# Trendy: Segmented regression analysis of expression dynamics for high-throughput ordered profiling experimentsa

Rhonda Bacher, Ning Leng, Ron Stewart

August 9, 2017

## Contents

## 1 Overview

Trendy is an R package that can be used to perform breakpoint analysis on microarray or RNA-seq expression data with ordered conditions (e.g. time-course, spatial-course). For each gene or other features, Trendy estimates the optimal number of breakpoints as well as the breakpoints by fitting a set of segmented regression models. The top dynamic genes are then identified by taking genes that can be well profiled by its gene-specific segmented regression model. Trendy also implements functions to visualize the dynamic genes and their trends, to order dynamic genes by their trends, and to compute breakpoint distribution at different time-points (e.g. detect time-points with a large number of expression changes).

### 1.1 The model

To illustrate Trendy, here we use time course gene expression data as an example. Although, Trendy may also be applied to other types of features (e.g. isoform or exon expression) and/or other types of experiments with ordered conditions (e.g. spatial course).

Denote the normalized gene expression of gene $g$ and sample/time $t$ as $Y_{g,t}$. Denote the total number of genes as $G$ and the total number of samples/times as $N$. For each gene, Trendy fits segmented regression models with varying numbers of breakpoints from 1 to $K$. $K$ defaults to 3 but can also be specified by the user. The segmented R package is used to fit the segmented regression models.

For a given gene, among the models with varying $k$, Trendy selects the optimal number of breakpoints for this gene by comparing the coefficient of determination ($R^2$) for each model.

To avoid overfitting, the optimal number of breakpoints will be set as $\tilde{k_g} = \tilde{k_g} - 1$ if any of the following happens: at least one segment has less than $c_{num}$ samples or $R^2_{g,\tilde{k_g}} - R^2_{g,\tilde{k_g}-1} < c_{diff}$. The thresholds $c_{num}$ and $c_{diff}$ can be specified by the user; the defaults are 5 and 0.1, respectively.

Trendy reports the following for the optimal model:

- Gene specific adjusted $R^2$ (penalized for the chosen value of $k$)
- Segment slopes
- Breakpoint estimates

Among all genes, the top dynamic genes are defined as those whose optimal model has high adjusted $R^2$s.

To compute the breakpoint distribution over the time-course, Trendy calculates the number of breakpoints for each time-point across all the genes.

The time-points with high $D_t$ can be considered as those with global expression changes.

Trendy also summarizes the fitted trend or expression pattern of top genes. For samples between the $i^{th}$ and $i+1^{th}$ breakpoint for a given gene, if the t-statistic of the segment slope has p-value greater than $c_{pval}$, the trend of this segment will be defined as no change. Otherwise the trend of this segment will be defined as up/down based on the slope coefficient. The default value of $c_{pval}$ is 0.1, but may also be specified by the user.

## 2 Installation

### 2.1 Install via GitHub

The Trendy package can be installed using functions in the devtools package.

To install, type the following code into R:

```r
install.packages("devtools")
library(devtools)
install_github("rhondabacher/Trendy/package/Trendy")
```

### 2.2 Install locally

Install packages segmented and gplots:

```r
install.packages(c("segmented","gplots"))
library("segmented")
library("gplots")
```

Download the Trendy package from: https://github.com/rhonda/Trendy/tree/master/package

And install the package locally.

### 2.3 Load the package

To load the Trendy package:

```
library(Trendy)
```

## 3 Analysis

### 3.1 Input

The input data should be a $G - by - N$ matrix containing the expression values for each gene and each sample, where $G$ is the number of genes and $N$ is the number of samples. The samples should be sorted following the time course order. These values should exhibit expression data after normalization across samples. For example, for RNA-seq data, the raw counts may be normalized using MedianNorm and `GetNormalizedMat` function in *EBSeq*. More details can be found in the *EBSeq* vignette:

http://www.bioconductor.org/packages/devel/bioc/vignettes/EBSeq/inst/doc/EBSeq_Vignette.pdf

The object TrendyExData is a simulated data matrix containing 50 rows of genes and 40 columns of samples.

```
data(TrendyExData)
str(TrendyExData)

##  num [1:50, 1:40] 240 199 198 239 202 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:50] "g1" "g2" "g3" "g4" ...
##   ..$ : chr [1:40] "s1" "s2" "s3" "s4" ...
```

### 3.2 Run Trendy

The `trendy` function will fit multiple segmented regressions model for each gene (via the *segmented* R package) and select the the optimal model. Here we want to only consider a maximum of two breakpoints for each gene.

```
res <- trendy(TrendyExData, Max.K = 2)
res.top <- toptrendy(res)
# default adjusted R square cutoff is 0.5
res.top$radj

##        g3        g1       g28       g20       g15        g2       g10       g23        g8
## 0.9787382 0.9775005 0.9751380 0.9739715 0.9729747 0.9710139 0.9705118 0.9701402 0.9691341
##        g5       g24       g17       g12       g29       g16       g22       g18       g25
## 0.9689555 0.9656732 0.9652141 0.9644343 0.9632348 0.9630272 0.9627092 0.9626837 0.9611528
##       g11       g30       g26        g7        g4        g9       g21        g6       g19
## 0.9600736 0.9597989 0.9572072 0.9529077 0.9420853 0.9377311 0.9304116 0.9291045 0.9259893
##       g27       g14       g13
## 0.9183375 0.8656596 0.8576471
```

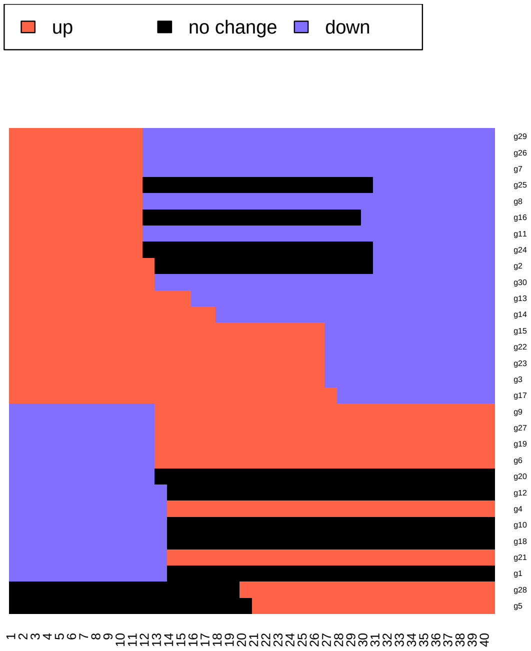
The `toptrendy` function may be used to extract top dynamic genes. By default, `toptrendy` will extract genes whose adjusted $R^2$, $\bar{R}^2$, is greater or equal to 0.5. To change this threshold, a user may specify the `AdjR.Cut` parameter in the `toptrendy` function. res.top$radj gives the $\bar{R}^2$ of the top dynamic genes sorted decreasingly.

By default the `trendy` function only considers genes whose mean expression is greater than 10. To use another threshold, the user may specify the parameter `Mean.Cut`.

### 3.3 Visualize trends of the top dynamic genes

res.top$id.sign gives the trend specification of the top genes. The function `trendheatmap` can be used to display these trends:

```
res.trend <- trendheatmap(res.top)
```

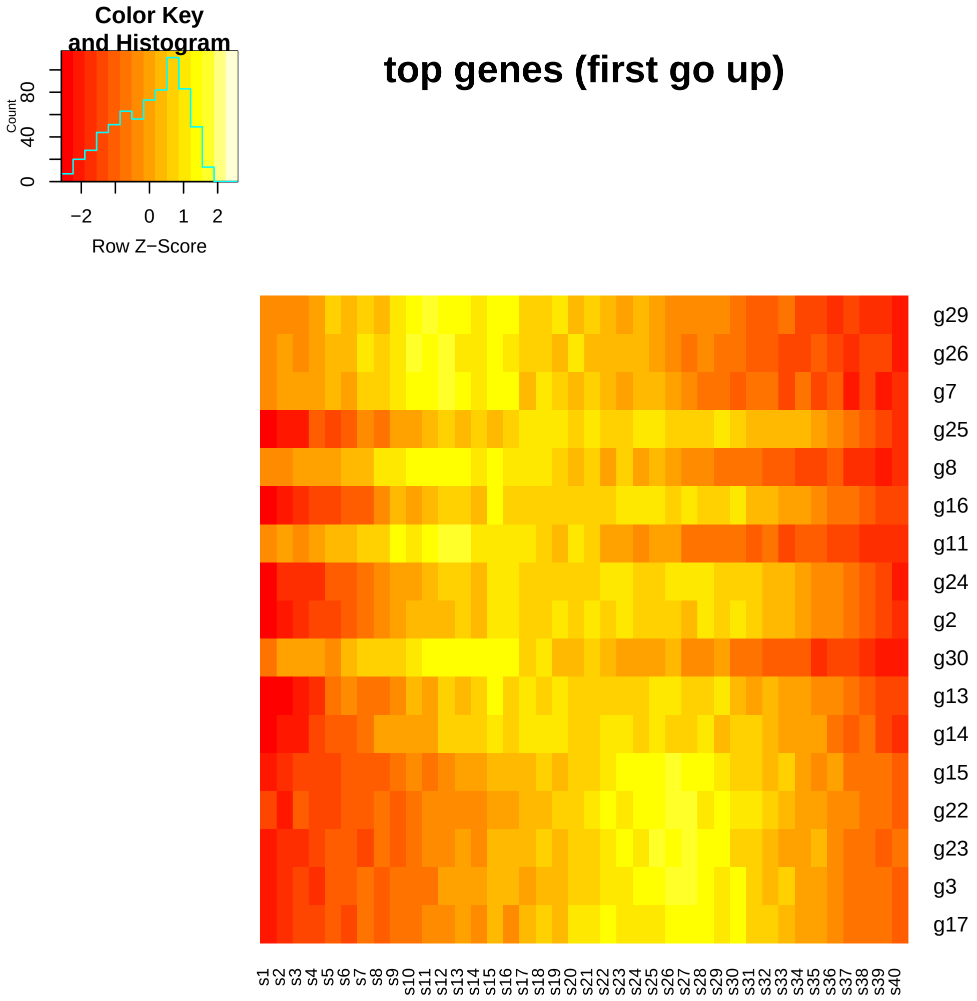


```
str(res.trend)

## List of 3
##  $ firstup      : Named num [1:17] 11.4 11.5 11.6 11.6 11.6 ...
##   ..- attr(*, "names")= chr [1:17] "g29" "g26" "g7" "g25" ...
##  $ firstdown    : Named num [1:11] 12.1 12.6 12.6 12.7 12.8 ...
##   ..- attr(*, "names")= chr [1:11] "g9" "g27" "g19" "g6" ...
##  $ firstnochange: Named num [1:2] 19 20.4
##   ..- attr(*, "names")= chr [1:2] "g28" "g5"
```

The `trendheatmap` function classifies the top dynamic genes into three groups: start with up, start with down and start with no change. Within each group, genes are sorted by the position of the first breakpoint.

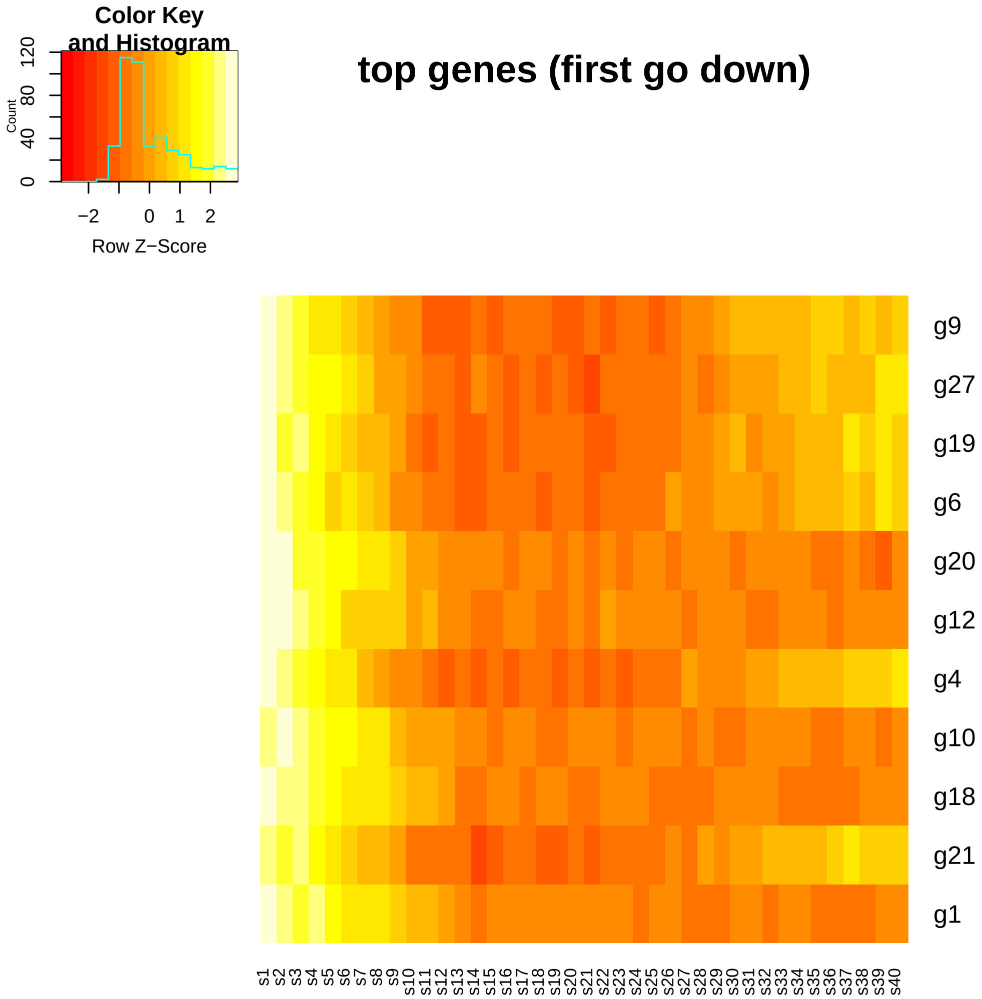
To generate an expression heatmap of the first group of genes (first go up):

```
library(gplots)
heatmap.2(TrendyExData[names(res.trend$firstup),],
  trace="none", Rowv=F,Colv=F,
        scale="row", main="top genes (first go up)")
```



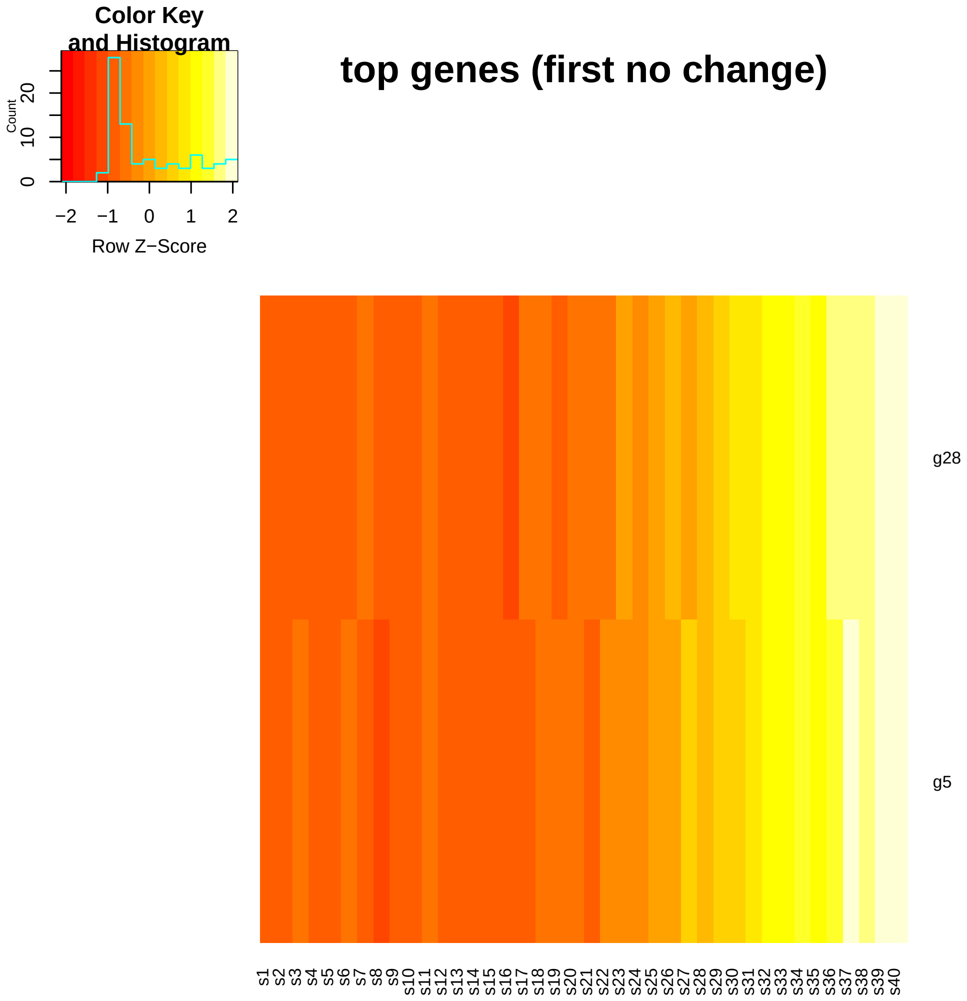
Similarly, to generate an expression heatmap of the second group of genes (first go down):

```
heatmap.2(TrendyExData[names(res.trend$firstdown),],
  trace="none", Rowv=F,Colv=F,
        scale="row", main="top genes (first go down)")
```



To generate an expression heatmap of the second group of genes (first no change):

```
heatmap.2(TrendyExData[names(res.trend$firstnochange),],
  trace="none", Rowv=F,Colv=F,
        scale="row", main="top genes (first no change)",
        cexRow=.8)
```

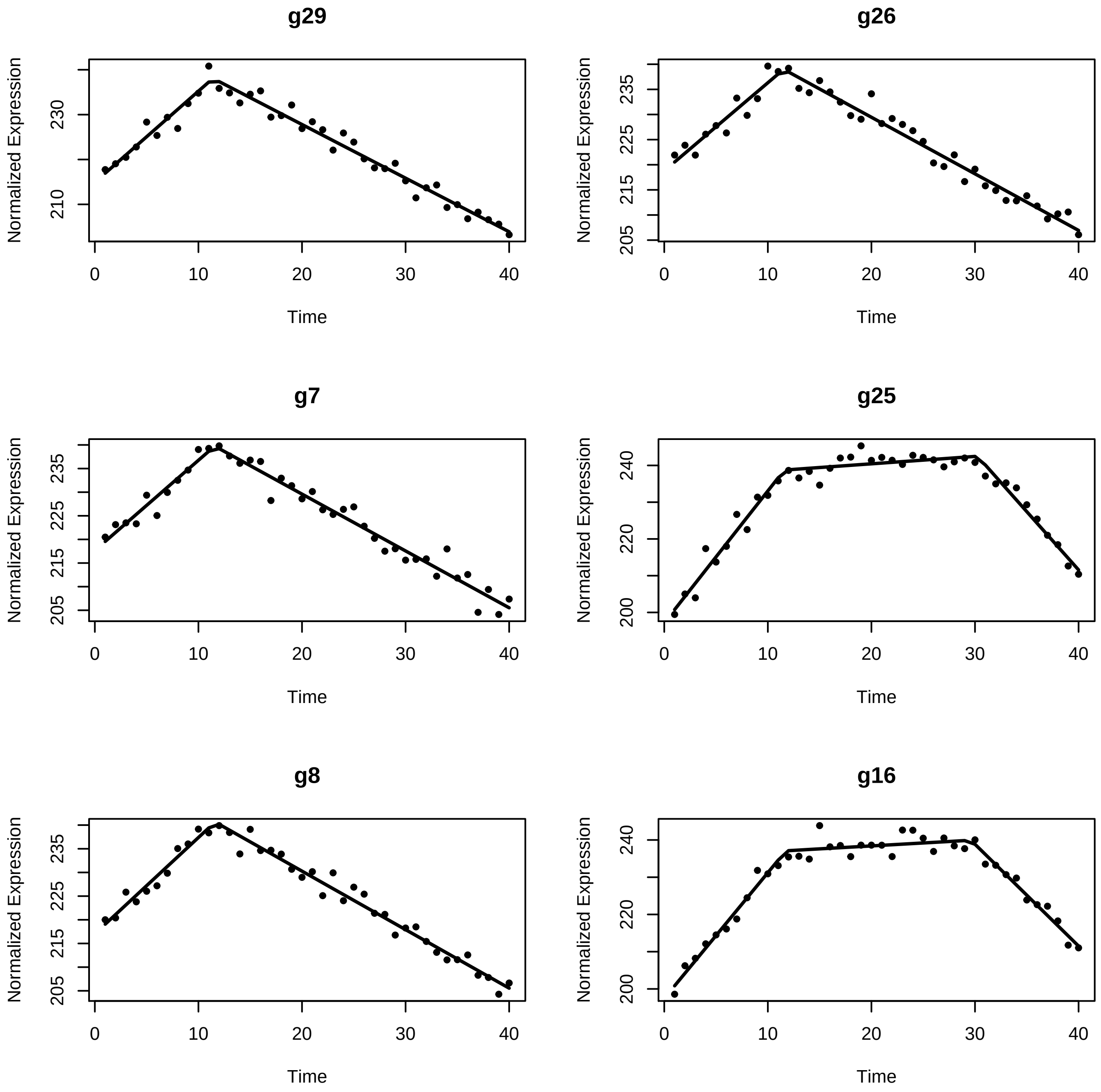


## 3.4 Visualize individual genes

The `plotfeature` function may be used to plot expression of individual features/genes and the fitted lines.

For example, to plot the top six genes in the first group of genes (first go up):

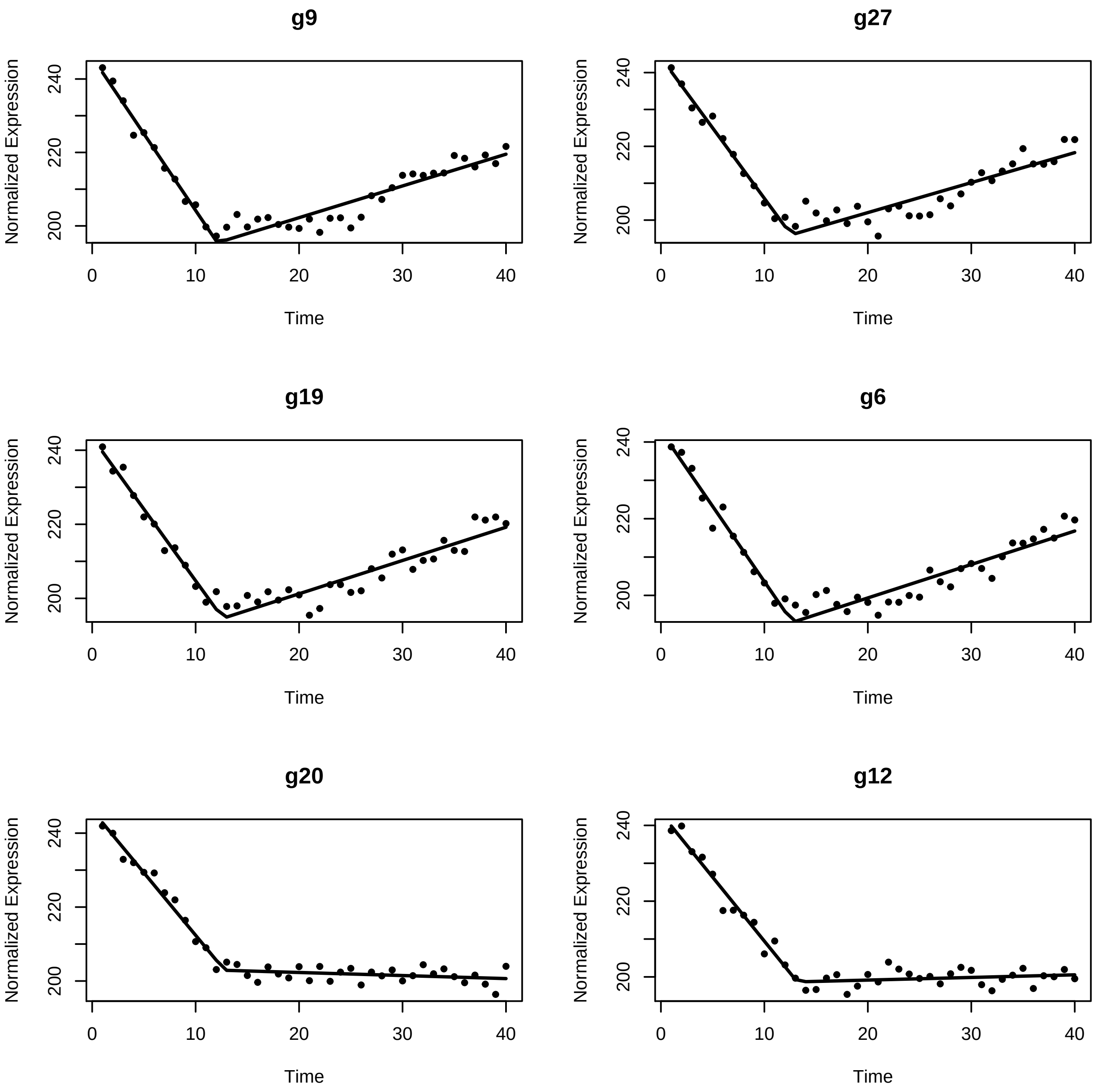
```
par(mfrow=c(3,2))
plot1 <- plotfeature(TrendyExData,
                     Feature.Names = names(res.trend$firstup)[1:6],
                     Seg.Data = res)
```



The input of function `plotfeature` requires the expression data and a list of genes of interest. The parameter `Seg.Data` results from the `trendy` function. If it is not specified, then `plotfeature` will run `trendy` on the genes of interest before plotting. Specifying the fitted results obtained from previous steps will save time by avoiding fitting the models again.

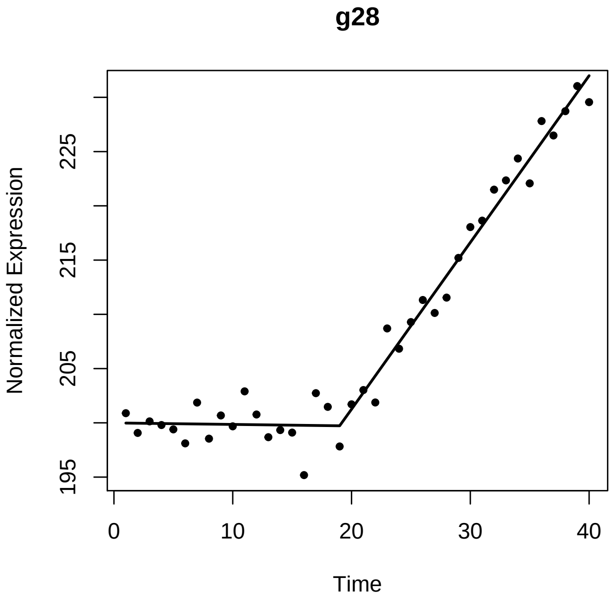
Similarly, to plot the top six genes in the second group of genes (first go down):

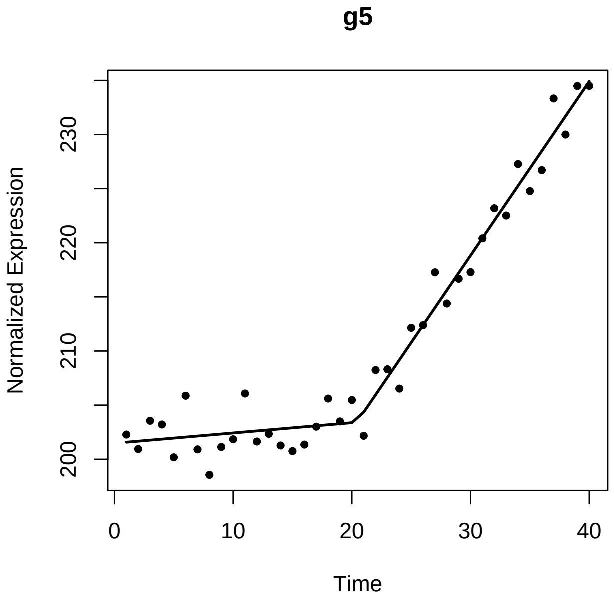
```
par(mfrow=c(3,2))
plot2 <- plotfeature(TrendyExData,
                     Feature.Names = names(res.trend$firstdown)[1:6],
                     Seg.Data = res)
```



To plot the two genes in the third group of genes (first no change):

```
par(mfrow=c(1,2))
plot2 <- plotfeature(TrendyExData,
                    Feature.Names = names(res.trend$firstnochange)[1:2],
                    Seg.Data = res)
```

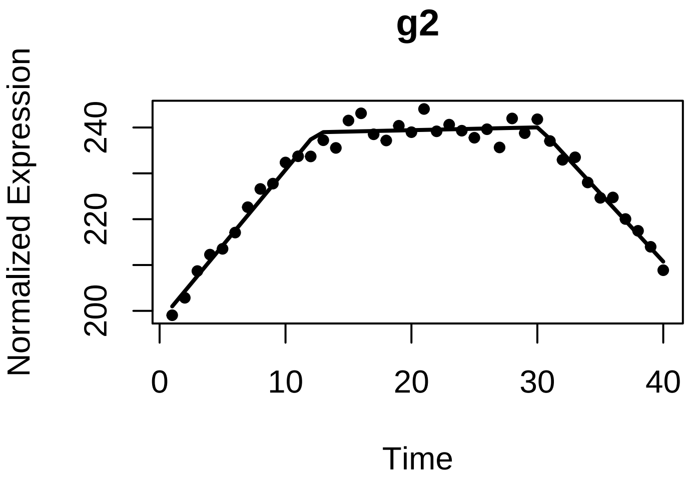




## 3.5 Gene specific estimates

For a given gene of interest, its estimated parameters can be obtained by (using g2 as an example):

```
par(mfrow=c(1,1))
plot2 <- plotfeature(TrendyExData,
                     Feature.Names = "g2",
                     Seg.Data = res)
```



```
print(res.top$bp["g2"]) # break points

## $g2
## breakpoint1 breakpoint2
##    12.47356    30.14908
```

```
print(res.top$radj["g2"]) # adjusted r squared

##        g2
## 0.9710139

print(res.top$slp["g2"]) # fitted slopes of the segments

## $g2
##  slope1  slope2  slope3
##  3.3110  0.0607 -2.9730

print(res.top$slp.pval["g2"]) # p value of each the segment

## $g2
##      pval1      pval2      pval3
## 0.01669386 0.31815050 0.02445599
```

The above printouts show that for gene g2 the optimal number of breakpoints is 2. Two estimated breakpoints are around time-points s12 and s30. The fitted slopes for the 3 segments are 3.31, 0.06 and -2.97, which indicate the trend is up-same-down.

These estimates can also be automatically formatted using the function `formatresults` which can be saved as a .txt. or .csv file. The output currently includes the estimate slope of each segment, the estimated breakpoint, and the adjusted $R^2$.

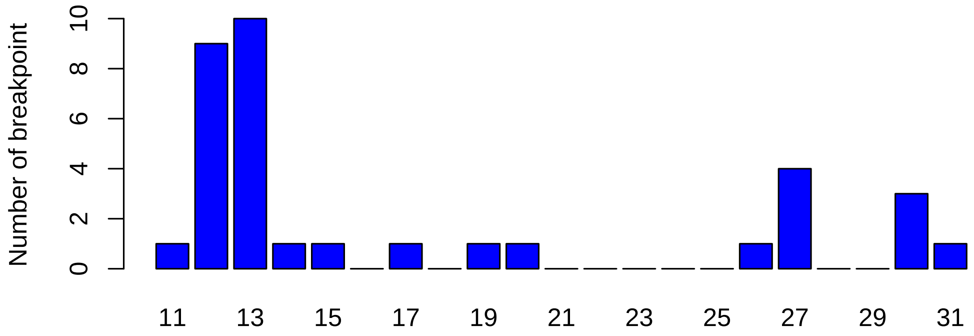
```
trendy.summary <- formatresults(res.top)
head(trendy.summary)

##     feature   slope1    slope2 slope3 breakpoint1 breakpoint2     adjR2
## g3       g3  1.57200 -2.548000     NA    26.97696          NA 0.9787382
## g1       g1 -3.14500  0.001548     NA    13.76450          NA 0.9775005
## g28     g28 -0.01381  1.537000     NA    19.00675          NA 0.9751380
## g20     g20 -3.38100 -0.082460     NA    12.80019          NA 0.9739715
## g15     g15  1.67500 -2.471000     NA    26.31814          NA 0.9729747
## g2       g2  3.31100  0.060700 -2.973    12.47356    30.14908 0.9710139

# write.table(trendy.summary, file="trendy_summary.txt")
```

The NA indicates that g3 does not have a slope3 since it only has one breakpoint (i.e two segments).

## 3.6 Breakpoint distribution over the time course

To calculate number of breakpoints over the time course:

```
res.bp <- bpdist(res.top)
barplot(res.bp, ylab="Number of breakpoint", col="blue")
```



The bar plot indicates that a number of genes have breakpoints around s12 and s13.

# 4 More advanced analysis

## 4.1 Time course with non-uniform sampling

If the samples were collected with different time intervals and the user wants to use the original time (instead of a vector of consecutive numbers), the user may specify it via the T.Vect parameter in the `trendy` function.

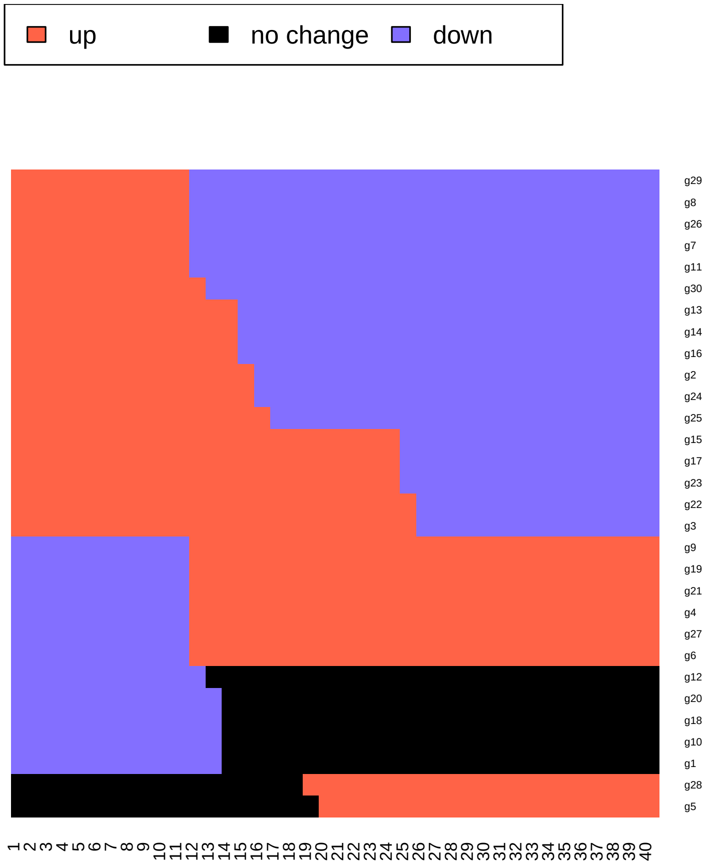
For example, suppose for the example data, the first 30 samples were collected every hour and the other 10 samples were collected every 5 hours. We may define the time vector as:

```
t.v <- c(1:30,seq(31,80,5))
names(t.v) <- colnames(TrendyExData)
print(t.v)
```

```
##  s1  s2  s3  s4  s5  s6  s7  s8  s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21 s22
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22
## s23 s24 s25 s26 s27 s28 s29 s30 s31 s32 s33 s34 s35 s36 s37 s38 s39 s40
##  23  24  25  26  27  28  29  30  31  36  41  46  51  56  61  66  71  76
```

To run Trendy model using the empirical collecting time instead of sample ID (1-40):

```
res2 <- trendy(TrendyExData, T.Vect=t.v, Max.K=2, Cut.Diff=.05)
res.top2 <- toptrendy(res2)
res.trend2 <- trendheatmap(res.top2)
```

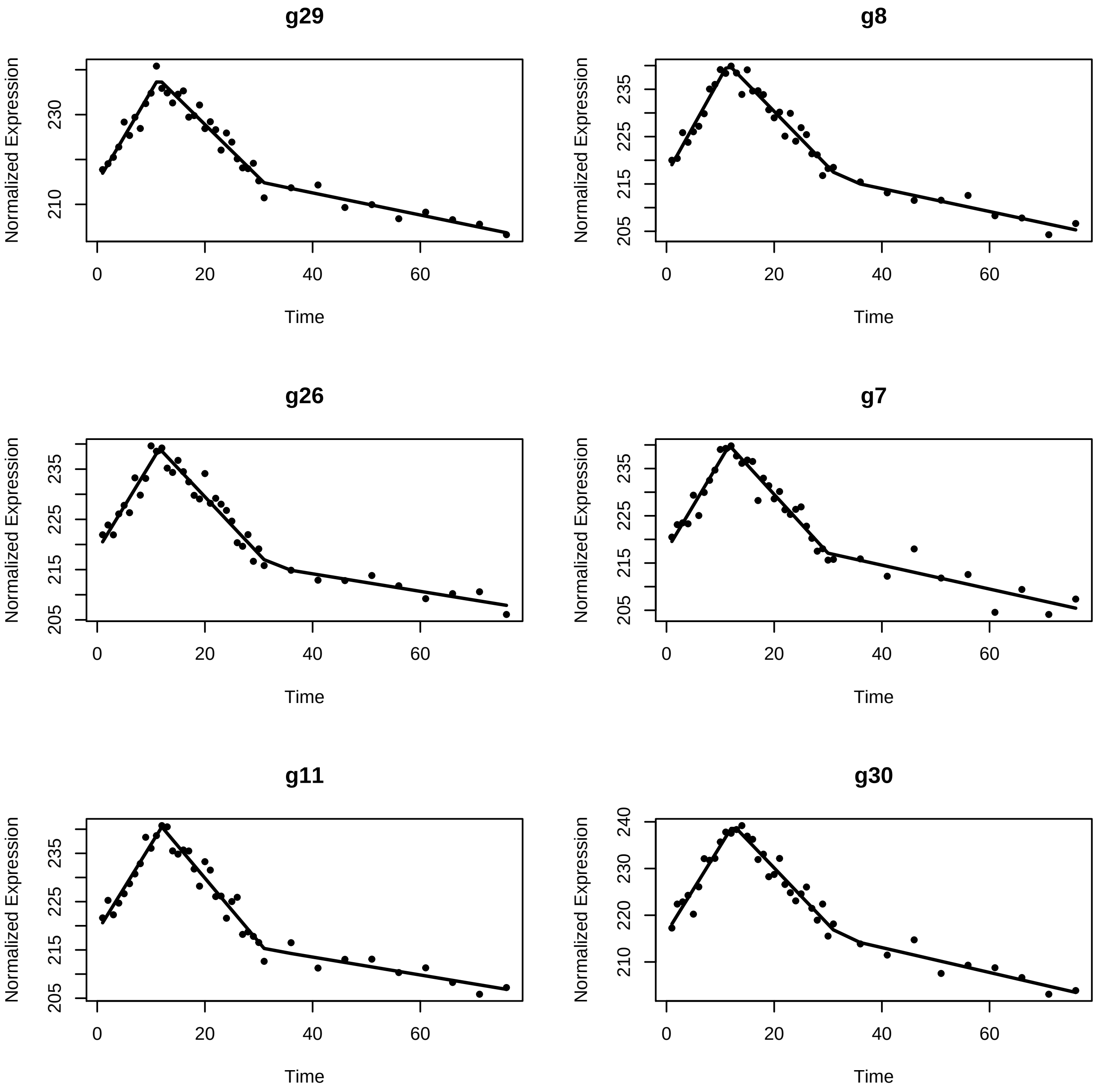


```
str(res.trend2)
```

```
## List of 3
##  $ firstup      : Named num [1:17] 11.4 11.4 11.6 11.7 12 ...
##   ..- attr(*, "names")= chr [1:17] "g29" "g8" "g26" "g7" ...
##  $ firstdown    : Named num [1:11] 11 11.2 11.3 11.4 11.4 ...
##   ..- attr(*, "names")= chr [1:11] "g9" "g19" "g21" "g4" ...
##  $ firstnochange: Named num [1:2] 19 19.5
##   ..- attr(*, "names")= chr [1:2] "g28" "g5"
```

To plot the first six genes that have up-regulated pattern at the beginning of the time course:

```
par(mfrow=c(3,2))
plot1.new <- plotfeature(TrendyExData, T.Vect=t.v,
                        Feature.Names = names(res.trend2$firstup)[1:6],
                        Seg.Data = res2)
```



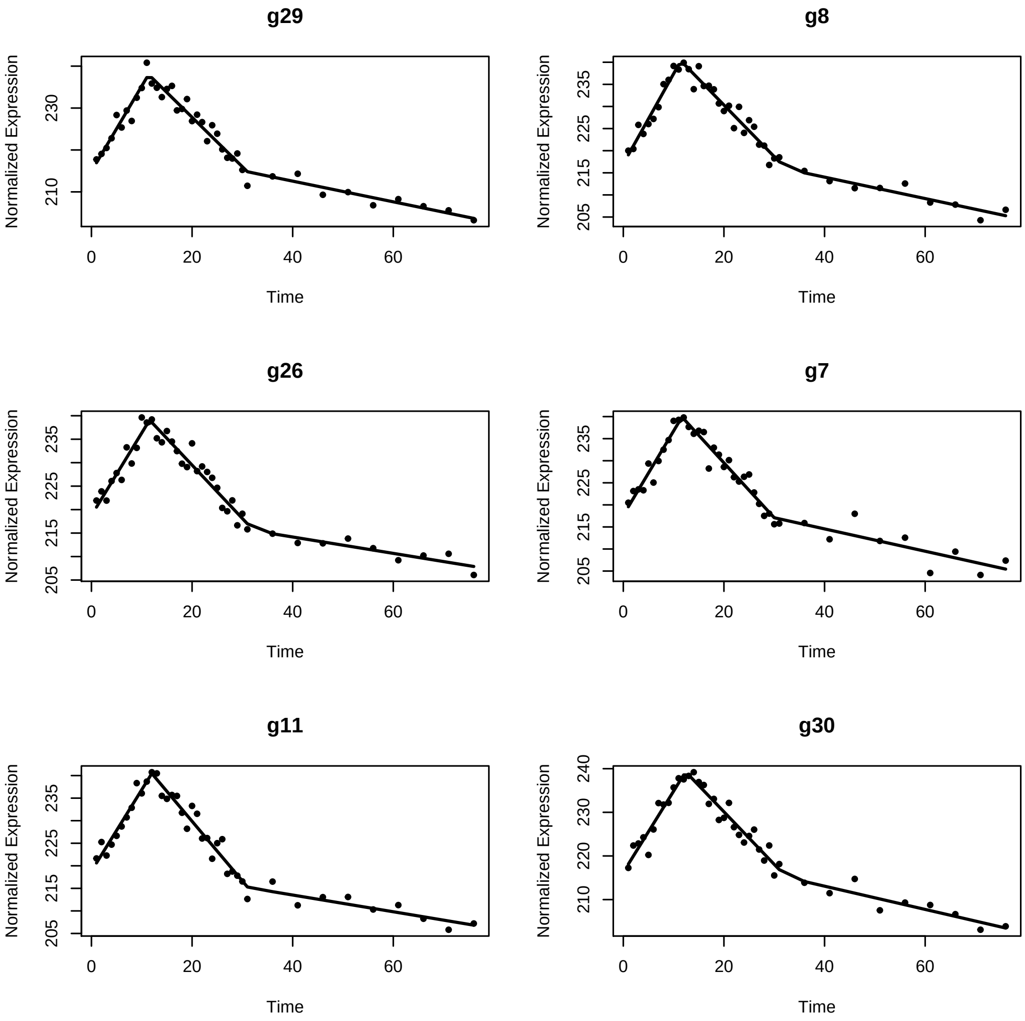
# 5 Extract genes with certain pattern

Genes that have a peak along the time-course will have fitted trend somewhere as "up-down". Genes that are oscillating may have the fitted trend "up-down-up-down". To extract a list of such genes we can use the `extractpattern`:

```
# Genes that peak
pat1 <- extractpattern(res2, Pattern = c("up","down"))
head(pat1)
```

```
##    Gene BreakPoint1
## 3   g29    11.35849
## 1    g8    11.43482
## 8   g26    11.56147
## 12   g7    11.65663
## 2   g11    11.99780
## 4   g30    12.38463
```

```
par(mfrow=c(3,2))
plotPat1 <- plotfeature(TrendyExData, T.Vect=t.v,
                      Feature.Names = pat1$Gene[1:6],
                      Seg.Data = res2)
```

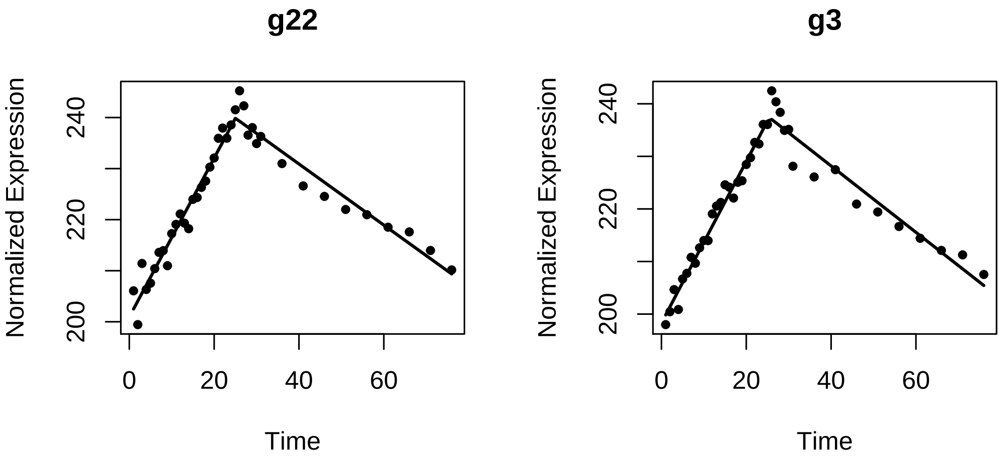


```
# Genes that peak after some time
pat3 <- extractpattern(res2, Pattern = c("up","down"), Delay = 25)
head(pat3)
```

```
##   Gene BreakPoint1
## 2  g22    25.01831
## 1   g3    25.43516
```

```
par(mfrow=c(1,2))
```

```
plotPat3 <- plotfeature(TrendyExData, T.Vect=t.v,
                        Feature.Names = pat3$Gene,
                        Seg.Data = res2)
```



## 6 Additional options

In the `trendy` function, the thresholds $c_{num}$, $c_{diff}$ and $c_{pval}$ can be specified via parameters Min.Num.In.Seg, Cut.Diff and Pval.Cut, respectively.

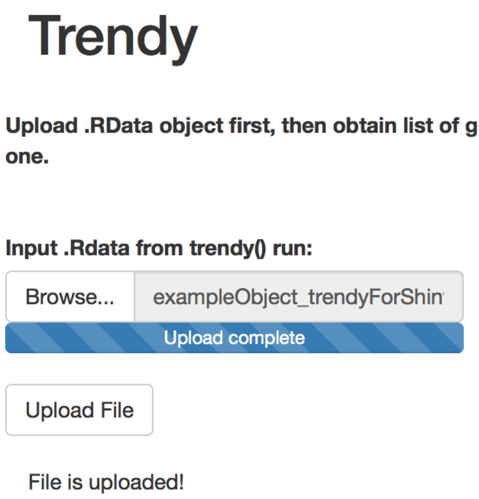
## 7 Trendy shiny app

The Trendy shiny app requires an .RData object output from the `trendy` function, which can be obtained by setting `Save.Object=TRUE`.

```
res <- trendy(TrendyExData, Max.K=2, Save.Object = TRUE, File.Name="exampleObject")
```

Then in R run:

```
library(shiny)
runGitHub('rhondabacher/Trendy')
```



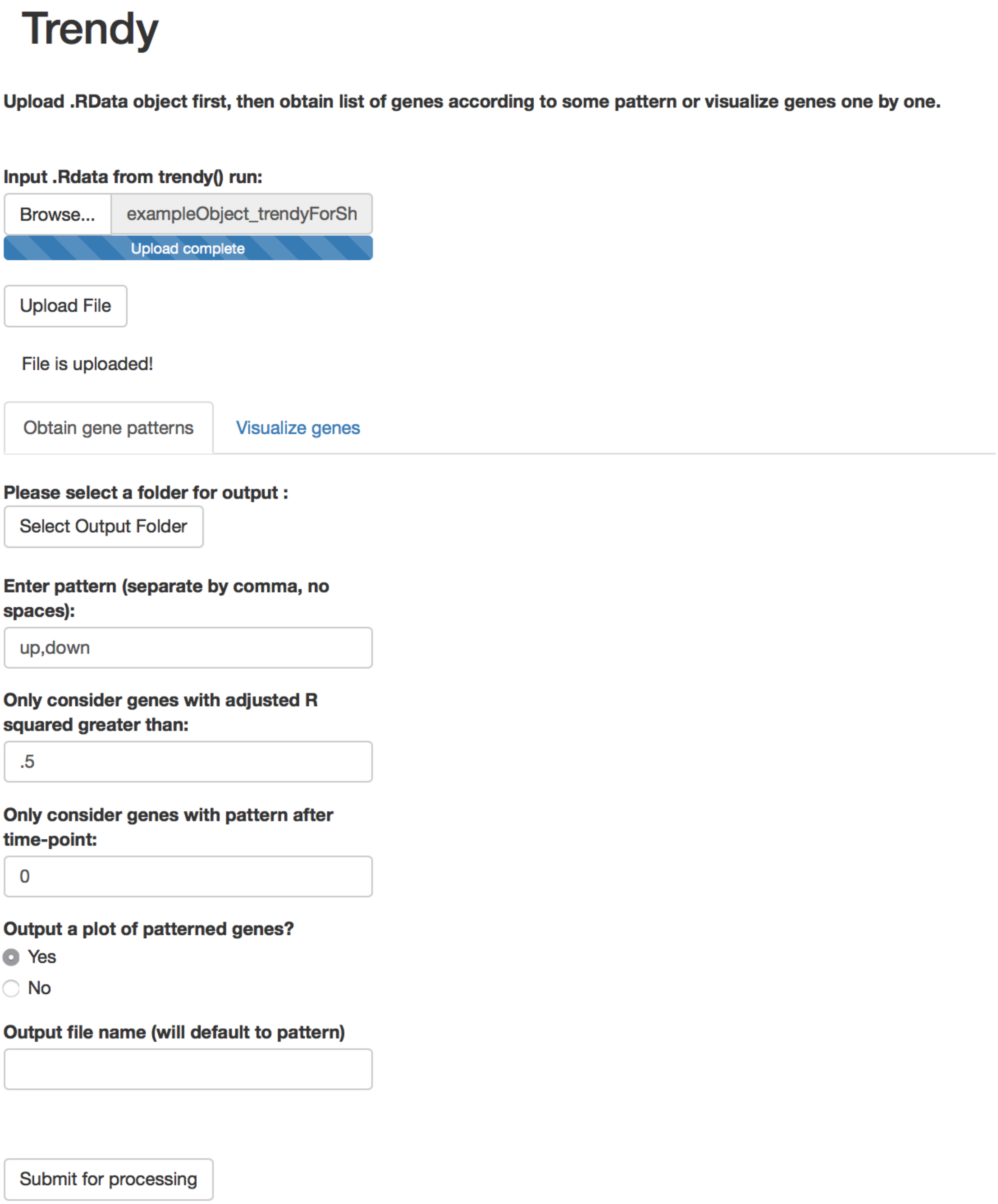
Trendy

Upload .RData object first, then obtain list of genes according to some pattern or visualize genes one by one.

Input .Rdata from trendy() run: exampleObject_trendyForShin

File is uploaded!

Figure 1: Upload shiny object



Trendy

Upload .RData object first, then obtain list of genes according to some pattern or visualize genes one by one.

Input .Rdata from trendy() run: exampleObject_trendyForSh

Upload complete

File is uploaded!

Please select a folder for output :

Enter pattern (separate by comma, no spaces): up,down

Only consider genes with adjusted R squared greater than: .5

Only consider genes with pattern after time-point: 0

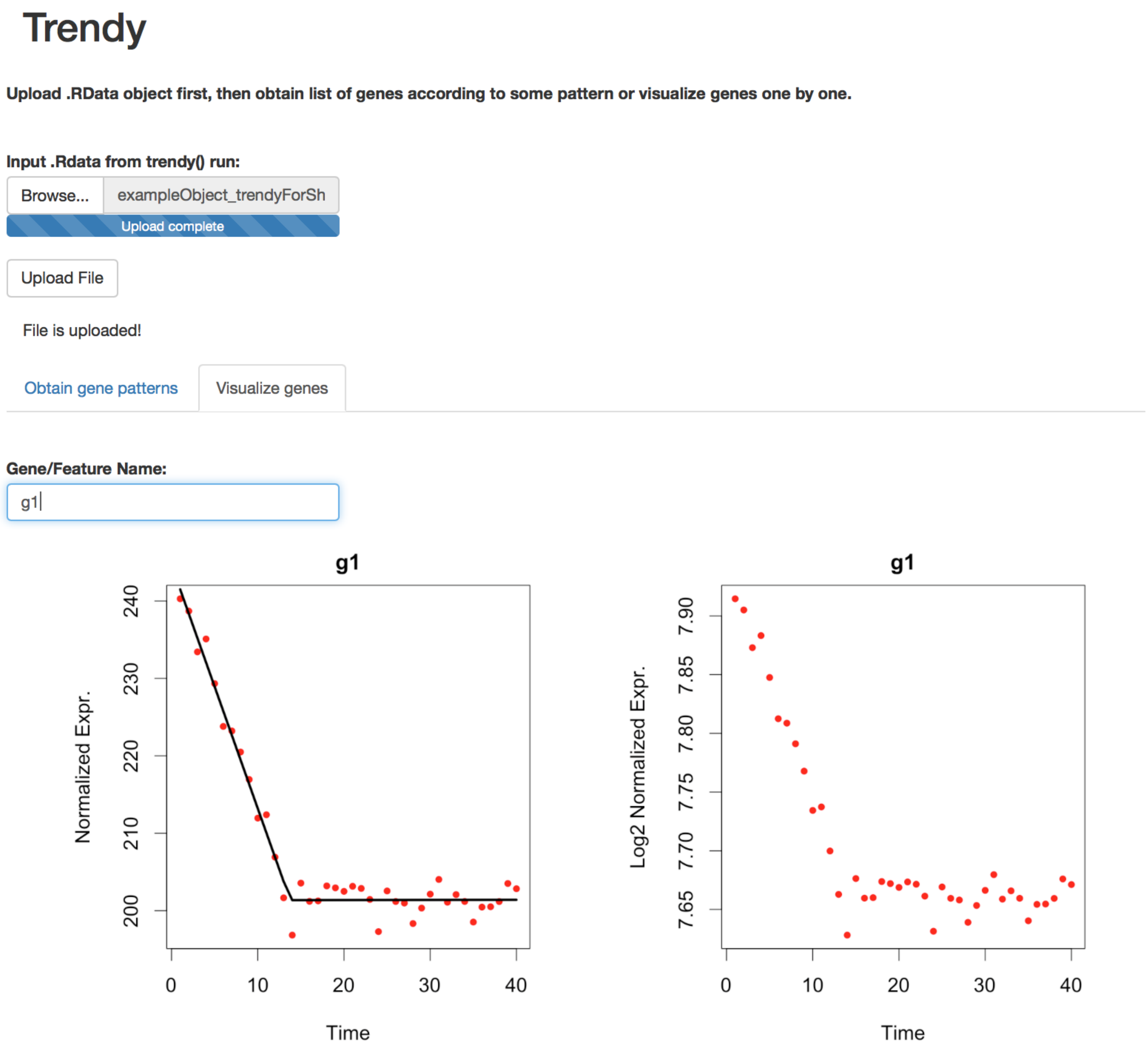
Output a plot of patterned genes?

● Yes

○ No

Output file name (will default to pattern)

Figure 2: Find all genes with a given pattern



Trendy

Upload .RData object first, then obtain list of genes according to some pattern or visualize genes one by one.

Input .Rdata from trendy() run: exampleObject_trendyForSh

Upload complete

File is uploaded!

Gene/Feature Name: g1

Figure 3: Search genes individually

# 8 SessionInfo

```
print(sessionInfo())

## R version 3.4.1 (2017-06-30)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: OS X El Capitan 10.11.6
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] gplots_3.0.1  Trendy_0.99.0 knitr_1.16
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.12       gtools_3.5.0       digest_0.6.12      rprojroot_1.2
##  [5] bitops_1.0-6       backports_1.1.0    magrittr_1.5       evaluate_0.10.1
##  [9] KernSmooth_2.23-15 highr_0.6          stringi_1.1.5      gdata_2.18.0
## [13] rmarkdown_1.6      BiocStyle_2.4.1    tools_3.4.1        stringr_1.2.0
## [17] parallel_3.4.1     yaml_2.1.14        compiler_3.4.1     segmented_0.5-2.1
## [21] caTools_1.17.1     htmltools_0.3.6
```