PLRS
Piecewise Linear Regression Splines for the association between DNA copy number and gene expression

Gwenaël G.R. Leday

g.g.r.leday@vu.nl

Department of Mathematics, VU University
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

1 Introduction

The PLRS package implements the methodology described by [**?**] for the joint analysis of DNA copy number and mRNA expression. The framework employs piecewise linear regression splines (PLRS), a broad class of interpretable models, to model *cis*-relationships between the two molecular levels. In the present vignette, we provide guidance for:

1. The analysis of a single DNA-mRNA relationship
    - model specification and fitting,
    - selection of an appropriate model,
    - testing the strength of the association and
    - obtaining uniform confidence bands.
2. The analysis of multiple DNA-mRNA relationships

We first provide an short explanation on data preparation and introduce the breast cancer data used to illustrate this vignette.

2 Data preparation

Data preparation consists in (1) preprocessing gene expression and aCGH data, and (2) matching their features. In the following, we give a brief overview on how to proceed with R.

2.1 Creating an "ExpressionSet" object

We assume that the user is familiar with microarray gene expression data and knows how to normalize these with appropriate R/Bioconductor packages. Consider the matrix (features $\times$ samples) of normalized gene expression `exprMat` and associated *data.frame* `exprAnn` containing annotations of features (such as chromosome number, start bp, end bp, symbol, ...). One can create an "ExpressionSet" object as follows:

```
GE <- new("ExpressionSet", exprs=exprMat)
fData(GE) <- data.frame(Chr=exprAnn$Chr, Start=exprAnn$Start,
End=exprAnn$End, Symbol=exprAnn$Symbol)
rownames(fData(verhaakGEraw)) <- exprAnn$ProbeID
```

We refer the user to the package `Biobase` for a more detail and complete description of this class of objects.

2.2 Preprocessing aCGH data

Here, we shortly illustrate steps for preprocessing aCGH data. Of course, this is only one way to proceed and different algorithms can be employed for different steps. In what follows, the user can find supplementary information within documentation of packages `CGHcall` and `sigaR`.

2.2.1 Create an "cghRaw" object

Consider a *data.frame* `cgh` where the first three columns contains the chromosome number, start and end position in bp (respectively), and where the following columns are the $\log_2$-ratios for each sample. Then, the raw aCGH data can be organized into an "cghRaw" object as follows:

```
# Raw object
cghraw <- make_cghRaw(cgh)
cghraw

# Some available methods
copynumber(cghraw) chromosomes(cghraw)
bpstart(cghraw)
bpend(cghraw)
```

Now, it is possible to use the R functions from package `CGHcall`.

2.2.2 Multi-step preprocessing of aCGH data

The preprocessing of copy number data usually consists in three steps: normalization, segmentation and calling (see [**?**]). The normalization aims to remove technical variability and make samples comparable. This step outputs normalized $\log_2$-values.

```
# Imputation of missing values
cghprepro <- preprocess(cghraw, maxmiss = 30, nchrom = 22)

# Mode normalization
cghnorm <- normalize(cghprepro, method = "median")
```

The segmentation consists in partitioning the genome of each sample into segments of constant $\log_2$-values. A common method to do so is the algorithm of [**?**].

```
# Segmentation using the CBS method of Olshen
cghseg <- segmentData(cghnorm, method = "DNAcopy")
```

Finally, calling attributes an aberration state to each segment. CGHcall is based on mixture models and classifies each segment into 3 (loss, normal and gain), 4 (loss, normal, gain and amplification) or 5 (double losses, single loss, normal, gain and amplification) types of genomic aberrations. The number of aberration states is specified via argument `nclass`.

```
# Calling using CGHcall
res <- CGHcall(cghseg, nclass = 4, ncpus=8)
cghcall <- ExpandCGHcall(res, cghseg)
save(cghcall, file="cghcall.RData")
```

The final object `cghcall` contains all data from previous steps. Here are some methods to access them:

```
# Some methods
norm <- copynumber(cghcall)
seg <- segmented(cghcall)
cal <- calls(cghcall)

# Membership probabilities associated with calls
ploss <- probloss(cghcall)
pnorm <- probnorm(cghcall)
pgain <- probgain(cghcall)
pamp <- probamp(cghcall)
```

2.3 Matching features of gene expression and copy number data

Matching of features is accomplished with package `sigaR`. There exists various methods of matching and we refer the interested reader to [**?**] for their introduction. Below we provide R code for the *DistanceAny* method with a window of 10,000 bp:

```
# distanceAny matching:
matchedIDs <- matchAnn2Ann(fData(cghcall)[,1], fData(cghcall)[,2],
fData(cghcall)[,3], fData(GE)[,1], fData(GE)[,2], fData(GE)[,3],
method = "distance", ncpus = 8, maxDist = 10000)

# add offset to distances (avoids infinitely large weights)
matchedIDs <- lapply(matchedIDs, function(Z, offset) Z[,3] <- Z[,3] + offset; return(Z)
, offset=1)

# extract ids for object subsetting
matchedIDsGE <- lapply(matchedIDs, function(Z) return(Z[, -2, drop=FALSE]) )
matchedIDsCN <- lapply(matchedIDs, function(Z) return(Z[, -1, drop=FALSE]) )

# generate matched objects
GEdata <- ExpressionSet2weightedSubset(GE, matchedIDsGE, 1, 2, 3, ncpus = 8)
CNdata <- cghCall2weightedSubset(cghcall, matchedIDsCN, 1, 2, 3, ncpus = 8)

# save matched data
save(GEdata, CNdata, file="GECNmatched_distanceAny10000.RData")
```

R code for other methods can be found in documentation of package `sigaR`.

3 Breast cancer data

We use the breast cancer data of [**?**] that are available from the Bioconductor package "Neve2006". aCGH and gene expression for 50 samples (cell lines) were profiled with an OncoBAC and Affymetrix HG-U133A arrays. We preprocessed and matched data as follows. Data were segmented with the CBS algorithm of [**?**] and discretized with CGHcall [**?**] (also available from Bioconductor). Matching of features was done using the Bioconductor package `sigaR` [**?**] and the *distanceAny* method with a window of 10,000 bp. The present package includes preprocessed data for chromosome 17 only.

Data are loaded in the following way:

```
> library(plrs)
> data(neveCN17)
> data(neveGE17)
```

This loads to the current environment an "ExpressionSet" object `neveGE17` and a "cghCall" object `neveCN17`, which contain **preprocessed** expression and copy number data with **matched** features:

```
> neveGE17

ExpressionSet (storageMode: lockedEnvironment)
assayData: 1185 features, 50 samples
  element names: exprs
protocolData: none
phenoData: none
featureData
  featureNames: 207311_at 221614_s_at ... 215055_at (1185 total)
  fvarLabels: Chr Start End Symbol
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:

> neveCN17

cghCall (storageMode: lockedEnvironment)
assayData: 1185 features, 50 samples
  element names: calls, copynumber, probamp, probgain, probloss, probnorm, segmented
protocolData: none
phenoData: none
featureData
  featureNames: GS1-68F18_207311_at GS1-68F18_221614_s_at ...
    RP11-165M24_215055_at (1185 total)
  fvarLabels: Chromosome Start End
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
```

4 Analysis of a single DNA-mRNA relationship

In this section, we focus on a single *cis*-effect and show how to specify and fit a model. Procedures for selecting an appropriate model, testing the association and obtaining confidence intervals for the mean response are presented.

4.1 Obtain data for a single gene

We choose gene PITPNA for illustration:

```
> # Index of gene PITPNA
> idx <- which(fData(neveGE17)$Symbol=="PITPNA")[1]
```

```
> # Obtain vectors of gene expression (normalized) and
> # copy number (segmented and called) data
> rna <- exprs(neveGE17)[idx,]
> cghseg <- segmented(neveCN17)[idx,]
> cghcall <- calls(neveCN17)[idx,]

> # Obtain vectors of posterior membership probabilities
> probloss <- probloss(neveCN17)[idx,]
> probnorm <- probnorm(neveCN17)[idx,]
> probgain <- probgain(neveCN17)[idx,]
> probamp <- probamp(neveCN17)[idx,]
```

Posterior probabilities are used for determining knots (or change points) of the model. Their calculation is explained in [**?**].

4.2 Number of observations per state

For a given gene, PLRS accommodates *differential* DNA-mRNA relationships across the different types of genomic aberrations or *states*. Presently, we (only) distinguish four of them: **loss** (coded as -1), **normal** (0), **gain** (1) and **amplification** (2). To ensure that the model is identifiable, the sample size for each state needs not to be too small. A minimum number of three samples is required for estimation, however any higher number may be chosen/preferred.

```
> # Check: how many observations per state?
> table(cghcall)

cghcall
-1  0  1
19 28  3
```

Here, it is possible to fit a 3-state model. If the minimum number of observations is not fulfilled for a given state, there are two possibilities: discard data corresponding to the given state or merging it to an adjacent one. The function `modify.conf()` accommodates these two options (see below). With this function one can actually control the minimum number of samples per state.

```
> # Set the minimum to 4 observations per state
> cghcall2 <- modify.conf(cghcall, min.obs=4, discard=FALSE)
> table(cghcall2)

cghcall2
-1  0
19 31

> # Set the minimum to 4 observations per state
> cghcall2 <- modify.conf(cghcall, min.obs=4, discard=TRUE)
> table(cghcall2)
```

```
cghcall2
-1  0
19 28
```

In practice, the user does not have to call directly `modify.conf()` as it is implemented internally in function `plrs()`, which is used to fit a model. However, one needs to be aware that the minimum number of observations per state (specified via arguments `min.obs` and `discard.obs` in `plrs()`) has a strong influence on the resulting model.

4.3 Model fitting

Function `plrs()` allows the fitting of a PLRS model. Different types of models may be specified conveniently. For instance, one may choose to fit a *continuous* PLRS (i.e. with no state-specific intercepts or discontinuities) or to change the type of restrictions on parameters or simply leave them out. Although we recommend users to use default argument values, we below describe how to specify alternative types of models.

The followings arguments of function `plrs()` offer flexibility for modeling:

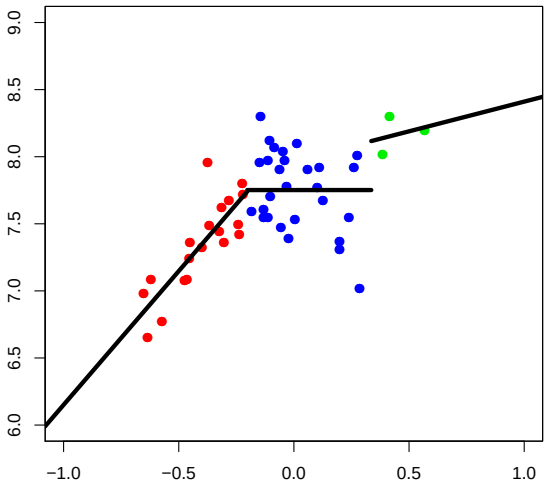
- `continuous = TRUE` or `FALSE` (default)
  Specify whether state-specific intercepts should be included.
- `constr = TRUE` (default) or `FALSE`
  Specify whether inequality constraints on parameter should be applied or not.
- `constr.slopes = 1` or `2` (default)
  Specify the type of constraints on slopes; options are:
  1. `constr.slopes = 1` forces all slopes to be non-negative.
  2. `constr.slopes = 2` forces the slope of the "normal" state (coded 0) to be non-negative while all others are forced to be at least equal to the latter.
- `constr.intercepts = TRUE` (default) or `FALSE`
  Specify whether state-specific intercepts are to be non-negative.

Note that when `constr = FALSE`, `constr.slopes` and `constr.intercepts` have no effect.

With default settings (recommended):

```
> # Fit a model
> model <- plrs(rna, cghseg, cghcall, probloss, probnorm, probgain, probamp)
> model

Object of class "plrs"

Spline coefficients:
theta0.0 theta0.1 theta1.0 theta1.1 theta2.0 theta2.1
```

```
 8.14440   1.99381   0.01071 -1.99381   0.36509   0.44272

Model is constrained:
constr.slopes = 2
constr.intercepts = TRUE

> plot(model)
```



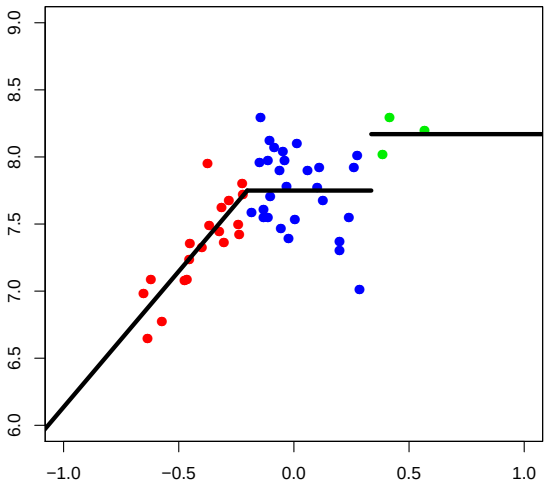
The `plrs()` function returns an S4 object of class "plrs". Various generic functions have been defined on this class:

- Functions `print()` and `summary()` display information about the model (e.g. estimated coefficients, type of constraints, etc...).
- Function `plot()` displays the fit of the PLRS model along with data. Segmented copy number is on x-axis while (normalized) gene expression is on y-axis; colors indicate the different aberration states, namely "loss" (red), "normal" (blue) and "gain" (green); the black line gives the fit of the PLRS model. Colors and symbols may be changed via the appropriate arguments of function `plrs()`. Note that the argument `lin`, if set to `TRUE`, will additionally display a dashed (default) line giving the fit of the simple linear model.
- Other useful functions include `coef()`, `fitted()`, `residuals()`, `model.matrix()`, `predict()` and `knots()`. These are standard functions. Information about these can be found in associated help files.

4.4 Select an appropriate model

Model selection is carried out by `plrs.select()`, which takes has input an object of class "plrs". Possible model selection criteria are `OSAIC`, `AIC`, `AICC` and `BIC`. When the model is constrained `OSAIC` is the only appropriate one. If the model has no restrictions on parameters, `OSAIC` and `AIC` are equivalent. See help file and associated references for more information.

```
> # Model selection
> modelSelection <- plrs.select(model)
> summary(modelSelection)

Object of class "plrs.select"

OSAIC model selection procedure

Coefficients of selected spline:
theta0.0 theta0.1 theta1.1 theta2.0
 8.15917  2.02349 -2.02349  0.42046

Model is constrained:
constr.slopes = 2
constr.intercepts = TRUE

Model selection table:
            osaic
model1  -93.81986
model2 -115.34894
model3 -112.01954
model4  -99.78747
model5  -97.89852
model6 -116.63119
...

> # Plot selected model
> plot(modelSelection)
```



The function `plrs.select()` returns an S4 object of class "plrs.select" (here `modelSelection`), which contains information on model selection. Slot `model` contains the selected model as a "plrs" object:

```
> selectedModel <- modelSelection@model
> selectedModel

Object of class "plrs"

Selected spline coefficients:
theta0.0 theta0.1 theta1.1 theta2.0
 8.15917  2.02349 -2.02349  0.42046

Model is constrained:
constr.slopes = 2
constr.intercepts = TRUE
```

Although `model` and `selectedModel` are both objects of class "plrs", generic functions defined on this class make implicitly the distinction between objects resulting from functions `plrs()` and `plrs.select()` (see above; "`Selected spline coefficients`").

⇒ It is important to note that, for correct inference, subsequent test and confidence intervals must be computed on the **full** model rather than the **selected** one. Therefore, we forced functions `plrs.test()` and `plrs.cb()` (used hereafter) to operate on the full model, regardless of the input model. This implies that inference results are the same with either objects (see below). If one wishes to obtain results for the selected model, set `selectedModel@selected` to `FALSE` and then apply the aforementioned functions.

4.5 Testing the strength of the association

The function `plrs.test()` implements a likelihood ratio test to evaluate the effect of DNA copy number on expression. It tests the hypothesis that all parameters (except the overall intercept) of the PLRS model equal 0. See [**?**, **?**] and associated references for more details on the test. The function `plrs.test()` takes as input an object class "plrs" and outputs an object from the same class. Testing results are contained in slot `test`.

```
> # Testing the full model with
> model <- plrs.test(model, alpha=0.05)
> model

Object of class "plrs"

Spline coefficients:
theta0.0 theta0.1 theta1.0 theta1.1 theta2.0 theta2.1
 8.14440  1.99381  0.01071 -1.99381  0.36509  0.44272

Model is constrained:
constr.slopes = 2
constr.intercepts = TRUE

Testing:
stat = 0.6028283
quantile = 0.1125146  (alpha = 0.05)
p-value =  1.5207e-09

> # or with
> selectedModel <- plrs.test(selectedModel, alpha=0.05)
> selectedModel

Object of class "plrs"

Selected spline coefficients:
theta0.0 theta0.1 theta1.1 theta2.0
 8.15917  2.02349 -2.02349  0.42046

Model is constrained:
constr.slopes = 2
constr.intercepts = TRUE

Testing:
stat = 0.6028283
quantile = 0.1125135  (alpha = 0.05)
p-value =  1.5197e-09

> # Testing the selected model
> selectedModel2 <- selectedModel
> selectedModel2@selected <- FALSE
> plrs.test(selectedModel2, alpha=0.05)
```

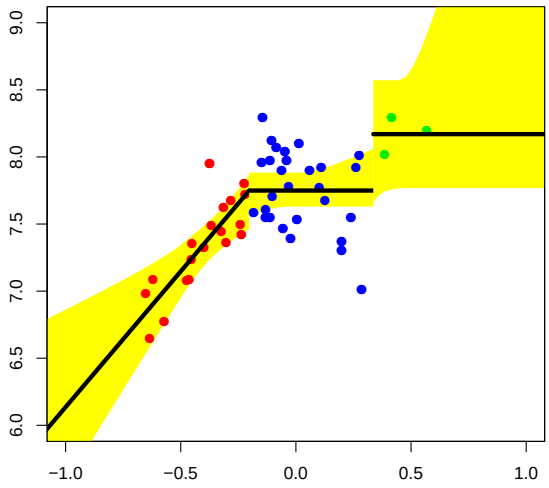
```
Object of class "plrs"

Spline coefficients:
theta0.0 theta0.1 theta1.1 theta2.0
 8.15917  2.02349 -2.02349  0.42046

Model is constrained:
constr.slopes = 2
constr.intercepts = TRUE

Testing:
stat = 0.5938289
quantile = 0.09207474  (alpha = 0.05)
p-value =  4.434e-10
```

The object `selectedModel` now contains information on testing, which are consequently displayed when printing.

4.6 Uniform confidence bands

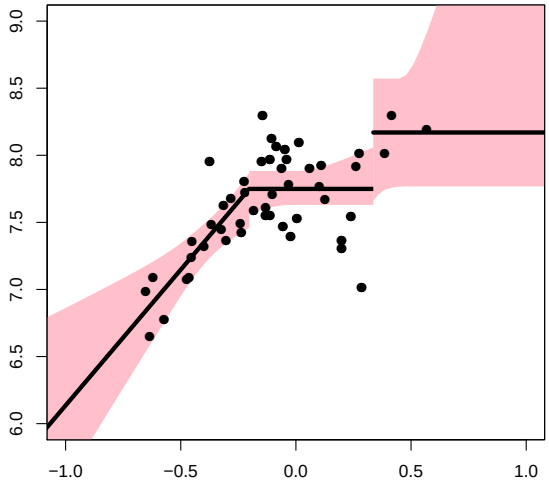
Confidence intervals for the spline are computed with function `plrs.cb`. Again, the function requires an object of class "plrs". However, the input object must result from function `plrs.test()`). This is because information from the test is required (mixture's weights). `plrs.cb` outputs an object of class "plrs" and computed lower and upper bounds for the mean response are stored in slot `cb`.

```
> # Compute and plot CBs
> selectedModel <- plrs.cb(selectedModel, alpha=0.05)
> str(selectedModel@cb)

List of 3
 $ inf: num [1:102] 4.17 4.26 4.35 4.43 4.52 ...
 $ sup: num [1:102] 6.44 6.46 6.49 6.51 6.54 ...
 $ x  : num [1:102] -1.5 -1.47 -1.44 -1.41 -1.38 ...

> plot(selectedModel)
```



Confidence bands are automatically plotted. Color may be change with input arguments `col.cb`. For example:

```
> plot(selectedModel, col.pts="black", col.cb="pink")
```



5 Analysis of multiple DNA-mRNA relationships

Function `plrs.series()` allows to apply the different aforementioned procedures to multiple relationships. Input data can be given as *matrix* objects or *ExpressionSet* and *cghCall* objects. We now show how to: 1) implement the PLRS screening test and 2) implement the model selection procedure. For the purpose of speed, we work on a subset of chromosome 17 (first 200 features).

```
> # Testing the full model, no model selection (fast)
> neveSeries <- plrs.series(neveGE17[1:200,], neveCN17[1:200,], control.select=NULL)

In progress...

10% done (21 genes),          time elapsed = 0:00:01
20% done (41 genes),          time elapsed = 0:00:02
30% done (61 genes),          time elapsed = 0:00:03
40% done (81 genes),          time elapsed = 0:00:03
50% done (100 genes),          time elapsed = 0:00:04
60% done (120 genes),          time elapsed = 0:00:05
70% done (140 genes),          time elapsed = 0:00:06
80% done (160 genes),          time elapsed = 0:00:07
90% done (180 genes),          time elapsed = 0:00:08
100% done (200 genes),          time elapsed = 0:00:09

> # Testing the full model and applying model selection
> neveSeries2 <- plrs.series(neveGE17[1:200,], neveCN17[1:200,])

In progress...

10% done (21 genes),          time elapsed = 0:00:05
20% done (41 genes),          time elapsed = 0:00:09
30% done (61 genes),          time elapsed = 0:00:13
40% done (81 genes),          time elapsed = 0:00:18
50% done (100 genes),          time elapsed = 0:00:22
60% done (120 genes),          time elapsed = 0:00:26
70% done (140 genes),          time elapsed = 0:00:30
80% done (160 genes),          time elapsed = 0:00:35
90% done (180 genes),          time elapsed = 0:00:40
100% done (200 genes),          time elapsed = 0:00:44
```

Function `plrs.series()` has arguments `control.model`, `control.select` and `control.test`, which allows to specify the type of model, model selection and whether the test and confidence bands should be computed. Argument `control.output` allows the user to save each "plrs" objects and/or associated plot in the working directory.
`plrs.series()` outputs an object of class "plrs.series". Generic functions `print()` and `summary()` display information on fitted models, selected models and/or the testing procedure.

```
> # Summary of screening test
> neveSeries

Object of class "plrs.series"

200 genes

Type of fitted model:
continuous = FALSE
constr.slopes = 2
constr.intercepts = TRUE

> summary(neveSeries)

Object of class "plrs.series"

200 genes

Type of fitted model:
continuous = FALSE
constr.slopes = 2
constr.intercepts = TRUE

Configuration:
                -1      0      1  2
Nb genes    200.00 200.00 200.00  0
Min. obs     19.00  25.00   3.00 NA
Median. obs  20.00  26.00   4.00 NA
Mean. obs    19.62  26.28   4.09 NA
Max. obs     20.00  28.00   5.00 NA

Nb of models regarding their types:
Intercept         0
Simple linear     0
Piecewise level   0
Piecewise linear  200

Testing:
Number of rejected null hypothesis: 99 genes
(at 0.1 significance level based on
 Benjamini-Hochberg corrected p-values)

> # Summary of screening test and model selection
> neveSeries2

Object of class "plrs.series"

200 genes
Models selected with OSAIC
```

```
> summary(neveSeries2)

Object of class "plrs.series"

200 genes
Models selected with OSAIC

Configuration:
                -1      0      1  2
Nb genes    200.00 200.00 200.00  0
Min. obs     19.00  25.00   3.00 NA
Median. obs  20.00  26.00   4.00 NA
Mean. obs    19.62  26.28   4.09 NA
Max. obs     20.00  28.00   5.00 NA

Nb of models regarding their types:
Intercept         61
Simple linear     46
Piecewise level   18
Piecewise linear  75

Testing:
Number of rejected null hypothesis: 99 genes
(at 0.1 significance level based on
 Benjamini-Hochberg corrected p-values)
```

Results of testing and model selection procedures are obtained as follows:

```
> # Testing results
> head(neveSeries@test)

              stat     raw.pval  BH.adj.pval
gene1 1.071594e-02 5.445356e-01 0.6936759389
gene2 5.027115e-18 8.215135e-01 0.8233470527
gene3 1.497315e-03 7.298322e-01 0.8200361816
gene4 4.041468e-02 2.660278e-01 0.4124461784
gene5 3.451090e-01 7.457854e-05 0.0002711947
gene6 3.030552e-01 2.838228e-04 0.0008869464

> head(neveSeries2@test)

              stat     raw.pval  BH.adj.pval
gene1 1.071594e-02 5.445639e-01 0.6937119622
gene2 5.027115e-18 8.215226e-01 0.8233336426
gene3 1.497315e-03 7.298386e-01 0.8200433334
gene4 4.041468e-02 2.660449e-01 0.4124727470
gene5 3.451090e-01 7.460674e-05 0.0002712972
gene6 3.030552e-01 2.838300e-04 0.0008869688
```

```
> # Coefficients of selected models
> head(neveSeries2@coefficients)

      theta0.0  theta0.1 theta1.0 theta1.1 theta2.0 theta2.1 theta3.0 theta3.1
gene1 5.816194        NA       NA       NA       NA       NA       NA       NA
gene2 4.058224        NA       NA       NA       NA       NA       NA       NA
gene3 3.726981        NA       NA       NA       NA       NA       NA       NA
gene4 4.700387 0.1453787       NA       NA       NA       NA       NA       NA
gene5 6.805236 1.0396287       NA       NA       NA 3.192212       NA       NA
gene6 6.514909 1.0953053       NA       NA       NA       NA       NA       NA

>
```