
pyiomica Documentation

Release 1.3.3

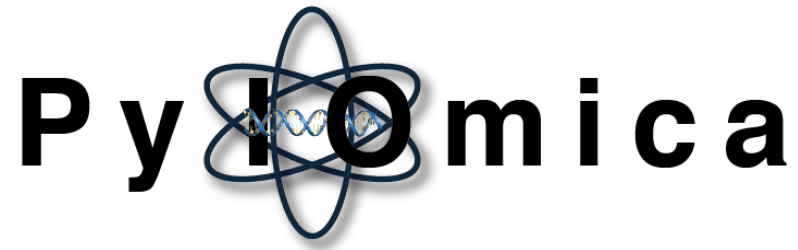
S. Domanskyi, C. Piermarocchi, G. Mias

Sep 26, 2022

CONTENTS:

1	Overview	3
1.1	Versions	3
1.2	Documentation	6
1.3	Additional information	6
1.4	Licensing	6
1.5	Contact information	6
1.6	Funding	6
2	Installation	7
2.1	Pre-Installation Requirements	7
2.2	Installation Instructions	7
2.3	Running PyIOmica	7
3	Functionality	9
3.1	Global variables	9
3.2	Categorization functions	10
3.3	Enrichment analyses functions	14
3.4	Extended DataFrame and data-processing functions	25
3.5	Clustering functions	34
3.6	Frequency Based Subject Match	38
3.7	Visibility graph preparation functions	41
3.8	Visibility Graph Community Detection	43
3.9	Visualization functions	44
3.10	Utility functions	55
3.11	Data Storage	56
4	Dependencies	59
5	Included data	61
6	Examples	63
6.1	Enrichment report	63
6.2	Import of MathIOmica Objects	64
6.3	Clustering object export example	65
6.4	GO Analysis examples	65
6.5	KEGG Analysis examples	67
6.6	Reactome Analysis examples	70
6.7	Visibility Graph examples	70
6.8	Visibility Graph Community Detection	76
6.9	Extended DataFrame	79
6.10	Time Series Categorization	81

7 Indices and tables	87
Python Module Index	89
Index	91



This documentation describes PyIOMica, a Python package that provides bioinformatics utilities for analyzing (dynamic) omics datasets. PyIOMica extends MathIOMica usage to Python and implements new visualizations and computational tools for graph analyses.

OVERVIEW

An open source Python package, for analyzing longitudinal omics datasets, which includes multiple tools for processing of multi-modal mapped data, characterizing time series in terms of periodograms and autocorrelations, classifying temporal behavior, visualizing visibility graphs, and testing data for gene ontology and pathway enrichment. PyIOmica includes optimized new algorithms adapted from MathIOmica (which runs on the proprietary Mathematica platform), now made available as Python open source code for all users, and additionally expands extensively graphical utilities for visualization of classified temporal data, and network representation of time series.

The publication describing this software:

Sergii Domanskyi, Carlo Piermarocchi, George I Mias, **PyIOmica: longitudinal omics analysis and trend identification**, *Bioinformatics*, Volume 36, Issue 7, 2020, Pages 2306–2307,
<https://doi.org/10.1093/bioinformatics/btz896>

1.1 Versions

- **1.3.3**
 - updated functionality for visibility graph community detection for `communityDetectByPathLength` to output community times as an option
- **1.3.2**
 - updated code for `ReactomeAnalysis`, set header parameter `headersGET = {'accept': 'text/CSV'}` for Reactome analysis service compatibility
- **1.3.1**
 - updated code for `calculateTimeSeriesCategorization` to output correct labels for Lag classifications when exporting to Excel files
 - modified `clusteringFunctions.exportClusteringObject` to output components of clustering objects in the same order as they appear in heatmaps (i.e. top to bottom, instead of bottom to up, which was the previous behavior)
- **1.3.0**
 - updated internal code for `annotate()` function to use “text” versus “s”
 - updated `makeVisibilityGraph` function
 - updated `addVisibilityGraph` function with additional selection for community detection methods
 - updated `getCommunitiesOfTimeSeries` to process horizontal and natural visibility graphs

- updated visualizeTimeSeriesCategorization for community detection options selection for displayed visibility graphs
- updated makeDendrogramHeatmapOfClusteringObject for community detection method options for displayed visibility graphs
- 1.2.9
 - **renamed inconsistencies:**
 - * PlotNVGBarGraphDual -> plotNVGBarGraphDual
 - * PlotHVGBarGraphDual -> plotHVGBarGraphDual
- 1.2.8
 - Documentation strings update for frequencySubjectMatch file - typos corrected.
 - **Functions changed to camelCase for consistency:**
 - * **in module frequencySubjectMatch:**
 - IOptimizeK -> optimizeK
 - get_community_genes_dict -> getCommunityGenesDict
 - split_genes -> splitGenes
 - get_community_top_genes_by_number -> getCommunityTopGenesByNumber
 - get_community_top_genes_by_frequency_ranking -> getCommunityTopGenesByFrequencyRanking
 - * **in module clusteringFunctions:**
 - get_n_clusters_from_linkage_Silhouette -> getNClustersFromLinkageSilhouette
 - get_n_clusters_from_linkage_Elbow -> getNClustersFromLinkageElbow
 - * **in module visualizationFunctions**
 - PlotNVGBarGraph_Dual -> PlotNVGBarGraphDual
 - PlotHVGBarGraphDual -> PlotHVGBarGraphDual
 - **example files were modified to reflect above changes**
 - * pyiomica_examples.ipynb
 - * examplesVisibilityGraphCommunityDetection.py
- 1.2.7
 - Added new functionality in frequencySubjectMatch to enable comparison across subjects, utilizing spectra to identify common changing components, construct networks with such connections, and identify clusters of similar temporal behavior.
- 1.2.6
 - Updated autocorrelation computation function to improve handling of nan values.
- 1.2.5
 - Updated visibilityGraphCommunityDetection functions (correction for division by zero in community calculations).
- 1.2.4

- Updated plotting heatmap functions (`categorizationFunctions.visualizeTimeSeriesCategorization` and `visualizationFunctions.makeDendrogramHeatmapOfClusteringObject`) to utilize optional custom strings for x-axis and plot labels.
 - Updated `categorizationFunctions.visualizeTimeSeriesCategorization` to avoid error for cases where the linkage array has only 1 row.
- 1.2.3
 - Applied minor fixes of numpy deprecation warnings.
 - Updated examples Jupyter notebook.
 - Updated enrichment report export function.
 - Updated PyIOmica dependency graph.
- 1.2.2
 - Applied minor fixes related to numpy upgrade.
 - Fixed typo in name of function `extendedDataFrame.getLombScarglePeriodogramOfDataframe`
- 1.2.1
 - Added Reactome pathway overrepresentation analysis functions.
 - Added Reactome analysis report export function.
 - Added examples of Reactome analysis.
- 1.2.0
 - Added new visibility graph based community detection functions.
 - Added plotting functions.
 - Added examples of visibility graph community detection.
 - Updated examples Jupyter notebook.
- 1.1.2
 - Small typographical fixes.
- 1.1.1
 - Updated examples Jupyter notebook.
 - Small typographical fixes.
- 1.1.0
 - Restructured all modules.
 - Developed ReadTheDocs documentation.
- 1.0.2
 - Updated setup dependencies for pip compatibility.
- 1.0.1
 - Updated setup dependencies.
- 1.0.0
 - First Release.

1.2 Documentation

Documentation for PyIOmica is built-in and is available through the `help()` functionality in Python or online at <https://pyiomica.readthedocs.io>.

1.3 Additional information

- PyIOmica is a multi-omics analysis framework distributed as a Python package that aims to assist in bioinformatics.
- The most current version of the package is maintained at <https://github.com/gmiaslab/pyiomica>
- News are distributed via twitter (@mathiomica)

1.4 Licensing

PyIOmica is released under an MIT License. Please also consult the folder `LICENSES` distributed with PyIOmica regarding Licensing information for use of external associated content.

1.5 Contact information

- Contributors: Sergii Domanskyi, Minzhang Zheng, Carlo Piermarocchi, George I. Mias.
- G.MiasLab (<https://georgemias.org>)
- e-mail: gmiaslab@gmail.com
- twitter: @gmiaslab

1.6 Funding

PyIOmica development and associated research were supported by the Translational Research Institute for Space Health through NASA Cooperative Agreement NNX16AO69A (Project Number T0412, PI: Mias). The content is solely the responsibility of the authors and does not necessarily represent the official views of the supporting funding agencies.

INSTALLATION

2.1 Pre-Installation Requirements

To install PyIOmica on any platform you need Python version 3.7 or higher

2.2 Installation Instructions

1. To install the current release from PyPI (Python Package Index) use pip:

```
pip install pyiomica
```

2. Alternatively, you can install directly from github using:

```
pip install git+https://github.com/gmiaslab/pyiomica/
```

2.3 Running PyIOmica

After installation run:

```
>>> import pyiomica
```


FUNCTIONALITY

3.1 Global variables

Submodule **pyiomica.globalVariables**

This module contains global constants used in PyIOmica. Some of the modules, classes and functions are imported in this module.

Data:

<i>printPackageGlobalDefaults</i>	Whether to print package global defaults listed in this module
<i>PackageDirectory</i>	Package directory
<i>ConstantPyIOmicaDataDirectory</i>	ConstantPyIOmicaDataDirectory is a global variable pointing to the PyIOmica data directory.
<i>ConstantPyIOmicaExamplesDirectory</i>	ConstantPyIOmicaExamplesDirectory is a global variable pointing to the PyIOmica example data directory.
<i>ConstantPyIOmicaExampleVideosDirectory</i>	ConstantPyIOmicaExampleVideosDirectory is a global variable pointing to the PyIOmica example videos directory.
<i>ConstantGeneDictionary</i>	ConstantGeneDictionary is a global gene/protein dictionary variable typically created by GetGeneDictionary.

printPackageGlobalDefaults = False

Whether to print package global defaults listed in this module

PackageDirectory =

'/home/docs/checkouts/readthedocs.org/user_builds/pyiomica/checkouts/latest/pyiomica'

Package directory

ConstantPyIOmicaDataDirectory = '/home/docs/checkouts/readthedocs.org/user_builds/pyiomica/checkouts/latest/pyiomica/data'

ConstantPyIOmicaDataDirectory is a global variable pointing to the PyIOmica data directory.

ConstantPyIOmicaExamplesDirectory = '/home/docs/checkouts/readthedocs.org/user_builds/pyiomica/checkouts/latest/pyiomica/data/ExampleData'

ConstantPyIOmicaExamplesDirectory is a global variable pointing to the PyIOmica example data directory.

ConstantPyIOmicaExampleVideosDirectory = '/home/docs/checkouts/readthedocs.org/user_builds/pyiomica/checkouts/latest/pyiomica/data/ExampleVideos'

ConstantPyIOmicaExampleVideosDirectory is a global variable pointing to the PyIOmica example videos directory.

ConstantGeneDictionary = None

ConstantGeneDictionary is a global gene/protein dictionary variable typically created by GetGeneDictionary.

3.2 Categorization functions

Submodule **pyiomica.categorizationFunctions**

Example of use of a function described in this module:

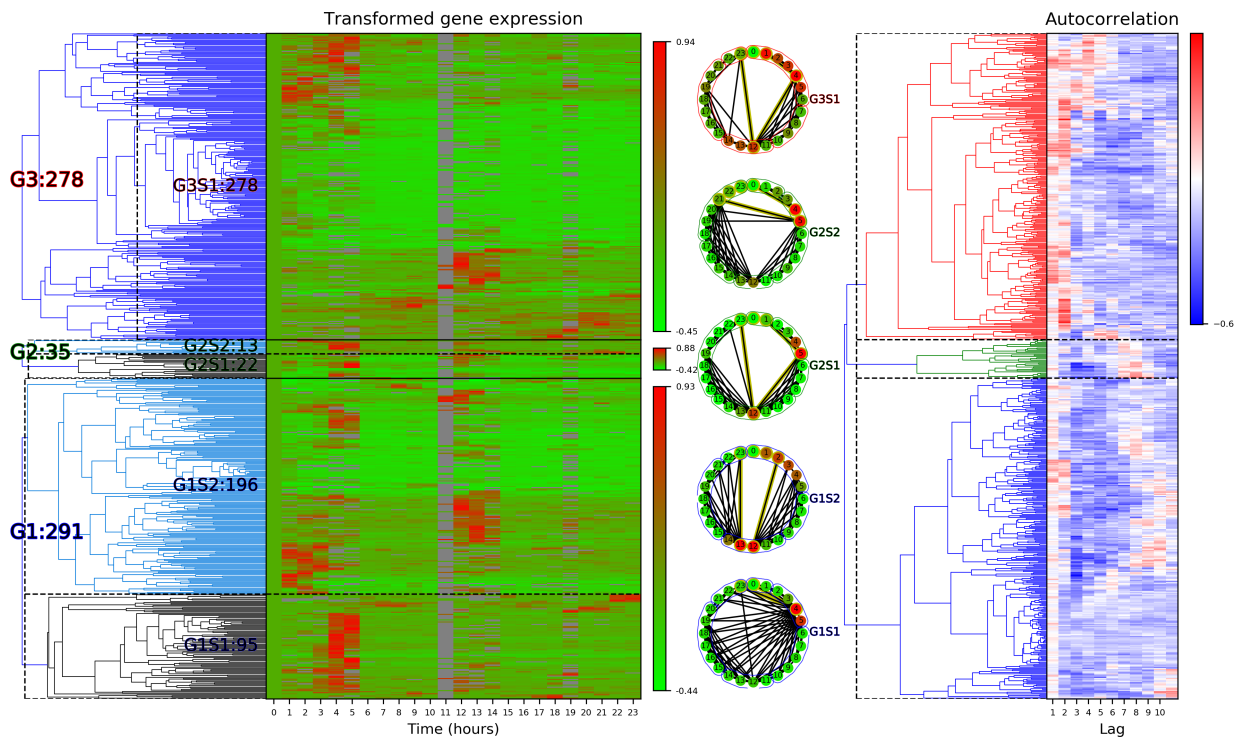
```
# import the package and its module Categorization functions
import pyiomica as pio
from pyiomica import categorizationFunctions as cf

# Location of this example data
dir = pio.ConstantPyIOmicaExamplesDirectory

# Unzip sample data
with pio.zipfile.ZipFile(pio.os.path.join(dir, 'SLV.zip'), "r") as zipFile:
    zipFile.extractall(path=dir)

# Process sample dataset SLV Hourly 1
dataName = 'SLV_Hourly1TimeSeries'
saveDir = pio.os.path.join('results', dataName, '')
dataDir = pio.os.path.join(dir, 'SLV')
df_data = pio.pd.read_csv(pio.os.path.join(dataDir, dataName + '.csv'), index_col=[0,1,
↪ 2], header=0)
cf.calculateTimeSeriesCategorization(df_data, dataName, saveDir, NumberOfRandomSamples_
↪ = 10**4)
cf.clusterTimeSeriesCategorization(dataName, saveDir)
cf.visualizeTimeSeriesCategorization(dataName, saveDir)
```

One of the figures generated by visualizeTimeSeriesCategorization is shown below:



Categorization functions

Functions:

<code>calculateTimeSeriesCategorization(df_data,</code>	Time series classification.
<code>...)</code>	
<code>clusterTimeSeriesCategorization(dataName, ...)</code>	Visualize time series classification.
<code>visualizeTimeSeriesCategorization(dataName,</code>	Visualize time series classification.
<code>...)</code>	

calculateTimeSeriesCategorization(*df_data*, *dataName*, *saveDir*, *hdf5fileName=None*, *p_cutoff=0.05*, *fraction=0.75*, *constantSignalsCutoff=0.0*, *lowValuesToTag=1.0*, *lowValuesToTagWith=1.0*, *NumberOfRandomSamples=100000*, *NumberOfCPUs=4*, *referencePoint=0*, *autocorrelationBased=True*, *calculateAutocorrelations=False*, *calculatePeriodograms=False*, *preProcessData=True*)

Time series classification.

Parameters:

df_data: `pandas.DataFrame`
Data to process

dataName: `str`
Data name, e.g. "myData_1"

saveDir: `str`
Path of directories pointing to data storage

hdf5fileName: str, Default None

Preferred hdf5 file name and location

p_cutoff: float, Default 0.05

Significance cutoff signals selection

fraction: float, Default 0.75

Fraction of non-zero point in a signal

constantSignalsCutoff: float, Default 0.

Parameter to consider a signal constant

lowValuesToTag: float, Default 1.

Values below this are considered low

lowValuesToTagWith: float, Default 1.

Low values to tag with

NumberOfRandomSamples: int, Default 105**

Size of the bootstrap distribution to generate

NumberOfCPUs: int, Default 4

Number of processes allowed to use in calculations

referencePoint: int, Default 0

Reference point

autocorrelationBased: boolean, Default True

Whether Autocorrelation of Frequency based

calculateAutocorrelations: boolean, Default False

Whether to recalculate Autocorrelations

calculatePeriodograms: boolean, Default False

Whether to recalculate Periodograms

preProcessData: boolean, Default True

Whether to preprocess data, i.e. filter, normalize etc.

Returns:

None

Usage:

`calculateTimeSeriesCategorization(df_data, dataName, saveDir)`

clusterTimeSeriesCategorization(*dataName*, *saveDir*, *numberOfLagsToDraw*=3, *hdf5fileName*=None, *exportClusteringObjects*=False, *writeClusteringObjectToBinaries*=True, *autocorrelationBased*=True, *method*='weighted', *metric*='correlation', *significance*='Elbow')

Visualize time series classification.

Parameters:

dataName: str

Data name, e.g. "myData_1"

saveDir: str

Path of directories pointing to data storage

numberOfLagsToDraw: int, Default 3

First top-N lags (or frequencies) to draw

hdf5fileName: str, Default None

HDF5 storage path and name

exportClusteringObjects: boolean, Default False

Whether to export clustering objects to xlsx files

writeClusteringObjectToBinaries: boolean, Default True

Whether to export clustering objects to binary (pickle) files

autocorrelationBased: boolean, Default True

Whether to label to print on the plots

method: str, Default 'weighted'

Linkage calculation method

metric: str, Default 'correlation'

Distance measure

significance: str, Default 'Elbow'

Method for determining optimal number of groups and subgroups

Returns:

None

Usage:

```
clusterTimeSeriesClassification('myData_1', '/dir1/dir2/')
```

```
visualizeTimeSeriesCategorization(dataName, saveDir, numberOfLagsToDraw=3,
                                   autocorrelationBased=True, xLabel='Time', plotLabel='Transformed
                                   Expression', horizontal=False, minNumberOfCommunities=2,
                                   communitiesMethod='WDPVG', direction='left', weight='distance')
```

Visualize time series classification.

Parameters:

dataName: str

Data name, e.g. "myData_1"

saveDir: str

Path of directories pointing to data storage

numberOfLagsToDraw: boolean, Default 3

First top-N lags (or frequencies) to draw

autocorrelationBased: boolean, Default True

Whether autocorrelation or frequency based

xLabel: str, Default 'Time'

X-axis label

plotLabel: str, Default 'Transformed Expression'

Label for the heatmap plot

horizontal: boolean, Default False

Whether to use horizontal or natural visibility graph.

minNumberOfCommunities: int, Default 2

Number of communities to find depends on the number of splits. This parameter is ignored in methods that automatically estimate optimal number of communities.

communitiesMethod: str, Default 'WDPVG'

String defining the method to use for communitiy detection:

'Girvan_Newman': edge betweenness centrality based approach

'betweenness_centrality': reflected graph node betweenness centrality based approach

‘WDPVG’: weighted dual perspective visibility graph method (note to also set weight variable)

direction: str, default ‘left’

The direction that nodes aggregate to communities:

None: no specific direction, e.g. both sides.

‘left’: nodes can only aggregate to the left side hubs, e.g. early hubs

‘right’: nodes can only aggregate to the right side hubs, e.g. later hubs

weight: str, Default ‘distance’

Type of weight for communitiesMethod=‘WDPVG’:

None: no weighted

‘time’: weight = abs(times[i] - times[j])

‘tan’: weight = abs((data[i] - data[j])/(times[i] - times[j])) + 10**(-8)

‘distance’: weight = A[i, j] = A[j, i] = ((data[i] - data[j])**2 + (times[i] - times[j])**2)**0.5

Returns:

None

Usage:

visualizeTimeSeriesClassification(‘myData_1’, ‘/dir1/dir2/’)

3.3 Enrichment analyses functions

Submodule `pyiomica.enrichmentAnalyses`

Example of use of a function described in this module:

```
import pyiomica as pio

# Import functions necessary for this demo
from pyiomica.enrichmentAnalyses import GOAnalysis, ExportEnrichmentReport

# Specify a directory for output
EnrichmentOutputDirectory = pio.os.path.join('results', 'EnrichmentOutputDirectory')

# Let's do a GO analysis for a group of genes, annotated with their "Gene Symbol":
goExample1 = GOAnalysis(["TAB1", "TNFSF13B", "MALT1", "TIRAP", "CHUK",
                        "TNFRSF13C", "PARP1", "CSNK2A1", "CSNK2A2", "CSNK2B", "LTBR",
                        "LYN", "MYD88", "GADD45B", "ATM", "NFKB1", "NFKB2", "NFKBIA",
                        "IRAK4", "PIAS4", "PLAU"])

# Export enrichment results in to .xlsx file
ExportEnrichmentReport(goExample1,
                        AppendString='goExample1',
                        OutputDirectory=EnrichmentOutputDirectory + 'GOAnalysis/')
```

Note: Function `ExportEnrichmentReport` generates “xlsx” file, described in [Examples](#).

Annotations and Enumerations

Functions:

<i>internalAnalysisFunction</i> (data, multiCorr, ...)	Analysis for Multi-Omics or Single-Omics input list The function is used internally and not intended to be used directly by user.
<i>OBOGODictionary</i> ([FileURL, ImportDirectly, ...])	Generate Open Biomedical Ontologies (OBO) Gene Ontology (GO) vocabulary dictionary.
<i>GetGeneDictionary</i> ([geneUCSCTable, ...])	Create an ID/accession dictionary from a UCSC search - typically of gene annotations.
<i>GOAnalysisAssigner</i> ([PyIomicaDataDirectory, ...])	Download and create gene associations and restrict to required background set.
<i>obtainConstantGeneDictionary</i> (GeneDictionary, ...)	Obtain gene dictionary - if it exists can either augment with new information or Species or create new, if not exist then create variable.
<i>GOAnalysis</i> (data[, GetGeneDictionaryOptions, ...])	Calculate input data over-representation analysis for Gene Ontology (GO) categories.
<i>GeneTranslation</i> (InputList, TargetIDList, ...)	Use geneDictionary to convert inputList IDs to different annotations as indicated by targetIDList.
<i>KEGGAnalysisAssigner</i> ([...])	Create KEGG: Kyoto Encyclopedia of Genes and Genomes pathway associations, restricted to required background set, downloading the data if necessary.
<i>KEGGDictionary</i> ([PyIomicaDataDirectory, ...])	Create a dictionary from KEGG: Kyoto Encyclopedia of Genes and Genomes terms - typically association of pathways and members therein.
<i>KEGGAnalysis</i> (data[, AnalysisType, ...])	Calculate input data over-representation analysis for KEGG: Kyoto Encyclopedia of Genes and Genomes pathways.
<i>MassMatcher</i> (data, accuracy[, ...])	Assign putative mass identification to input data based on monoisotopic mass (using PyIomica's mass dictionary).
<i>MassDictionary</i> ([PyIomicaDataDirectory])	Load PyIomica's current mass dictionary.
<i>ExportEnrichmentReport</i> (data[, AppendString, ...])	Export results from enrichment analysis to Excel spreadsheets.
<i>BenjaminiHochbergFDR</i> (pValues[, ...])	HypothesisTesting BenjaminiHochbergFDR correction
<i>ReactomeAnalysis</i> (data[, uploadURL, ...])	Reactome POST-GET-style analysis.
<i>ExportReactomeEnrichmentReport</i> (data[, ...])	Export results from enrichment analysis to Excel spreadsheets.

internalAnalysisFunction(data, multiCorr, MultipleList, OutputID, InputID, Species, totalMembers, pValueCutoff, ReportFilterFunction, ReportFilter, TestFunction, HypothesisFunction, FilterSignificant, AssignmentForwardDictionary, AssignmentReverseDictionary, prefix, infoDict)

Analysis for Multi-Omics or Single-Omics input list The function is used internally and not intended to be used directly by user.

Usage:

Intended for internal use

OBOGODictionary(FileURL='http://purl.obolibrary.org/obo/go/go-basic.obo', ImportDirectly=False, PyIomicaDataDirectory=None, OBOFile='goBasicObo.txt')

Generate Open Biomedical Ontologies (OBO) Gene Ontology (GO) vocabulary dictionary.

Parameters:

FileURL: str, Default “<http://purl.obolibrary.org/obo/go/go-basic.obo>”

Provides the location of the Open Biomedical Ontologies (OBO) Gene Ontology (GO) file in case this will be downloaded from the web

ImportDirectly: boolean, Default False

Import from URL regardless if the file already exists

PyIomicaDataDirectory: str, Default None

Path of directories to data storage

OBOFile: str, Default “goBasicObo.txt”

Name of file to store data in (file will be zipped)

Returns:

dictionary

Dictionary of definitions

Usage:

```
OBODict = OBODictionary()
```

GetGeneDictionary(*geneUCSCTable=None, UCSCSQLString=None, UCSCSQLSelectLabels=None, ImportDirectly=False, Species='human', KEGGUCSCSplit=[True, 'KEGG Gene ID']*)

Create an ID/accession dictionary from a UCSC search - typically of gene annotations.

Parameters:

geneUCSCTable: str, Default None

Path to a geneUCSCTable file

UCSCSQLString: str, Default None

An association to be used to obtain data from the UCSC Browser tables. The key of the association must match the Species option value used (default: human). The value for the species corresponds to the actual MySQL command used

UCSCSQLSelectLabels: str, Default None

An association to be used to assign key labels for the data imported from the UCSC Browser tables. The key of the association must match the Species option value used (default: human). The value is a multi component string list corresponding to the matrices in the data file, or the tables used in the MySQL query provided by UCSCSQLString

ImportDirectly: boolean, Default False

Import from URL regardless if the file already exists

Species: str, Default “human”

Species considered in the calculation, by default corresponding to human

KEGGUCSCSplit: list, Default [True, “KEGG Gene ID”]

Two component list, {True/False, label}. If the first component is set to True the initially imported KEGG IDs, identified by the second component label, are split on + string to fix nomenclature issues, retaining the string following +

Returns:

dictionary

Gene dictionary

Usage:

```
geneDict = GetGeneDictionary()
```



```
GOAnalysisAssigner(PyIomicaDataDirectory=None, ImportDirectly=False, BackgroundSet=[],
                     Species='human', LengthFilter=None, LengthFilterFunction=<ufunc 'greater_equal'>,
                     GOFileName=None, GOFileColumns=[2, 5],
                     GOURL='http://current.geneontology.org/annotations/')
```

Download and create gene associations and restrict to required background set.

Parameters:

PyIomicaDataDirectory: str, Default None

The directory where the default package data is stored

ImportDirectly: boolean, Default False

Import from URL regardless if the file already exists

BackgroundSet: list, Default []

Background list to create annotation projection to limited background space, involves considering pathways/groups/sets and that provides a list of IDs (e.g. gene accessions) that should be considered as the background for the calculation

Species: str, Default “human”

Species considered in the calculation, by default corresponding to human

LengthFilterFunction: function, Default np.greater_equal

Performs computations of membership in pathways/ontologies/groups/sets, that specifies which function to use to filter the number of members a reported category has compared to the number typically provided by LengthFilter

LengthFilter: int, Default None

Argument for LengthFilterFunction

GOFileName: str, Default None

The name for the specific GO file to download from the GOURL if option ImportDirectly is set to True

GOFileColumns: list, Default [2, 5]

Columns to use for IDs and [GO:accessions](#) respectively from the downloaded GO annotation file, used when ImportDirectly is set to True to obtain a new GO association file

GOURL: str, Default “<http://current.geneontology.org/annotations/>”

The location (base URL) where the GO association annotation files are downloaded from

Returns:

dictionary

Dictionary of IDToGO and GOTOID dictionaries

Usage:

```
GOassignment = GOAnalysisAssigner()
```

```
obtainConstantGeneDictionary(GeneDictionary, GetGeneDictionaryOptions, AugmentDictionary)
```

Obtain gene dictionary - if it exists can either augment with new information or Species or create new, if not exist then create variable.

Parameters:

GeneDictionary: dictionary or None

An existing variable to use as a gene dictionary in annotations. If set to None the default ConstantGeneDictionary will be used

GetGeneDictionaryOptions: dictionary

A list of options that will be passed to this internal GetGeneDictionary function

AugmentDictionary: boolean

A choice whether or not to augment the current ConstantGeneDictionary global variable or create a new one

Returns:

None

Usage:

```
obtainConstantGeneDictionary(None, {}, False)
```

```
GOAnalysis(data, GetGeneDictionaryOptions={}, AugmentDictionary=True, InputID=['UniProt ID', 'Gene Symbol'], OutputID='UniProt ID', GOAnalysisAssignerOptions={}, BackgroundSet=[], Species='human', OntologyLengthFilter=2, ReportFilter=1, ReportFilterFunction=<ufunc 'greater_equal'>, pValueCutoff=0.05, TestFunction=<function <lambda>>, HypothesisFunction=<function <lambda>>, FilterSignificant=True, OBODictionaryVariable=None, OBOGODictionaryOptions={}, MultipleListCorrection=None, MultipleList=False, GeneDictionary=None)
```

Calculate input data over-representation analysis for Gene Ontology (GO) categories.

Parameters:**data: pd.DataFrame or list**

Data to analyze

GetGeneDictionaryOptions: dictionary, Default {}

A list of options that will be passed to this internal GetGeneDictionary function

AugmentDictionary: boolean, Default True

A choice whether or not to augment the current ConstantGeneDictionary global variable or create a new one

InputID: list, Default ["UniProt ID", "Gene Symbol"]

Kind of identifiers/accessions used as input

OutputID: str, Default "UniProt ID"

Kind of IDs/accessions to convert the input IDs/accession numbers in the function's analysis

GOAnalysisAssignerOptions: dictionary, Default {}

A list of options that will be passed to the internal GOAnalysisAssigner function

BackgroundSet: list, Default []

Background list to create annotation projection to limited background space, involves considering pathways/groups/sets and that provides a list of IDs (e.g. gene accessions) that should be considered as the background for the calculation

Species: str, Default "human"

The species considered in the calculation, by default corresponding to human

OntologyLengthFilter: int, Default 2

Function that can be used to set the value for which terms to consider in the computation, by excluding GO terms that have fewer items compared to the OntologyLengthFilter value. It is used by the internal GOAnalysisAssigner function

ReportFilter: int, Default 1

Functions that use pathways/ontologies/groups, and provides a cutoff for membership in ontologies/pathways/groups in selecting which terms/categories to return. It is typically used in conjunction with ReportFilterFunction

ReportFilterFunction: function, Default np.greater_equal

Specifies what operator form will be used to compare against ReportFilter option value in selecting which terms/categories to return

pValueCutoff: float, Default 0.05

Significance cutoff

TestFunction: function, Default `lambda n, N, M, x: 1 - scipy.stats.hypergeom.cdf(x-1, M, n, N)`

Test function

HypothesisFunction: function, Default `lambda data, SignificanceLevel:`

`BenjaminiHochbergFDR(data, SignificanceLevel=SignificanceLevel)["Results"]`

Allows the choice of function for implementing multiple hypothesis testing considerations

FilterSignificant: boolean, Default True

Can be set to True to filter data based on whether the analysis result is statistically significant, or if set to False to return all membership computations

OBODictionaryVariable: str, Default None

A GO annotation variable. If set to None, OBOGODictionary will be used internally to automatically generate the default GO annotation

OBOGODictionaryOptions: dictionary, Default {}

A list of options to be passed to the internal OBOGODictionary function that provides the GO annotations

MultipleListCorrection: boolean, Default None

Specifies whether or not to correct for multi-omics analysis. The choices are None, Automatic, or a custom number, e.g protein+RNA

MultipleList: boolean, Default False

Specifies whether the input accessions list constituted a multi-omics list input that is annotated so

GeneDictionary: str, Default None

Points to an existing variable to use as a gene dictionary in annotations. If set to None the default ConstantGeneDictionary will be used

Returns:

dictionary

Enrichment dictionary

Usage:

```
goExample1 = GOAnalysis(["TAB1", "TNFSF13B", "MALT1", "TIRAP", "CHUK",
    "TNFRSF13C", "PARP1", "CSNK2A1", "CSNK2A2", "CSNK2B", "LTBR", "LYN", "MYD88",
    "GADD45B", "ATM", "NFKB1", "NFKB2", "NFKBIA", "IRAK4", "PIAS4", "PLAU"])
```

GeneTranslation(*InputList*, *TargetIDList*, *GeneDictionary*, *InputID=None*, *Species='human'*)

Use geneDictionary to convert inputList IDs to different annotations as indicated by targetIDList.

Parameters:

InputList: list

List of names

TargetIDList: list

Target ID list

GeneDictionary: dictionary

An existing variable to use as a gene dictionary in annotations. If set to None the default ConstantGeneDictionary will be used

InputID: str, Default None

The kind of identifiers/accessions used as input

Species: str, Default “human”

The species considered in the calculation, by default corresponding to human

Returns:

dictionary

Gene dictionary

Usage:

```
GenDict = GeneTranslation(data, “UniProt ID”, ConstantGeneDictionary, InputID = [“UniProt ID”, “Gene  
Symbol”], Species = “human”)
```

```
KEGGAnalysisAssigner(PyIomicaDataDirectory=None, ImportDirectly=False, BackgroundSet=[],  
KEGGQuery1='pathway', KEGGQuery2='hsa', LengthFilter=None,  
LengthFilterFunction=<ufunc 'greater_equal'>, Labels=['IDToPath', 'PathToID'])
```

Create KEGG: Kyoto Encyclopedia of Genes and Genomes pathway associations, restricted to required background set, downloading the data if necessary.

Parameters:

PyIomicaDataDirectory: str, Default None

Directory where the default package data is stored

ImportDirectly: boolean, Default False

Import from URL regardless if the file already exists

BackgroundSet: list, Default []

A list of IDs (e.g. gene accessions) that should be considered as the background for the calculation

KEGGQuery1: str, Default “pathway”

Make KEGG API calls, and sets string query1 in <http://rest.kegg.jp/link/<> query1 <> / <> query2>. Typically this will be used as the target database to find related entries by using database cross-references

KEGGQuery2: str, Default “hsa”

KEGG API calls, and sets string query2 in <http://rest.kegg.jp/link/<> query1 <> / <> query2>. Typically this will be used as the source database to find related entries by using database cross-references

LengthFilterFunction: function, Default np.greater_equal

Option for functions that perform computations of membership in pathways/ontologies/groups/sets, that specifies which function to use to filter the number of members a reported category has compared to the number typically provided by LengthFilter

LengthFilter: int, Default None

Allows the selection of how many members each category can have, as typically restricted by the LengthFilterFunction

Labels: list, Default [“IDToPath”, “PathToID”]

A string list for how keys in a created association will be named

Returns:

dictionary

IDToPath and PathToID dictionary

Usage:

```
KEGGAssignment = KEGGAnalysisAssigner()
```

```
KEGGDictionary(PyIomicaDataDirectory=None, ImportDirectly=False, KEGGQuery1='pathway',  
KEGGQuery2='hsa')
```

Create a dictionary from KEGG: Kyoto Encyclopedia of Genes and Genomes terms - typically association of pathways and members therein.

Parameters:**PyIomicaDataDirectory:** str, Default None

directory where the default package data is stored

ImportDirectly: boolean, Default False

import from URL regardless if the file already exists

KEGGQuery1: str, Default “pathway”make KEGG API calls, and sets string query1 in <http://rest.kegg.jp/link/<> query1 <> / <> query2>. Typically this will be used as the target database to find related entries by using database cross-references**KEGGQuery2:** str, Default “hsa”KEGG API calls, and sets string query2 in <http://rest.kegg.jp/link/<> query1 <> / <> query2>. Typically this will be used as the source database to find related entries by using database cross-references**Returns:****dictionary**

Dictionary of definitions

Usage:

KEGGDict = KEGGDictionary()

```
KEGGAnalysis(data, AnalysisType='Genomic', GetGeneDictionaryOptions={}, AugmentDictionary=True,
InputID=['UniProt ID', 'Gene Symbol'], OutputID='KEGG Gene ID', MolecularInputID=['cpd'],
MolecularOutputID='cpd', KEGGAnalysisAssignerOptions={}, BackgroundSet=[],
KEGGOrganism='hsa', KEGGMolecular='cpd', KEGGDatabase='pathway',
PathwayLengthFilter=2, ReportFilter=1, ReportFilterFunction=<ufunc 'greater_equal'>,
pValueCutoff=0.05, TestFunction=<function <lambda>>, HypothesisFunction=<function
<lambda>>, FilterSignificant=True, KEGGDictionaryVariable=None,
KEGGDictionaryOptions={}, MultipleListCorrection=None, MultipleList=False,
GeneDictionary=None, Species='human', MolecularSpecies='compound', NonUCSC=False,
PyIomicaDataDirectory=None)
```

Calculate input data over-representation analysis for KEGG: Kyoto Encyclopedia of Genes and Genomes pathways. Input can be a list, a dictionary of lists or a clustering object.

Parameters:**data:** pandas.DataFrame or list

Data to analyze

AnalysisType: str, Default “Genomic”

Analysis methods that may be used, “Genomic”, “Molecular” or “All”

GetGeneDictionaryOptions: dictionary, Default {}

A list of options that will be passed to this internal GetGeneDictionary function

AugmentDictionary: boolean, Default True

A choice whether or not to augment the current ConstantGeneDictionary global variable or create a new one

InputID: list, Default [“UniProt ID”, “Gene Symbol”]

The kind of identifiers/accessions used as input

OutputID: str, Default “KEGG Gene ID”

A string value that specifies what kind of IDs/accessions to convert the input IDs/accession numbers in the function’s analysis

MolecularInputID: list, Default [“cpd”]

A string list to indicate the kind of ID to use for the input molecule entries

MolecularOutputID: str, Default “cpd”

A string list to indicate the kind of ID to use for the input molecule entries

KEGGAnalysisAssignerOptions: dictionary, Default {}

A list of options that will be passed to this internal KEGGAnalysisAssigner function

BackgroundSet: list, Default []

A list of IDs (e.g. gene accessions) that should be considered as the background for the calculation

KEGGOrganism: str, Default “hsa”

Indicates which organism (org) to use for “Genomic” type of analysis (default is human analysis: org=“hsa”)

KEGGMolecular: str, Default “cpd”

Which database to use for molecular analysis (default is the compound database: cpd)

KEGGDatabase: str, Default “pathway”

KEGG database to use as the target database

PathwayLengthFilter: int, Default 2

Pathways to consider in the computation, by excluding pathways that have fewer items compared to the PathwayLengthFilter value

ReportFilter: int, Default 1

Provides a cutoff for membership in ontologies/pathways/groups in selecting which terms/categories to return. It is typically used in conjunction with ReportFilterFunction

ReportFilterFunction: function, Default np.greater_equal

Operator form will be used to compare against ReportFilter option value in selecting which terms/categories to return

pValueCutoff: float, Default 0.05

A cutoff p-value for (adjusted) p-values to assess statistical significance

TestFunction: function, Default lambda n, N, M, x: 1. - scipy.stats.hypergeom.cdf(x-1, M, n, N)

A function used to calculate p-values

HypothesisFunction: function, Default lambda data, SignificanceLevel:**BenjaminiHochbergFDR(data, SignificanceLevel=SignificanceLevel)[“Results”]**

Allows the choice of function for implementing multiple hypothesis testing considerations

FilterSignificant: boolean, Default True

Can be set to True to filter data based on whether the analysis result is statistically significant, or if set to False to return all membership computations

KEGGDictionaryVariable: str, Default None

KEGG dictionary, and provides a KEGG annotation variable. If set to None, KEGGDictionary will be used internally to automatically generate the default KEGG annotation

KEGGDictionaryOptions: dictionary, Default {}

A list of options to be passed to the internal KEGGDictionary function that provides the KEGG annotations

MultipleListCorrection: boolean, Default None

Specifies whether or not to correct for multi-omics analysis. The choices are None, Automatic, or a custom number

MultipleList: boolean, Default False

Whether the input accessions list constituted a multi-omics list input that is annotated so

GeneDictionary: str, Default None

Existing variable to use as a gene dictionary in annotations. If set to None the default ConstantGeneDictionary will be used

Species: str, Default “human”

The species considered in the calculation, by default corresponding to human

MolecularSpecies: str, Default “compound”

The kind of molecular input

NonUCSC: , Default

If UCSC browser was used in determining an internal GeneDictionary used in ID translations, where the KEGG identifiers for genes are number strings (e.g. 4790). The NonUCSC option can be set to True if standard KEGG accessions are used in a user provided GeneDictionary variable, in the form OptionValue[KEGGOrganism] <>:<>numberString, e.g. hsa:4790

PyIomicaDataDirectory: str, Default None

Directory where the default package data is stored

Returns:

dictionary

Enrichment dictionary

Usage:

```
keggExample1 = KEGGAnalysis(["TAB1", "TNFSF13B", "MALT1", "TIRAP", "CHUK",
                             "TNFRSF13C", "PARP1", "CSNK2A1", "CSNK2A2", "CSNK2B", "LTBR", "LYN", "MYD88",
                             "GADD45B", "ATM", "NFKB1", "NFKB2", "NFKBIA", "IRAK4", "PIAS4", "PLAU",
                             "POLR3B", "NME1", "CTPS1", "POLR3A"])
```

MassMatcher(*data*, *accuracy*, *MassDictionaryVariable*=None, *MolecularSpecies*='cpd')

Assign putative mass identification to input data based on monoisotopic mass (using PyIomica's mass dictionary). The accuracy in parts per million.

Parameters:

data: np.array

Input data

accuracy: float

Accuracy

MassDictionaryVariable: boolean, Default None

Mass dictionary variable. If set to None, inbuilt mass dictionary (MassDictionary) will be loaded and used

MolecularSpecies: str, Default “cpd”

The kind of molecular input

Returns:

list

List of IDs

Usage:

```
result = MassMatcher(18.010565, 2)
```

MassDictionary(*PyIomicaDataDirectory*=None)

Load PyIomica's current mass dictionary.

Parameters:

PyIomicaDataDirectory: str, Default None

Directory where the default package data is stored

Returns:

dictionary

Mass dictionary

Usage:

MassDict = MassDictionary()

ExportEnrichmentReport(*data*, *AppendString=""*, *OutputDirectory=None*)

Export results from enrichment analysis to Excel spreadsheets.

Parameters:

data: dictionary

Enrichment results

AppendString: str, Default ""

Custom report name, if empty then time stamp will be used

OutputDirectory: boolean, Default None

Path of directories where the report will be saved

Returns:

None

Usage:

ExportEnrichmentReport(goExample1, AppendString='goExample1', OutputDirectory=None)

BenjaminiHochbergFDR(*pValues*, *SignificanceLevel=0.05*)

HypothesisTesting BenjaminiHochbergFDR correction

Parameters:

pValues: 1d numpy.array

Array of p-values

SignificanceLevel: float, Default 0.05

Significance level

Returns:

dictionary

Corrected p-Values, p- and q-Value cutoffs

Usage:

result = BenjaminiHochbergFDR(pValues)

ReactomeAnalysis(*data*, *uploadURL='https://reactome.org/AnalysisService/identifiers/projection'*,
preDownloadURL='https://reactome.org/AnalysisService/download/',
postDownloadURL='/pathways/TOTAL/result.csv', *headersPOST={'accept': 'application/json'*,
'content-type': 'text/plain'}, *headersGET={'accept': 'text/CSV'*,
URLparameters= (('interactors', 'false'), ('pageSize', '20'), ('page', '1'), ('sortBy',
'ENTITIES_PVALUE'), ('order', 'ASC'), ('resource', 'TOTAL')))

Reactome POST-GET-style analysis.

Parameters:

data: pd.DataFrame or list

Data to analyze

uploadURL: str, Default 'https://reactome.org/AnalysisService/identifiers/projection'

URL for POST request

preDownloadURL: str, Default 'https://reactome.org/AnalysisService/download/'

Part 1 of URL for GET request

postDownloadURL: str, Default `‘/pathways/TOTAL/result.csv’`

Part 2 of URL for GET request

headersPOST: dict, Default `{‘accept’: ‘application/json’, ‘content-type’: ‘text/plain’}`

URL headers for POST request

headersGET: dict, Default `{‘accept’: ‘text/CSV’}`

URL headers for GET request

URLparameters: tuple, Default `((‘interactors’, ‘false’), (‘pageSize’, ‘20’), (‘page’, ‘1’), (‘sortBy’, ‘ENTITIES_PVALUE’), (‘order’, ‘ASC’), (‘resource’, ‘TOTAL’))`

Parameters for POST request

Returns:

returning

Enrichment object

Usage:

```
goExample1 = ReactomeAnalysis(["TAB1", "TNFSF13B", "MALT1", "TIRAP", "CHUK",
                              "TNFRSF13C", "PARP1", "CSNK2A1", "CSNK2A2", "CSNK2B", "LTBR", "LYN", "MYD88",
                              "GADD45B", "ATM", "NFKB1", "NFKB2", "NFKBIA", "IRAK4", "PIAS4", "PLAU"])
```

ExportReactomeEnrichmentReport(*data*, *AppendString*=",", *OutputDirectory*=None)

Export results from enrichment analysis to Excel spreadsheets.

Parameters:

data: dictionary or `pandas.DataFrame`

Reactome pathway enrichment results

AppendString: str, Default `“”`

Custom report name, if empty then time stamp will be used

OutputDirectory: boolean, Default `None`

Path of directories where the report will be saved

Returns:

None

Usage:

```
ExportReactomeEnrichmentReport(example1, AppendString='example1', OutputDirectory=None)
```

3.4 Extended DataFrame and data-processing functions

Submodule `pyiomica.extendedDataFrame`

PyIOmica Dataframe extending Pandas DataFrame with new functions

Classes:

`DataFrame`([*data*, *index*, *columns*, *dtype*, *copy*])

Class based on `pandas.DataFrame` extending capabilities into the domain of PyIOmica

Functions:

<code>mergeDataframes(listOfDataframes[, axis])</code>	Merge a list of Dataframes (outer join).
<code>getLombScarglePeriodogramOfDataFrame(df_data)</code>	Calculate Lomb-Scargle periodogram of DataFrame.
<code>getRandomSpikesCutoffs(df_data, p_cutoff[, ...])</code>	Calculate spikes cutoffs from a bootstrap of provided data, given the significance cutoff <code>p_cutoff</code> .
<code>getRandomAutocorrelations(df_data[, ...])</code>	Generate autocorrelation null-distribution from permuted data using Lomb-Scargle Autocorrelation.
<code>getRandomPeriodograms(df_data[, ...])</code>	Generate periodograms null-distribution from permuted data using Lomb-Scargle function.

class DataFrame (*data=None, index=None, columns=None, dtype=None, copy=False*)

Bases: DataFrame

Class based on pandas.DataFrame extending capabilities into the domain of PyIOMica

Initialization parameters are identical to those in pandas.DataFrame See <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html> for detail.

Methods:

<code>__init__([data, index, columns, dtype, copy])</code>	Initialization method
<code>filterOutAllZeroSignals([inplace])</code>	Filter out all-zero signals from a DataFrame.
<code>filterOutFractionZeroSignals(...[, inplace])</code>	Filter out fraction-zero signals from a DataFrame.
<code>filterOutFractionMissingSignals(...[, inplace])</code>	Filter out fraction-zero signals from a DataFrame.
<code>filterOutReferencePointZeroSignals(...)</code>	Filter out first time point zeros signals from a DataFrame.
<code>tagValueAsMissing([value, inplace])</code>	Tag zero values with NaN.
<code>tagMissingAsValue([value, inplace])</code>	Tag NaN with zero.
<code>tagLowValues(cutoff, replacement[, inplace])</code>	Tag low values with replacement value.
<code>removeConstantSignals(theta_cutoff[, inplace])</code>	Remove constant signals.
<code>boxCoxTransform([axis, inplace])</code>	Box-cox transform data.
<code>modifiedZScore([axis, inplace])</code>	Z-score (Median-based) transform data.
<code>normalizeSignalsToUnity([referencePoint, ...])</code>	Normalize signals to unity.
<code>quantileNormalize([output_distribution, ...])</code>	Quantile Normalize signals in a DataFrame.
<code>compareTimeSeriesToPoint([point, inplace])</code>	Subtract a particular point of each time series (row) of a Dataframe.
<code>compareTwoTimeSeries(df[, function, ...])</code>	Create a new Dataframe based on comparison of two existing Dataframes.
<code>imputeMissingWithMedian([axis, inplace])</code>	Normalize signals to unity.

`__init__` (*data=None, index=None, columns=None, dtype=None, copy=False*)

Initialization method

filterOutAllZeroSignals (*inplace=False*)

Filter out all-zero signals from a DataFrame.

Parameters:

inplace: boolean, Default False

Whether to modify data in place or return a new one

Returns:

Dataframe or None

Processed data

Usage:

```
df_data = df_data.filterOutAllZeroSignals()
```

or

```
df_data.filterOutAllZeroSignalse(inplace=True)
```

filterOutFractionZeroSignals(*min_fraction_of_non_zeros, inplace=False*)

Filter out fraction-zero signals from a DataFrame.

Parameters:

min_fraction_of_non_zeros: float

Maximum fraction of allowed zeros

inplace: boolean, Default False

Whether to modify data in place or return a new one

Returns:

Dataframe or None

Processed data

Usage:

```
df_data = df_data.filterOutFractionZeroSignals(0.75)
```

or

```
df_data.filterOutFractionZeroSignals(0.75, inplace=True)
```

filterOutFractionMissingSignals(*min_fraction_of_non_missing, inplace=False*)

Filter out fraction-zero signals from a DataFrame.

Parameters:

min_fraction_of_non_missing: float

Maximum fraction of allowed zeros

inplace: boolean, Default False

Whether to modify data in place or return a new one

Returns:

Dataframe or None

Processed data

Usage:

```
df_data = df_data.filterOutFractionMissingSignals(0.75)
```

or

```
df_data.filterOutFractionMissingSignals(0.75, inplace=True)
```

filterOutReferencePointZeroSignals(*referencePoint=0, inplace=False*)

Filter out out first time point zeros signals from a DataFrame.

Parameters:

referencePoint: int, Default 0

Index of the reference point

inplace: boolean, Default False

Whether to modify data in place or return a new one

Returns:

Dataframe or None

Processed data

Usage:

```
df_data = df_data.filterOutFirstPointZeroSignals()
```

or

```
df_data.filterOutFirstPointZeroSignals(inplace=True)
```

tagValueAsMissing(*value=0.0, inplace=False*)

Tag zero values with NaN.

Parameters:

inplace: boolean, Default False

Whether to modify data in place or return a new one

Returns:

Dataframe or None

Processed data

Usage:

```
df_data = df_data.tagValueAsMissing()
```

or

```
df_data.tagValueAsMissing(inplace=True)
```

tagMissingAsValue(*value=0.0, inplace=False*)

Tag NaN with zero.

Parameters:

inplace: boolean, Default False

Whether to modify data in place or return a new one

Returns:

Dataframe or None

Processed data

Usage:

```
df_data = df_data.tagMissingAsValue()
```

or

```
df_data.tagMissingAsValue(inplace=True)
```

tagLowValues(*cutoff, replacement, inplace=False*)

Tag low values with replacement value.

Parameters:

cutoff: float

Values below the “cutoff” are replaced with “replacement” value

replacement: float

Values below the “cutoff” are replaced with “replacement” value

inplace: boolean, Default False

Whether to modify data in place or return a new one

Returns:

Dataframe or None

Processed data

Usage:

```
df_data = df_data.tagLowValues(1., 1.)
```

or

```
df_data.tagLowValues(1., 1., inplace=True)
```

removeConstantSignals(*theta_cutoff*, *inplace=False*)

Remove constant signals.

Parameters:**theta_cutoff: float**

Parameter for filtering the signals

inplace: boolean, Default False

Whether to modify data in place or return a new one

Returns:**Dataframe or None**

Processed data

Usage:

```
df_data = df_data.removeConstantSignals(0.3)
```

or

```
df_data.removeConstantSignals(0.3, inplace=True)
```

boxCoxTransform(*axis=1*, *inplace=False*)

Box-cox transform data.

Parameters:**axis: int, Default 1**

Direction of processing, columns (1) or rows (0)

inplace: boolean, Default False

Whether to modify data in place or return a new one

Returns:**Dataframe or None**

Processed data

Usage:

```
df_data = df_data.boxCoxTransformDataframe()
```

or

```
df_data.boxCoxTransformDataframe(inplace=True)
```

modifiedZScore(*axis=0*, *inplace=False*)

Z-score (Median-based) transform data.

Parameters:**axis: int, Default 1**

Direction of processing, rows (1) or columns (0)

inplace: boolean, Default False

Whether to modify data in place or return a new one

Returns:**Dataframe or None**

Processed data

Usage:`df_data = df_data.modifiedZScoreDataframe()`

or

`df_data.modifiedZScoreDataframe(inplace=True)`**normalizeSignalsToUnity**(*referencePoint=0, inplace=False*)

Normalize signals to unity.

Parameters:**referencePoint: int, Default 0**

Index of the reference point

inplace: boolean, Default False

Whether to modify data in place or return a new one

Returns:**Dataframe or None**

Processed data

Usage:`df_data = df_data.normalizeSignalsToUnityDataframe()`

or

`df_data.normalizeSignalsToUnityDataframe(inplace=True)`**quantileNormalize**(*output_distribution='original', averaging=<function mean>, ties=<function mean>, inplace=False*)

Quantile Normalize signals in a DataFrame.

Note that it is possible there may be equal values within the dataset. In such a scenario, by default, the quantile normalization implementation considered here works by replacing the degenerate values with the mean over all the degenerate ranks. Note, that for the default option to work the data should not have any missing values. If `output_distribution` is set to 'uniform' or 'normal' then the scikit-learn's Quantile Transformation is used.

Parameters:**output_distribution: str, Default 'original'**

Output distribution. Other options are 'normal' and 'uniform'

averaging: function, Default np.mean

With what value to replace the same-rank elements across samples. Default is to take the mean of same-rank elements

ties: function or str, Default np.mean

Function or name of the function. How ties should be handled. Default is to replace ties with their mean. Other possible options are: 'mean', 'median', 'prod', 'sum', etc.

inplace: boolean, Default False

Whether to modify data in place or return a new one

Returns:

Dataframe or None

Processed data

Usage:

```
df_data = pd.DataFrame(index=['Gene 1','Gene 2','Gene 3','Gene 4'], columns=['Col 0','Col 1','Col 2'], data=np.array([[5, 4, 3], [2, 1, 4], [3, 4, 6], [4, 2, 8]]))
```

```
df_data = df_data.quantileNormalize()
```

or

```
df_data.df_data.quantileNormalize(inplace=True)
```

compareTimeSeriesToPoint (*point='first', inplace=False*)

Subtract a particular point of each time series (row) of a Dataframe.

Parameters:

point: str, int or float

Possible options are 'first', 'last', 0, 1, ... , 10, or a value.

inplace: boolean, Default False

Whether to modify data in place or return a new one

Returns:**Dataframe or None**

Processed data

Usage:

```
df_data = df_data.compareTimeSeriesToPoint()
```

or

```
df_data.compareTimeSeriesToPoint(df_data)
```

compareTwoTimeSeries (*df, function=<ufunc 'subtract'>, compareAllLevelsInIndex=True, mergeFunction=<function mean>*)

Create a new Dataframe based on comparison of two existing Dataframes.

Parameters:

df: pandas.DataFrame

Data to compare

function: function, Default np.subtract

Other options are np.add, np.divide, or another <ufunc>.

compareAllLevelsInIndex: boolean, Default True

Whether to compare all levels in index. If False only "source" and "id" will be compared

mergeFunction: function, Default np.mean

Input Dataframes are merged with this function, i.e. np.mean (default), np.median, np.max, or another <ufunc>.

Returns:**DataFrame or None**

Processed data

Usage:

```
df_data = df_dataH2.compareTwoTimeSeries(df_dataH1, function=np.subtract, compareAllLevelsInIndex=False, mergeFunction=np.median)
```

imputeMissingWithMedian(*axis=1, inplace=False*)

Normalize signals to unity.

Parameters:

axis: int, Default 1

Axis to apply transformation along

inplace: boolean, Default False

Whether to modify data in place or return a new one

Returns:

Dataframe or None

Processed data

Usage:

```
df_data = df_data.imputeMissingWithMedian()
```

or

```
df_data.imputeMissingWithMedian(inplace=True)
```

mergeDataframes(*listOfDataframes, axis=0*)

Merge a list of Dataframes (outer join).

Parameters:

listOfDataframes: list

List of pandas.DataFrames

axis: int, Default 0

Merge direction. 0 to stack vertically, 1 to stack horizontally

Returns:

pandas.DataFrame

Processed data

Usage:

```
df_data = mergeDataframes([df_data1, df_data2])
```

getLombScarglePeriodogramOfDataframe(*df_data, NumberOfCPUs=4, parallel=True*)

Calculate Lomb-Scargle periodogram of DataFrame.

Parameters:

df: pandas.DataFrame

Data to process

parallel: boolean, Default True

Whether to calculate in parallel mode (>1 process)

NumberOfCPUs: int, Default 4

Number of processes to create if parallel is True

Returns:

pandas.DataFrame

Lomb-Scargle periodograms

Usage:

```
df_periodograms = getLombScarglePeriodogramOfDataframe(df_data)
```


getRandomSpikesCutoffs(*df_data*, *p_cutoff*, *NumberOfRandomSamples=1000*)

Calculate spikes cutoffs from a bootstrap of provided data, given the significance cutoff *p_cutoff*.

Parameters:

df_data: pandas.DataFrame
Data where rows are normalized signals

p_cutoff: float
p-Value cutoff, e.g. 0.01

NumberOfRandomSamples: int, Default 1000
Size of the bootstrap distribution

Returns:

dictionary
Dictionary of spike cutoffs.

Usage:

```
cutoffs = getSpikesCutoffs(df_data, 0.01)
```

getRandomAutocorrelations(*df_data*, *NumberOfRandomSamples=100000*, *NumberOfCPUs=4*, *fraction=0.75*, *referencePoint=0*)

Generate autocorrelation null-distribution from permuted data using Lomb-Scargle Autocorrelation. NOTE: there should be already no missing or non-numeric points in the input Series or Dataframe

Parameters:

df_data: pandas.Series or pandas.DataFrame

NumberOfRandomSamples: int, Default 10**5
Size of the distribution to generate

NumberOfCPUs: int, Default 4
Number of processes to run simultaneously

Returns:

pandas.DataFrame
Dataframe containing autocorrelations of null-distribution of data.

Usage:

```
result = getRandomAutocorrelations(df_data)
```

getRandomPeriodograms(*df_data*, *NumberOfRandomSamples=100000*, *NumberOfCPUs=4*, *fraction=0.75*, *referencePoint=0*)

Generate periodograms null-distribution from permuted data using Lomb-Scargle function.

Parameters:

df_data: pandas.Series or pandas.DataFrame

NumberOfRandomSamples: int, Default 10**5
Size of the distribution to generate

NumberOfCPUs: int, Default 4
Number of processes to run simultaneously

Returns:

pandas.DataFrame
Dataframe containing periodograms

Usage:

```
result = getRandomPeriodograms(df_data)
```

3.5 Clustering functions

Submodule **pyiomica.clusteringFunctions**

Clustering-related functions

Functions:

<i>getEstimatedNumberOfClusters</i> (data, ..., ...)	Get estimated number of clusters using ARI with KMeans
<i>getNClustersFromLinkageElbow</i> (Y)	Get optimal number clusters from linkage.
<i>getNClustersFromLinkageSilhouette</i> (Y, data, ...)	Determine the optimal number of cluster in data maximizing the Silhouette score.
<i>runForClusterNum</i> (arguments)	Calculate Adjusted Rand Index of the data for a range of cluster numbers.
<i>getGroupingIndex</i> (data[, n_groups, method, ...])	Cluster data into N groups, if N is provided, else determine N return: linkage matrix, cluster labels, possible cluster labels.
<i>makeClusteringObject</i> (df_data, df_data_autocorr)	Make a clustering Groups-Subgroups dictionary object.
<i>exportClusteringObject</i> (ClusteringObject, ...)	Export a clustering Groups-Subgroups dictionary object to a SpreadSheet.
<i>getCommunitiesOfTimeSeries</i> (data, times[, ...])	Get communities of time series

getEstimatedNumberOfClusters(data, cluster_num_min, cluster_num_max, trials_to_do, numberOfAvailableCPUs=4, plotID=None, printScores=False)

Get estimated number of clusters using ARI with KMeans

Parameters:

data: 2d numpy.array

Data to analyze

cluster_num_min: int

Minimum possible number of clusters

cluster_num_max: int

Maximum possible number of clusters

trials_to_do: int

Number of trials to do in ARI function

numberOfAvailableCPUs: int, Default 4

Number of processes to run in parallel

plotID: str, Default None

Label for the plot of peaks

printScores: boolean, Default False

Whether to print all scores

Returns:

tuple

Largest peak, other possible peaks.

Usage:

```
n_clusters = getEstimatedNumberOfClusters(data, 1, 20, 25)
```

getNClustersFromLinkageElbow(*Y*)

Get optimal number clusters from linkage. A point of the highest acceleration of the fusion coefficient of the given linkage.

Parameters:

Y: 2d numpy.array
Linkage matrix

Returns:

int
Optimal number of clusters

Usage:

```
n_clusters = getNClustersFromLinkageElbow(Y)
```

getNClustersFromLinkageSilhouette(*Y, data, metric*)

Determine the optimal number of cluster in data maximizing the Silhouette score.

Parameters:

Y: 2d numpy.array
Linkage matrix

data: 2d numpy.array
Data to analyze

metric: str or function
Distance measure

Returns:

int
Optimal number of clusters

Usage:

```
n_clusters = getNClustersFromLinkageSilhouette(Y, data, 'euclidean')
```

runForClusterNum(*arguments*)

Calculate Adjusted Rand Index of the data for a range of cluster numbers.

Parameters:

arguments: tuple

A tuple of three parameters in the form (cluster_num, data_array, trials_to_do), where

cluster_num: int
Maximum number of clusters

data_array: 2d numpy.array
Data to test

trials_to_do: int
Number of trials for each cluster number

Returns:

1d numpy.array
Numpy array

Usage:

```
instPool = multiprocessing.Pool(processes = NumberOfAvailableCPUs)
```

```
scores = instPool.map(runForClusterNum, [(cluster_num, copy.deepcopy(data), trials_to_do) for cluster_num in range(cluster_num_min, cluster_num_max + 1)])

instPool.close()

instPool.join()
```

getGroupingIndex(*data*, *n_groups*=None, *method*='weighted', *metric*='correlation', *significance*='Elbow')

Cluster data into N groups, if N is provided, else determine N return: linkage matrix, cluster labels, possible cluster labels.

Parameters:

data: 2d numpy.array

Data to analyze

n_groups: int, Default None

Number of groups to split data into

method: str, Default 'weighted'

Linkage calculation method

metric: str, Default 'correlation'

Distance measure

significance: str, Default 'Elbow'

Method for determining optimal number of groups and subgroups

Returns:

tuple

Linkage matrix, cluster index, possible groups

Usage:

```
x, y, z = getGroupingIndex(data, method='weighted', metric='correlation', significance='Elbow')
```

makeClusteringObject(*df_data*, *df_data_autocorr*, *method*='weighted', *metric*='correlation', *significance*='Elbow')

Make a clustering Groups-Subgroups dictionary object.

Parameters:

df_data: pandas.DataFrame

Data to analyze in DataFrame format

df_data_autocorr: pandas.DataFrame

Autocorrelations or periodograms in DataFrame format

method: str, Default 'weighted'

Linkage calculation method

metric: str, Default 'correlation'

Distance measure

significance: str, Default 'Elbow'

Method for determining optimal number of groups and subgroups

Returns:

dictionary

Clustering object

Usage:

```
myObj = makeClusteringObject(df_data, df_data_autocorr, significance='Elbow')
```

exportClusteringObject(*ClusteringObject*, *saveDir*, *dataName*, *includeData=True*, *includeAutocorr=True*)

Export a clustering Groups-Subgroups dictionary object to a SpreadSheet. Linkage data is not exported.

Parameters:

ClusteringObject: dictionary

Clustering object

saveDir: str

Path of directories to save the object to

dataName: str

Label to include in the file name

includeData: boolean, Default True

Export data

includeAutocorr: boolean, Default True

Export autocorrelations of data

Returns:

str

File name of the exported clustering object

Usage:

```
exportClusteringObject(myObj, '/dir1', 'myObj')
```

getCommunitiesOfTimeSeries(*data*, *times*, *minNumberOfCommunities=2*, *horizontal=False*, *method='WDPVG'*, *direction='left'*, *weight='distance'*)

Get communities of time series

Parameters:

data: 1d numpy.array

Data array

times: 1d numpy.array

Times corresponding to data points

minNumberOfCommunities: int, Default 2

Number of communities to find depends on the number of splits. This parameter is ignored in methods that automatically estimate optimal number of communities.

horizontal: boolean, Default False

Whether to use horizontal of normal visibility graph

method: str, Default 'betweenness centrality'

Name of the method to use:

'Girvan_Newman': edge betweenness centrality based approach

'betweenness centrality': reflected graph node betweenness centrality based approach

'WDPVG': weighted dual perspective visibility graph method (also set weight variable)

direction: str, default 'left'

The direction that nodes aggregate to communities:

None: no specific direction, e.g. both sides. 'left': nodes can only aggregate to the left side hubs, e.g. early hubs 'right': nodes can only aggregate to the right side hubs, e.g. later hubs

weight: str, Default 'distance'

Type of weight for method='WDPVG':

None: no weighted

'time': weight = abs(times[i] - times[j])

'tan': weight = abs((data[i] - data[j])/(times[i] - times[j])) + 10**(-8)

'distance': weight = A[i, j] = A[j, i] = ((data[i] - data[j])**2 + (times[i] - times[j])**2)**0.5

Returns:**(list, graph)**

List of communities and a networkx graph

Usage:

res = getCommunitiesOfTimeSeries(data, times)

3.6 Frequency Based Subject Match

Submodule `pyiomica.frequencySubjectMatch`**Functions:**

<code>bootstrapGeneral(df, N[, shuffling])</code>	To generate bootstrap samples
<code>calculateLinksBetweenSubjectsByDistance(df1, ...)</code>	To calculate the linked time series/Genes from two dataframes base on the Euclidean distance
<code>calculateLinksBetweenSubjectsByCorrelation(...)</code>	To calculate the linked time series/Genes from two dataframes base on the pearson correlation
<code>getCommunityStructure(cs)</code>	To change community structure from {node1:community1, node2:community2,...} to {community1:[node1, node2,...], community2:[node3, node4,...]}
<code>getCommunityGenesDict(community_structure, ...)</code>	To get gene IDs list of each community within selected individuals' category
<code>splitGenes(community_gene_dict)</code>	Split gene ids, to seperate the genes name from attached labels
<code>getCommunityTopGenesByNumber(...[, ...])</code>	To get the top ranking genes of each community
<code>getCommunityTopGenesByFrequencyRanking(...)</code>	To get the top frequency genes of each community
<code>optimizeK(df, rangeK[, saveFig])</code>	To optimize the k value of k-mean cluster

bootstrapGeneral (*df*, *N*, *shuffling=True*)

To generate bootstrap samples

Parameters:**df: pandas dataframe**

the source dataframe using to generate bootstrap samples

N: integer

the size of bootstrap samples

shuffling

[boolean] shuffle the data or not, The default is True.

Returns:**bootstrapDF: pandas dataframe**

the bootstrap samples

calculateLinksBetweenSubjectsByDistance(*df1*, *df2*, *cutoff*)

To calculate the linked time series/Genes from two dataframes base on the Euclidean distance

Parameters:

df1: pandas dataframe

the first time series from df1

df2: pandas dataframe

the second time series from df2

cutoff: float

if the distance between two time series less than cutoff, the two time series is linked time series

Returns:

numlinkedGenes: integer/float

number of linked time series

commonGenes: integer/float

number of common time series in df1 and df2

linkedGenes: list of string

the ids/names of linked time series

calculateLinksBetweenSubjectsByCorrelation(*df1*, *df2*, *cutoff*)

To calculate the linked time series/Genes from two dataframes base on the pearson correlation

Parameters:

df1: pandas dataframe

the first time series from df1

df2: pandas dataframe

the second time series from df2

cutoff: float

if the pearson correlation between two time series less than cutoff, the two time series is linked time series

Returns:

numlinkedGenes: integer/float

number of linked time series

commonGenes: integer/float

number of common time series in df1 and df2

linkedGenes: list of string

the ids/names of linked time series

getCommunityStructure(*cs*)

To change community structure from {node1:community1, node2:community2,...} to {community1:[node1, node2,...], community2:[node3, node4,...]}

Parameters:

cs: dictionary

the community structure as {node1:community1, node2:community2,...}

Returns:

community_structure: dictionary

the community structure as {community1:[node1, node2,...], community2:[node3, node4,...]}

getCommunityGenesDict(*community_structure*, *genelist*, *endwithString*)

To get gene IDs list of each community within selected individuals' category

Parameters:

community_structure: dictionary

the community structure as {community1:[node1, node2,...], community2:[node3, node4,...]}

genelist: dictionary

the gene list of each individuals, the key is the id of individual

endwithString: list of string

the selected individuals categories, which attached to the end of the individual ids

Returns:

community_genes_dict: dictionary

the genes list of each community

splitGenes(*community_gene_dict*)

Split gene ids, to separate the genes name from attached labels

Parameters:

community_gene_dict: dictionary

the genes ids list of each community

Returns:

new_dict: dictionary

the gene names list of each community

getCommunityTopGenesByNumber(*community_structure*, *genelist*, *endwithString*, *numberOfTopGenes=500*)

To get the top ranking genes of each community

Parameters:

community_structure: dictionary

the community structure as {community1:[node1, node2,...], community2:[node3, node4,...]}

genelist: dictionary

the genes list of each community

endwithString: list of string

the selected individuals categories, which attached to the end of the individual ids

numberOfTopGenes: integer, optional

the number of top ranking genes. The default is 500.

Returns:

community_genes_dict: dictionary

the top ranking genes of each community

getCommunityTopGenesByFrequencyRanking(*community_structure*, *genelist*, *endwithString*,
frequencyPercentage=50)

To get the top frequency genes of each community

Parameters:

community_structure: dictionary

the community structure as {community1:[node1, node2,...], community2:[node3, node4,...]}

genelist: dictionary

the genes list of each community

endwithString: list of string

the selected individuals categories, which attached to the end of the individual ids

frequencyPercentage: float, optional

the top percentage frequency of choosed genes, The default is 50.

Returns:

community_genes_dict: dictionary

the top percentage frequency genes of each community

optimizeK(*df*, *rangeK*, *saveFig=False*, ***kargs*)

To optimize the k value of k-mean cluster

Parameters:

df: pandas dataframe

the data source to do k-mean cluster

rangeK: python range, e.g. `rangeK = range(0,10)`

the K value range

saveFig: boolean, optional

save figure or not. The default is False.

****kargs:** figure name

if saveFig is true, the **kargs is the figure name

Returns:

optimizek: integer

the optimized K value

3.7 Visibility graph preparation functions

Submodule **pyiomica.visibilityGraphAuxiliaryFunctions**

Functions to generate adjacency matrix of visibility graphs

Functions:

<code>getAdjacencyMatrixOfNVG(data, times)</code>	Calculate adjacency matrix of visibility graph.
<code>getAdjacencyMatrixOfNVGbyNUMPY(data, times)</code>	Calculate adjacency matrix of visibility graph.
<code>getAdjacencyMatrixOfHVG(data)</code>	Calculate adjacency matrix of horizontal visibility graph.
<code>getAdjacencyMatrixOfHVGbyNUMPY(data)</code>	Calculate adjacency matrix of horizontal visibility graph.

getAdjacencyMatrixOfNVG(*data*, *times*)

Calculate adjacency matrix of visibility graph. JIT-accelerated version (a bit faster than NumPy-accelerated version). Allows use of Multiple CPUs.

Parameters:

data: 2d numpy.array

Numpy array of floats

times: 1d numpy.array
Numpy array of floats

Returns:

2d numpy.array
Adjacency matrix

Usage:

A = getAdjacencyMatrixOfNVG(data, times)

getAdjacencyMatrixOfNVGbyNUMPY(data, times)

Calculate adjacency matrix of visibility graph. NumPy-accelerated version. Somewhat slower than JIT-accelerated version. Use in serial applications.

Parameters:

data: 2d numpy.array
Numpy array of floats

times: 1d numpy.array
Numpy array of floats

Returns:

2d numpy.array
Adjacency matrix

Usage:

A = getAdjacencyMatrixOfNVGbyNUMPY(data, times)

getAdjacencyMatrixOfHVG(data)

Calculate adjacency matrix of horizontal visibility graph. JIT-accelerated version (a bit faster than NumPy-accelerated version). Single-threaded beats NumPy up to 2k data sizes. Allows use of Multiple CPUs.

Parameters:

data: 2d numpy.array
Numpy array of floats

Returns:

2d numpy.array
Adjacency matrix

Usage:

A = getAdjacencyMatrixOfHVG(data)

getAdjacencyMatrixOfHVGbyNUMPY(data)

Calculate adjacency matrix of horizontal visibility graph. NumPy-accelerated version. Use with datasets larger than 2k. Use in serial applications.

Parameters:

data: 2d numpy.array
Numpy array of floats

Returns:

2d numpy.array
Adjacency matrix

Usage:

A = getAdjacencyMatrixOfHVGbyNUMPY(data)

3.8 Visibility Graph Community Detection

Submodule **pyiomica.visibilityGraphCommunityDetection**

Visibility Graph Community detection functions.

Functions:

<code>createVisibilityGraph(data, times[, ...])</code>	Calculate adjacency matrix of visibility graph, create the <code>networkx.Graph</code> network.
<code>communityDetectByPathLength(G[, ...])</code>	Calculate community structure by shortest path length algorithm.

createVisibilityGraph(data, times, graph_type='natural', weight=None, withsign=False)

Calculate adjacency matrix of visibility graph, create the `networkx.Graph` network.

Parameters:

data: 2d numpy.array

Data to process

times: 1d numpy.array

Times corresponding to provided data points

weight: str, Default None

Type of normalization:

None: no weighted

'time': $\text{weight} = \text{abs}(\text{times}[i] - \text{times}[j])$

'tan': $\text{weight} = \text{abs}((\text{data}[i] - \text{data}[j]) / (\text{times}[i] - \text{times}[j])) + 10^{**}(-8)$

'distance': $\text{weight} = A[i, j] = A[j, i] = ((\text{data}[i] - \text{data}[j])**2 + (\text{times}[i] - \text{times}[j])**2)**0.5$

graph_type: str, Default 'natural'

Type of the visibility graph:

"horizontal", Horizontal Visibility Graph

"natural", Natural Visibility Graph

"dual_horizontal", Dual Perspective Horizontal Visibility Graph

"dual_natural", Dual Perspective Natural Visibility Graph

withsign: boolean, Default False

Whether to return the sign of adjacency matrix, the link from normal perspective VG is positive, the link from reflected perspective VG is negative

Returns: tuple

Tuple of two objects:

networkx.Graph

Graph of `networkx` type

2d numpy.array

Adjacency matrix

Usage:

`A = createVisibilityGraph(data, times)`

communityDetectByPathLength(*G*, *outputTimePoints=False*, *setSourceTarget=None*, *direction=None*, *cutoff=None*)

Calculate community structure by shortest path length algorithm.

Parameters:

G: networkx.Graph

Graph of networkx type

outputTimePoints: boolean, default False

Whether to output timepoints for communities or indices

setSourceTarget: list, default None

List for nodes to consider to use when finding weighted shortest path. First element used for source (starting node), and last for target (ending node)

direction: str, default None

The direction that nodes aggregate to communities:

None: no specific direction, e.g. both sides.

‘left’: nodes can only aggregate to the left side hubs, e.g. early hubs

‘right’: nodes can only aggregate to the right side hubs, e.g. later hubs

cutoff: int, float or str, Default None

Cutoff is used to combine initial communities, e.g. whenever the shortest path length of two adjacent hub nodes is smaller than cutoff, the communities with the two hub nodes will be combined:

int or float: the percentile of all shortest path length distribution, between 0 ~ 100

‘auto’: use optimized cutoff

None: no cutoff

Returns:

list of list

Detected communities in the form of nested list.

Usage:

`c = communityDetectByPathLength(G)`

3.9 Visualization functions

Submodule **pyiomica.visualizationFunctions**

Visualization functions

Functions:

<code>saveFigure(fig, saveDir, label, extension, dpi)</code>	Function primarily used internally to save and close figures
<code>makeDataHistograms(df, saveDir, dataName[, ...])</code>	Make a histogram for each Series (time point) in a Dataframe.
<code>makeLombScarglePeriodograms(df, saveDir, ...)</code>	Make a combined plot of the signal and its Lomb-Scargle periodogram for each pandas Series (time point) in a Dataframe.
<code>addVisibilityGraph(data, times[, dataName, ...])</code>	Draw a Visibility graph of data on a provided Matplotlib figure.
<code>makeVisibilityGraph(intensities, positions, ...)</code>	Make either horizontal or normal visibility graph of a time series using function <code>addVisibilityGraph</code> .
<code>makeVisibilityBarGraph(data, times, saveDir, ...)</code>	Bar-plot style visibility graph.
<code>makePlotOfPeak(data_all, scores, ...)</code>	Plot peaks.
<code>addColorbarToFigure(fig, data[, ...])</code>	Add colorbar to figure
<code>makeDendrogramHeatmapOfClusteringObject(...)</code>	Make Dendrogram-Heatmap plot along with Visibility graphs.
<code>plotHVGBarGraphDual(A, data, times, fileName)</code>	Bar-plot style horizontal visibility graph with different link colors for different perspectives
<code>plotNVGBarGraphDual(A, data, times, fileName)</code>	Bar-plot style natural visibility graph with different link colors for different perspectives

saveFigure(fig, saveDir, label, extension, dpi, close=True)

Function primarily used internally to save and close figures

Parameters:

saveDir: str

Path of directories to save the object to

extension: str, Default 'png'

Path of directories to save the object to

dpi: int, Default 300

Figure resolution if rasterized

close: boolean: Default True

Whether to close the figure after saving

Returns:

None

Usage:

`saveFigure(fig, saveDir, label, extension, dpi)`

makeDataHistograms(df, saveDir, dataName, figsize=(8, 8), range_min=<function amin>, range_max=<function amax>, includeTitle=True, title='Data @ timePoint:', fontsize=8, fontcolor='b', N_bins=100, color='b', extension='.png', dpi=300)

Make a histogram for each Series (time point) in a Dataframe.

Parameters:

df: pandas.DataFrame

Data to visualize

saveDir: str

Path of directories to save the object to

dataName: str
Label to include in the file name

figsize: tuple, Default (8,8)
Size of the figure in inches

range_min: str, Default
How to determine data minimum

range_max: int, float or function, Default
How to determine data maximum

includeTitle: boolean, Default True
Path of directories to save the object to

title: str, Default 'Data @ timePoint:'
Text of the title

fontsize: str, Default 8
Fontsize of the labels

fontcolor: str, Default 'b'
Color of the title font

N_bins: int, Default 100
Number of bins in the histogram

color: str, Default 'b'
Color of the bars

extension: str, Default '.png'
Path of directories to save the object to

dpi: int, Default 300
Figure resolution if rasterized

Returns:
None

Usage:
makeDataHistograms(df, 'dir1', 'myData')

makeLombScarglePeriodograms(df, saveDir, dataName, minNumberOfNonzeroPoints=5,
oversamplingRate=100, figsize=(5, 5), title1='TimeSeries Data',
title2='Lomb-Scargle periodogram', extension='.png', dpi=300)

Make a combined plot of the signal and its Lomb-Scargle periodogram for each pandas Series (time point) in a Dataframe.

Parameters:

df: pandas.DataFrame
Data to visualize

saveDir: str
Path of directories to save the object to

dataName: str
Label to include in the file name

minNumberOfNonzeroPoints: int, Default 5
Minimum number of non-zero points in signal to use it

oversamplingRate: int, Default 100
Periodogram oversampling rate

figsize: tuple, Default (8,8)

Size of the figure in inches

title1: str, Default ‘TimeSeries Data’

Text of the upper title

title2: str, Default ‘Lomb-Scargle periodogram’

Text of the lower title

extension: str, Default ‘.png’

Path of directories to save the object to

dpi: int, Default 300

Figure resolution if rasterized

Returns:

None

Usage:

```
makeLombScarglePeriodograms(df, 'dir1', 'myData')
```

```
addVisibilityGraph(data, times, dataName='G1S1', coords=[0.05, 0.95, 0.05, 0.95], numberOfVGs=1,
                    groups_ac_colors=['b'], fig=None, numberOfCommunities=6, printCommunities=False,
                    fontsize=None, nodesize=None, level=0.55, commLineWidth=0.5, lineWidth=1.0,
                    withLabel=True, withTitle=False, layout='circle', radius=0.07, noplot=False,
                    horizontal=False, communities=None, minNumberOfCommunities=2,
                    communitiesMethod='betweenness_centrality', direction='left', weight='distance')
```

Draw a Visibility graph of data on a provided Matplotlib figure. We represent each timepoint in a series as a node. Temporal events are detected and indicated with solid blue lines encompassing groups of points, or communities. The shortest path identifies nodes (i.e. timepoints) that display high intensity, and thus dominate the global signal profile, are robust to noise, and are likely drivers of the global temporal behavior.

Parameters:

data: 2d numpy.array

Array of data to visualize

times: 1d numpy.array

Times corresponding to each data point, used for labels

dataName: str, Default ‘G1S1’

label to include in file name

coords: list, Default [0.05,0.95,0.05,0.95]

Coordinates of location of the plot on the figure

numberOfVGs: int, Default 1

Number of plots to add to this figure

groups_ac_colors: list, Default ['b']

Colors corresponding to different groups of graphs

fig: matplotlib.figure, Default None

Figure object

numberOfCommunities: int, Default 6

Number of communities

printCommunities: boolean, Default False

Whether to print communities details to screen

fontsize: float, Default None

Size of labels

nodesize: float, Default None

Size of nodes

level: float, Default 0.55

Distance of the community lines to nodes

commLineWidth: float, Default 0.5

Width of the community lines

lineWidth: float, Default 1.0

Width of the edges between nodes

withLabel: boolean, Default True

Whether to include label on plot

withTitle: boolean, Default False

Whether to include title on plot

layout: str, Default 'circle'

Type of the layout. Other option is 'line'

radius: float, Default 0.07

Radius of the circle

noplot: boolean, Default False

Whether to make a plot or only calculate communities

horizontal: boolean, Default False

Whether to use horizontal or natural visibility graph.

communities: tuple, Default None

A tuple containing communities sturcture of network, and networkx Graph:

List of list, e.g. [[],[],...]

networkx.Graph

minNumberOfCommunities: int, Default 2

Number of communities to find depends on the number of splits. This parameter is ignored in methods that automatically estimate optimal number of communities.

communitiesMethod: str, Default 'betweenness centrality'

String defining the method to use for communitiy detection:

'Girvan_Newman': edge betweenness centrality based approach

'betweenness centrality': reflected graph node betweenness centrality based approach

'WDPVG': weighted dual perspective visibility graph method (note to also set weight variable)

direction:str, default 'left'

The direction that nodes aggregate to communities:

None: no specific direction, e.g. both sides.

'left': nodes can only aggregate to the left side hubs, e.g. early hubs

'right': nodes can only aggregate to the right side hubs, e.g. later hubs

weight: str, Default 'distance'

Type of weight for communitiesMethod='WDPVG':

None: no weighted

'time': weight = abs(times[i] - times[j])


```

'tan': weight = abs((data[i] - data[j])/(times[i] - times[j])) + 10*(-8)
'distance': weight = A[i, j] = A[j, i] = ((data[i] - data[j])**2 + (times[i] - times[j])**2)**0.5

```

Returns:

tuple
(graph_nx, data, communities)

Usage:

```

addVisibilityGraph(exampleData, exampleTimes, fig=fig, fontsize=16, nodesize=700,
                    level=0.85, commLineWidth=3.0, lineWidth=2.0, withLabel=False)

```

```

makeVisibilityGraph(intensities, positions, saveDir, fileName, communities=None,
                    minNumberOfCommunities=2, communitiesMethod='betweenness_centrality',
                    direction='left', weight='distance', printCommunities=False, fontsize=16, nodesize=500,
                    level=0.5, commLineWidth=3.0, lineWidth=2.0, layout='circle', horizontal=False,
                    radius=0.03, figsize=(10, 10), addColorbar=True, colorbarAxisCoordinates=[0.9, 0.7,
                    0.02, 0.2], colorbarLabelsSize=12, colorbarPrecision=2, extension='png', dpi=300)

```

Make either horizontal or normal visibility graph of a time series using function `addVisibilityGraph`. We represent each timepoint in a series as a node. Temporal events are detected and indicated with solid blue lines encompassing groups of points, or communities. The shortest path identifies nodes (i.e. timepoints) that display high intensity, and thus dominate the global signal profile, are robust to noise, and are likely drivers of the global temporal behavior.

Parameters:**intensities:**

Data to plot

positions:

Time points corresponding to data

saveDir: str

Path of directories to save the object to

fileName: str

Label to include in the file name

horizontal: boolean, Default False

Whether to use horizontal or natural visibility graph. Note that if `communitiesMethod` 'WDPVG' is set, this setting has no effect.

communities: tuple, Default None

A tuple containing communities sturcture of network, and networkx Graph:

List of list, e.g. `[[[]],[],...]`

`networkx.Graph`

minNumberOfCommunities: int, Default 2

Number of communities to find depends on the number of splits. This parameter is ignored in methods that automatically estimate optimal number of communities.

communitiesMethod: str, Default 'betweenness_centrality'

String defining the method to use for communitiy detection:

'Girvan_Newman': edge betweenness centrality based approach

'betweenness_centrality': reflected graph node betweenness centrality based approach

'WDPVG': weighted dual perspective visibility graph method (note to also set weight variable)

direction: str, default 'left'

The direction that nodes aggregate to communities:

None: no specific direction, e.g. both sides.

'left': nodes can only aggregate to the left side hubs, e.g. early hubs

'right': nodes can only aggregate to the right side hubs, e.g. later hubs

weight: str, Default 'distance'

Type of weight for communitiesMethod='WDPVG':

None: no weighted

'time': $\text{weight} = \text{abs}(\text{times}[i] - \text{times}[j])$

'tan': $\text{weight} = \text{abs}((\text{data}[i] - \text{data}[j]) / (\text{times}[i] - \text{times}[j])) + 10^{**}(-8)$

'distance': $\text{weight} = A[i, j] = A[j, i] = ((\text{data}[i] - \text{data}[j])**2 + (\text{times}[i] - \text{times}[j])**2)**0.5$

printCommunities: boolean, Default False

Whether to print communities details to screen

fontsize: float, Default 16

Labels fontsize

nodesize: int, Default 500

Node size

level: float, Default 0.5

Level

commLineWidth: float, Default 3.0

Communities lines width

lineWidth: float, Default 2.0

Edge lines width

layout: str, Default 'circle'

Type of layout, 'circle' or 'line'

horizontal: boolean, Default False

Whether to make horizontal of normal visibility graph

radius: float, Default 0.03

Rounding of the lines

figsize: tuple, Default (10,10)

Figure size in inches

addColorbar: boolean, Default True

Whether to add colorbar

colorbarAxisCoordinates: list, Default [0.90,0.7,0.02,0.2]

colorbar axis coordinates

colorbarLabelsize: float, Default 12

Colorbar labels size

colorbarPrecision: int, Default 2

colorbar labels rounding

extension: str, Default '.png'

Figure extension

dpi: int, Default 300

Figure resolution

Returns:

None

Usage:`makeVisibilityGraph(data, times, 'dir1/', 'myData')`

makeVisibilityBarGraph(*data, times, saveDir, fileName, AdjacencyMatrix=None, horizontal=False, barWidth=0.2, dotColor='b', barColor='r', arrowColor='k', id='', extension='.png', figsize=(8, 4), dpi=300*)

Bar-plot style visibility graph. Representing the intensities as bars, this is equivalent to connecting the top of each bar to another top if there is a direct line-of-sight to that top. The resulting visibility graph has characteristics that reflect the equivalent time series temporal structure and can be used to identify trends.

Parameters:**data: 2d numpy.array**

Numpy array of floats

times: 2d numpy.array

Numpy array of floats

fileName: str

Path where to save the figure file

fileName: str

Name of the figure file to save

AdjacencyMatrix: 2d numpy.array, Default None

Adjacency matrix of network

horizontal: boolean, default False

Horizontal or normal visibility graph

barWidth: float, default 0.2

Horizontal or normal visibility graph

dotColor: str, default 'b'

Color of the data points

barColor: str, default 'r'

Color of the bars

arrowColor: str, default 'k'

Color of lines

id: str or int, default ''

Label to add to the figure title

extension: str, Default '.png'

Figure format

figsize: tuple of int, Default (8,4)

Figure size in inches

dpi: int, Default 300

Figure resolution

Returns:

None

Usage:

makeVisibilityBarGraph(A, data, times, 'my_figure')

makePlotOfPeak(data_all, scores, selected_peak, selected_peak_value, plotID)

Plot peaks. Function used internally during certain data processing steps

addColorbarToFigure(fig, data, axisCoordinates=[0.9, 0.7, 0.02, 0.2], cmap=None, norm=None, labelsize=12, precision=2)

Add colorbar to figure

Parameters:

fig: matplotlib.figure

Data to plot

cmap: matplotlib.colors.LinearSegmentedColormap, Default None

Colormap to use

norm: matplotlib.colors.Normalize, Default None

Colormap normalization

axisCoordinates: list, Default [0.90,0.7,0.02,0.2]

colorbar axis coordinates

labelsize: float, Default 12

Colorbar labels size

precision: int, Default 2

Colorbar labels rounding

Returns:

None

Usage:

addColorbarToFigure(fig, data)

makeDendrogramHeatmapOfClusteringObject(ClusteringObject, saveDir, dataName, AutocorrNotPeriodogr=True, textScale=1.0, figsize=(12, 8), extension='.png', dpi=300, xLabel='Time', plotLabel='Transformed Expression', horizontal=False, minNumberOfCommunities=2, communitiesMethod='WDPVG', direction='left', weight='distance')

Make Dendrogram-Heatmap plot along with Visibility graphs.

Parameters:

ClusteringObject:

Clustering object

saveDir: str

Path of directories to save the object to

dataName: str

Label to include in the file name

AutocorrNotPeriodogr: boolean, Default True

Whether to use autocorrelation method instead of periodograms

textScale: float, Default 1.0

scaling of text size

figsize: tuple, Default (12,8)

Figure size in inches

extension: str, Default ‘.png’

Figure format extension

dpi: int, Default 300

Figure resolution

xLabel: str, Default ‘Time’

Label for the x axis in the heatmap

plotLabel: str, Default ‘Transformed Expression’

Label for the heatmap plot

horizontal: boolean, Default False

Whether to use horizontal or natural visibility graph. Note that if communitiesMethod ‘WDPVG’ is set, this setting has no effect.

minNumberOfCommunities: int, Default 2

Number of communities to find depends on the number of splits. This parameter is ignored in methods that automatically estimate optimal number of communities.

communitiesMethod: str, Default ‘WDPVG’

String defining the method to use for communitiy detection:

‘Girvan_Newman’: edge betweenness centrality based approach

‘betweenness_centrality’: reflected graph node betweenness centrality based approach

‘WDPVG’: weighted dual perspective visibility graph method (note to also set weight variable)

direction:str, default ‘left’

The direction that nodes aggregate to communities:

None: no specific direction, e.g. both sides.

‘left’: nodes can only aggregate to the left side hubs, e.g. early hubs

‘right’: nodes can only aggregate to the right side hubs, e.g. later hubs

weight: str, Default ‘distance’

Type of weight for communitiesMethod=‘WDPVG’:

None: no weighted

‘time’: $\text{weight} = \text{abs}(\text{times}[i] - \text{times}[j])$

‘tan’: $\text{weight} = \text{abs}((\text{data}[i] - \text{data}[j]) / (\text{times}[i] - \text{times}[j])) + 10^{**}(-8)$

‘distance’: $\text{weight} = A[i, j] = A[j, i] = ((\text{data}[i] - \text{data}[j])**2 + (\text{times}[i] - \text{times}[j])**2)**0.5$

Returns:

None

Usage:

`makeDendrogramHeatmap(myObj, ‘/dir1’, ‘myData’, AutocorrNotPeriodogr=True)`

plotHVGBarGraphDual(*A*, *data*, *times*, *fileName*, *title*=“, *fontsize*=8, *barwidth*=0.05, *figsize*=(8, 4), *dpi*=600)

Bar-plot style horizontal visibility graph with different link colors for different perspectives

Parameters:

A: 2d numpy.array

Adjacency matrix

data: 2d numpy.array

Data used to make the visibility graph

times: 1d numpy.array
Times corresponding to data points

fileName: str
Name of the figure file to save

title: str, Default ''
Label to add to the figure title

figsize: tuple of int, Default (8,4)
Figure size in inches

dpi: int, 600
Resolution of the image

barwidth: float, Default 0.05
The bar width

fontsize:int, Default 8
The text font size

Returns:
None

Usage:
PlotHorizontalVisibilityGraph(A, data, times, 'Figure.png', 'Test Data')

plotNVGBarGraphDual(A, data, times, fileName, title='', fontsize=8, barwidth=0.05, figsize=(8, 4), dpi=600)

Bar-plot style natural visibility graph with different link colors for different perspectives

Parameters:

A: 2d numpy.array
Adjacency matrix

data: 2d numpy.array
Data used to make the visibility graph

times: 1d numpy.array
Times corresponding to data points

fileName: str
Name of the figure file to save

title: str, Default ''
Label to add to the figure title

figsize: tuple of int, Default (8,4)
Figure size in inches

dpi: int, 600
Resolution of the image

barwidth: float, Default 0.05
The bar width

fontsize:int, Default 8
The text font size

Returns:
None

Usage:
PlotVisibilityGraph(A, data, times, 'Figure.png', 'Test Data')

3.10 Utility functions

Submodule **pyiomica.utilityFunctions**

Utility functions

Functions:

<code>readMathIOmicaData(fileName)</code>	Read text files exported by MathIOmica and convert to Python data
<code>runCPUs(NumberOfAvailableCPUs, func, ...)</code>	Parallelize function call with multiprocessing.Pool.
<code>createReverseDictionary(inputDictionary)</code>	Efficient way to create a reverse dictionary from a dictionary.
<code>createDirectories(path)</code>	Create a path of directories, unless the path already exists.

readMathIOmicaData(*fileName*)

Read text files exported by MathIOmica and convert to Python data

Parameters:

fileName: str

Path of directories and name of the file containing data

Returns:

data

Python data

Usage:

```
data = readMathIOmicaData("../MathIOmica/MathIOmica/MathIOmicaData/ExampleData/rnaExample")
```

runCPUs(*NumberOfAvailableCPUs, func, list_of_tuples_of_func_params*)

Parallelize function call with multiprocessing.Pool.

Parameters:

NumberOfAvailableCPUs: int

Number of processes to create

func: function

Function to apply, must take at most one argument

list_of_tuples_of_func_params: list

Function parameters

Returns:

2d numpy.array

Results of func in a numpy array

Usage:

```
results = runCPUs(4, pAutocorrelation, [(times[i], data[i], allTimes) for i in range(10)])
```

createReverseDictionary(*inputDictionary*)

Efficient way to create a reverse dictionary from a dictionary. Utilizes Pandas.DataFrame.groupby and Numpy arrays indexing.

Parameters:

inputDictionary: dictionary

Dictionary to reverse

Returns:

dictionary

Reversed dictionary

Usage:

```
revDict = createReverseDictionary(Dict)
```

createDirectories(*path*)

Create a path of directories, unless the path already exists.

Parameters:

path: str

Path directory

Returns:

None

Usage:

```
createDirectories("/pathToFolder1/pathToSubFolder2")
```

3.11 Data Storage

Submodule **pyiomica.dataStorage**

Data storage functions

Functions:

<code>read(fileName[, withPKLZextension, ...])</code>	Read object from a file recorded by function "write".
<code>write(data, fileName[, withPKLZextension, ...])</code>	Write object into a file.

read(*fileName*, *withPKLZextension*=True, *hdf5fileName*=None, *jsonFormat*=False)

Read object from a file recorded by function "write". Pandas and Numpy objects are read from HDF5 file when provided, otherwise attempt to read from PKLZ file.

Parameters:

fileName: str

Path of directories ending with the file name

withPKLZextension: boolean, Default True

Add ".pklz" to a pickle file

hdf5fileName: str, Default None

Path of directories ending with the file name. If None then data is read from a pickle file

jsonFormat: boolean, Default False

Save data into compressed json file

Returns:

data

Data object to write into a file

Usage:

```
exampleDataFrame = read('/dir1/exampleDataFrame', hdf5fileName='/dir2/data.h5')
```

```
write(data, fileName, withPKLZextension=True, hdf5fileName=None, jsonFormat=False)
```

Write object into a file. Pandas and Numpy objects are recorded in HDF5 format when 'hdf5fileName' is provided otherwise pickled into a new file.

Parameters:**data: any type**

Data object to write into a file

fileName: str

Path of directories ending with the file name

withPKLZextension: boolean, Default True

Add ".pklz" to a pickle file

hdf5fileName: str, Default None

Path of directories ending with the file name. If None then data is pickled

jsonFormat: boolean, Default False

Save data into compressed json file

Returns:

None

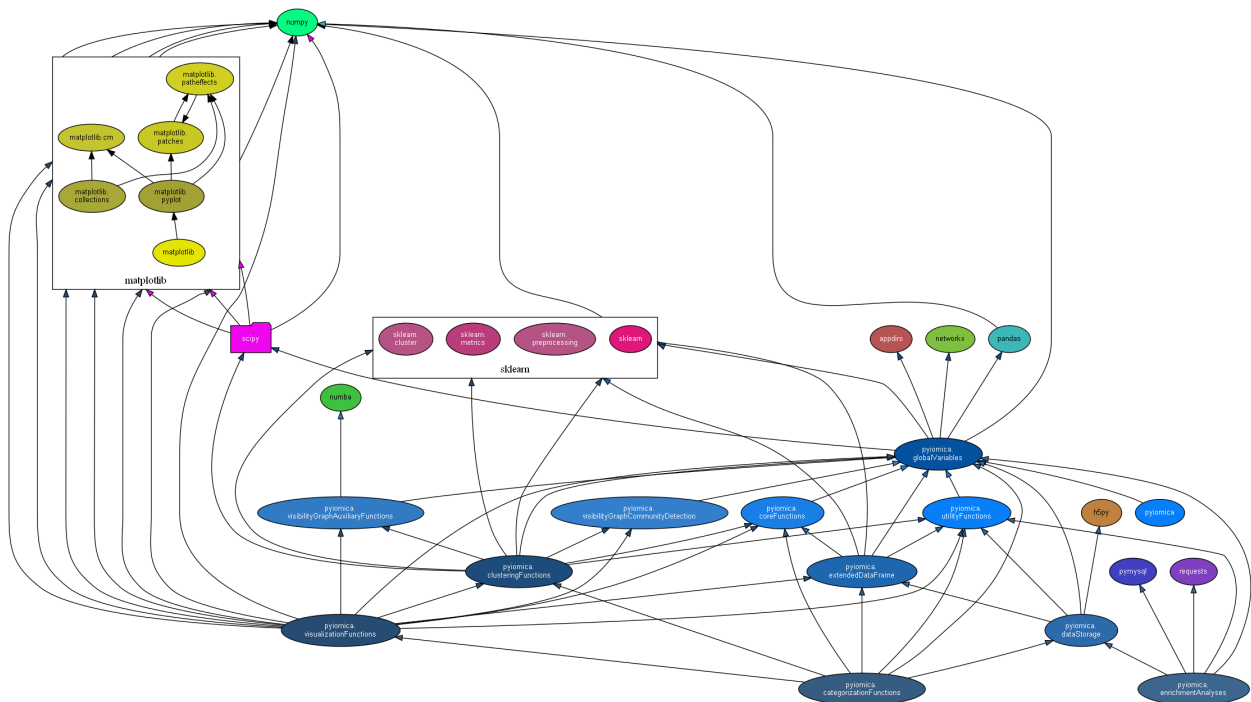
Usage:

```
write(exampleDataFrame, '/dir1/exampleDataFrame', hdf5fileName='/dir2/data.h5')
```


DEPENDENCIES

This graph was generated with **Python** module dependency visualization tool `pydeps`, see [GitHub](#), by running the following (after installation of the necessary components):

```
pydeps pyiomica --reverse --max-bacon=2 --cluster --max-cluster-size=6 --min-cluster-size=2 -T=png -o=docs/pyiomica_pydeps_current.png
pydeps pyiomica --reverse --max-bacon=2 --cluster --max-cluster-size=6 --min-cluster-size=2 -T=svg -o=docs/pyiomica_pydeps_current.svg
```



[View raw graph](#)

INCLUDED DATA

PyIOmica is supplied with a set of examples that require folder `data` described here. This folder is created during installation of PyIOmica, see section Installation, and its location is stored in PyIOmica's global variable `ConstantPyIOmicaDataDirectory`.

Annotation and Enumeration functions such as Enrichment Analyses functions require various dictionaries and information files to be downloaded from the Internet. We provide a set of such dictionaries and files (in PyIOmica's data directory described above) to make it possible for user run PyIOmica in the absence of the Internet connection. However the user can always override these files with updates ones by specifying proper flags to the functions that use the dictionaries.

EXAMPLES

Examples of different uses of PyIOmica package API.

A Jupyter notebook with all examples is available at: <https://github.com/gmiaslab/pyiomica/tree/master/docs/examples>

6.1 Enrichment report

Function `ExportEnrichmentReport` generates enrichment report in form of “.xlsx” file.

A	B	C	D	E	F	G	H	I	J
	p-Value	BH-corrected p-Value	Significant	Counts in list	Counts in family	Total members	Counts in members	Description	List of gene hits
1	GO:0051092	9.89839E-11	5.35503E-08	TRUE	20	151	19774	7 positive regulation of NF-kappaB transcription factor activity, biological_process	TAB1 MALT1 TIRAP CHUK NFKB1 NFKB2 IRAK4
2	GO:0070498	3.37091E-08	9.11831E-06	TRUE	20	95	19774	5 interleukin-5-mediated signaling pathway, biological_process	TAB1 CHUK NFKB1 NFKBIA IRAK4
3	GO:0002223	7.72621E-08	1.19307E-05	TRUE	20	112	19774	5 stimulatory C-type lectin receptor signaling pathway, biological_process	TAB1 MALT1 CHUK LYN NFKB1
4	GO:0033209	8.8212E-08	1.19307E-05	TRUE	20	115	19774	5 tumor necrosis factor-mediated signaling pathway, biological_process	TNFSF13B CHUK TNFRSF13C LTBR NFKBIA
5	GO:0038095	6.36629E-07	5.81544E-05	TRUE	20	171	19774	5 Fc-epsilon receptor signaling pathway, biological_process	TAB1 MALT1 CHUK LYN NFKB1
6	GO:0043123	7.98199E-07	5.81544E-05	TRUE	20	179	19774	5 positive regulation of I-kappaB kinase/NF-kappaB signaling, biological_process	MALT1 TIRAP CHUK LTBR IRAK4
7	GO:0004674	1.18244E-06	5.81544E-05	TRUE	20	365	19774	6 protein serine/threonine kinase activity, molecular_function	CHUK CSNK2A1 CSNK2A2 CSNK2B ATM IRAK4
8	GO:0006915	7.1017E-07	5.81544E-05	TRUE	20	551	19774	7 apoptotic process, biological_process	PARP1 CSNK2A1 CSNK2A2 LTBR GADD45B NFKB1 NFKBIA
9	GO:0001782	1.16322E-06	5.81544E-05	TRUE	20	21	19774	3 B cell homeostasis, biological_process	TNFSF13B TNFRSF13C LYN
10	GO:1905818	9.71958E-07	5.81544E-05	TRUE	20	2	19774	2 regulation of chromosome separation, biological_process	CSNK2A1 CSNK2A2
11	GO:0033256	9.71958E-07	5.81544E-05	TRUE	20	2	19774	2 I-kappaB/NF-kappaB complex, cellular_component	NFKB1 NFKBIA
12	GO:0005956	2.91395E-06	0.00010597	TRUE	20	3	19774	2 protein kinase C2 complex, cellular_component	CSNK2A1 CSNK2B
13	GO:0031296	2.91395E-06	0.00010597	TRUE	20	3	19774	2 B cell costimulation, biological_process	TNFSF13B TNFRSF13C
14	GO:0002636	2.91395E-06	0.00010597	TRUE	20	3	19774	2 positive regulation of germinal center formation, biological_process	TNFSF13B TNFRSF13C
15	GO:0035159	2.54827E-06	0.00010597	TRUE	20	27	19774	3 Pcs protein complex, cellular_component	CSNK2A1 CSNK2A2 CSNK2B
16	GO:0001485	3.53797E-06	0.000139614	TRUE	20	106	19774	4 protein N-terminus binding, molecular_function	PARP1 CSNK2A1 CSNK2A2 ATM
17	GO:0042802	4.458E-06	0.00013482	TRUE	20	1054	19774	8 identical protein binding, molecular_function	MALT1 TIRAP PARP1 CSNK2A1 CSNK2B LTBR NFKB1 NFKBIA
18	GO:0002755	4.7349E-06	0.00013482	TRUE	20	33	19774	3 MyD88-dependent toll-like receptor signaling pathway, biological_process	TAB1 TIRAP IRAK4
19	GO:0006656	4.7349E-06	0.00013482	TRUE	20	33	19774	3 phosphatidylcholine biosynthetic process, biological_process	CSNK2A1 CSNK2A2 CSNK2B
20	GO:0011663	5.18970E-06	0.000140383	TRUE	20	34	19774	3 lipopolysaccharide-mediated signaling pathway, biological_process	MALT1 LYN NFKBIA
21	GO:1901796	8.20930E-06	0.000211489	TRUE	20	131	19774	4 regulation of signal transduction by p53 class mediator, biological_process	CSNK2A1 CSNK2A2 CSNK2B ATM
22	GO:0030890	9.20351E-06	0.000226523	TRUE	20	41	19774	3 positive regulation of B cell proliferation, biological_process	TNFSF13B TIRAP TNFRSF13C
23	GO:0005515	1.23024E-05	0.000289374	TRUE	20	9629	19774	19 protein binding, molecular_function	TAB1 TNFSF13B MALT1 TIRAP CHUK PARP1 CSNK2A1 CSNK2A2 CSNK2B
24	GO:0018105	1.80798E-05	0.000407548	TRUE	20	160	19774	4 peptidyl-serine phosphorylation, biological_process	CHUK CSNK2A1 CSNK2A2 ATM

The are following columns in each Worksheet of the output “.xlsx” file

- ID: Annotation term identifier
- p-Value: Probability to find at least “Counts in members” number of genes when drawing “Counts in list” genes “Counts in family” times (without replacement) from “Counts in members” identifiers. Note: this is the behavior when using default setting, i.e. Hypergeometric distribution testing function
- BH-corrected p-Value: p-Value corrected for false discovery rate (FDR) via Benjamini-Hochberg procedure
- Significant: Whether the “BH-corrected p-Value” is below the threshold (typically 0.05) specified for the enrichment analysis
- Counts in list: Number of genes in the input list
- Counts in family: Number of genes in a particular annotation term
- Total members: Number of members (e.g. UniprotIDs) in the IDs to annotation terms dictionary

- Counts in members: Number of annotation terms in which at least of gene from the input list appear
- Description: Description of a particular annotation term, e.g. details/type
- List of gene hits: Gene identifiers that are found in a particular annotation term. The identifiers are separated by a Vertical bar

6.2 Import of MathIOmica Objects

```
#import sys
#sys.path.append("../..")

import pyiomica as pio

from pyiomica.utilityFunctions import readMathIOmicaData

print(pio.ConstantPyIOmicaExamplesDirectory, '\n')
tempPath = pio.os.path.join(pio.ConstantPyIOmicaExamplesDirectory, 'MathIOmicaExamples')

rnaExample = readMathIOmicaData(pio.os.path.join(tempPath, 'rnaExample'))
print('rnaExample:', str(rnaExample)[:400], '\t. . .\n')

proteinClassificationExample = readMathIOmicaData(pio.os.path.join(tempPath,
↪ 'proteinClassificationExample'))
print('proteinClassificationExample:', str(proteinClassificationExample)[:400], '\t. . .\n'
↪ '\n')

proteinTimeSeriesExample = readMathIOmicaData(pio.os.path.join(tempPath,
↪ 'proteinTimeSeriesExample'))
print('proteinTimeSeriesExample:', str(proteinTimeSeriesExample)[:400], '\t. . .\n')

proteinExample = readMathIOmicaData(pio.os.path.join(tempPath, 'proteinExample'))
print('proteinExample:', str(proteinExample)[:400], '\t. . .\n')

metabolomicsPositiveModeExample = readMathIOmicaData(pio.os.path.join(tempPath,
↪ 'metabolomicsPositiveModeExample'))
print('metabolomicsPositiveModeExample:', str(metabolomicsPositiveModeExample)[:400], '\n'
↪ '\t. . .\n')

metabolomicsNegativeModeExample = readMathIOmicaData(pio.os.path.join(tempPath,
↪ 'metabolomicsNegativeModeExample'))
print('metabolomicsNegativeModeExample:', str(metabolomicsNegativeModeExample)[:400], '\n'
↪ '\t. . .\n')
```


6.3 Clustering object export example

Example of a clustering object exported to “xlsx” file.

The screenshot shows an Excel spreadsheet titled 'ExampleClusteringObject.GroupsSubgroups - Excel'. The spreadsheet contains a table with columns labeled 'source', 'id', and various data points. The data is organized into rows for different gene clusters. The first column is 'source', the second is 'id', and the remaining columns contain data and autocorrelations. The data is organized into rows for different gene clusters.

The file contains one Spreadsheet for each subgroup in a clustering object. The spreadsheet is structured as follows. The first column indicates the data source, the second column has gene identifiers, all the remaining columns contain data and autocorrelations. Note, this is the structure when using the default settings. User can modify output by setting various options.

6.4 GO Analysis examples

```
#import sys
#sys.path.append("../..")

import pyiomica as pio

from pyiomica.enrichmentAnalyses import GOAnalysis, ExportEnrichmentReport
from pyiomica import dataStorage as ds

EnrichmentOutputDirectory = 'results/EnrichmentOutputDirectory/'

#Let's do a GO analysis for a group of genes, annotated with their "Gene Symbol":
goExample1 = GOAnalysis(["TAB1", "TNFSF13B", "MALT1", "TIRAP", "CHUK",
                        "TNFRSF13C", "PARP1", "CSNK2A1", "CSNK2A2", "CSNK2B",
                        "LTBR",
                        "LYN", "MYD88", "GADD45B", "ATM", "NFKB1", "NFKB2",
                        "NFKBIA",
                        "IRAK4", "PIAS4", "PLAU"])

ExportEnrichmentReport(goExample1, AppendString='goExample1',
                        OutputDirectory=EnrichmentOutputDirectory + 'GOAnalysis/')

#The information can be computed for multiple groups, if these are provided as an
association:
analysisGOAssociation = GOAnalysis({"Group1": ["C6orf57", "CD46", "DHX58", "HMGB3",
```

(continues on next page)

(continued from previous page)

```

↪ "MAP3K5", "NFKB2", "NOS2", "PYCARD", "PYDC1", "SSC5D"],
    "Group2": ["TAB1", "TNFSF13B", "MALT1",
↪ "TIRAP", "CHUK", "TNFRSF13C", "PARP1", "CSNK2A1", "CSNK2A2", "CSNK2B", "LTBR",
    "LYN", "MYD88", "GADD45B",
↪ "ATM", "NFKB1", "NFKB2", "NFKBIA", "IRAK4", "PIAS4", "PLAU"]])

ExportEnrichmentReport(analysisGOAssociation, AppendString='analysisGOAssociation',
↪ OutputDirectory=EnrichmentOutputDirectory + 'GOAnalysis/')

#The data can be computed with or without a label. If labeled, the gene ID must be the
↪ first element for each ID provided. The data is in the form {ID,label}:
analysisGOLabel = GOAnalysis([["C6orf57", "Protein"], ["CD46", "Protein"], ["DHX58",
↪ "Protein"], ["HMGB3", "Protein"], ["MAP3K5", "Protein"],
    ["NFKB2", "Protein"], ["NOS2", "Protein"], [
↪ "PYCARD", "Protein"], ["PYDC1", "Protein"], ["SSC5D", "Protein"]])

ExportEnrichmentReport(analysisGOLabel, AppendString='analysisGOLabel',
↪ OutputDirectory=EnrichmentOutputDirectory + 'GOAnalysis/')

#The data can be mixed, e.g. proteins and RNA with different labels:
analysisGOMixed = GOAnalysis([["C6orf57", "Protein"], ["CD46", "Protein"], ["DHX58",
↪ "Protein"], ["HMGB3", "RNA"], ["HMGB3", "Protein"], ["MAP3K5", "Protein"],
    ["NFKB2", "RNA"], ["NFKB2", "Protein"], ["NOS2",
↪ "RNA"], ["PYCARD", "RNA"], ["PYDC1", "Protein"], ["SSC5D", "Protein"]])

ExportEnrichmentReport(analysisGOMixed, AppendString='analysisGOMixed',
↪ OutputDirectory=EnrichmentOutputDirectory + 'GOAnalysis/')

#We can instead treat the data as different by setting the MultipleList and
↪ MultipleListCorrection options:
analysisGOMixedMulti = GOAnalysis([["C6orf57", "Protein"], ["CD46", "Protein"], ["DHX58",
↪ "Protein"], ["HMGB3", "RNA"], ["HMGB3", "Protein"], ["MAP3K5", "Protein"],
    ["NFKB2", "RNA"], ["NFKB2", "Protein"], [
↪ "NOS2", "RNA"], ["PYCARD", "RNA"], ["PYDC1", "Protein"], ["SSC5D", "Protein"]],
    MultipleList=True, MultipleListCorrection=
↪ 'Automatic')

ExportEnrichmentReport(analysisGOMixedMulti, AppendString='analysisGOMixedMulti',
↪ OutputDirectory=EnrichmentOutputDirectory + 'GOAnalysis/')

#Enrichment analysis function GOAnalysis can be used with a clustering object.
#First run examples of use of categorization functions to generate clustering objects.
#Then run "results = GOAnalysis(ds.read(pathToClusteringObjectOfInterest))",
#to calculate enrichment for each group in each class, and then export enrichment
↪ results to a file if necessary.

```

6.5 KEGG Analysis examples

```
#import sys
#sys.path.append("../..")

import pyiomica as pio

from pyiomica.enrichmentAnalyses import KEGGAnalysis, ExportEnrichmentReport
from pyiomica import dataStorage as ds

EnrichmentOutputDirectory = pio.os.path.join('results', 'EnrichmentOutputDirectory', '')

#Let's do a KEGG pathway analysis for a group of genes (most in the NFKB pathway),
↪ annotated with their "Gene Symbol":
keggExample1 = KEGGAnalysis(["TAB1", "TNFSF13B", "MALT1", "TIRAP", "CHUK", "TNFRSF13C",
↪ "PARP1", "CSNK2A1", "CSNK2A2", "CSNK2B", "LTBR", "LYN", "MYD88",
↪ "GADD45B", "ATM", "NFKB1", "NFKB2", "NFKBIA",
↪ "IRAK4", "PIAS4", "PLAU", "POLR3B", "NME1", "CTPS1", "POLR3A"])

ExportEnrichmentReport(keggExample1, AppendString='keggExample1',
↪ OutputDirectory=EnrichmentOutputDirectory + 'KEGGAnalysis/')

#The information can be computed for multiple groups, if these are provided as an
↪ association:
analysisKEGGAssociation = KEGGAnalysis({"Group1": ["C6orf57", "CD46", "DHX58", "HMGB3",
↪ "MAP3K5", "NFKB2", "NOS2", "PYCARD", "PYDC1", "SSC5D"],
↪ "Group2": ["TAB1", "TNFSF13B", "MALT1", "TIRAP", "CHUK", "TNFRSF13C", "PARP1", "CSNK2A1", "CSNK2A2", "CSNK2B", "LTBR",
↪ "LYN", "MYD88", "GADD45B", "ATM", "NFKB1", "NFKB2", "NFKBIA", "IRAK4", "PIAS4", "PLAU", "POLR3B", "NME1", "CTPS1", "POLR3A"]})

ExportEnrichmentReport(analysisKEGGAssociation, AppendString='analysisKEGGAssociation',
↪ OutputDirectory=EnrichmentOutputDirectory + 'KEGGAnalysis/')

#The data can be computed with or without a label. If labeled, the gene ID must be the
↪ first element for each ID provided. The data is in the form {ID,label}:
analysisKEGGLabel = KEGGAnalysis([["C6orf57", "Protein"], ["CD46", "Protein"], ["DHX58",
↪ "Protein"], ["HMGB3", "Protein"], ["MAP3K5", "Protein"],
↪ ["NFKB2", "Protein"], ["NOS2", "Protein"], [
↪ "PYCARD", "Protein"], ["PYDC1", "Protein"], ["SSC5D", "Protein"]])

ExportEnrichmentReport(analysisKEGGLabel, AppendString='analysisKEGGLabel',
↪ OutputDirectory=EnrichmentOutputDirectory + 'KEGGAnalysis/')

#The same result is obtained if IDs are enclosed in list brackets:
analysisKEGGNoLabel = KEGGAnalysis([["C6orf57"], ["CD46"], ["DHX58"], ["HMGB3"], ["MAP3K5",
↪ "NFKB2"], ["NOS2"], ["PYCARD"], ["PYDC1"], ["SSC5D"]])

ExportEnrichmentReport(analysisKEGGNoLabel, AppendString='analysisKEGGNoLabel',
↪ OutputDirectory=EnrichmentOutputDirectory + 'KEGGAnalysis/')

#The same result is obtained if IDs are input as strings:
```

(continues on next page)

(continued from previous page)

```

analysisKEGGstrings = KEGGAnalysis(["C6orf57", "CD46", "DHX58", "HMGB3", "MAP3K5", "NFKB2
↳", "NOS2", "PYCARD", "PYDC1", "SSC5D"])

ExportEnrichmentReport(analysisKEGGstrings, AppendString='analysisKEGGstrings',
↳OutputDirectory=EnrichmentOutputDirectory + 'KEGGAnalysis/')

#The data can be mixed, e.g. proteins and RNA with different labels:
analysisKEGGMixed = KEGGAnalysis(["C6orf57", "Protein"], ["CD46", "Protein"], ["DHX58",
↳"Protein"], ["HMGB3", "RNA"], ["HMGB3", "Protein"], ["MAP3K5", "Protein"],
↳["NFKB2", "RNA"], ["NFKB2", "Protein"], [
↳"NOS2", "RNA"], ["PYCARD", "RNA"], ["PYDC1", "Protein"], ["SSC5D", "Protein"]])

ExportEnrichmentReport(analysisKEGGMixed, AppendString='analysisKEGGMixed',
↳OutputDirectory=EnrichmentOutputDirectory + 'KEGGAnalysis/')

#The data in this case treated as originating from a single population. Protein and RNA
↳labeled data for the same identifier are treated as equivalent.
#We can instead treat the data as different by setting the MultipleList and
↳MultipleListCorrection options:
analysisKEGGMixedMulti = KEGGAnalysis(["C6orf57", "Protein"], ["CD46", "Protein"], [
↳"DHX58", "Protein"], ["HMGB3", "RNA"], ["HMGB3", "Protein"], ["MAP3K5", "Protein"],
↳["NFKB2", "RNA"], ["NFKB2", "Protein"], [
↳"NOS2", "RNA"], ["PYCARD", "RNA"], ["PYDC1", "Protein"], ["SSC5D", "Protein"],
↳MultipleList=True,
↳MultipleListCorrection='Automatic')

ExportEnrichmentReport(analysisKEGGMixedMulti, AppendString='analysisKEGGMixedMulti',
↳OutputDirectory=EnrichmentOutputDirectory + 'KEGGAnalysis/')

#We can carry out a "Molecular" analysis for compound data. We consider the following
↳metabolomics data, which has labels "Meta"
#and additional mass and retention time information in the form {identifier,mass,
↳retention time, label}:
compoundsExample = [{"cpd:C19691", 325.2075, 10.677681, "Meta"}, {"cpd:C17905", 594.2002,
↳8.727458, "Meta"}, {"cpd:C09921", 204.0784, 12.3909445, "Meta"},
↳{"cpd:C18218", 272.2356, 13.473582, "Meta"}, {"cpd:C14169", 235.1573,
↳12.267084, "Meta"}, {"cpd:C14245", 262.2296, 13.545572, "Meta"},
↳{"cpd:C09137", 352.2615, 14.0554285, "Meta"}, {"cpd:C09674", 296.
↳1624, 12.147417, "Meta"}, {"cpd:C00449", 276.1334, 11.004139, "Meta"},
↳{"cpd:C02999", 364.1497, 12.147243, "Meta"}, {"cpd:C07915", 309.194,
↳7.3625283, "Meta"}, {"cpd:C08760", 496.2309, 8.7241125, "Meta"},
↳{"cpd:C14549", 276.0972, 11.078914, "Meta"}, {"cpd:C20533", 601.3378,
↳12.75722, "Meta"}, {"cpd:C20790", 212.1051, 7.127666, "Meta"},
↳{"cpd:C09137", 352.2613, 12.869867, "Meta"}, {"cpd:C17648", 400.2085,
↳10.843841, "Meta"}, {"cpd:C07807", 240.1471, 0.48564285, "Meta"},
↳{"cpd:C08564", 324.0948, 10.281, "Meta"}, {"cpd:C19426", 338.2818,
↳13.758765, "Meta"}, {"cpd:C02943", 468.3218, 14.263261, "Meta"},
↳{"cpd:C04882", 1193.342, 14.707576, "Meta"}]

compoundsExampleKEGG = KEGGAnalysis(compoundsExample, FilterSignificant=True,
↳AnalysisType='Molecular')

```

(continues on next page)

(continued from previous page)

```

ExportEnrichmentReport(compoundsExampleKEGG, AppendString='compoundsExampleKEGG',
↳OutputDirectory=EnrichmentOutputDirectory + 'KEGGAnalysis/')

#We can carry out multiomics data analysis. We consider the following simple example:
multiOmicsData = [{"C6orf57", "Protein"}, {"CD46", "Protein"}, {"DHX58", "Protein"}, [
↳"HMGB3", "RNA"], [{"HMGB3", "Protein"},
↳["MAP3K5", "Protein"], [{"NFKB2", "RNA"}, [{"NFKB2", "Protein"}, [{"NOS2",
↳"RNA"}, [{"PYCARD", "RNA"}, [{"PYDC1", "Protein"},
↳["SSC5D", "Protein"}, [{"cpd:C19691", 325.2075, 10.677681, "Meta"}, [
↳"cpd:C17905", 594.2002, 8.727458, "Meta"],
↳["cpd:C09921", 204.0784, 12.3909445, "Meta"], [{"cpd:C18218", 272.
↳2356, 13.473582, "Meta"},
↳["cpd:C14169", 235.1573, 12.267084, "Meta"], [{"cpd:C14245", 262.2296,
↳13.545572, "Meta"},
↳["cpd:C09137", 352.2615, 14.0554285, "Meta"], [{"cpd:C09674", 296.
↳1624, 12.147417, "Meta"},
↳["cpd:C00449", 276.1334, 11.004139, "Meta"], [{"cpd:C02999", 364.1497,
↳12.147243, "Meta"},
↳["cpd:C07915", 309.194, 7.3625283, "Meta"], [{"cpd:C08760", 496.2309,
↳8.7241125, "Meta"},
↳["cpd:C14549", 276.0972, 11.078914, "Meta"], [{"cpd:C20533", 601.3378,
↳12.75722, "Meta"},
↳["cpd:C20790", 212.1051, 7.127666, "Meta"], [{"cpd:C09137", 352.2613,
↳12.869867, "Meta"},
↳["cpd:C17648", 400.2085, 10.843841, "Meta"], [{"cpd:C07807", 240.1471,
↳0.48564285, "Meta"},
↳["cpd:C08564", 324.0948, 10.281, "Meta"], [{"cpd:C19426", 338.2818,
↳13.758765, "Meta"},
↳["cpd:C02943", 468.3218, 14.263261, "Meta"], [{"cpd:C04882", 1193.342,
↳14.707576, "Meta"}]]

#We can carry out "Genomic" and "Molecular" analysis concurrently by setting
↳AnalysisType = "All":
multiOmicsDataKEGG = KEGGAnalysis(multiOmicsData, AnalysisType='All', MultipleList=True,
↳MultipleListCorrection='Automatic')

ExportEnrichmentReport(multiOmicsDataKEGG, AppendString='multiOmicsDataKEGG',
↳OutputDirectory=EnrichmentOutputDirectory + 'KEGGAnalysis/')

#Enrichment analysis function KEGGAnalysis can be used with a clustering object.
#First run examples of use of categorization functions to generate clustering objects.
#Then run "results = KEGGAnalysis(ds.read(pathToClusteringObjectOfInterest))",
#to calculate enrichment for each group in each class, and then export enrichment
↳results to a file if necessary.

```

6.6 Reactome Analysis examples

```
#import sys
#sys.path.append("../..")

import pyiomica as pio
from pyiomica.enrichmentAnalyses import ReactomeAnalysis, ExportReactomeEnrichmentReport
from pyiomica import dataStorage as ds

EnrichmentOutputDirectory = pio.os.path.join('results', 'EnrichmentOutputDirectory', '')

#Let's do a Reactome analysis for a group of genes, annotated with their "Gene Symbol":
ReactomeExample1 = ReactomeAnalysis(["TAB1", "TNFSF13B", "MALT1", "TIRAP", "CHUK",
                                     "TNFRSF13C", "PARP1", "CSNK2A1", "CSNK2A2", "CSNK2B", "LTBR",
                                     "LYN", "MYD88", "GADD45B", "ATM", "NFKB1", "NFKB2", "NFKBIA",
                                     "IRAK4", "PIAS4", "PLAU"])

ExportReactomeEnrichmentReport(ReactomeExample1,
                               AppendString='ReactomeExample1',
                               OutputDirectory=EnrichmentOutputDirectory +
                               ↪ 'ReactomeAnalysis/')

#The information can be computed for multiple groups, if these are provided as an
↪ association:
analysisReactomeAssociation = ReactomeAnalysis({"Group1": ["C6orf57", "CD46", "DHX58",
↪ "HMGB3", "MAP3K5", "NFKB2", "NOS2", "PYCARD", "PYDC1", "SSC5D"],
                                               "Group2": ["TAB1", "TNFSF13B", "MALT1", "TIRAP",
↪ "CHUK", "TNFRSF13C", "PARP1", "CSNK2A1", "CSNK2A2", "CSNK2B", "LTBR", "LYN", "MYD88",
↪ "GADD45B", "ATM", "NFKB1", "NFKB2", "NFKBIA", "IRAK4", "PIAS4", "PLAU"]})

ExportReactomeEnrichmentReport(analysisReactomeAssociation,
                               AppendString='analysisReactomeAssociation',
                               OutputDirectory=EnrichmentOutputDirectory +
                               ↪ 'ReactomeAnalysis/')

#Enrichment analysis function ReactomeAnalysis can be used with a clustering object.
#First run examples of use of categorization functions to generate clustering objects.
#Then run "results = ReactomeAnalysis(ds.read(pathToClusteringObjectOfInterest))",
#to calculate enrichment for each group in each class, and then export enrichment.
↪ results to a file if necessary using ExportReactomeEnrichmentReport function.
```

6.7 Visibility Graph examples

We represent each timepoint in a series as a node. Temporal events are detected and indicated with solid blue lines encompassing groups of points, or communities. The shortest path identifies nodes (i.e. timepoints) that display high intensity, and thus dominate the global signal profile, are robust to noise, and are likely drivers of the global temporal behavior.

Representing the intensities as bars, this is equivalent to connecting the top of each bar to another top if there is a direct line-of-sight to that top. The resulting visibility graph has characteristics that reflect the equivalent time series temporal structure and can be used to identify trends.

```

#import sys
#sys.path.append("../..")

import pyiomica as pio

from pyiomica.visualizationFunctions import makeVisibilityGraph, makeVisibilityBarGraph
from pyiomica import dataStorage as ds
from pyiomica.extendedDataFrame import DataFrame

# Make an example of time series
# Set intensities of all signals and the measurement time points (here positions)
positions = pio.np.array(range(24))
intensities = pio.np.cos(pio.np.array(range(24))*0.25 + 1.0)**2 + 0.5*(pio.np.random.
↳random(24) - 0.5)
intensities[intensities < 0.0] = 0.

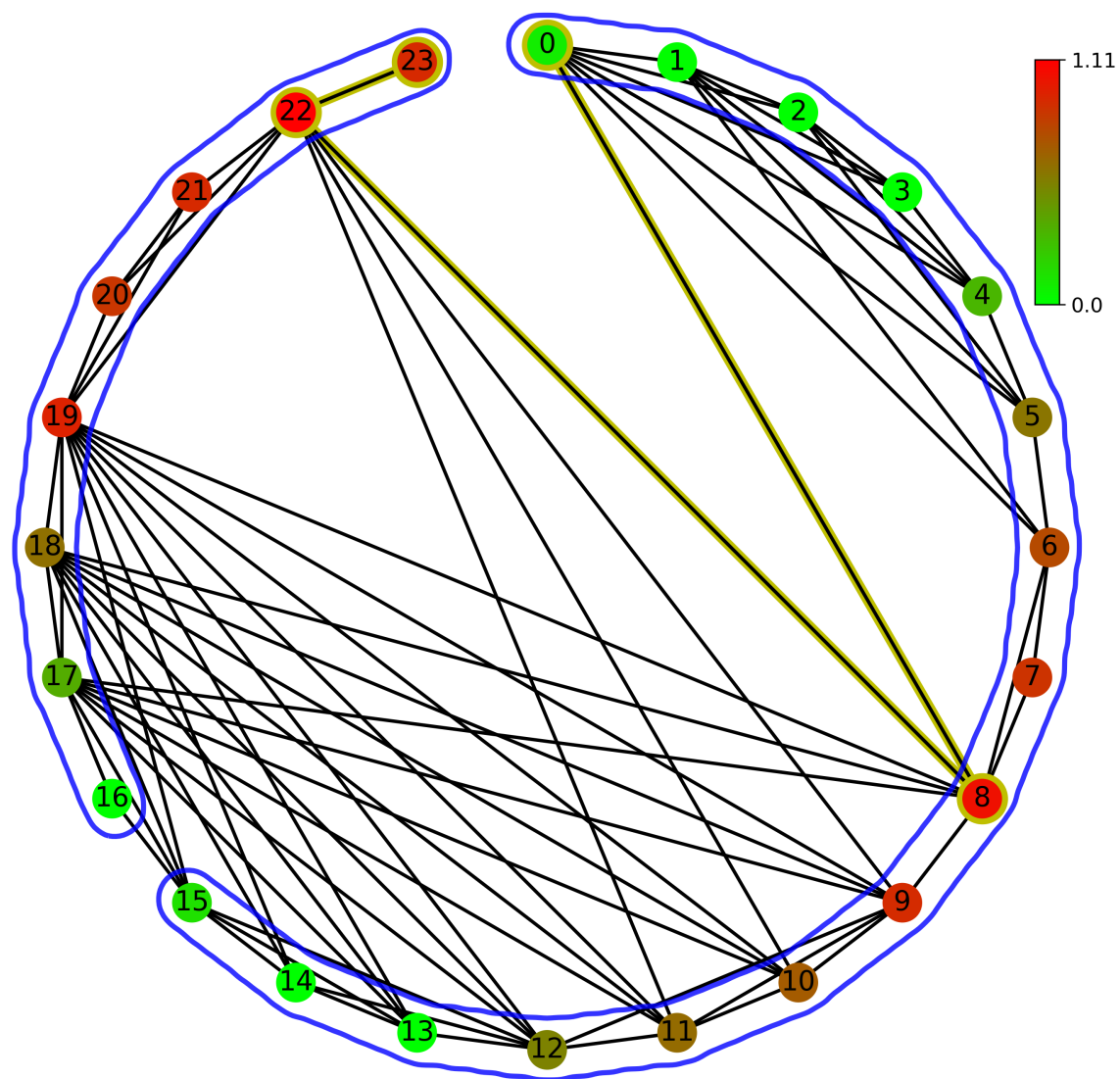
# Make normal visibility graphs on a 'circle' and 'line' layouts
makeVisibilityGraph(intensities, positions, 'results', 'circle_VG', layout='circle')
makeVisibilityGraph(intensities, positions, 'results', 'line_VG', layout='line')

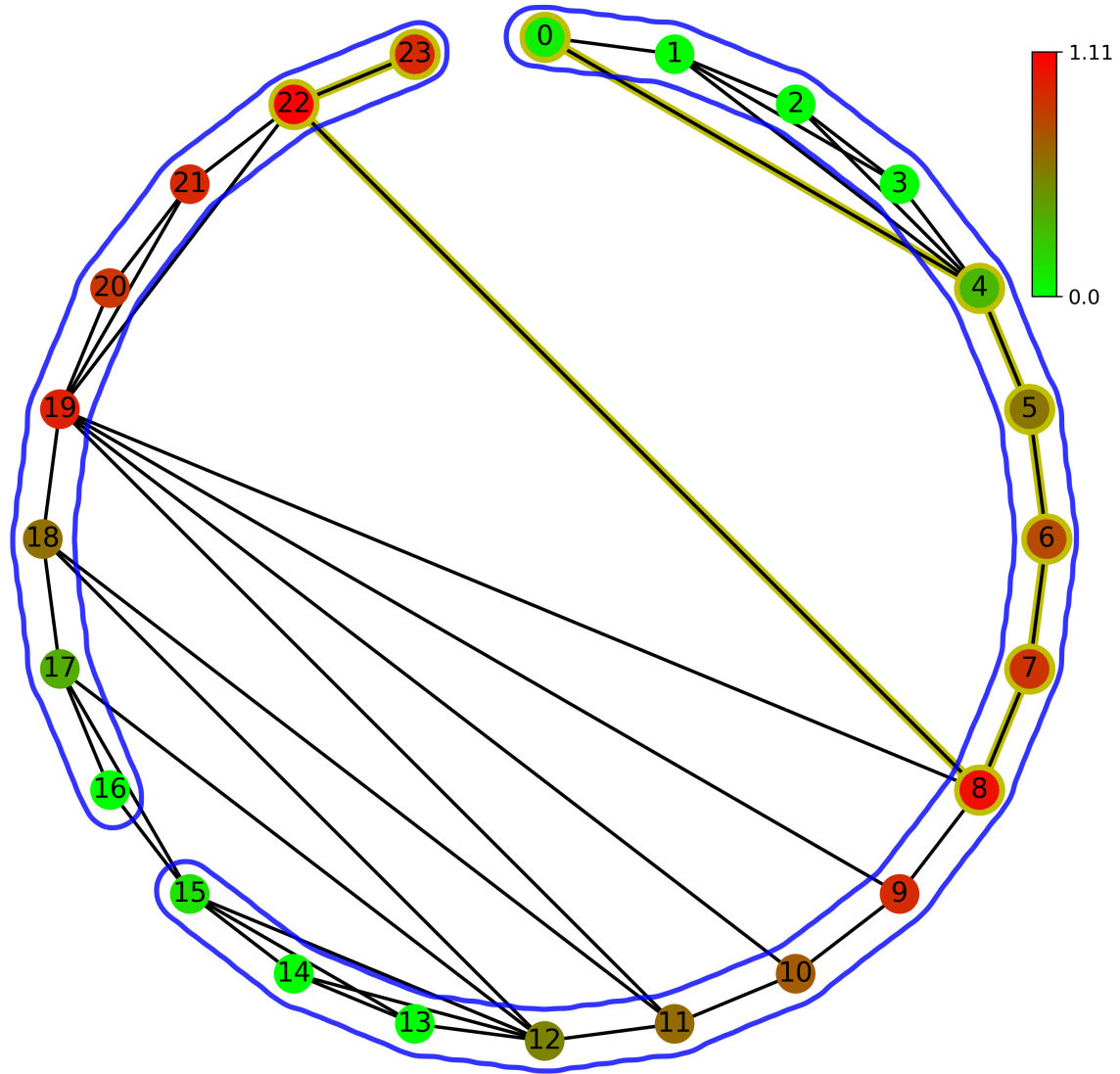
# Make horizontal visibility graphs on a 'circle' and 'line' layouts
makeVisibilityGraph(intensities, positions, 'results', 'circle_VG', layout='circle',
↳horizontal=True)
makeVisibilityGraph(intensities, positions, 'results', 'line_VG', layout='line',
↳horizontal=True)

# Make horizontal and normal bar-style visibility graphs
makeVisibilityBarGraph(intensities, positions, 'results', 'barVG')
makeVisibilityBarGraph(intensities, positions, 'results', 'barVG', horizontal=True)

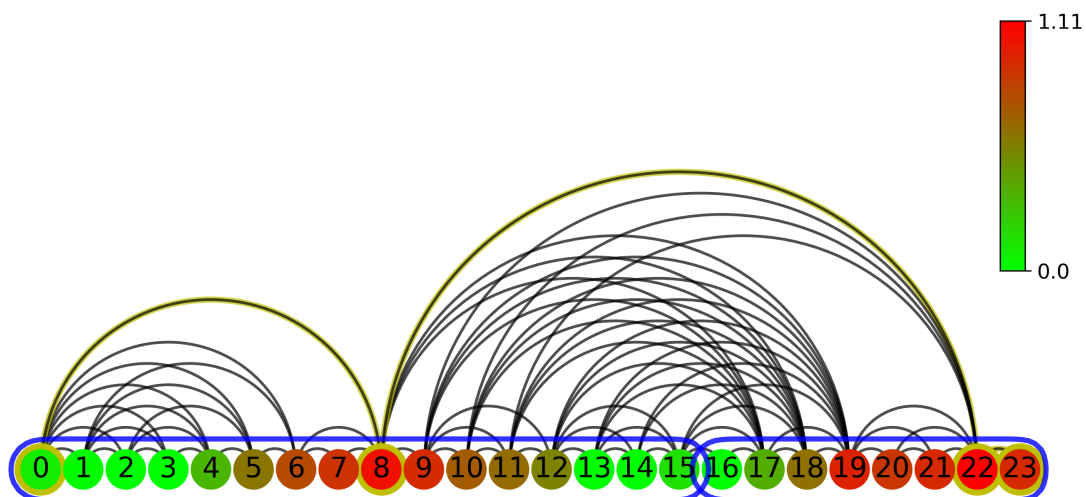
```

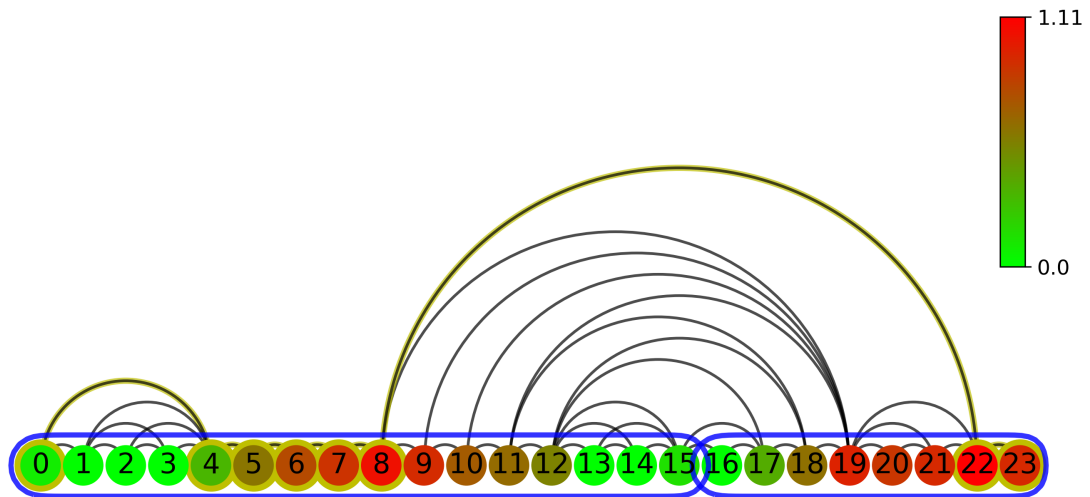
Normal (left) and Horizontal (right) visibility graph on a circular layout:



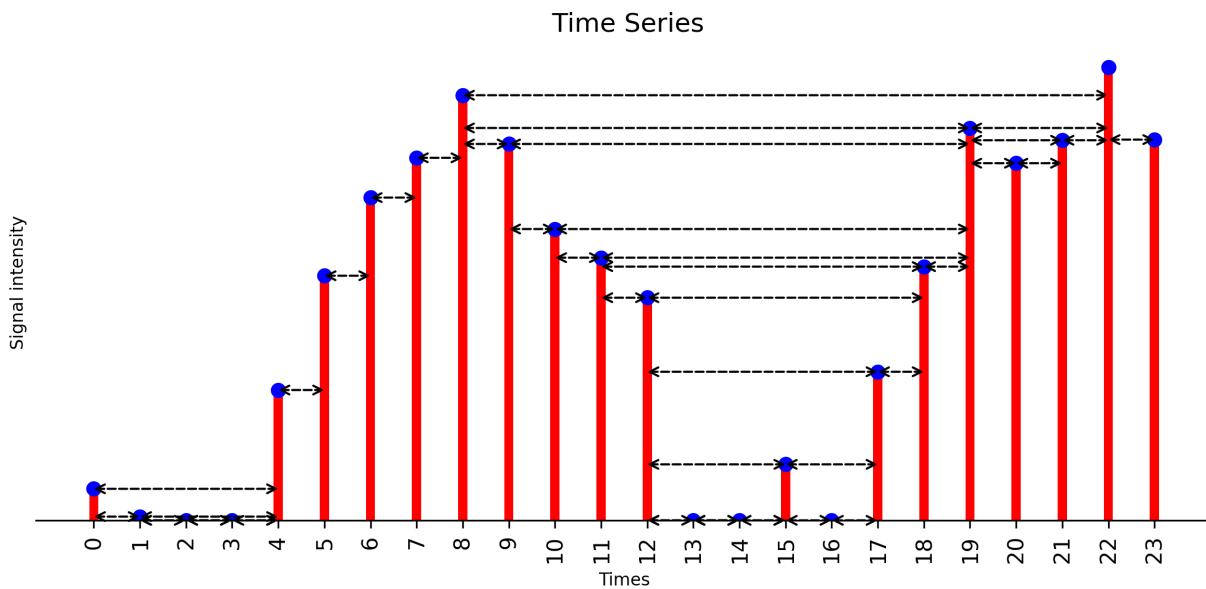
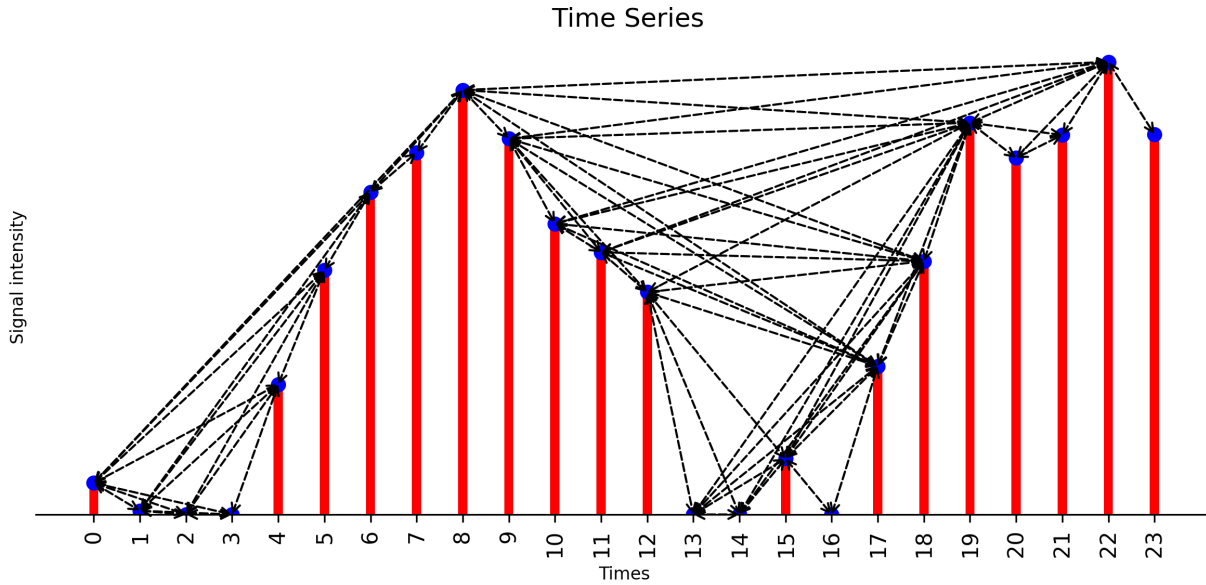


Normal (left) and Horizontal (right) visibility graph on a linear layout:





Normal (left) and Horizontal (right) bar-style visibility graph:



6.8 Visibility Graph Community Detection

```
import numpy as np
import networkx as nx

import pyiomica as pio
from pyiomica import visualizationFunctions
from pyiomica import visibilityGraphCommunityDetection
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```

### create time series
np.random.seed(11)
times = np.arange( 0, 2*np.pi, 0.35)
tp = list(range(len(times)))
data = 5*np.cos(times) + 2*np.random.random(len(times))

### plot time series
fig, ax = plt.subplots(figsize=(8,3))
ax.plot(tp,data)
ax.set_title('Time Series', fontdict={'color': 'k'},fontsize=20)
ax.set_xlabel('Times', fontsize=20)
ax.set_ylabel('Signal intensity', fontsize=20)
ax.set_xticks(tp)
ax.set_xticklabels([str(item) for item in np.round(tp,2)],fontsize=20, rotation=0)
ax.set_yticks([])
fig.tight_layout()
filename = './Time_series_graph.png'
fig.savefig(filename, dpi=300)
plt.close(fig)

### plot weighted natural visibility graph, weight is Euclidean distance
g_nx_NVG, A_NVG = visibilityGraphCommunityDetection.createVisibilityGraph(data, tp,
    ↪ "natural", weight = 'distance')
filename = './Natural_Visibility_Graph.png'
visualizationFunctions.plotNVGBarGraphDual(A_NVG, data, tp, fileName = filename,
    ↪ title = 'Natural Visibility Graph',
    ↪ fontsize=20, figsize=(8,3), dpi=300)

### plot reflected perspective weighted natural visibility graph, weight is Euclidean
    ↪ distance
g_nx_revNVG, A_revNVG = visibilityGraphCommunityDetection.createVisibilityGraph(-data,
    ↪ tp, "natural", weight = 'distance')
filename = './Reflected_perspective_Natural_Visibility_Graph.png'
visualizationFunctions.plotNVGBarGraphDual(A_revNVG, -data, tp, fileName = filename,
    ↪ title='Reflected perspective Natural
    ↪ Visibility Graph', fontsize=20, figsize=(8,3), dpi=300)

### plot dual perspective natural visibility graph, weight is Euclidean distance
g_nx_dualNVG, A_dualNVG = visibilityGraphCommunityDetection.createVisibilityGraph(data,
    ↪ tp, "dual_natural",
    ↪ weight=
    ↪ 'distance', withsign=True)
filename = './Dual_perspective_Natural_Visibility_Graph.png'
visualizationFunctions.plotNVGBarGraphDual(A_dualNVG, data, tp, fileName=filename,
    ↪ title='Dual perspective Natural Visibility
    ↪ Graph', fontsize=20, figsize=(10,4), dpi=300)

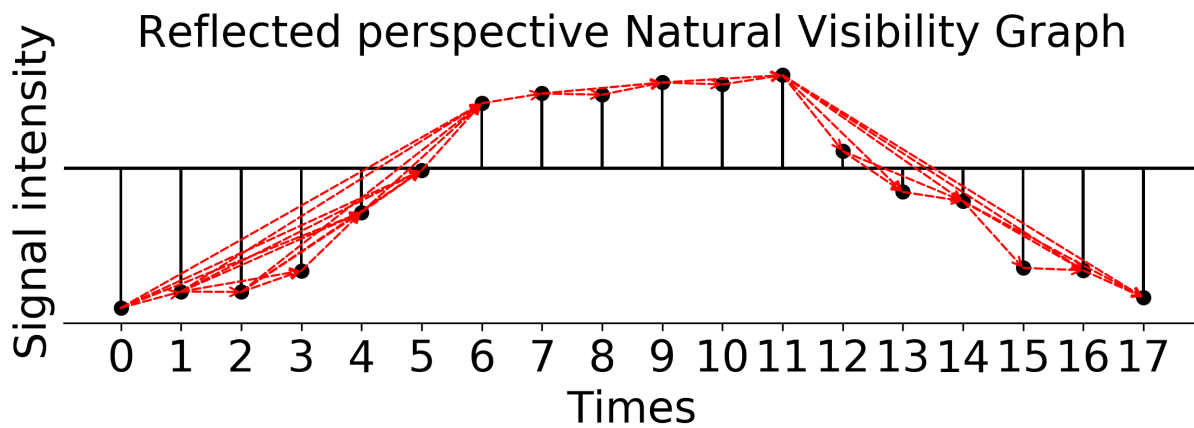
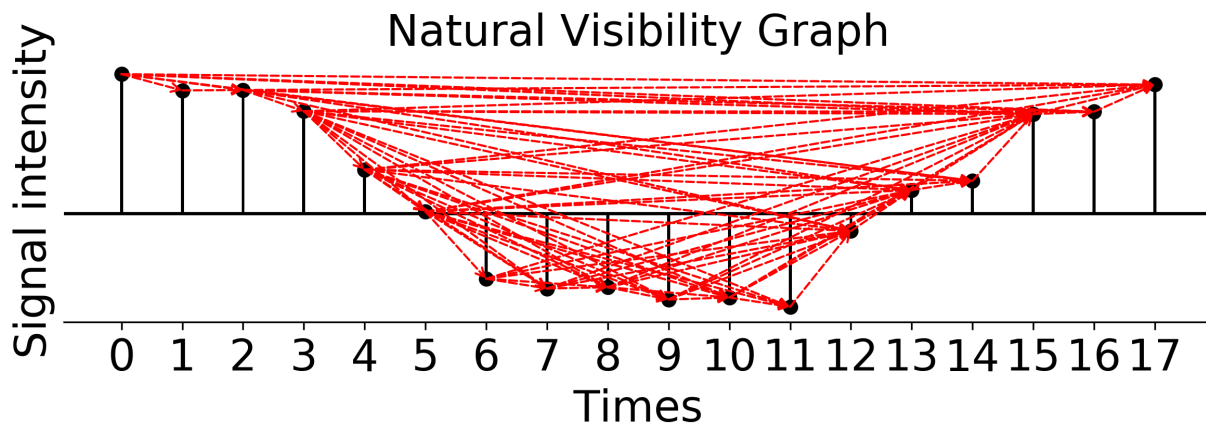
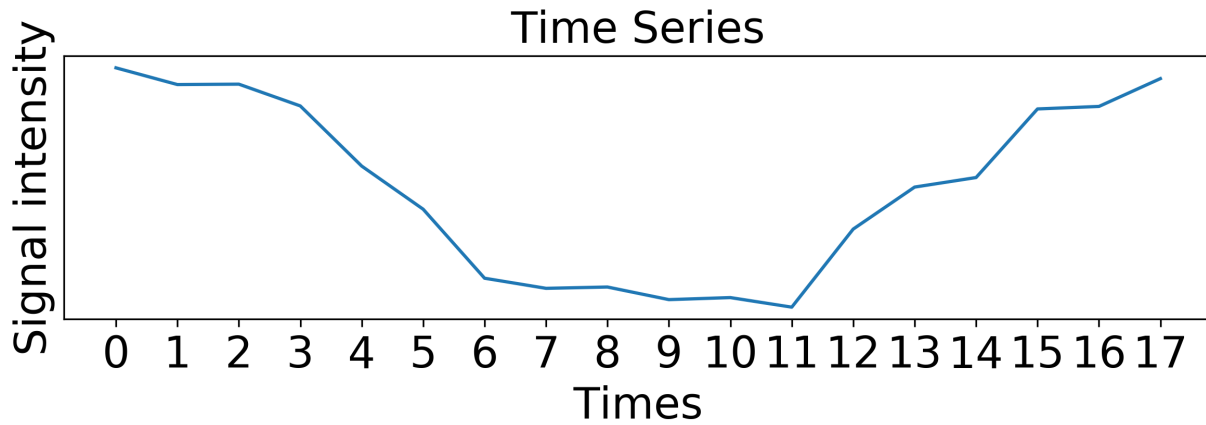
### plot line layout dual perspective natural visibility graph with community structure,
    ↪ weight is Euclidean distance
communities = visibilityGraphCommunityDetection.communityDetectByPathLength(g_nx_dualNVG,
    ↪ direction=None, cutoff='auto')
com = (communities, g_nx_dualNVG)

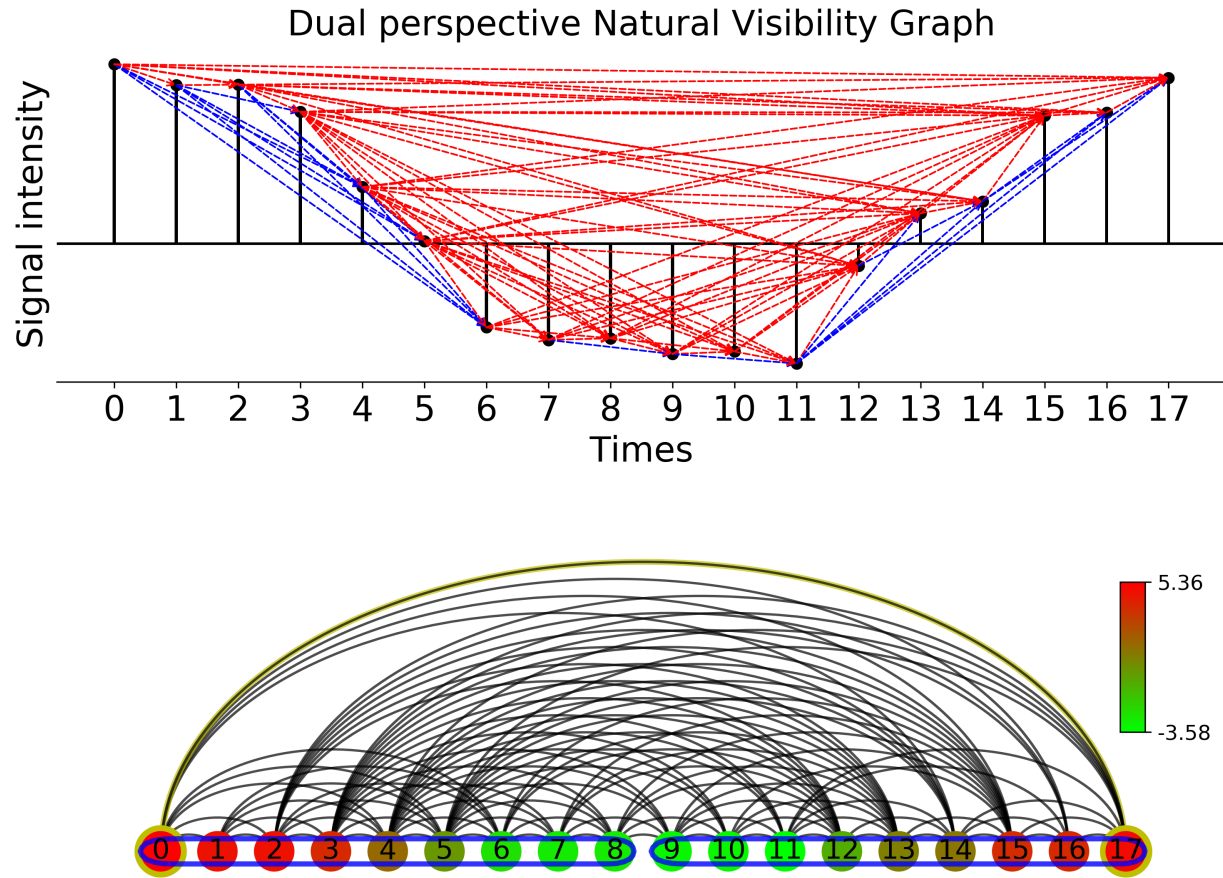
```

(continues on next page)

(continued from previous page)

```
visualizationFunctions.makeVisibilityGraph(data, tp, './', 'Communities_of_Visibility_
↳ Graph', layout='line', communities=com,
    level=0.8, figsize=(10,6), extension='png', dpi=300)
```





6.9 Extended DataFrame

Usage of some of the functions added to a standard DataFrame.

```
#import sys
#sys.path.append("../..")

# Import PyIOMica
import pyiomica as pio

# Import Extended DataFrame from PyIOMica
```

(continues on next page)

(continued from previous page)

```

from pyiomica.extendedDataFrame import DataFrame

# Create a simple data for testing and demonstration
df_data = DataFrame(data=pio.numpy.array([[0.5,2,3],
                                           [0,2,6],
                                           [7,3,0],
                                           [2,2,8],
                                           [1,pio.numpy.nan,pio.numpy.nan],
                                           [6,0,0],
                                           [0,0,0],
                                           [3,3,3.1],
                                           [3,2,pio.numpy.nan],
                                           [4,pio.numpy.nan,4]]).astype(float),
                    index=['s1','s2','s3','s4','s5','s6','s7','s8','s9','s10'],
                    columns=['c1', 'c2', 'c3'])
print(df_data, '\n')

# Remove all-zero signals from the data
df_data.filterOutAllZeroSignals(inplace=True)
print(df_data, '\n')

# Remove first-point-zero signals from the data
df_data.filterOutReferencePointZeroSignals(inplace=True)
print(df_data, '\n')

# Remove nearly-constant signals from the data
df_data.removeConstantSignals(0.2, inplace=True)
print(df_data, '\n')

# Remove signals with >75% non-zero points
df_data.filterOutFractionZeroSignals(0.6, inplace=True)
print(df_data, '\n')

# Remove signals with >75% non-zero points
df_data.filterOutFractionMissingSignals(0.8, inplace=True)
print(df_data, '\n')

# Add a signal with zeros
df_data.loc['s11'] = [2,0,6]
print(df_data, '\n')

# Replace any zeros with np.NaN (missing)
df_data.tagValueAsMissing(inplace=True)
print(df_data, '\n')

# Replace any missing values (np.NaN) with values
df_data.tagMissingAsValue(value=0, inplace=True)
print(df_data, '\n')

# Replace any values smaller than 'a' with 'b'
df_data.tagLowValues(1., 1., inplace=True)
print(df_data, '\n')

```

(continues on next page)

(continued from previous page)

```

# Calculate modified zscore (median-based) of data
df_data_zm = df_data.modifiedZScore()
print(df_data_zm, '\n')

# Quantile normalize the data
df_data_qn = df_data.quantileNormalize()
print(df_data_qn, '\n')

# Box-cox transform data
df_data_bc = df_data.boxCoxTransform()
print(df_data_bc, '\n')

# Normalize signals to unity
df_data_un = df_data.normalizeSignalsToUnity()
print(df_data_un, '\n')

```

6.10 Time Series Categorization

```

#import sys
#sys.path.append("../..")

import pyiomica as pio

from pyiomica import categorizationFunctions as cf

if __name__ == '__main__':

    # Unzip example data
    with pio.zipfile.ZipFile(pio.os.path.join(pio.ConstantPyIOmicaExamplesDirectory,
↪ 'SLV.zip'), "r") as zipFile:
        zipFile.extractall(path=pio.ConstantPyIOmicaExamplesDirectory)

    # Process sample dataset SLV_Hourly1
    # Name of the first data set
    dataName = 'SLV_Hourly1TimeSeries'

    # Define a directory name where results are be saved
    saveDir = pio.os.path.join('results', dataName, '')

    # Directory name where example data is (*.csv files)
    dataDir = pio.os.path.join(pio.ConstantPyIOmicaExamplesDirectory, 'SLV')

    # Read the example data into a DataFrame
    df_data = pio.pd.read_csv(pio.os.path.join(dataDir, dataName + '.csv'), index_col=[0,
↪ 1,2], header=0)

    # Calculate time series categorization
    cf.calculateTimeSeriesCategorization(df_data, dataName, saveDir, ↪
↪ NumberOfRandomSamples = 10**5)

```

(continues on next page)

(continued from previous page)

```

# Cluster the time series categorization results
cf.clusterTimeSeriesCategorization(dataName, saveDir)

# Make plots of the clustered time series categorization
cf.visualizeTimeSeriesCategorization(dataName, saveDir)

# Process sample dataset SLV_Hourly2, in the same way as SLV_Hourly1 above
dataName = 'SLV_Hourly2TimeSeries'
saveDir = pio.os.path.join('results', dataName, '')
dataDir = pio.os.path.join(pio.ConstantPyIomicaExamplesDirectory, 'SLV')
df_data = pio.pd.read_csv(pio.os.path.join(dataDir, dataName + '.csv'), index_col=[0,
↪1,2], header=0)
cf.calculateTimeSeriesCategorization(df_data, dataName, saveDir, ↪
↪NumberOfRandomSamples = 10**5)
cf.clusterTimeSeriesCategorization(dataName, saveDir)
cf.visualizeTimeSeriesCategorization(dataName, saveDir)

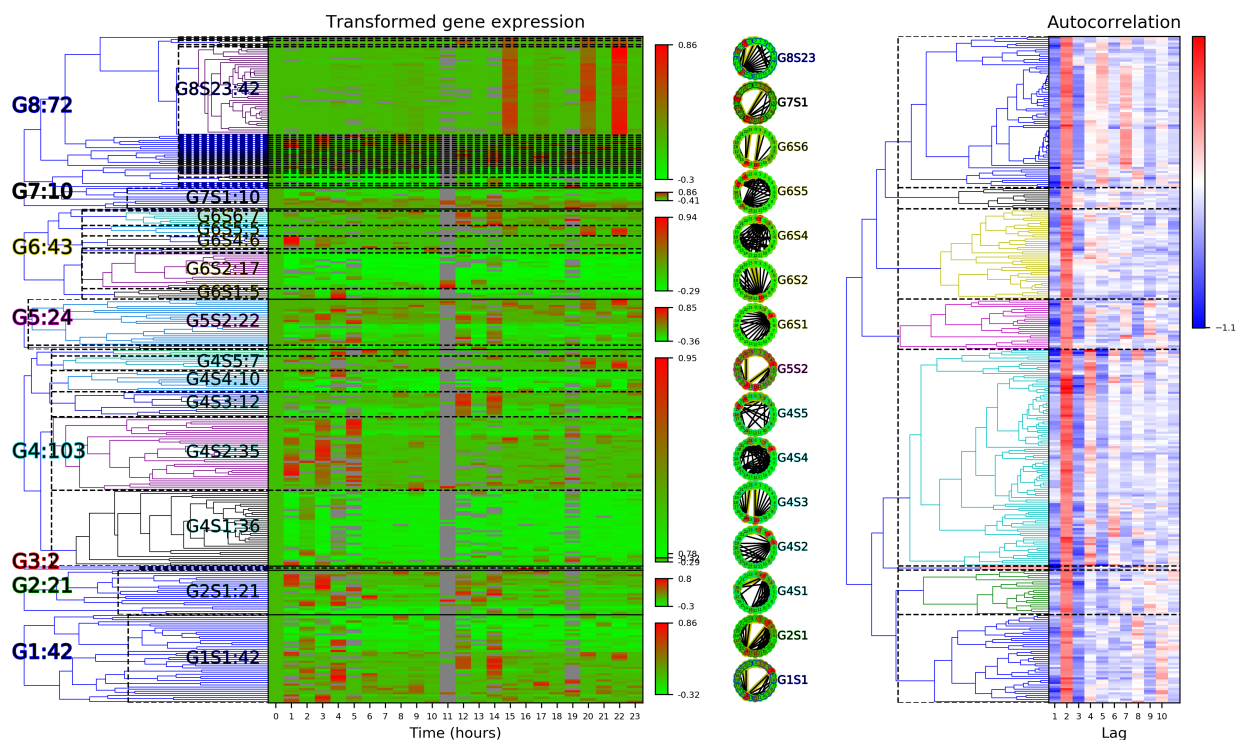
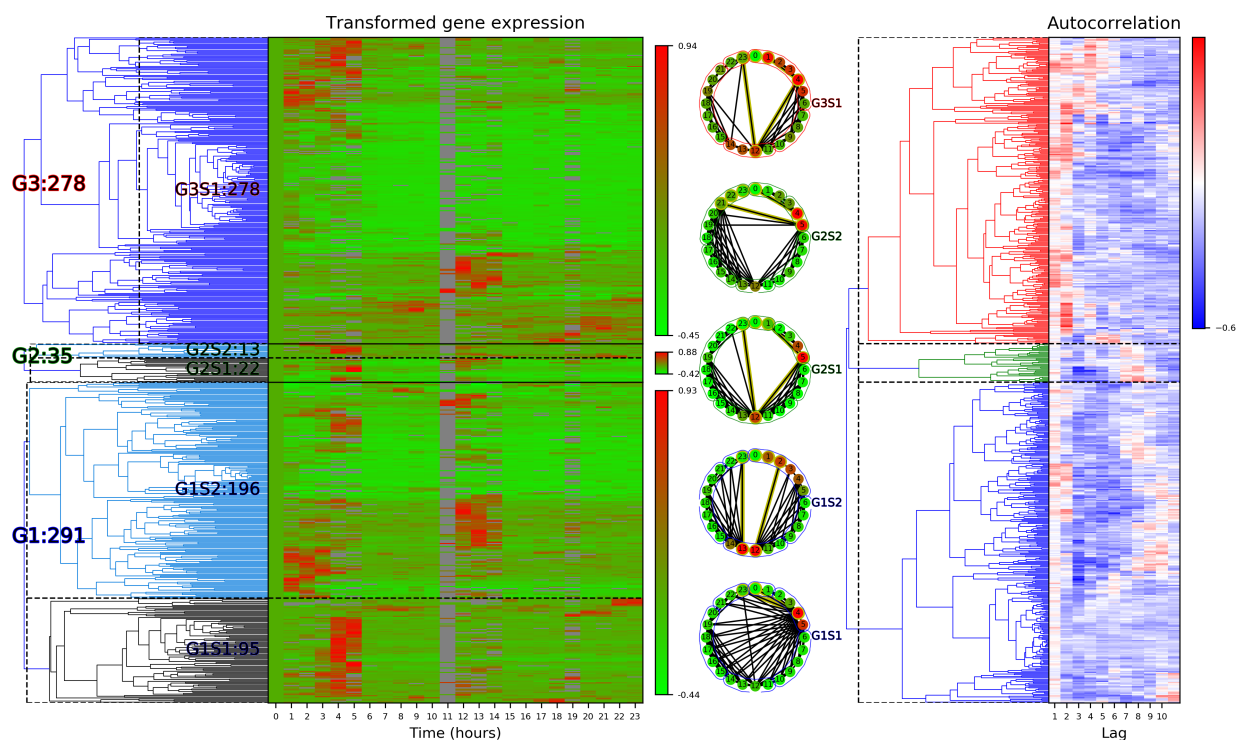
# Import data storage submodule to read results of processing sample datasets SLV_
↪Hourly1 and SLV_Hourly2
from pyiomica import dataStorage as ds

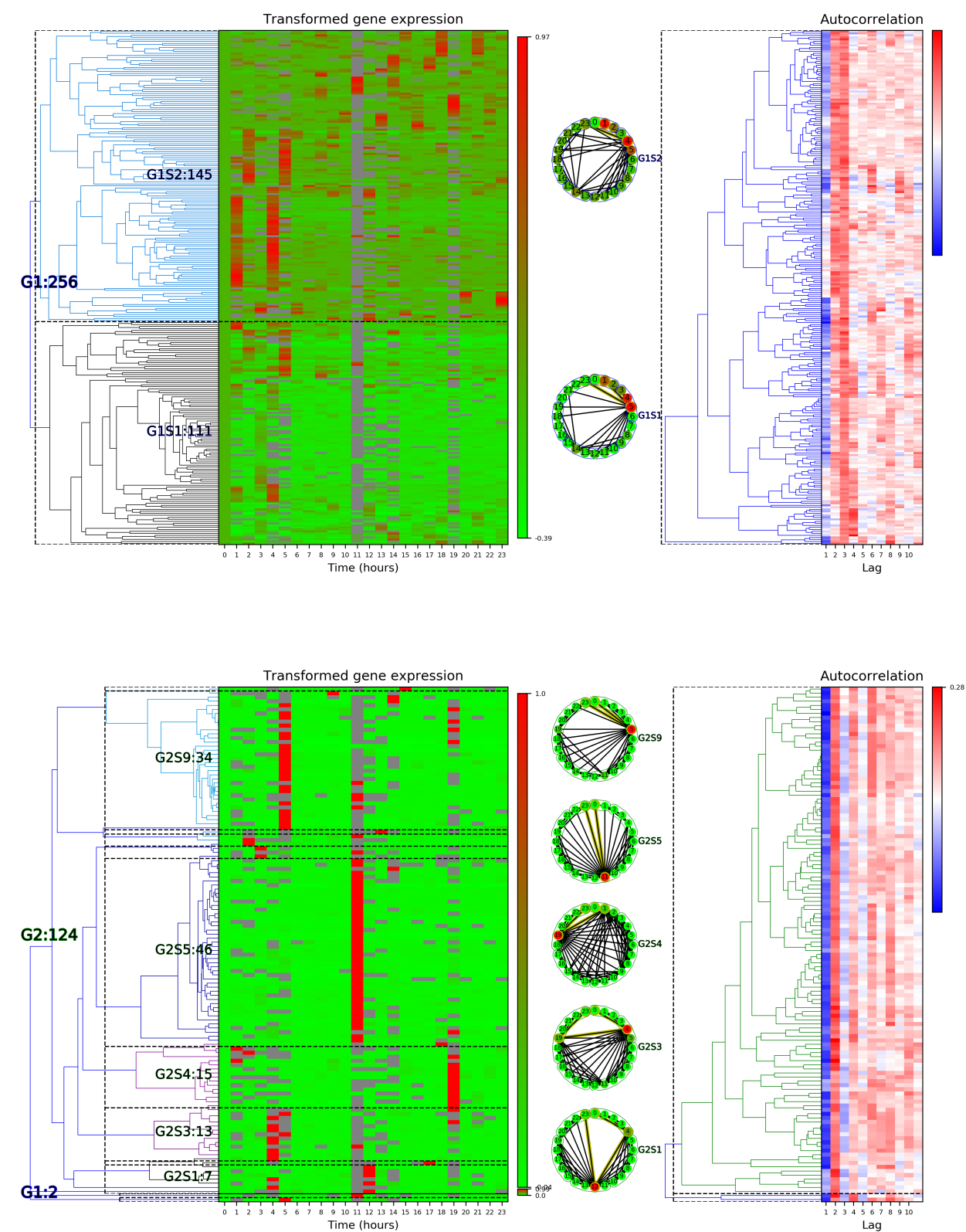
# Use results from processing sample datasets SLV_Hourly1 and SLV_Hourly2 to ↪
↪calculate "Delta"
dataName = 'SLV_Hourly1TimeSeries'
df_data_processed_H1 = ds.read(dataName+'_df_data_transformed', hdf5fileName=pio.os.
↪path.join('results', dataName, dataName+'.h5'))

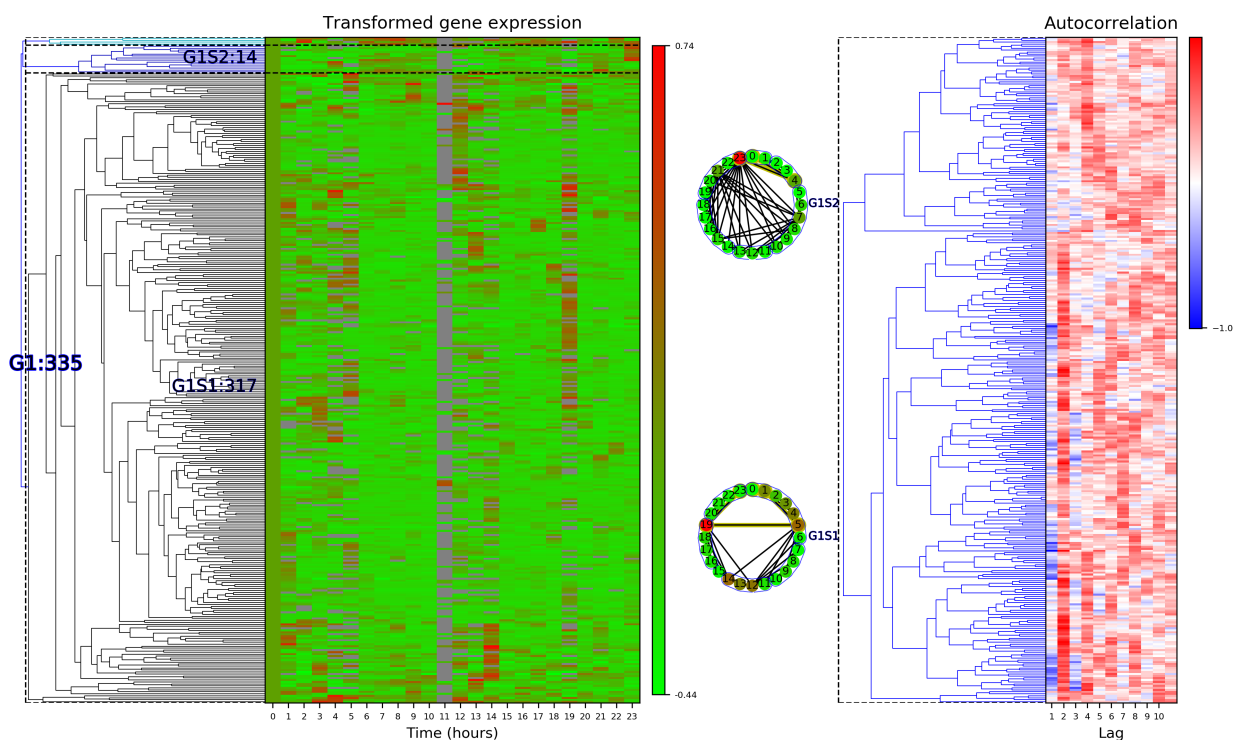
dataName = 'SLV_Hourly2TimeSeries'
df_data_processed_H2 = ds.read(dataName+'_df_data_transformed', hdf5fileName=pio.os.
↪path.join('results', dataName, dataName+'.h5'))

dataName = 'SLV_Delta'
saveDir = pio.os.path.join('results', dataName, '')
df_data = df_data_processed_H2.compareTwoTimeSeries(df_data_processed_H1, ↪
↪compareAllLevelsInIndex=False, mergeFunction=pio.np.median).fillna(0.)
cf.calculateTimeSeriesCategorization(df_data, dataName, saveDir, ↪
↪NumberOfRandomSamples = 10**5)
cf.clusterTimeSeriesCategorization(dataName, saveDir)
cf.visualizeTimeSeriesCategorization(dataName, saveDir)

```







```
#import sys
#sys.path.append("../..")

import pyiomica as pio

from pyiomica import categorizationFunctions as cf
from pyiomica import dataStorage as ds
from pyiomica import enrichmentAnalyses as ea

if __name__ == '__main__':

    # Unzip example data
    with pio.zipfile.ZipFile(pio.os.path.join(pio.ConstantPyIOmicaExamplesDirectory,
    ↪ 'SLV.zip'), "r") as zipFile:
        zipFile.extractall(path=pio.ConstantPyIOmicaExamplesDirectory)

    # Name of the first data set
    dataName = 'DailyTimeSeries_SLV_Protein'

    # Define a directory name where results are to be saved
    saveDir = pio.os.path.join('results', dataName, '')

    # Directory name where example data is (*.csv files)
    dataDir = pio.os.path.join(pio.ConstantPyIOmicaExamplesDirectory, 'SLV')

    # Read the example data into a DataFrame
```

(continues on next page)

(continued from previous page)

```

df_data = pio.pd.read_csv(pio.os.path.join(dataDir, dataName + '.csv'), index_col=[0,
↳ 1], header=0)

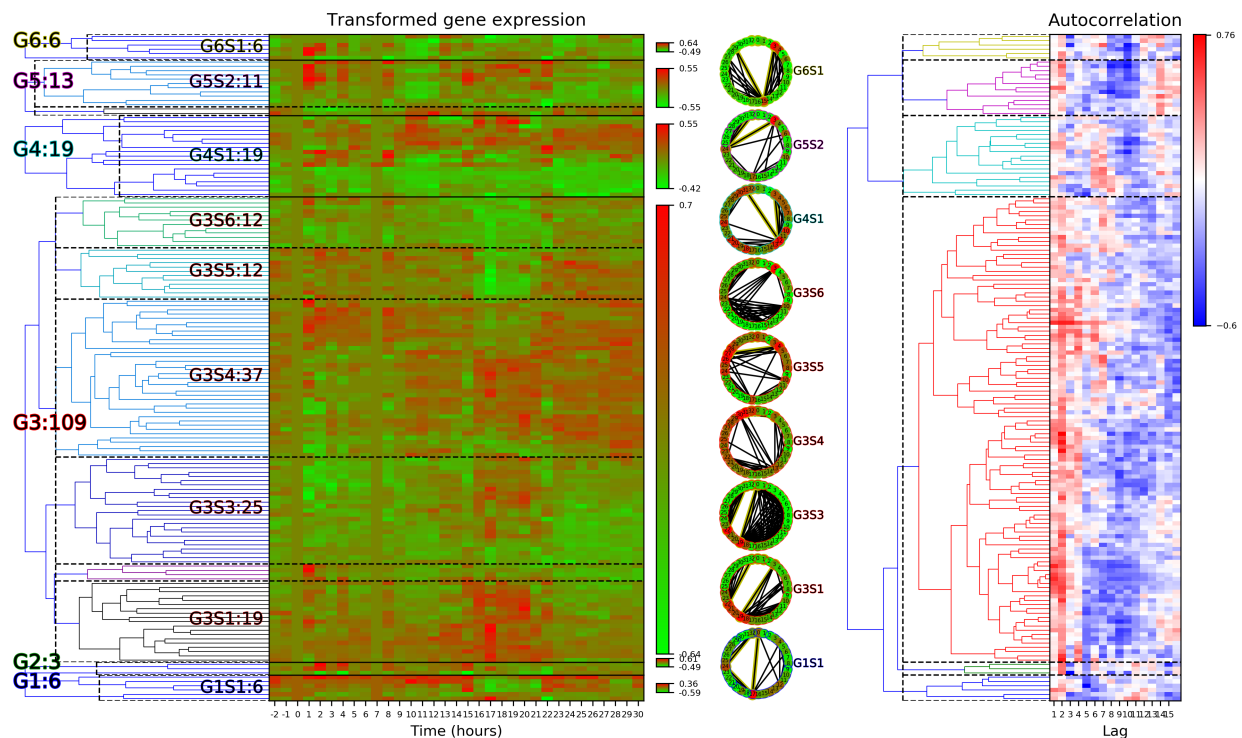
# Calculate time series categorization
cf.calculateTimeSeriesCategorization(df_data, dataName, saveDir,
↳ NumberOfRandomSamples = 10**5, referencePoint=2, preProcessData=False)

# Cluster the time series categorization results
cf.clusterTimeSeriesCategorization(dataName, saveDir)

# Make plots of the clustered time series categorization
cf.visualizeTimeSeriesCategorization(dataName, saveDir)

# Do enrichment GO and KEGG analysis on LAG1 results
LAG1 = ds.read('results/DailyTimeSeries_SLV_Protein/consolidatedGroupsSubgroups/
↳ DailyTimeSeries_SLV_Protein_LAG1_Autocorrelations_GroupsSubgroups')
ea.ExportEnrichmentReport(ea.GOAnalysis(LAG1), AppendString='GO_LAG1',
↳ OutputDirectory='results/DailyTimeSeries_SLV_Protein/')
ea.ExportEnrichmentReport(ea.KEGGAnalysis(LAG1), AppendString='KEGG_LAG1',
↳ OutputDirectory='results/DailyTimeSeries_SLV_Protein/')

```



INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

p

- `pyiomica.categorizationFunctions`, [11](#)
- `pyiomica.clusteringFunctions`, [34](#)
- `pyiomica.dataStorage`, [56](#)
- `pyiomica.enrichmentAnalyses`, [14](#)
- `pyiomica.extendedDataFrame`, [25](#)
- `pyiomica.frequencySubjectMatch`, [38](#)
- `pyiomica.globalVariables`, [9](#)
- `pyiomica.utilityFunctions`, [55](#)
- `pyiomica.visibilityGraphAuxiliaryFunctions`,
[41](#)
- `pyiomica.visibilityGraphCommunityDetection`,
[43](#)
- `pyiomica.visualizationFunctions`, [44](#)

Symbols

`__init__()` (*DataFrame method*), 26

A

`addColorbarToFigure()` (in module *pyiomica.visualizationFunctions*), 52

`addVisibilityGraph()` (in module *pyiomica.visualizationFunctions*), 47

B

`BenjaminiHochbergFDR()` (in module *pyiomica.enrichmentAnalyses*), 24

`bootstrapGeneral()` (in module *pyiomica.frequencySubjectMatch*), 38

`boxCoxTransform()` (*DataFrame method*), 29

C

`calculateLinksBetweenSubjectsByCorrelation()` (in module *pyiomica.frequencySubjectMatch*), 39

`calculateLinksBetweenSubjectsByDistance()` (in module *pyiomica.frequencySubjectMatch*), 38

`calculateTimeSeriesCategorization()` (in module *pyiomica.categorizationFunctions*), 11

`clusterTimeSeriesCategorization()` (in module *pyiomica.categorizationFunctions*), 12

`communityDetectByPathLength()` (in module *pyiomica.visibilityGraphCommunityDetection*), 43

`compareTimeSeriesToPoint()` (*DataFrame method*), 31

`compareTwoTimeSeries()` (*DataFrame method*), 31

`ConstantGeneDictionary` (in module *pyiomica.globalVariables*), 9

`ConstantPyIOMicaDataDirectory` (in module *pyiomica.globalVariables*), 9

`ConstantPyIOMicaExamplesDirectory` (in module *pyiomica.globalVariables*), 9

`ConstantPyIOMicaExampleVideosDirectory` (in module *pyiomica.globalVariables*), 9

`createDirectories()` (in module *pyiomica.utilityFunctions*), 56

`createReverseDictionary()` (in module *pyiomica.utilityFunctions*), 55

`createVisibilityGraph()` (in module *pyiomica.visibilityGraphCommunityDetection*), 43

D

`DataFrame` (class in *pyiomica.extendedDataFrame*), 26

E

`exportClusteringObject()` (in module *pyiomica.clusteringFunctions*), 36

`ExportEnrichmentReport()` (in module *pyiomica.enrichmentAnalyses*), 24

`ExportReactomeEnrichmentReport()` (in module *pyiomica.enrichmentAnalyses*), 25

F

`filterOutAllZeroSignals()` (*DataFrame method*), 26

`filterOutFractionMissingSignals()` (*DataFrame method*), 27

`filterOutFractionZeroSignals()` (*DataFrame method*), 27

`filterOutReferencePointZeroSignals()` (*DataFrame method*), 27

G

`GeneTranslation()` (in module *pyiomica.enrichmentAnalyses*), 19

`getAdjacencyMatrixOfHVG()` (in module *pyiomica.visibilityGraphAuxiliaryFunctions*), 42

`getAdjacencyMatrixOfHVGbyNUMPY()` (in module *pyiomica.visibilityGraphAuxiliaryFunctions*), 42

`getAdjacencyMatrixOfNVG()` (in module *pyiomica.visibilityGraphAuxiliaryFunctions*), 41

`getAdjacencyMatrixOfNVGbyNUMPY()` (in module *pyiomica.visibilityGraphAuxiliaryFunctions*), 42

`getCommunitiesOfTimeSeries()` (in module *pyiomica.clusteringFunctions*), 37

[getCommunityGenesDict\(\)](#) (in module *pyiomica.frequencySubjectMatch*), 39
[getCommunityStructure\(\)](#) (in module *pyiomica.frequencySubjectMatch*), 39
[getCommunityTopGenesByFrequencyRanking\(\)](#) (in module *pyiomica.frequencySubjectMatch*), 40
[getCommunityTopGenesByNumber\(\)](#) (in module *pyiomica.frequencySubjectMatch*), 40
[getEstimatedNumberOfClusters\(\)](#) (in module *pyiomica.clusteringFunctions*), 34
[GetGeneDictionary\(\)](#) (in module *pyiomica.enrichmentAnalyses*), 16
[getGroupingIndex\(\)](#) (in module *pyiomica.clusteringFunctions*), 36
[getLombScarglePeriodogramOfDataframe\(\)](#) (in module *pyiomica.extendedDataFrame*), 32
[getNClustersFromLinkageElbow\(\)](#) (in module *pyiomica.clusteringFunctions*), 34
[getNClustersFromLinkageSilhouette\(\)](#) (in module *pyiomica.clusteringFunctions*), 35
[getRandomAutocorrelations\(\)](#) (in module *pyiomica.extendedDataFrame*), 33
[getRandomPeriodograms\(\)](#) (in module *pyiomica.extendedDataFrame*), 33
[getRandomSpikesCutoffs\(\)](#) (in module *pyiomica.extendedDataFrame*), 32
[GOAnalysis\(\)](#) (in module *pyiomica.enrichmentAnalyses*), 18
[GOAnalysisAssigner\(\)](#) (in module *pyiomica.enrichmentAnalyses*), 16

I

[imputeMissingWithMedian\(\)](#) (*DataFrame* method), 31
[internalAnalysisFunction\(\)](#) (in module *pyiomica.enrichmentAnalyses*), 15

K

[KEGGAnalysis\(\)](#) (in module *pyiomica.enrichmentAnalyses*), 21
[KEGGAnalysisAssigner\(\)](#) (in module *pyiomica.enrichmentAnalyses*), 20
[KEGGDictionary\(\)](#) (in module *pyiomica.enrichmentAnalyses*), 20

M

[makeClusteringObject\(\)](#) (in module *pyiomica.clusteringFunctions*), 36
[makeDataHistograms\(\)](#) (in module *pyiomica.visualizationFunctions*), 45
[makeDendrogramHeatmapOfClusteringObject\(\)](#) (in module *pyiomica.visualizationFunctions*), 52
[makeLombScarglePeriodograms\(\)](#) (in module *pyiomica.visualizationFunctions*), 46

[makePlotOfPeak\(\)](#) (in module *pyiomica.visualizationFunctions*), 52
[makeVisibilityBarGraph\(\)](#) (in module *pyiomica.visualizationFunctions*), 51
[makeVisibilityGraph\(\)](#) (in module *pyiomica.visualizationFunctions*), 49
[MassDictionary\(\)](#) (in module *pyiomica.enrichmentAnalyses*), 23
[MassMatcher\(\)](#) (in module *pyiomica.enrichmentAnalyses*), 23
[mergeDataframes\(\)](#) (in module *pyiomica.extendedDataFrame*), 32
[modifiedZScore\(\)](#) (*DataFrame* method), 29
 module
 [pyiomica.categorizationFunctions](#), 11
 [pyiomica.clusteringFunctions](#), 34
 [pyiomica.dataStorage](#), 56
 [pyiomica.enrichmentAnalyses](#), 14
 [pyiomica.extendedDataFrame](#), 25
 [pyiomica.frequencySubjectMatch](#), 38
 [pyiomica.globalVariables](#), 9
 [pyiomica.utilityFunctions](#), 55
 [pyiomica.visibilityGraphAuxiliaryFunctions](#), 41
 [pyiomica.visibilityGraphCommunityDetection](#), 43
 [pyiomica.visualizationFunctions](#), 44

N

[normalizeSignalsToUnity\(\)](#) (*DataFrame* method), 30

O

[OBOGODictionary\(\)](#) (in module *pyiomica.enrichmentAnalyses*), 15
[obtainConstantGeneDictionary\(\)](#) (in module *pyiomica.enrichmentAnalyses*), 17
[optimizeK\(\)](#) (in module *pyiomica.frequencySubjectMatch*), 41

P

[PackageDirectory](#) (in module *pyiomica.globalVariables*), 9
[plotHVGBarGraphDual\(\)](#) (in module *pyiomica.visualizationFunctions*), 53
[plotNVGBarGraphDual\(\)](#) (in module *pyiomica.visualizationFunctions*), 54
[printPackageGlobalDefaults](#) (in module *pyiomica.globalVariables*), 9
[pyiomica.categorizationFunctions](#) module, 11
[pyiomica.clusteringFunctions](#) module, 34
[pyiomica.dataStorage](#)

module, 56
 pyiomica.enrichmentAnalyses
 module, 14
 pyiomica.extendedDataFrame
 module, 25
 pyiomica.frequencySubjectMatch
 module, 38
 pyiomica.globalVariables
 module, 9
 pyiomica.utilityFunctions
 module, 55
 pyiomica.visibilityGraphAuxiliaryFunctions
 module, 41
 pyiomica.visibilityGraphCommunityDetection
 module, 43
 pyiomica.visualizationFunctions
 module, 44

Q

quantileNormalize() (*DataFrame method*), 30

R

ReactomeAnalysis() (in module *pyiomica.enrichmentAnalyses*), 24
 read() (in module *pyiomica.dataStorage*), 56
 readMathIOmicaData() (in module *pyiomica.utilityFunctions*), 55
 removeConstantSignals() (*DataFrame method*), 29
 runCPUs() (in module *pyiomica.utilityFunctions*), 55
 runForClusterNum() (in module *pyiomica.clusteringFunctions*), 35

S

saveFigure() (in module *pyiomica.visualizationFunctions*), 45
 splitGenes() (in module *pyiomica.frequencySubjectMatch*), 40

T

tagLowValues() (*DataFrame method*), 28
 tagMissingAsValue() (*DataFrame method*), 28
 tagValueAsMissing() (*DataFrame method*), 28

V

visualizeTimeSeriesCategorization() (in module *pyiomica.categorizationFunctions*), 13

W

write() (in module *pyiomica.dataStorage*), 57