

maSigPro User's Guide

Ana Conesa and María J. Nueda

4 September 2017

1. Centro de Investigacion Principe Felipe, Valencia, Spain.
aconesa@cipf.es
2. Departamento de Matemáticas, Universidad de Alicante, Spain.
mj.nueda@ua.es

Contents

1	Introduction	2
2	Getting started	2
3	Multiple Series Time Course Experiment	2
3.1	Defining the regression model	5
3.2	Finding significant genes	5
3.3	Finding significant differences	5
3.4	Obtaining lists of significant genes	6
4	Graphical display	7
4.1	Venn Diagrams	7
4.2	see.genes()	7
4.3	PlotGroups()	8
4.4	PlotProfiles()	11
5	Other designs	11
5.1	Single Series Time Course	11
5.2	Common Starting Time	12
6	Next Generation-Sequencing series	13
7	Iso-maSigPro: analysis of alternative isoform expression in time course transcriptomics experiments	15
7.1	IsoModel() and getDS()	15
7.2	Clustering strategy: seeDS() and tableDS()	17
7.3	PodiumChange()	19
7.4	IsoPlot()	20

1 Introduction

maSigPro is a R package that initially was developed for the analysis of single and multiseres time course microarray experiments (Conesa et al., 2006). **maSigPro** has been adapted to deal also with Next Generation-Sequencing (NGS) series of data in a proper way (Nueda et al., 2014). The usage of this option is explained in 6 section.

maSigPro also includes several tools for the analysis of alternative isoform expression in time course transcriptomics experiments. In 7 section the usage of this option is explained.

maSigPro follows a two steps regression strategy to find genes with significant temporal expression changes and significant differences between experimental groups. The method defines a general regression model for the data where the experimental groups are identified by dummy variables. The procedure first adjusts this global model by the least-squared technique to identify differentially expressed genes and selects significant genes applying false discovery rate control procedures. Secondly, stepwise regression is applied as a variable selection strategy to study differences between experimental groups and to find statistically significant different profiles. The coefficients obtained in this second regression model will be useful to cluster together significant genes with similar expression patterns and to visualize the results. This document is a example-based guide for the use of **maSigPro**. We recommend to open a R session and go through this tutorial running the code given at the different sections. The guide does not provide a detailed description of the functions of the package or demonstrates the statistical basis of the methodology. The later is described in the work by (Conesa et al., 2006) and (Nueda et al., 2014).

2 Getting started

The **maSigPro** package can be obtained from the Bioconductor repository or downloaded from <http://www.ua.es/personal/mj.nueda> and <http://bioinfo.cipf.es/downloads>. Load **maSigPro** by typing at the R prompt:

```
> library(maSigPro) # load maSigPro library
```

The on-line help of **maSigPro** can be started by typing at the R prompt:

```
>help(package="maSigPro") #for package help
>?p.vector #for function help
```

The analysis approach implemented in **maSigPro** is executed in 5 major steps (Figure 1) which are run by the package core functions **make.design.matrix()**, **p.vector()**, **T.fit()**, **get.siggenes()** and **see.genes()**. Additionally, the package provides the wrapping function **maSigPro()** which executes the entire analysis in one go.

In the following section we will explain the usage of each of these functions using as example a data set from a multiple series time course experiment. At the end of this document we will also explain how to apply **maSigPro** to other experimental designs.

3 Multiple Series Time Course Experiment

For this section we will use a public data set from a plant abiotic stress study performed at the TIGR Institute by (Rensink et al., 2005). In this study, potato plants were subjected

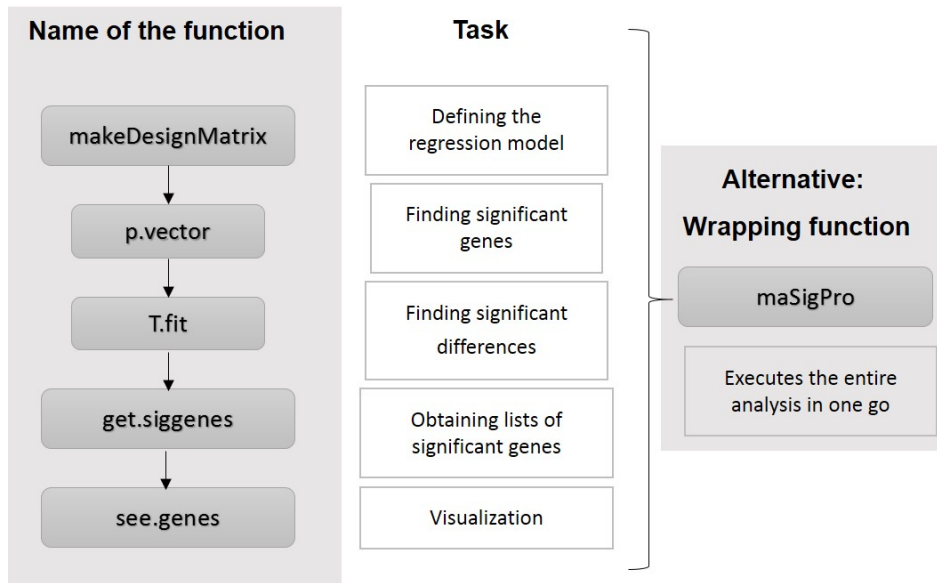


Figure 1: maSigPro pipeline. Major functions.

to three different types of abiotic stresses and gene expression was monitored at three time points after the start of the treatments. RNA was also collected from non-stressed plants at the same time points and all samples were hybridised against a common control on a 11K cDNA potato chip. There are three biological replicates for each experimental condition. For speed in this example we will use a random 1000 genes data subset of this study. This data set is part of the data supplied by the **maSigPro** package. The original data can be found at <http://www.tigr.org/tdb/potato/index.shtml>.

Before proceeding with this tutorial we will define some of the terms that will be used along this document. We denote experimental groups as the experimental factor for which temporal profiles are defined, like "*Treatment A*", "*Tissue1*", etc; conditions are each experimental group vs. time combination like "*Treatment A at Time 0*" conditions can have or not replicates. Variables are the regression variables defined by the **maSigPro** approach for the experiment regression model. **maSigPro** defines dummy variables to model differences between experimental groups. Dummy variables, Time and their interactions are the variables of the regression model. Load the data in your workspace:

```
> data(data.abiotic)
> data(edesign.abiotic)
```

The `edesign.abiotic` object describes the experimental design of this experiment in **maSigPro** format.

```
> edesign.abiotic
```

	Time	Replicate	Control	Cold	Heat	Salt
Control_3H_1	3	1	1	0	0	0
Control_3H_2	3	1	1	0	0	0

Control_3H_3	3	1	1	0	0	0
Control_9H_1	9	2	1	0	0	0
Control_9H_2	9	2	1	0	0	0
Control_9H_3	9	2	1	0	0	0
Control_27H_1	27	3	1	0	0	0
Control_27H_2	27	3	1	0	0	0
Control_27H_3	27	3	1	0	0	0
Cold_3H_1	3	4	0	1	0	0
Cold_3H_2	3	4	0	1	0	0
Cold_3H_3	3	4	0	1	0	0
Cold_9H_1	9	5	0	1	0	0
Cold_9H_2	9	5	0	1	0	0
Cold_9H_3	9	5	0	1	0	0
Cold_27H_1	27	6	0	1	0	0
Cold_27H_2	27	6	0	1	0	0
Cold_27H_3	27	6	0	1	0	0
Heat_3H_1	3	7	0	0	1	0
Heat_3H_2	3	7	0	0	1	0
Heat_3H_3	3	7	0	0	1	0
Heat_9H_1	9	8	0	0	1	0
Heat_9H_2	9	8	0	0	1	0
Heat_9H_3	9	8	0	0	1	0
Heat_27H_1	27	9	0	0	1	0
Heat_27H_2	27	9	0	0	1	0
Heat_27H_3	27	9	0	0	1	0
Salt_3H_1	3	10	0	0	0	1
Salt_3H_2	3	10	0	0	0	1
Salt_3H_3	3	10	0	0	0	1
Salt_9H_1	9	11	0	0	0	1
Salt_9H_2	9	11	0	0	0	1
Salt_9H_3	9	11	0	0	0	1
Salt_27H_1	27	12	0	0	0	1
Salt_27H_2	27	12	0	0	0	1
Salt_27H_3	27	12	0	0	0	1

Note that arrays are given in rows and experiment descriptors are provided in columns. The first column shows the value that variable Time takes in each array. Replicates column is an index column that indicates the replicated arrays: all arrays belonging to the same experimental condition must be given the same number. The remaining columns are binary columns that give the assignment of arrays to experimental groups. There are as many binary columns as experimental groups and arrays take the value 1 or 0 whether they belong or not to that experimental group.

The `data.abiotic` object is a matrix with normalized gene expression data. Genes must be in rows and arrays in columns. **maSigPro** uses row and column names of the data and edesign objects throughout the package. Array names are the labels of the rows of the edesign object and columns of the data object that must be in the same order. GeneIDs are the labels of the rows in the data object. And experiment descriptors are put in the column names of

the edesign object.

```
> colnames(data.abiotic)
> rownames(edesign.abiotic)
> colnames(edesign.abiotic)
> rownames(data.abiotic)
```

3.1 Defining the regression model

Create a regression matrix for the full regression model:

```
> design <- make.design.matrix(edesign.abiotic, degree = 2)
```

This example has three time points, so we can consider up to a quadratic regression model (`degree = 2`). Larger number of time points would potentially allow a higher polynomial degree. `design` is a list. Its element `dis` is the actual regression design matrix. `groups.vector` contains the assignment of regression variables to experimental groups.

```
> design$groups.vector

[1] "ColdvsControl" "HeatvsControl" "SaltvsControl" "Control"
[5] "ColdvsControl" "HeatvsControl" "SaltvsControl" "Control"
[9] "ColdvsControl" "HeatvsControl" "SaltvsControl"
```

3.2 Finding significant genes

The next step is to compute a regression fit for each gene. This is done by the function `p.vector()`. This function also computes the p -value associated to the F -Statistic of the model, which is used to select significant genes. By default **maSigPro** corrects this p -value for multiple comparisons by applying the linear step-up (B-H) false discovery rate (FDR) procedure (Benjamini and Hochberg, 1995). This procedure can be modified by choosing another option of the `p.adjust` function that is controlled by the function parameter `MT.adjust` of `p.vector`. The level of FDR control is given by the function parameter `Q`.

```
> fit <- p.vector(data.abiotic, design, Q = 0.05, MT.adjust = "BH", min.obs = 20)
```

`p.vector()` returns a list of values:

```
> fit$i # returns the number of significant genes
> fit$alfa # gives p-value at the Q false discovery control level
> fit$SELEC # is a matrix with the significant genes and their expression values
```

3.3 Finding significant differences

Once significant genes have been found, **maSigPro** applies a variable selection procedure to find significant variables for each gene. This will ultimately be used to find which are the profile differences between experimental groups. This step is done by the `T.fit()` function.

```
> tstep <- T.fit(fit, step.method = "backward", alfa = 0.05)
```

T.fit() executes stepwise regression. The `step.method` can be "backward" or "forward" indicating whether the step procedure starts from the model with all or none variables. Use method "two.ways.backward" and "two.ways.forward" to allow variables to both get in and out. At each regression step the p -value of each variable is computed and variables get in/out the model when this p -value is lower or higher than the given cut-off value `alfa`. `tstep` is also a list. Its element `sol` is a matrix of statistical results obtained by the stepwise regression. For each selected gene the following values are given:

- p-value of the regression ANOVA
- R-squared of the model
- p-value of the regression coefficients of the selected variables

3.4 Obtaining lists of significant genes

The following step is to generate lists of significant genes according to the way we want to see results. This is done by the function `get.siggenes()`. This function has two major arguments, `rsq` and `vars`.

`rsq`: is a cut-off value for the R-squared of the regression model.

`vars`: is used to indicate how to group variables to show results. There are 3 possible values:

- `groups`: This will generate a list of significant genes for each experimental group. The list corresponding to the reference group will contain genes whose expression profile is significantly different from a 0 profile. The lists corresponding to the remaining experimental groups will contain genes whose profiles are different from the reference group.
- `all`: One unique list of significant genes a any model variable will be produced.
- `each`: There will be as many lists as variables in the regression model. This can be used to analyze specific differences, for example genes that have linear or saturation kinetics.

```
> sigs <- get.siggenes(tstep, rsq = 0.6, vars = "groups")
> names(sigs)
```

```
[1] "sig.genes" "summary"
```

The element `summary` is a data frame containing the significant genes for the selected `vars`. The element `sig.genes` is a list with all the information needed for the graphical display explained in the following section.

You can further explore your results by:

```
> names(sigs$sig.genes)
[1] "Control"          "ColdvsControl"  "HeatvsControl"  "SaltvsControl"

> names(sigs$sig.genes$ColdvsControl)
[1] "sig.profiles"  "coefficients"  "group.coeffs"  "sig.pvalues"
[5] "g"             "edesign"       "groups.vector"
```

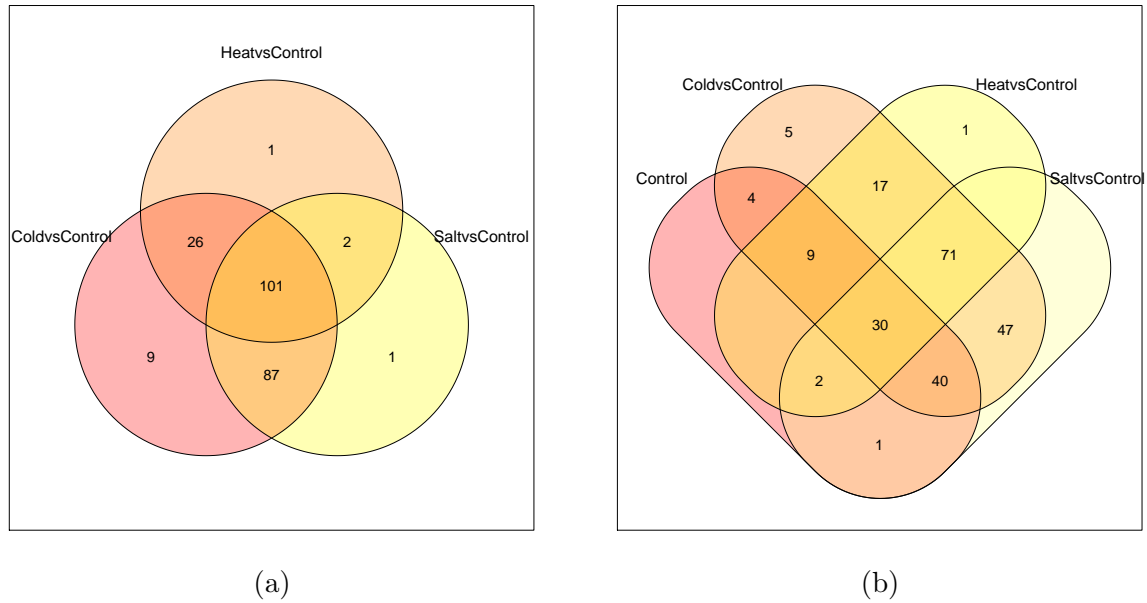


Figure 2: (a) Venn Diagram for Cold, Heat and Salt stress significant genes. (b) Venn Diagram adding also the Control group.

4 Graphical display

maSigPro has several functions for the visual exploration of the results: Venn diagrams, profiles of specific genes and profiles of groups of genes from different perspectives. The main principal function is `see.genes` that makes a clustering analysis and call `PlotProfiles` and `PlotGroups` functions to visualize the clusters.

4.1 Venn Diagrams

`suma2Venn()` function displays the `summary` result, obtained with `get.siggenes()` function, as a Venn diagram. This can be a first view of the obtained results.

Figure 2 shows 2 Venn diagrams made with the following commands. The first one is a Venn diagram of the significant genes for the three stress experimental groups and the second one includes also the control group.

```
> suma2Venn(sigs$summary[, c(2:4)])
```

```
> suma2Venn(sigs$summary[, c(1:4)])
```

4.2 see.genes()

Use `see.genes()` to visualize the result of a group of genes, for example, to visualize the significant genes obtained as significant in the previous step in *ControlvsSalt*, that are genes with significant differences between Salt and Control gene expression.

```
> sigs$sig.genes$SaltvsControl$g
```

```
[1] 191
```

```
see.genes(sigs$sig.genes$ColdvsControl, show.fit = T, dis = design$dis,  
cluster.method="hclust" , cluster.data = 1, k = 9)
```

see.genes() performs a cluster analysis to group genes by similar profiles. Main arguments for cluster analysis are:

k: number of clusters for data partitioning. By default it is 9. Mclust cluster method can compute an optimal k, choosing `k.mclust=TRUE`.

cluster.method: clustering method for data partitioning. `hclust`, `kmeans` and `Mclust` are supported.

distance: distance measurement function when `cluster.method` is `hclust`. By default it is `'cor'` to compute a distance based on the correlation because we are interested in similar trends or changes.

agglo.method: aggregation method used when `cluster.method` is `hclust`. By default `ward.D`.

The resulting clusters are then plotted in two fashions: as experiment-wide expression profiles and as by-groups profiles. The first plot (Figure 3) will help to evaluate the consistency of the clusters while the second plot shows clearly the differences between groups (Figure 4).

4.3 PlotGroups()

As it is already explained, function `Plot Groups` is used for `see.genes`. Here we are going to show the performance of this secondary function because they can also be useful out of the clustering display.

PlotGroups() creates a plot of gene expression profiles where time-factor is in the x-axis, gene expression in the y-axis, gene expression of the same experimental group is represented with the same colour and it is drawn a line that join the averages of each time-group to see the trend of each experimental group through time. This plot can be made for specific genes or for groups of genes where the median will be computed. For example, `STMDE66` is a gene which shows significant profile differences between the control and the cold and salt strees experimental groups, but not significant differences between control and heat experimental groups (Figure 5).

```
> STMDE66 <- data.abiotic[rownames(data.abiotic)=="STMDE66", ]
```

```
> PlotGroups (STMDE66, edesign = edesign.abiotic)
```

We can also add to the plot the regression curve computed for this gene.

```
> PlotGroups (STMDE66, edesign = edesign.abiotic, show.fit = T,  
+ dis = design$dis, groups.vector = design$groups.vector)
```

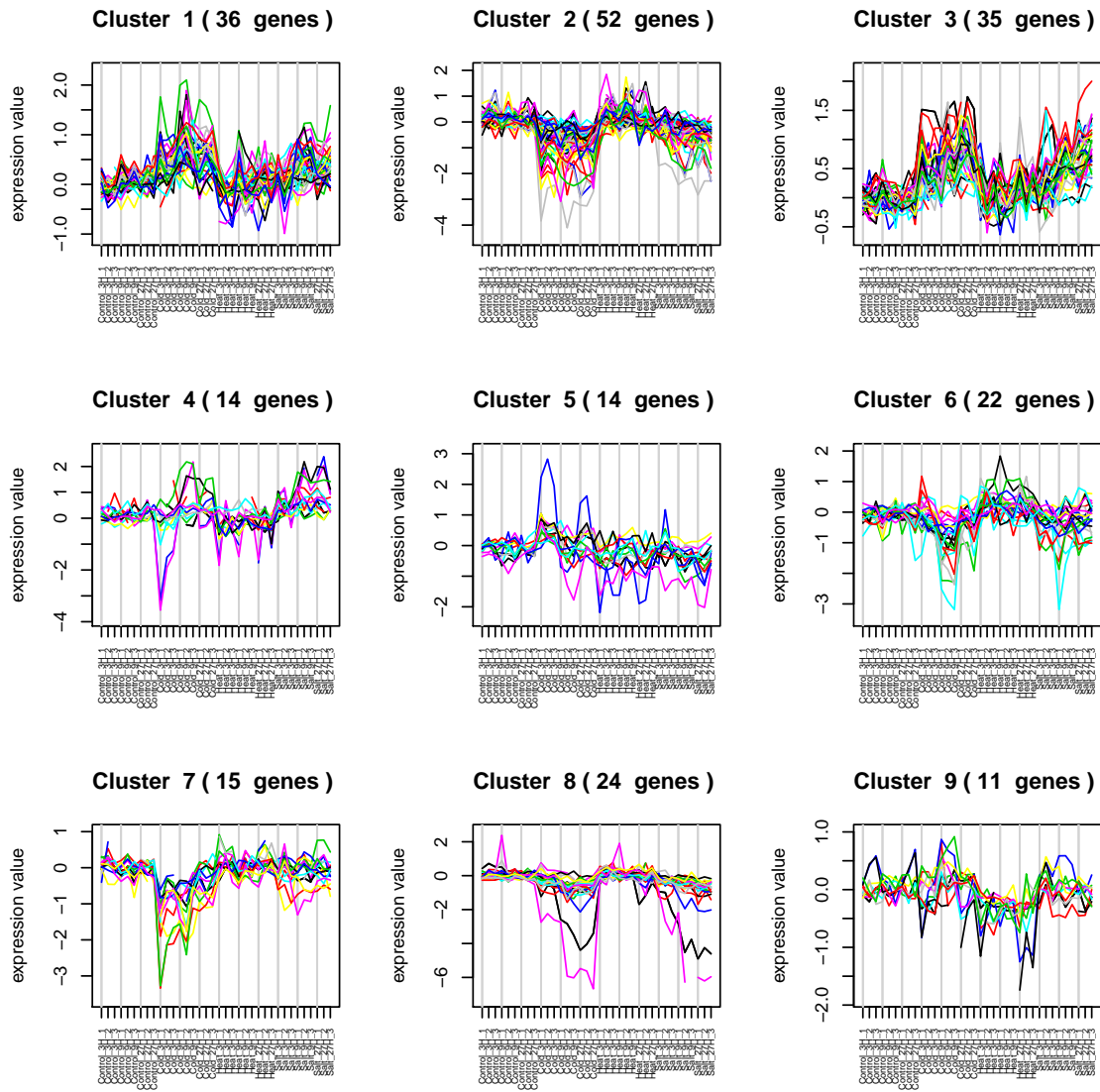



Figure 3: Cluster Analysis ColdvsControl significant genes

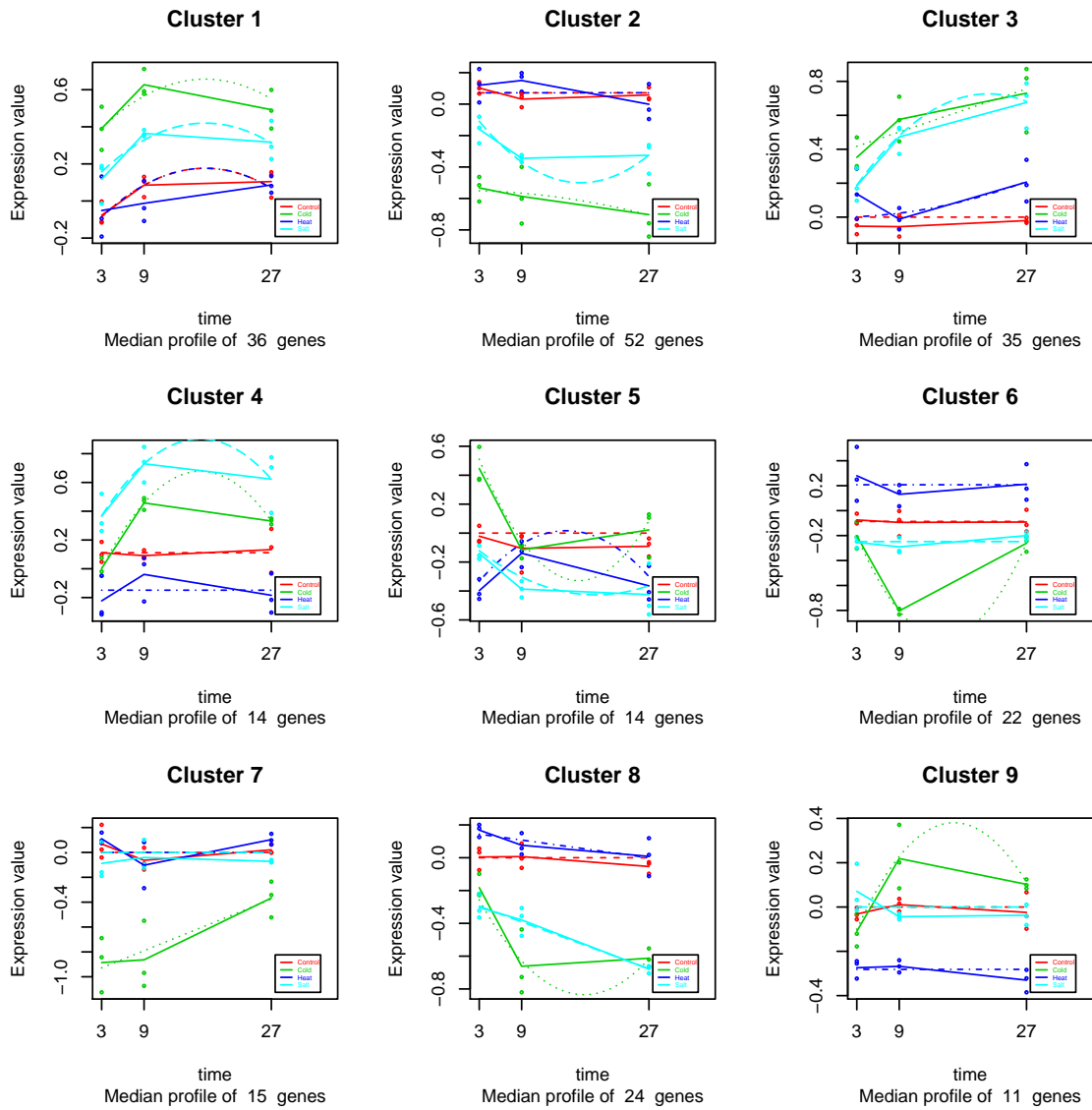


Figure 4: Expression Profiles ColdvsControl significant genes

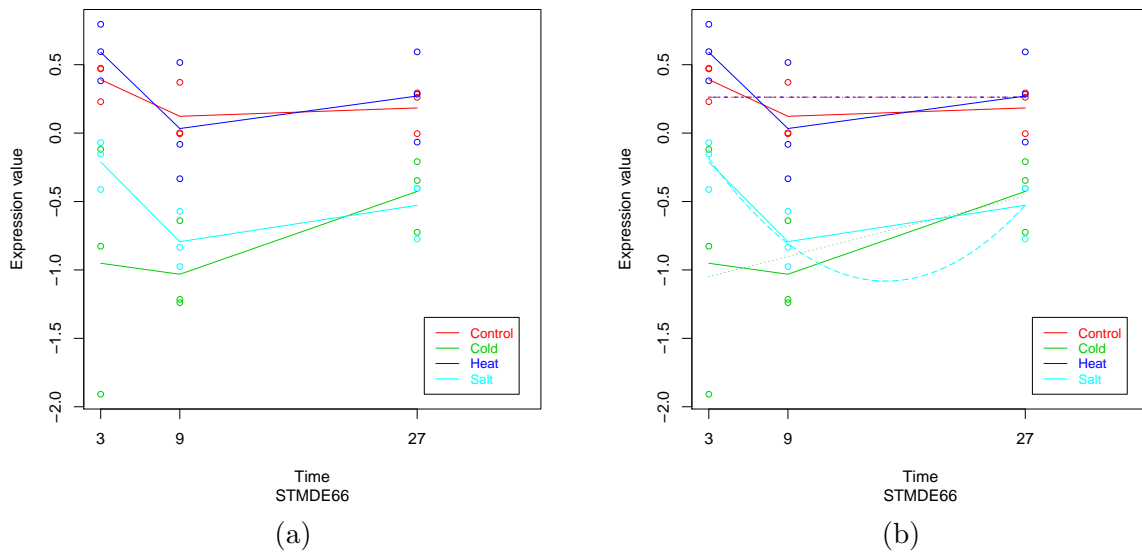


Figure 5: PlotGroups of gene STMDE66 without and with display of regression curves

4.4 PlotProfiles()

This secondary function creates a plot of gene expression profiles where x-axis represents the order of the columns of the data matrix and the y-axis represents the gene expression. When a group of genes is plotted they will be represented with different colours. The main goal of this graph is to check the homogeneity of the clusters.

5 Other designs

5.1 Single Series Time Course

The use of **maSigPro** in Single Series Time Course experiment is straightforward. Make a **edesign** object with just one group column containing all 1s and proceed as described above. Note that when using the **get.siggenes()** function the options "all" and "groups" of the argument **vars** will return the same result. You can use option "each" to analyze the type of responses present in the significant genes: significant genes at the "intercept" term will have a significant expression value at the starting time; genes associated to the variable "Time" will have a significant linear component, which can be induction or repression depending on the sign of their coefficient; genes associated to the variable "Time2" will show a change in the linear response that might be indicating transitory or saturation responses, etc...

Here follows an example of a Single Series analysis.

```
## make a single series edesign
> Time <- rep(c(1,5,10,24), each = 3)
> Replicates <- rep(c(1:4), each = 3)
> Group <- rep(1,12)
```

```

> ss.edesign <- cbind(Time,Replicates,Group)
> rownames(ss.edesign) <- paste("Array", c(1:12), sep = "")
## Create data set
> ss.GENE <- function(n, r, var11 = 0.01, var12 = 0.02, var13 = 0.02,
  var14 = 0.02, a1 = 0, a2 = 0, a3 = 0, a4 = 0) {
  tc.dat <- NULL
  for (i in 1:n) {
    gene <- c(rnorm(r, a1, var11), rnorm(r, a1, var12),
      rnorm(r, a3, var13), rnorm(r, a4, var14))
    tc.dat <- rbind(tc.dat, gene)
  }
  tc.dat }
> flat <-ss.GENE(n = 85, r = 3) # flat
> induc <- ss.GENE(n = 5, r = 3, a1 = 0, a2 = 0.2, a3 = 0.6, a4 = 1) # induction
> sat <- ss.GENE(n = 5, r = 3, a1 = 0, a2 = 1, a3 = 1.1, a4 = 1.2) # saturation
> ord <- ss.GENE(n = 5, r = 3, a1 = -0.8, a2 = -1, a3 = -1.3, a4 = -0.9) # intercept
> ss.DATA <- rbind(flat, induc,sat,ord)
> rownames(ss.DATA) <- paste("feature", c(1:100), sep = "")
> colnames(ss.DATA) <- paste("Array", c(1:12), sep = "")
# run maSigPro
> ss.example <- maSigPro(ss.DATA, ss.edesign, vars="each")

```

5.2 Common Starting Time

The following example illustrates how to build the `edesign` matrix when a common 0 time is applicable to the different experimental groups.

```
> data(edesignCT)
```

In this example `Array1` and `Array2` do not belong to any treatment. They are a common reference for all groups, values without any treatment at time 0.

6 Next Generation-Sequencing series

maSigPro uses `lm()` function to fit a linear model where statistics for inference use normal distribution. This is the right treatment when dealing with normal distributed data or big samples. However, for non-normal small samples, results can not be correct.

The statistical distribution for tag counts data as RNA-Seq data may be Poisson or Binomial. However, the overdispersion of the data suggest the Negative Binomial (NB) can model these distributions in a better way. This distribution depends on a θ parameter to model the overdispersion that is related to the mean(μ) in the following way:

$$Y \sim NB(\theta), Var(Y) = \mu + \frac{\mu^2}{\theta}$$

maSigPro has been adapted to take into account non-normal distribution of the data (Nueda et al., 2014). New arguments have been added in `p.vector` function to deal with this type of data in a proper way:

counts: a logical indicating whether your data are counts. By default is FALSE for microarray treatment.

theta: θ parameter for negative.binomial family. By default $\theta = 10$.

family: the distribution function to be used in the GLM. It must be specified as a function: `gaussian()`, `poisson()`, `negative.binomial(theta)`... If NULL family will be `negative.binomial(theta)` when `counts=TRUE` or `gaussian()` when `counts=FALSE`.

The recommended analysis to deal with RNA-Seq data is the GLM with `negative.binomial` family. θ must be specified and it can be computed by using available methods as `edgeR` ((Robinson et al., 2010)). The application of **maSigPro** with several values of θ to the same datasets did not reveal significant differences in gene selection. Taking this into consideration we have put by default $\theta = 10$ for being an average value. Moreover we give to the user the option of applying whatever exponential family to explore other possibilities.

Data must be normalized before the application of **maSigPro** as it is not integrated any normalized method.

NBdata is a subset of a bigger normalized dataset with 2 experimental groups, 6 time-points and 3 replicates. Simulation has been done by using a negative binomial distribution with $\theta = 5$ to illustrate this section. The first 20 genes are simulated with changes among time. **NBdesign** is the design matrix.

```
> data(NBdata)
> data(NBdesign)

> d <- make.design.matrix(NBdesign)
```

If we can use **maSigPro** with $theta = 10$:

```
> library(MASS)
> NBp <- p.vector(NBdata, d, counts=TRUE)
> NBt <- T.fit(NBp)
> get<-get.siggenes(NBt, vars="all")
> get$summary
```

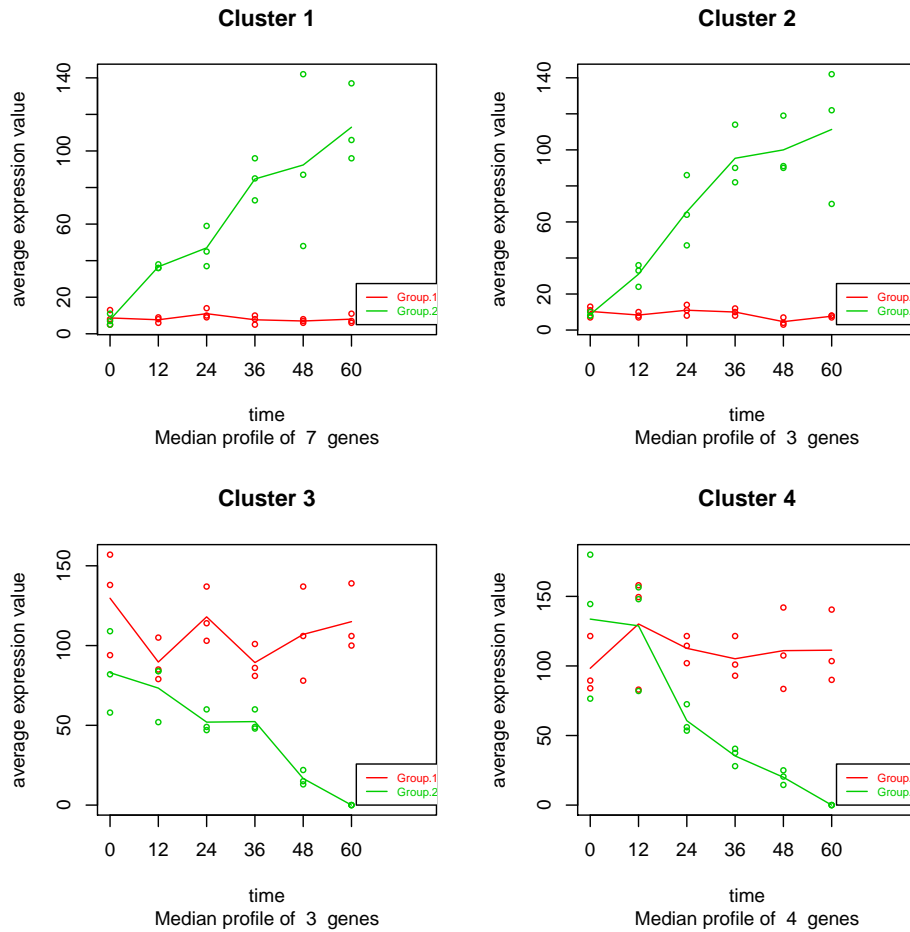


Figure 6: Cluster Analysis significant genes

These genes can be grouped in 4 clusters 6:

```
see.genes(get$sig.genes, k = 4)
```

If we can use maSigPro with a specific θ , in this case, $\theta = 5$:

```
> NBp <- p.vector(NBdata, d, counts=TRUE, theta=5)
```

Also, a specific family, for instance, poisson can be specified. In such case counts and theta parameters are omitted.

```
> NBp <- p.vector(NBdata, d, family=poisson() )
```

Changing this arguments in p.vector is enough. Further functions will take these arguments from a p.vector object.

7 Iso-maSigPro: analysis of alternative isoform expression in time course transcriptomics experiments

maSigPro package includes several tools for the analysis of alternative isoform expression in time course transcriptomics experiments. Figure 7 shows the analysis pipeline with functions involved in this analysis.

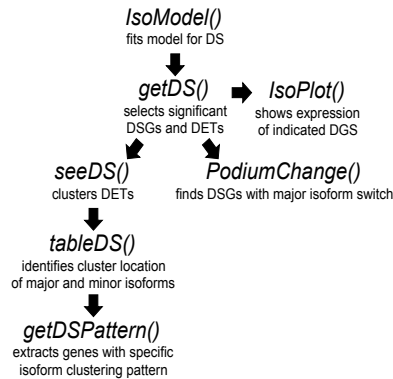


Figure 7: ISO-maSigPro pipeline.

For Iso-maSigPro analysis, data must be provided as a transcript-level expression data set, similarly to regular maSigPro. This `data.frame` must include TranscriptIDs as rownames and the GeneID of each transcript in the first column. The Iso-maSigPro package does not include functions for quantification of isoform expression.

The package provides a test dataset for differential splicing analysis named `ISOdata`. It consists of 2782 isoforms (rows) belonging to 1009 genes. The experimental design is a 2 series experiment with 6 time points and three replicates per experimental condition. First column of `ISOdata` contains the name of the gene each isoform belongs to, while the remaining columns are the RNA-Seq data samples associated to the experimental conditions.

```
> data(ISOdata)
> data(ISOdesign)

> dis <- make.design.matrix(ISOdesign)
```

7.1 `IsoModel()` and `getDS()`

First of all `IsoModel()` function is applied to identify Differentially Spliced Genes (DSGs) and also Differentially Expressed Transcripts (DETs), which are transcripts of significant DSGs detected as statistically significant with regular NextmaSigPro. `IsoModel()` returns a list of DSGs and all the information about the estimated models of associated isoforms to be used as input in `getDS()` function to obtain a selection of DSGs at a preestablished level of goodness of fit for each model.

```
> MyIso <- IsoModel(data=ISOdata[,-1], gen=ISOdata[,1], design=dis, counts=TRUE)
```

```

[1] "2782 transcripts"
[1] "1009 genes"
[1] "fitting gene 100 out of 568"
[1] "fitting gene 200 out of 568"
[1] "fitting gene 300 out of 568"
[1] "fitting gene 400 out of 568"
[1] "fitting gene 500 out of 568"
[1] "fitting isoform 100 out of 241"
[1] "fitting isoform 200 out of 241"
[1] "fitting isoform 100 out of 174"
[1] "Influence: 17 genes with influential data at slot influ.info."

```

```
> Myget <- getDS(MyIso)
```

```

[1] "51 DSG selected"
[1] "97 DETs selected"
DSG_distributed_by_number_of_DETs
  1  2  3  4  9 10
17 21  1  4  1  1

```

When applying `getDS()` function the number of DSGs and DETs are showed by console and also a table detailing the number of DETs that each selected DSG contains. Names of DSGs and DETs can be showed with:

```
> Myget$DSG
```

```

[1] "Gene239" "Gene1009" "Gene1005" "Gene800" "Gene64" "Gene1003"
[7] "Gene1006" "Gene440" "Gene63" "Gene41" "Gene1008" "Gene1001"
[13] "Gene500" "Gene860" "Gene145" "Gene793" "Gene236" "Gene857"
[19] "Gene682" "Gene107" "Gene696" "Gene96" "Gene409" "Gene1002"
[25] "Gene901" "Gene427" "Gene684" "Gene116" "Gene129" "Gene852"
[31] "Gene1007" "Gene927" "Gene638" "Gene850" "Gene342" "Gene733"
[37] "Gene1004" "Gene836" "Gene491" "Gene571" "Gene183" "Gene611"
[43] "Gene368" "Gene463" "Gene390" "Gene782" "Gene737" "Gene995"
[49] "Gene285" "Gene652" "Gene595"

```

```
> Myget$DET
```

```

[1] "Transcript122" "Transcript125" "Transcript147" "Transcript150"
[5] "Transcript269" "Transcript270" "Transcript416" "Transcript422"
[9] "Transcript423" "Transcript445" "Transcript526" "Transcript528"
[13] "Transcript698" "Transcript701" "Transcript702" "Transcript950"
[17] "Transcript951" "Transcript985" "Transcript987" "Transcript988"
[21] "Transcript1016" "Transcript1034" "Transcript1035" "Transcript1036"
[25] "Transcript1037" "Transcript1038" "Transcript1039" "Transcript1040"
[29] "Transcript1041" "Transcript1042" "Transcript1043" "Transcript1073"
[33] "Transcript1074" "Transcript1077" "Transcript1087" "Transcript1088"
[37] "Transcript1151" "Transcript1152" "Transcript1193" "Transcript1194"

```



```

[41] "Transcript1195" "Transcript1196" "Transcript1296" "Transcript1300"
[45] "Transcript1373" "Transcript1374" "Transcript1377" "Transcript1393"
[49] "Transcript1395" "Transcript1396" "Transcript1397" "Transcript1398"
[53] "Transcript1400" "Transcript1466" "Transcript1467" "Transcript1468"
[57] "Transcript1519" "Transcript1722" "Transcript1723" "Transcript1797"
[61] "Transcript1836" "Transcript1876" "Transcript1899" "Transcript1900"
[65] "Transcript2063" "Transcript2067" "Transcript2268" "Transcript2298"
[69] "Transcript2322" "Transcript2323" "Transcript2324" "Transcript2326"
[73] "Transcript2330" "Transcript2331" "Transcript2332" "Transcript2333"
[77] "Transcript2334" "Transcript2341" "Transcript2342" "Transcript2405"
[81] "Transcript2413" "Transcript2414" "Transcript2416" "Transcript2417"
[85] "Transcript2460" "Transcript2461" "Transcript2462" "Transcript2463"
[89] "Transcript2472" "Transcript2482" "Transcript2483" "Transcript2485"
[93] "Transcript2486" "Transcript2633" "Transcript2634" "Transcript2731"
[97] "Transcript2732"

```

Note that it is possible that a gene called DSG but no significant DET can be found under the significance level, goodness of fit and multiple testing correction constraints of the regular maSigPro analysis. Names of such cases can be shown with:

```
> Myget$List0
```

```
[1] "Gene857" "Gene491" "Gene611" "Gene995" "Gene285" "Gene652"
```

7.2 Clustering strategy: `seeDS()` and `tableDS()`

The Iso-maSigPro clustering approach identifies groups of DSGs with similar isoform expression patterns, as well as the expression profiles of DETs within DSGs. This strategy is implemented in two steps corresponding to `seeDS()` and `tableDS()` functions.

Function `seeDS` looks for general transcript expression trends analysing DETs and clusters them into k groups with any of the available maSigPro clustering approaches (`cluster.method` input can be: `hclust`, `kmeans` or `Mclust`, the most used clustering methods). The clustering can be applied to only DETs from DSGs (`cluster.all=FALSE`) or after computing DETs from all available genes (`cluster.all=TRUE`), which is recommended when the interest is to characterize the expression pattern of the whole transcriptomics experiment and not only from DSGs. Figure 8 shows the clusters obtained for DETs in our example.

```
> see <- seeDS(Myget, cluster.all=FALSE, k=6)
```

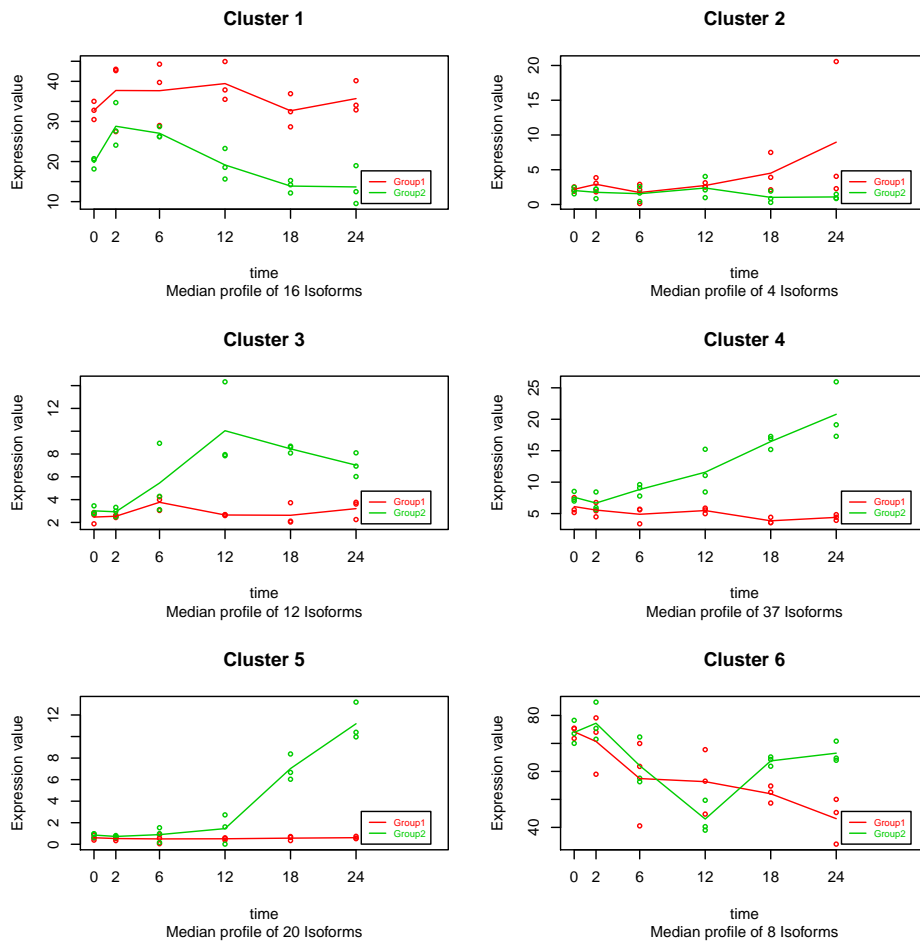


Figure 8: Cluster Analysis Differentially Expressed Transcripts (DETs)

Next, the `tableDS()` function takes DSGs with 2 or more DETs to identify the time-course Alternative Splicing events. To do this, each DET is labeled as major isoform (defined as the most expressed isoform across conditions) or minor isoform and for each DSG the clusters where their major and minor DET(s) belong are identified. This information is used to compute a classification table that indicates the distribution of DETs of DSG across different clusters.

```
> table <- tableDS(see)
> table$IsoTable
```

Cluster.Mayor	Cluster.minor							
	1	1&5	2	3	3&4	4	5	
1	0	1	1	1	0	3	1	
2	1	0	0	0	0	0	0	
3	0	0	1	0	1	2	0	
4	0	0	0	1	2	4	2	
5	0	0	0	0	0	0	2	
6	0	0	0	2	0	1	2	

By comparing the classification table with the cluster profiles, the user can readily identify genes with strong or subtle expression differences among their set of isoforms. For instance, in Figure 8 it is observed that cluster 1 and 4 have different trends and `IsoTable` object shows there are 3 genes with major isoform in cluster 1 and minor isoform(s) in cluster 4. The names of these genes can be obtained with the `getDSPatterns()` function:

```
> getDSPatterns(table, 1, 4)

[1] "Gene1002" "Gene1003" "Gene1004"
```

The gene selection can be plotted with `IsoPlot()` function (see example in Figure 9).

7.3 PodiumChange()

Alternatively, `PodiumChange()` function extracts from the data those DSGs that undergo a switch of their most expressed isoform during the time course. `PodiumChange()` can be applied taking into consideration only the DETs (`only.sig.iso=TRUE`) or considering all the isoforms of DSGs. This last option is interesting when the DSG has only one isoform called as DET. The function takes as input the result of `getDS()` and returns a list of genes with podium changes. The function can detect changes at any time point (eventual changes), for an indicated experimental condition or at specific subranges of time and experimental conditions.

```
> PC<-PodiumChange(Myget, only.sig.iso=TRUE, comparison="specific",
+ group.name="Group2", time.points=c(18,24))
> PC$L

[1] "Gene239" "Gene1008" "Gene1005"
```

Again, the gene selection can be plotted with `IsoPlot()` function (see example in Figure 9).

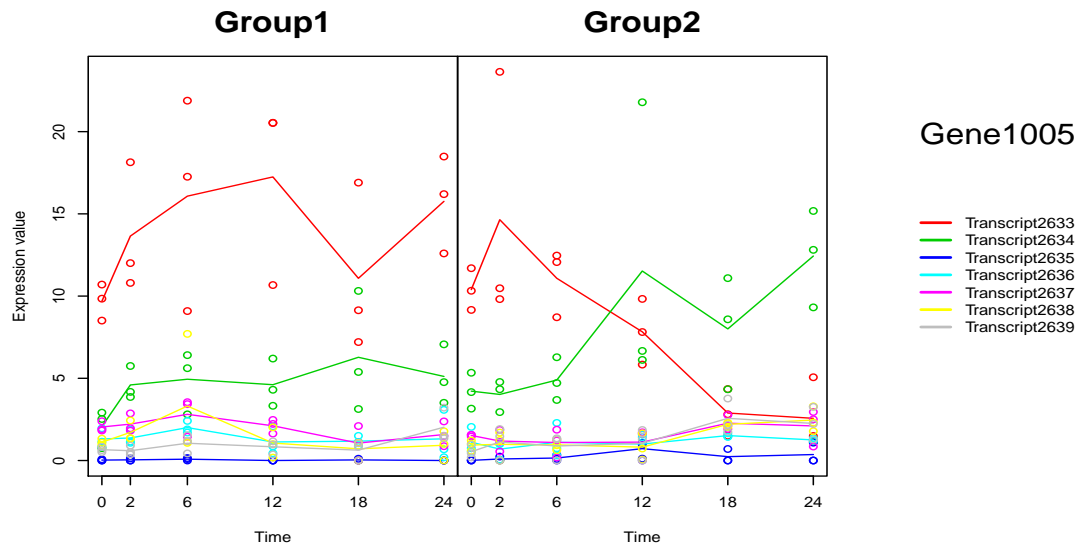


Figure 9: IsoPlot() example. Gene1005 detected as podium-change gene.

7.4 IsoPlot()

This function provides gene-level plots of the expression profiles of all transcripts in the input vector of gene names. Optionally, the user can choose visualizing all transcripts or only DETs of the selected genes. Typically, IsoPlot() can be used to inspect specific genes identified by the PodiumChange() or the tableDS() functions. Figure 9 shows the IsoPlot() of Gene1005 detected as podium-change gene.

```
> IsoPlot(Myget, "Gene1005", only.sig.iso=FALSE, cex.main=2, cex.legend=1)
```

References

- Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society Series B*, 57:289–300, 1995. URL <http://www.tigr.org/tdb/potato/index.shtml>.
- A. Conesa, M. Nueda, A. Ferrer, and M. Talón. maSigPro: a method to identify significantly differential expression profiles in time-course microarray experiments. *Bioinformatics*, 22(9):1096–1102, 2006. URL <http://bioinformatics.oupjournals.org/cgi/content/abstract/22/9/1096>.
- M. Nueda, S. Tarazona, and A. Conesa. Next maSigPro: updating maSigPro bioconductor package for RNA-seq time series. *Bioinformatics*, 30(18):2598–2602, 2014. URL <http://bioinformatics.oxfordjournals.org/content/30/18/2598>.
- W. Rensink, S. Iobst, A. Hart, S. Stegalkina, J. Liu, and C. Buell. Gene expression profiling of potato responses to cold, heat and salt stress. *Funct Integr Genomics*, 5(4):201–207, 2005. URL <http://www.tigr.org/tdb/potato/index.shtml>.
- M. Robinson, D. McCarthy, and G. Smyth. edgeR: a Bioconductor package for differential expression analysis. *Bioinformatics*, 26:139–140, 2010.