The biosvd package for high-throughput data processing, outlier detection, noise removal and dynamic modeling

Anneleen Daemen*[1] and Matthew Brauer†[1]

[1]Department of Bioinformatics and Computational Biology, Genentech Inc., South San Francisco, CA

October 29, 2019

Contents

*daemena@gene.com
†matthejb@gene.com

1 Introduction

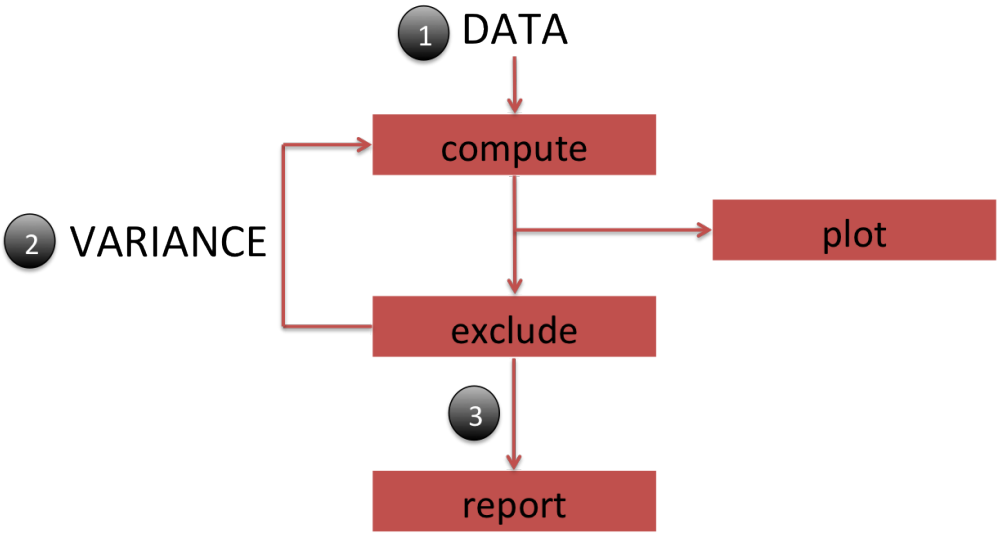
Singular Value Decomposition (SVD) is a popular method, suited for longitudinal data processing and modeling. Simply stated, SVD decomposes data to a linear combination of main major modes of intensity. SVD for the analysis of genome-wide expression data was introduced by Alter and colleagues in 2000, with the major modes referred to as eigengenes and eigenarrays [Alter et al, 2000]. Sorting the expression data to these modes revealed clusters of genes or arrays with similar function or biologic phenotype. Here, we extend the application of SVD to any data type. This BioConductor R package allows for high-throughput data processing, outlier detection, noise removal and dynamic modeling, based on the framework of Singular Value Decomposition. It provides the user with summary graphs and an interactive html report. This gives a global picture of the dynamics of expression/intensity levels, in which individual features and assays are classified in groups of similar regulation and function or similar cellular state and biological phenotype.

1.1 Singular Value Decomposition (SVD)

Singular Value Decomposition is a linear transformation of a data set E from the M features x N assays space to a reduced L eigenfeatures x L eigenassays space, with $L = \min\{M, N\}$. In mathematical terms, this corresponds to $E = U\Sigma V^T$. U and V^T define the M features x L eigenassays and the L eigenfeatures x N assays orthonormal basis sets. Each column in U corresponds to a left singular vector, representing genome-wide expression, proteome-wide abundance or metabolome-wide intensity in the k-th eigenassay. Accordingly, each row in V^T is called a right singular vector and represents the expression, abundance or intensity of the k-th eigenfeature across all assays.

Σ is a diagonal matrix with expression of each eigenfeature restricted to the corresponding eigenassay, reflecting the decoupling and decorrelation of the data. The eigenexpression levels along the diagonal indicate the relative significance of each {eigenfeature, eigenassay}-pair. The relative fraction of overall expression that the k-th eigenfeature and eigenassay capture is called *eigenexpression fraction* and is defined as $f_l = \epsilon_l^2 / \sum_{k=1}^{L} \epsilon_k^2$. Finally, data complexity is expressed as the Shannon entropy, defined as $0 \leqslant \frac{-1}{log(L)} \sum_{k=1}^{L} f_k log(f_k) \leqslant 1$. An entropy of 0 corresponds to an ordered and redundant data set, with all expression captured by a single {eigenfeature, eigenassay}-pair. On the other hand, the entropy is 1 in case of a disordered and random data set with all {eigenfeature, eigenassay}-pairs equally expressed. We refer to [1] for a more detailed description of SVD.

1.2 biosvd Package Overview



The biosvd package consists of 4 main functions. First, `compute` reduces the input data set from the feature x assay space to the reduced diagonalized eigenfeature x eigenassay space, with the eigenfeatures and eigenassays unique orthonormal superpositions of the features and assays, respectively. Results of SVD applied to the data can subsequently be inspected based on the graphs generated with `plot`. These graphs include a heatmap of the eigenfeature x assay matrix (V^T), a heatmap of the feature x eigenassay matrix (U), a bar plot with the eigenexpression fractions of all eigenfeatures, and the levels of the eigenfeatures across the assays, and polar plots of the assays and features, displaying the features/assays according to their correlation with two selected eigenfeatures/eigenassays. These graphs aid in deciding which eigenfeatures and eigenassays to filter out (i.e., eigenfeatures representing steady state, noise, or experimental artifacts) (`exclude`). Filtering out steady-state expression/intensity corresponds to centering the expression/intensity patterns at steady-state level (arithmetic mean of intensity ~ 0).

Secondly, the three functions `compute`, `plot` and `exclude` can be applied to the variance in the data, in order to filter out steady-scale variance. This corresponds to a normalization by the steady scale of expression/intensity variance (geometric mean of variance ~ 1).

Thirdly, after possible removal of steady state expression, steady-scale variance, noise and experimental artifacts, SVD is re-applied to the normalized data, followed by the generation of plots with `plot` , and a summary txt file with `report`, containing the list of features, their coordinates, radius and phase in the polar plot, and any additional feature data provided by the user.

1.3 Case Studies Overview

In this vignette, three case studies are provided. In the first 2 case studies, the expression pattern of genes throughout the cell cycle are studied in yeast and human, respectively. As use of this package is not restricted to expression data, the third case study focuses on cellular metabolites in bacteria and yeast after carbon and nitrogen starvation.

The essential data must be provided as a feature x assay matrix, a data frame, ExpressionSet or an object from class eigensystem obtained from a former run. For the examples provided in this vignette, ExpressionSets were created containing the gene x sample expression data with gene annotation and sample information, or the metabolite x assay intensity data with metabolite annotation and assay information.

2 Case Study 1: Yeast Cell Cycle Expression, Alpha-factor Block.

As a first example, Spellman and colleagues created a comprehensive catalog of genes in *Saccharomyces cerevisiae* whose transcript levels vary periodically within the cell cycle [2]. To this end, mRNA levels in samples from yeast cultures were synchronized in G1 phase with α factor arrest. After release of the α factor, cells were sampled every 7 minutes over a timespan of 140 minutes, during which the cells synchronously completed two cell cycles. The gene x sample expression data comprise the (un-logtransformed) ratio of gene expression to reference mRNA from an asynchronous yeast culture. For each sample, the cell cycle phase is known as determined by Spellman *et al*. For 800 cell cycle-regulated genes, the phase in which these genes reach their peak expression was determined by Spellman *et al* based on published timing of the expression of known cell cycle-regulated genes.

We start the example by loading the data and all required libraries:

```
> library(biosvd)
> data(YeastData_alpha)
> colnames(pData(YeastData))[match("Cell.cycle.stage", colnames(pData(YeastData)))] <-
+ "Cellcycle.sample"
> colnames(fData(YeastData))[match("Cell.cycle.stage", colnames(fData(YeastData)))] <-
+ "Cellcycle.gene"
> YeastData

ExpressionSet (storageMode: lockedEnvironment)
assayData: 2302 features, 18 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: 0min_y744n40 7min_y744n98 ... 119min_y744n72 (18 total)
  varLabels: Sample.ID Experiment ... Cellcycle.sample (5 total)
  varMetadata: labelDescription
featureData
  featureNames: YAL001C YAL002W ... YPR198W (2302 total)
  fvarLabels: Clone.ID Gene.symbol Cellcycle.gene
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
```

The input data set is first reduced from the gene x sample space to the reduced diagonalized eigenfeature x eigenassay space.

```
> eigensystem <- compute(YeastData)
```

The eigenfeatures and eigenassays can subsequently be inspected based on the graphs generated with `plot`. All plot-related settings can be specified with an object of class `EigensystemPlotParam` . The default settings for all plots are as follows:

```
> params <- new("EigensystemPlotParam")
> params

class: EigensystemPlotParam
plots :  eigenfeatureHeatmap eigenassayHeatmap sortedHeatmap fraction scree zoomedFraction lines allLines
whichAssays :
whichFeatures :
```

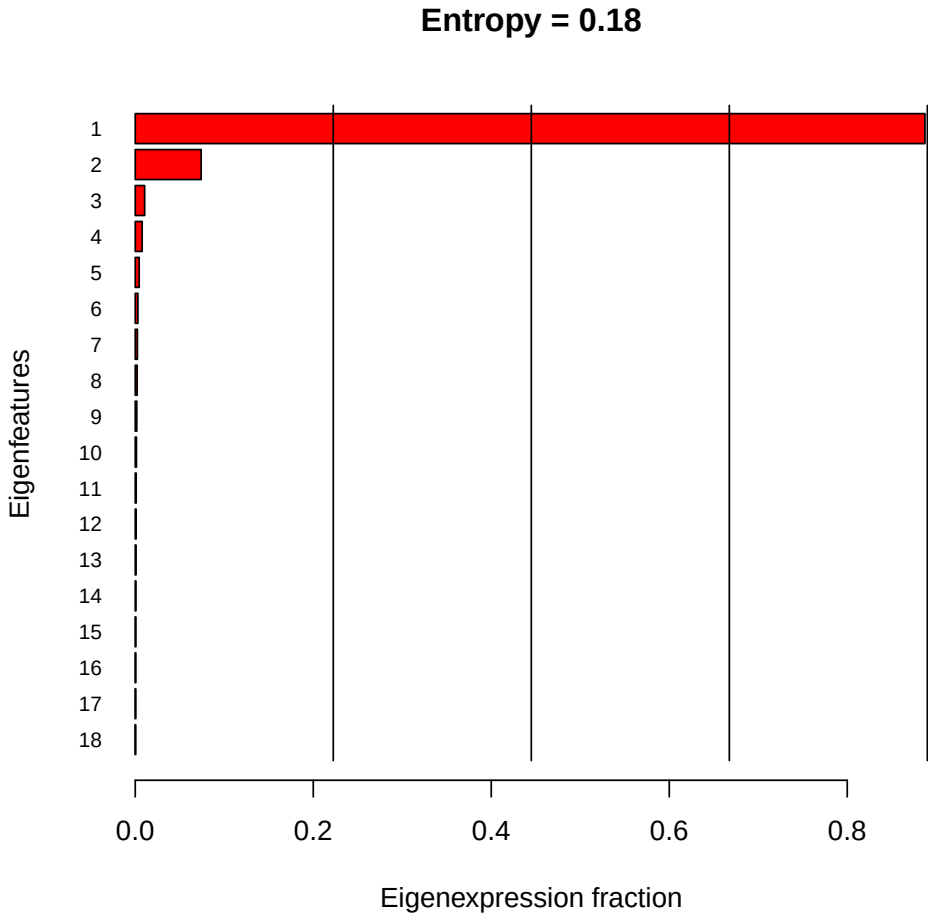
```
whichEigenassays :
whichEigenfeatures :  1 2 3 4
whichPolarAxes :  2 1
assayColorMap :
featureColorMap :
contrast :  3
negativeValues :  TRUE
path :  /tmp/RtmpGyL5or/Rbuild2a135ccaf5ab/biosvd
prefix :  biosvd
filenames :
figure :  FALSE
```

Up to 10 figures are displayed (`figure(params)=TRUE`) or saved as a pdf file (default `figure(params)=FALSE`), using the `plots(params)` argument as follows:

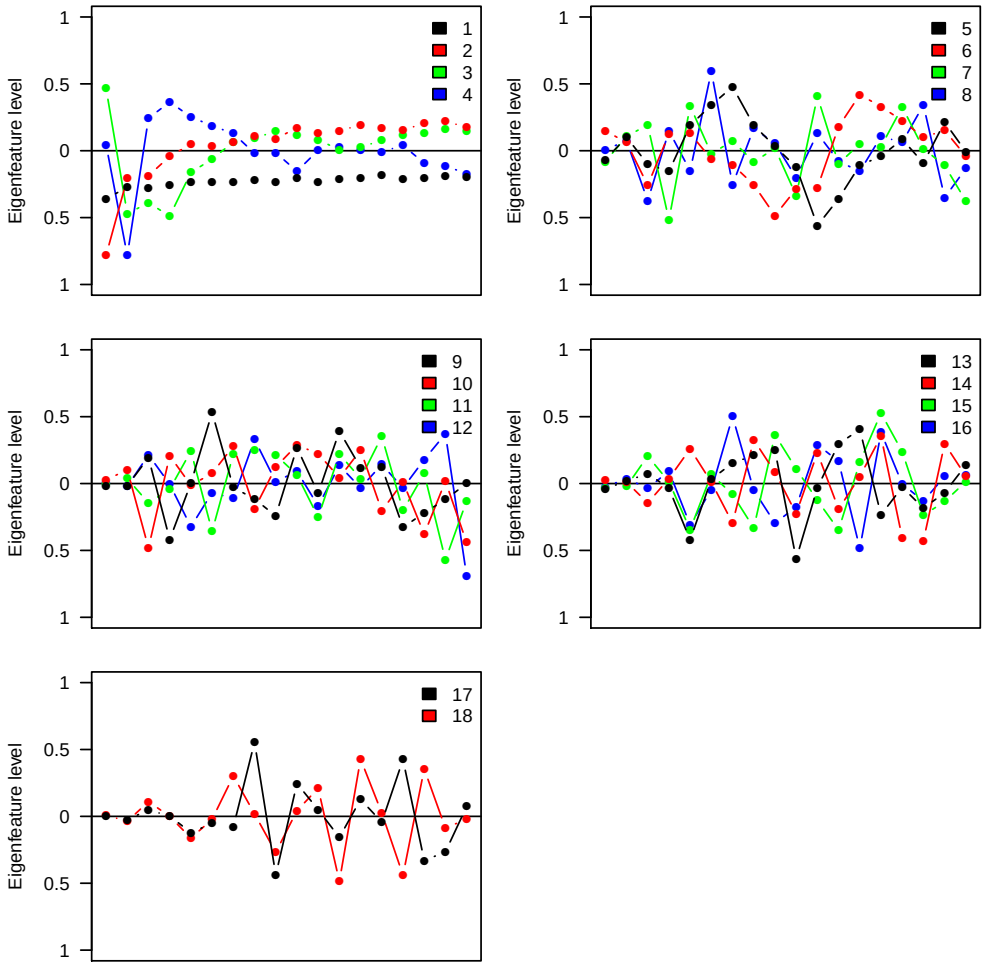
- **fraction**: bar plot with the eigenexpression fractions of all eigenfeatures
- **zoomedFraction**: bar plot with the eigenexpression fractions of the eigenfeatures after removal of the dominant eigenfeature(s)
- **scree**: screeplot for the eigenexpression fractions
- **eigenfeatureHeatmap**: heatmap of the eigenfeature x assay matrix with use of a given contrast factor `contrast(params)`
- **eigenassayHeatmap**: heatmap of the eigenfeature x assay matrix with use of a given contrast factor `contrast(params)`
- **sortedHeatmap**: heatmap of the feature x assay matrix with use of a given contrast factor `contrast(params)`, with features ordered according to their correlation with two eigenfeatures, specified with `whichPolarAxes(params)`
- **lines**: expression/intensity levels of specified eigenfeatures across the assays (`whichEigenfeatures(params)`, by default 1-4)
- **allLines**: expression/intensity levels of all eigenfeatures across the assays
- **eigenfeaturePolar**: polar plot for the features according to their correlation with two eigenfeatures, specified with `whichPolarAxes(params)`
- **eigenassayPolar**: polar plot for the assays according to their correlation with two eigenassays, specified with `whichPolarAxes(params)`

For this example, the bar plot with all eigenfeatures (`fraction`), the expression levels of all eigenfeatures across the samples (`allLines`), and the heatmap of the eigenfeature x assay matrix (`eigenfeatureHeatmap`) are generated, with *YeastData* as prefix for visualization purpose (`prefix(params)`). The bar plot shows that the first eigenfeature captures 89% of the overall relative expression in the experiment. The entropy of the data set is therefore low ($0.18 \ll 1$). The expression level of the first eigenfeature across the samples as displayed in the `allLines` graph shows a time-invariant relative expression during the cell cycle. The low entropy in combination with the steady-state expression captured in the first eigenfeature suggests that the underlying processes are manifested by weak perturbations of a steady state of expression. The second eigenfeature describes an initial transient increase in relative expression superimposed over time-invariant relative expression, and is therefore inferred to represent response to synchronization in the cell cycle. Inspecting the remaining eigenfeature patterns across the samples reveals that eigenfeature 8 and 10 to 18 all show rapidly varying relative expression during the cell cycle. They can therefore be considered as noise. The heatmap of the eigenfeatures by samples reveals the same information, with a clear constant expression for eigenfeature 1.

```
> fractions(eigensystem)[[1]]

[1] 0.887291

> plots(params) <- "fraction"
> figure(params) <- TRUE
> prefix(params) <- "YeastData"
> plot(eigensystem, params)
```




```
> plots(params) <- "allLines"
> plot(eigensystem, params)
```




```
> plots(params) <- "eigenfeatureHeatmap"
> plot(eigensystem, params)
```

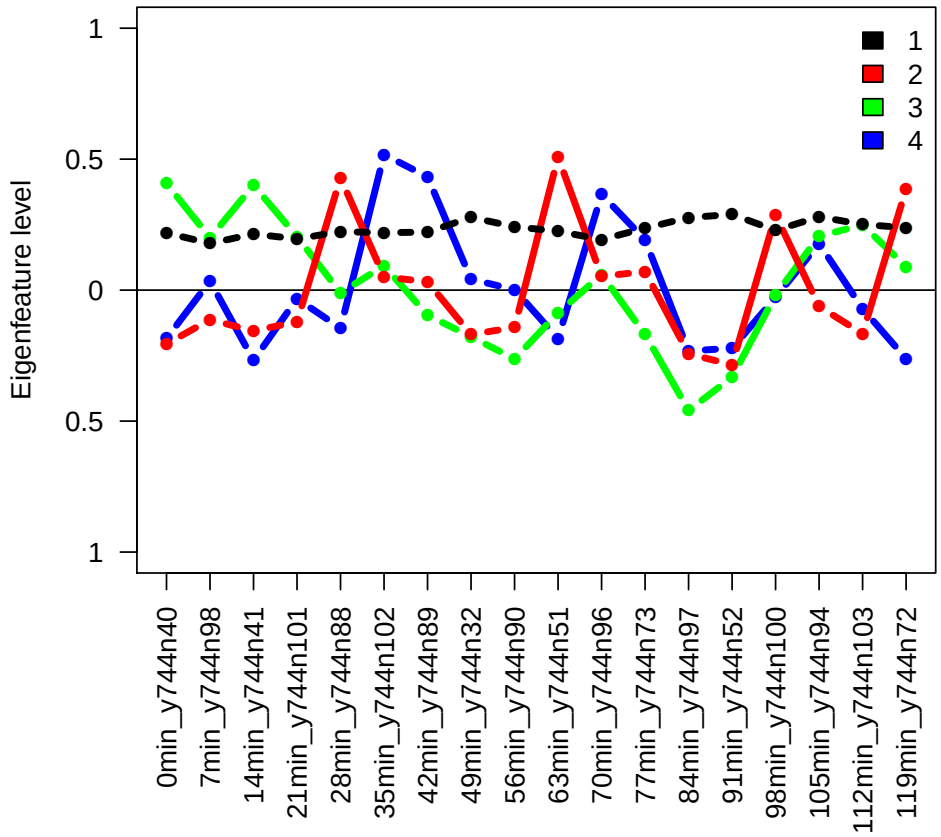
We now use `exclude` to filter out steady-state expression captured by eigenfeature 1, an experimental artifact captured by eigenfeature 2 (i.e. initial response to synchronization in the cell cycle), and noise captured by eigenfeatures 8 and 10 to 18.

```
> eigensystem <- exclude(eigensystem,excludeEigenfeatures=c(1,2,8,10:18))
```

Subsequently, we apply the same strategy to the variance in the data. The first eigenfeature now captures 88% of overall information, representing a time-invariant scale of expression variance, with an entropy of 0.23. This eigenfeature is therefore removed from the data set. Besides exclusion of the specified eigenfeatures, `exclude` regenerates the eigensystem for the normalized expression data after removal of steady-scale variance.

```
> eigensystem <- compute(eigensystem, apply='variance')
> entropy(eigensystem)

[1] 0.23

> fractions(eigensystem)[[1]]

[1] 0.8844306
```

```
> plots(params) <- "lines"
> plot(eigensystem, params)
> eigensystem <- exclude(eigensystem, excludeEigenfeatures=1)
```

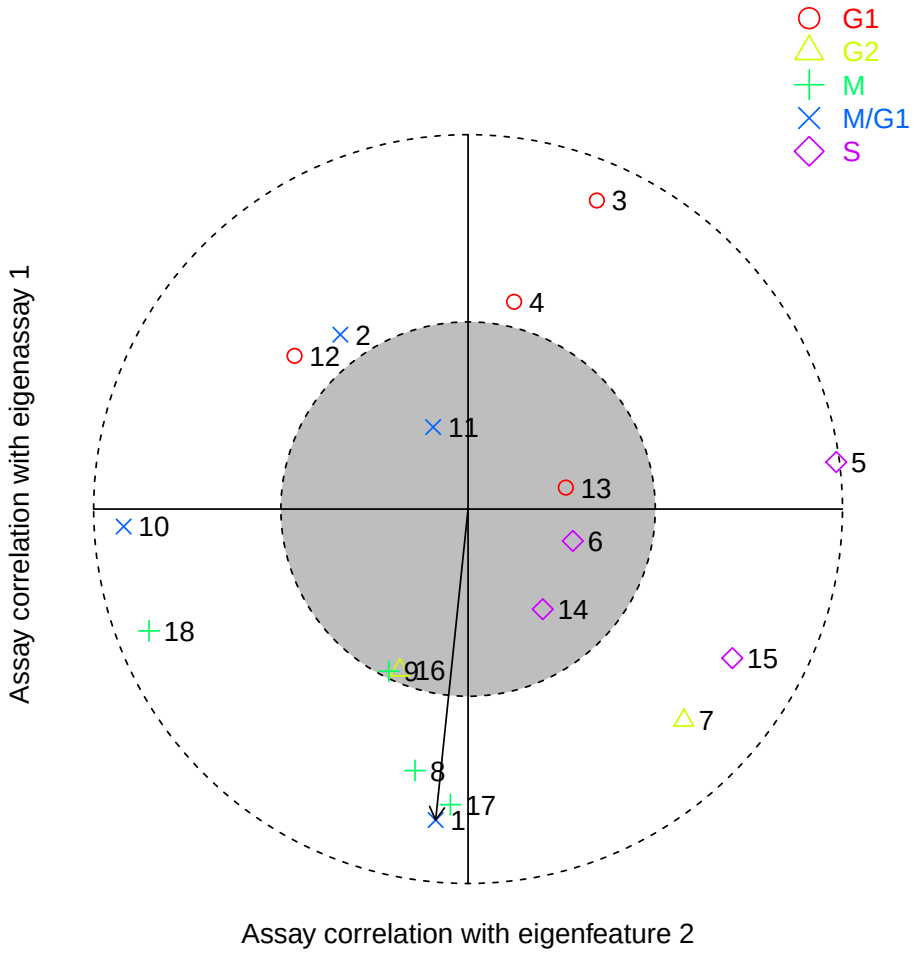


Now that steady-state expression, steady-scale variance, experimental artifacts and noise have been removed, a summary txt file and key graphs are generated. The txt file contains for all genes the cartesian coordinates, radius and phase in the eigenfeature polar plot, and annotation information that was provided with the input ExpressionSet. For the coloring of the genes and samples in the polar plots and heatmap, the cell cycle phase information from the ExpressionSet `YeastData` is used. We specify `assayColorMap(params)`and `featureColorMap(params)`, using variable `Cellcycle.sample`and `Cellcycle.gene`in the respective `pData`and `fData`objects of the ExpressionSet `YeastData` . Given that we don't want to specify the coloring for the various cell cycle phases ourselves, we set these color maps to `NA`.

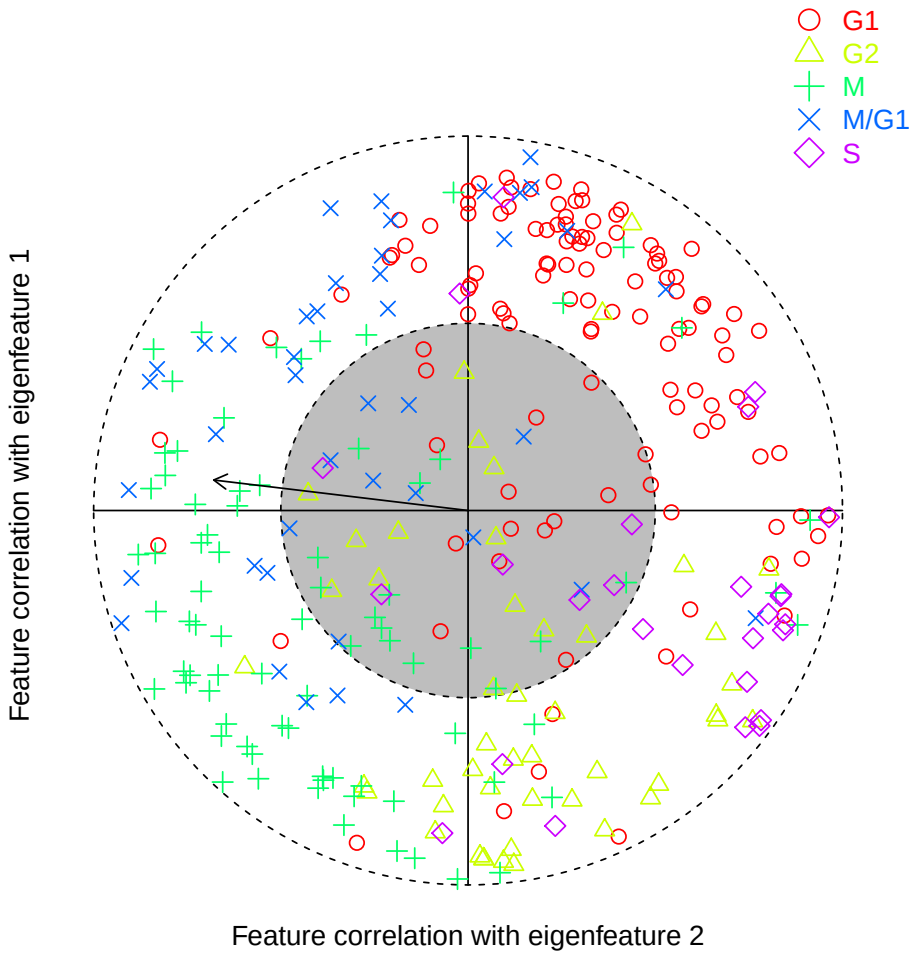
The polar plots show the genes and samples in the subspace spanned by two selected eigenfeatures and eigenassays, respectively. Because the first and second eigenfeature capture together more than 45% of the overall normalized expression, these eigenfeatures were used as subspace (default for `whichPolarAxes(params)`). These two {eigenfeature, eigenassay}-pairs are sufficient to approximate the expression data when genes and samples have most of their normalized expression in this subspace (i.e., $0.5 <$ radius < 1).

```
> assayColorMap(params) <- list(Cellcycle.sample=NA)
```

```
> plots(params) <- "eigenassayPolar"
> plot(eigensystem, params)
```




```
> featureColorMap(params) <- list(Cellcycle.gene=NA)
> plots(params) <- "eigenfeaturePolar"
> plot(eigensystem, params)
```




```
> fractions(eigensystem)[c(1, 2)]

        1         2
0.2334416 0.2184166

> report(eigensystem, params)
```

As can be seen on the assay polar plot, eigenassay 1 is positively associated with mainly samples from G1 phase, whilst this eigenassay is negatively associated with samples from the S, G2 and M phases. Eigenassay 2 is positively associated with G1 and S, whilst negatively associated with M and M/G1. The right upper quadrant is therefore dominated by samples from G1, the right lower quadrant by S and G2, and the left lower quadrant by M. Genes in each of these quadrants in the feature polar plot can subsequently be linked to and interpreted in function of each of these cell cycle phases. The genes are colored according to their expression correlation with known cell cycle-regulated genes (as determined by Spellman *et al*), with mainly G1 genes in the right upper quadrant, S and G2 genes in the right lower quadrant, and M genes in the left lower quadrant, confirming the findings obtained with the biosvd package.

Data were also sorted according to the first and second eigenfeature, and displayed in a gene x sample heatmap using the `sortedHeatmap`option in `plots(params)`. This heatmap with genes sorted according to the selected eigenfeatures shows a traveling wave of expression throughout the cell cycle. This visualization further aids in interpreting how genes are regulated by or function in the cell cycle.

```
> plots(params) <- "sortedHeatmap"
> plot(eigensystem, params)
```

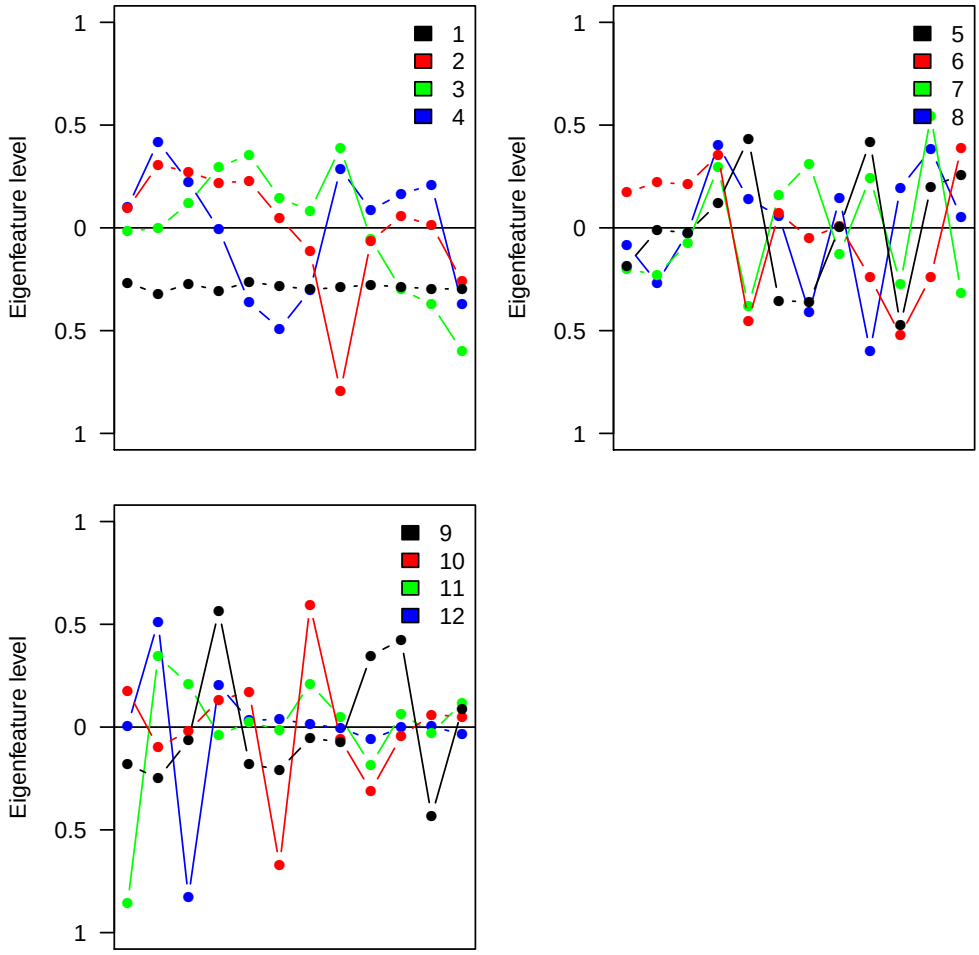
3 Case Study 2: Human HeLa Cell Cycle Expression

Secondly, a similar experiment was performed in the human HeLa cervical carcinoma cell line [3]. Cells were arrested at the beginning of S phase by using a double thymidine block. Upon release from the thymidine block, cells were sampled every 1-2 hours for 44 hours during which the cells completed three cell cycles. Similar as for the Yeast experiment, expression data comprise the un-logtransformed ratio of gene expression to reference mRNA from an asynchronous HeLa culture. Moreover, cell cycle phase is known for each sample. For >850 genes that were identified by Whitfield *et al* to be periodically expressed during the cell cycle, the phase was determined based on correlation with genes known to be expressed in each cell cycle phase (e.g. *cyclin E1* at the G1/S boundary, *RAD51* in S phase, and *TOP2A* in G2).

Similar as for the first case study, we load the HeLa data and compute the eigensystem. In this case, the first eigenfeature captures more than 90% of the relative expression, with an entropy of 0.20. As can be seen on the plot of the expression level of eigenfeatures across samples, this eigenfeature represents steady-state expression. Besides eigenfeature 1, we also decided to remove eigenfeatures 7, 10, 11 and 12, all showing rapidly varying expression during the cell cycle.

```
> data(HeLaData_exp_DoubleThym_2)
> colnames(pData(HeLaData))[match("Cell.cycle.stage", colnames(pData(HeLaData)))] <-
+ "Cellcycle.sample"
> colnames(fData(HeLaData))[match("Cell.cycle.stage", colnames(fData(HeLaData)))] <-
+ "Cellcycle.gene"
> HeLaData

ExpressionSet (storageMode: lockedEnvironment)
assayData: 11779 features, 12 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: 0hr_10127 2hr_10129 ... 44hr_10149 (12 total)
  varLabels: Sample.ID Timepoint Experiment Cellcycle.sample
  varMetadata: labelDescription
featureData
  featureNames: 1049030 1049033 ... IMAGE:998080 (11779 total)
  fvarLabels: Gene.symbol Gene.description Cellcycle.gene
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:

> eigensystem <- compute(HeLaData)
> fractions(eigensystem)[[1]]

[1] 0.9127186

> entropy(eigensystem)

[1] 0.2

> params <- new("EigensystemPlotParam")
> plots(params) <- "allLines"
```

```
> figure(params) <- TRUE
> plot(eigensystem, params)
> eigensystem <- exclude(eigensystem,excludeEigenfeature=c(1,7,10:12))
```



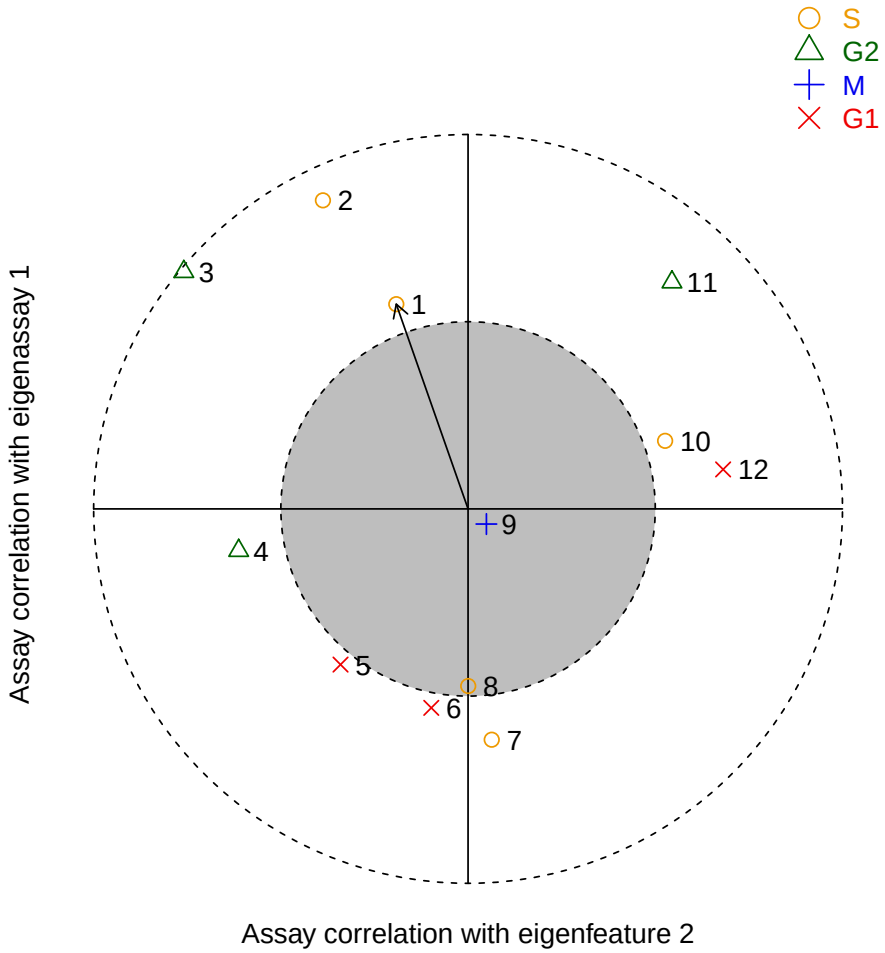
As a second step, we apply the same strategy to the variance in the data. A time-invariant scale of expression variance was captured by the first eigenfeature and therefore removed. Regarding the plots generated by `plot`, these are not shown but saved as pdf files because we set `figure(params)` to `FALSE`.

```
> eigensystem <- compute(eigensystem, apply='variance')
> entropy(eigensystem)

[1] 0.31

> fractions(eigensystem)[[1]]

[1] 0.8502717

> plots(params) <- c("eigenfeatureHeatmap", "fraction", "lines")
> figure(params) <- FALSE
> prefix(params) <- "HeLaData"
> plot(eigensystem, params)
> eigensystem <- exclude(eigensystem, excludeEigenfeatures=1)
```

As final step, we now generate the summary txt file, the eigenassay/eigenfeature polar plots, and the sorted gene x sample heatmap for the first and second eigenfeature. Similar as before, the cell cycle phase information for both the genes and samples from the ExpressionSet `HeLaData` is used for coloring of the polar plots. This time, we pre-defined the colors for the various cell cycle phases by specifying various arguments of the EigensystemPlotParam object. The gene x sample heatmap with genes sorted according to eigenfeatures 1 and 2 displays a traveling wave of expression throughout the cell cycle.

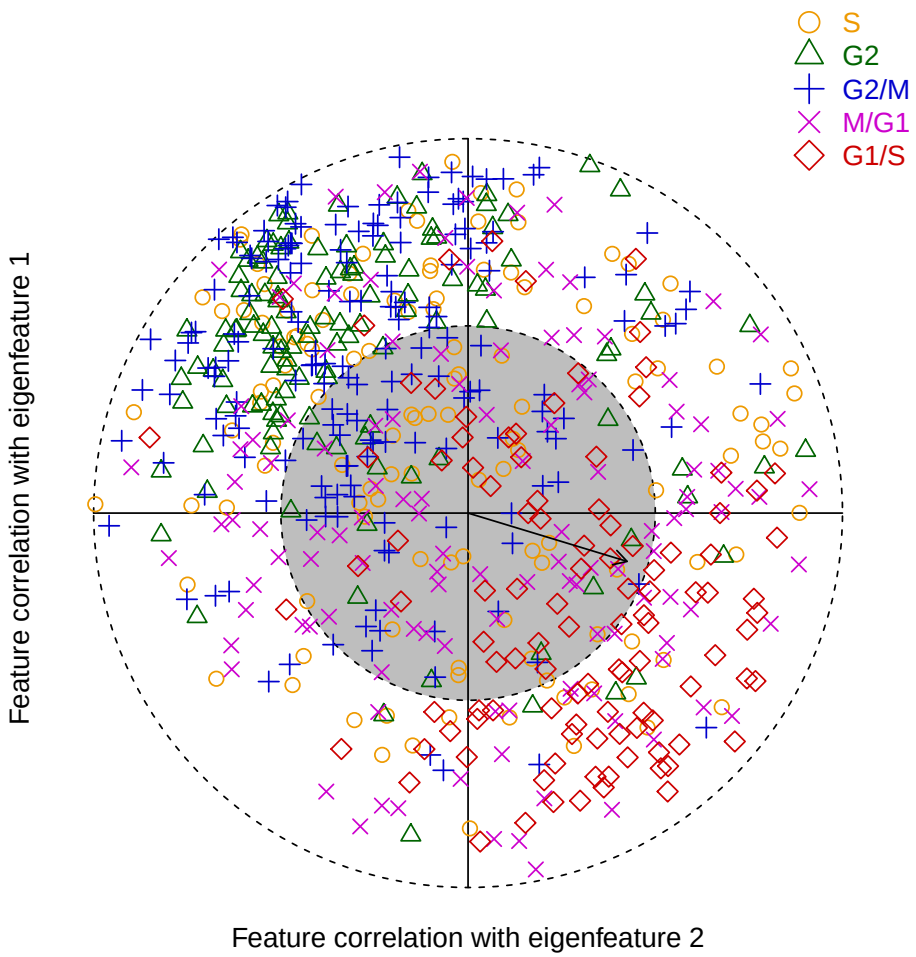
```
> report(eigensystem, params)
```

```
> cellcycle.col.map <- c("orange2", "darkgreen", "blue2", "red2")
> names(cellcycle.col.map) <- c("S", "G2", "M", "G1")
> assayColorMap(params) <- list(Cellcycle.sample=cellcycle.col.map)
> plots(params) <- "eigenassayPolar"
> figure(params) <- TRUE
> plot(eigensystem, params)
```




```
> cellcycle.col.map <- c("orange2", "darkgreen", "blue3", "magenta3", "red3")
> names(cellcycle.col.map) <- c("S", "G2", "G2/M", "M/G1", "G1/S")
> featureColorMap(params) <- list(Cellcycle.gene=cellcycle.col.map)
```

```
> plots(params) <- "eigenfeaturePolar"
> plot(eigensystem, params)
```




```
> plots(params) <- "sortedHeatmap"
> plot(eigensystem, params)
```

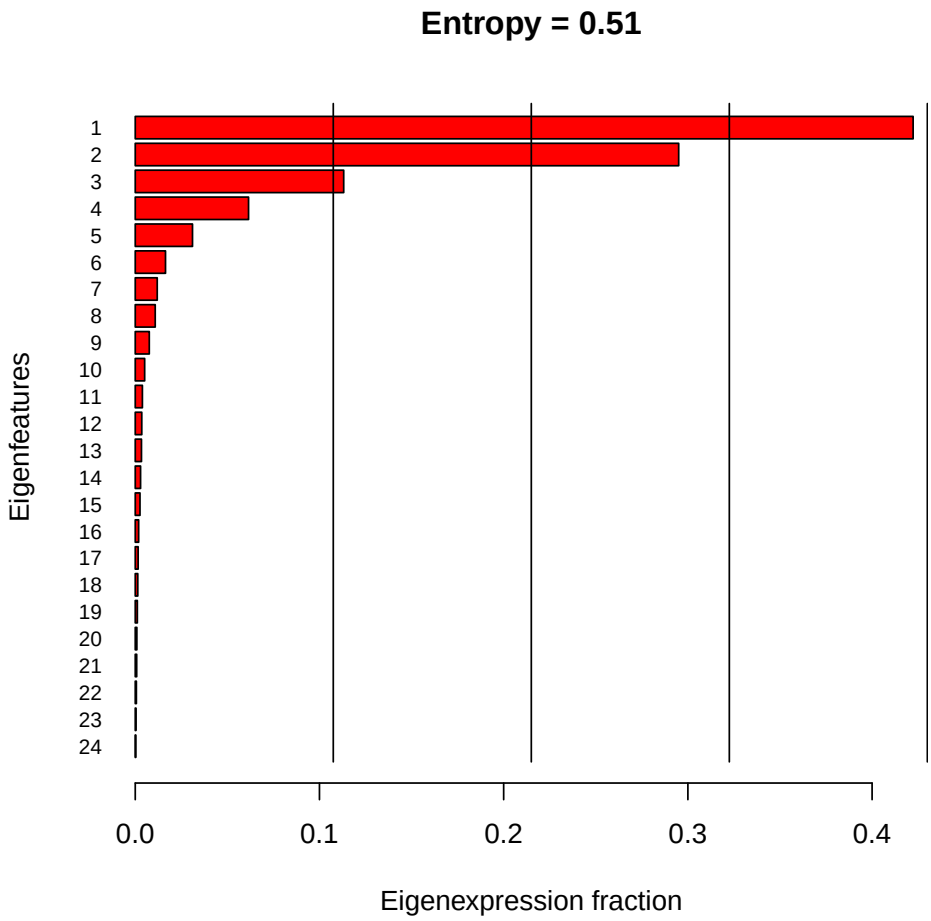
4 Case Study 3: Starvation Metabolomics

To show that use of this R package is not restricted to expression data, this third example features metabolomics data. Brauer and colleagues studied metabolic response to starvation in two microbes, *Escherichia coli* and *Saccharomyces cerevisae*, to determine whether metabolome response to nutrient deprivation is similar across both organisms [4]. Sixty-eight cellular metabolites were analyzed by LC-MS/MS in both bacteria and yeast, after nutrient starvation with carbon and nitrogen. Cells were sampled for 8 hours. The metabolomics data comprise the log-transformed relative metabolite concentration changes with respect to experiment initiation at time point 0 hours.

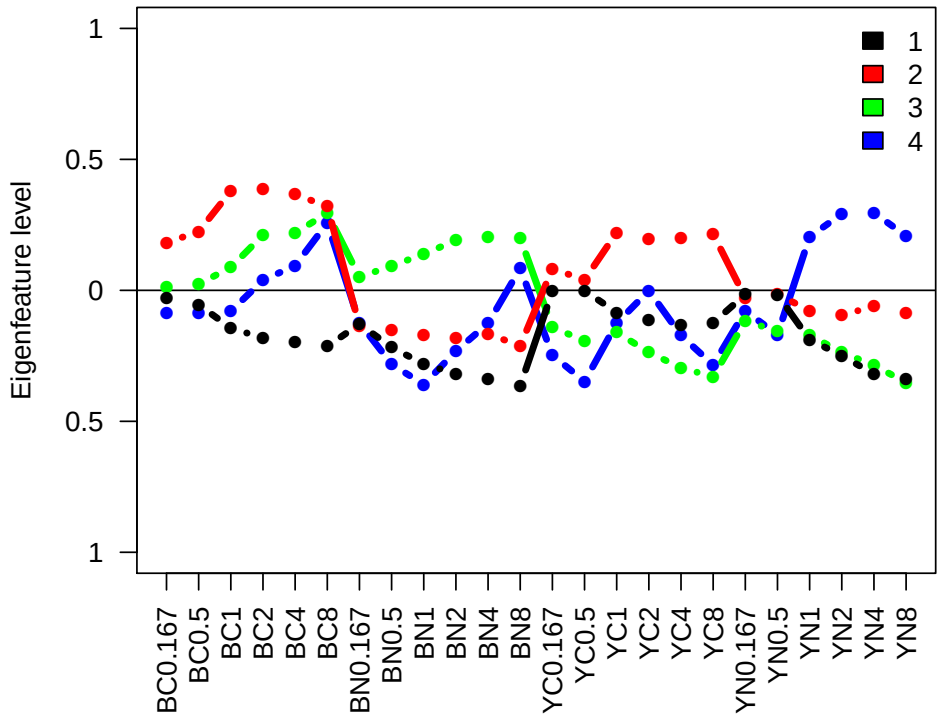
This case study not only differs from the two previous case studies in data type, but also in heterogeneity with four experiments combined into this data set (bacteria - carbon, bacteria - nitrogen, yeast - carbon, and yeast - nitrogen). This increased heterogeneity in the data explains the much higher entropy compared to the two previous studies (0.51), with the first and second eigenfeature capturing 42% and 29% of the relative intensity, respectively. Contrary to the two previous case studies, eigenfeature 1 no longer captures steady-state expression but shows a decreasing trend over time for each of the experiments, regardless of organism (B for bacteria vs. Y for yeast) and nutrient (C for carbon vs. N for nitrogen). Because we are not interested in a generic starvation response, we decided to remove the first eigenfeature at the metabolite intensity level.

Furthermore, eigenfeatures 11, 12, and 14 to 24 rapidly vary along the assays and can therefore be considered as noise.

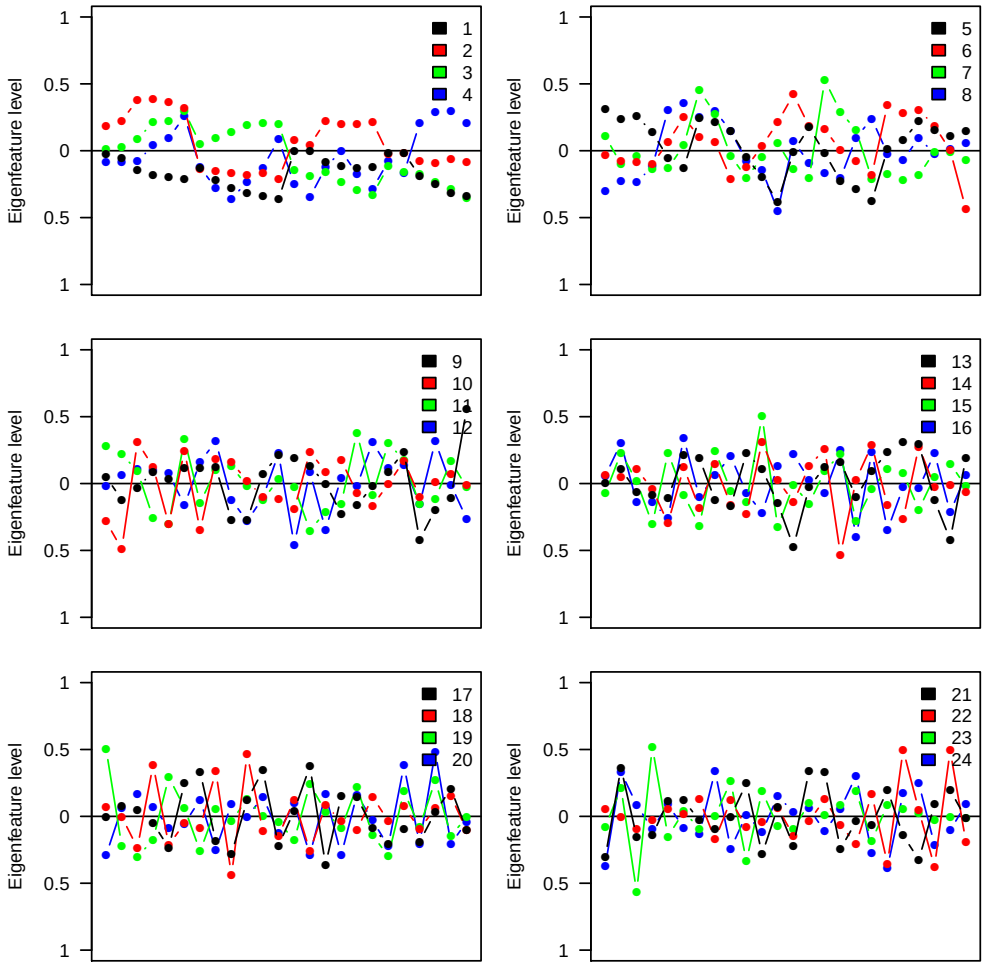
```
> data(StarvationData)
> StarvationData

ExpressionSet (storageMode: lockedEnvironment)
assayData: 57 features, 24 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: BC0.167 BC0.5 ... YN8 (24 total)
  varLabels: Species Starvation Time(hrs)
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation:

> eigensystem <- compute(StarvationData)
> fractions(eigensystem)[c(1, 2)]

        1         2
0.4222721 0.2950063

> params <- new("EigensystemPlotParam")
> plots(params) <- c("fraction")
> figure(params) <- TRUE
> prefix(params) <- "StarvationData"
> plot(eigensystem, params)
```




```
> plots(params) <- "lines"
> plot(eigensystem, params)
```

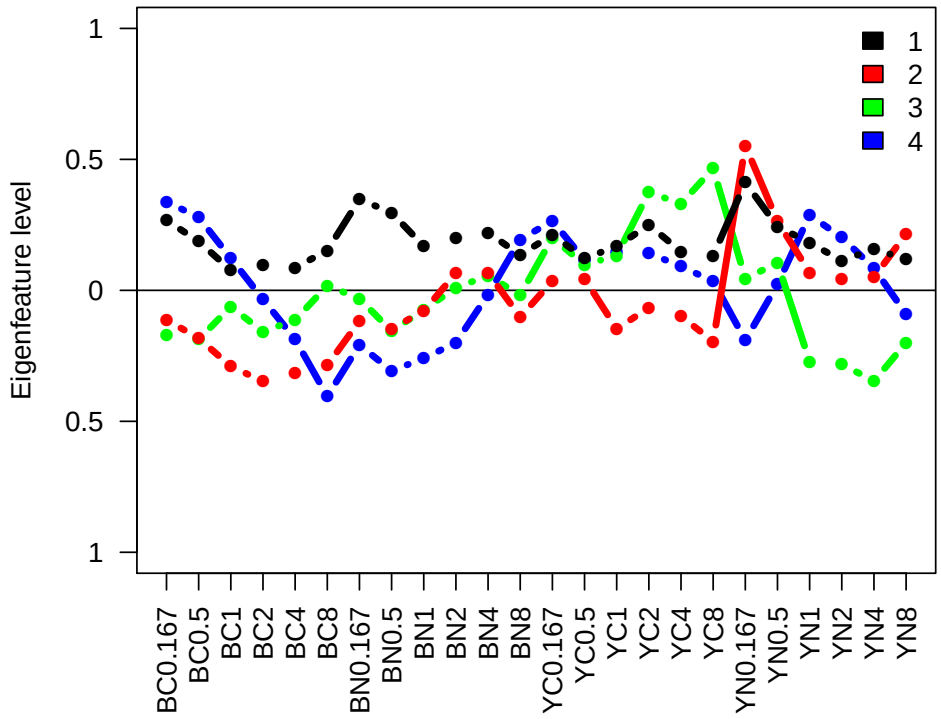



```
> plots(params) <- "allLines"
> plot(eigensystem, params)
> eigensystem <- exclude(eigensystem,excludeEigenfeature=c(1,11,12,14:24))
```



After removal of noise and the organism- and nutrient-inspecific starvation response, we investigated the variance in the data. Contrary to the two previous case studies, no steady-scale variance was present in the data and therefore no eigenfeatures were removed at the variance level.

```
> eigensystem <- compute(eigensystem, apply='variance')
> plots(params) <- "lines"
> plot(eigensystem, params)
> eigensystem <- exclude(eigensystem, excludeEigenfeatures=0)
```



Finally, the summary txt file and sorted metabolite x assay heatmap is generated, using the default eigenfeatures 1 and 2 for the calculation of cartesian and polar coordinates. It is also possible to change those settings, and obtain results for eigenfeatures 3 and 4. For the metabolite x assay heatmap, species information was used for the coloring of the assays. This heatmap allows determining the organism- and nutrient-specific metabolites.

```
> report(eigensystem, params)
> plots(params) <- "sortedHeatmap"
> assayColorMap(params) <- list(Species=NA)
> plot(eigensystem, params)
> whichPolarAxes(params) <- c(4,3)
> prefix(params) <- "StarvationData34"
> report(eigensystem, params)
```

5 Session Info

These analyses were done using the following versions of R, the operating system, and add-on packages:

- R Under development (unstable) (2019-10-24 r77329), `x86_64-pc-linux-gnu`

- Running under: `Ubuntu 18.04.3 LTS`
- Random number generation:
- RNG: `Mersenne-Twister`
- Normal: `Inversion`
- Sample: `Rounding`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.11-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.11-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, utils
- Other packages: Biobase 2.47.0, BiocGenerics 0.33.0, bigmemory 4.5.33, biosvd 2.23.0
- Loaded via a namespace (and not attached): NMF 0.21.0, R6 2.4.0, RColorBrewer 1.1-2, Rcpp 1.0.2, assertthat 0.2.1, bibtex 0.4.2, bigmemory.sri 0.1.3, cluster 2.1.0, codetools 0.2-16, colorspace 1.4-1, compiler 4.0.0, crayon 1.3.4, digest 0.6.22, doParallel 1.0.15, dplyr 0.8.3, foreach 1.4.7, ggplot2 3.2.1, glue 1.3.1, grid 4.0.0, gridBase 0.4-7, gtable 0.3.0, iterators 1.0.12, lazyeval 0.2.2, magrittr 1.5, munsell 0.5.0, pillar 1.4.2, pkgconfig 2.0.3, pkgmaker 0.27, plyr 1.8.4, purrr 0.3.3, registry 0.5-1, reshape2 1.4.3, rlang 0.4.1, rngtools 1.4, scales 1.0.0, stringi 1.4.3, stringr 1.4.0, tibble 2.1.3, tidyselect 0.2.5, tools 4.0.0, withr 2.1.2, xtable 1.8-4

References

[1] Alter O, Brown PO, and Botstein D. *Singular value decomposition for genome-wide expression data processing and modeling*. Proc Nat Acad Sci U.S.A. 97(18), 10101-10106 (2000).

[2] Spellman PT, Sherlock G, Zhang MQ, Iyer VR, Anders K, Eisen MB, Brown PO, Botstein D, and Futcher B. *Comprehensive identification of cell cycle-regulated genes of the Yeast Saccharomyces cerevisiae by microarray hybridization*. Mol Biol Cell 9, 3273-3297 (1998).

[3] Whitfield ML, Sherlock G, Saldanha AJ, Murray JI, Ball CA, Alexander KE, Matese JC, Perou CM, Hurt MM, Brown PO, and Botstein D. *Identification of genes periodically expressed in the human cell cycle and their expression in tumors*. Mol Biol Cell 13, 1977-2000 (2002).

[4] Brauer MJ, Yuan J, Bennett BD, Lu W, Kimball E, Botstein D, and Rabinowitz JD. *Conservation of the metabolomic response to starvation across two divergent microbes*. Proc Nat Acad Sci U.S.A. 103(51), 19302-19307 (2006).