Overview

Pseudotime analysis based on single-cell RNA-seq (scRNA-seq) data has been widely used to study dynamic gene regulatory programs along continuous biological processes such as cell differentiation, immune responses, and disease development. Existing pseudotime analysis methods primarily address the issue of reconstructing cellular pseudotemporal trajectories and inferring gene expression changes along the reconstructed trajectory in one biological sample. As scRNA-seq studies are increasingly performed on multiple patient samples, comparing gene expression dynamics across samples has been emerging as a new demand for which a systematic analytical solution is lacking.

We develop a systematic computational and statistical framework, Lamian, for multi-sample pseudotime analysis. Given scRNA-seq data from multiple biological samples with covariates (e.g., age, sex, sample type, disease status, etc.), this framework allows one to (1) construct cellular pseudotemporal trajectory, evaluate the uncertainty of the trajectory branching structure, (2) evaluate changes in branching structure associated with sample covariates, (3) identify changes in cell abundance and gene expression along the pseudotime, and (4) characterize how sample covariates modifies the pseudotemporal dynamics of cell abundance and gene expression. Importantly, when identifying cell abundance or gene expression changes associated with pseudotime and sample covariates, Lamian accounts for variability across biological samples which other existing pseudotime methods do not consider. As a result, Lamian is able to more appropriately control the false discovery rate (FDR) when analyzing multi-sample data, a property not offered by any other methods.

For more details, see our paper describing the **Lamian** package:

- (give paper info here)

```
# load in Lamian
options(warn=-1)
suppressMessages(library(Lamian))
```

Module 1: tree variability

The module 1 of Lamian is designed for detecting the stability of branches in a pseudotime tree structure. We automatically enumerate all branches from the pseudotime tree structure, and then test for their detection rate through 10,000 bootstraps. After each cell-bootstrapping, we reconstruct the tree structure and re-identify all branches. We apply both Jaccard statistics and overlap coefficient as the statistics for evaluating whether any branch in a bootstrap setting matches with one of the original branches. A branch's detection rate is defined as the percentage of bootstrap settings that a branch finds it match. Module 1 also reports and tests for samples' proportion in each branch.

2.1 data preparation

We will need a gene by cell expression matrix, the low-dimension representation of the cells, and the cell annotation (which sample each cell belongs to). Here, we use example data **hca_bm_saver**, **hca_bm_pca**, **hca_bm_cellanno** to demonstrate their data structures. Users can read in their own data of interest in this step.

```
data(hca_bm_saver)
data(hca_bm_pca)
data(hca_bm_cellanno)
```

hca_bm_saver is a gene by cell expression matrix.

```
str(hca_bm_saver)
```

```
##  num [1:121, 1:11139] 0.3984 0.2302 0.3173 0.1763 0.0594 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:121] "CD38" "CDC27" "CD79B" "CDK11A" ...
##   ..$ : chr [1:11139] "BM1:52:female_350" "BM1:52:female_1262" "BM1:52:fe
```

hca_bm_pca is the low-dimension representation of the cells.

```
str(hca_bm_pca)
```

```
##  num [1:11139, 1:50] -8.79 -3.22 -5.51 -1.94 -4.68 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:11139] "BM1:52:female_350" "BM1:52:female_1262" "BM1:52:fe
##   ..$ : chr [1:50] "PC_1" "PC_2" "PC_3" "PC_4" ...
```

hca_bm_cellanno is the cell annotation where the first column are the cell names, the second column are the sample names, and the row names are the cell names.

```
str(hca_bm_cellanno)
```

```
## 'data.frame':    11139 obs. of  2 variables:
##  $ cell  : chr  "BM1:52:female_350" "BM1:52:female_1262" "BM1:52:female_2
##  $ sample: chr  "BM1" "BM1" "BM1" "BM1" ...
```

2.2 infer tree structure

```
res = infer_tree_structure(pca = hca_bm_pca,
                           expression = hca_bm_saver,
                           cellanno = hca_bm_cellanno,
                           origin.marker = c('CD34'),
                           xlab='Principal component 1',
                           ylab = 'Principal component 2')
```

```
##  num [1:5(1d)] 0.00935 1.07903 0.02155 0.00967 0.02627
##  - attr(*, "dimnames")=List of 1
##   ..$ : chr [1:5] "1" "2" "3" "4" ...
## List of 3
##  $ backbone 2,3,1: chr [1:4496] "BM6:26:female_13022" "BM1:52:female_266
##  $ branch: 2,4   : chr [1:2017] "BM6:26:female_184254" "BM4:29:male_2423
##  $ branch: 2,5   : chr [1:4626] "BM4:29:male_77649" "BM3:39:male_78475"
```

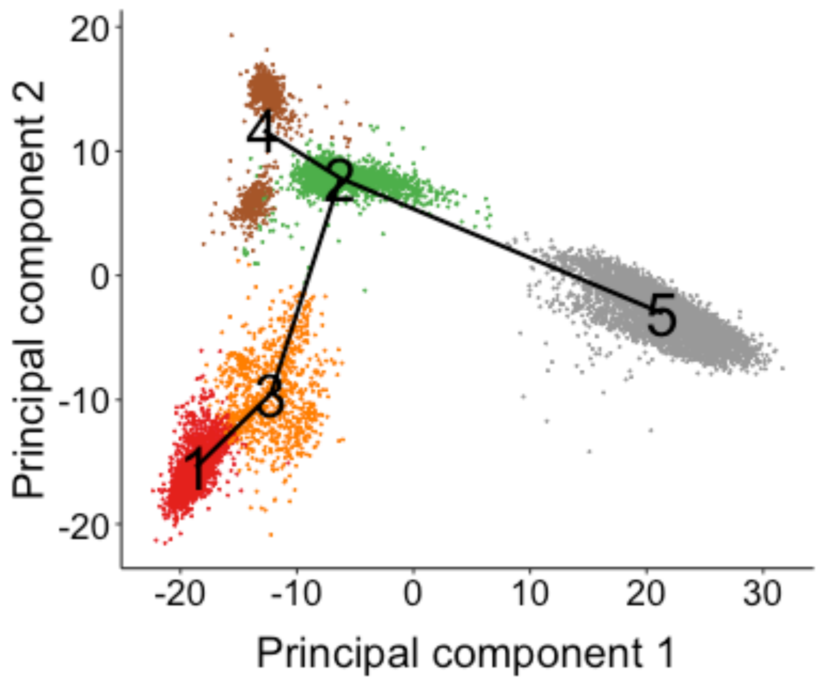
As we can see from the above inputs, there are five clusters, and the tree structure inferred based on these clusters consists of three branches: 2 –> 3 –> 1, 2 –> 4, and 2 –> 5. The result object **res** is a list containing information about the tree structure, branches, cell clusters, pseudotime ordering the cells, etc..

```
names(res)
```

```
##  [1] "pcareduceres" "MSTtree"      "clusterid"    "clucenter"    "pseudot
##  [6] "branch"       "js.cut"       "oc.cut"       "pca"          "order"
## [11] "allsample"
```

plot the tree structure

```
plotmclust(res, cell_point_size = 0.1,
           x.lab = 'Principal component 1',
           y.lab = 'Principal component 2')
```



2.3 evaluate the uncertainty of tree branches

```
result <- evaluate_uncertainty(res, n.permute=3)
```

```
## [1] 1
## [1] 2
## [1] 3
```

```
names(result)
```

```
## [1] "detection.rate"       "sample.cellcomp.mean" "sample.cellcomp.sd"
```

The result is a list of three elements. The first selement is the detection rate of each branch.

```
result[[1]]
```

```
##            detection.rate
## c(2, 3, 1)            1.0
## c(2, 4)               1.0
## c(2,5)                0.5
```

The second element is the sample proportion mean information.

```
result[[2]]
```

```
##                  BM1       BM2      BM3     BM4 BM5 BM6 BM7 BM8
## c(2, 3, 1) 0.4063686 0.6473673 0.674068 0.43813   0   0   0   0
## c(2, 4)    0.5936314 0.3526327 0.325932 0.56187   1   1   1   1
## c(2,5)     0.0000000 0.0000000 0.000000 0.00000   0   0   0   0
```

The thrid element is the sample proportion sd (standard deviation) information.

```
result[[3]]
```

```
##                   BM1         BM2        BM3        BM4 BM5 BM6 BM7 BM8
## c(2, 3, 1) 0.004411669 0.008335887 0.01032617 0.01871458   0   0   0   0
## c(2, 4)    0.004411669 0.008335887 0.01032617 0.01871458   0   0   0   0
## c(2,5)     0.000000000 0.000000000 0.00000000 0.00000000   0   0   0   0
```

Module 2: Trajectory differential tests

2.1 sample covariate test

data preparation

In the following, we will use an example dataset **expdata** of 100 genes and 1000 cells to demonstration the workflow. In practice, we can input any other interesting dataset.

```
data(expdata)
```

The inputs should contain: (a) **expr**: a gene by cell expression matrix. Values are library-size-normalized log-transformed gene expression matrix. They can be either imputed or non-imputed. Zero-expression genes should have been filtered out.

```
str(expdata$expr)
```

```
##  num [1:120, 1:1000] 1.973 1.163 0.526 1.637 0.34 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:120] "TPM4:ENSG00000167460" "HLA-DMB:ENSG00000242574" "KHS
##   ..$ : chr [1:1000] "BM4:26:female_219004" "BM1:26:female_219241" "BM4:2
```

(b) **cellanno**: a dataframe where the first column are cell names and second column are sample names.

```
head(expdata$cellanno)
```

```
##                    Cell Sample
## 1 BM4:26:female_219004    BM4
## 2 BM1:26:female_219241    BM1
## 3   BM4:29:male_179571    BM4
## 4  BM3:26:female_52851    BM3
## 5  BM8:52:female_54061    BM8
## 6 BM8:52:female_117933    BM8
```

© **pseudotime**: a numeric vector of pseudotime, and the names of this vector are the corresponding cell names.

```
str(expdata$pseudotime)
```

```
##  Named int [1:1000] 1 2 3 4 5 6 7 8 9 10 ...
##  - attr(*, "names")= chr [1:1000] "BM4:26:female_219004" "BM1:26:female_2
```

(d) **design**: a matrix. Number of rows should be the same as the number of unique samples. Rownames are sample names. First column is the intercept (all 1), second column is the covariate realization valuels for each of the samples.

```
print(expdata$design)
```

```
##     intercept group
## BM1         1     1
## BM2         1     1
## BM3         1     0
## BM4         1     0
## BM5         1     1
## BM6         1     1
## BM7         1     0
## BM8         1     0
```

The function `lamian.test()` is designed to perform multiple tests. To perform the Sample Covariate Test, we need to set **test.type = 'variable'**.

perform sample covariate test

```
Res <- lamian.test(expr = expdata$expr,
                   cellanno = expdata$cellanno,
                   pseudotime = expdata$pseudotime,
                   design = expdata$design,
                   test.type = 'variable')
```

```
## [1] "fitting model: overall: CovariateTest (Model 3 vs.1), or ConstantTes
## [1] "Number of overall DDG: 100"
## [1] "meanDiff pvalues: Model 2 vs. model 1..."
## [1] "trendDiff pvalues: Model 3 vs. model 2..."
```

downstream analysis

identify differential dynamic genes (DDG)

We will know which are DDG from the **statistics** data frame in the result object **Res**.

```
## get differential dynamic genes statistics
stat <- Res$statistics
head(stat)
```

```
##                             fdr.overall  pval.overall z.overall  fdr.meanDi
## TPM4:ENSG00000167460     2.424242e-02  2.000000e-02  3.498514  1.916968e-
## HLA-DMB:ENSG00000242574 2.731441e-120 1.092576e-120 12.043571  2.848972e-
## KHSRP:ENSG00000088247   3.073896e-232 4.610844e-233 13.820409  0.000000e+
## ACER3:ENSG00000078124   1.056615e-114 4.402562e-115 10.594220  3.774976e-
## TMEM14A:ENSG00000096092 1.118422e-166 3.355265e-167 12.962281 5.484130e-2
## FAM168B:ENSG00000152102  1.314007e-24  9.307553e-25  6.110638  8.841498e-
##                           pval.meanDiff z.meanDiff fdr.trendDiff pval.trend
## TPM4:ENSG00000167460     1.054332e-03  3.2059082  3.665855e-02   2.01622
## HLA-DMB:ENSG00000242574  1.367507e-14  3.9939583  4.008455e-16   1.68355
## KHSRP:ENSG00000088247    0.000000e+00 14.6381159  1.605933e-01   9.47500
## ACER3:ENSG00000078124    1.849738e-09  2.9803194  1.467708e-12   6.31114
## TMEM14A:ENSG00000096092 1.645239e-217 12.7611075  1.589712e-01   9.22032
## FAM168B:ENSG00000152102  7.844230e-01 -0.6691631  4.894896e-26   1.90900
##                           z.trendDiff
## TPM4:ENSG00000167460        2.542818
## HLA-DMB:ENSG00000242574     5.870766
## KHSRP:ENSG00000088247       1.418965
## ACER3:ENSG00000078124       5.089068
## TMEM14A:ENSG00000096092     1.608075
## FAM168B:ENSG00000152102     8.007251
```

```
stat <- stat[order(stat[,1], -stat[,3]), ]
diffgene <- rownames(stat[stat[, grep('^fdr.*overall$', colnames(stat))] < 0
str(diffgene)
```

```
##  chr [1:100] "MAP3K5:ENSG00000197442" "CRY1:ENSG00000008405" ...
```

population-level model-fitted patterns

We will get the population-level estimates for all the DDG by applying function `getPopulationFit()`.

```
## population fit
Res$populationFit <- getPopulationFit(Res, gene = diffgene, type = 'variable
```

clustering

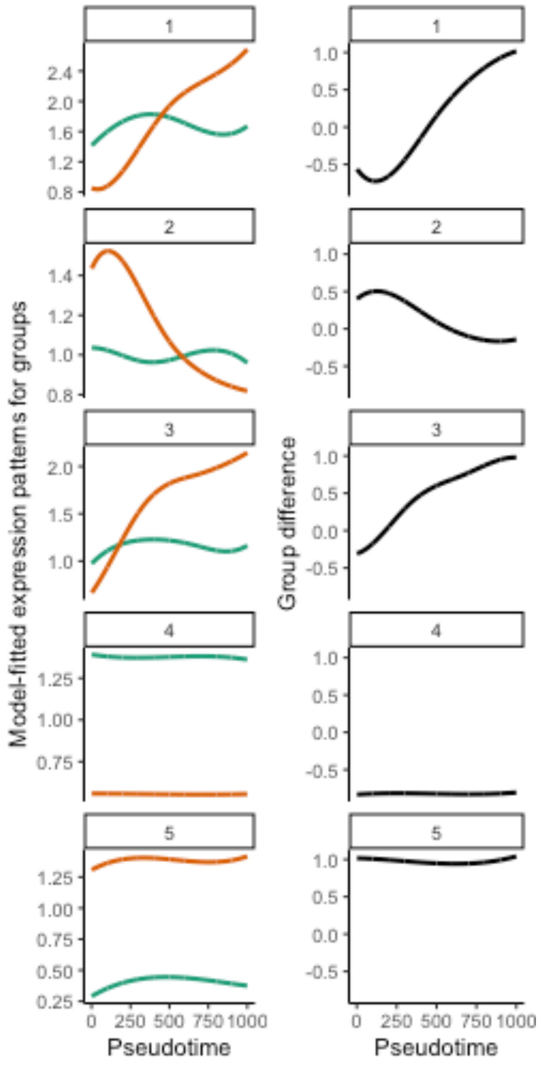
We can apply `getCovariateGroupDiff()` to calculate the group difference regarding this sample covariate, and then cluster the DDG based on the group difference. By setting **k = 4**, we will get two clusters for meanSig DDG, and the other two clusters for DDG of other types.

```
## clustering
Res$covariateGroupDiff <- getCovariateGroupDiff(testobj = Res, gene = diffge
Res$cluster <- clusterGene(Res, gene = diffgene, type = 'variable', k=5)
table(Res$cluster)
```

```
##
##  1  2  3  4  5
## 23 20 13 19 25
```

plot the cluster mean and difference

```
## plotClusterMeanAndDiff
plotClusterMeanAndDiff(Res)
```



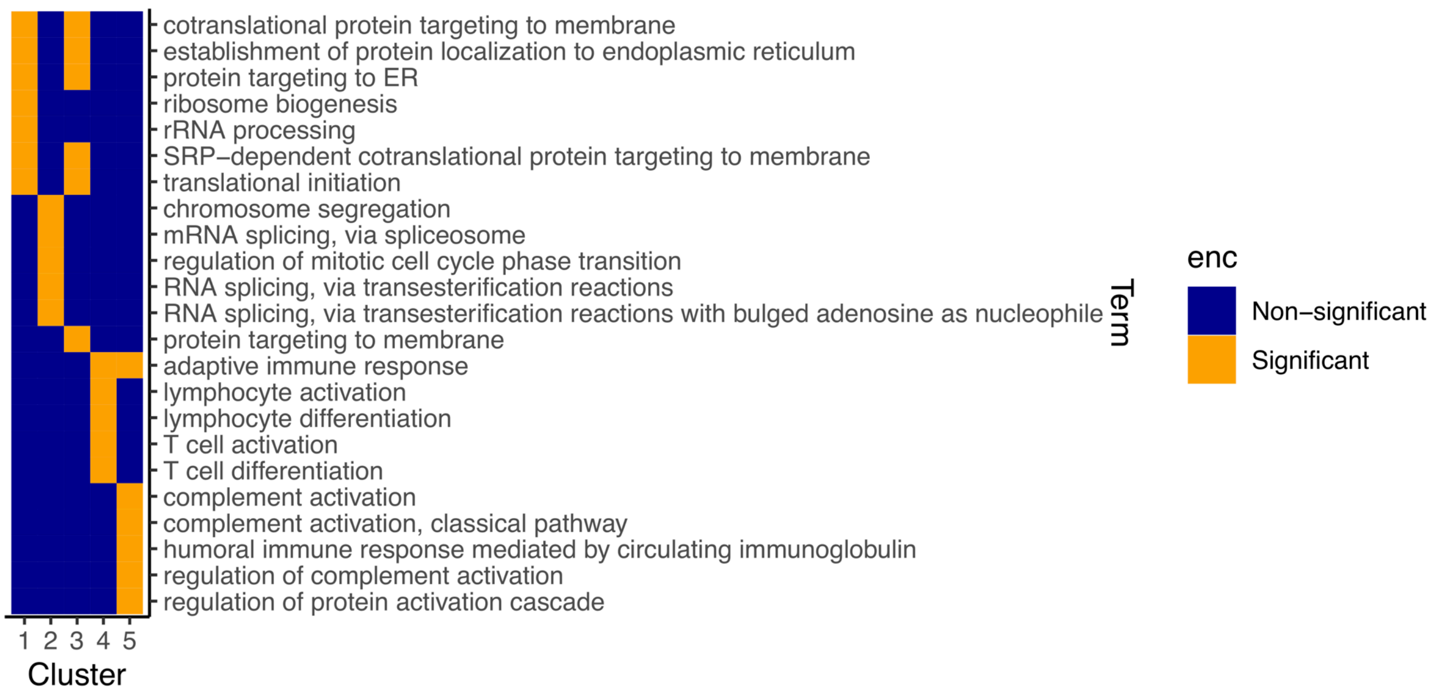
We can also plot the cluster difference seperately by calling `plotClusterDiff(testobj=Res, gene = diffgene)`.

GO analysis

We can further check out whether the DDG in each cluster have any enriched genome functions by applying Gene Ontology (GO) analysis.

```
## GO analysis
goRes <- GOEnrich(testobj = Res, type = 'variable')
plotGOEnrich(goRes = goRes, fdr.cutoff = 0.05, fc.cutoff = 2)
```

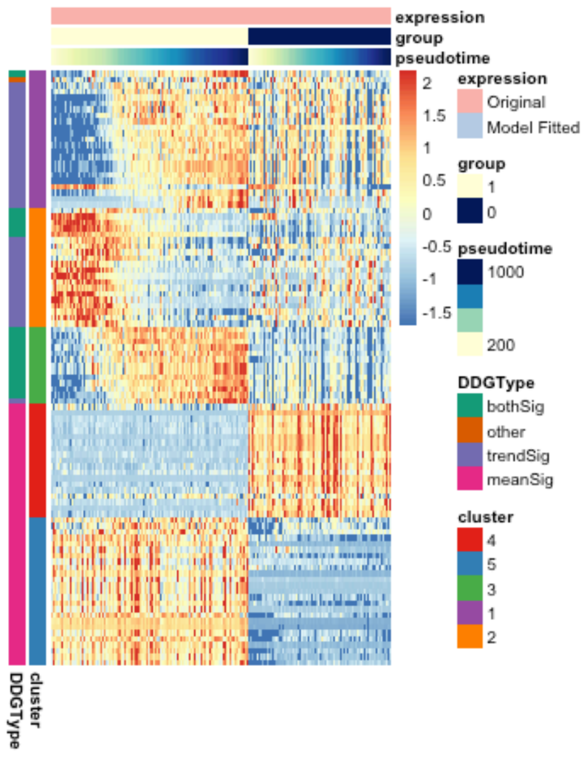
In practice, if there are significant GO terms in the clusters, the above `plotGOEnrich()` function will generate a heatmap to show the top GO terms, as follows.

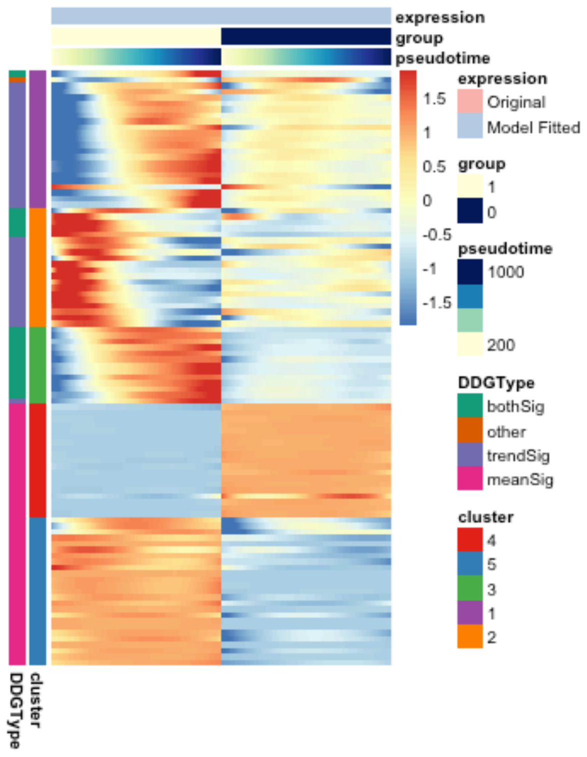


compare original and fitted expression

```
plotFitHm(Res, type = 'variable', cellWidthTotal = 200, cellHeightTotal = 35
```

```
## [1] "subsample done!"
```



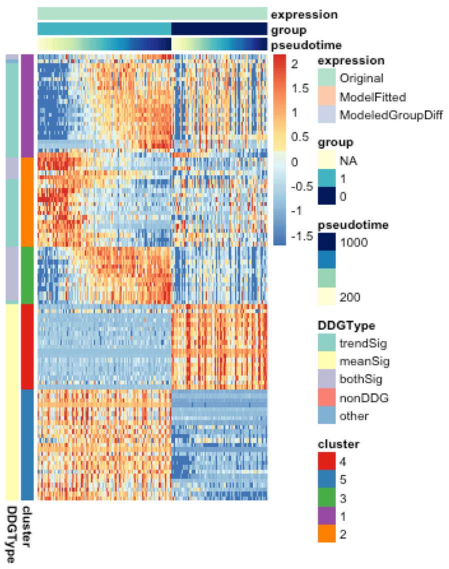


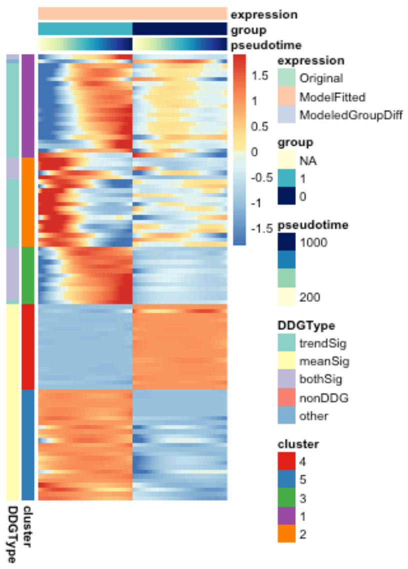
```
## TableGrob (1 x 9) "arrange": 3 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-4) arrange gtable[layout]
## 2 2 (1-1,5-5) arrange gtable[layout]
## 3 3 (1-1,6-9) arrange gtable[layout]
```

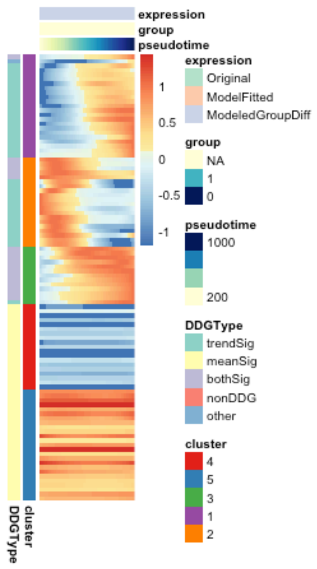
We can also plot original expression values, model fitted expression values, and model-fitted group difference in three seperate heatmaps, as follows.

```
plotDiffFitHm3(Res, cellWidthTotal = 180, cellHeightTotal = 350)
```

```
## [1] "subsample done!"
```







2.2 constant time test

data preparation

The inputs for Constant Time Test are the same as those for Sample Covariate Test (see above section), except that the **design** matrix can have only one intercept column. If there are more than one columns in **design**, only the first column will be considered.

perform constant time test

```
Res <- lamian.test(expr = expdata$expr,
                   cellanno = expdata$cellanno,
                   pseudotime = expdata$pseudotime,
                   design = expdata$design,
                   test.type = 'time',
                   ncores = 1)
```

```
## [1] "fitting model: overall: CovariateTest (Model 3 vs.1), or ConstantTes
## [1] "Number of overall DDG: 85"
## [1] "Not returning meanDiff and trendDiff: constantTest, user required, c
```

```
names(Res)
```

```
##  [1] "statistics"  "parameter"   "llr.overall" "knotnum"     "pseudotime'
##  [6] "design"      "cellanno"    "expr"        "test.type"   "test.methoc
```

downstream analysis

The result object is a list containing multiple elements. The first element is a dataframe of statistics.

```
head(Res$statistics)
```

```
##                          fdr.overall pval.overall z.overall
## TPM4:ENSG00000167460               0            0 124.39535
## HLA-DMB:ENSG00000242574            0            0  53.86389
## KHSRP:ENSG00000088247              0            0  29.46020
## ACER3:ENSG00000078124              0            0 114.14288
## TMEM14A:ENSG00000096092            0            0  32.97112
## FAM168B:ENSG00000152102            0            0  48.39983
```

We can further determine the dynamic genes (DG) as the genes with **fdr.overall** < 0.05.

```
diffgene <- rownames(Res$statistics)[Res$statistics[,1] < 0.05]
str(diffgene)
```

```
##  chr [1:85] "TPM4:ENSG00000167460" "HLA-DMB:ENSG00000242574" ...
```

population-level model-fitted patterns

We can estimate the population-level fitting for all DG, which is useful for the downstream analysis.

```
Res$populationFit <- getPopulationFit(Res, gene = diffgene, type = 'time')
```

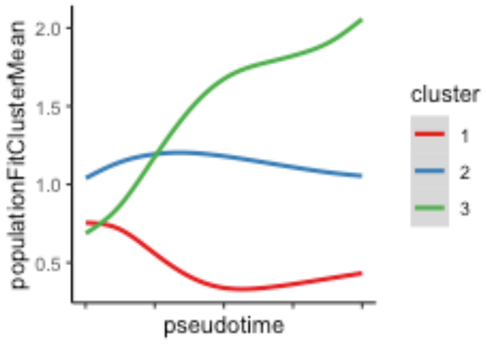
clustering

```
Res$cluster <- clusterGene(Res, gene = diffgene, type = 'time', k=3)
```

plot cluster mean

```
plotClusterMean(testobj=Res, cluster = Res$cluster, type = 'time')
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

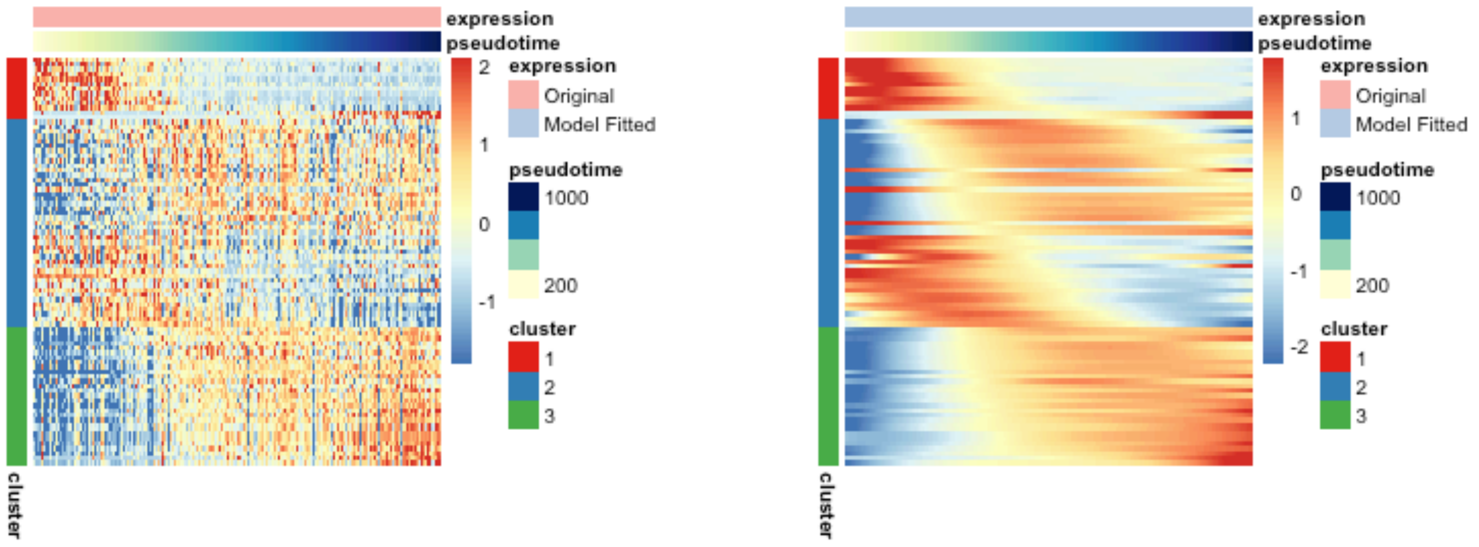


GO analysis

```
goRes <- GOEnrich(testobj = Res, type = 'time', sep = ':.*')
plotGOEnrich(goRes = goRes)
```

compare original and fitted expression

```
plotFitHm(
  Res,
  subsampleCell  = F,
  showCluster = T,
  type = 'time',
  cellWidthTotal = 200,
  cellHeightTotal = 200
)
```



```
## TableGrob (1 x 9) "arrange": 3 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-4) arrange gtable[layout]
## 2 2 (1-1,5-5) arrange gtable[layout]
## 3 3 (1-1,6-9) arrange gtable[layout]
```

2.3 cell proportion test

data preparation

The inputs for Cell Proportion test are the same as those for Sample Covariate Test (see above section).

perform cell proportion test

```
Res <-
  cell_prop_test(
    cellanno = expdata$cellanno,
    pseudotime = expdata$pseudotime,
    design = expdata$design,
    ncores = 4
  )
```

```
## [1] "fitting model: overall: CovariateTest (Model 3 vs.1), or ConstantTes
## [1] "Number of overall DDG: 0"
## [1] "Not returning meanDiff and trendDiff: constantTest, user required, c
```

The dataframe **statistics** in the **Res** object contains the test result: a *p*-value **pval.overall** and a effect size *z*-score **z.overall**.

```
Res$statistics
```

```
##      pval.overall z.overall
## prop    0.8365149 -0.927402
```

Session Info

```
sessionInfo()
```

```
## R version 4.0.2 (2020-06-22)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Mojave 10.14.5
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRbl
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRl
##
## locale:
## [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
## [1] Lamian_0.0.1
##
## loaded via a namespace (and not attached):
##   [1] TSCAN_1.7.0          nlme_3.1-148         matrixStats_0.57.0
##   [4] bitops_1.0-6         fs_1.4.2             usethis_1.6.1
##   [7] devtools_2.3.0       bit64_0.9-7          RColorBrewer_1.1-2
##  [10] rprojroot_2.0.2      tools_4.0.2          R6_2.5.0
##  [13] KernSmooth_2.23-17   DBI_1.1.0            BiocGenerics_0.35.4 
##  [16] mgcv_1.8-33          colorspace_1.4-1     withr_2.3.0         
##  [19] tidyselect_1.1.0     gridExtra_2.3        prettyunits_1.1.1   
##  [22] processx_3.4.4       bit_4.0.4            compiler_4.0.2      
##  [25] graph_1.67.1         cli_2.1.0            Biobase_2.49.0      
##  [28] SparseM_1.78         desc_1.2.0           labeling_0.4.2      
##  [31] topGO_2.42.0         caTools_1.18.0       scales_1.1.1        
##  [34] callr_3.5.1          stringr_1.4.0        digest_0.6.27       
##  [37] rmarkdown_2.7        pkgconfig_2.0.3      htmltools_0.5.0     
##  [40] sessioninfo_1.1.1    highr_0.8            fastmap_1.0.1       
##  [43] rlang_0.4.8          RSQLite_2.2.0        shiny_1.5.0         
##  [46] farver_2.0.3         generics_0.1.0       combinat_0.0-8      
##  [49] mclust_5.4.6         gtools_3.8.2         dplyr_1.0.2         
##  [52] magrittr_1.5         GO.db_3.12.1         Matrix_1.2-18       
##  [55] Rcpp_1.0.5           munsell_0.5.0        S4Vectors_0.27.12   
##  [58] fansi_0.4.1          viridis_0.5.1        lifecycle_0.2.0     
##  [61] stringi_1.5.3        org.Hs.eg.db_3.12.0  pkgbuild_1.1.0      
##  [64] gplots_3.0.4         plyr_1.8.6           matrixcalc_1.0-3    
##  [67] grid_4.0.2           blob_1.2.1           gdata_2.18.0        
##  [70] promises_1.1.1       crayon_1.3.4         lattice_0.20-41     
##  [73] splines_4.0.2        knitr_1.33           ps_1.4.0            
##  [76] pillar_1.4.6         igraph_1.2.5         reshape2_1.4.4      
##  [79] stats4_4.0.2         pkgload_1.1.0        glue_1.4.2          
##  [82] evaluate_0.14        remotes_2.1.1        vctrs_0.3.4         
##  [85] httpuv_1.5.4         testthat_3.0.0       gtable_0.3.0        
##  [88] purrr_0.3.4          scattermore_0.7      assertthat_0.2.1    
##  [91] ggplot2_3.3.2        xfun_0.22            mime_0.9            
##  [94] xtable_1.8-4         later_1.1.0.1        viridisLite_0.3.0   
##  [97] tibble_3.0.4         pheatmap_1.0.12      AnnotationDbi_1.52.0
## [100] memoise_1.1.0        IRanges_2.23.10      fastICA_1.2-2       
## [103] ellipsis_0.3.1
```

Citation

If the **Lamian** package is useful in your work, please cite the following paper:

- (give paper info here)