object for the gene for which you want to create HTML pages. The default is the 10 first rows (or the number of row of the *expressionMatrix* if inferior to 10)

6 Changing the parameter values to improve the detection results

In this section, we will use a simulated dataset to appreciate the effect of the parameters. The principal parameters of the detection are *mlk.2* and *rsd.2*.

The dataset is simulated from three different normal distributions. The default expression values for each probeset is randomly generated from a normal distribution of mean=7 and sd=0.6. The probesets 1 to 100 have specific expression values for the conditions 10, 20 and 30 coming from a normal distribution of mean=11 and sd=0.5. The probesets 200 to 300 have specific expression values for the conditions 9, 18 and 27 coming from a normale distribution of mean=13 and sd=0.4. We will change the parameter *mlk.2* and *rsd.2* to improve the specific detection (detecting the two types of specific behavior) starting with the default parameters. (The following code is present the *SpeCond.R* file available on the Bioconductor package page.)

Load the simulated dataset:

```
> data(simulatedSpeCondData)
```

Perform the first analysis using the default parameters:

```
> generalResult_S1=SpeCond(simulatedSpeCondData, param.detection=NULL,
+   multitest.correction.method="BY", prefix.file="S1", print.hist.pv=TRUE,
+   fit1=NULL, fit2=NULL, specificOutlierStep1=NULL)

[1] "Step1"
[1] "Step1, fitting"
[1] "start: get null distributions"
[1] "end: get null distributions"
[1] "start: specific detection from p-values"
[1] "end: specific detection from p-values"
[1] "Step2"
[1] "Step2, fitting"
[1] "start: get null distributions"
[1] "end: get null distributions"
[1] "start: specific detection from p-values"
[1] "end: specific detection from p-values"
```



```

> specificResult_S1=generalResult_S1$specificResult
> getFullHtmlSpeCondResult(L.specific.result=specificResult_S1$L.specific.result,
+   page.name="simulatedSpeCondData_results", page.title="Condition specific results
+   default", prefix.file="S1", sort.condition="all")

[1] "Advice: Implement the param.detection attribute or used the SpeCondResult attrib
[1] "to present the parameter set used on the S1_simulatedSpeCondData_results.html pa
[1] "The following files are created in the directory:"
[1] "/tmp/RtmpZcmwGr/S1_General_Result"
[1] "S1_barplot_nb_tissue_nb_genes.png"
[1] "S1_nb_specific_gene_in_condition.png"
[1] "S1_profile_heatmap.png" "S1_profile_heatmap.pdf"
[1] "S1_result_specific_probeset.txt"

> genePageInfo_S1=getGeneHtmlPage(simulatedSpeCondData, specificResult_S1,
+   name.index.html="index_simulatedSpeCondData.html", prefix.file="S1",
+   gene.html.ids=c(1:20))

[1] "The gene html page(s) will be created in the S1_Single_result_pages directory"

```

We open *S1_simulatedSpeCondData_results.html* and *S1_index_simulatedSpeCondData.html* files present in the current directory to observe the results.

Looking at the first results, only few probeset are detected as specific. If you look at the gene result pages you can observe that the *mlk* value of the Step2 in particular is too high as the Normal 2 is still part of the null distribution whereas it can be consider as separated from the Normal 1. As a consequence we change the *mlk.2* parameter to 10. Additionally we are using here the result of the fitting from the first detection as described in the Stepwise analysis paragraph.

Change the parameter:

```

> param.detection10=createParameterMatrix(param.detection=param.detection, mlk.2=10)
> param.detection10

```

	beta	lambda	per	md	mlk	rsd	pv
Step 1	6	1	0.1	0.75	5	0.1	0.05
Step 2	0	1	0.3	0.75	10	0.1	0.05

Perform the analysis changing the prefix value:

```

> generalResult_S2=SpeCond(simulatedSpeCondData, param.detection=param.detection10,
+   multitest.correction.method="BY", prefix.file="S2", print.hist.pv=TRUE,
+   fit1=generalResult_S1$fit1, fit2=generalResult_S1$fit2,
+   specificOutlierStep1=generalResult_S1$specificOutliersStep1)

[1] "Step1"
[1] "Use specificOutlierStep1 as result of step1"
[1] "Step2"
[1] "start: get null distributions"
[1] "end: get null distributions"
[1] "start: specific detection from p-values"
[1] "end: specific detection from p-values"

> specificResult_S2=generalResult_S2$specificResult
> genePageInfo_S2=getGeneHtmlPage(simulatedSpeCondData, specificResult_S2,
+   name.index.html="index_simulatedSpeCondData.html", prefix.file="S2",
+   gene.html.ids=c(1:20,200:250))

```

```
[1] "The gene html page(s) will be created in the S2_Single_result_pages directory"

> getFullHtmlSpeCondResult(L.specific.result=specificResult_S2$L.specific.result,
+   param.detection=param.detection10, page.name="simulatedSpeCondData_results",
+   page.title="Condition specific results mlk 10", prefix.file="S2", sort.condition=
+   gene.page.info=genePageInfo_S2)

[1] "The following files are created in the directory:"
[1] "/tmp/RtmpZcmwGr/S2_General_Result"
[1] "S2_barplot_nb_tissue_nb_genes.png"
[1] "S2_nb_specific_gene_in_condition.png"
[1] "S2_profile_heatmap.png" "S2_profile_heatmap.pdf"
[1] "S2_result_specific_probeset.txt"
```

Looking at *S2_simulatedSpeCondData_results.html* and *S2_index_simulatedSpeCondData.html* files we observe that the detection improves but we are still missing some specific probesets with ids between 1 and 100 and more importantly we do not yet detect several specific probe with ids between 200 and 300 for which their specific values are more spread than the core of the expression values. To detect them we decrease *mlk.2* and increase *rsd.2* to 8 and to 0.3 respectively.

Change the parameter:

```
> param.detection8_0.3=createParameterMatrix(param.detection=param.detection,
+   mlk.2=8, rsd.2=0.3)
> param.detection8_0.3
```

	beta	lambda	per	md	mlk	rsd	pv
Step 1	6	1	0.1	0.75	5	0.1	0.05
Step 2	0	1	0.3	0.75	8	0.3	0.05

Perform the analysis changing the prefix value:

```
> generalResult_S3=SpeCond(simulatedSpeCondData, param.detection=param.detection8_0.3,
+   multitest.correction.method="BY", prefix.file="S3", print.hist.pv=TRUE,
+   fit1=generalResult_S1$fit1, fit2=generalResult_S1$fit2,
+   specificOutlierStep1=generalResult_S1$specificOutliersStep1)

[1] "Step1"
[1] "Use specificOutlierStep1 as result of step1"
[1] "Step2"
[1] "start: get null distributions"
[1] "end: get null distributions"
[1] "start: specific detection from p-values"
[1] "end: specific detection from p-values"

> specificResult_S3=generalResult_S3$specificResult
> genePageInfo_S3=getGeneHtmlPage(simulatedSpeCondData, specificResult_S3,
+   name.index.html="index_simulatedSpeCondData.html", prefix.file="S3",
+   gene.html.ids=c(1:20,195:310))

[1] "The gene html page(s) will be created in the S3_Single_result_pages directory"

> getFullHtmlSpeCondResult(L.specific.result=specificResult_S3$L.specific.result,
+   param.detection=param.detection8_0.3, page.name="simulatedSpeCondData_results",
+   page.title="Condition specific results mlk 8, rsd 0.3", prefix.file="S3",
+   sort.condition="all", gene.page.info=genePageInfo_S3)
```

```
[1] "The following files are created in the directory:"
[1] "/tmp/RtmpZcmwGr/S3_General_Result"
[1] "S3_barplot_nb_tissue_nb_genes.png"
[1] "S3_nb_specific_gene_in_condition.png"
[1] "S3_profile_heatmap.png" "S3_profile_heatmap.pdf"
[1] "S3_result_specific_probeset.txt"
```

Looking at *S3_simulatedSpeCondData_results.html* and *S3_index_simulatedSpeCondData.html* files we observe that the detection has largely improved. Change the *gene.html.ids* or *gene.html* values to generate different HTML pages and keep improving the detection results.

7 Advice

The fitting of the normal component using *mclust* can be done only once, in each of the steps of the procedure, as a consequence it is beneficial to save the fitting results as shown below.

```
> save(fit1, file="fit1.RData")
> ## load("fit1.RData")
```

The parameter effect can be evaluated separately for the two steps. Saving the outliers detected by the first steps allows to test only the effect of the second set of parameters.

```
> save(specificOutlierStep1, file="specificOutlierStep1.RData")
> ## load("specificOutlierStep1.RData")
> save(fit2, file="fit2.RData")
> ## load("fit2.Rdata")
> save(specificResult, file="specificResult.RData")
```

Session Info

```
> toLatex(sessionInfo())
```

- R version 4.5.2 (2025-10-31), x86_64-pc-linux-gnu
- Locale: LC_CTYPE=en_US.UTF-8, LC_NUMERIC=C, LC_TIME=en_US.UTF-8, LC_COLLATE=C, LC_MONETARY=en_US.UTF-8, LC_MESSAGES=en_US.UTF-8, LC_PAPER=en_US.UTF-8, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8, LC_IDENTIFICATION=C
- Time zone: Etc/UTC
- TZcode source: system (glibc)
- Running under: Ubuntu 24.04.3 LTS
- Matrix products: default
- BLAS: /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
- LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblas-p0.3.26.so ; LAPACK version3.12.0
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: Biobase 2.71.0, BiocGenerics 0.57.0, RColorBrewer 1.1-3, SpeCond 1.65.0, fields 17.1, generics 0.1.4, hwriter 1.3.2.1, mclust 6.1.2, spam 2.11-1, viridisLite 0.4.2
- Loaded via a namespace (and not attached): Rcpp 1.1.0.8.1, buildtools 1.0.0, cli 3.6.5, compiler 4.5.2, dotCall64 1.2, evaluate 1.0.5, grid 4.5.2, knitr 1.51, maketools 1.3.2, maps 3.4.3, rlang 1.1.6, sys 3.4.3, tools 4.5.2, xfun 0.55

8 References

Florence MG Cavalli, Juan-Manuel Vaquerizas, Richard Bourgon, Nicholas M Luscombe (in preparation)

C. Fraley and A. E. Raftery, Model-based clustering, discriminant analysis, and density estimation, *Journal of the American Statistical Association*, Vol. 97, pages 611-631 (2002).

C. Fraley and A. E. Raftery, MCLUST Version 3 for R: Normal Mixture Modeling and Model-based Clustering, Technical Report No. 504, Department of Statistics, University of Washington, September 2006.