FunPat: Function-based Pattern analysis on RNA-seq time series data

Tiziana Sanavia, Francesca Finotello, Barbara Di Camillo

Department of Information Engineering
University of Padova

May 29, 2015

Contents

1 Introduction

Dynamic expression data, nowadays obtained using high-throughput RNA sequencing (RNA-seq), are essential to monitor transient gene expression changes and to study the dynamics of their transcriptional activity in the cell or response to stimuli. `FunPat` is an R package designed to provide:

- a useful tool to analyze time series genomic data;
- a computational pipeline which integrates gene selection, clustering and functional annotations into a single framework to identify the main temporal patterns associated to functional groups of differentially expressed (DE) genes;
- an easy way to exploit different types of annotations from currently available databases (e.g. Gene Ontology) to extract the most meaningful information characterizing the main expression dynamics;
- a user-friendly organization and visualization of the outcome, automatically linking the DE genes and their temporal patterns to the functional information for an easy biological interpretation of the results.

1.1 Citation

```
> citation("FunPat")

Please cite package 'FunPat' in publications using:

  Sanavia T, Finotello F and Di Camillo B (2015). FunPat:
  function-based pattern analysis on RNA-seq time series data. BMC
  Bioinformatics

A BibTeX entry for LaTeX users is

  @Article{,
    title = {FunPat: function-based pattern analysis on RNA-seq time series data},
    author = {Tiziana Sanavia and Francesca Finotello and Barbara Di Camillo},
    journal = {BMC Bioinformatics},
    year = {2015},
  }
```

Other publications illustrating the methods used in FunPat and possible applications are:

- Di Camillo B, Irving BA, Schimke J, Sanavia T, Toffolo G, Cobelli C, Nair KS. Function-based discovery of significant transcriptional temporal patterns in insulin stimulated muscle cells. *PLoS One*, 7(3):e32391, 2012
- Di Camillo B, Toffolo G, Nair SK, Greenlund LJ, Cobelli C. Significance analysis of microarray transcript levels in time series experiments. *BMC Bioinformatics*, 8(Suppl 1):S10, 2007

1.2 How to get help

Most questions about `FunPat` will hopefully be answered by the documentation and the references. Every function mentioned in this vignette has its own help page. For example, a detailed description of the arguments and the output of the function `PatternAnalysis` can be read by typing `?PatternAnalysis` or `help(PatternAnalysis)` at the R prompt. We always appreciate receiving suggestions for possible improvements of the package, as well as reports of bugs in the package functions or in the documentation. For any questions, please contact us by sending an email to Tiziana Sanavia (tiziana.sanavia@dei.unipd.it) and Barbara Di Camillo (barbara.dicamillo@dei.unipd.it).

2 Installation

This software is written in R language, so it is necessary to have R installed on your computer. For more information and download of R, please refer to http://www.r-project.org/. For more information about the installation of R packages, please refer to http://cran.r-project.org/doc/manuals/R-admin.html#Installing-packages. R version 3.0.3 or later is required to be able to install and run `FunPat`. This package is dependent on Bioconductor package `tseries`.

To install `FunPat` and `tseries` from Bioconductor website, open an R session and type:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("tseries")
> biocLite("FunPat") # soon available
```

FunPat package can be also installed from a tar ball as follows:

```
> install.packages("path_to_FunPat/FunPat_0.99.0.tar.gz",
+     repos=NULL, type="source")
```

3 Getting started with `FunPat`

3.1 Overview

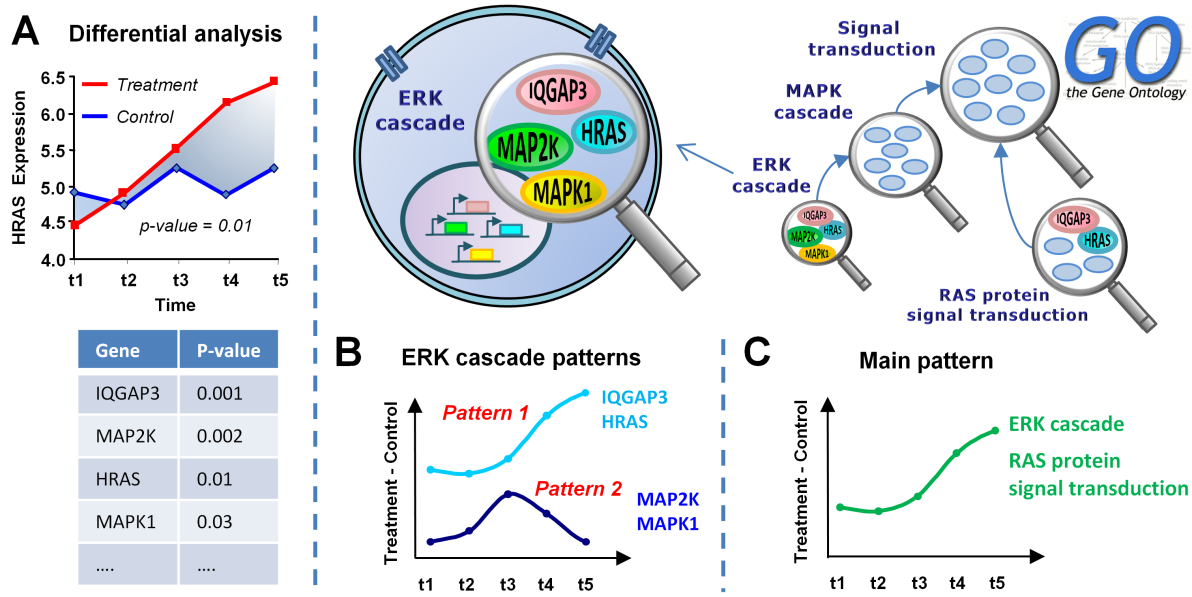
`FunPat` works with transcriptional expression data derived from experimental designs monitoring either two experimental conditions (e.g. treatment *vs.* control) or a single condition *vs.* a baseline (Sections 3.2 for an example and 5 for a detailed description). Starting from these data, `FunPat` offers the following types of analysis:

- differential expression analysis of dynamic expression data (Section 3.3);
- model-based clustering of temporal expression profiles, without requiring the user to fix either *a priori* or *a posteriori* the number of clusters (Section 3.4);
- if prior knowledge from databases (Section 3.5) or from custom-based annotations (Section 6) is available, an integrated selection-clustering analysis to identify DE genes which share common dynamic expression profiles and annotations to a specific biological concept, e.g. a pathway or a Gene Ontology (GO) term (Section 3.6).

Figure 1 gives a glance of what the user can obtain from the analyses:

- a list of genes ranked according to p-values representing their differential expression across time between the experimental conditions (Figure 1-A);
- a set of clusters of DE genes (Figure 1-B), each characterized by a specific temporal pattern and, if genes can be grouped into *Gene Sets* according to common biological annotations, by the most specific biological information (e.g. clusters of genes annotated to the GO term *ERK cascade*);
- if available from annotations, a set of clusters of *Gene Sets* (Figure 1-C) with annotated genes characterized by a common temporal pattern (e.g. the *Main Pattern* common to genes belonging to the GO terms *ERK cascade* and *RAS protein signal transduction*).

`FunPat` allows a user-friendly organization of the outcome (Sections 3.7 and 3.8) and an advanced visualization through HTML pages for an immediate interpretation of the results (see Section 4).
In the following, the application of `FunPat` functions to a toy example is presented, focusing on simulated RNA-seq data. Since `FunPat` is not constrained by any specific statistical distributions, the same analysis pipeline can be also applied to data from different technologies, such as microarrays, as well.



Gene	P-value
IQGAP3	0.001
MAP2K	0.002
HRAS	0.01
MAPK1	0.03
....	....

Figure 1: **Description of `FunPat` output.** Genes are first ranked according to a p-value assigned according to the area bounded by the expression profiles in two experimental conditions, e.g. *treatment vs. control* (A). If prior knowledge is available, genes are then organized into *Gene Sets*, e.g. genes annotated to common GO terms. A model-based clustering is applied to obtain both Gene Set-specific temporal patterns (B), characterizing clusters of genes (e.g. patterns 1 and 2 for the GO term *ERK cascade*), and *Main Patterns* (C), characterizing clusters of Gene Sets (e.g. as for GO terms *ERK cascade* and *RAS protein signal transduction*). If the prior knowledge is hierarchically structured as in GO, the analysis starts from the most specific terms, removing the selected genes from the ancestor nodes to avoid redundancy of information.

3.2 Loading Data

To demonstrate the package functionality, `FunPat` provides a toy example of an RNA-seq study simulating an experimental design where, for each gene, the dynamical response is measured in two conditions: *treatment vs. control* (Figure 2). Data are characterized by N=1000 genes monitored across M=13 time samples[1]. The dataset can be loaded by typing:

```
> library(FunPat)
> data(Simdata)
```

Simdata is a list containing three entries:

- `ctrl`, an N genes $\times$ M samples matrix of time series expression profiles at the *control* condition;
- `treat`, an N genes $\times$ M samples matrix of time series expression profiles at the *treatment* condition;
- `replicates`, a two-columns matrix containing the measurements of the available biological replicates.

Row names of `ctrl` and `treat` matrices correspond to gene Entrez IDs, considering Homo sapiens as model organism. The `replicates` matrix was built by setting in the first and second columns the expression values of `ctrl` and `treat` at time 0, respectively, excluding genes with null expression in both conditions. For a complete guideline on how organizing data and the biological replicates for `FunPat`, see Section 5.

[1]Expression data were simulated according to a Negative Binomial distribution, with dispersion parameter equal to 0.1. The sequencing depths were sampled from a uniform distribution in the interval $[10^6, 10^7]$. Simulated count data were normalized with an approach based on the trimmed mean of M values (TMM) method [1] and then log2-transformed. In particular, the normalization factors were re-scaled by the median of the normalized library sizes. See [2] for further details.

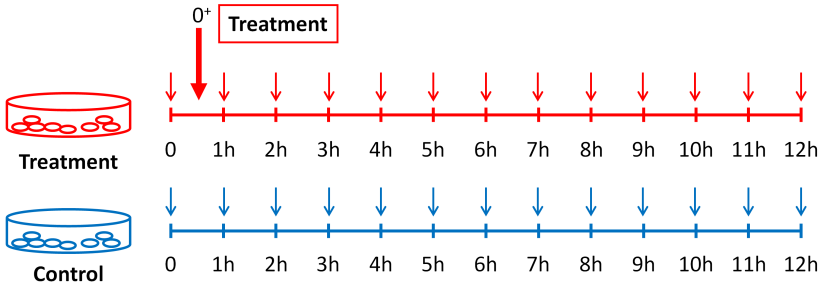


Figure 2: **Simulated experimental design.** The simulated data represent a typical example of experimental design in time series gene expression studies. Samples are collected from two cell cultures measured into two different experimental conditions, in this case *treatment vs. control*. The small arrows indicate the time points at which the gene expression is monitored. The big red arrow indicates when treatment is applied.

3.3 Detection of time-dependent differential expression

In order to rank the genes according to p-values representing their differential expression across time between the two experimental conditions, `FunPat` uses the function `SEL.TS.AREA`. This function calculates, for each gene, the area A of the region bounded by the corresponding time series expression profiles in the two conditions and a p-value is assigned by evaluating the statistical significance of A against a null hypothesis distribution, based on a model describing the biological and technical variability derived from the experimental replicates [3]. The method requires only at least two replicates for a single time point.
Starting from the loaded data `ctrl`, `treat` and `replicates`, a basic usage of `SEL.TS.AREA` is:

```
> replic<-Simdata$replicates
> nC<-Simdata$ctrl
> nD<-Simdata$treat
> rank.res<-SEL.TS.AREA(replicates=replic,data1=nC,data2=nD)
```

The user can specify the time grid at which data are sampled using the argument `sampling.grid`, which must be a vector with lenght equal to the number of columns in `data1` and `data2`. If not specified as above, the time samples are considered equally spaced. It is worth noting that setting time samples with different intervals can strongly affect the results, since larger intervals have a higher weight in the area calculation. To avoid weighting too much some intervals of the time series, the user can keep the default. The simulated data do not contain missing values, but `SEL.TS.AREA` can handle these cases by the argument `NAcontrol`, representing the minimum number of non-missing values allowed for each time series (default=4).
The algorithm works interactively. First, `SEL.TS.AREA` allows the user to investigate whether there is an expression-level dependency of the model error built from the replicates, as shown in Figure 3. In particular, data from `replicates` are divided into bins, calculating the mean expression level (x-axis) and the variance of the differences between the replicates (i.e. the difference between the two columns reported in `replicates`, y-axis). By looking at the plot provided by `SEL.TS.AREA`, the user can choose either a constant or an expression level-dependent model. If the latter is chosen, as in this tutorial, a spline fit is used to represent the model error (black line in Figure 3). In order to achieve the best description of the variability characterizing the experimental data, `SEL.TS.AREA` provides the following arguments:

- `sp`: the smoothing parameter used to fit the expression level-dependency of the experimental error;
- `quantile_sampling` and `binsize`: the former argument indicates whether using a fixed-size (default) or a quantiles-based grid of intervals of gene expression levels to model the error variance; according to `quantile_sampling`, `binsize` can indicate either the size of the intervals (default equal to 0.1) or the proportion used to define the quantiles (e.g. `binsize`=0.01 to use the percentiles), respectively.

For a detailed description of `SEL.TS.AREA` input arguments, see the corresponding help page:

```
> help(SEL.TS.AREA)
```

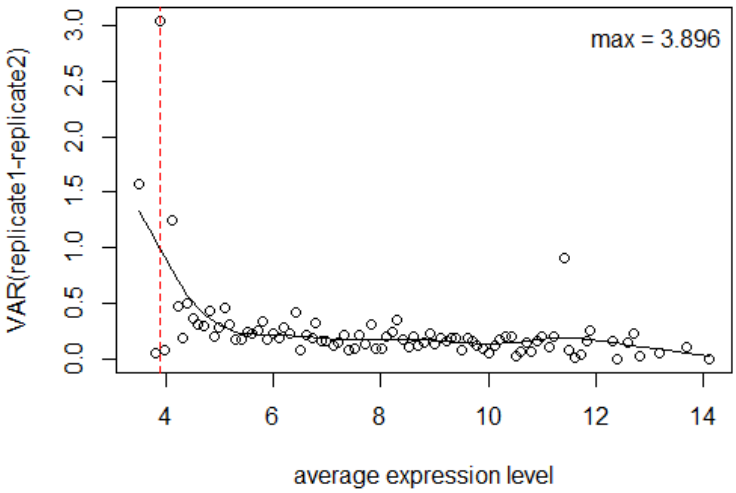


Figure 3: **Average variance of differences between replicates at different expression levels.** Dots represent the variance of the deviation between replicates at different expression levels (in log2-scale), averaged on intervals of constant size. The dashed red line indicates the maximum variance observed, and the corresponding gene expression level is reported in the top-left side of the figure.

Once the error model is built, for each gene the bounded-area between the related time series in `ctrl` and `treat` is calculated and compared to the empirical null hypothesis distribution, derived from the model described above. Different models (Gamma, Log-normal, Weibull) can be used to describe the null distribution (Figure 4). The related fit results are displayed in the command window:

```
GAMMA DISTRIBUTION MODEL - FIT RESULTS:

parameters (SHAPE, RATE): 19.85802 2.417127
Precision(SHAPE, RATE): 0.278507 0.03433118

Maximum number of iteration: 1000

Relative convergence tolerance: 1e-08

LOGNORMAL DISTRIBUTION MODEL - FIT RESULTS:

parameters (MEAN, SD): 2.080666 0.2280809
Precision(MEAN, SD): 0.002280809 0.00161285

Maximum number of iteration: 1000

Relative convergence tolerance: 1e-08

WEIBULL DISTRIBUTION MODEL - FIT RESULTS:

parameters (SHAPE, SCALE): 4.727832 8.94812
Precision(SHAPE, SCALE): 0.03455785 0.02003932

Maximum number of iteration: 1000

Relative convergence tolerance: 1e-08
```

The estimates might slightly change since a Monte Carlo procedure is used to build the null hypothesis distribution, generated from B=10000 (default) profiles derived from expression data in `replicates`, under the hypothesis that the error at different time samples is independent and identically distributed [3]. According to the available data, the user can increase the number of generated profiles by the argument `B`, but it is worth noting that this choice increases the execution time too.

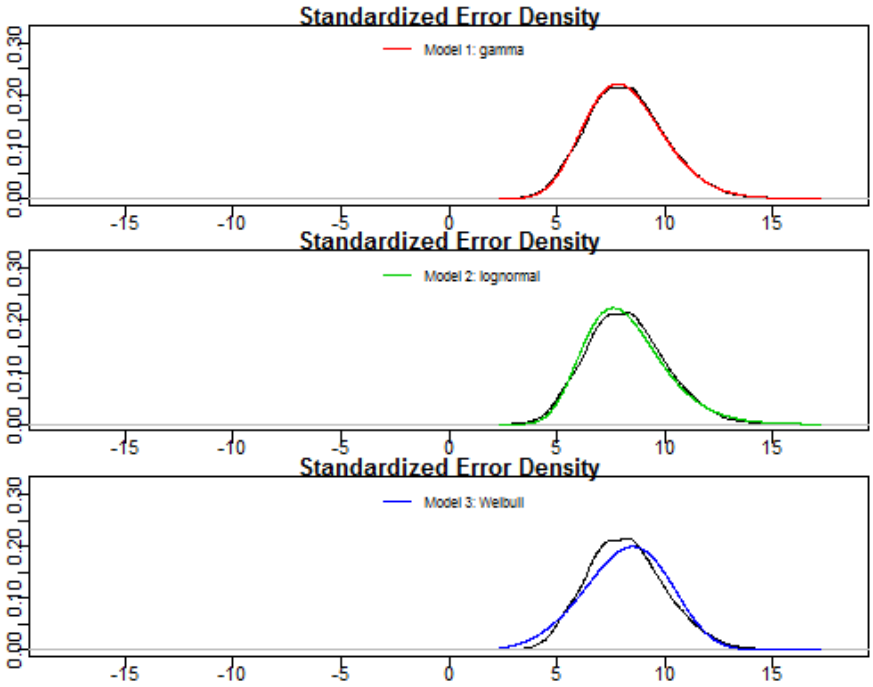


Figure 4: **Fit of the null hypothesis distribution.** Empirical null hypothesis distribution (in black) fitted with a Gamma (upper panel) a Log-normal (central panel) and a Weibull distribution (lower panel).

The user can choose the best model among the ones fitting the null hypothesis distribution. Precision of the estimates can help in the decision. Quantiles of the empirical distribution can be used as an alternative to the above models. In this example, we chose the Gamma distribution since the plot shows a good fit at the distribution tails with respect to the empirical distribution, allowing a better estimation of the p-values. After choosing the best model, `SEL.TS.AREA` estimates the proportion of not differentially expressed genes (P0) and uses this value to correct the p-values for multiple testing according to the False Discovery Rate (FDR), i.e. the number of false positives divided by the number of selected genes [4]. The user can also change the correction method using the argument `adj.method`.
`SEL.TS.AREA` returns a data frame reporting for each gene: the rank index related to decreasing areas, the gene identifier, the area standardized according to the error variance, the p-value resulting from the statistics, its adjustment according to the chosen correction method and the expected number of false positives:

```
> head(rank.res,5)
```

To save the results, the user can indicate an existing folder or automatically generate a new one using the input argument `mainDir`. The default uses the current working directory. At the indicated directory, `SEL.TS.AREA` generates a sub-folder named "Gene_Ranking" containing two text files:

- "Gene_Ranking.txt" reporting the data frame returned by the function.
- "Excluded_genes.txt" reporting, whether present, genes excluded from the analysis (e.g. genes with too many missing values).

3.4 Search for Temporal Patterns

SEL.TS.AREA provides a p-value enabling the user to set a threshold in order to identify the DE genes. In order to identify the temporal patterns associated to these genes, a linear model-based clustering is provided by FunPat [5], implemented by the function find.all.patterns. The model-based approach offers the advantage of automatically choosing the number of clusters without requiring the user to fix it *a priori*. Details about the algorithm are reported in Appendix 8.
find.all.patterns works with the differential temporal profiles, e.g. the *treatment-control* time series. The following code is an example of the application of find.all.patterns to the simulated data using the p-values resulting from SEL.TS.AREA:

```
> sel.diff.data<-nD-nC
> colnames(sel.diff.data)<-colnames(nD)
> row.names(sel.diff.data)<-row.names(nD)

> ind<-order(as.numeric(rank.res$p_value)) # sorting according to the p-value
> rank.res<-rank.res[ind,]

> ind<-which(as.numeric(rank.res$adjusted_p_value)<=0.05)
> pval<-rank.res$p_value[ind]
> selected<-as.character(rank.res[ind,2])

> ind<-match(selected,row.names(sel.diff.data))
> sel.diff.data<-sel.diff.data[ind,]

> CLUSTERS<-find.all.patterns(DATA=sel.diff.data,seeds=selected,p.sel=pval,
+                             sizecl=5,singletons=TRUE,Psign=FALSE)
```

In this example, a threshold equal to 0.05 is applied on p-values, controlling for the false discovery rate. The p-values are then exploited by the clustering algorithm to prioritize the initialization of each cluster, starting from the profile of the DE gene with the lowest p-value.
As for SEL.TS.AREA, find.all.patterns can handle data with missing values by NA.control. The user can decide to be more or less stringent in the clustering changing the input arguments alphacorr and correction. The former represents the threshold applied to the p-values resulting from a goodness of fit test (default 0.05) and the latter is the correction method used to adjust the p-values (default FDR).
According to the experimental design, the linear model used in the clustering algorithm can be also opportunely constrained by the user using the input arguments onlyK and Psign. A demonstration is provided in Appendix 8. Moreover, the user can decide to fix a minimum cluster size by the argument sizecl (default set to 3). If there are genes whose temporal profile is not highly correlated with any other genes, or clusters with a number of genes less than sizecl, or with too many missing values, find.all.patterns classifies them as *singletons*.
find.all.patterns provides as output a list where each element, representing a cluster, reports the associated temporal pattern, the related genes with the p-values from SEL.TS.AREA and the identified parameters of the linear model. The user can decide to display also the singletons at the end of the output list by setting to TRUE the input argument singletons. To obtain a more user-friendly organization of the results, we suggest using the function resGSPatterns, which is deeply described in Section 3.7.

Further details about find.all.patterns are reported in the corresponding help page:

```
> help(find.all.patterns)
```

3.5 Loading database annotations

Both functions SEL.TS.AREA and find.all.patterns can be applied when no prior knowledge on genes is available. However, FunPat is able to exploit database annotations in order to improve both selection and clustering steps through an integrative approach [2]. There are no constraints about the annotations used, which can convey different types of information, e.g. Gene Ontology (GO) annotations [6] or pathways. This prior knowledge is used by FunPat to group the genes into biologically relevant *Gene Sets*.
There are two types of information that FunPat can receive as input to codify the prior knowledge:

- Annotations of genes to the Gene Sets;
- Relationships between Gene Sets (optional), if the prior knowledge is organized according to a hierarchical structure (e.g. the directed acyclic graph defined by the GO database).

This information can be provided by the user either as lists or as data frames. As an example, we use the GO information from category *Molecular Function* and available from two Bioconductor packages: org.Hs.eg.db, which provides the list of GO annotations for Homo sapiens; GO.db, which provides the GO directed acyclic graph (DAG) as a list of relationships between the GO terms:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("org.Hs.eg.db")
> biocLite("GO.db")

> library(org.Hs.eg.db)
> GOannot<-as.list(org.Hs.egGO2ALLEGS) # list of GO annotations

> library(GO.db)
> gt<-as.list(GOTERM) # list of the names of GO terms

> ontologylab<-unlist(lapply(gt,Ontology))
> ind<-which(ontologylab=="MF") # Molecular Function GO category
> GOterm<-lapply(gt[ind],Term)
> GOanc<-as.list(GOMFANCESTOR) # list of ancestor GO terms

> ind<-which(names(GOannot)%in%names(GOterm))
> GOannot<-GOannot[ind] # GO annotations filtered by GO terms of Molecular Function category
```

GOannot represents the mapping of the Entrez IDs to the GO terms. GOterm and GOanc are lists reporting the names of the GO terms and the relationships of each GO term with the corresponding ancestor terms in the DAG, respectively. Although the information on the relationships between Gene Sets, e.g. GO terms, is not mandatory, we recommend to use it when available, since it is exploited by FunPat to prioritize the most informative terms in the analysis.
All the prior information available on GO terms and their gene annotations is then organized by FunPat using the function called TABpreprocessing:

```
> GSdata<-TABpreprocessing(TAB=GOannot,term=GOterm,TABgraph=GOanc,
+                          root="GO:0003674", GraphType="Ancestors")
```

When no information on the relationships between Gene Sets is available, TABpreprocessing can be used as well by setting only the input arguments TAB and terms. Otherwise, if this information is provided by TABgraph, the user has to specify two additional arguments:

- root, indicating the identifier of the Gene Set associated to the root node in the hierarchical graph. In this case, "GO:0003674" is the identifier of the GO term *Molecular Function*;
- GraphType, indicating the type of relationships set as input to TABgraph. In this case, since we are using GOanc, GraphType is set to "Ancestors". As alternatives, information on parent or on child terms can be provided as well, setting this argument to "Parents" or "Children", respectively.

The output of TABpreprocessing, GSdata, is a list of lists collecting all the available information to be exploited by FunPat in the analysis presented in the next section. Further details on GSdata and how using TABpreprocessing to organize custom-based annotations are explained in Section 6 and in the help page:

```
> help(TABpreprocessing)
```

3.6 Gene Set-based clustering and gene selection

As presented in [2], the integration of gene selection, clustering and functional analysis can address several issues related to the independent application of these analysis steps, as decreasing the number of false negatives which result from the multiple tests correction preserving the false discovery rate. This is accomplished by the function PatternAnalysis, which exploits the p-values provided by SEL.TS.AREA, the model-based clustering provided by find.all.patterns and the biological information contained in GSdata to search for temporal patterns of DE genes associated to common specific Gene Sets.
Starting from the p-values of SEL.TS.AREA, two lists of genes are defined: the *seeds*, i.e. the DE genes passing a user-defined threshold on p-values adjusted for multiple testing (e.g. FDR), and the *candidates*, i.e. genes passing a soft-threshold applied to unadjusted p-values. Other p-value correction approaches to define the seeds, such as Bonferroni correction, can be considered by the user as well. Then, as shown for find.all.patterns, the differential temporal profiles are considered, but in this case the expression matrix is extended also to the candidates, representing possible false negatives which can be reconsidered for the selection by PatternAnalysis:

```
> sel.diff.data<-nD-nC
> colnames(sel.diff.data)<-colnames(nD)
> row.names(sel.diff.data)<-row.names(nD)

> ind<-order(as.numeric(rank.res$p_value)) # sorting according to the p-value
> rank.res<-rank.res[ind,]

> ind<-which(as.numeric(rank.res$adjusted_p_value)<=0.05)
> selected<-as.character(rank.res[ind,2]) # seed genes: FDR<=0.05

> ind<-which(as.numeric(rank.res$p_value)<=0.05)
> pval<-rank.res$p_value[ind]
> candidates<- as.character(rank.res[ind,2]) # candidate genes: p_value<=0.05

> ind<-match(candidates,row.names(sel.diff.data))
> sel.diff.data<-sel.diff.data[ind,]

> RES<-PatternAnalysis(DATA=sel.diff.data,p.sel=pval,seeds=selected,
+                      GSdata=GSdata,sizecl=5)
```

Differently from find.all.patterns, PatternAnalysis requires as input the biological information organized by TABpreprocessing, named GSdata, which allows the search of the Gene Set-specific temporal patterns. When no information on relationships between the Gene Sets is available, each Gene Set is analyzed independently. Otherwise, PatternAnalysis exploits the relationships between the Gene Sets in order to reduce the redundancy of information which generally characterizes hierarchically structured databases such as GO. In particular, PatternAnalysis assumes that the genes annotated to a Gene Set are also annotated to all its ancestors and that the farther the Gene Set is from the root node in the graph, the more specific information it conveys. Thus, to guarantee the association of the selected genes and temporal patterns to the most specific biological information, the function:

- classifies the Gene Sets according to levels based on the maximum path from the root node;
- searches for temporal patterns starting from the highest level nodes, i.e. the most informative terms, removing the annotations of the genes belonging to a significant pattern from the ancestor nodes [7].

Another argument which is important to consider in `PatternAnalysis` is `sizecl`. Indeed, there is a trade-off between the biological information provided by the selected Gene Sets and the number of Gene Set-specific patterns. The Gene Sets associated to the most specific biological terms are usually poorly annotated, thus setting a large minimum cluster size can lead to associate the significant patterns to less informative biological functions; on the other hand, a small minimum cluster size might increase the number of selected Gene Sets, spreading the associations between selected genes and Gene Sets.
As output, `PatternAnalysis` provides a list with two elements. The first element is a list named *RES.PAT*, collecting the results of the function `find.all.patterns` for those Gene Sets characterized by at least one significant pattern. The genes selected by `PatternAnalysis` are either seeds or candidates which share both a significant temporal pattern and a common annotation to a Gene Set with at least one seed[2]. If there are seeds excluded by the analysis, they are reported in the second element of the output list, named *Singletons*. This element can be useful to understand if missing annotations are present in the prior knowledge used for the analysis. Indeed, a seed can be excluded if either it is not associated to any significant temporal pattern or it does not belong to any Gene Sets. For an exhaustive explanation of all the attributes of the output list, see the corresponding help page:

```
> help(PatternAnalysis)
```

A more user-friendly organization of the results through tables is presented in the next section.

3.7 Displaying temporal patterns representing clusters of genes

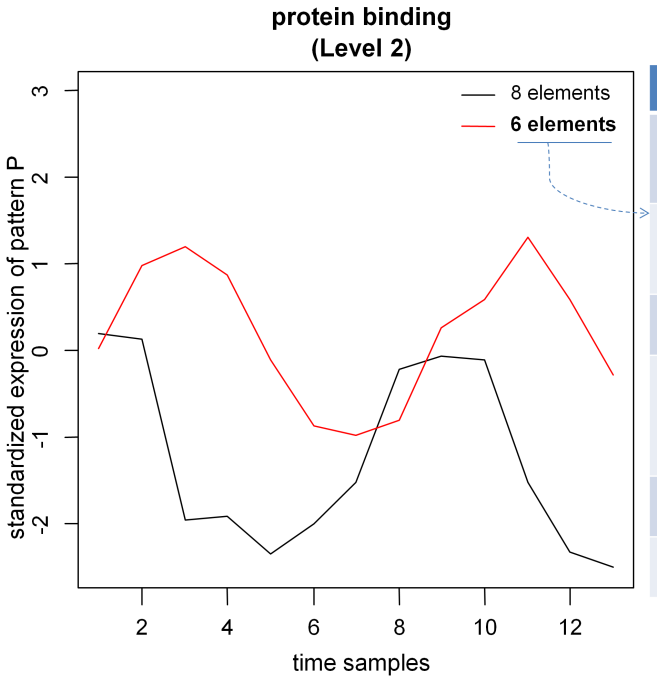
As first result, the pattern analysis provides clusters of DE genes associated to Gene Set-specific temporal patterns. They can be easily displayed using the function `resGSPatterns`, which receives as input the lists provided by `PatternAnalysis`:

```
> mat.res<-resGSPatterns(RES,sampling.grid=0:12)
> head(mat.res,5)
```

The input argument `sampling.grid` indicates the time grid at which data are sampled (here 0-12 hours), which here is used only to visualize the data. As output, `resGSPatterns` generates a data frame, whose columns report: the information of the selected Gene Sets (columns `GeneSet_ID`, `GeneSet_Term` and `Level`); the gene IDs of the corresponding selected gene (`Element_ID`); the significance of each gene given as input to `PatternAnalysis` (in this case the p-value resulting from `SEL.TS.AREA`, column `Score`); the gene-specific parameters `K` and `Q` estimated by the model-based clustering (see Appendix 8), and the identified patterns (columns `t0-t12` for this example). When present, the singletons are reported at the end of the data frame naming the Gene Set identifiers and terms as "Singleton". Not available information, such as the Gene Set level, is set to NA.
`resGSPatterns` also generates a sub-folder named "Pattern_Analysis_Results" in the working directory (which can be controlled by the user through the argument `mainDir`) containing three files:

- "GSPatterns.pdf", reporting, for each selected Gene Set, the plot of the associated temporal patterns (see Figure 5 as an example).
- "GSPatterns.txt", reporting the data frame returned by the function, without singletons.
- "Singletons.txt", reporting all the information provided by `PatternAnalysis` about the seed genes excluded from the analysis.

[2]Intuitively, if a gene characterized by a significant nominal p-value is excluded by the multiple tests correction, but it shares the same temporal expression pattern and the same functional annotation with genes selected as differentially expressed, the gene is likely to be a false negative. As a consequence, recovering it in the pool of DE genes might increase the recall without negatively affecting the precision.



Gene Symbol	Gene Name
CACNA1H	calcium channel, voltage-dependent, T type, alpha 1H subunit
CACNG2	calcium channel, voltage-dependent, gamma subunit 2
TPCN2	two pore segment channel 2
TRPM6	transient receptor potential cation channel, subfamily M, member 6
IL1RAPL1	interleukin 1 receptor accessory protein-like 1
NCS1	neuronal calcium sensor 1

Figure 5: **Example of Gene Set-specific patterns.** Each color-coded pattern represents a cluster of DE genes selected in the same Gene Set. Note that a Gene Set can have more than one pattern.

The function `resGSPatterns` can also visualize the results from the function `find.all.patterns`. As an example, try to type the following command:

```
> mat.res<-resGSPatterns(CLUSTERS,sampling.grid=0:12)
> head(mat.res,5)
```

Also in this case, the sub-folder "Pattern_Analysis_Results" is created, generating the plots of each identified pattern (file "Patterns.pdf") and the information about the clusters and the singletons (files "Patterns.txt" and "Singletons.txt", respectively), which mat.res reports into a unique data frame.

3.8 Displaying main temporal patterns representing clusters of Gene Sets

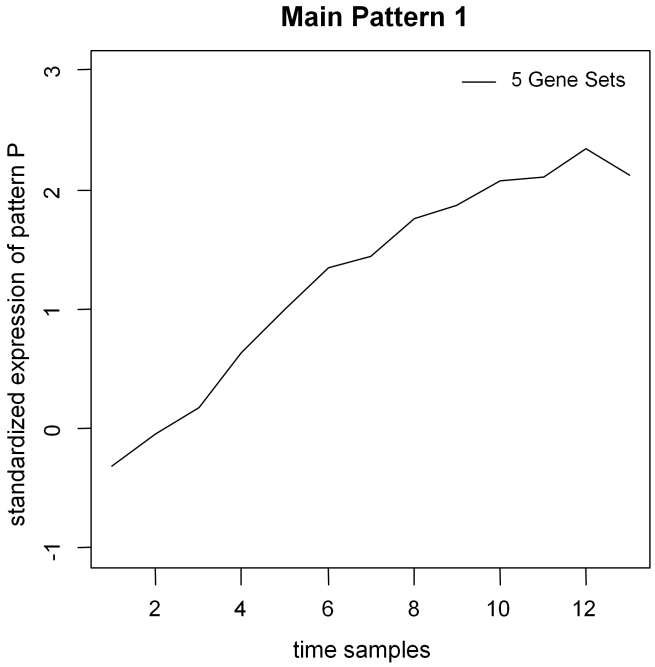
`resGSPatterns` provides a description of the identified Gene Set-specific patterns. However, similar patterns can be identified for different Gene Sets [3] and there could be singletons without annotations but characterized by a temporal profile that is highly correlated to one of the identified Gene Set-specific patterns.
As a further result of the pattern analysis, `FunPat` allows the user to summarize the Gene Set-specific patterns into *Main Patterns* characterizing clusters of Gene Sets. This is accomplished by the function `resMainPatterns`, which applies the model-based clustering to the identified temporal patterns. If present, singletons can be included into the analysis (using the argument `keep_singletons`, default = `TRUE`), in order to test if there are seeds excluded by lack of annotation showing a similar temporal profile to some significant Gene Set-specific pattern:

```
> mat.res.super<-resMainPatterns(mat.res,sampling.grid=0:12,sizecl=1,num_elem=5)
> head(mat.res.super,5)
```

[3] Note that the selected genes displayed in the data frame resulting from `resGSPatterns` can belong to multiple clusters, since the clustering is Gene Set-specific and, even if a hierarchical prior information is available, the selected genes are removed from the ancestor terms but not from possible siblings, which can convey different specific biological information.

As in `find.all.patterns` and in `PatternsAnalysis`, the user can decide to fix a minimum cluster size by the argument `sizecl` (default=3). However, note that in `resMainPatterns` the cluster size refers to the minimum number of patterns (or singleton profiles) belonging to the same Main Pattern. Since the temporal patterns represent groups of genes, the user can also set the minimum number of genes associated to each Main Pattern using the argument `num_elem` (default=3). In the example above, a Main Pattern can represent either multiple patterns or a single pattern identified by `PatternAnalysis` characterizing at least 5 genes or, since `keep_singletons`=TRUE, a cluster with at least 5 singletons. `resMainPatterns` generates a data frame that organizes the information obtained by `resGSPatterns` according to the resulting clusters of Gene Sets (first column of `mat.res.super`), represented by the Main Patterns reported at the last columns of the data frame. Gene Sets or singletons whose corresponding temporal pattern/profile does not generate a new cluster according to the arguments specified in `resMainPatterns` are included at the end of the data frame. In the following, we will refer to them as *Single Patterns*. All the results from `resMainPatterns` are automatically saved into three files placed in a sub-folder of the main directory generated by the function and named "Pattern_Analysis_Results":

- "MainPatterns.pdf", reporting the plots of the identified Main Patterns (see Figure 6 as an example).
- "MainPatterns.txt", reporting the data frame returned by the function, without Single Patterns.
- "SinglePatterns.txt", reporting the Single Patterns.



Gene Set ID	Gene Set Name
GO:0008137	NADH dehydrogenase (ubiquinone) activity
GO:0000166	nucleotide binding
GO:0016651	oxidoreductase activity, acting on NAD(P)H
GO:0019899	enzyme binding
GO:0043167	ion binding

Figure 6: **Example of Main Pattern.** The Main Pattern represents a cluster of Gene Sets characterized by similar Gene Set-specific patterns identified in the previous analyses.

`resMainPatterns` can also allow the user to merge multiple results obtained by `PatternAnalysis`. Suppose to have `mat.res1` and `mat.res2` representing the results from `resGSPatterns` of the pattern analysis performed on two different types of annotations independently. Before merging the two tables, the user has only to change the identifiers of the temporal patterns reported in the column "Cluster" in order to be unique:

```
> M<-max(as.numeric(mat.res1$Cluster))
> mat.res2$Cluster<-as.numeric(mat.res2$Cluster)+M
> mat.res<-rbind(mat.res1,mat.res2)
> mat.res.super<-resMainPatterns(mat.res,sampling.grid=0:12)
```

4 Advanced visualization of results

FunPat also allows the user to easily visualize the results into HTML tables with sortable and filterable columns, plots and hyperlinks to other data sources such as NCBI and Gene Ontology databases. To generate HTML reports, Bioconductor packages ReportingTools, hwriter and Rgraphviz are required:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("ReportingTools")
> biocLite("hwriter")
> biocLite("Rgraphviz")
```

```
> library(ReportingTools)
> library(hwriter)
> library(Rgraphviz)
```

In the following, we review the additional input arguments which FunPat provides in the main functions to generate HTML reports. For a correct visualization of the web pages, we recommend using web browsers Google Chrome and Safari.
The function SEL.TS.AREA generates a report page named "Gene_Ranking_report.html" using the following input arguments:

```
> rank.res<-SEL.TS.AREA(replicates=replic,data1=nC,data2=nD,
+                       htmlreport=TRUE,link_to_Entrez=NA,kplot=100)
```

htmlreport is a logical value enabling the function to generate the HTML report page in the sub-folder "Gene_Ranking". link_to_Entrez is an optional argument which links the Entrez IDs to the corresponding web pages of the NCBI database (http://www.ncbi.nlm.nih.gov/gene). If link_to_Entrez is set to NA, it means that the gene IDs used in the analysis are already Entrez IDs. Otherwise, the Entrez IDs can be provided with an additional data frame, loaded by the argument gene_info (see Section 6), linking the Entrez IDs to the gene IDs used in the analysis. In this latter case, to link the Entrez IDs to the NCBI database in the HTML report, link_to_Entrez=N must be specified, where N is an integer value indicating the column index in the additional data frame corresponding to the Entrez IDs. kplot is an integer value k, enabling the visualization in the HTML report of the temporal profiles of the top k genes, sorted according to the resulting p-values. Figure 7 shows an example of the final HTML report page. Since the bounded-area method calculates the area according to the argument sampling.grid, which can be set differently from the original time grid to avoid weighting too much some intervals of the time series, to plot the time series with the original time grid, the user can report it using the argument sampling.grid_view, which it is used only for the visualization in the HTML report.
The function resGSPatterns generates the report on the Gene Set-specific clusters of DE genes, visualizing the corresponding temporal patterns:

```
> mat.res<-resGSPatterns(RES,sampling.grid=0:12,htmlreport=TRUE,
+                        link_to_Entrez=NA,link_to_GO=TRUE)
```

In this case, two HTML report pages named "GSPatterns_report.html" and "Singletons_report.html" are generated in the sub-folder "Pattern_Analysis_Results", displaying the results collected in the data frame returned by the function but reporting the singletons, if present, in a separated HTML page. Figure 8 shows an example of the resulting "GSPatterns_report.html". Similar reports are created also for the results obtained by find.all.patterns. If the Gene Sets represent Gene Ontology terms, the function also allows linking each GO identifier to the corresponding AmiGO web page (http://amigo.geneontology.org/cgi-bin/amigo/go.cgi) using the optional argument link_to_GO.
Similarly, a report of the Main Patterns is provided by the function resMainPatterns:

```
> mat.res.super<-resMainPatterns(mat.res,sampling.grid=0:12,sizecl=1,num_elem=5,
+                                GraphPar=GSdata$GraphPar,htmlreport=TRUE,link_to_GO=TRUE)
```

Also in this case, two HTML report pages named "MainPatterns_report.html" and "SinglePatterns_report.html" are generated in "Pattern_Analysis_Results", displaying the main results collected in the data frame returned by the function, reporting, if present, the *Single Patterns* in a separated HTML page. Figure 9 shows an example of the resulting "MainPatterns_report.html".
If the relationships between Gene Sets are provided, the user can visualize how the Gene Sets belonging to the same Main Pattern are distributed in the hierarchical graph using the input argument `GraphPar`. This argument receives as input the list of parent nodes automatically provided as output by `TABpreprocessing`. The resulting plots display, for each Main Pattern, the minimum sub-graph including the Gene Sets belonging to the related cluster (Figure 9).

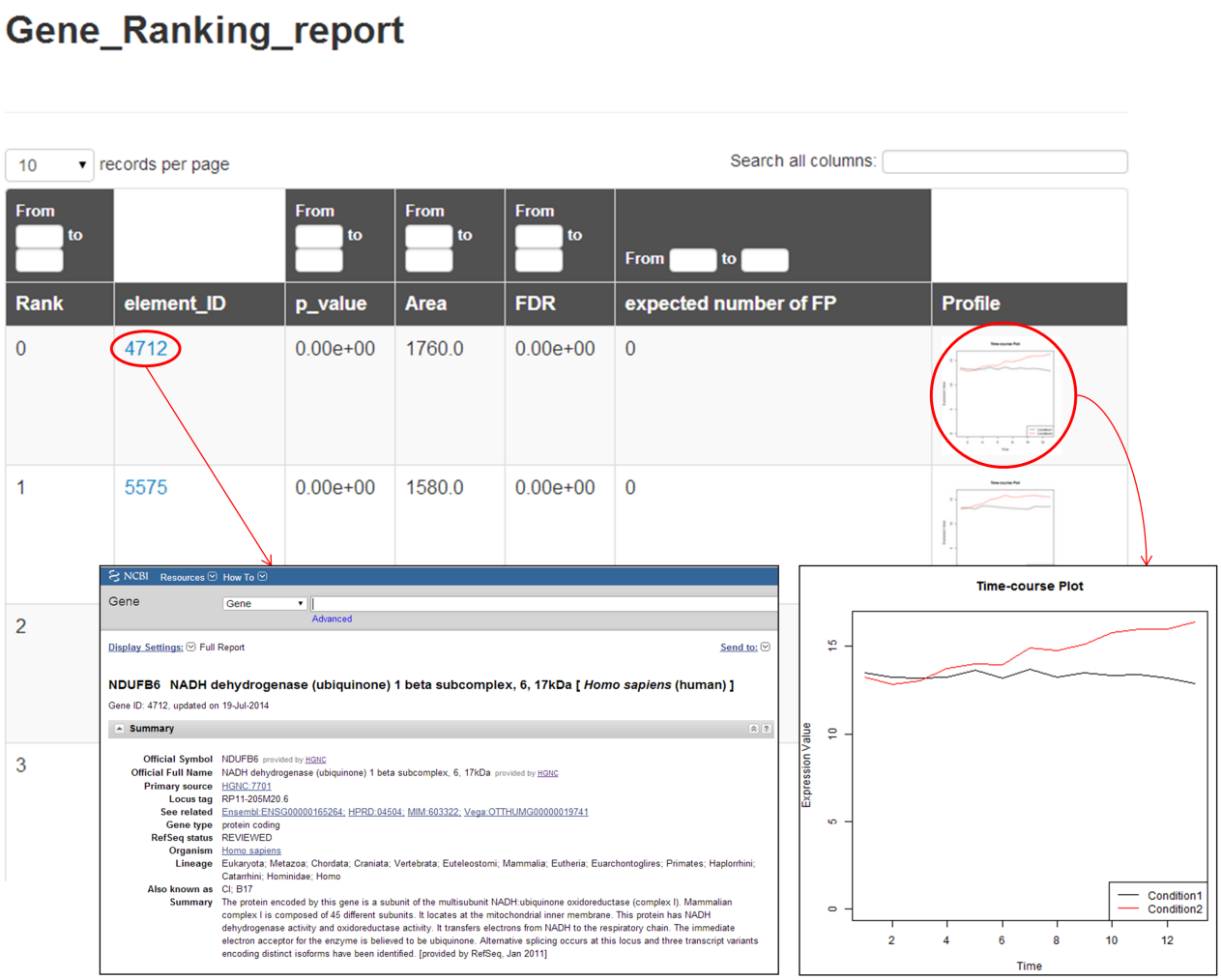


Figure 7: **HTML page generated by** `SEL.TS.AREA`. For each element, the Entrez identifier is linked to the corresponding NCBI web page and the plot of the related time series profiles in the two conditions is displayed.

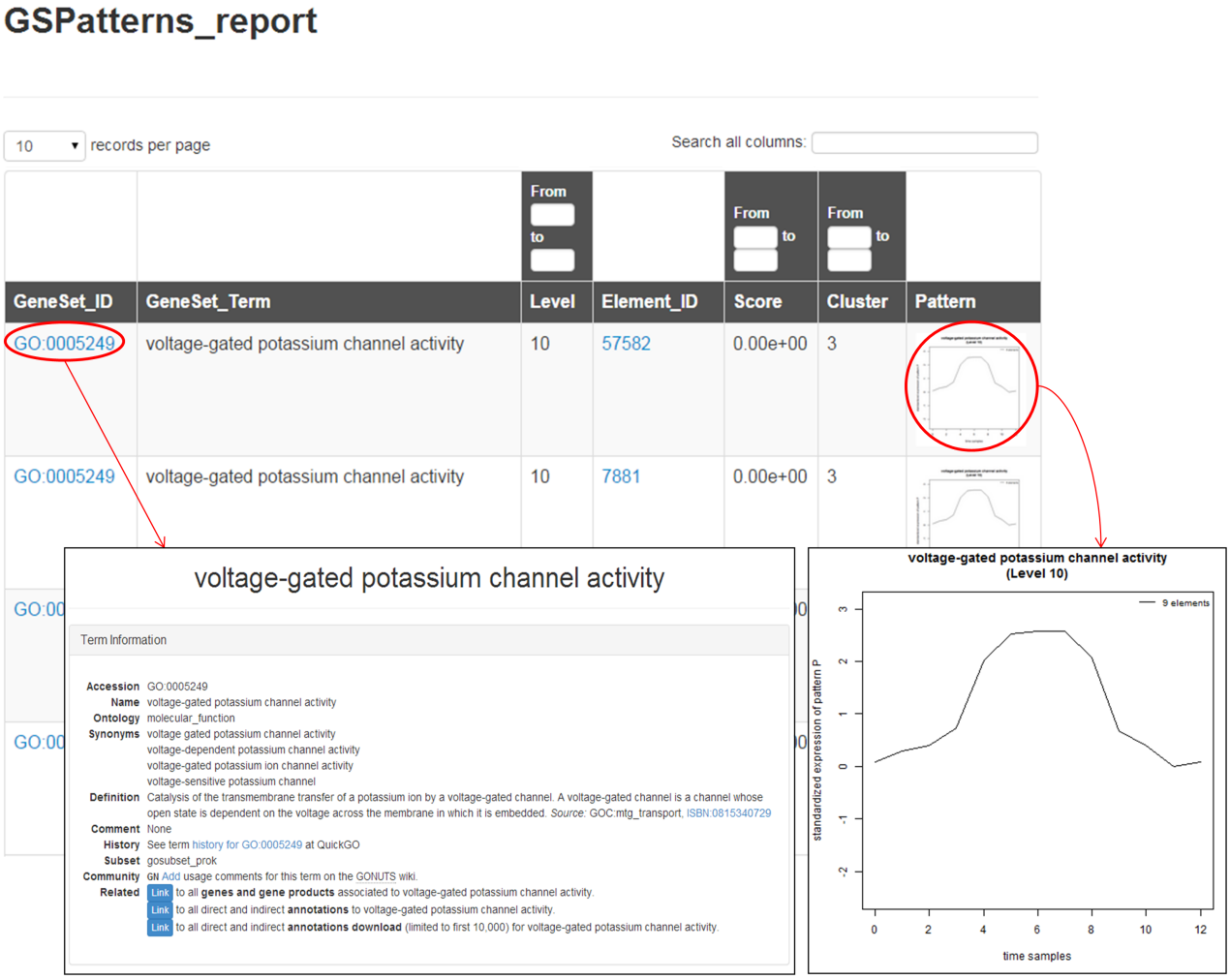


Figure 8: **HTML page generated by `resGSPatterns`.** For each element, the GO term identifier is linked to the corresponding AmiGO web page and the plot of the related temporal pattern is displayed.

Figure 9: **HTML page generated by** `resMainPatterns`. For each element, the plots of the main pattern and the sub-graph of the related Gene Sets are displayed. In the graph plot, red nodes indicate the selected Gene Sets sharing the same main pattern, whereas the yellow nodes are the available nodes in the graph used to connect all the selected Gene Sets belonging to the cluster.

5 Organizing data according to the experimental design

Before giving some useful guidelines about the organization of the expression data, it is important to remember that the data must be normalized before using `FunPat` in order to remove systematic biases which can compromise the correct estimation of the error variability.

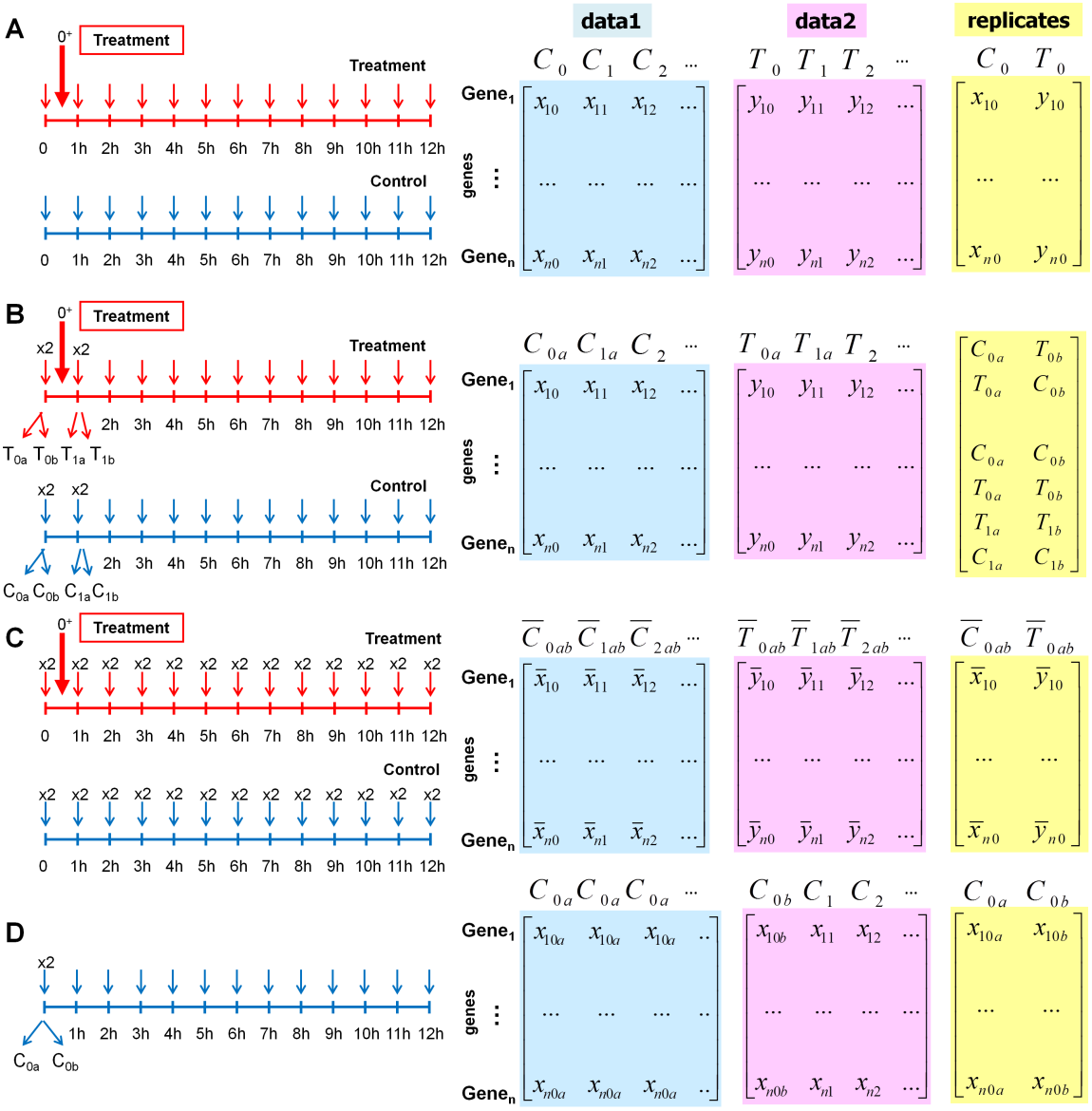


Figure 10: `FunPat` **input data organization according to different experimental designs.** Starting from the original normalized expression data matrix, three different matrices are required: `data1`, `data2` and `replicates`. Four different examples of experimental designs are displayed. For examples A, B and C two conditions are monitored (e.g. samples C and T) across different time points (e.g. C_1 refers to the samples in the first condition at time point 1) and biological replicates (e.g. a and b). Example D shows the data organization when only one condition is monitored.

Once data are normalized, FunPat requires as input both the expression matrices associated to two experimental conditions, i.e. data1 and data2, and a matrix with the replicates to be compared to build the error model, i.e. replicates. In particular, these input matrices are used by the function SEL.TS.AREA, which assesses the differential expression from the null hypothesis distribution of the area bounded by the time series in the two conditions, using the available biological replicates. It is worth highlighting that we consider as "biological replicates" only those replicates which represent both technical and biological variability of the measurements. These input matrices can be prepared in multiple ways, allowing the user to organize the data according to the specific experimental design. Figure 10 provides a general overview of data preparation, addressing some typical examples. Let's call x_{ij}^T and x_{ij}^C the expression measurements in treated (T) and control (C) cultures, available for a generic gene i at time point t_j. The replicates can be also available for a subset of time samples, since the function requires at least two replicates from only one time point.
The example A shows the experimental design used for the simulated data. Expression data are splitted into data1 and data2, reporting, for each time point, the expression values x_{ij}^C and x_{ij}^T corresponding to the control and treated cultures respectively (in the simulated data, i=1,...,n, with n=1000, and j=0,...,12h). Since the area between the time series is considered, data1 and data2 are interchangeable, thus the expression matrices of the treated and control cultures can be assigned also to data1 and data2, respectively, as well.
The available replicates used to model the biological variability are organized into the matrix replicates which represents, at each row, pairs of replicates for a gene at a specific time point. In this example, since the treatment is applied immediately after the time point 0, the samples T_0 and C_0 can be considered as biological replicates and can be set as columns of replicates, as shown in Figure 10.
When there are other replicates available for some time points, as in the example B, data1 and data2 should be prepared as before, chosing one of the replicates available. replicates should be organized in order to have the maximum number of possible pair-wise comparisons between the replicates for achieving the best description of the error variance. In this example, the best representation of the biological variability are the pair-wise comparisons T_{0a} vs. C_{0a} and T_{0b} vs. C_{0b}. Since there are more than one pair-wise comparison, these must be aligned in each column of the replicates matrix. According to the characteristics of the experiment and how the cellular replicates a and b are prepared (e.g. independent replicates), comparisons C_{0a} vs. C_{0b}, T_{0a} vs. T_{0b}, C_{1a} vs. C_{1b}, T_{1a} vs. T_{1b} can be included into replicates as well.
Example C shows an experimental design where there are two replicates for all the time points. In this case we suggest using the mean expression in data1, data2 and replicates since the average provides more robust expression values to test, thus a better estimation of the p-values for the gene ranking. As seen for example A, for replicates the mean expression in T_0 and C_0 can be used. It is worth noting that, since replicates is used to describe the variability of the differences observed between data1 and data2, whose expression values are averaged across the replicates, it is important that the error variance estimated from replicates reporting the averaged expression values of the replicates, otherwise the estimated variance tend to be over-estimated. The over-estimation of the variance does not affect the gene ranking, but leads to higher p-values thus a lower number of selected seeds. Whether it is not possible to organize replicates with the mean expression values, we still suggest considering, if possible, the mean expression in data1 and data2 and adopting strategies that can lead to a less stringent gene selection. Anyway, FunPat then allows exploiting the clustering analysis and the functional annotations to recover significant genes lost in the differential analysis performed by SEL.TS.AREA. When averaging is possible, remember that data1 and data2 must be averaged in each time point of the time series by the **same** number of replicates. If, as it often happens in data-poor conditions, replicates are available only for a subset of time points, data1 and data2 are defined using **only** the replicates describing the entire time series, whereas the remaining ones can be included into replicates considering all the possible pairwise combinations as seen in example B.
Finally, example D represents an experimental design monitoring only a single condition C. In this case, it is possible to use a baseline. In the example there are two replicates available at C_0, which can be used in replicates. If the expression at C_0 is chosen as baseline, as displayed in Figure 10, the user can define data2 reporting for each gene i the expression values x_{ij}^C of the entire time series (j=0,...,12), whereas data1 can be defined repeating across the time series the expression values of the replicate at C_0 not used in data2. When more replicates are available, the comments made for the previous examples still apply to this case.

6 Dealing with custom Prior Knowledge

If the available annotations are not organized according to lists as shown in Section 3.5 or the user has to define custom annotations, the function `TABpreprocessing` is able to handle also prior information organized according to data frames. As an example, `FunPat` provides the GO information organized into data frames. To load it, just type:

```
> data(FunInfo)
```

`FunInfo` contains:

- `GOannot`: a two-column data frame reporting the mapping between the GO term identifiers (first column) and the annotated genes (Entrez IDs reported in the second column), using a delimiter character to separate multiple entries.
- `GOchild`; `GOpar`; `GOanc`: two-column data frames reporting in the first column the GO term identifiers and in the second column the associated child (`GOchild`), parent (`GOpar`) or ancestor (`GOanc`) term identifiers, using a delimiter character to separate multiple entries.
- `GOterm`: a two-column data frame reporting in the first column the GO term identifiers and in the second column the associated name or term description.
- `geneinfo`: additional information related to the genes (e.g. Gene Name, Gene Symbols); it is an $N \times P$ data frame, where N indicates the number of genes and P the number of additional categories included. The first column reports the gene identifiers used for the analysis.

These simple data frames can be generated from many other types of annotation retrieved from other databases (e.g. Biomart, KEGG). `TABpreprocessing` directly processes them to organize the input data required by `PatternAnalysis`:

```
> GSdata<-TABpreprocessing(TAB=FunInfo$GOannot,delimiterTAB=" /// ",TABgraph=FunInfo$GOanc,
+                          delimiterTABgraph=" /// ",root="GO:0003674",term=FunInfo$GOterm,
+                          GraphType="Ancestors",extend=FALSE)
```

`delimiterTAB` and `delimiterTABgraph` indicate the delimiter characters used to separate multiple entries. `FunInfo` provides all the possible types of information to describe the GO graph (`GOchild`, `GOpar` and `GOanc`); however, as stated before, only one type is necessary. You can try to apply TABpreprocessing using also `GOpar` and `GOchild` as input for the argument `TABgraph`, and setting the argument `GraphType` as "Parents" and "Children", respectively. Another input argument is `extend`, which is important to consider when the relationships between Gene Sets are included. This is a logical value indicating if the annotations should be extended across the ancestor terms, so that the genes annotated to a Gene Set are also annotated to all its ancestor terms in the graph. In the example above, the GO annotations are already extended, thus the default value (`FALSE`) is used. However, for Gene Ontology terms, several databases report the gene annotations only for the most specific nodes. For a correct usage of `PatternAnalysis`, we suggest to set `extend` to `TRUE` when this happens. As output, `TABpreprocessing` provides a list of five attributes:

- Annot: annotation list with the mapping between genes and Gene Sets that they are associated with.
- GraphAnc: ancestor list describing associations between Gene Sets and their ancestors in the graph.
- GraphPar: parent list, describing associations between Gene Sets and their parents in the graph.
- Graph.levels: a vector describing the maximum path from the root node to each node representing a Gene Set. Levels are calculated to prioritize the node visiting order according to the graph structure.
- Terms: name list, associating the full name to each Gene Set identifier.

If no information is provided on the graph or the Gene Sets, the related attributes are empty lists.

Finally, FunInfo provides the data frame geneinfo, representing other additional information on genes which the user can find useful to display at the end of the analysis. geneinfo can be used directly as input argument in the functions SEL.TS.AREA and resGSPatterns. The columns of this data frame will be included in the corresponding final tables. For a correct usage, it is important to report in the first column the gene identifiers used to perform the analysis. As an example, let's try to include this type of information in SEL.TS.AREA:

```
> rank.res<-SEL.TS.AREA(replicates=replic,data1=nC,data2=nD,gene_info=FunInfo$geneinfo)
> head(rank.res,5)
```

If htmlreport is TRUE, the information provided by geneinfo are displayed also in the resulting HTML reports.

7 Setup

This analysis was conducted on:

```
> sessionInfo()

R version 3.0.3 (2014-03-06)
Platform: x86_64-w64-mingw32/x64 (64-bit)

locale:
[1] LC_COLLATE=Italian_Italy.1252  LC_CTYPE=Italian_Italy.1252
[3] LC_MONETARY=Italian_Italy.1252 LC_NUMERIC=C
[5] LC_TIME=Italian_Italy.1252

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

loaded via a namespace (and not attached):
[1] tools_3.0.3
```

8 Appendix: Model-based clustering algorithm

The search for patterns is performed using a linear model-based clustering [5], implemented by the function `find.all.patterns`. This function requires as input a set of gene expression profiles, the error matrix E, a set of seed genes and, if available, the scores resulting from the bounded-area method. The method searches for subsets of genes, whose time series expression profile $x_i =< x_i(1), ..., x_i(M) >$ can be modeled according to the equation (1), by iteratively performing a gene-specific parameter identification step (M step) and a temporal pattern search (E step), using an Expectation-Maximization approach. A pseudocode of the algorithm is illustrated in Figure 11. At the first iteration, the temporal pattern P is initialized with the time series expression profile of a seed gene given as input. If a ranking score is available, the seed with the most significant score (e.g. the lowest p-value resulting from `SEL.TS.AREA`) is considered to initialize P. Given the gene expression data X and the pattern P, in the M step the parameters k and q are identified for each gene using weighted least squares method, defining the cluster C of genes fitting the pattern P. The membership of a gene in a cluster C is based on: 1) a goodness of fit test to P by applying a runs test and a chi-square test to the residuals, 2) a statistical assessment in comparison to a flat profile. The user can fix a significance level α for the tests and only genes with significant p-values are kept in the cluster. In the E step, P is identified at each sampling time, applying again the weighted least squares, using the parameters k and q of the genes belonging to C and estimated at the M step. All the analyzed genes go again through the M step, so to update the estimation of the parameters k and q and re-define the cluster membership based on the newly estimated pattern P. The algorithm re-iterates all the steps until the genes belonging to C do not change or a maximum number of iterations is reached. Finally, the mean pattern P_m representing the average time series expression profile across all genes joining the cluster C is defined using the parameters k_m and q_m estimated at the last iteration:

$$P_m = k_m \cdot P + q_m \tag{1}$$

with

$$k_m = \frac{k}{\sum_{i=1}^{r} k_i}, q_m = -\frac{\sum_{i=1}^{r} q_i}{\sum_{i=1}^{r} k_i} \cdot k + q \tag{2}$$

where $\sum_{i=1}^{r} k_i$ and $\sum_{i=1}^{r} q_i$ indicating the mean k and q across the r genes belonging to C. After identifying a pattern and a cluster of genes, these are removed and the entire procedure is repeated again on the remaining genes, until the number of genes is no sufficient to create a new cluster or no other significant patterns can be discovered. At the end, the algorithm returns the sets of genes belonging to each cluster, the set of identified patterns and the estimated model parameters. If available, the related ranking scores used for the initialization of the patterns are also returned.

INPUT: Set of N gene profiles $X=<x_1,\ldots,x_N>$, Error Matrix E, Set of selected genes SEED $\subseteq$ X, Vector of Scores $S=<s_1,\ldots,s_N>$

OUTPUT: a list of clusters "Sharing Annotation and expression Pattern" (SAP)

- ?GSP$\leftarrow\emptyset$; ¬ GSP$\leftarrow$ X
- $GS\_P_m \leftarrow GS\_k_m \leftarrow GS\_q_m \leftarrow \emptyset$
- w$\leftarrow$1

WHILE $((size(\neg GSP) \geq \varphi) \wedge ((\neg GSP \cap SEED) \neq \emptyset))$

- j$\leftarrow$which.min(score(¬GSP $\cap$ SEED)); P $\leftarrow x_j$
- C$\leftarrow$C.old$\leftarrow\{x_j\}$
- STOP$\leftarrow$FALSE

WHILE (STOP=FALSE)

{ **FOR** i **FROM** 1 **TO** N
- $<k_i, q_i, WRSS_i> \leftarrow WLS(X_i, P, diag(\Sigma)=E(i,\times))$
- $p\text{-}val_i \leftarrow$goodness of fit $(WRSS_i)$
- **IF** $(p\text{-}val_i<\alpha)$
  THEN $C \leftarrow C \cup \{X_i\}$

M step

IF size(C)>1
THEN
FOR t **FROM** 1 **TO** M
- $P(t) \leftarrow WLS(X(t), k, q, diag(\Sigma)=E(i \in C,t))$
- $X=\{X_i\}$; $k=\{k_i\}$; $q=\{q_i\}$ $\forall$ $X_i \in C$

E step

IF (C=C.old) OR (max.it reached)
THEN
STOP $\leftarrow$TRUE
$k_m \leftarrow k/\Sigma(k)$
$q_m \leftarrow -(\Sigma(q)/\Sigma(k))*k+q$
$P_m \leftarrow k_m*P+ q_m$
ELSE C.old $\leftarrow$C
}

- ?GSP(w)$\leftarrow$C; ¬ GSP$\leftarrow$¬ GSP – C
- $GS\_P_m(w) \leftarrow P_m$; $GS\_k_m(w) \leftarrow k_m$; $GS\_q_m(w) \leftarrow q_m$
- w$\leftarrow$w+1

- **Return** SAP$\leftarrow$<?GSP, GS_P_m, GS_k_m, GS_q_m, score(?GSP)>

Figure 11: **Pseudocode description of the model-based clustering approach.** The algorithm iteratively searches the main characteristic patterns starting from a set of N gene profiles and a set of seed genes provided as input. The function indentifies a list of clusters ?GSP, each characterized by a pattern P_m by updating the set of genes which do not fit any pattern (¬GSP). If the size of ¬GSP is greater than a minimum cluster size φ fixed by the user AND there is at least a seed gene available to initialize a pattern P, the algorithm searches for new patterns. The M step identifies the fitting parameters k and q and, if there are at least two genes with a p-value lower then a fixed significance threshold α, the E step is performed updating the pattern P. The two steps are re-iterated until the set of genes does not change or a maximum number of iterations ($max.it$) is reached, defining the cluster C, the mean pattern P_m and the parameters k_m and q_m. ¬GSP is updated and the entire procedure is then applied again to ¬GSP, until it does not contain a sufficient number of genes or no other significant patterns can be discovered. The algorithm returns the set of genes belonging to each cluster ?GSP, the set of identified patterns GS_P_m and the estimated model parameters GS_k_m and GS_q_m. If available, the related ranking scores used for the initialization of the patterns are also returned.

According to the experimental design and the available data, it is possible to constrain the clustering algorithm using the input arguments `onlyK` and `Psign`. The former limits the linear model to the identification of only the gene-specific parameter k, whereas the latter constrains the clustering algorithm to group only temporal profiles characterized by differential expression values with the same sign in all (or almost all) the time points. To better explain how these two arguments affect the clustering results, let's generate a simple expression matrix with 5 time series:

```
> t<-0:10
> y<-(2*sin(t/6))-3
> sel.diff.data<-matrix(,5,length(t))
> sel.diff.data[1,]<-y+8
> sel.diff.data[2,]<-y+4
> sel.diff.data[3,]<-y+2.9
> sel.diff.data[4,]<-y
> sel.diff.data[5,]<-y*3
> row.names(sel.diff.data)<-seeds<-c("a","b","c","d","e")
```

`sel.diff.data` represents different linear transformations of the variable `y`. In particular, in rows "a", "b" and "c" different constants are added to `y` so that the corresponding time series can either have all the time points with the same sign ("a" and "b") or one time point with different sign ("c"). Row "e" reports instead an amplification of the time series represented by `y` (reported in row "d"). Figure 12 shows the results obtained by the clustering algorithm using four diffent combinations of `onlyK` and `Psign`::

```
> pval<-rep(0.01,5) # p.values
> clust<-find.all.patterns(sel.diff.data,seeds=seeds,p.sel=pval,
+                          singletons=TRUE,sizecl=2) # A
> clust<-find.all.patterns(sel.diff.data,seeds=seeds,p.sel=pval,
+                          onlyK=TRUE,singletons=TRUE,sizecl=2) # B
> clust<-find.all.patterns(sel.diff.data,seeds=seeds,p.sel=pval,
+                          Psign=0,singletons=TRUE,sizecl=2) # C
> clust<-find.all.patterns(sel.diff.data,seeds=seeds,p.sel=pval,
+                          Psign=1,singletons=TRUE,sizecl=2) # D
> mat.res<-resGSPatterns(clust) # to visualize the results
```

The example A reports the default options, where both the gene-specific model parameters are identified and no restrictions on the sign of the time series are applied. In this case, the clustering algorithm allows the identification of a unique pattern grouping all the profiles reported in `sel.diff.data`. In the example B, the argument `onlyK` is set to `TRUE` and, as reported in Figure 12, the algorithm clusters together only `y` and its amplified version (profile "e"), since the gene-specific parameter q is not accounted for. In the example C the argument `onlyK` is `FALSE` and `Psign` is set equal to 0, which means that only genes with correlated temporal patterns according to the linear model and showing the same sign across all the time points can be included into the same cluster. This option might be useful when genes which show at the same time point differential expressions between the two conditions with opposite sign should be kept separated even if they share a common temporal pattern, since they could represent completely different processes. In this case, the algorithm clusters together only the profiles "e" and "d" (Pattern 1) and the profiles "a" and "b" (Pattern 2). The profile "c" results as singleton since the differential expression at the first time point is negative, thus it can not be associated to Pattern 2. It is possible to be less stringent by allowing a minimum number of time points with different sign. If `Psign` is set equal to 1, as in example D, the clustering includes into the same cluster the profiles showing a different sign in at most one time point and the profile "c" is associated to Pattern 2.

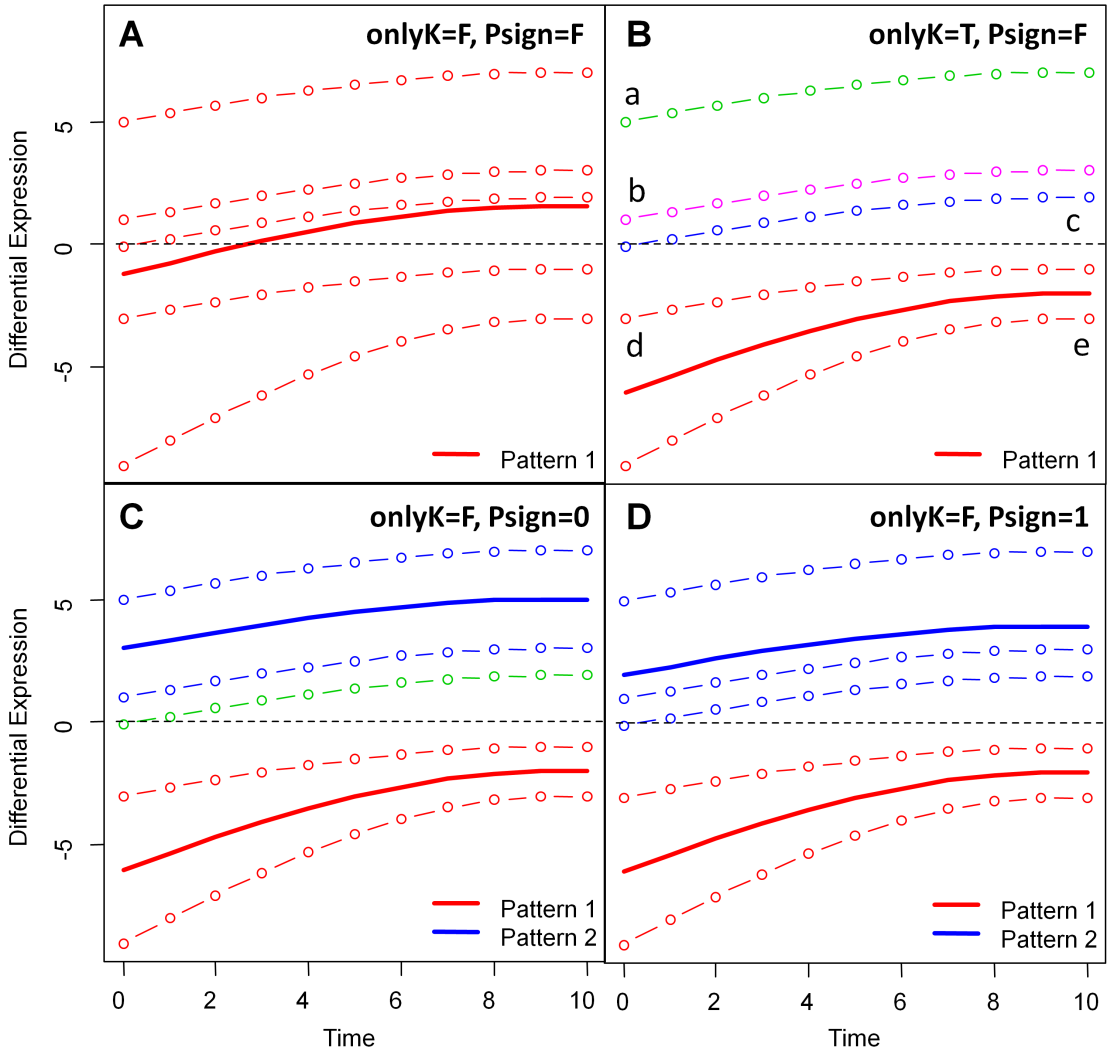


Figure 12: **Clustering examples with different input arguments.** Four different examples of results provided by the function `find.all.pattern` starting from the same data set but using different conmbinations of the input aregument `onlyK` and `Psign`

.

References

[1] Robinson MD, Oshlack A. A scaling normalization method for differential expression analysis of RNA-seq data. *Genome Biol.*, 11(3):R25, 2010.

[2] Sanavia T, Finotello F, Di Camillo B. FunPat: function-based pattern analysis on RNA-seq time series data. *BMC Bioinformatics*, 2015.

[3] Di Camillo B, Toffolo G, Nair SK, Greenlund LJ, Cobelli C. Significance analysis of microarray transcript levels in time series experiments. *BMC Bioinformatics*, 8(Suppl 1):S10, 2007.

[4] Storey JD. A direct approach to false discovery rates. *J R Stat Soc.*, 3:479-498, 2002.

[5] Di Camillo B, Irving BA, Schimke J, Sanavia T, Toffolo G *et al.* Function-based discovery of significant transcriptional temporal patterns in insulin stimulated muscle cells. *PLoS One*, 7(3):e32391, 2012.

[6] Ashburner M, Ball CA, Blake JA, Botstein D, Butler H *et al.* Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat. Genet.*, 25(1):25-29, 2000.

[7] Alexa A, Rahnenfuhrer J, Lengauer T. Improved scoring of functional groups from gene expression data by decorrelating GO graph structure. *Bioinformatics*, 22(13):1600-1607, 2006.

[8] Finotello F, Di Camillo B. Measuring differential gene expression with RNA-seq: challenges and strategies for data analysis. *Brief. Funct. Genomics*, doi:10.1093/bfgp/elu035, 2014.

[9] Dillies MA, Rau A, Aubert J, Hennequet-Antier C, Jeanmougin M *et al.* A comprehensive evaluation of normalization methods for Illumina high-throughput RNA sequencing data analysis. *Brief. Bioinform.*, 14(6):671-683, 2013.