Package 'isa2'

October 13, 2022

Version 0.3.5

Title The Iterative Signature Algorithm

Author Gabor Csardi <`csardi.gabor@gmail.com`>

Maintainer Gabor Csardi <`csardi.gabor@gmail.com`>

Description The ISA is a biclustering algorithm that finds modules in an input matrix. A module or bicluster is a block of the reordered input matrix.

Imports grDevices, lattice, stats, utils

Depends methods

Suggests igraph (>= 0.5.5), biclust

License CC BY-NC-SA 4.0

URL `https://github.com/gaborcsardi/ISA`

BugReports `https://github.com/gaborcsardi/ISA/issues`

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-03-02 08:09:30

R topics documented:

`isa2-package` *The isa package*

Description

The Iterative Signature Algorithm

Introduction

The Iterative Signature Algorithm (ISA) is a biclustering algorithm. Biclustering algorithms classify simultaneously the rows and columns of an input matrix into biclusters, or as we will call them here, modules.

For the impatient

The easiest way to run ISA is to call the `isa` function with your input matrix as the single argument. This does all steps of a typical ISA work flow, with the default parameters.

ISA biclusters

An ISA module is pair; a subset of the rows of the input matrix and a subset of its columns. In other words, a bicluster is a block of the reordered input matrix, where reordering means a permutation of both the rows and columns. (Another bicluster might be block of the same permuted input matrix or one after a different permutation.)

The criteria of a good bicluster is that 1) its rows are significantly different than the other rows, when we consider only the positions defined by the columns of the same bicluster, and (symmetrically) 2) its columns are significantly different than the other columns, when we consider only the positions defined by the rows of the same bicluster.

In other words, the rows of the bicluster are correlated, but only on the columns defined by the same bicluster; and the opposite is also true, the columns of the bicluster are correlated, but only on the rows defined by the same bicluster.

ISA biclusters are soft, two biclusters may overlap in their rows, columns or even both. It is also possible that some rows and/or columns of the input matrix are not found to be part of any ISA biclusters. Depending on the stringency parameters, it might even happen that ISA does not find any biclusters.

ISA row and column scores

ISA biclusters are not only soft, but every row and column in a given bicluster has a score, a number between minus one and one. The further this number is from zero, then stronger is the association of the given row or column to the bicluster.

How ISA works

ISA works in an iterative way. For an $E(m \times n)$ input matrix it starts from seed vector r_0, which is typically a sparse 0/1 vector of length m. This defines a set of rows in E. Then E' is multiplied by r_0 and the result is thresholded. (Please see also 'Normalization' below.)

The thresholding is an important step of the ISA, without thresholding ISA would be equivalent to a (not too effective) numerical singular value decomposition (SVD). Currently thresholding is done by calculating the mean and standard deviation of the vector and keeping only elements that are further than a given number of standard deviations from the mean. Based on the `direction` parameter, this means 1) keeping values that are significantly higher than the mean (`direction="up"`), significantly lower (`direction="down"`) or both (`direction="updown"`).

The thresholded vector c_0 is the (column) 'signature' of r_0. Then the (row) signature of c_0 is calculated, E is multiplied by c_0 and then thresholded to get r_1.

This iteration is performed until it converges, i.e. r_i and r_{i-1} are "close", and c_i and c_{i-1} are also close. The convergence criteria, i.e. what "close" means is by default defined by high Pearson correlation.

It is very possible that the ISA finds the same modules more than once; two or more seeds might converge to the same module. The function `isa.unique` eliminates every module from the result of `isa.iterate` that is very similar (in terms of Pearson correlation) to the one that was already found before it.

Parameters

The two main parameters of ISA are the two thresholds (one for the rows and one for the columns). They basically define the stringency of the modules. If the row threshold is high, then the modules will have very similar rows. If it is mild, then modules will be bigger, with less similar rows than in the first case.

Random seeding and smart seeding

By default (i.e. if the `isa` function is used) the ISA is performed from random sparse starting seeds, generated by `generate.seeds`. This way the algorithm is completely unsupervised, but also stochastic: it might give different results for different runs.

It is possible to use non-random seeds as well, if you have some knowledge about the data or are interested in a particular subset of rows/columns, then you can feed in your seeds into the `isa.iterate` function directly. In this case the algorithm is deterministic, for the same seed you will always get the same results.

Normalization

On in silico data we observed that ISA has the best performance if the input matrix is normalized (see `isa.normalize`). The normalization produces two matrices: E_r and E_c. E_r is calculated by transposing E and centering and scaling its rows (see `scale`). E_c is calculated by centering and scaling the rows of E. E_r is used to calculate the column signature of rows and E_c is used to calculate the signature of the columns.

It is possible to use another normalization, then the user is requested to supply the normalized input data in a named list, including the two matrices of appropriate dimensions. '`Er`' will be used for calculating the signature of the rows, 'Ec' the signature of the columns. If you want to use the same matrix in both steps, then supply it twice, the first one transposed.

Robustness

As ISA is an unsupervised algorithm, it may very well find some modules, even if you feed in noise as an input matrix. To avoid these spurious modules we defined a robustness measure, a single number for a modules that gives how well the rows and the columns are correlated.

It recommended that the user uses `isa.filter.robust` to run ISA on the scrambled input matrix with the same threshold parameters and then drop every module, which has a robustness score lower than the highest robustness score among modules found in the scrambled data.

A typical ISA work flow

Please see the manual page and the source code of `isa` for a typical ISA work flow. (You can obtain the source code by typing '`isa`' (without the apostrophes) into your R prompt and pressing ENTER.)

Author(s)

Gabor Csardi <`Gabor.Csardi@unil.ch`>

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

Ihmels J, Friedlander G, Bergmann S, Sarig O, Ziv Y, Barkai N: Revealing modular organization in the yeast transcriptional network *Nat Genet.* 2002 Aug;31(4):370-7. Epub 2002 Jul 22

Ihmels J, Bergmann S, Barkai N: Defining transcription modules using large-scale gene expression data *Bioinformatics* 2004 Sep 1;20(13):1993-2003. Epub 2004 Mar 25.

See Also

The vignette in the package and `isa` for running ISA.

`generate.seeds` *Generate seed vectors for the Iterative Signature Algorithm*

Description

Generate random input seeds for the ISA.

Usage

```
generate.seeds (length, count = 100, method = c("uni"), sparsity=2)
```

Arguments

`length`	The length of the seeds, should be the number of rows in your input data for row seeds and the number of columns for column seeds.
`count`	The number of seeds to gnerate.
`method`	The method for generating the seeds. Currently only `"uni"` is supported, that picks the 1 elements in each seed uniformly randomly.
`sparsity`	A numeric scalar, an integer number giving the number of non-zero values in each seed vector. It will be recycled to have the same length as the number of seeds.

Details

This function can generate a 0/1 matrix whose columns are the seeds of the ISA. The result can be use as the `row.seeds` (or `col.seeds`) argument of the `isa.iterate` function.

Value

A numeric matrix with 0/1 values.

Author(s)

Gabor Csardi `<Gabor.Csardi@unil.ch>`

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

Ihmels J, Friedlander G, Bergmann S, Sarig O, Ziv Y, Barkai N: Revealing modular organization in the yeast transcriptional network *Nat Genet.* 2002 Aug;31(4):370-7. Epub 2002 Jul 22

Ihmels J, Bergmann S, Barkai N: Defining transcription modules using large-scale gene expression data *Bioinformatics* 2004 Sep 1;20(13):1993-2003. Epub 2004 Mar 25.

See Also

isa2-package for a short introduction on the Iterative Signature Algorithm. See `isa` for an easy way of running ISA.

Examples

```
## Just to get always the same result
set.seed(24)

## Create some random seeds with different sparseness
data <- isa.in.silico()
sparsity <- rep( c(1,5,25,125), length=100)
row.seeds <- generate.seeds(length=nrow(data[[1]]), count=100,
                            sparsity=sparsity)
```

```
## Do ISA with the seeds
normed.data <- isa.normalize(data[[1]])
isaresult <- isa.iterate(normed.data, thr.row=1, thr.col=1,
                         row.seeds=row.seeds)

## Add the sparsity to the seed data
isaresult$seeddata$sparsity <- sparsity

## Check which ones leed to higher robustness scores
rob <- robustness(normed.data, isaresult$rows, isaresult$columns)
tapply(rob, sparsity, mean)
## About the same

## How many unique modules did we find for the different sparsity
isaresult.unique <- isa.unique(normed.data, isaresult)
tapply(seq_len(ncol(isaresult.unique$rows)),
       isaresult.unique$seeddata$sparsity, length)
## We usually find more modules with sparser seeds
```

`isa` *Iterative Signature Algorithm*

Description

Run ISA with the default parameters

Usage

```
## S4 method for signature 'matrix'
isa(data, ...)
```

Arguments

`data`	The input. It must be a numeric matrix. It may contain `NA` and/or `NaN` values, but then the algorithm might be a bit slower, as R matrix multiplication might be slower for these matrices, depending on your platform.
`...`	Additional arguments, see details below.

Details

Please read the isa2-package manual page for an introduction on ISA.

This function can be called as

```
    isa(data, thr.row=seq(1,3,by=0.5),
        thr.col=seq(1,3,by=0.5), no.seeds=100,
        direction=c("updown", "updown"))
```

where the arguments are:

data The input. It must be a numeric matrix. It may contain `NA` and/or `NaN` values, but then the algorithm might be a bit slower, as R matrix multiplication might be slower for these matrices, depending on your platform.

thr.row Numeric vector. The row threshold parameters for which the ISA will be run. We use all possible combinations of `thr.row` and `thr.col`.

thr.col Numeric vector. The column threshold parameters for which the ISA will be run. We use all possible combinations of `thr.row` and `thr.col`.

no.seeds Integer scalar, the number of seeds to use.

direction Character vector of length two, one for the rows, one for the columns. It specifies whether we are interested in rows/columns that are higher (‘`up`’) than average, lower than average (‘`down`’), or both (‘`updown`’).

The `isa` function provides an easy to use interface to the ISA. It runs all steps of a typical ISA work flow with their default parameters.

This involves:

1. Normalizing the data by calling `isa.normalize`.
2. Generating random input seeds via `generate.seeds`.
3. Running ISA with all combinations of given row and column thresholds, (by default 1, 1.5, 2, 2.5, 3); by calling `isa.iterate`.
4. Merging similar modules, separately for each threshold combination, by calling `isa.unique`.
5. Filtering the modules separately for each threshold combination, by calling `isa.filter.robust`.
6. Putting all modules from the runs with different thresholds into a single object.
7. Merging similar modules, across all threshold combinations, if two modules are similar, then the larger one, the one with the milder thresholds is kept.

Please see the manual pages of these functions for the details or if you want to change their default parameters.

Value

A named list is returned with the following elements:

`rows`	The row components in the biclusters, a numeric matrix. Every column in it corresponds to a bicluster, if an element (the score of the row) is non-zero, that means that the row is included in the bicluster, otherwise it is not. Scores are between -1 and 1. If the scores of two rows have the same (nonzero) sign, that means that the two corresponding rows “behave” the same way. If they have opposite sign, that means that they behave the opposite way. If the corresponding seed has not converged during the allowed number of iterations, then that column of `rows` contains `NA` values.
`columns`	The column components of the biclusters, in the same format as the rows. If the corresponding seed has not converged during the allowed number of iterations, then that column of `columns` contains `NA` values.

`seeddata` A data frame containing information about the biclusters. There is one row for each bicluster. The data frame has the following columns:

- `iterations` The number of iterations needed to converge to the bicluster.
- `oscillation` The oscillation period for oscillating biclusters. It is zero for non-oscillating ones.
- `thr.row` The row threshold that was used for find the bicluster.
- `thr.col` The column threshold that was used for finding the bicluster.
- `freq` The number of times the bicluster was found.
- `rob` The robustness score of the bicluster, see `robustness` for details.
- `rob.limit` The robustness limit that was used to filter the module. See `isa.filter.robust` for details.

`rundata` A named list with information about the ISA runs. It has the following entries:

- `direction` Character vector of length two. Specifies which side(s) of the score distribution were kept in each ISA step. See the `direction` argument of `isa.iterate` for details.
- `convergence` Character scalar. The convergence criteria for the iteration. See the `convergence` argument of `isa.iterate` for details.
- `eps` Numeric scalar. The threshold for convergence, if the ‘eps’ convergence criteria was used.
- `cor.limit` Numeric scalar. The threshold for convergence, if the ‘cor’ convergence criteria was used.
- `corx` Numeric scalar, the shift in number of iterations, to check convergence. See the `convergence` argument of `isa.iterate` for details.
- `maxiter` Numeric scalar. The maximum number of iterations that were allowed for an input seed.
- `N` Numeric scalar. The total number of seeds that were used for all the thresholds.
- `prenormalize` Logical scalar. Whether the data was pre-normalized.
- `hasNA` Logical scalar. Whether the (normalized) data had `NA` or `NaN` values.
- `unique` Logical scalar. Whether the similar biclusters were merged by calling `isa.unique`.
- `oscillation` Logical scalar. Whether the algorithm looked for oscillating modules as well.
- `rob.perms` Numeric scalar, the number of permutations that were used to calculate the baseline robustness for filtering. See the `perms` argument of the `isa.filter.robust` function for details.

Author(s)

Gabor Csardi <`Gabor.Csardi@unil.ch`>

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

Ihmels J, Friedlander G, Bergmann S, Sarig O, Ziv Y, Barkai N: Revealing modular organization in the yeast transcriptional network *Nat Genet.* 2002 Aug;31(4):370-7. Epub 2002 Jul 22

Ihmels J, Bergmann S, Barkai N: Defining transcription modules using large-scale gene expression data *Bioinformatics* 2004 Sep 1;20(13):1993-2003. Epub 2004 Mar 25.

See Also

isa2-package for a short introduction on the Iterative Signature Algorithm. See the functions mentioned above if you want to change the default ISA parameters.

Examples

```
## Not run:
## We generate some noisy in-silico data with modules and try to find
## them with the ISA. This might take one or two minutes.
data <- isa.in.silico(noise=0.1)
isa.result <- isa(data[[1]])

## Find the best bicluster for each block in the input
best <- apply(cor(isa.result$rows, data[[2]]), 2, which.max)

## Check correlation
sapply(seq_along(best),
       function(x) cor(isa.result$rows[,best[x]], data[[2]][,x]))

## The same for the columns
sapply(seq_along(best),
       function(x) cor(isa.result$columns[,best[x]], data[[3]][,x]))

## Plot the data and the modules found
if (interactive()) {
  layout(rbind(1:2,3:4))
  image(data[[1]], main="In-silico data")
  sapply(best, function(b) image(outer(isa.result$rows[,b],
                                       isa.result$columns[,b]),
                                 main=paste("Module", b)))
}

## End(Not run)
```

---

isa.biclust *Convert ISA modules to a Biclust class, as defined by the biclust package*

Description

This function converts the object with ISA modules to a `Biclust` object, so all the functions in the `biclust` package can be used on it.

Usage

```
isa.biclust(modules)
```

Arguments

`modules`	The ISA modules, as returned by the `isa` or some other function.

Details

`biclust` is an R package that implements many biclustering algorithms in a unified framework. This function converts a set of ISA biclusters to a `Biclust` object, this class is used to store all biclustering results by the `biclust` package.

The `Biclust` class only supports binary biclusters, so the ISA modules are binarized during the conversion.

Value

A `Biclust` object.

Author(s)

Gabor Csardi <`Gabor.Csardi@unil.ch`>

Examples

```
## You need the biclust package for this
## Not run:
if (require(biclust)) {
  set.seed(1)
  data <- isa.in.silico()
  modules <- isa(data[[1]])
  bc <- isa.biclust(modules)

  ## A heatmap
  drawHeatmap(data[[1]], bc, 1)

  ## A "bubble" plot
  bubbleplot(data[[1]], bc)

  ## Compare values inside and outside the bicluster
  plotclust(bc, data[[1]])

  ## Plot profiles of bicluster elements
  parallelCoordinates(data[[1]], bc, number=1)

  ## Coherence measures vs. ISA robustness
  cV <- sapply(seq(bc@Number), function(x)
     constantVariance(data[[1]], bc, x, dimension="both"))
  aV <- sapply(seq(bc@Number), function(x)
     additiveVariance(data[[1]], bc, x, dimension="both"))
  mV <- sapply(seq(bc@Number), function(x)
     multiplicativeVariance(data[[1]], bc, x, dimension="both"))
  sV <- sapply(seq(bc@Number), function(x)
     signVariance(data[[1]], bc, x, dimension="both"))

  rob <- robustness(isa.normalize(data[[1]]), modules$rows,
     modules$columns)

  cor( cbind(cV, aV, mV, sV, rob) )
}

## End(Not run)
```

`isa.in.silico` *Generate in-silico input data for biclustering algorithms*

Description

This function generates a test data set for ISA, containing modules of prescribed number, size, signal level, internal noise and background noise.

Usage

```
isa.in.silico (num.rows = 300, num.cols = 50, num.fact = 3,
     mod.row.size = round(0.5 * num.rows/num.fact),
     mod.col.size = round(0.5 * num.cols/num.fact), noise = 0.1,
     mod.signal = rep(1, num.fact), mod.noise = rep(0, num.fact),
     overlap.row = 0, overlap.col = overlap.row)
```

Arguments

`num.rows`	The number of rows in the data matrix.
`num.cols`	The number of columns in the data matrix.
`num.fact`	The number of modules to put into the data.
`mod.row.size`	The size of the modules, the number of rows per module. It can be a scalar or a vector and it is recycled.
`mod.col.size`	The size of the modules, the number of columns per module. It can be a scalar or a vector and it is recycled.
`noise`	The level of the background noise to be added to the data matrix. It gives the standard deviation of the normal distribution from which the noise is generated.
`mod.signal`	The signal level of the modules.
`mod.noise`	The noise levels of the different modules. Normally distributed noise with standard deviation `mod.noise` is added to the data. This is in addition to the background noise.
`overlap.row`	The overlap of the modules, for the rows. Zero means no overlap, one means one overlapping row, etc.
`overlap.col`	The overlap of the modules, for the columns. Zero means no overlap, one means one overlapping column, etc.

Details

`isa.in.silico` creates an artificial data set to test the ISA or any other biclustering algorithm. It creates a data matrix with a checkerboard matrix. In other words, potentially overlapping blocks are planted into a noisy background matrix.

These blocks may have different signal and noise levels and they might also overlap. See the parameters above.

Value

A list with three matrices. The first matrix is the in silico data, the second contains the rows of the correct modules, the third the columns.

Author(s)

Gabor Csardi <`Gabor.Csardi@unil.ch`>

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

Ihmels J, Friedlander G, Bergmann S, Sarig O, Ziv Y, Barkai N: Revealing modular organization in the yeast transcriptional network *Nat Genet.* 2002 Aug;31(4):370-7. Epub 2002 Jul 22

Ihmels J, Bergmann S, Barkai N: Defining transcription modules using large-scale gene expression data *Bioinformatics* 2004 Sep 1;20(13):1993-2003. Epub 2004 Mar 25.

See Also

isa2-package for a short introduction on the Iterative Signature Algorithm. See `isa` for an easy way of running ISA.

Examples

```
## Define a function for plotting if we are interactive
if (interactive()) { layout( rbind(1:2,3:4) ) }
myimage <- function(mat) {
  if (interactive()) { par(mar=c(1,2,2,1)); image(mat[[1]]) }
}

## Create a simple checkerboard without overlap and noise
silico1 <- isa.in.silico(100, 100, 10, mod.row.size=10, mod.col.size=10,
                         noise=0)
myimage(silico1)

## The same, but with some overlap and noise
silico2 <- isa.in.silico(100, 100, 10, mod.row.size=10, mod.col.size=10,
                         noise=0.1, overlap.row=3)
myimage(silico2)
```

```
## Modules with different noise levels
silico3 <- isa.in.silico(100, 100, 5, mod.row.size=10, mod.col.size=10,
                         noise=0.01, mod.noise=seq(0.1,by=0.1,length=5))
myimage(silico3)

## Modules with different signal levels
silico4 <- isa.in.silico(100, 100, 5, mod.row.size=10, mod.col.size=10,
                         noise=0.01, mod.signal=seq(1,5,length=5))
myimage(silico4)
```

isa.iterate *The Iterative Signature Algorithm*

Description

Perform ISA on the (normalized) input matrix.

Usage

```
## S4 method for signature 'list'
isa.iterate(normed.data, ...)
```

Arguments

`normed.data`	The normalized data. A list of two matrices, usually coming from `isa.normalize`.
`...`	Additional arguments, see details below.

Details

`isa.iterate` performs the ISA iteration on the specified input seeds. It can be called as

```
    isa.iterate(normed.data, row.seeds, col.seeds,
                thr.row, thr.col = thr.row,
direction = c("updown", "updown"),
convergence = c("corx", "cor", "eps"),
cor.limit = 0.99, eps = 1e-04, corx=3,
oscillation = FALSE, maxiter = 100)
```

where the arguments are:

normed.data The normalized data. A list of two matrices, usually coming from `isa.normalize`.

row.seeds The row seed vectors to start the ISA runs from. Every column is a seed vector. (If this argument and `col.seeds` are both present, then both of them are used.)

col.seeds The column seed vectors to start the ISA runs from, every column is a seed vector. (If this argument and `row.seeds` are both present, then both of them are used.)

thr.row Numeric scalar or vector giving the threshold parameter for the rows. Higher values indicate a more stringent threshold and the result biclusters will contain less rows on average. The threshold is measured by the number of standard deviations from the mean, over the values of the row vector. If it is a vector then it must contain an entry for each seed.

thr.col Numeric scalar or vector giving the threshold parameter for the columns. The analogue of `thr.row`.

direction Character vector of length two, one for the rows, one for the columns. It specifies whether we are interested in rows/columns that are higher (‘`up`’) than average, lower than average (‘`down`’), or both (‘`updown`’).

convergence Character scalar, the convergence criteria for the ISA iteration. If it is ‘`cor`’, then convergence is measured based on high Pearson correlation (see the `cor.limit` argument below) of the subsequent row and column vectors. If it is ‘`eps`’, then all entries of the subsequent row and column vectors must be close to each other, see the `eps` argument below.

‘`corx`’ is similar to ‘`cor`’, but the current row/column vectors are compared to the ones `corx` steps ago, instead of the ones in the previous step. See the `corx` argument below, that gives the size of the shift.

cor.limit The correlation limit for convergence if the ‘`cor`’ method is used.

eps Limit for convergence if the ‘`eps`’ method is used.

corx The number of iterations to use as a shift, for checking convergence with the ‘`corx`’ method.

oscillation Logical scalar, whether to look for oscillating seeds. Usually there are not too many oscillating seeds, so it is safe to leave this on `FALSE`.

maxiter The maximum number of iterations allowed.

Value

A named list with many components. Please see the manual page of `isa` for a complete description.

Author(s)

Gabor Csardi <`Gabor.Csardi@unil.ch`>

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

Ihmels J, Friedlander G, Bergmann S, Sarig O, Ziv Y, Barkai N: Revealing modular organization in the yeast transcriptional network *Nat Genet.* 2002 Aug;31(4):370-7. Epub 2002 Jul 22

Ihmels J, Bergmann S, Barkai N: Defining transcription modules using large-scale gene expression data *Bioinformatics* 2004 Sep 1;20(13):1993-2003. Epub 2004 Mar 25.

See Also

isa2-package for a short introduction on the Iterative Signature Algorithm. See `isa` for an easy way of running ISA.

Examples

```
## A basic ISA work flow for a single threshold combination
## In-silico data
set.seed(1)
insili <- isa.in.silico()

## Random seeds
seeds <- generate.seeds(length=nrow(insili[[1]]), count=100)

## Normalize input matrix
nm <- isa.normalize(insili[[1]])

## Do ISA
isares <- isa.iterate(nm, row.seeds=seeds, thr.row=2, thr.col=1)

## Eliminate duplicates
isares <- isa.unique(nm, isares)

## Filter out not robust ones
isares <- isa.filter.robust(insili[[1]], nm, isares)

## Print the sizes of the modules
cbind( colSums(isares$rows!=0), colSums(isares$columns!=0) )

## Plot the original data and the modules found
if (interactive()) {
  layout(rbind(1:2))
  image(insili[[1]], main="In silico data")
  image(outer(isares$rows[,1],isares$columns[,1])+
        outer(isares$rows[,2],isares$columns[,2])+
        outer(isares$rows[,3],isares$columns[,3]), main="ISA modules")
}
```

---

isa.normalize *Normalize input data for use with ISA*

Description

Normalize a matrix and create a form that can be effectively used for ISA runs.

Usage

```
## S4 method for signature 'matrix'
isa.normalize(data, ...)
```

Arguments

`data`	A numeric matrix, the input data. It might contains `NA` and/or `NaN` values.
`...`	Additional arguments, see details below.

Details

This function can be called as

```
isa.normalize(data, prenormalize = FALSE)
```

where the arguments are:

data A numeric matrix, the input data. It might contains `NA` and/or `NaN` values.

prenormalize Logical scalar, see details below.

It was observed that the ISA works better if the input matrix is scaled and its rows have mean zero and standard deviation one.

An ISA step consists of two sub-steps, and this implies two different normalizations, in the first the rows, in the second the columns of the input matrix will be scaled.

If the `prenormalize` argument is set to `TRUE`, then row-wise scaling is calculated on the column-wise scaled matrix and not on the input matrix directly.

Value

A list of two normalized matrices, the first one is transposed.

Author(s)

Gabor Csardi <`Gabor.Csardi@unil.ch`>

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

Ihmels J, Friedlander G, Bergmann S, Sarig O, Ziv Y, Barkai N: Revealing modular organization in the yeast transcriptional network *Nat Genet.* 2002 Aug;31(4):370-7. Epub 2002 Jul 22

Ihmels J, Bergmann S, Barkai N: Defining transcription modules using large-scale gene expression data *Bioinformatics* 2004 Sep 1;20(13):1993-2003. Epub 2004 Mar 25.

See Also

isa2-package for a short introduction on the Iterative Signature Algorithm. See `isa` for an easy way of running ISA.

Examples

```
## In-silico data
set.seed(1)
insili <- isa.in.silico()
nm <- isa.normalize(insili[[1]])

## Check it
```

```
max(abs(rowSums(nm[[1]])))
max(abs(rowSums(nm[[2]])))

max(abs(apply(t(nm[[1]]), 2, sd) - 1))
max(abs(apply(t(nm[[2]]), 2, sd) - 1))

## Plot them
if (interactive()) {
  layout(rbind(1:2,3:4))
  image(insili[[1]], main="Original data")
  image(t(nm[[1]]), main="Row normalized")
  image(nm[[2]], main="Column normalized")
}
```

isa.option *Options for the isa package*

Description

This function can be used to set various options that affect many functions in the isa package.

Usage

```
isa.option(...)
```

Arguments

`...`	A single option query, or option assignments, these must be named, too. See details below.

Details

The `isa.option` function can be used in three forms. First, calling it without any arguments returns a named list of the current values of all isa options.

Second, calling it with a character scalar as the single argument, it returns the value of the specified option.

Third, calling it with a named argument (or more named arguments) set the specified options to the given values.

Here is a list of all the currently supported options:

`verbose` Logical scalar. Whether to report what the isa functions are currently doing. Defaults to `FALSE`.

`status.function` A function object, it serves as a callback for printing status messages.

Value

In the first form, `isa.option` returns a named list with the current values of all options.

In the second form, it returns the value of the specified option.

In the third form, it returns a named list with the current values of all options, invisibly.

Author(s)

Gabor Csardi <`Gabor.Csardi@unil.ch`>

Examples

```
## Make isa functions verbose
isa.option(verbose=TRUE)

## Query the value of 'verbose'
isa.option("verbose")

## Query all options
isa.option()
```

`isa.sweep` *Create a hierarchical structure of ISA biclusters*

Description

Relate the biclusters found in many ISA runs on the same input data.

Usage

```
## S4 method for signature 'matrix'
isa.sweep(data, ...)
## S4 method for signature 'list'
sweep.graph(sweep.result, ...)
```

Arguments

`data`	The input matrix.
`...`	Additional arguments, see details below. `sweep.graph` has no additional arguments currently.
`sweep.result`	An ISA result with hierarchy information in the seed data, typically calculated by the `isa.sweep` function.

Details

`isa.sweep` can be called as

```
    isa.sweep(data, normed.data, isaresult,
              method = c("cor"), neg.cor = TRUE,
      cor.limit = 0.9)
```

where the arguments are:

data The input matrix.

normed.data The normalized input matrix, usually the output of the `isa.normalize` function.

isaresult An object containing the biclusters, the result of `isa` or `isa.iterate`.

method Character scalar giving the method to determine which seed converged which bicluster. Right now only '`cor`' is supported, this is based on Pearson correlation.

neg.cor Logical scalar, whether to consider negative correlation as convergence.

cor.limit Numeric scalar giving the minimum correlation for convergence.

Many ISA runs with different thresholds typically create a bunch of biclusters and it is useful to visualize how these are related.

From a set of biclusters for which of the `thr.row` and `thr.col` parameters was the same, but the other was not, `isa.sweep` creates a hierarchy of modules.

The hierarchy is a directed graph of modules in which every node has an out degree at most one. An edge pointing from module m to module n means that module n is "part of" module m; in the sense that an ISA iteration started from module n converges to module m at the (milder) thresholds of module m.

The information about the module relationships is stored in a column of the seed data.

`sweep.graph` takes the output of `isa.sweep` and creates a graph object of it. For this the 'igraph' package is required to be installed on the system.

Value

`isa.sweep` returns a named list with the same components as in the input (`isaresult`), but the '`father`' and the '`level`' columns are added to the '`seeddata`' member. `father` contains the edges of the sweep graph: if bicluster m is the father of bicluster n that means that bicluster n converges to bicluster m at the same threshold parameters that were used to find biclusters m.

`level` is a simple numbering of the different thresholds for which the sweep tree was built. I.e. the most strict threshold is level one, the second most is level two, etc.

`sweep.graph` returns and igraph graph with a lot of attributes:

1	The `layout` graph attribute contains a two-column matrix with the coordinates for an optimal tree-like layout to plot the graph.
2	The `width` and `height` graph attributes contain the optimal width and height of the plot, in inches.
3	The `thr` vertex attribute contains the ISA threshold that varies along the sweeping.

4	The `id` vertex attribute contains the id of the module, these correspond to the indices in the result matrix.
5	The `rows` and `cols` vertex attributes contain the number of rows and columns in the module.
6	The `shape`, `size`, `size2`, `label` vertex attributes and the `arrow.size` edge attribute contain various graphical parameters.

Author(s)

Gabor Csardi <`Gabor.Csardi@unil.ch`>

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

Ihmels J, Friedlander G, Bergmann S, Sarig O, Ziv Y, Barkai N: Revealing modular organization in the yeast transcriptional network *Nat Genet.* 2002 Aug;31(4):370-7. Epub 2002 Jul 22

Ihmels J, Bergmann S, Barkai N: Defining transcription modules using large-scale gene expression data *Bioinformatics* 2004 Sep 1;20(13):1993-2003. Epub 2004 Mar 25.

See Also

isa2-package for a short introduction on the Iterative Signature Algorithm. See `isa` for an easy way of running ISA.

Examples

```
## In-silico data
set.seed(1)
insili <- isa.in.silico()

## Do ISA with a bunch of row thresholds while keeping the column
## threshold fixed. This is quite an artificial example...
isares <- isa(insili[[1]], thr.row=c(0.5,1,2), thr.col=0)

## Create a nice tree from the modules, we need the normed data for this
nm <- isa.normalize(insili[[1]])
isa.tree <- isa.sweep(insili[[1]], nm, isares)
network <- sweep.graph(isa.tree)

## Plot the network of modules, only if the igraph package is
## installed
if (interactive() && require(igraph) &&
    compareVersion(packageDescription("igraph")$Version, "0.6")>=0) {
  lab <- paste(sep="", seq_len(ncol(isa.tree$rows)), ": ",
               colSums(isa.tree$rows!=0), ",",
               colSums(isa.tree$columns!=0))
  par(mar=c(1,1,1,1))
  roots <- tapply(topological.sort(network, mode="out"),
                  clusters(network)$membership, function(x) x[1])
  rootlevel <- isa.tree$seeddata$level-1
  coords <- layout.reingold.tilford(network, root=roots,
                                    rootlevel=rootlevel[roots+1])
  plot(network, layout=coords,
       vertex.shape="rectangle", vertex.color="green",
       vertex.label=lab, vertex.size=30, vertex.size2=10)
}

## Plot the modules themselves as well
if (interactive()) {
  plotModules(isa.tree)
}

## Yet another plot, the scores for the rows within the modules
if (interactive()) {
  layout(matrix( 1:15, ncol=3 ))
  for (i in seq(ncol(isa.tree$rows))) {
    par(mar=c(2,2,1,1))
    plot(isa.tree$rows[,i], axes=FALSE, ylim=c(-1,1))
    axis(1); axis(2)
    text(nrow(isa.tree$rows), 1, adj=c(1,1), paste(sep="", "#", i), cex=2)
  }
}
```

isa.unique *Filter out biclusters that are very similar to each other*

Description

From a potentially non-unique set of ISA biclusters, create a unique set by removing all biclusters that are similar to others.

Usage

```
## S4 method for signature 'list,list'
 isa.unique(normed.data, isaresult, ...)
```

Arguments

normed.data	The normalized input data, a list of two matrices, usually the output of isa.normalize.
isaresult	The result of an ISA run, a set of biclusters.
...	Additional arguments, see details below.

Details

This function can we called as

```
isa.unique(normed.data, isaresult, method = c("cor"),
           ignore.div = TRUE, cor.limit = 0.9,
   neg.cor = TRUE, drop.zero = TRUE)
```

where the arguments are:

normed.data The normalized input data, a list of two matrices, usually the output of `isa.normalize`.

isaresult The result of an ISA run, a set of biclusters.

method Character scalar giving the method to be used to determine if two biclusters are similar. Right now only '`cor`' is implemented, this keeps both biclusters if their Pearson correlation is less than `cor.limit`, both for their row and column scores. See also the `neg.cor` argument.

ignore.div Logical scalar, if `TRUE`, then the divergent biclusters will be removed.

cor.limit Numeric scalar, giving the correlation limit for the '`cor`' method.

neg.cor Logical scalar, if `TRUE`, then the '`cor`' method considers the absolute value of the correlation.

drop.zero Logical scalar, whether to drop biclusters that have all zero scores.

Because of the nature of the ISA algorithm, the set of biclusters created by `isa.iterate` is not unique; many input seeds may converge to the same biclusters, even if the input seeds are not random.

`isa.unique` filters a set of biclusters and removed the ones that are very similar to ones that were already found for another seed.

Value

A named list, the filtered `isaresult`. See the return value of `isa.iterate` for the details.

Author(s)

Gabor Csardi <`Gabor.Csardi@unil.ch`>

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

Ihmels J, Friedlander G, Bergmann S, Sarig O, Ziv Y, Barkai N: Revealing modular organization in the yeast transcriptional network *Nat Genet.* 2002 Aug;31(4):370-7. Epub 2002 Jul 22

Ihmels J, Bergmann S, Barkai N: Defining transcription modules using large-scale gene expression data *Bioinformatics* 2004 Sep 1;20(13):1993-2003. Epub 2004 Mar 25.

See Also

isa2-package for a short introduction on the Iterative Signature Algorithm. See `isa` for an easy way of running ISA.

Examples

```
## Create an ISA module set
set.seed(1)
insili <- isa.in.silico(noise=0.01)

## Random seeds
seeds <- generate.seeds(length=nrow(insili[[1]]), count=20)

## Normalize input matrix
nm <- isa.normalize(insili[[1]])

## Do ISA
isares <- isa.iterate(nm, row.seeds=seeds, thr.row=2, thr.col=1)

## Check correlation among modules
cc <- cor(isares$rows)
if (interactive()) { hist(cc[lower.tri(cc)],10) }

## Some of them are quite high, how many?
undiag <- function(x) { diag(x) <- 0; x }
sum(undiag(cc) > 0.99, na.rm=TRUE)

## Eliminate duplicated modules
isares.unique <- isa.unique(nm, isares)

## How many modules left?
ncol(isares.unique$rows)

## Double check
cc2 <- cor(isares.unique$rows)
if (interactive()) { hist(cc2[lower.tri(cc2)],10) }

## High correlation?
sum(undiag(cc2) > 0.99, na.rm=TRUE)
```

`plotModules` *Image plots of biclusters*

Description

Make several image plots, one for each bicluster, and optionally one for the original data as well.

Usage

```
images(matrices, names=NULL, ...)
## S4 method for signature 'list'
 plotModules(modules, ...)
```

Arguments

`matrices`	A list of matrices to plot. Please note that this argument is *always* interpreted as a list, even if want to plot a single matrix, put it into a list.
`names`	Character vector, the labels to show above the image plots. If you give the `data` argument to `plotModules`, then the first label corresponds to that.
`...`	Additional arguments, for `images` these are passed to `levelplot`, for `plotModules` see the details below.
`modules`	The object with the ISA modules, as returned by the `isa` function or other such functions.

Details

`images` creates image plots for a series of matrices, using `levelplot` from the `lattice` package.

`plotModules` calls `images` from the to create image plots for a set of modules. It can be called as

```
    plotModules(modules, to.plot=seq_len(ncol(modules$rows)),
                data, binary=TRUE, names=NULL, xlab="", ylab="",
\dots)
```

where the arguments are:

modules The object with the ISA modules, as returned by the `isa` function or other such functions.

to.plot Numeric vector, the modules to plot, the numbers correspond to the columns in `modules$rows` and `modules$columns`. By default all modules will be plot.

data An optional data matrix to plot. Most often this is the original data. If given, its dimension must much the dimensions in the `modules` object. If given, then this matrix is plotted first, before the modules.

binary Logical scalar, whether to binarize the biclusters before plotting or use the actual ISA scores. By default the biclusters are binarized.

names Character vector, the labels to show above the image plots. If you give the `data` argument to `plotModules`, then the first label corresponds to that.

xlab Character scalar, the label to put on the horizontal axis.

ylab Character scalar, the label to put on the vertical axis.

... Further arguments are passed to `levelplot`.

Note, that if you want to export these plots to a file, then a bitmap-based format might be more appropriate. For larger matrices vector formats tend to generate huge file because of the many dots.

Value

Since these function use the `lattice` package, they return an object of class `trellis`. You will need to `print` this object to create the actual plots.

Author(s)

Gabor Csardi <`Gabor.Csardi@unil.ch`>

See Also

`image` and the other version: `image` from the `Matrix` package, for alternatives to create image plots.

Examples

```
## The following should plot the input matrix and the four modules
## found by ISA
set.seed(1)   # to get same plot every time
data <- isa.in.silico(100, 100, num.fact=4)
modules <- isa(data[[1]], thr.row=2, thr.col=2)
plotModules(modules, data=data[[1]], binary=FALSE,
            names=c("Input matrix",
                    paste("Module", seq_len(ncol(modules$rows)))))
```

`ppa` *The Ping-Pong Algorithm*

Description

Run the PPA with the default parameters

Usage

```
## S4 method for signature 'list'
ppa(data, ...)
```

Arguments

`data`	The input, a list of two numeric matrices, with the same number of columns. They may contain `NA` and/or `NaN` values, but then the algorithm might get slower, as R matrix multiplication is slower sometimes slower for these matrices, depending on your platform.
`...`	Additional arguments, see details below.

Details

Please read the isa2-package manual page for and introductino on ISA and PPA.

This function can be called as

```
ppa(data, thr.row1 = seq(1, 3, by = 0.5),
    thr.row2 = seq(1, 3, by = 0.5),
    thr.col = seq(1, 3, by = 0.5),
    no.seeds = 100, direction = "updown")
```

where the arguments are:

data The input, a list of two numeric matrices, with the same number of columns. They may contain `NA` and/or `NaN` values, but then the algorithm might get slower, as R matrix multiplication is slower sometimes slower for these matrices, depending on your platform.

thr.row1 Numeric scalar or vector giving the threshold parameter for the rows of the first matrix. Higher values indicate a more stringent threshold and the result comodules will contain less rows for the first matrix on average. The threshold is measured by the number of standard deviations from the mean, over the values of the first row vector. If it is a vector then it must contain an entry for each seed.

thr.row2 Numeric scalar or vector, the threshold parameter(s) for the rows of the second matrix. See `thr.row1` for details.

thr.col Numeric scalar or vector giving the threshold parameter for the columns of both matrices. The analogue of `thr.row1`.

no.seeds Integer scalar, the number of random seeds to use.

direction Character vector of length four, one for each matrix multiplication performed during a PPA iteration. It specifies whether we are interested in rows/columns that are higher (‘`up`’) than average, lower than average (‘`down`’), or both (‘`updown`’). The first and the second entry both corresponds to the common column dimension of the two matrices, so they should be equal, otherwise a warning is given.

The `ppa` function provides and easy interface to the PPA. It runs all sptes of a typical PPA work flow, with their default paramers.

This involves:

1. Normalizing the input matrices by calling `ppa.normalize`.
2. Generating random input seeds via `generate.seeds`.
3. Running the PPA with all combinations of the given row1, row2 and column thresholds (by default 1, 1.5, 2, 2.5, 3); by calling `ppa.iterate`.
4. Merging similar co-modules, separately for each threshold combination, by calling `ppa.unique`.
5. Filtering the co-modules separately for each threshold combination, by calling `ppa.filter.robust`.
6. Putting all co-modules from the run with different thresholds, into a single object.
7. Merging similar co-modules, again, but now across all threshold combinations. If two co-modules are similar, then the larger one, the one with milder thresholds is kept.

Please see the manual pages of these functions for the details.

Value

A named list is returned with the following elements:

`rows1`	The first components of the co-modules, corresponding to the rows of the first input matrix. Every column corresponds to a co-module, if an element (the score of the row) is non-zero, that means that that component is included in the co-module, otherwise it is not. Scores are between -1 and 1. If two scores have the same non-zero sign, then the corresponding first matrix rows are collelated. If they have an opposite sign, then they are anti-correlated. If an input seed did not converge within the allowed number of iterations, then that column of `rows1` contains `NA` values. The `ppa` function does not produce such columns, because it always drops the non-convergent seeds via a call to `ppa.unique`. The result of the `ppa.iterate` function might contain such columns, though.
`rows2`	This is the same as `rows1`, but for the second input matrix.
`columns`	The same as `rows1` and `rows2`, but for the columns of both input matrices.
`seeddata`	A data frame containing information about the co-modules. There is one row for each co-module. The data frame has the following columns:

`iterations` The number of iterations needed for convergence.

`oscillation` The oscillation cycle of this is oscillating co-module. Zero otherwise.

`thr.row1` The threshold used for the rows of the first matrix.

`thr.row2` The threshold used for the rows of the second matrix.

`thr.col` The threshold used for the common column dimension.

`freq` Numeric scalar, the number of times the same (or a very similar) co-module was found.

`rob` The robustness score of the module.

`rob.limit` The robustness limit that was used to filter the module. See `ppa.filter.robust` for details.

`rundata` A named list with information about the PPA run. It has the following entries:

`direction` Character vector of length four, the `direction` argument of the `ppa.iterate` call.

`convergence` Character scalar, the convergence criteria that was used, see the `ppa.iterate` function for details.

`cor.limit` Numeric scalar, the correlation threshold, that was used if the convergence criteria was '`cor`'.

`maxiter` The maximum number of PPA iterations.

`N` The total number of input seeds that were used to find the co-modules.

`prenormalize` Logical scalar, whether the input matrices were pre-normalized, see `ppa.normalize` for details.

`hasNA` Logical vector of length two. Whether the two input matrices contained any `NA` or `NaN` values.

`unique` Logical scalar, whether the co-modules are unique, i.e. whether `ppa.unique` was called.

`oscillation` Logical scalar, whether the `ppa.iterate` run looked for oscillating modules.

`rob.perms` The number of data permutations that was performed during the robustness filtering, see `ppa.filter.robust` for details.

Author(s)

Gabor Csardi <`Gabor.Csardi@unil.ch`>

References

Kutalik Z, Bergmann S, Beckmann, J: A modular approach for integrative analysis of large-scale gene-expression and drug-response data *Nat Biotechnol* 2008 May; 26(5) 531-9.

See Also

isa2-package for a short introduction to the ISA and the Ping-Pong algorithms. See the functions mentioned above if you want to change the default ISA parameters.

Examples

```
## WE do not run this, it takes relatively long
## Not run:
data <- ppa.in.silico(noise=0.1)
ppa.result <- ppa(data[1:2], direction="up")

## Find the best bicluster for each block in the input
## (based on the rows of the first input matrix)
best <- apply(cor(ppa.result$rows1, data[[3]]), 2, which.max)

## Check correlation
sapply(seq_along(best),
       function(x) cor(ppa.result$rows1[,best[x]], data[[3]][,x]))

## The same for the rows of the second matrix
sapply(seq_along(best),
       function(x) cor(ppa.result$rows2[,best[x]], data[[4]][,x]))

## The same for the columns
sapply(seq_along(best),
       function(x) cor(ppa.result$columns[,best[x]], data[[5]][,x]))

## Plot the data and the modules found
if (interactive()) {
  layout(rbind(1:2,c(3,6),c(4,7), c(5,8)))
  image(data[[1]], main="In-silico data, first matrix")
  image(data[[2]], main="In-silico data, second matrix")
  sapply(best[1:3], function(b) image(outer(ppa.result$rows1[,b],
                                      ppa.result$columns[,b]),
                                main=paste("Module", b)))
  sapply(best[1:3], function(b) image(outer(ppa.result$rows2[,b],
                                      ppa.result$columns[,b]),
                                    main=paste("Module", b)))
}

## End(Not run)
```

`ppa.in.silico` *Generate in-silico input data for testing the PPA algorithm*

Description

This function generates an artificial test data set for the PPA algorithm: two matrices, with common column dimension, containing co-modules of prescribed number, size, signal level, noise level and background noise.

Usage

```
ppa.in.silico (num.rows1 = 300, num.rows2 = 200, num.cols = 50,
    num.fact = 3, mod.row1.size = round(0.5 * num.rows1/num.fact),
    mod.row2.size = round(0.5 * num.rows2/num.fact),
    mod.col.size = round(0.5 * num.cols/num.fact),
    noise = 0.1, mod.signal = rep(1, num.fact),
    mod.noise = rep(0, num.fact), overlap.row1 = 0,
    overlap.row2 = overlap.row1, overlap.col = overlap.row1)
```

Arguments

`num.rows1`	The number of rows in the first data matrix.
`num.rows2`	The number of rows in the second data matrix.
`num.cols`	The number of columns in both matrices.
`num.fact`	The number of co-modules to put in the data.
`mod.row1.size`	The size of the co-modules, the number of rows from the first data matrix. It can be a scalar or a vector, and it is recycled.
`mod.row2.size`	The size of the co-modules, the number of rows from the second data matrix. It can be a scalar or a vector, and it is recycled.
`mod.col.size`	The size of the co-modules, the number of columns (from both matrices). It can be a scalar or a vector, and it is recycled.
`noise`	The level of the background noise to be added to the data matrices. It gives the standard deviation of the normal distribution from which the noise is generated.
`mod.signal`	The signal level of the co-modules.
`mod.noise`	The noise levels of the different co-modules. Normally distributed noise with standard deviation `mod.noise` is added to the data. This is in addition to the background noise.
`overlap.row1`	The overlap of the modules, for the rows of the first matrix. Zero means no overlap, one means one overlapping row, etc.
`overlap.row2`	The same as `overlap.row1`, but for the rows of the second matrix.
`overlap.col`	The same as `overlap.row1`, but for the columns of both matrices.

Details

ppa.in.silico creates an artificial data set to test the PPA (or potentially other) algorithm. It creates two data matrices with an overlapping dimension and a checkerboard-like structure. The fields of the checkerboard are the co-modules, and they may have different signal and noise levels, and they may also overlap.

Value

A list with five matrices. The first two are the two data matrices, they have the same number of columns. The last three matrices contain the correct modules, for the rows of the first matrix, the rows of the second matrix, and finally for the common column dimension.

Author(s)

Gabor Csardi <Gabor.Csardi@unil.ch>

References

Kutalik Z, Bergmann S, Beckmann, J: A modular approach for integrative analysis of large-scale gene-expression and drug-response data *Nat Biotechnol* 2008 May; 26(5) 531-9.

See Also

See ppa for an easy way of running the PPA

Examples

```
## Define a function for plotting if we are interactive
if (interactive()) { layout(cbind(1:2,3:4,5:6,7:8)) }
myimage <- function(mat) {
  if (interactive()) {
    par(mar=c(3,2,1,1)); image(mat[[1]])
    par(mar=c(3,2,1,1)); image(mat[[2]])
  }
}

## Co-modules without overlap and noise
silico1 <- ppa.in.silico(100, 100, 100, 10, mod.row1.size=10,
                         mod.row2.size=10, mod.col.size=10, noise=0)
myimage(silico1)

## The same, but with overlap and noise
silico2 <- ppa.in.silico(100, 100, 100, 10, mod.row1.size=10,
                         mod.row2.size=10, mod.col.size=10, noise=0.1,
                         overlap.row1=3)
myimage(silico2)

## Co-modules with different noise levels
silico3 <- ppa.in.silico(100, 100, 100, 5, mod.row1.size=10,
                         mod.row2.size=10, mod.col.size=10, noise=0.01,
                         mod.noise=seq(0.1,by=0.1,length=5))
```

```
myimage(silico3)

## Co-modules withe different signal levels
silico4 <- ppa.in.silico(100, 100, 100, 5, mod.row1.size=10,
                         mod.row2.size=10, mod.col.size=10, noise=0.01,
                         mod.signal=seq(1,5,length=5))
myimage(silico4)
```

`ppa.iterate` *The Ping-Pong Algorithm*

Description

Perform PPA on two (normalized) input matrices.

Usage

```
## S4 method for signature 'list'
ppa.iterate(normed.data, ...)
```

Arguments

`normed.data`	The normalized input matrices. A list of four matrices, usually coming from the `ppa.normalize` function.
`...`	Additional arguments, see details below.

Details

`ppa.iterate` performs the PPA iteration on the specified input matrices using given input seeds. It can be called as

```
    ppa.iterate(normed.data, row1.seeds, col1.seeds,
                row2.seeds, col2.seeds,
                thr.row1, thr.col=thr.row1, thr.row2=thr.row1,
direction="updown",
convegence=c("cor"), cor.limit=0.9,
oscillation=FALSE, maxiter=100)
```

where the arguments are:

normed.data The normalized data, a list of four matrices with the appropriate size. They are usually coming from the output of the `ppa.normalize` function.

row1.seeds The row seed vectors for the first matrix. At least one of the four possible seed vectors must be present and they will be concatenated, after doing the suitable half-iterations.

col1.seeds The column seed vectors for the first matrix. At least one of the four possible seed vectors must be present and they will be concatenated, after doing the suitable half-iterations.

row2.seeds The row seed vectors for the second matrix. At least one of the four possible seed vectors must be present and they will be concatenated, after doing the suitable half-iterations.

col2.seeds The column seed vectors for the second matrix. At least one of the four possible seed vectors must be present and they will be concatenated, after doing the suitable half-iterations.

thr.row1 Numeric scalar or vector giving the threshold parameter for the rows of the first matrix. Higher values indicate a more stringent threshold and the result comodules will contain less rows for the first matrix on average. The threshold is measured by the number of standard deviations from the mean, over the values of the first row vector. If it is a vector then it must contain an entry for each seed.

thr.col Numeric scalar or vector giving the threshold parameter for the columns of both matrices. The analogue of `thr.row1`.

thr.row2 Numeric scalar or vector, the threshold parameter(s) for the rows of the second matrix. See `thr.row1` for details.

direction Character vector of length four, one for each matrix multiplication performed during a PPA iteration. It specifies whether we are interested in rows/columns that are higher (‘`up`’) than average, lower than average (‘`down`’), or both (‘`updown`’). The first and the second entry both corresponds to the common column dimension of the two matrices, so they should be equal, otherwise a warning is given.

convergence Character scalar, the convergence criteria for the PPA iteration. If it is ‘`cor`’, then convergence is measured based on high Pearson correlation (see the `cor.limit` argument below) of the subsequent row and column vectors. Currently this is the only option implemented.

cor.limit The correlation limit for convergence if the ‘`cor`’ method is used.

oscillation Logical scalar, whether to look for oscillating seeds. Usually there are not too many oscillating seeds, so it is safe to leave this on `FALSE`.

maxiter The maximum number of iterations allowed.

Value

A named list with many components. Please see the manual page of `link{isa}` for a complete description.

Author(s)

Gabor Csardi <`Gabor.Csardi@unil.ch`>

References

Kutalik Z, Bergmann S, Beckmann, J: A modular approach for integrative analysis of large-scale gene-expression and drug-response data *Nat Biotechnol* 2008 May; 26(5) 531-9.

See Also

See `ppa` for an easy way of running the PPA

Examples

```
## A basic PPA workflow with a single threshold combination
## In-silico data
set.seed(1)
insili <- ppa.in.silico()

## Random seeds
seeds <- generate.seeds(length=nrow(insili[[1]]), count=100)

## Normalize input matrices
nm <- ppa.normalize(insili[1:2])

## Do PPA
ppares <- ppa.iterate(nm, row1.seeds=seeds, thr.row1=1, direction="up")

## Eliminate duplicates
ppares <- ppa.unique(nm, ppares)

## Fitler out not robust ones
ppares <- ppa.filter.robust(insili[1:2], nm, ppares)

## Print the sizes of the co-modules
cbind(colSums(ppares$rows1 != 0), colSums(ppares$rows1 != 0),
      colSums(ppares$columns != 0))

## Plot the original data and the first three modules found
if (interactive()) {
  layout(rbind(1:2,3:4))
  image(insili[[1]], main="In silico data 1")
  image(outer(ppares$rows1[,1],ppares$columns[,1])+
        outer(ppares$rows1[,2],ppares$columns[,2])+
        outer(ppares$rows1[,3],ppares$columns[,3]), main="PPA co-modules 2")
  image(insili[[2]], main="In silico data 2")
  image(outer(ppares$rows2[,1],ppares$columns[,1])+
        outer(ppares$rows2[,2],ppares$columns[,2])+
        outer(ppares$rows2[,3],ppares$columns[,3]), main="PPA co-modules 2")
}
```

`ppa.normalize` *Normalize input data for use with the PPA*

Description

Normalize the two input matrices and store them in a form that can be used effectively to perform the Ping-Pong Algorithm

Usage

```
## S4 method for signature 'list'
ppa.normalize(data, ...)
```

Arguments

`data`	A list of two numeric matrices, with matching number of columns. They might contain `Na` and/or `NaN` values.
`...`	Additional arguments, see details below.

Details

This function can be called as

```
isa.normalize(data, prenormalize = FALSE)
```

where the arguments are:

data A list of two numeric matrices, with matching number of columns. They might contain `Na` and/or `NaN` values.

prenormalize Logical scalar, see details below.

It was observed that the PPA works best if the input matrices are scaled to have mean zero and standard deviation one.

A PPA step consist of four matrix-multiplications and this requires four different matrices, each of the two input matrices scaled row-wise and column-wise.

If the `prenormalized` argument is set to `TRUE`, then row-wise scaling is calculated on the column-wise scaled matrices and not on the raw input.

Value

A list of four matrices, the first two corresponds to the first input matrix, the second two to the second matrix.

Author(s)

Gabor Csardi <`Gabor.Csardi@unil.ch`>

References

Kutalik Z, Bergmann S, Beckmann, J: A modular approach for integrative analysis of large-scale gene-expression and drug-response data *Nat Biotechnol* 2008 May; 26(5) 531-9.

See Also

See `ppa` for an easy way of running the PPA

Examples

```
## In-silico data
set.seed(1)
insili <- ppa.in.silico()
nm <- ppa.normalize(insili[1:2])

## Check it
max(abs(rowSums(nm[[1]])))
max(abs(rowSums(nm[[2]])))
max(abs(rowSums(nm[[3]])))
max(abs(rowSums(nm[[4]])))

max(abs(rowSums(nm[[1]])))
max(abs(rowSums(nm[[2]])))
max(abs(rowSums(nm[[3]])))
max(abs(rowSums(nm[[4]])))

## Plot them
if (interactive()) {
  layout(rbind(1:3,4:6))
  image(insili[[1]], main="Original data 1")
  image(t(nm[[1]]), main="Row normalized 1")
  image(nm[[2]], main="Column normalized 1")

  image(insili[[2]], main="Original data 2")
  image(t(nm[[3]]), main="Row normalized 2")
  image(nm[[4]], main="Column normalized 2")
}
```

---

ppa.unique *Filter co-modules that are very similar to each other*

Description

From a potentially non-unique set of PPA co-modules, create a unique set by removing all co-modules that are similar to others.

Usage

```
## S4 method for signature 'list,list'
 ppa.unique(normed.data, ppareult, ...)
```

Arguments

`normed.data`	The normalized input data, a list of four matrices, usually the output of the `ppa.normalize` function.
`pparesult`	The result of a PPA run, a set of co-modules.
`...`	Additional arguments, see details below.

Details

This function can be called as

```
    ppa.unique(normed.data, pparesult, method = c("cor"),
               ignore.div = TRUE, cor.limit = 0.9,
       neg.cor = TRUE, drop.zero = TRUE)
```

where the arguments are:

normed.data The normalized input data, a list of four matrices, usually the output of `ppa.normalize`.

pparesult The results of a PPA run, a set of co-modules.

method Character scalar giving the method to be used to determine if two co-modules are similar. Right now only '`cor`' is implemented, this keeps both co-modules if their Pearson correlation is less than `cor.limit`, for their row1, row2 and column scores. See also the `neg.cor` argument.

ignore.div Logical scalar, if `TRUE`, then the divergent co-modules will be removed.

cor.limit Numeric scalar, giving the correlation limit for the '`cor`' method.

neg.cor Logical scalar, if `TRUE`, then the '`cor`' method considers the absolute value of the correlation.

drop.zero Logical scalar, whether to drop co-modules that have all zero scores.

Value

A named list, the filtered `pparesult`. See the return value of `ppa.iterate` for the details.

Author(s)

Gabor Csardi <`Gabor.Csardi@unil.ch`>

References

Kutalik Z, Bergmann S, Beckmann, J: A modular approach for integrative analysis of large-scale gene-expression and drug-response data *Nat Biotechnol* 2008 May; 26(5) 531-9.

See Also

See `ppa` for an easy way of running the PPA

Examples

```
## Create an PPA module set
set.seed(1)
insili <- ppa.in.silico(noise=0.01)

## Random seeds
seeds <- generate.seeds(length=nrow(insili[[1]]), count=20)
```

```
## Normalize input matrix
nm <- ppa.normalize(insili[1:2])

## Do PPA
ppares <- ppa.iterate(nm, row1.seeds=seeds,
                      thr.row1=2, thr.row2=2, thr.col=1)

## Check correlation among modules
cc <- cor(ppares$rows1)
if (interactive()) { hist(cc[lower.tri(cc)],10) }

## Some of them are quite high, how many?
undiag <- function(x) { diag(x) <- 0; x }
sum(undiag(cc) > 0.99, na.rm=TRUE)

## Eliminate duplicated modules
ppares.unique <- ppa.unique(nm, ppares)

## How many modules left?
ncol(ppares.unique$rows1)

## Double check
cc2 <- cor(ppares.unique$rows1)
if (interactive()) { hist(cc2[lower.tri(cc2)],10) }

## High correlation?
sum(undiag(cc2) > 0.99, na.rm=TRUE)
```

`robustness` *Robustness of ISA biclusters and PPA co-modules*

Description

Robustness of ISA biclusters and PPA co-modules. The more robust biclusters/co-modules are more significant in the sense that it is less likely to see them in random data.

Usage

```
## S4 method for signature 'list'
robustness(normed.data, ...)
## S4 method for signature 'matrix'
isa.filter.robust(data, ...)
## S4 method for signature 'list'
ppa.filter.robust(data, ...)
```

Arguments

`normed.data`	The normalized input data, usually calculated with `isa.normalize`.

`data`	The original, not normalized input data, a matrix for `isa.filter.robust`, a list of two matrices for `ppa.filter.robust`.
`...`	Additional arguments, see details below.

Details

`robustness` can be called as

```
robustness(normed.data, row.scores, col.scores)
```

`isa.filter.robust` can be called as

```
isa.filter.robust(data, normed.data, isares,
                  perms = 1, row.seeds, col.seeds)
```

and `ppa.filter.robust` can be called as

```
ppa.filter.robust(data, normed.data, ppares,
                  perms = 1, row1.seeds, col1.seeds,
  row2.seeds, col2.seeds)
```

These arguments are:

normed.data The normalized input data, usually calculated with `isa.normalize` (for ISA) or `ppa.normalize` (for PPA).

row.scores The scores of the row components of the biclusters. Usually the `rows` member of the result list, as returned by `isa`, or `isa.iterate` or some other ISA function.

col.scores The scores of the columns of the biclusters, usually the `columns` member of the result list from `isa`.

data The original, not normalized input data.

isares The result of ISA, coming from `isa` or `isa.iterate` or any other function that return the same format.

perms The number of permutations to perform on the input data.

row.seeds Optionally the row seeds for the ISA run on the scrambled data. If this and `col.seeds` are both omitted the same number of random seeds are used as for `isaresult`.

col.seeds Optionally the column seed to use for the ISA on the scrambled input matrix. If both this and `row.seeds` are omitted, then the same number of random (row) seeds will be used as for `isares`.

ppares The result of a PPA run, the output of the `ppa.iterate` or the `ppa.unique` function (or any other function that returns the same format).

row1.seeds Optionally, the seeds based of the rows of the first input matrix, can be given here. Otherwise random seeds are used, the same number as it was used to find the original co-modules.

col1.seeds Optionally, the seeds based of the columns of the first input matrix, can be given here. Otherwise random seeds are used, the same number as it was used to find the original co-modules.

row2.seeds Optionally, the seeds based of the rows of the second input matrix, can be given here. Otherwise random seeds are used, the same number as it was used to find the original co-modules.

col2.seeds Optionally, the seeds based of the columns of the second input matrix, can be given here. Otherwise random seeds are used, the same number as it was used to find the original co-modules.

Even if you generate a matrix with uniform random noise in it, if you calculate ISA on it, you will get some biclusters, except maybe if you use very strict threshold parameters. These biclusters contain rows and columns that are correlated just by chance. The same is true for PPA.

To circumvent this, you can use the so-called robustness measure of the biclusters/co-modules. The robustness of a bicluster is the function of its rows, columns and the input data, and it is a real number, usually positive. It is roughly equivalent to the principal singular value of the submatrix (of the reordered input matrix) defined by the bicluster.

`robustness` calculates the robustness score of a set of biclusters/co-modules, usually coming from one or more ISA/PPA iterations.

`isa.filter.robust` provides filtering based on the robustness measure. It reshuffles the input matrix and calculates ISA on it, with the parameters that were used to find the biclusters under evaluation. It then calculates the robustness for the modules that were found in the scrambled matrix (if there is any) and removes any modules from the data set that have a lower robustness score than at least one module in the scrambled data.

You can think of `isa.filter.robust` as a permutation test, but the input data is shuffled only once (at least by default), because of the relatively high computational demands of the ISA.

`ppa.filter.robust` does essentially the same, but for PPA co-modules.

Value

`robustness` returns a numeric vector, the robustness score of each bicluster.

`isa.filter.robust` returns a named list, the filtered `isares`, see the return value of `isa.iterate` for the structure of the list.

`ppa.filter.robust` resturns a named list, the filtered `ppares`, see the return value of `ppa.iterate` for the structure of the list.

Author(s)

Gabor Csardi <`Gabor.Csardi@unil.ch`>

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

Ihmels J, Friedlander G, Bergmann S, Sarig O, Ziv Y, Barkai N: Revealing modular organization in the yeast transcriptional network *Nat Genet.* 2002 Aug;31(4):370-7. Epub 2002 Jul 22

Ihmels J, Bergmann S, Barkai N: Defining transcription modules using large-scale gene expression data *Bioinformatics* 2004 Sep 1;20(13):1993-2003. Epub 2004 Mar 25.

Kutalik Z, Bergmann S, Beckmann, J: A modular approach for integrative analysis of large-scale gene-expression and drug-response data *Nat Biotechnol* 2008 May; 26(5) 531-9.

See Also

isa2-package for a short introduction on the Iterative Signature Algorithm. See `isa` for an easy way of running ISA, `ppa` for an easy way of running the PPA.

Examples

```
## A basic ISA work flow for a single threshold combination
## In-silico data
set.seed(1)
insili <- isa.in.silico()

## Random seeds
seeds <- generate.seeds(length=nrow(insili[[1]]), count=100)

## Normalize input matrix
nm <- isa.normalize(insili[[1]])

## Do ISA
isares <- isa.iterate(nm, row.seeds=seeds, thr.row=2, thr.col=1)

## Eliminate duplicates
isares <- isa.unique(nm, isares)

## Calculate robustness
rob <- robustness(nm, isares$rows, isares$columns)
rob

## There are three robust ones and a lot of less robust ones
## Plot the three robust ones and three others
if (interactive()) {
  toplot1 <- rev(order(rob))[1:3]
  toplot2 <- sample(seq_along(rob)[-toplot1], 3)
  layout( rbind(1:3,4:6) )
  for (i in c(toplot1, toplot2)) {
    image(outer(isares$rows[,i], isares$column[,i]),
          main=round(rob[i],2))
  }
}

## Filter out not robust ones
isares2 <- isa.filter.robust(insili[[1]], nm, isares)

## Probably there are only three of them left
ncol(isares2$rows)
```

Index