SegReg: Breakpoint analysis of time course expression data

Rhonda Bacher, Ning Leng, Ron Stewart

Contents

Overview

SegReg is an R package that can be used to perform breakpoint analysis on microarrays or RNA-seq expression data with ordered conditions (e.g. time course, spatial course). For each gene or other features, SegReg 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. SegReg 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).

The model

To illustrate SegReg, here we use time course gene expression data as an example. Note SegReg 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 s is $X_{g,s}$. Denote the total number of genes as G and the total number of samples as S. For each gene, SegReg fits segmented regression models with

varying numbers of breakpoints from 1 to n_k. In which n_k defaults to 3 but can also be specified by the user. The model with k breakpoints can then be written as:

$$M_g^k : X_g \sim \beta_0^k + \beta_1^k * I\{s : s \geq 1, s \leq b_{g,1}^k\} * s + \beta_2^k * I\{s : s \geq b_{g,1}^k + 1, s \leq b_{g,2}^k\} * (s - b_{g,1}^k) +, ...,$$

$$+\beta_{k+1}^k * I\{s : s \geq b_{g,k}^k + 1, s \leq S\} * (s - b_{g,k}^k)$$

For each k, the segmented regression estimates k breakpoints ($b_{g,1}^k$, $b_{g,2}^k$, ... $b_{g,k}^k$) between 1 and S. The segmented regression also estimates $k + 2$ βs. In which β_0^k indicates the intercept, and the other βs indicate slopes for the $k + 1$ segments separated by the k breakpoints. We denote the R^2 for this model as r_g^k.

For a given gene, among the models with varying k, SegeReg picks the optimal number of breakpoints for this gene by comparing the R^2s:

$$\tilde{k_g} = \text{argmax}_{k=1,...,n_k}(r_g^k)$$

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 of one segments having less than c_{num} samples, or $r_g^{\tilde{k}} - r_g^{\tilde{k}-1} < c_{diff}$. The thresholds c_{num} and c_{diff} can be specified by the user; defaults are 5 and 0.1, respectively.

Then the gene specific adjusted R^2 and breakpoint estimates are then obtained from this optimal model: $r_g = r_g^{\tilde{k_g}}$; $(\beta_{g,0}, ..., \beta_{g,\tilde{k_g}+1}) = (\beta_{g,0}^{\tilde{k_g}}, ..., \beta_{g,\tilde{k_g}+1}^{\tilde{k_g}})$ and $(b_{g,1}, ..., b_{g,\tilde{k_g}}) = (b_{g,1}^{\tilde{k_g}}, ..., b_{g,\tilde{k_g}}^{\tilde{k_g}})$. Among all genes, the top dynamic genes are defined as those whose optimal model has high adjusted R^2s.

To compute the breakpoint distribution over the time course, SegReg calculates:

$$N_s = \sum_{g=1,...,G} \sum_{j=1,...,\tilde{k_g}} I\{b_{g,j} = s\}$$

The time points with hign N_s might be considered as time points with a large amount of expression changes.

SegReg also outputs fitted trend of each gene. For samples between the j^{th} and $j + 1^{th}$ breakpoint for a given gene, if the t statistic of $\beta_{g,j+1}$ 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 coefficient of $\beta_{g,j+1}$. The c_pval defaults to 0.1, but can also be specified by the user.

Installation

Install via GitHub

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

To install, type the following codes in R:

```
install.packages("devtools")
```

```
library(devtools)
```

```
install_github("rhondabacher/SegReg/package/SegReg")
```

Install locally

Install packages segmented and gplots:

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

Download the SegReg package from:

https://github.com/rhondabacher/SegReg/tree/master/package

And install the package locally.

Load the package

To load the SegReg package:

```
library(SegReg)
```

Analysis

Input

The input data should be a $G - by - S$ matrix containing the expression values for each gene and each sample, where G is the number of genes and S 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() functions 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 SegRegExData is a simulated data matrix containing 50 rows of genes and 40 columns of samples.

```
data(SegRegExData)
str(SegRegExData)
```

```
##  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" ...
```

Run segmented regressions

The segreg() function is used to run gene specific segmented regressions. Here we want to only consider up to 2 breakpoints for each gene. To do so we may specify maxk = 2:

```
res <- segreg(SegRegExData, maxk = 2)
```

```
## SegReg has finished running and the the output object for shiny has been output to /Users/rbacher/Des
```

```
res.top <- topsegreg(res) # default adjusted R squared cutoff is 0.5
res.top$radj
```

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

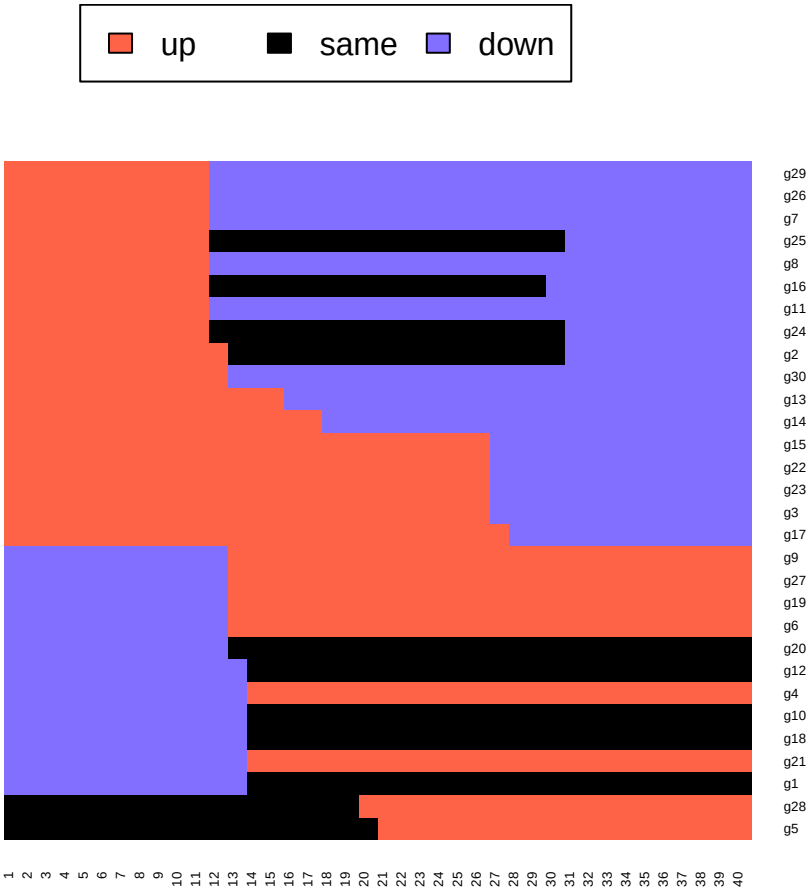
The topsegreg() function may be used to extract top dynamic genes. By default, topsegreg() will extract genes whose adjusted R^2 is greater or equal to 0.5. To change this threshold, a user may specify the r.cut parameter in topsegreg() function. res.top$radj gives $r_{g,\text{adj}}$ of the top dynamic genes, sorted decreasingly by $r_{g,\text{adj}}$.

By default the segreg() function only consider genes whose mean expression is greater than 10. To use another threshold, a user may specify the parameter meancut in the segreg() function.

Visualize trends of the top dynamic genes

res.top$id.sign gives trend specification of the top genes. Function trendheatmap() can be used to display these trends:

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



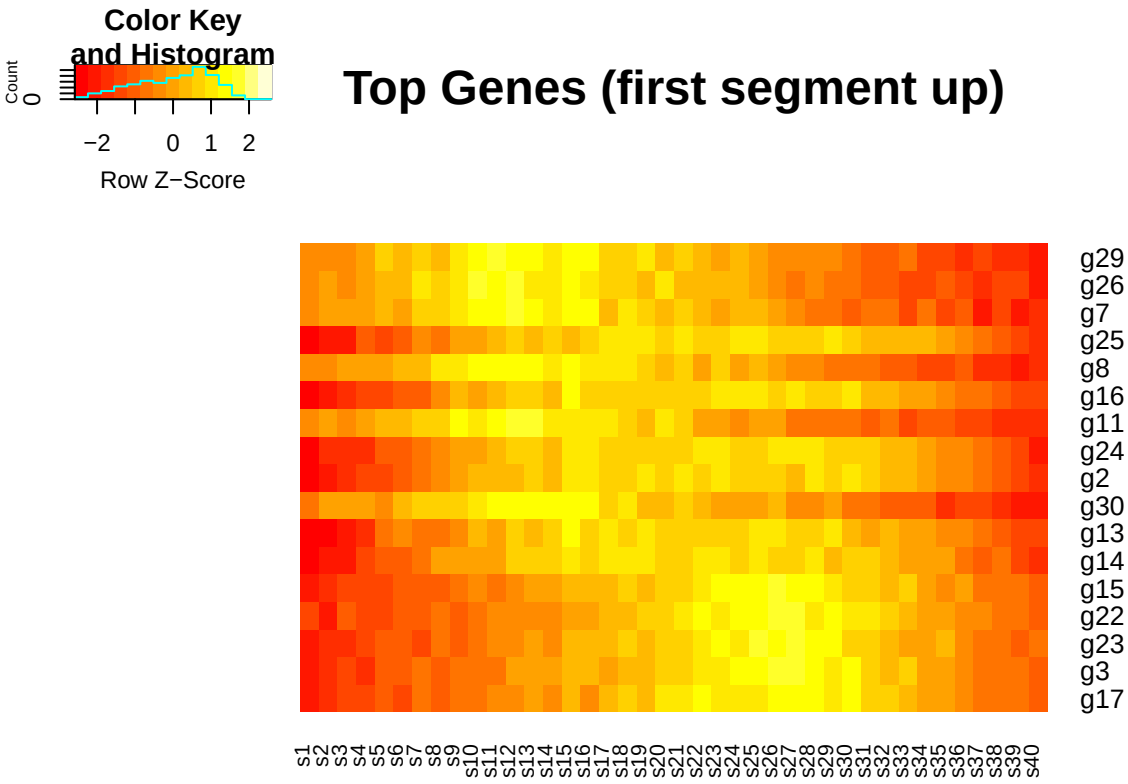
```
str(res.trend)
```

```
## 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 classify 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.

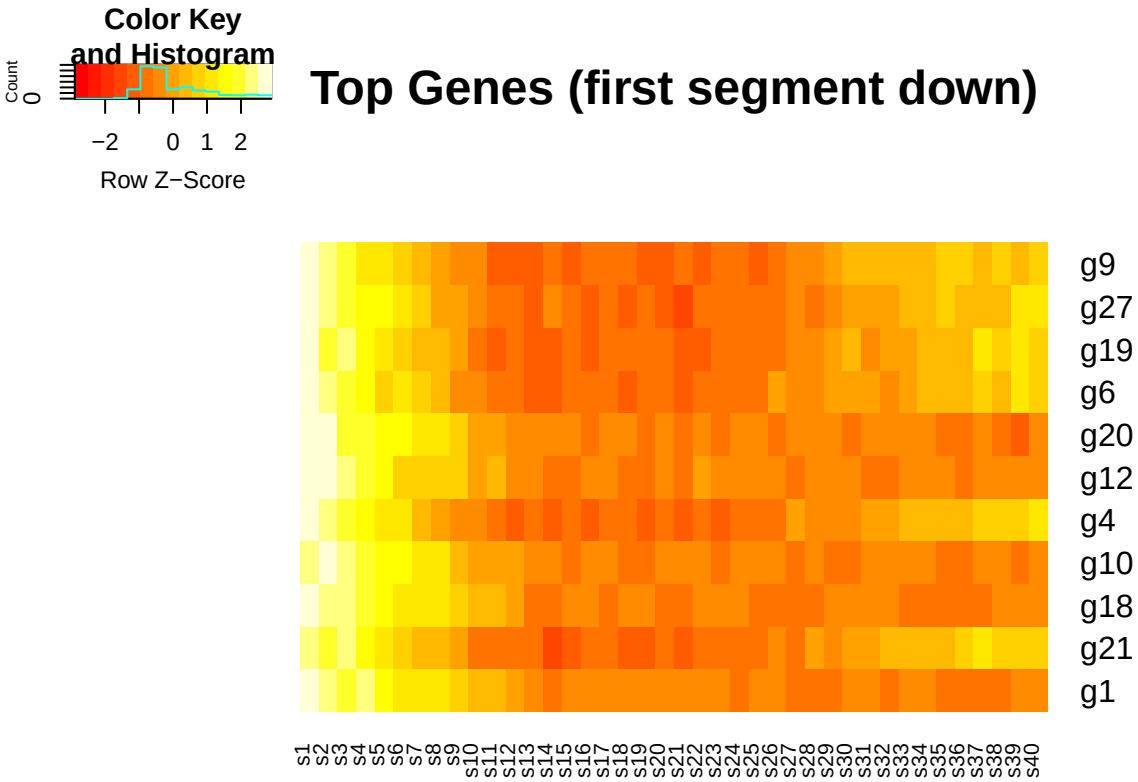
To generate expression heatmap of the first group of genes (first segment goes up):

```
heatmap.2(SegRegExData[names(res.trend$firstup),],trace="none", Rowv=F,Colv=F, lwd=2,
                    scale="row", main="Top Genes (first segment up)")
```



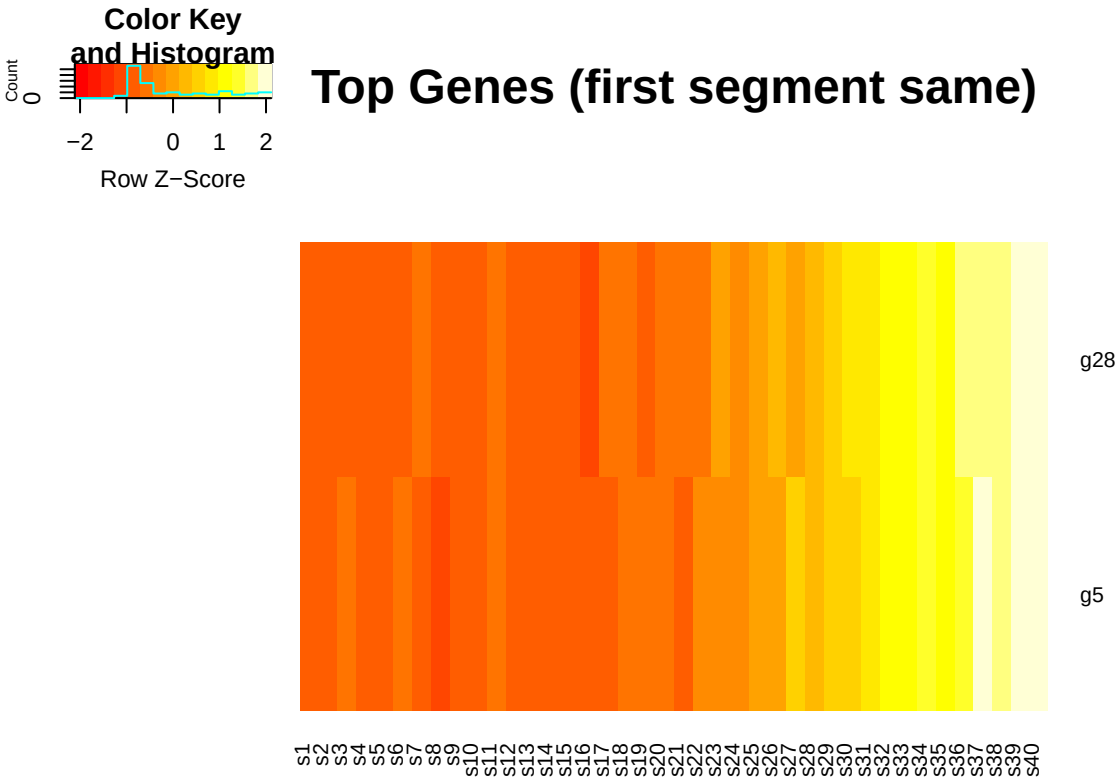
Similarly, to generate expression heatmap of the second group of genes (first segment goes down):

```
heatmap.2(SegRegExData[names(res.trend$firstdown),],trace="none", Rowv=F,Colv=F,
                              scale="row", main="Top Genes (first segment down)")
```



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

```
heatmap.2(SegRegExData[names(res.trend$firstnochange),],trace="none", Rowv=F,Colv=F,
                                        scale="row", main="Top Genes (first segment same)",
                                                    cexRow=.8)
```

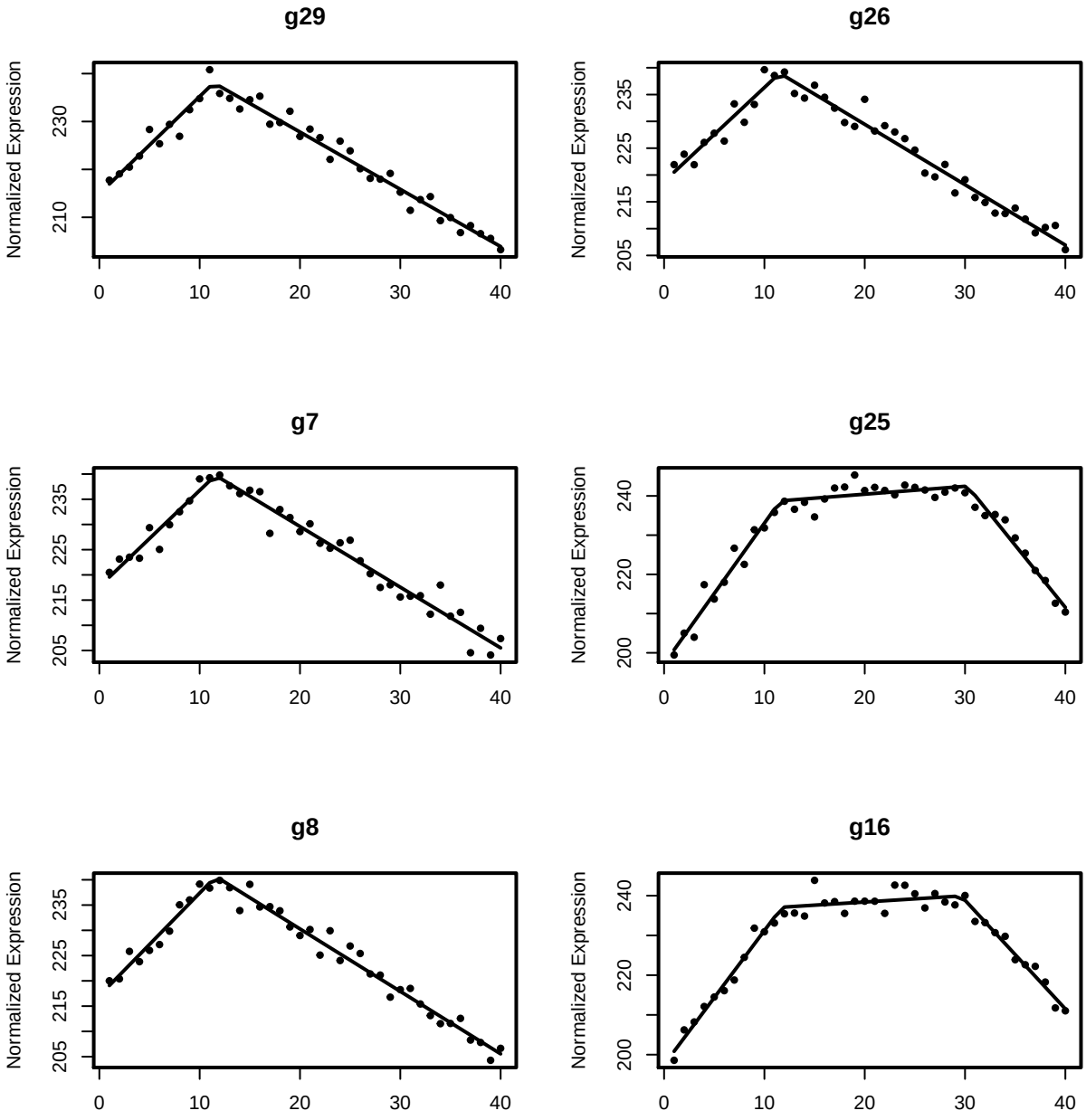


Visualize individual genes

The plotmarker() function may be used to plot expression of individual genes and the fitted lines.

For example, to plot the top 6 genes in the first group of genes (first go up):

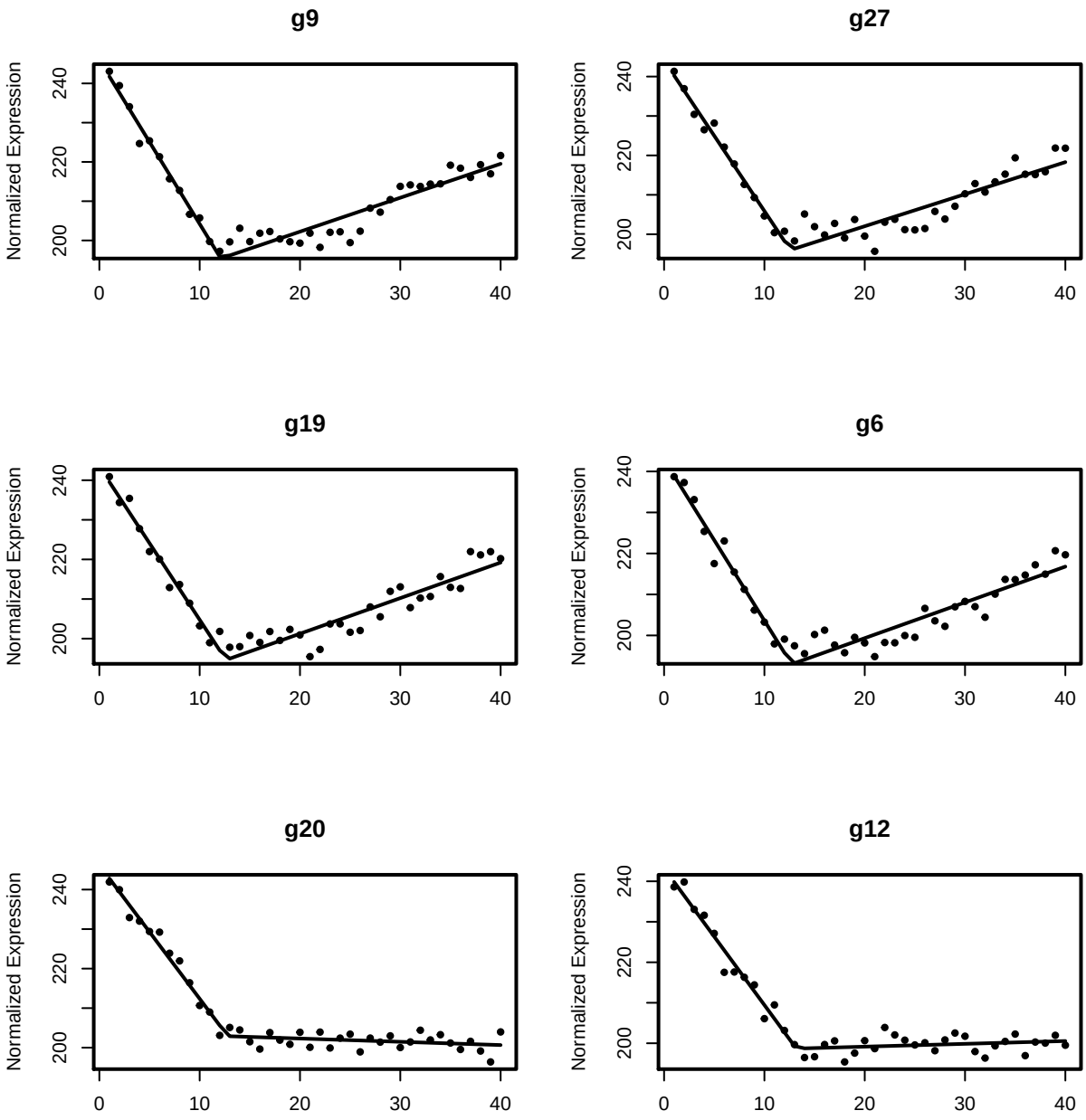
```
plot1 <- plotmarker(SegRegExData,listfeatures=names(res.trend$firstup)[1:6],fittedreg=res)
```



The input of function plotmarker() requires the expression data and a list of genes of interest. The parameter fittedres in function plotmarker() takes segreg() fitted results. If it is not specified, the function plotmarker() will run SegReg model on the genes of interest before plotting. Specifying fitted results obtained from previous steps will save time by avoiding fitting the models again.

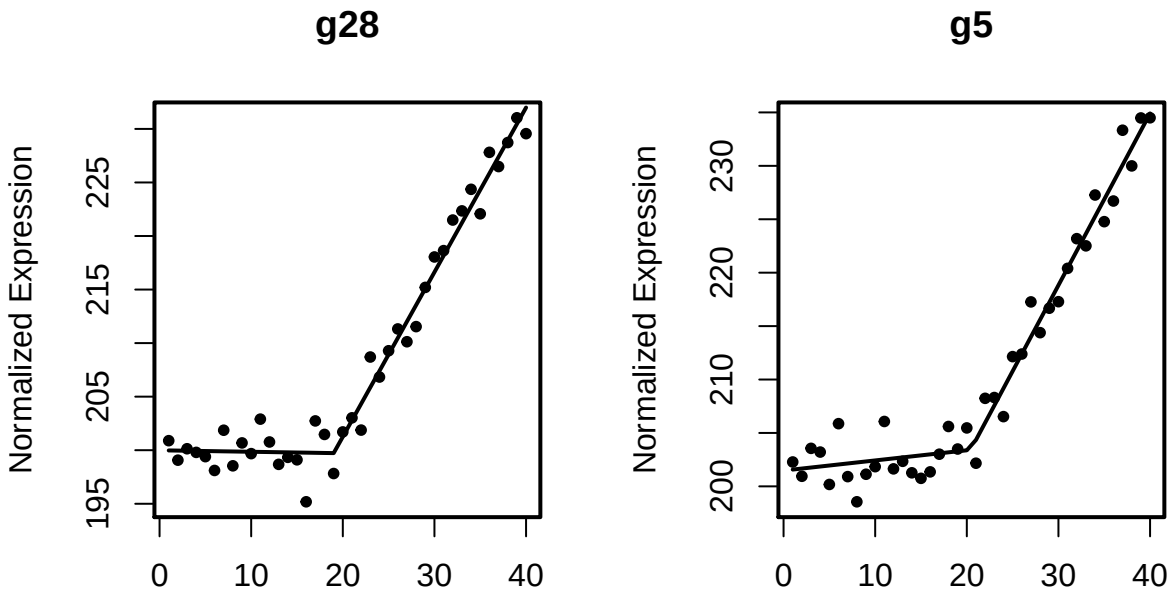
Similarly, to plot the top 6 genes in the second group of genes (first go down):

```
plot2 <- plotmarker(SegRegExData,listfeatures=names(res.trend$firstdown)[1:6],fittedreg=res)
```



To plot the 2 genes in the third group of genes (first no change):

```
plot2 <- plotmarker(SegRegExData,listfeatures=names(res.trend$firstnochange)[1:2],fittedreg=res,par.para
```



Gene specific estimates

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

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

```
## $g2
## psi1.t.use psi2.t.use
##   12.47356   30.14908
```

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

```
##        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
```

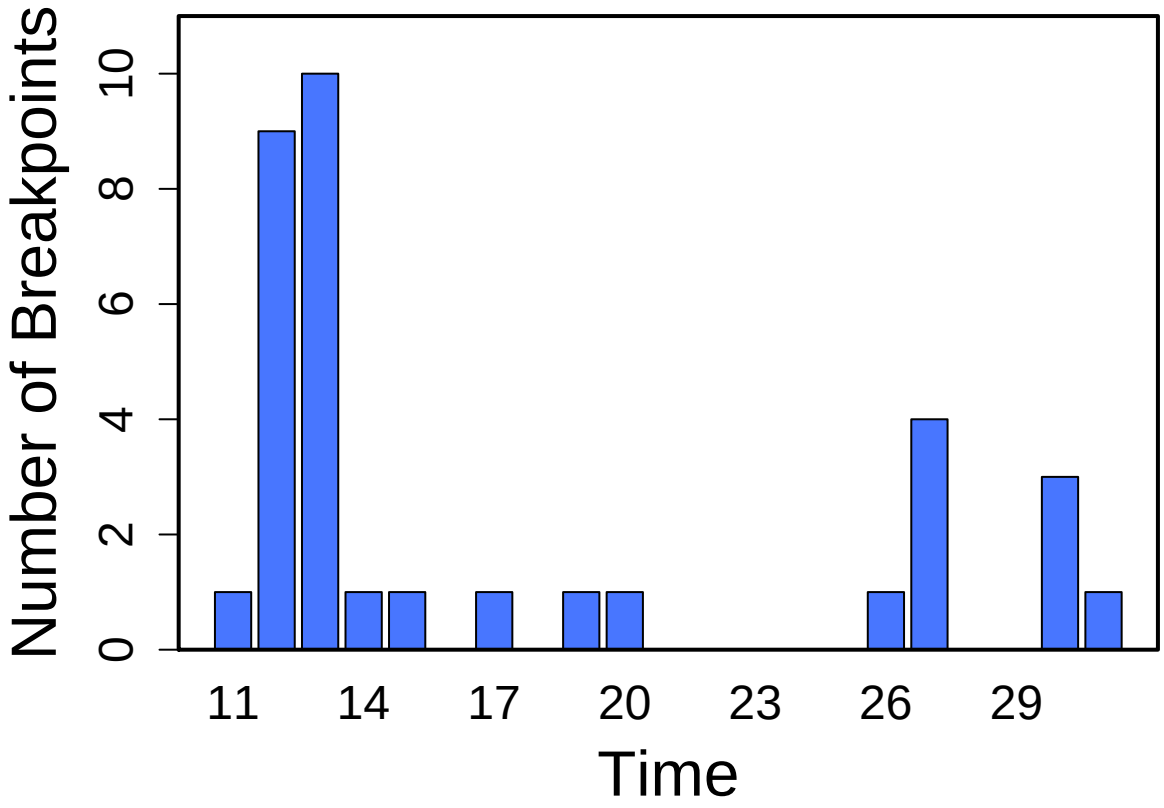
```
## $g2
##     slope1     slope2     slope3
## 0.01669386 0.31815050 0.02445599
```

The above printouts show that for gene g2, the optimal number of breakpoints is 2. Two estimated breakpoints are close to s12 and s30. The fitted slopes for the 3 segments are 3.31, 0.06 and -2.97.

Breakpoint distribution over the time course

To calculate number of breakpoints over the time course:

```
res.bp <- bpdist(res.top)
```



The bar plot indicates that many genes have breakpoint around s12 and s13.

More advanced analysis

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 segreg() 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(SegRegExData)
print(t.v)
```

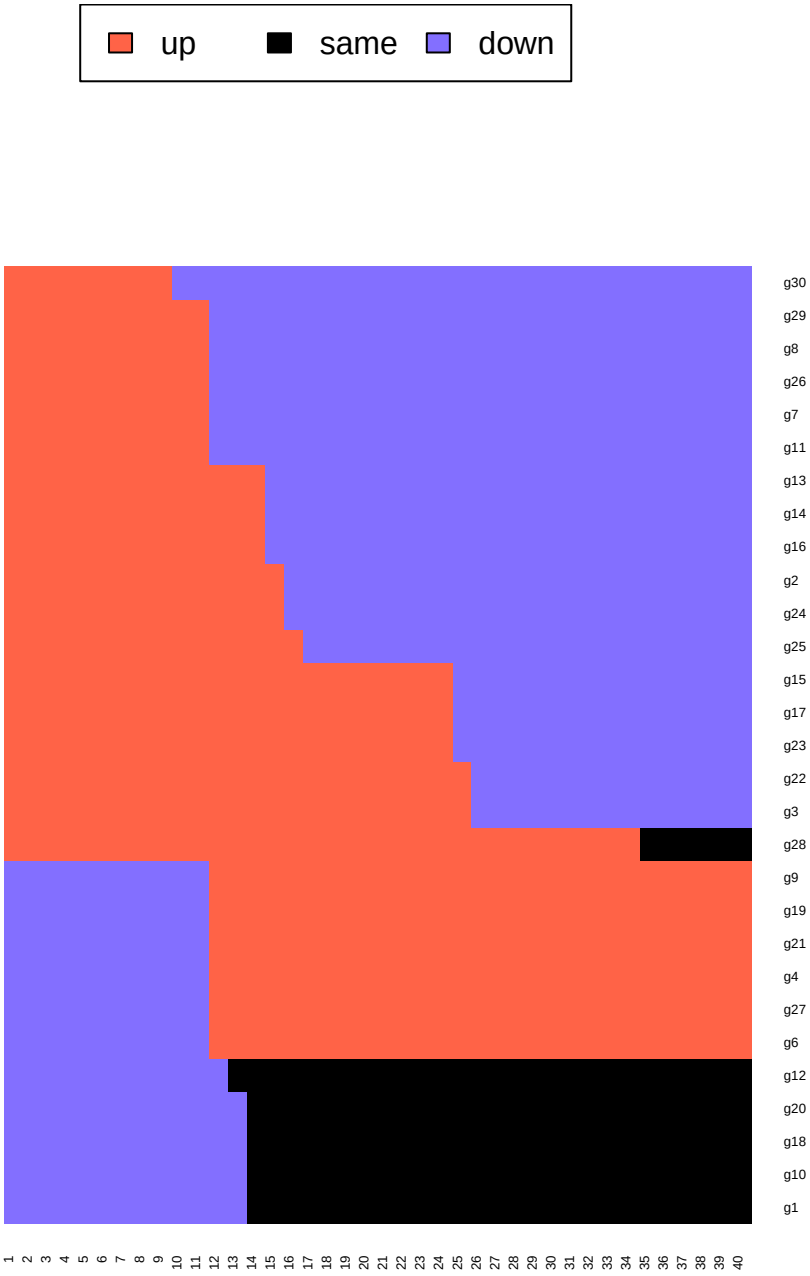
```
##  s1  s2  s3  s4  s5  s6  s7  s8  s9 s10 s11 s12 s13 s14 s15 s16 s17 s18
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
## s19 s20 s21 s22 s23 s24 s25 s26 s27 s28 s29 s30 s31 s32 s33 s34 s35 s36
##  19  20  21  22  23  24  25  26  27  28  29  30  31  36  41  46  51 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

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

```
res2 <- segreg(SegRegExData, t.vect=t.v, maxk=2)
```

```
## SegReg has finished running and the the output object for shiny has been output to /Users/rbacher/Des
```

```
res.top2 <- topsegreg(res2)
res.trend2 <- trendheatmap(res.top2, showplot=TRUE, savePDF = FALSE)
```



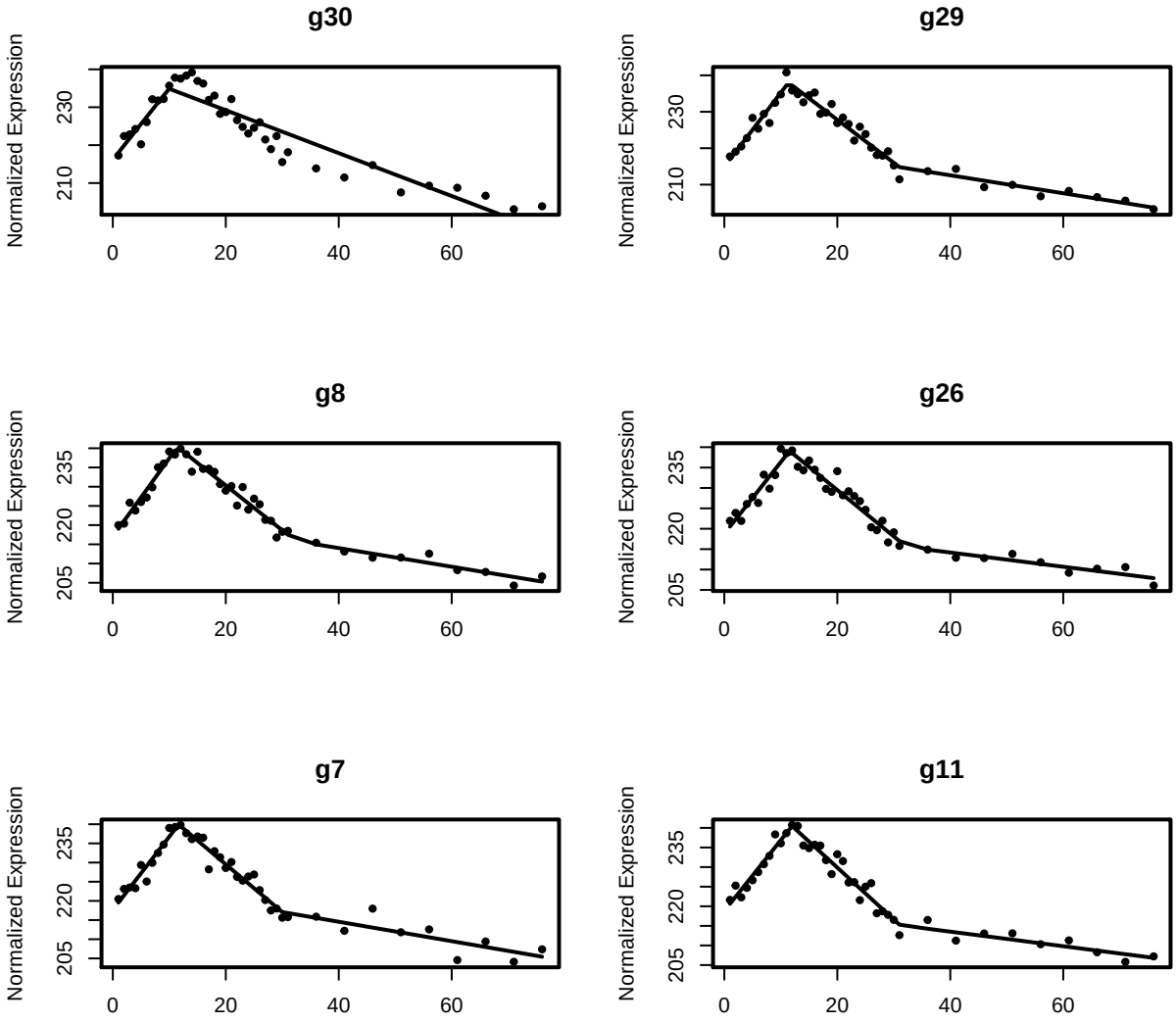

```
str(res.trend2)
```

```
## List of 3
##  $ firstup      : Named num [1:18] 9.95 11.36 11.43 11.56 11.66 ...
##   ..- attr(*, "names")= chr [1:18] "g30" "g29" "g8" "g26" ...
##  $ 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(0)
##   ..- attr(*, "names")= chr(0)
```

To plot the first 6 genes that have up-regulated pattern at the beginning of the time course, by showing

empirical time at x axis:

```
plot1.new <- plotmarker(SegRegExData,t.vect=t.v,listfeatures=names(res.trend2$firstup)[1:6],fittedreg=r
```



Additional options

In segreg() function, the thresholds c_{num}, c_{diff} and c_{pval} can be specified via parameters min.num.in.seg, cutdiff and pvalcut.

Using the Shiny app

This R package also contains a shiny app, which takes as input the object created while running the segreg() function (the default name of this object is rdata_object_forShiny). # `{r, cache=TRUE} # data(rdata_object_forShiny)`

The shiny app can be used to interactively view genes one at a time. After uploading the RData object saved during segreg(), entering a gene name will produce a plot of the normalized gene expression on the raw and log2 scales. The fitted trend is displayed on the raw scale data. Only genes which passed the mean filter in segreg() are searchable.

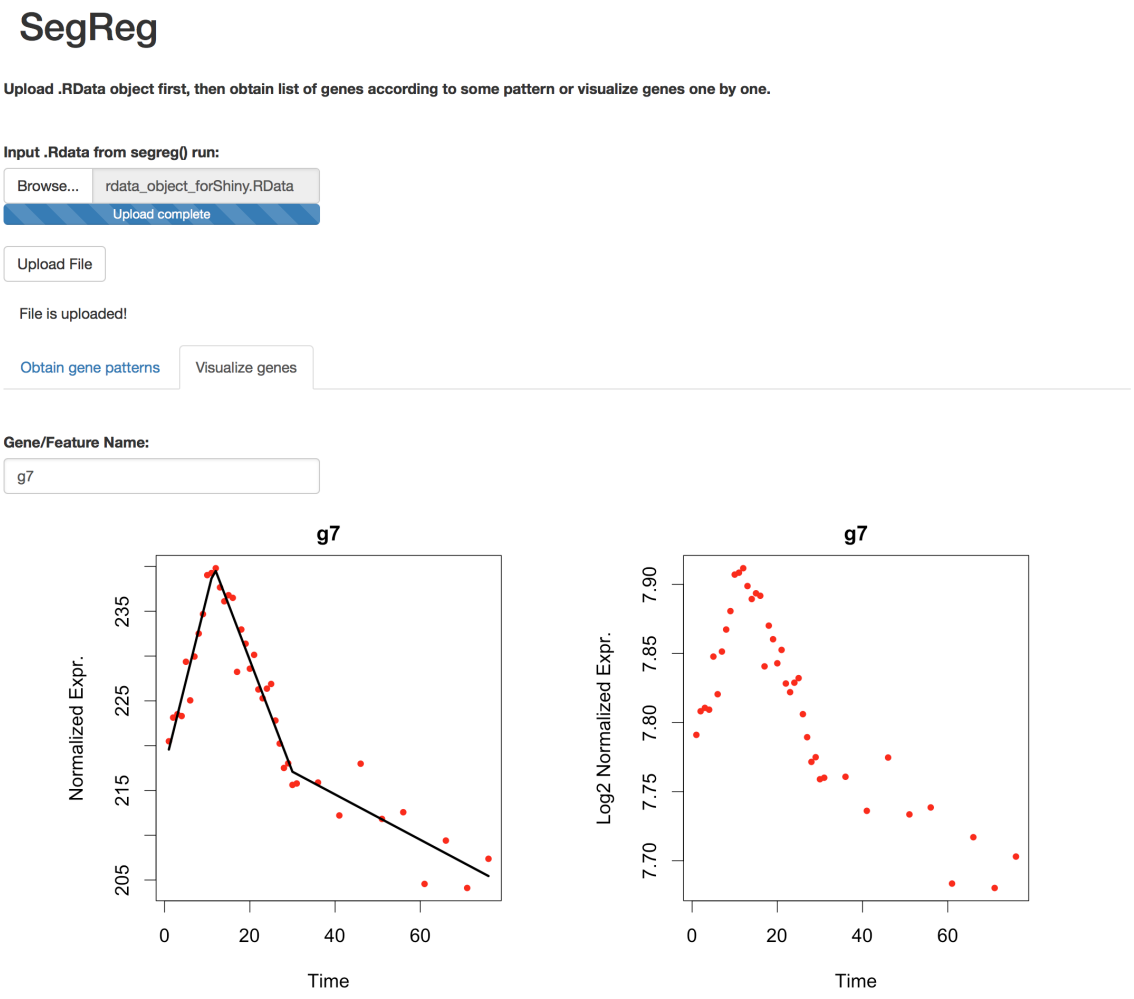


Figure 1: Vizualize genes individually.

Users may also identify genes which follow a given pattern. A pattern such as "up,down" indicates a gene is peaking during the timecourse.

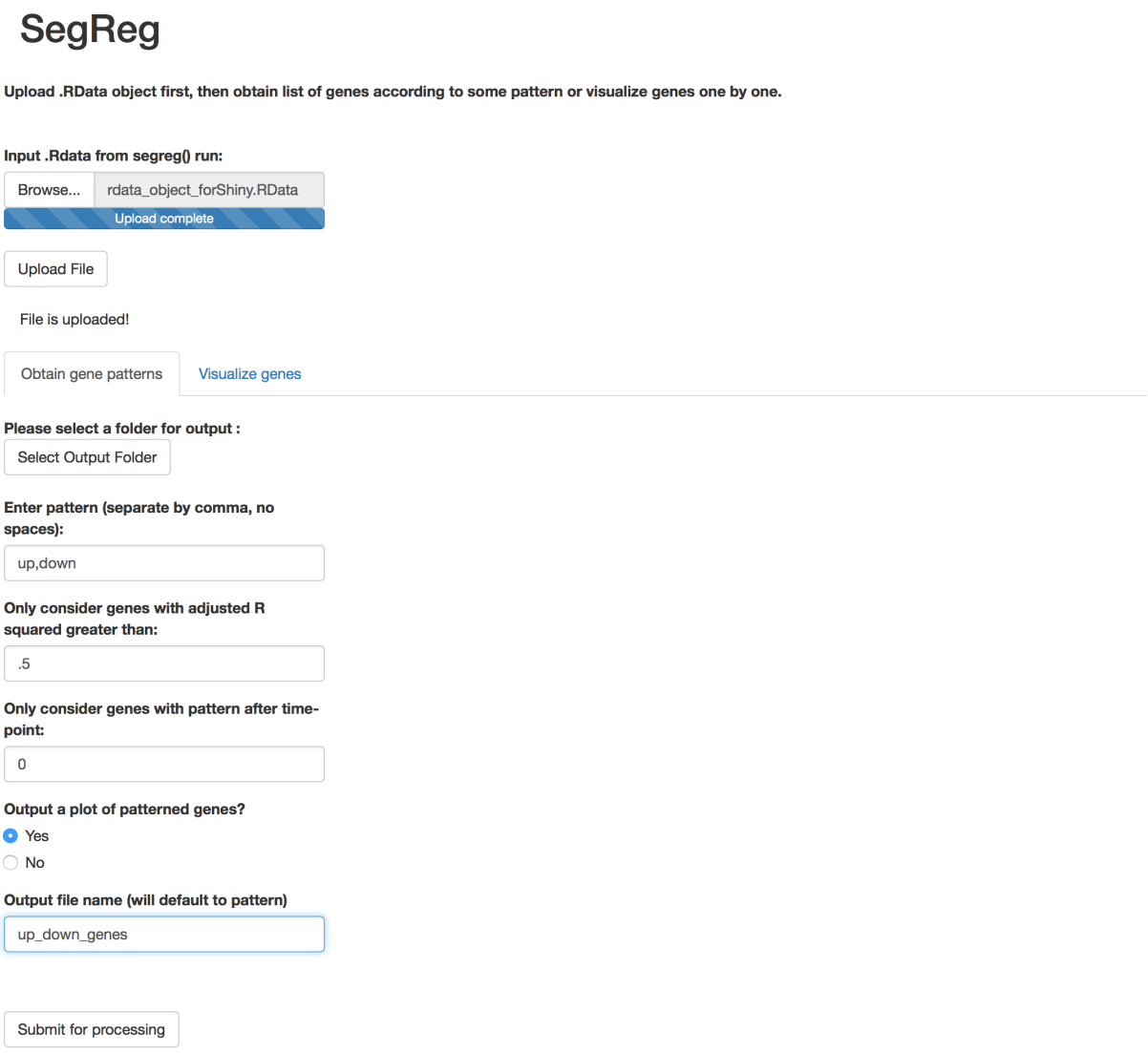


Figure 2: Extract list of genes having a specific pattern and output a plot and list of genes.

SessionInfo

```
print(sessionInfo())
```

```
## R version 3.3.2 (2016-10-31)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X El Capitan 10.11.6
##
## 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
##
## loaded via a namespace (and not attached):
##  [1] backports_1.0.4 magrittr_1.5    rprojroot_1.1   tools_3.3.2
##  [5] htmltools_0.3.5 yaml_2.1.14     Rcpp_0.12.8     stringi_1.1.2
##  [9] rmarkdown_1.2   knitr_1.15.1    stringr_1.1.0   digest_0.6.11
## [13] evaluate_0.10
```