Documentation:
geneset$_{n}etwork_{a}nalysis_{t}ool$

Release 3.4.0

Viola Fanfani
Giovanni Stracquadanio

Jan 28, 2022

Contents

PyGNA is a Python Geneset Network Analysis package: it performs various statistical analyses on biological networks and genesets.

CHAPTER 1

Quickstart

1.1 Installation

The easiest and fastest way to install *pygna* using *conda*:

```
$ conda install -c stracquadaniolab -c bioconda -c conda-forge pygna
```

Alternatively you can install it through *pip*:

```
$ pip install pygna
```

We have prepared also a docker image that can be found at the following link:

https://github.com/stracquadaniolab/pygna/packages

By downloading the docker image named "Pygna", you will download a virtual machine with the latest version of pygna and its requirements. First download and install the docker environment. Then run it either by command line or by opening the application. We have setup the docker image in order to let you use it "out of the box" as stand-alone app, so just by running the following line:

```
$ docker run docker.pkg.github.com/stracquadaniolab/pygna/pygna:latest
```

you will be prompt to the Pygna help section.

You can use pygna commands I/O functions using the classic docker functionality. Here below is an example that paint the dataset statistics:

```
$ docker run --rm -v "$PWD:$PWD" -w "$PWD" docker.pkg.github.com/stracquadaniolab/
↪pygna/pygna:latest paint-datasets-stats pygna_gnt_results.csv pygna_gnt.png
```

1.2 Getting started

A typical *pygna* analysis consists of 3 steps:

1. Generate the RWR and SP matrices for the network you are using (once they are generated, you won't need to repeat the same step again)
2. Make sure that the input genesets are in the right format. If a network uses entrez ID, and your file is in HUGO symbols, use the pygna utility for the name conversion.
3. Run the analysis you are interested into.
4. Once you have the output tables, you can choose to visualize one or more plots.

Otherwise you can check our snakemake workflow (https://github.com/stracquadaniolab/workflow-pygna) for the full geneset analysis; our workflow contains sample data that you can use to familiarize with our software.

The examples below show some basic analysis that can be carried out with pygna.

1.2.1 Example 1: Running pygna GNT analysis

In the *min-working-example* we provide some basic files to run this minimal example. You can follow this instructions to run a network topology analysis on a single geneset file.

WARNING: Pay attention to the fact we set the number of permutations to 1000, if you are just willing to test the behaviour, use 50 or 100 to speed up the process*

```
$ cd ./your-path/min-working-example/

$ pygna build-rwr-diffusion barabasi.interactome.tsv --output-file interactome_RWR.
↪hdf5

$ pygna test-topology-rwr barabasi.interactome.tsv disgenet_cancer_groups_subset.gmt␣
↪interactome_RWR.hdf5 table_topology_rwr.csv --number-of-permutations 1000 --cores 4

$ pygna paint-datasets-stats table_topology_rwr.csv barplot_rwr.png
```

You can look at the plot of the results in the *barplot_rwr.png* file, and the corresponding table in *table_topology_rwr.csv*.

1.2.2 Example 2: Running pygna GNA analysis

In the *min-working-example* we provide some basic files to run this minimal example. You can follow this instructions to run a network association analysis between two genesets.

```
$ cd ./your-path/min-working-example/
```

Skip this step if the matrix is already computed

```
$ pygna build-rwr-diffusion barabasi.interactome.tsv --output-file interactome_RWR.
↪hdf5
```

The association analysis is run N x M times (N number of genesets, M number of pathways), we use only 50 permutations in this example to avoid long computations; however, the recommended value is 1000.

```
$ pygna test-association-rwr barabasi.interactome.tsv disgenet_cancer_groups_subset.
↪gmt interactome_RWR.hdf5 table_association_rwr.csv -B disgenet_cancer_groups_subset.
↪gmt --keep --number-of-permutations 100 --cores 4
```

If you don't include the –results-figure flag at the comparison step, plot the matrix as follows

```
$ pygna paint-comparison-matrix table_association_rwr.csv heatmap_association_rwr.png␣
↪--rwr --annotate
```

Be careful to include the -rwr flag for the right color coding and –annotate for annotating the heatmap with the pvalue of each test

The -k flag, keeps the -B geneset and permutes only on the set A.

WARNING: In this case, both A and B genesets are the same, usually you would use a different B geneset to check all the combinations.

If setname B is not passed, the analysis is run between each couple of setnames in the geneset.

```
$ pygna test-association-rwr barabasi.interactome.tsv disgenet_cancer_groups_subset.
↪gmt interactome_RWR.hdf5 table_within_comparison_rwr.csv --number-of-permutations␣
↪100 --cores 4
$ pygna paint-comparison-matrix table_within_comparison_rwr.csv heatmap_within_
↪comparison_rwr.png --rwr --single-geneset
```

CHAPTER 2

Tutorial

2.1 General Usage

2.1.1 Data Input/Output

PyGNA relies requires two data sources, a network and a geneset. Regardless of the test, the genes in the geneset are mapped to the input network (e.g. BioGrid) and then statistical tests are carried out.

Networks are read as tab separated text files (.tsv), where edges are represented by a node pair. An example of a .tsv file is the fowllowing:

```
node1\tnode2
node1\tnode3
node1\tnode4
node2\tnode5
```

Genesets use the GMT format, where each geneset is reported as:

```
<name>\t<descriptor>\tgene1\tgene2\tgene3...
```

Otherwise, a single geneset could be passed as a TXT file, with a list of genes, one for each line.

```
gene1
gene2
gene3
...
```

Since GMT file can have multiple genesets, PyGNA can either run the analyses on all of them or on a user-specified subset.

Results are stored in CSV files, along with the parameters of the analysis. Results can be easily visualised using PyGNA plotting utilities, and save either as PDF or PNG files.

2.1.2 Matrix computation

Computing the shortest path and the interaction probabilities between each pair of nodes in the network can be computationally taxing. However, since matrices are never modified by statistical tests, they can be computed as part of a pre-processing step.

For this reason, we have implemented a separate step for evaluating and saving the shortest path and RWR matrices.

```
$ pygna build-rwr-diffusion barabasi.interactome.tsv --output-file interactome_RWR.
↪hdf5
```

```
$ pygna build-distance-matrix barabasi.interactome.tsv interactome_SP.hdf5
```

2.1.3 Analysis

PyGNA provides commands for running geneset network topology (GNT) and geneset network analysis (GNA) tests. Running *pygna -h* shows all the available analyses, whereas extend documentation can be found *Command Line Tool* here.

Here a simplified structure of the available tests:

- **geneset network topology:**
  - module
  - internal degree
  - total degree
  - shortest path
  - random walk
- **geneset network association:**
  - shortest path
  - random walk

The analyses commands have all the same interface; for example, the available RWR GNT analysis options can be checked by running:

```
$ pygna test-topology-rwr -h
```

2.2 Complete analysis of one geneset

In case you have your own geneset you can completely characterise it using PyGNA as follows (names of min_working_example).

INPUT: <network> and <geneset>, and/or a <geneset_pathway> to run the association analysis.

OUTPUT: <network_sp>.hdf5, <network_rwr>.hdf5, <table_results_test>_<test>.csv

Generate the network matrices:

```
$ pygna build-distance-matrix <network> <network_sp>.hdf5
$ pygna build-rwr-diffusion <network> --output-file <network_rwr>.hdf5
```

Topology tests:

```
$ pygna test-topology-module <network> <geneset> <table_results_test>_topology_module.
↪csv --number-of-permutations 100 --cores 4
$ pygna test-topology-rwr <network> <geneset> <network_rwr>.hdf5 <table_results_test>_
↪topology_rwr.csv --number-of-permutations 100 --cores 4
$ pygna test-topology-internal-degree <network> <geneset> <table_results_test>_
↪topology_internal_degree.csv --number-of-permutations 100 --cores 4
$ pygna test-topology-sp <network> <geneset> <network_sp>.hdf5 <table_results_test>_
↪topology_sp.csv --number-of-permutations 100 --cores 4
$ pygna test-topology-total-degree <network> <geneset> <table_results_test>_topology_
↪total_degree.csv --number-of-permutations 100 --cores 4
```

Association tests:

```
$ pygna test-association-sp <network> <geneset> <network_sp>.hdf5 <table_results_test>
↪_association_sp.csv -B <geneset_pathways> --keep --number-of-permutations 100 --
↪cores 4
$ pygna test-association-rwr <network> <geneset> <network_rwr>.hdf5 <table_results_
↪test>_association_rwr.csv -B <geneset_pathways> --keep --number-of-permutations 100␣
↪--cores 4
```

2.3 Analysis of a geneset from a table (e.g. DeSeq2)

In many workflows, the genes to analyse are stored in a table-like format. Hence, we provide a function to create a GMT geneset from a table, with the possibility of applying a filter to the data. You can even just use it to return a GMT with all the genes in a column by applying a dummy filter.

NOTE: In case you would like to apply more filters, just use the output_csv command, instead of GMT, in order to only filter the original data and return the table in the original format.

Here, for example, we obtain a GMT file of the differentially expressed genes computed using DeSeq2, by selecting genes with padj<0.01.

```
$ pygna geneset-from-table <deseq>.csv diff_exp <deseq>.gmt --descriptor deseq
```

PyGNA implements a generic interface to filter any CSV file. Filters are applied to the values in the filter_column (for example pvalue) and the cut is performed using the alternative parameter. The threshold parameters are used to specify what the filter should be used. Bare in mind the filter is supposed to be applied to **numerical values**. The output GMT will have the gene-names in the <name_column>

```
$ pygna geneset-from-table <filename>.csv <setname> <filename>.gmt --name-colum <gene_
↪names_column> --filter-column <filter-col> <'less'> --threshold <th> --descriptor
↪<descriptor string>
```

2.4 Pipelines

PyGNA can be seamlessly integrated into Snakemake workflows, and we provide a basic [snakemake workflow](https://github.com/stracquadaniolab/workflow-pygna) to carry out network analysis with PyGNA.

2.4.1 Converting data using PyGNA

One of the most important feature in pygna is the possibility to convert a file from a format to another. PyGNA supports:

Converting into GMT format

Geneset in table-like formats (e.g. CSV) can be converted into GMT format as follows:

```
$ pygna geneset-from-table gene_analysis.csv brca --output-gmt brca_analysis.gmt -f␣
↪significant -d significant -n genes.Entrezid -t 0.5 -a greater
```

It is also possible to merge different setnames in a single gmt file through the function *generate-group-gmt*. You can override the default parameters, to match the columns in your table.

```
$ pygna generate-group-gmt gene_analysis.csv setnames_gmt.gmt group-col Cancer_
↪Setnames
```

If you want to add a column corresponding to the EntrezID or the gene's symbol, you can use the following command:

```
$ pygna convert-csv mygenefile.csv e2s original-col-name EntrezID new-name-col␣
↪Symbols geneset brca
```

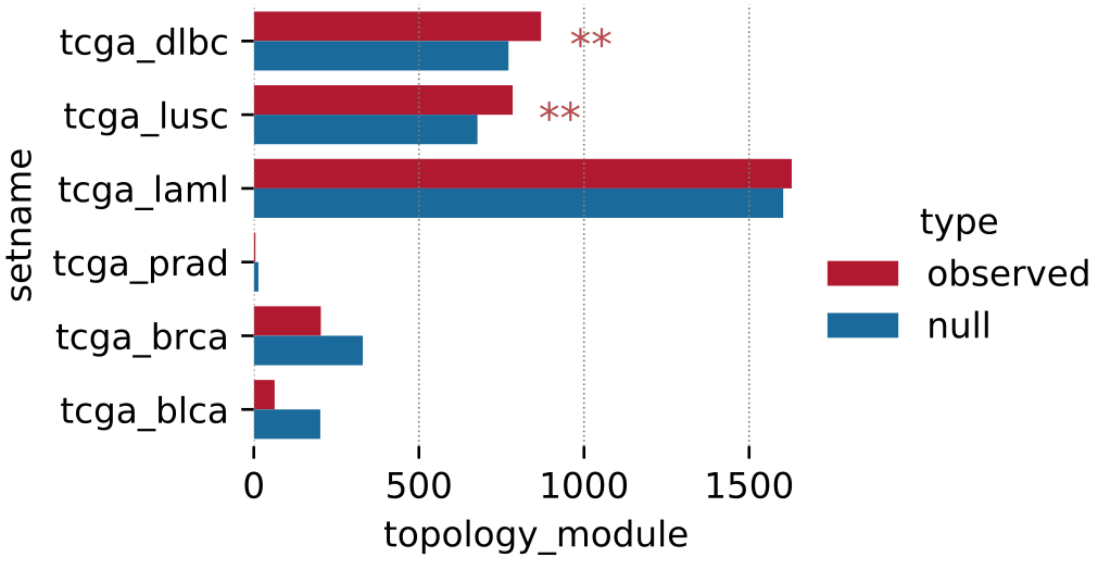
2.5 Showing the results

PyGNA generates publication-ready plots for each analysis.

For example, a barplot of the GNT RWR analysis for multiple genesets can be generated by running:

```
$ pygna paint-datasets-stats pygna_gnt_results.csv pygna_gnt.png
```

which produced a plot similar to the following:



For a complete list of the plots refer to *Visualisation*

2.6 Adding GNT or GNA test statistics

PyGNA can be easily extended to run user-defined statistical tests. Check " *Customizing the statistic elaboration* " for a full tutorial on how to do that.

2.6.1 Customizing the statistic elaboration

In the following section, it is shown how to use the pygna classes in order to customize the statistic elaboration

Adding a GNT test

In addition to the provided functions, the user can define and use any function. Follow the example below.

Step 0: define your function

Define your custom function that implements your statistical test. It must always return a *float*, representing the statistical value calculated in the function.

For example here it is reported the calculation of the geneset total degree

```
def geneset_total_degree_statistic(network, geneset):
    degree = nx.degree(network)
    geneset = list(geneset)
    total = np.array([degree[g] for g in geneset])
    return np.average(total)
```

Step 1: import the essential libraries

Import Pygna and its libraries

```
>>> from pygna import *
```

Step 2: loading your data

Load the network and the geneset you want to use, and initialise the output table

```
>>> network = pygna.reading_class.ReadTsv("mynetwork.tsv").get_network()
>>> geneset = rc.ReadGmt(geneset_file).get_geneset(setname)
```

```
>>> setnames = [key for key in geneset.keys()]
>>> output1 = out.Output(network_file, output_table, "topology_total_degree", geneset_
↪file, setnames)
```

Step 4: setup the statistical comparison

Pass your custom function to Pygna statistical class like:

```
>>> st_test = pygna.statistical_test.StatisticalTest(geneset_total_degree_statistic,␣
↪network)
```

Step 5: apply the function to all the genes and update the output table

```
>>> for setname, item in geneset.items():
>>>     if len(item) > 20:
>>>         item = set(item)
>>>         observed, pvalue, null_d, n_mapped, n_geneset = st_test.empirical_
↪pvalue(item,
...                                                                          ␣
↪max_iter=number_of_permutations,
...                                                                          ␣
↪alternative="greater",
...                                                                          ␣
↪cores=cores)
>>>         output1.update_st_table_empirical(setname, n_mapped, n_geneset, number_of_
↪permutations, observed,
...                                   pvalue, np.mean(null_d), np.var(null_d))
```

Step 6: save the results

```
>>>     output1.close_temporary_table()
```

Adding a GNA test

In addition to the provided functions, the user can define and use any function. Follow the example below.

Step 0: define your function

Before preparing the test define your custom function that implements your statistical test. It must always return a *float*, representing the statistical value calculated in the function.

For example here it is reported the calculation of comparison shortest path between two genesets. The return value is the average shortest path between the two genesets.

```
def comparison_shortest_path(network, genesetA, genesetB, diz):
    n = np.array([diz["nodes"].index(i) for i in genesetA])
    m = np.array([diz["nodes"].index(i) for i in genesetB])
    cum_sum = calculate_sum(n, m, diz)
    cum_sum += calculate_sum(m, n, diz)
    d_AB = cum_sum / float(len(genesetA) + len(genesetB))
    d_A = st.geneset_localisation_statistic(network, genesetA, diz)
    d_B = st.geneset_localisation_statistic(network, genesetB, diz)
    return d_AB - (d_A + d_B) / 2
```

Step 1: import the essential libraries

Import Pygna and its libraries.

```
>>> from pygna import *
```

Step 2: loading your data

Load the network and the geneset you want to use, if needed load an hdf5 table

```
>>> rw_dict = {"nodes": read_distance_matrix(rwr_matrix_filename, in_memory=in_
↪memory)[0],
...             "matrix": read_distance_matrix(rwr_matrix_filename, in_memory=in_
↪memory)[1]}
>>> network = pygna.reading_class.ReadTsv("mynetwork.tsv").get_network()
>>> geneset_a = rc.ReadGmt(file_geneset_a).get_geneset(setname_a)
```

Step 4: setup the statistical comparison

Pass your custom function to Pygna statistical class like:

```
>>> st_comparison = pygna.statistical_comparison.StatisticalComparison(comparison_
↪shortest_path, network, n_proc=cores, diz=rw_dict)
```

Step 5: apply the function to all the genes

You can now perform the statistical comparison among all pairs in the geneset and update the output table

```
>>> setnames = [key for key in geneset_a.keys()]
>>> for pair in itertools.combinations(setnames, 2):
>>>     if len(set(geneset_a[pair[0]])) > size_cut and len(set(geneset_a[pair[1]])) >␣
↪size_cut:
>>>         n_overlaps = len(set(geneset_a[pair[0]]).intersection(set(geneset_
↪a[pair[1]])))
>>>         observed, pvalue, null_d, a_mapped, b_mapped = st_comparison.comparison_
↪empirical_pvalue(
...         set(geneset_a[pair[0]]), set(geneset_a[pair[1]]), max_iter=number_of_
↪permutations, keep=keep)
>>>         output1.update_comparison_table_empirical(pair[0], pair[1],␣
↪len(set(geneset_a[pair[0]])), a_mapped,
...                                                   len(set(geneset_
↪a[pair[1]])), b_mapped, n_overlaps,
...                                                 number_of_permutations,␣
↪observed, pvalue, np.mean(null_d),
...                                                 np.var(null_d))
```

Step 6: save the results

Save the results using the pygna output function

```
>>>     output1.close_temporary_table()
```

It is possible to define custom functions outside the pygna package and use them in a stand-alone test. The code below shows how it is possible to implement a function such as the average closeness centrality of a geneset, using Pygna.

```
import argh
import logging
import networkx as nx
import pygna.reading_class as rc
import pygna.output as out
import pygna.statistical_test as st
import pygna.painter as paint
import pygna.diagnostic as diagnostic
import pygna.command as cmd
import numpy as np


def average_closeness_centrality(graph: nx.Graph, geneset: set, diz: dict, observed_
↪flag: bool = False) -> float:
    """
    This function calculates the average closeness centrality of a geneset.
    For a single node, the closeness centrality is defined as the inverse
    of the shortest path distance of the node from all the other nodes.

    Given a network with N nodes and a distance shortest path function
    between two nodes d(u,v)
    closeness centrality (u)= (N -1) / sum (v != u) d(u,v)

    where sp is the distance of the node with each other node and tot_sp is the total␣
↪shortest paths for the whole graph.

    """

    graph_centrality = []

    ids = [diz["nodes"].index(n) for n in geneset]
    graph_centrality = [(len(matrix["nodes"]) - 1) / matrix['vector'][idx] for idx in␣
↪ids]

    return np.mean(graph_centrality)



def test_topology_centrality(
    network_file: "network file",
    geneset_file: "GMT geneset file",
    distance_matrix_filename: "The matrix with the SP for each node",
    output_table: "output results table, use .csv extension",
    setname: "Geneset to analyse" = None,
    size_cut: "removes all genesets with a mapped length < size_cut" = 20,
    number_of_permutations: "number of permutations for computing the empirical pvalue
↪" = 500,
    cores: "Number of cores for the multiprocessing" = 1,
    results_figure: "barplot of results, use pdf or png extension" = None,
    in_memory: 'load hdf5 data onto memory' = False,
    diagnostic_null_folder: "plot null distribution, pass the folder where all the␣
↪figures are going to be saved "
                            "(one for each dataset)" = None):

    """
    This function calculates the average closeness centrality of a geneset.
    For a single node, the closeness centrality is defined as the inverse
```

(continues on next page)

(continued from previous page)

```
    of the shortest path distance of the node from all the other nodes.
    """



    logging.info("Evaluating the test topology total degree, please wait")
    network = rc.ReadTsv(network_file).get_network()
    network = nx.Graph(network.subgraph(max(nx.connected_components(network),␣
↪key=len)))
    geneset = rc.ReadGmt(geneset_file).get_geneset(setname)
    setnames = [key for key in geneset.keys()]


    diz = {"nodes": cmd.read_distance_matrix(distance_matrix_filename, in_memory=in_
↪memory)[0],
           "matrix": cmd.read_distance_matrix(distance_matrix_filename, in_memory=in_
↪memory)[1]}
    diz["matrix"] = diz["matrix"] + np.transpose(diz["matrix"])

    np.fill_diagonal(diz["matrix"], float(0))

    diz['vector'] = np.sum(diz["matrix"],axis = 0)

    # Generate output
    output1 = out.Output(network_file, output_table, "topology_total_degree", geneset_
↪file, setnames)
    logging.info("Results file = " + output1.output_table_results)
    # Create table
    output1.create_st_table_empirical()
    st_test = st.StatisticalTest(average_closeness_centrality, network, diz)
    for setname, item in geneset.items():
        # Geneset smaller than size cut are not taken into consideration
        if len(item) > size_cut:
            item = set(item)
            observed, pvalue, null_d, n_mapped, n_geneset = st_test.empirical_
↪pvalue(item,
                                                                                ␣
↪max_iter=number_of_permutations,
                                                                                ␣
↪alternative="greater",
                                                                                ␣
↪cores=cores)
            logging.info("Setname:" + setname)
            if n_mapped < size_cut:
                logging.info("%s removed from results since nodes mapped are < %d" %␣
↪(setname, size_cut))
            else:
                logging.info("Observed: %g p-value: %g" % (observed, pvalue))
                logging.info("Null mean: %g null variance: %g".format(np.mean(null_d),
↪ np.var(null_d)))
                output1.update_st_table_empirical(setname, n_mapped, n_geneset,␣
↪number_of_permutations, observed,
                                                  pvalue, np.mean(null_d), np.var(null_
↪d))
                if diagnostic_null_folder:
                    diagnostic.plot_null_distribution(null_d, observed, diagnostic_
↪null_folder + setname +
```

(continues on next page)

(continued from previous page)

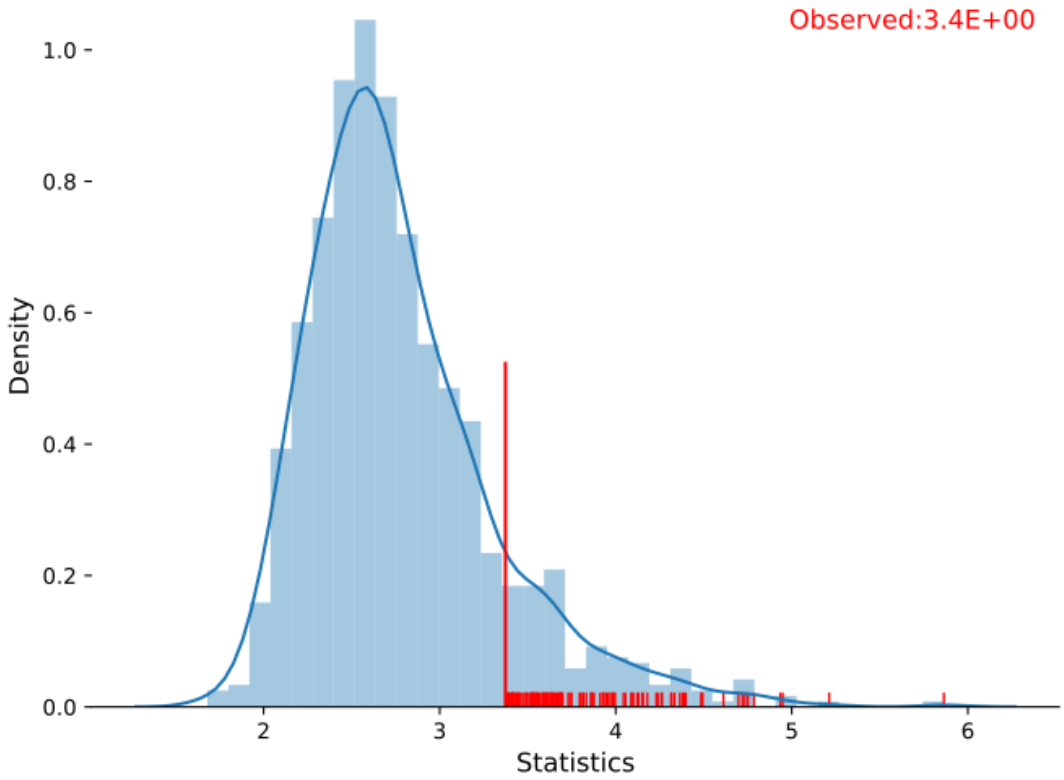
```
                                                    '_total_degree_null_distribution.
↪pdf', setname=setname)
    output1.close_temporary_table()
    if results_figure:
        paint.paint_datasets_stats(output1.output_table_results, results_figure,␣
↪alternative='greater')
    logging.info("Test topology CENTRALITY completed")


def main():
    """
    argh dispatch
    """
    argh.dispatch_commands([test_topology_centrality])
```

2.7 Diagnostic

2.7.1 Distribution plot

When running a statistical test, one might want to visually assess the null distribution. By passing *-d <diagnostic_folder/>* through command line, a distribution plot of the empirical null is shown for each test.

Here is an example.


Observed:3.4E+00
1.0
0.8
0.6
0.4
0.2
0.0
Density
2
3
4
5
6
Statistics

CHAPTER 3

Command Line Tool

PyGNA provides a complete analysis pipeline from command line.

The user can easily have summary statistics on a network or a single dataset or run a full analysis complete with visualisation features. Here we describe how to access all of them.

3.1 Network Info

Before running any geneset-network analysis is a good practice to extract basic information on the network and the geneset or to visualise the network. We provide a function to obtain the network summary (or the summary of a single genesets) and another utility to write an annotated graphml file (visualise it on Cytoscape).

3.1.1 Network Properties

network-summary saves the principal info of a graph:

- network properties
- degree distribution
- connected components diagnostic

If a geneset/setname is passed to the function, the properties of the subgraph are evaluated

```
pygna network-summary [-h] [-g GENESET_INPUT_FILE] [-s SETNAME] network-file text-
↪output degree-figure-file c-components-figure-file

positional arguments:
  network-file          network file
  text-output           output text file for the summary
  degree-figure-file    pdf or png file for the degree distribution
  c-components-figure-file
                        pdf or png file for the connected components distribution
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help            show this help message and exit
  -g GENESET_INPUT_FILE, --geneset-input-file GENESET_INPUT_FILE
                        geneset file (default: -)
  -s SETNAME, --setname SETNAME
                        specify a single geneset (default: -)
```

3.1.2 Cytoscape visualisation

network-graphml generates a graphml file with nodes annotation. Given a geneset, with k setnames, each node has k False/True annotations for each set.

Warning: without minimal, this function saves the full network. The minimal graph saves only the nodes in the geneset and those that connect them with a shortest path.

```
usage: pygna network-graphml [-h] [-s SETNAME] [--giant-component-only] [-m] network-
↪file geneset-file output-file

positional arguments:
  network-file          network file
  geneset-file          geneset file
  output-file           graphml file for network for visualisation

optional arguments:
  -h, --help            show this help message and exit
  -s SETNAME, --setname SETNAME
                        set name (default: -)
  --giant-component-only
                        saves only the giant component of the network (default: True)
  -m, --minimal         saves only the minimal graph (default: False)
```

3.1.3 Connected components

get-connected-components evaluate all the connected components in the subgraph pf the network with a given setname. Multiple setnames can be passed to this function to analyze all of them in a run. The file produces a GMT output and optionally a plot of the subnetwork with the connected components analysed.

Please notice that to convert the entrezID into Symbols, a stable internet connection is required

```
usage: pygna get-connected-components [-h] [--geneset-file GENESET_FILE]
                                      [-s SETNAME] [--graphml GRAPHML]
                                      [-t THRESHOLD] [-c]
                                      network-file output-gmt name

    This function evaluate all the connected components in the subgraph pf the␣
↪network with a given setname.
    Multiple setnames can be passed to this function to analyze all of them in a run.
    The file produces a GMT output and optionally a plot of the subnetwork with the␣
↪connected components analysed.
    Please notice that to convert the entrezID into Symbols, a stable internet␣
↪connection is required
```

(continues on next page)

(continued from previous page)

```
positional arguments:
network-file          network tsv file
output-gmt            The output file name (should be gmt)
name                  pass a name for the putput gmt terms

optional arguments:
-h, --help            show this help message and exit
--geneset-file GENESET_FILE
                        GMT of the geneset file, is a file is passed please
                        add the setname (default: -)
-s SETNAME, --setname SETNAME
                        The setname to analyse (default: -)
--graphml GRAPHML     Pass a graphml filename to show the results on
                        Cytoscape (default: -)
-t THRESHOLD, --threshold THRESHOLD
                        ignores all CC smaller than this value (default: 1)
-c, --convert-entrez  pass flag to convert EntrezID->Symbol (default: False)
```

3.2 Large Matrices creation

Along with the network and the geneset, some analyses require an additional large matrix to be passed as input. In particular, the analysis of shortest path and diffusion are evaluating a matrix of shape NxN (N being the number of nodes), since those are invariant to the geneset analysed, they must be evaluated and saved only once.

3.2.1 Shortest Paths matrix

Build a shortest path distance matrix for a given network. Matrix can be saved as a .txt file or a .hdf5 one.

```
usage: pygna build-distance-matrix [-h] [-g] network-file output-file

positional arguments:
  network-file          network file
  output-file           distance matrix output file, use .hdf5

optional arguments:
  -h, --help            show this help message and exit
  -g, --giant-component-only
                        compute the shortest paths only for nodes in the giant␣
↪component (default: True)
```

3.2.2 Diffusion matrix

To evaluate the diffusion matrix we have the below function that implements a Randowm Walk with Restart algorithm. The $beta$ parameter is set to 0.80 as default, but can be given by the user.

```
usage: pygna build-rwr-diffusion [-h] [-b BETA] [-o OUTPUT_FILE] network-file

positional arguments:
  network-file          network file

optional arguments:
```

(continues on next page)

(continued from previous page)

```
  -h, --help            show this help message and exit
  -b BETA, --beta BETA  0.85
  -o OUTPUT_FILE, --output-file OUTPUT_FILE
                        distance matrix output file (use .hdf5) (default: -)
```

3.3 Converting tables and names

3.3.1 Dataset from table

Converts a csv file to a GMT allowing to filter the elements using the values of one of the columns. The user can specify the column used to retrieve the name of the objects and the filter condition. The output can be either a GMT with the names of the genes that pass the filter or a csv with the whole filtered table, otherwise both can be created.

```
usage: pygna geneset-from-table [-h] [--output-gmt OUTPUT_GMT] [--output-csv OUTPUT_
↪CSV] [-n NAME_COLUMN] [-f FILTER_COLUMN] [-a ALTERNATIVE] [-t THRESHOLD]
                                [-d DESCRIPTOR]
                                input-file setname

positional arguments:
  input-file            input csv file
  setname               name of the set

optional arguments:
  -h, --help            show this help message and exit
  --output-gmt OUTPUT_GMT
                        output gmt name (default: -)
  --output-csv OUTPUT_CSV
                        output csv name (default: -)
  -n NAME_COLUMN, --name-column NAME_COLUMN
                        column with the names (default: 'Unnamed: 0')
  -f FILTER_COLUMN, --filter-column FILTER_COLUMN
                        column with the values to be filtered (default: 'padj')
  -a ALTERNATIVE, --alternative ALTERNATIVE
                        alternative to use for the filter, with less the filter is␣
↪applied <threshold, otherwise >= threshold (default: 'less')
  -t THRESHOLD, --threshold THRESHOLD
                        threshold for the filter (default: 0.01)
  -d DESCRIPTOR, --descriptor DESCRIPTOR
                        descriptor for the gmt file (default: -)
```

3.3.2 Convert gene names

convert-gmt is used to convert a GMT file, adding information about the Entrez ID or the symbol

```
usage: pygna convert-gmt [-h] [-e ENTREZ_COL] [-s SYMBOL_COL] gmt-file output-gmt-
↪file conversion converter-map-filename

positional arguments:
  gmt-file              gmt file to be converted
  output-gmt-file       output file
  conversion            e2s or s2e
  converter-map-filename
```

(continues on next page)

(continued from previous page)

```
                        tsv table used to convert gene names

optional arguments:
  -h, --help            show this help message and exit
  -e ENTREZ_COL, --entrez-col ENTREZ_COL
                        name of the entrez column (default: 'NCBI Gene ID')
  -s SYMBOL_COL, --symbol-col SYMBOL_COL
                        name of the symbol column (default: 'Approved symbol')
```

generate-group-gmt generates a GMT file of multiple setnames. From the table file, it groups the names in the group_col (the column you want to use to group them) and prints the genes in the name_col. Set the descriptor according to your needs

```
usage: pygna generate-group-gmt [-h] [-n NAME_COL] [-g GROUP_COL] [-d DESCRIPTOR]␣
↪input-table output-gmt

positional arguments:
  input-table           table to get the geneset from
  output-gmt            output GMT file

optional arguments:
  -h, --help            show this help message and exit
  -n NAME_COL, --name-col NAME_COL
                        'Gene'
  -g GROUP_COL, --group-col GROUP_COL
                        'Cancer'
  -d DESCRIPTOR, --descriptor DESCRIPTOR
                        'cancer_genes'
```

convert-csv is used to add a column with the entrezID or Symbols to a CSV file.

```
usage: pygna convert-csv [-h] [--converter-map-filename CONVERTER_MAP_FILENAME] [--
↪output-file OUTPUT_FILE] [-e ENTREZ_COL] [-s SYMBOL_COL]
                     csv-file conversion original-name-col new-name-col geneset

positional arguments:
  csv-file              csv file where to add a name column
  conversion            e2s or s2e
  original-name-col     column name to be converted
  new-name-col          name of the new column with the converted names
  geneset               the geneset to convert

optional arguments:
  -h, --help            show this help message and exit
  --converter-map-filename CONVERTER_MAP_FILENAME
                        tsv table used to convert gene names (default: 'entrez_name.
↪tsv')
  --output-file OUTPUT_FILE
                        if none, table is saved in the same input file (default: -)
  -e ENTREZ_COL, --entrez-col ENTREZ_COL
                        name of the entrez column (default: 'NCBI Gene ID')
  -s SYMBOL_COL, --symbol-col SYMBOL_COL
                        name of the symbol column (default: 'Approved symbol')
```

3.4 Geneset Network Topology Analysis

Here are all the GNT analyses that pygna can perform. For each of them a single geneset topology is tested with the specified test statistics.

3.4.1 GNT Module analysis

Test Topology Module

test-topology-module performs geneset network topology module analysis.

It computes a p-value for the largest connected component of the geneset being bigger than the one expected by chance for a geneset of the same size.

```
usage: pygna test-topology-module [-h] [--setname SETNAME] [--size-cut SIZE_CUT] [--
↪number-of-permutations NUMBER_OF_PERMUTATIONS] [-c CORES] [--n-bins N_BINS] [--
↪output-lcc OUTPUT_LCC]
                                [-r RESULTS_FIGURE] [-d DIAGNOSTIC_NULL_FOLDER]
                                network-file geneset-file output-table

    Performs geneset network topology module analysis.
    It computes a p-value for the largest connected component of the geneset being␣
↪bigger than the one expected by chance
    for a geneset of the same size.


positional arguments:
network-file          network file
geneset-file          GMT geneset file
output-table          output results table, use .csv extension

optional arguments:
-h, --help            show this help message and exit
--setname SETNAME     Geneset to analyse (default: -)
--size-cut SIZE_CUT   removes all genesets with a mapped length < size_cut (default:␣
↪20)
--number-of-permutations NUMBER_OF_PERMUTATIONS
                        number of permutations for computing the empirical pvalue␣
↪(default: 500)
-c CORES, --cores CORES
                        Number of cores for the multiprocessing (default: 1)
--n-bins N_BINS       if >1 applies degree correction by binning the node degrees and␣
↪sampling according to geneset distribution (default: 1)
--output-lcc OUTPUT_LCC
                        for creating a GMT file with the LCC lists pass a GMT␣
↪filename (default: -)
-r RESULTS_FIGURE, --results-figure RESULTS_FIGURE
                        barplot of results, use pdf or png extension (default: -)
-d DIAGNOSTIC_NULL_FOLDER, --diagnostic-null-folder DIAGNOSTIC_NULL_FOLDER
                        plot null distribution, pass the folder where all the figures␣
↪are going to be saved (one for each dataset) (default: -)
```

3.4.2 GNT Degree analysis

Test topology internal degree

test-topology-internal-degree performs the analysis of internal degree. It computes a p-value for the ratio of internal degree of the geneset being bigger than the one expected by chance for a geneset of the same size.

```
usage: pygna test-topology-internal-degree [-h] [--setname SETNAME] [--size-cut SIZE_
↪CUT] [--number-of-permutations NUMBER_OF_PERMUTATIONS] [-c CORES] [--n-bins N_BINS]␣
↪[-r RESULTS_FIGURE]
                                           [-d DIAGNOSTIC_NULL_FOLDER]
                                           network-file geneset-file output-table

    Performs the analysis of internal degree.
    It computes a p-value for the ratio of internal degree of the geneset being␣
↪bigger than the one expected by chance
    for a geneset of the same size.


positional arguments:
network-file          network file
geneset-file          GMT geneset file
output-table          output results table, use .csv extension

optional arguments:
-h, --help            show this help message and exit
--setname SETNAME     Geneset to analyse (default: -)
--size-cut SIZE_CUT   removes all genesets with a mapped length < size_cut (default:␣
↪20)
--number-of-permutations NUMBER_OF_PERMUTATIONS
                        number of permutations for computing the empirical pvalue␣
↪(default: 500)
-c CORES, --cores CORES
                        Number of cores for the multiprocessing (default: 1)
--n-bins N_BINS       if >1 applies degree correction by binning the node degrees and␣
↪sampling according to geneset distribution (default: 1)
-r RESULTS_FIGURE, --results-figure RESULTS_FIGURE
                        barplot of results, use pdf or png extension (default: -)
-d DIAGNOSTIC_NULL_FOLDER, --diagnostic-null-folder DIAGNOSTIC_NULL_FOLDER
                        plot null distribution, pass the folder where all the figures␣
↪are going to be saved (one for each dataset) (default: -)
```

Test topology total degree

test-topology-total-degree performs the analysis of total degree of the network. It computes a p-value for the ratio of total degree of the geneset being bigger than the one expected by chance for a geneset of the same size.

```
usage: pygna test-topology-total-degree [-h] [--setname SETNAME] [--size-cut SIZE_
↪CUT] [--number-of-permutations NUMBER_OF_PERMUTATIONS] [-c CORES] [-r RESULTS_
↪FIGURE]
                                        [-d DIAGNOSTIC_NULL_FOLDER]
                                        network-file geneset-file output-table

    Performs the analysis of total degree of the .

    It computes a p-value for the ratio of total degree of the geneset being bigger␣
↪than the one expected by chance
    for a geneset of the same size.
```

(continues on next page)

(continued from previous page)

```
positional arguments:
network-file          network file
geneset-file          GMT geneset file
output-table          output results table, use .csv extension

optional arguments:
-h, --help            show this help message and exit
--setname SETNAME     Geneset to analyse (default: -)
--size-cut SIZE_CUT   removes all genesets with a mapped length < size_cut (default:␣
↪20)
--number-of-permutations NUMBER_OF_PERMUTATIONS
                      number of permutations for computing the empirical pvalue␣
↪(default: 500)
-c CORES, --cores CORES
                      Number of cores for the multiprocessing (default: 1)
-r RESULTS_FIGURE, --results-figure RESULTS_FIGURE
                      barplot of results, use pdf or png extension (default: -)
-d DIAGNOSTIC_NULL_FOLDER, --diagnostic-null-folder DIAGNOSTIC_NULL_FOLDER
                      plot null distribution, pass the folder where all the figures
                      are going to be saved (one for each dataset) (default: -)
```

3.4.3 GNT Shortest Path Analysis

Test topology shortest path

test-topology-sp performs geneset network topology shortest path analysis. It computes a p-value for the average shortest path length of the geneset being smaller than expected by chance for a geneset of the same size.

Consider that the computation of the shortest path matrix for a network can be considerably slow, however it only needs to be done once for each network (and can then be reused). We have separated the generation of the matrix step, refer to *build-distance-matrix*.

```
usage: pygna test-topology-sp [-h] [--setname SETNAME] [--size-cut SIZE_CUT] [--
↪number-of-permutations NUMBER_OF_PERMUTATIONS] [-c CORES] [-i] [--n-bins N_BINS] [-
↪r RESULTS_FIGURE]
                              [--diagnostic-null-folder DIAGNOSTIC_NULL_FOLDER]
                              network-file geneset-file distance-matrix-filename output-
↪table

    Performs geneset network topology shortest path analysis.

    It computes a p-value for the average shortest path length
    of the geneset being smaller than expected by chance
    for a geneset of the same size.


positional arguments:
network-file          network file
geneset-file          GMT geneset file
distance-matrix-filename
                      distance hdf5 matrix file generated by pygna
output-table          output results table, use .csv extension
```

(continues on next page)

(continued from previous page)

```
optional arguments:
-h, --help            show this help message and exit
--setname SETNAME     Geneset to analyse (default: -)
--size-cut SIZE_CUT   removes all genesets with a mapped length < size_cut (default:␣
↪20)
--number-of-permutations NUMBER_OF_PERMUTATIONS
                      number of permutations for computing the empirical pvalue␣
↪(default: 500)
-c CORES, --cores CORES
                      Number of cores for the multiprocessing (default: 1)
-i, --in-memory       set if you want the large matrix to be read in memory (default:␣
↪False)
--n-bins N_BINS       if >1 applies degree correction by binning the node degrees and␣
↪sampling according to geneset distribution (default: 1)
-r RESULTS_FIGURE, --results-figure RESULTS_FIGURE
                      barplot of results, use pdf or png extension (default: -)
--diagnostic-null-folder DIAGNOSTIC_NULL_FOLDER
                      plot null distribution, pass the folder where all the figures␣
↪are going to be saved (one for each dataset) (default: -)
```

3.4.4 GNT Random Walk Analysis

Test topology random walk with restart

test-topology-rwr performs the analysis of random walk probabilities. Given the Random Walk with Restart matrix, it compares the probability of walking between the genes in the geneset compared to those of walking between the nodes of a geneset with the same size

For the generation of the RWR matrix check *build-rwr-diffusion*

```
usage: pygna test-topology-rwr [-h] [--setname SETNAME] [--size-cut SIZE_CUT] [--
↪number-of-permutations NUMBER_OF_PERMUTATIONS] [-c CORES] [-i] [--n-bins N_BINS]
                               [--results-figure RESULTS_FIGURE] [-d DIAGNOSTIC_NULL_
↪FOLDER]
                               network-file geneset-file rwr-matrix-filename output-table

    Performs the analysis of random walk probabilities.
    Given the RWR matrix, it compares the probability of walking between the genes in␣
↪the geneset compared to
    those of walking between the nodes of a geneset with the same size


positional arguments:
network-file          network file, use a network with weights
geneset-file          GMT geneset file
rwr-matrix-filename   hdf5 RWR matrix obtained with pygna
output-table          output results table, use .csv extension

optional arguments:
-h, --help            show this help message and exit
--setname SETNAME     Geneset to analyse (default: -)
--size-cut SIZE_CUT   removes all genesets with a mapped length < size_cut (default:␣
↪20)
--number-of-permutations NUMBER_OF_PERMUTATIONS
                      number of permutations for computing the empirical pvalue␣
↪(default: 500)
```

(continues on next page)

(continued from previous page)

```
-c CORES, --cores CORES
                        Number of cores for the multiprocessing (default: 1)
-i, --in-memory       set if you want the large matrix to be read in memory (default:␣
↪False)
--n-bins N_BINS       if >1 applies degree correction by binning the node degrees and␣
↪sampling according to geneset distribution (default: 1)
--results-figure RESULTS_FIGURE
                        barplot of results, use pdf or png extension (default: -)
-d DIAGNOSTIC_NULL_FOLDER, --diagnostic-null-folder DIAGNOSTIC_NULL_FOLDER
                        plot null distribution, pass the folder where all the figures␣
↪are going to be saved (one for each dataset) (default: -)
```

3.5 Geneset Network Association Analysis

Here are all the GNA analyses that PyGNA can perform. For each of them two sets are compared, one can pass:

- only one geneset: association between each set is computed
- two genesets: association between all the terms in the two genesets is computed

3.5.1 GNA Shortest Path

Test association shortest path

pygna test-association-sp performs comparison of network location analysis. If the flag –keep is passed, the B geneset is kept fixed, and doesnt't get permuted It computes a p-value for the shortest path distance between two genesets being smaller than expected by chance If only A_geneset_file is passed the analysis is run on all the pair of sets in the file, if both A_geneset_file and B_geneset_file are passed, one can specify the setnames for both, if there is only one geneset in the file, setname_X can be omitted, if both sets are in the same file, B_geneset_file can be not specified, but setnames are needed.

```
usage: pygna test-association-sp [-h] [--setname-a SETNAME_A] [--file-geneset-b FILE_
↪GENESET_B] [--setname-b SETNAME_B] [--size-cut SIZE_CUT] [-k] [-c CORES] [-i]
                                 [--number-of-permutations NUMBER_OF_PERMUTATIONS] [--
↪n-bins N_BINS] [-r RESULTS_FIGURE]
                                 network-file file-geneset-a distance-matrix-filename␣
↪output-table

    Performs comparison of network location analysis. If the flag
    --keep  is passed, the B geneset is kept fixed, and doesnt't get permuted.

    It computes a p-value for the shortest path distance between two genesets being␣
↪smaller than expected by chance
    If only A_geneset_file is passed the analysis is run on all the pair of sets in␣
↪the file, if both
    A_geneset_file and B_geneset_file are passed, one can specify the setnames for␣
↪both, if there is only one
    geneset in the file, setname_X can be omitted, if both sets are in the same file,␣
↪B_geneset_file can be not
    specified, but setnames are needed.


positional arguments:
```

(continues on next page)

(continued from previous page)

```
network-file          network file
file-geneset-a        GMT geneset file, if it's the only parameter passed the␣
↪analysis is gonna be run on all the pair of datasets, otherwise specify the other␣
↪files and setnames
distance-matrix-filename
                        distance matrix file generated by pygna
output-table          output results table, use .csv extension

optional arguments:
-h, --help            show this help message and exit
--setname-a SETNAME_A
                        Geneset A to analyse (default: -)
--file-geneset-b FILE_GENESET_B
                        GMT geneset file (default: -)
--setname-b SETNAME_B
                        Geneset B to analyse (default: -)
--size-cut SIZE_CUT   removes all genesets with a mapped length < size_cut (default:␣
↪20)
-k, --keep            if true, keeps the geneset B not permuted (default: False)
-c CORES, --cores CORES
                        Number of cores for the multiprocessing (default: 1)
-i, --in-memory       set if you want the large matrix to be read in memory (default:␣
↪False)
--number-of-permutations NUMBER_OF_PERMUTATIONS
                        number of permutations for computing the empirical pvalue␣
↪(default: 500)
--n-bins N_BINS       if >1 applies degree correction by binning the node degrees and␣
↪sampling according to geneset distribution (default: 1)
-r RESULTS_FIGURE, --results-figure RESULTS_FIGURE
                        barplot of results, use pdf or png extension (default: -)
```

3.5.2 GNA Random Walk

Test association random walk with restart

test-association-rwr performs comparison of network location analysis. It computes a p-value for the shortest path distance between two genesets being smaller than expected by chance If only A_geneset_file is passed the analysis is run on all the couples of sets in the file, if both A_geneset_file and B_geneset_file are passed, one can specify the setnames for both, if there is only one geneset in the file, setname_X can be omitted, if both sets are in the same file, B_geneset_file can be not specified, but setnames are needed.

```
usage: pygna test-association-rwr [-h] [--setname-a SETNAME_A] [--file-geneset-b FILE_
↪GENESET_B] [--setname-b SETNAME_B] [--size-cut SIZE_CUT] [-k] [-c CORES] [-i]
                                  [--number-of-permutations NUMBER_OF_PERMUTATIONS] [--
↪n-bins N_BINS] [--results-figure RESULTS_FIGURE]
                                  network-file file-geneset-a rwr-matrix-filename␣
↪output-table

    Performs comparison of network location analysis.

    It computes a p-value for the shortest path distance
    between two genesets being smaller than expected by chance.

    If only A_geneset_file is passed the analysis is run on all the pair of sets in␣
↪the file, if both
```

(continues on next page)

(continued from previous page)

```
    A_geneset_file and B_geneset_file are passed, one can specify the setnames for␣
↪both, if there is only one
    geneset in the file, setname_X can be omitted, if both sets are in the same file,␣
↪B_geneset_file can be not
    specified, but setnames are needed.


positional arguments:
network-file          network file
file-geneset-a        GMT geneset file
rwr-matrix-filename   .hdf5 file with the RWR matrix obtained by pygna
output-table          output results table, use .csv extension

optional arguments:
-h, --help            show this help message and exit
--setname-a SETNAME_A
                        Geneset A to analyse (default: -)
--file-geneset-b FILE_GENESET_B
                        GMT geneset file (default: -)
--setname-b SETNAME_B
                        Geneset B to analyse (default: -)
--size-cut SIZE_CUT   removes all genesets with a mapped length < size_cut (default:␣
↪20)
-k, --keep            if true, keeps the geneset B unpermuted (default: False)
-c CORES, --cores CORES
                        Number of cores for the multiprocessing (default: 1)
-i, --in-memory       set if you want the large matrix to be read in memory (default:␣
↪False)
--number-of-permutations NUMBER_OF_PERMUTATIONS
                        number of permutations for computing the empirical pvalue␣
↪(default: 500)
--n-bins N_BINS       if >1 applies degree correction by binning the node degrees and␣
↪sampling according to geneset distribution (default: 1)
--results-figure RESULTS_FIGURE
                        heatmap of results (default: -)
```

3.6 Weights Diffusion Analysis

3.6.1 Test diffusion hotnet

Performs the analysis of random walk applying the weights of an upstream analysis. Given a csv file the user needs to specify the columns of interest and the threshold of significance. For the analysis the StatisticalDiffusion is used with hotnet_diffusion_statistic function.

```
usage: pygna test-diffusion-hotnet [-h] [--name-column NAME_COLUMN] [-w WEIGHT_
↪COLUMN] [--filter-column FILTER_COLUMN] [--filter-condition FILTER_CONDITION]
                                   [--filter-threshold FILTER_THRESHOLD] [--normalise] [-
↪s SIZE_CUT] [--number-of-permutations NUMBER_OF_PERMUTATIONS] [-c CORES] [-i]
                                   network-file geneset-file rwr-matrix-filename output-
↪table

positional arguments:
  network-file          network file, use a network with weights
```

(continues on next page)

(continued from previous page)

```
  geneset-file          csv geneset file
  rwr-matrix-filename   hdf5 RWR matrix obtained with pygna
  output-table          output results table, use .csv extension

optional arguments:
  -h, --help            show this help message and exit
  --name-column NAME_COLUMN
                        Column to use as name (default is deseq2) (default: 'gene_name
↪')
  -w WEIGHT_COLUMN, --weight-column WEIGHT_COLUMN
                        Column to use as weight (default is deseq2) (default: 'stat')
  --filter-column FILTER_COLUMN
                        Column used to define the significant genes (default is␣
↪deseq2) (default: 'padj')
  --filter-condition FILTER_CONDITION
                        Condition for significance (default: 'less')
  --filter-threshold FILTER_THRESHOLD
                        threshold for significance (default: 0.01)
  --normalise           pass this flag for using only positive values in the analysis␣
↪(default: False)
  -s SIZE_CUT, --size-cut SIZE_CUT
                        removes all genesets with a mapped length < size_cut␣
↪(default: 20)
  --number-of-permutations NUMBER_OF_PERMUTATIONS
                        number of permutations for computing the empirical pvalue␣
↪(default: 500)
  -c CORES, --cores CORES
                        Number of cores for the multiprocessing (default: 1)
  -i, --in-memory       set if you want the large matrix to be read in memory␣
↪(default: False)
```

3.7 Hypothesis testing

For both GNT and GNA we carry out hypothesis testing where the null hypothesis is that the statistic for the tested geneset is not more extreme than expected by change. Since closed form definitions for all null distributions are difficult to define, we compute the null distributions by randomly resampling the nodes of the network conditioned on the geneset size.

For example, for a GNT on a geneset of size n by passing the Number Of Permutations NOP parameter, NOP random sets or nodes are drawn and on each of the test statistic to build the null distribution.

Let $\bar{q}$ the observed value for the statistic and Q the null distribution we derive an empirical p-value as follows:

$$P(\bar{q} \geq Q) = \frac{(\sum_{i=1}^{NOP} I(Q_i \geq \bar{q})) + 1}{NOP + 1}$$

where I is the indicator function returning 1 if and only if the evaluated condition is true and continuity correction is applied.

3.7.1 Degree corrected sampling

We also allow to correct the resampling strategy on the degree distribution of the input geneset. This way the null distribution is normalised on the degree of the test set.

To enable this function pass the parameter *–n-bins Nb* with $Nb > 1$. In this case, the degrees of all nodes in the network are estimated and their frequency computed on a binned histogram. The Nb parameter is indeed used to specify the number of bins for the histogram. Consider that a low number of bins corresponds to a low resolution of the histogram, however specifying too many bins might correspond to sampling at random from the same nodes of the tested geneset.

In particular the probability of sampling node y is proportional to (tested nodes in bin k)/((tested nodes) (all nodes in bin k))

3.7.2 GNA: association sampling

In a GNA two genesets are tested for their association. When testing a signle geneset against many pathways we recommend the use of the *–keep* flag. This way, while resampling only the geneset a will be randomly permuted and the geneset b is going to be kept as it is. This strategy is more conservative and is helpful in testing whether the tested geneset is more strongly connected to the pathway (or any other geneset of interest) than expected by chance.

3.8 Visualisation

Pygna offers a range of plots to visualise the results of the analysis.

3.8.1 GNT plots

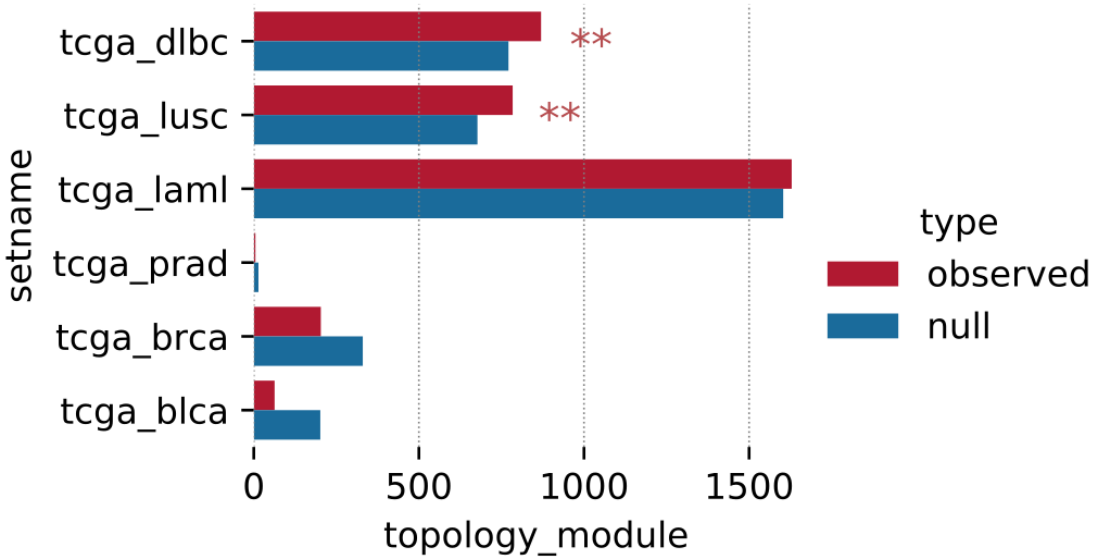
GNT barplot

paint-dataset-stats plots the results of of a GNT test on a barplot. Pass the results table generated by one of the functions and the output figure file (png or pdf). In case you are using a SP test, pass also 'less' as an alternative.

```
usage: pygna paint-datasets-stats [-h] [-a ALTERNATIVE] table-filename output-file

positional arguments:
  table-filename        pygna results table
  output-file           figure file, use pdf or png extension

optional arguments:
  -h, --help            show this help message and exit
  -a ALTERNATIVE, --alternative ALTERNATIVE
                        'greater'
```

Example usage: .. code-block:: bash

    $ pygna paint-dataset-stats table_topology_module.csv figures/barplot_module.pdf

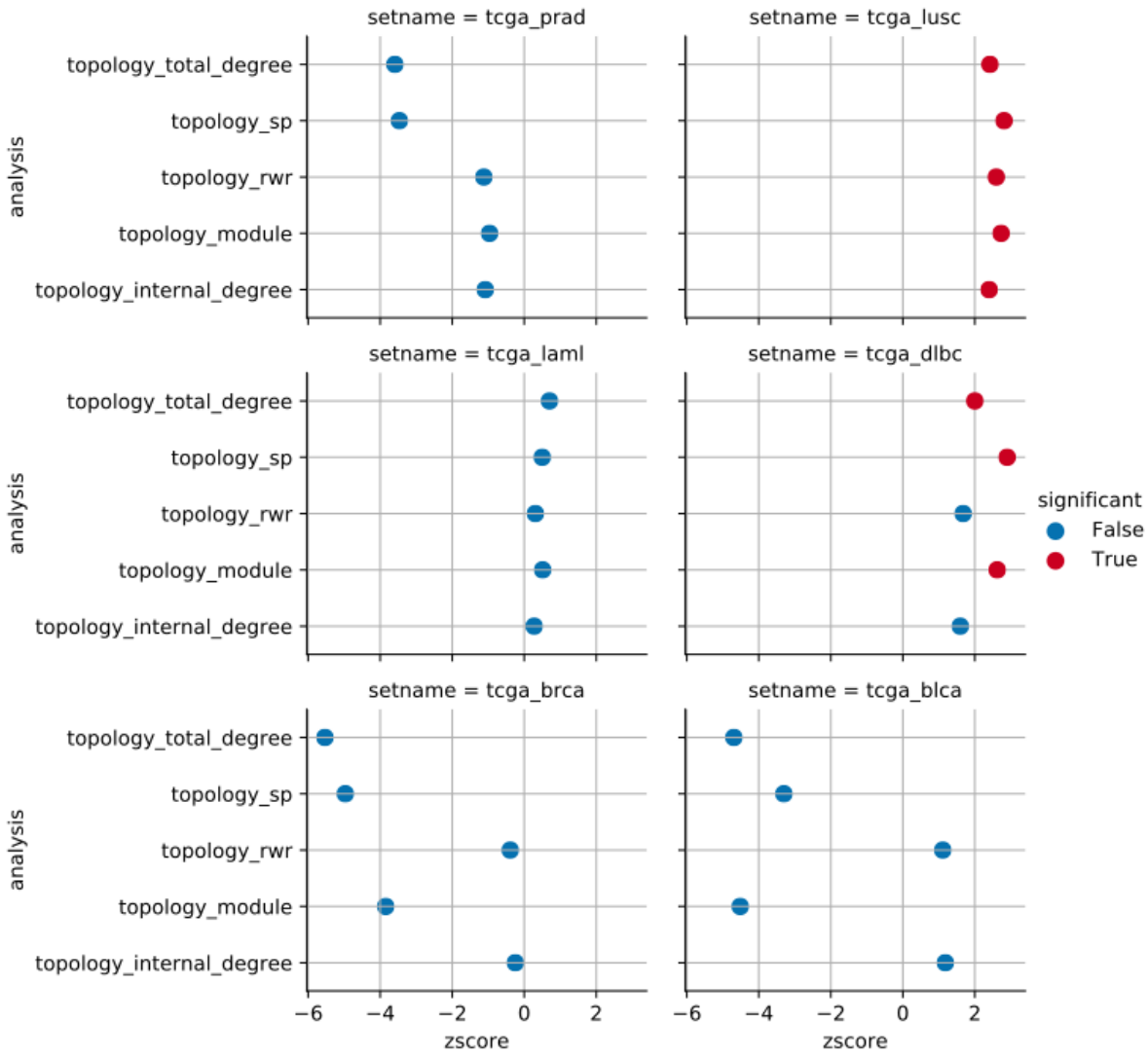


GNT summary

paint_summary_gnt plots a pointplot with a general summary for the GNT.

For example, given multiple GNT test results saved as *table_topology_<method>.csv* we could visualise all results as follows.

```
$ pygna paint-summary-gnt figures/summary_gnt.pdf table_topology_*
```




```
usage: pygna paint-summary-gnt [-h] [-s SETNAME] [-t THRESHOLD] [-c COLUMN_FILTER] [--
↪larger] [--less-tests LESS_TESTS] output-figure [input_tables [input_tables ...]]

positional arguments:
  output-figure         output figure filename
  input_tables          -

optional arguments:
  -h, --help            show this help message and exit
  -s SETNAME, --setname SETNAME
                        name of the dataset (default: -)
  -t THRESHOLD, --threshold THRESHOLD
                        Value to threshold the colors (default: 0.05)
  -c COLUMN_FILTER, --column-filter COLUMN_FILTER
                        column where the threshold is applied (default: 'empirical_
↪pvalue')
  --larger              if True the threshold is set as lower limit (default: False)
  --less-tests LESS_TESTS
                        comma separated string of the tests that are significant if␣
↪lower than expected, otherwise pass empty string (default: 'topology_sp')
```

(continues on next page)

(continued from previous page)

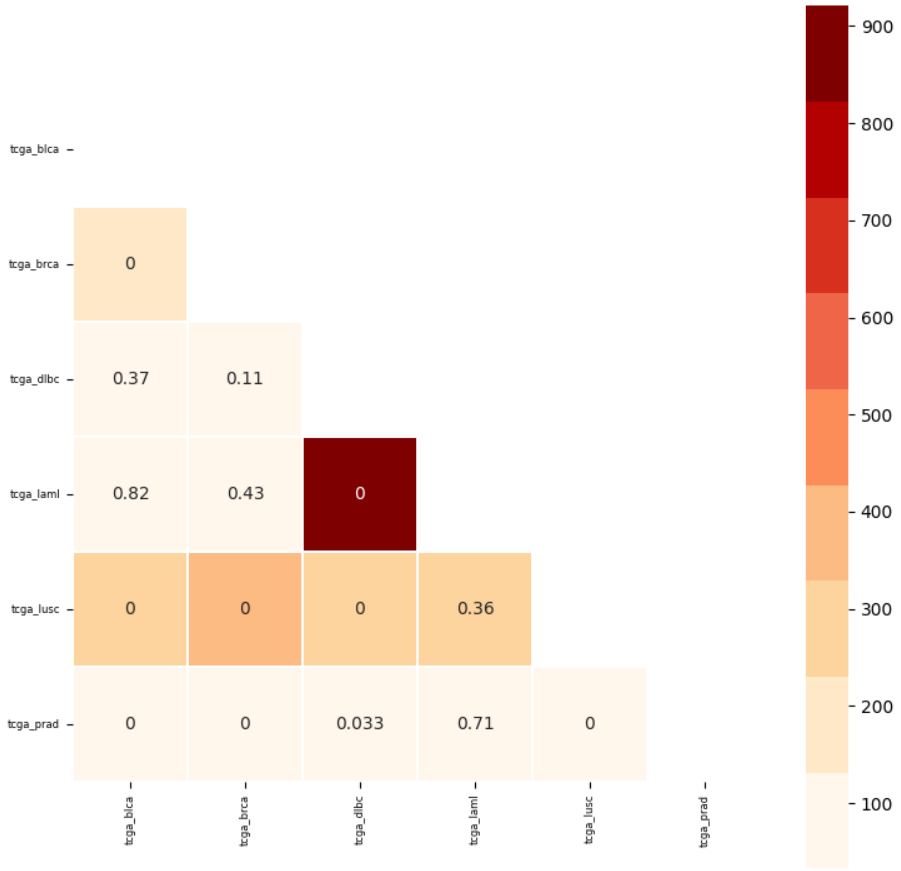
3.8.2 GNA plots

GNA heatmap

paint-comparison-matrix plots the results of of a GNA test. Pass the results table generated by one of the functions and the output figure file (png or pdf). With RWR you can specify whether the test is a RWR association, in this case a different palette and limits are sets. Specify if the results are obtained using association with only one genesets (multiple setnames in the same file). Pass the annotate flag to have the pvalue annotation on the plot

```
usage: pygna paint-comparison-matrix [-h] [-r] [-s] [-a] table-filename output-file

positional arguments:
  table-filename        pygna comparison output
  output-file           output figure file, specify png or pdf file

optional arguments:
  -h, --help            show this help message and exit
  -r, --rwr             use RWR is the table comes from a RWR analysis (default:␣
↪False)
  -s, --single-geneset  use true if the comparison has been done for a single file␣
↪(default: False)
  -a, --annotate        set true if uou want to print the pvalue inside the cell␣
↪(default: False)
```

```
$ pygna paint-comparison-matrix  table_comparison_rwr.csv figures/comparison_rwr.pdf -
↪r -s -a
```



GNA association volcano

paint-volcano-plot plots the results of of a GNA test of association of a single geneset against multiple pathways. Pass the results table generated by one of the functions and the output figure file (png or pdf). From the results table, a multiple testing correction is applied and the results are those plotted.

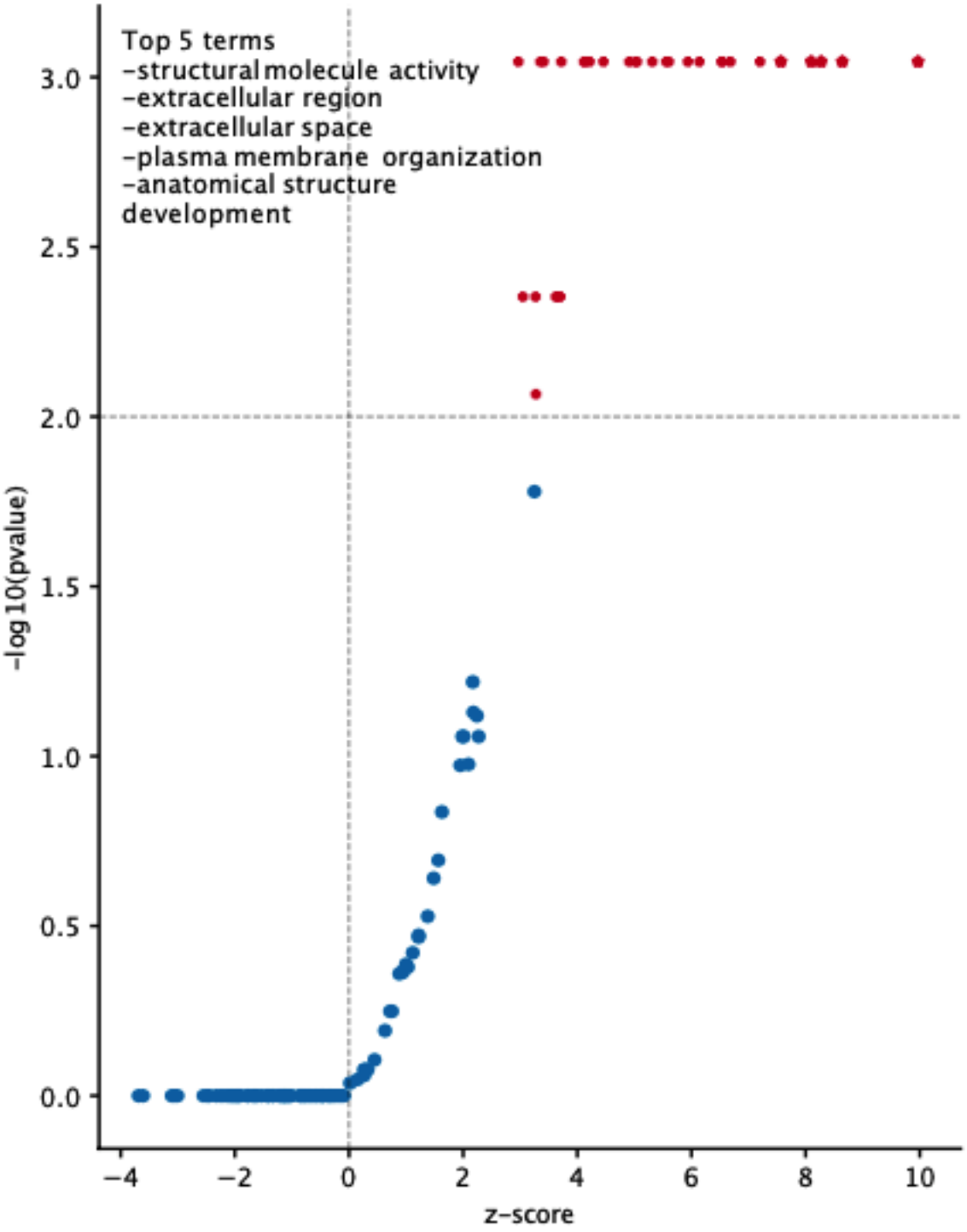
The defined threshold are for x: zscore and $y : -log10(pvalue)$

```
usage: pygna paint-volcano-plot [-h] [-r] [-i ID_COL] [--threshold-x THRESHOLD_X] [--
↪threshold-y THRESHOLD_Y] [-a] table-filename output-file

positional arguments:
  table-filename        pygna comparison output
```

(continues on next page)

(continued from previous page)

```
  output-file           output figure file, specify png or pdf file

optional arguments:
  -h, --help            show this help message and exit
  -r, --rwr             use RWR is the table comes from a RWR analysis (default:␣
↪False)
  -i ID_COL, --id-col ID_COL
                        'setname_B'
  --threshold-x THRESHOLD_X
                        0
  --threshold-y THRESHOLD_Y
                        2
  -a, --annotate        False
```


Top 5 terms
-structural molecule activity
-extracellular region
-extracellular space
-plasma membrane organization
-anatomical structure development
3.0
2.5
2.0
1.5
1.0
0.5
0.0
-log10(pvalue)
−4
−2
0
2
4
6
8
10
z-score

CHAPTER 4

Benchmarking

We have created a framework for testing the performance of Geneset Network analysis: it allows the experimenter to create simulated networks specifying many parameters. The model we use is the Stochastic Block Model and Degree Model that provides a way to create the network and identify clusters into it.

4.1 Stochastic Block Model

We provide both functions to generate stochastic block model simualated data like below. On the left is the model used for GNT benchmarking and on the right the one for GNA benchmarking.

For a quick primer on SBM, you can check the page below:

4.1.1 Stochastic Block Model Simulations

This notebook is used to illustrate how the stochastic block model simulations have been prepared for PyGNA.

In order to have an understandable figure of the different models, we are going to use different values for the parameters compared to the actual ones. We noticed that on a 1000 nodes network showing different clusters in a 2d representation is very diffult.

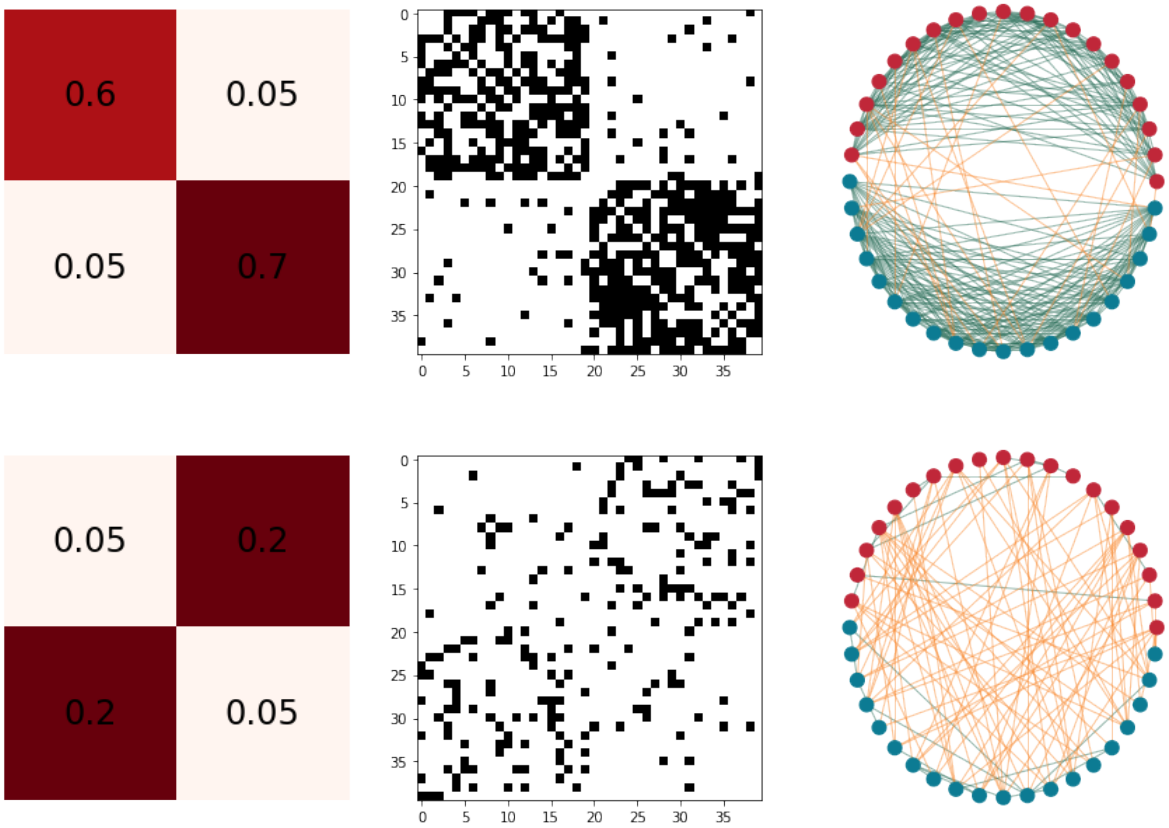
Usual representation of a stochastic block model

First of all, we show here what the different parameters of a block model represent. In a stochastic block model, the connectivity of the network is defined by the probability of connection of the different nodes. Each node belongs to a block, one parameter, p_{ii} controls the probability of connection within the i-th cluster while the p_{ij} parameters control the probability of connection between block i and j.

Here are some examples of a SBM with 2 blocks, each of them with 20 nodes.

For each of the configurations we show: the SBM matrix, the adjacency matrix, and the graph.

We are going to show a strongly assortative network, a strongly disassortative network, and an intermediate case.

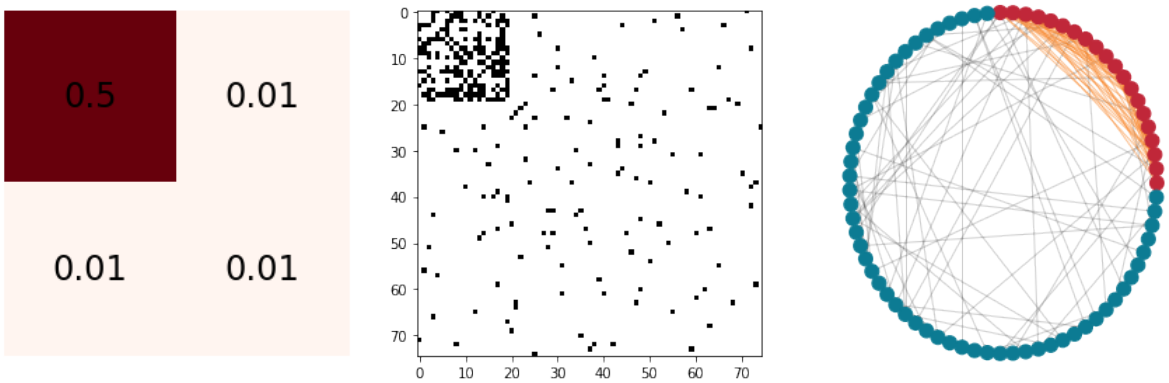


4.1.2 PyGNA models

In order to simulate a realistic network, we use the stochastic block model to generate both the whole network and the clusters of intereset.

To do so, we have to define a larger block, whose probability of connection represents the general connectivity of the network, and a smaller block which is going to be the cluster of interest.

All the simulations in PyGNA are done by parametrising the number of nodes in each block and the SMB matrix.



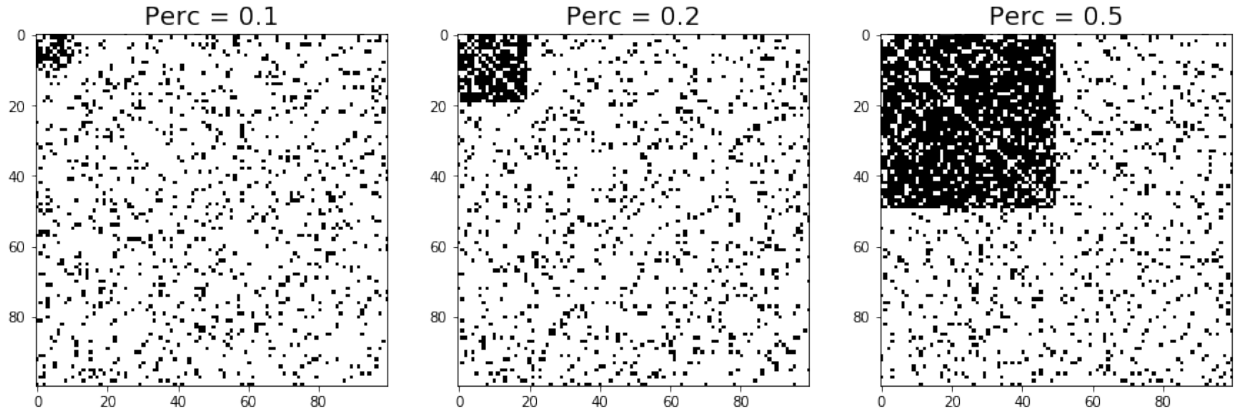
4.1.3 Two clusters SBM

In the simplest model, we parametrise two blocks by varying the number of nodes in each cluster and the SBM matrix.

With the parameter $perc$ we specify the percentage of the total number of nodes that is assigned to the cluster of interest. Instead, the SBM matrix is parametrised by varying p_00 and $p_{01} = p_{10} = p_{11=1-p-00}$.

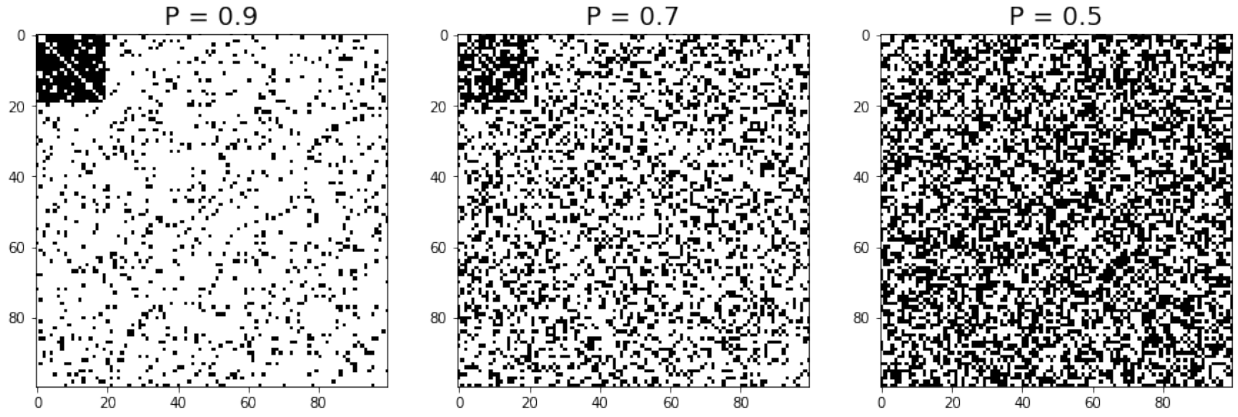
Percentage of nodes

Here is how the adjacency matrix varies with different percentages of nodes in the blocks.

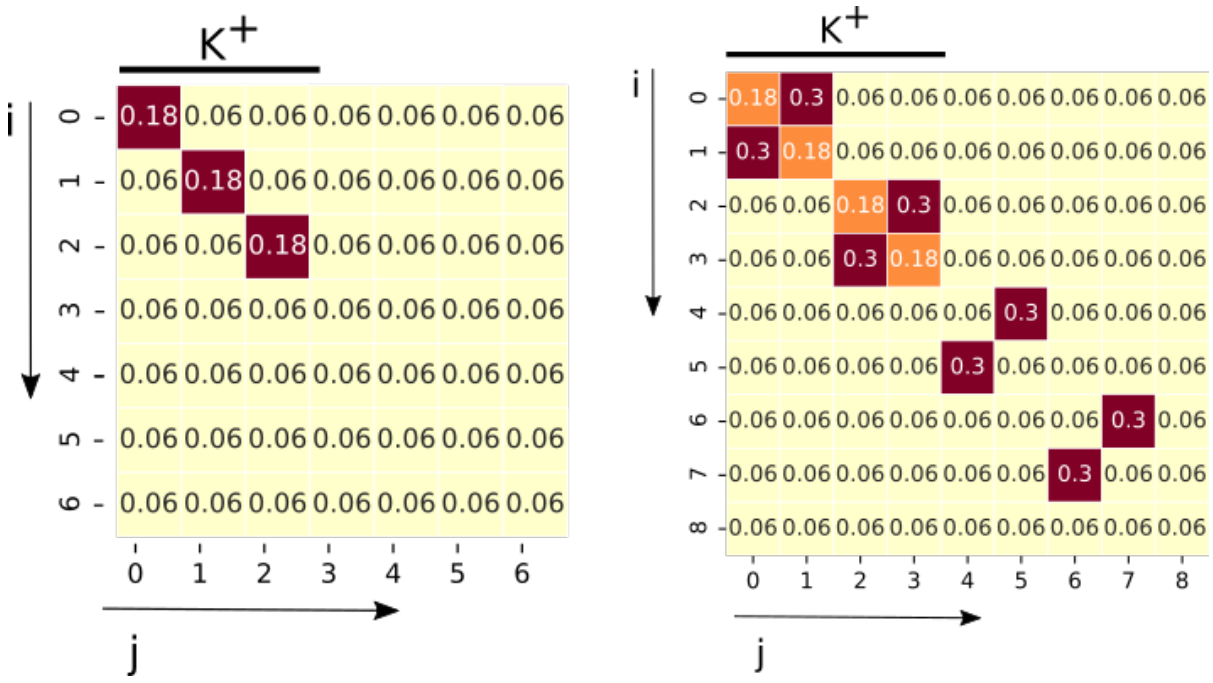


Connection probabilities

Here we show how the adjacency matrix varies for different probabilities of connection.



For each simulation we return both the whole network and a gmt file with the blocks.



4.1.4 GNT Benchmarking

```
usage: pygna generate-gnt-sbm [-h] [-N N] [-b BLOCK_SIZE] [--d D]
                              [-f FOLD_CHANGE] [--descriptor DESCRIPTOR]
                              output-tsv output-gmt

    This function generates 3 blocks with d*fold_change probability
    and other 3 blocks with d probability.
    Make sure that 6*cluster_size < N


positional arguments:
output-tsv            output network filename
output-gmt            output geneset filename, this contains only the blocks

optional arguments:
-h, --help            show this help message and exit
-N N, --N N           number of nodes in the network (default: 1000)
-b BLOCK_SIZE, --block-size BLOCK_SIZE
                        size of the first 6 blocks (default: 50)
--d D                 baseline probability of connection, p0 in the paper
                        (default: 0.06)
-f FOLD_CHANGE, --fold-change FOLD_CHANGE
                        positive within-block scaling factor for the
                        probability of connection, Mii = fold_change * d
                        (alpha parameter in the paper) (default: 2.0)
--descriptor DESCRIPTOR
                        descriptor for the gmt file (default: 'mixed_sbm')
```

4.1.5 GNA Benchmarking

```
usage: pygna generate-gna-sbm [-h] [--output-gmt2 OUTPUT_GMT2] [-N N]
                              [-b BLOCK_SIZE] [--d D] [--fc-cis FC_CIS]
                              [--fc-trans FC_TRANS] [-p PI]
                              [--descriptor DESCRIPTOR] [-s SBM_MATRIX_FIGURE]
                              output-tsv output-gmt

    This function generates benchmark network and geneset to test
    the crosstalk between two blocks.

    This function generates 4 blocks with d*fold change probability
    and other 4 blocks with d probability.
    The crosstalk is set both between the the first 4 blocks and the others.

    Make sure that 8*cluster_size < N



positional arguments:
output-tsv            output_network
output-gmt            output geneset filename, this contains only the blocks

optional arguments:
-h, --help            show this help message and exit
--output-gmt2 OUTPUT_GMT2
                        mixture output geneset filename, this contains the
                        mixture blocks (default: -)
-N N, --N N           number of nodes in the network (default: 1000)
-b BLOCK_SIZE, --block-size BLOCK_SIZE
                        size of the first 8 blocks (default: 50)
--d D                 baseline probability of connection, p0 in the paper
                        (default: 0.06)
--fc-cis FC_CIS       positive within-block scaling factor for the
                        probability of connection, Mii = fc_cis * d (alpha
                        parameter in the paper) (default: 2.0)
--fc-trans FC_TRANS   positive between-block scaling factor for the
                        probability of connection, (beta parameter in the
                        paper) (default: 0.5)
-p PI, --pi PI        percentage of block-i nodes for the genesets made of
                        block-i and block-j. Use symmetrical values (5,95),use
                        string comma separated (default:
                        '4,6,10,12,88,90,94,96')
--descriptor DESCRIPTOR
                        'crosstalk_sbm'
-s SBM_MATRIX_FIGURE, --sbm-matrix-figure SBM_MATRIX_FIGURE
                        shows the blockmodel matrix (default: -)
```

4.2 High Degree Nodes simulations

4.2.1 Generate the network and geneset file

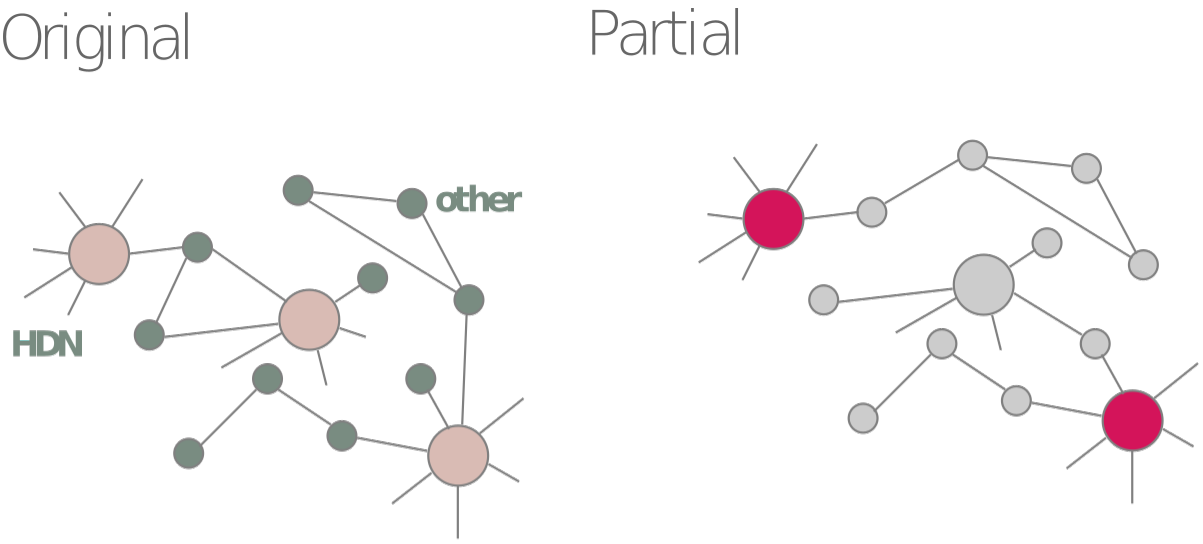
The high degree nodes (HDN) model generates networks with a controllable number of hubs, HDNs, whose probability of connection with another node is higher than the baseline probability assigned to any other node in the network.

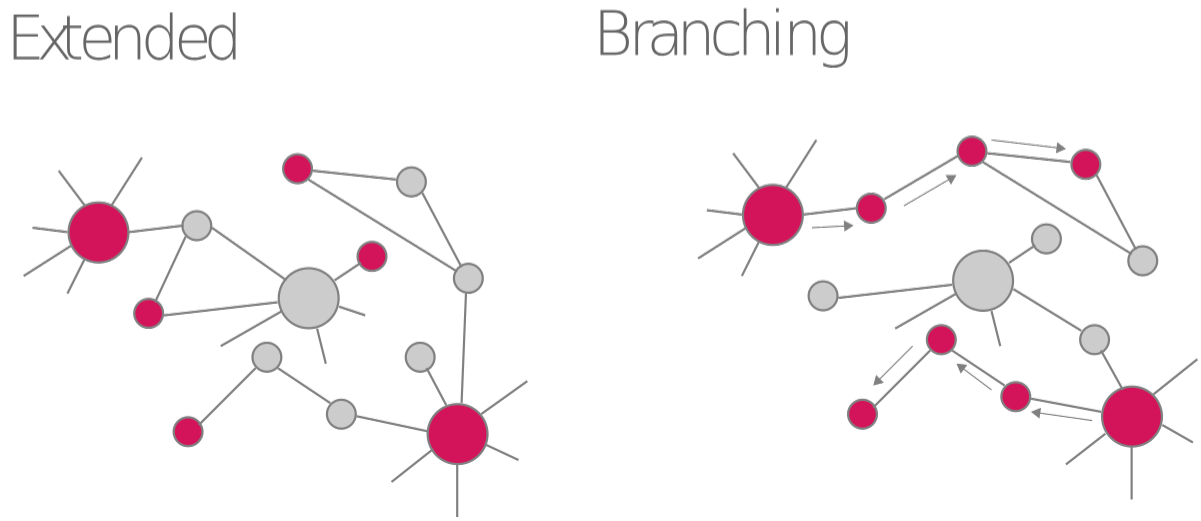
```
usage: pygna generate-hdn-network [-h] [--n-nodes N_NODES]
                                  [--network-prob NETWORK_PROB]
                                  [--hdn-probability HDN_PROBABILITY]
                                  [--hdn-percentage HDN_PERCENTAGE]
                                  [--number-of-simulations NUMBER_OF_SIMULATIONS]
                                  output-folder prefix

    This function generates a simulated network using the VIP model



positional arguments:
output-folder         the output folder path
prefix                the prefix of the file to be saved

optional arguments:
-h, --help            show this help message and exit
--n-nodes N_NODES     the number of nodes in the network (default: 1000)
--network-prob NETWORK_PROB
                        probability of connection in the network (default:
                        0.005)
--hdn-probability HDN_PROBABILITY
                        probability of connection of the HDNs (default: 0.3)
--hdn-percentage HDN_PERCENTAGE
                        percentage of HDNs (default: 0.05)
--number-of-simulations NUMBER_OF_SIMULATIONS
```

4.2.2 Add genesets

Given the generated network and node list of HDNs, we can then generate novel genesets made of mixtures of the two.

We show the idea of how they are generated below. First is the original network with a number of HDNs, then the partial, extended, and branching genesets.





Add extended genesets

```
usage: pygna hdn-add-extended [-h] [--hdn-set HDN_SET] [-g GENERAL_SET]
                              [--reps REPS] [-p PERCENTAGE_EXTENDED_VIPS]
                              [--ratio-others RATIO_OTHERS]
                              [-o OUTPUT_GENESET_FILE]
                              input-geneset-file

    Creates new genesets from the vip list, number of genesets and portion of genes
    can be specified by input. The final new geneset is going to be formed by:
    percentage ev * HDN_total + ratio* percentage ev*vips total.



positional arguments:
input-geneset-file    input geneset containing the sets to be merged
```

(continues on next page)

(continued from previous page)

```
optional arguments:
-h, --help            show this help message and exit
--hdn-set HDN_SET     setname of the HDNs in the geneset (default:
                        'cluster_1')
-g GENERAL_SET, --general-set GENERAL_SET
                        other setname in the geneset (default: 'cluster_0')
--reps REPS           number of new genesets made of part of HDNs (default:
                        3)
-p PERCENTAGE_EXTENDED_VIPS, --percentage-extended-vips PERCENTAGE_EXTENDED_VIPS
                        percentage of HDNs in the new genesett (default:
                        '[0.2]')
--ratio-others RATIO_OTHERS
                        ratio of genes to add to HDNs (default: '[2,2.5,3,4]')
-o OUTPUT_GENESET_FILE, --output-geneset-file OUTPUT_GENESET_FILE
                        if no output gmt filename is passed, the data is added
                        to the input file (default: -)
```

Add partial genesets

```
usage: pygna hdn-add-partial [-h] [--hdn-set HDN_SET] [-g GENERAL_SET]
                             [-r REPS] [-p PERCENTAGE_PARTIAL_VIPS]
                             [-o OUTPUT_GENESET_FILE]
                             input-geneset-file

    Creates new genesets from the vip list, number of genesets and portion of
    genes can be specified by input.


positional arguments:
input-geneset-file    input geneset containing the sets to be merged

optional arguments:
-h, --help            show this help message and exit
--hdn-set HDN_SET     setname of the HDNs in the geneset (default:
                        'cluster_1')
-g GENERAL_SET, --general-set GENERAL_SET
                        other setname in the geneset (default: 'cluster_0')
-r REPS, --reps REPS  number of new genesets made of part of vips (default:
                        3)
-p PERCENTAGE_PARTIAL_VIPS, --percentage-partial-vips PERCENTAGE_PARTIAL_VIPS
                        percentage of HDNs in the new geneset (default:
                        '0.1,0.2,0.5')
-o OUTPUT_GENESET_FILE, --output-geneset-file OUTPUT_GENESET_FILE
                        if no output gmt filename is passed, the data is added
                        to the input file (default: -)
```

Add branching genesets

```
usage: pygna hdn-add-branching [-h] [--hdn-set HDN_SET] [-g GENERAL_SET]
                               [--number-of-hdns NUMBER_OF_HDNS]
                               [--number-of-reps NUMBER_OF_REPS]
                               [--output-geneset-file OUTPUT_GENESET_FILE]
```

(continues on next page)

(continued from previous page)

```
                            [--output-graph-file OUTPUT_GRAPH_FILE]
                            input-geneset-file network-file

    Creates new genesets from the vip list, new genesets are created adding 1 step
    nodes to vips. The new genes are created as branches.


positional arguments:
input-geneset-file    input geneset containing the sets to be merged
network-file          network filename

optional arguments:
-h, --help            show this help message and exit
--hdn-set HDN_SET     setname of the HDNs in the geneset (default:
                        'cluster_1')
-g GENERAL_SET, --general-set GENERAL_SET
                        other setname in the geneset (default: 'cluster_0')
--number-of-hdns NUMBER_OF_HDNS
                        number of seed HDNs to start adding nodes (default: 3)
--number-of-reps NUMBER_OF_REPS
                        number of new genesets created for each condition
                        (default: 3)
--output-geneset-file OUTPUT_GENESET_FILE
                        if no output gmt filename is passed, the data is added
                        to the input file (default: -)
--output-graph-file OUTPUT_GRAPH_FILE
                        if graphml file is passed the network with labels is
                        written (default: -)
```

4.2.3 General model (old)

This function is still working, however now networkx provides a function to generate SBM networks. We will then change this function to use directly networkx for the generation.

All the parameters can be specified in a yaml file that is passed as input. As output, we obtain a file with the network and a .gmt file where the nodes have been grouped by the respective cluster they are in.

Example yaml

```yaml
BlockModel :
  n_nodes: 100
  matrix: [[0.8,0.1,0.1],[0.1,0.8,0.1],[0.1,0.1,0.8]]
  nodes: ""
  nodes_in_cluster: None
Simulations:
  n_simulated: 1
  output_folder: "/home/viola/Desktop/geneset-network-analysis/processed_data/
↪simulated_data/"
  suffix: attempt1
```

The **Simulations** parameters are used to specify the number and name of the output:

- `n_simulated` we can specify the number of networks we want to generate with the same settings
- `output_folder` specifies the folder where the output files are going to be saved

- `suffix` is the suffix used for the output.

For each simulated datasets there are two output files:

- suffix_network_$simulation.tsv : the network file
- suffix_genes_$simulation.gmt : the gene list grouped by cluster

The **BlockModel** parameters are those used to generate the SBM:

- `n_nodes` number of nodes of the network
- `matrix` SBM matrix
- `nodes: ""` names of the nodes, if not specified N$number
- `nodes_in_cluster: None`: Number of nodes thatwe want to assign to each cluster

Example simulated dataset generation and Analysis

Generation of the network and genesets

```
$ pygna generate-simulated-network ../simulationBM.yaml
```

CHAPTER 5

API

The following sections describe the different PyGNA classes and functions available.

5.1 Block Model

class pygna.block_model.**BlockModel**(*object*)

create_graph() → None
Create a graph from the parameters passed in the constructor of the class

Example

```
>>> bm = BlockModel(np.array(config["BlockModel"]["matrix"]), n_nodes=config[
↪"BlockModel"]["n_nodes"], nodes_percentage=config["BlockModel"]["nodes_
↪percentage"])
>>> bm.create_graph()
```

plot_graph(*output_folder: str*) → None
Plot the block model graph

Parameters **output_folder** – the folder where to save the result

Example

```
>>> p = 0.5
>>> n_nodes = 1000
>>> matrix = np.array([[1, 2], [3, 4]])
>>> bm = BlockModel(matrix, n_nodes=n_nodes, nodes_percentage=[p, 1 - p])
>>> bm.plot_graph("block_model_path.pdf")
```

set_bm(*block_model_matrix: pandas.core.frame.DataFrame*) → None
Change block model matrix used in the class

Parameters **block_model_matrix** – the block model matrix

Example

```
>>> p = 0.5
>>> n_nodes = 1000
>>> matrix = np.array([[1, 2], [3, 4]])
>>> bm = BlockModel(matrix, n_nodes=n_nodes, nodes_percentage=[p, 1 - p])
>>> bmm = pd.DataFrame(mydata_matrix)
>>> bm.set_bm(bmm)
```

set_nodes(*nodes_names: list*) → None
Set the nodes name of the block model

Parameters **nodes_names** – the names list

Example

```
>>> p = 0.5
>>> n_nodes = 1000
>>> matrix = np.array([[1, 2], [3, 4]])
>>> bm = BlockModel(matrix, n_nodes=n_nodes, nodes_percentage=[p, 1 - p])
>>> nodes = list("A", "B", "C")
>>> bm.set_nodes(nodes)
```

set_nodes_in_block(*nodes_in_block: int*) → None
Set the nodes number in the block model

Parameters **nodes_in_block** – the number of nodes in the block list

Example

```
>>> p = 0.5
>>> n_nodes = 1000
>>> matrix = np.array([[1, 2], [3, 4]])
>>> bm = BlockModel(matrix, n_nodes=n_nodes, nodes_percentage=[p, 1 - p])
>>> bm.set_nodes_in_block(1000)
```

set_nodes_in_block_percentage(*nodes_percentage: list*) → None
Pass the percentage of nodes in each block as a list, for example [0.5, 0.5]

Parameters **nodes_percentage** – percentage of the nodes

Example

```
>>> p = 0.5
>>> n_nodes = 1000
>>> matrix = np.array([[1, 2], [3, 4]])
>>> bm = BlockModel(matrix, n_nodes=n_nodes, nodes_percentage=[p, 1 - p])
>>> bm.set_nodes_in_block_percentage([0.5, 0.5])
```

write_cluster_genelist(*output_file: str*) → None
Save the gene list to a GMT file

Parameters **output_file** – the output path where to save the results

Example

```
>>> p = 0.5
>>> n_nodes = 1000
>>> matrix = np.array([[1, 2], [3, 4]])
>>> bm = BlockModel(matrix, n_nodes=n_nodes, nodes_percentage=[p, 1 - p])
>>> bm.write_cluster_genelist("genes.gmt")
```

write_network(*output_file: str*) → None
Save the network on a given file

Parameters **output_file** – the output path where to save the results

Example

```
>>> p = 0.5
>>> n_nodes = 1000
>>> matrix = np.array([[1, 2], [3, 4]])
>>> bm = BlockModel(matrix, n_nodes=n_nodes, nodes_percentage=[p, 1 - p])
>>> bm.write_network("network.tsv")
```

pygna.block_model.**generate_graph_from_sm**(*n_nodes: int*, *block_model: pandas.core.frame.DataFrame*, *nodes_in_block: list = False*, *node_names: list = None*, *nodes_percentage: list = None*) → networkx.classes.graph.Graph
This function creates a graph with n_nodes number of vertices and a matrix block_model that describes the intra e inter-block connectivity. The nodes_in_block is parameter, list, to control the number of nodes in each cluster

Parameters

- **n_nodes** – the number of nodes in the block model
- **block_model** – the block model to elaborate
- **nodes_in_block** – the list of nodes in the block model
- **node_names** – the list of names in the block model
- **nodes_percentage** – the percentage of nodes to use for the calculations, passed through a list for example [0.5, 0.5]

Example

```
>>> bm = pd.DataFrame(mydata_matrix)
>>> nodes = list("A","B","C")
>>> graph = generate_graph_from_sm(n_nodes, bm, nodes_in_block, nodes, nodes_
↪percentage)
```

pygna.block_model.**plot_bm_graph**(*graph: networkx.classes.graph.Graph*, *block_model: pandas.core.frame.DataFrame*, *output_folder: str = None*) → None

Save the graph on a file

Parameters

- **graph** – the graph with name of the nodes
- **block_model** – the block model
- **output_folder** – the folder where to save the file

Example

```
>>> bm = pd.DataFrame(mydata_matrix)
>>> graph = nx.complete_graph(100)
>>> plot_bm_graph(graph, bm, output_folder="./results/")
```

pygna.block_model.**generate_sbm_network**(*input_file: yaml configuration file*) → None

This function generates a simulated network, using the block model matrix given as input and saves both the network and the cluster nodes. All parameters must be specified in a yaml file. This function allows to create network and geneset for any type of SBM

5.2 Converters

class pygna.converters.**Converters**

This class is wrap static methods that can be used to convert the data from a format to another. Please refer to each class method for the specific function

classmethod convert_e2s(*geneset: pandas.core.frame.DataFrame*, *tsv_data: pandas.core.frame.DataFrame*, *entrez_col: str = 'NCBI Gene ID'*, *symbol_col: str = 'Approved symbol'*) → list

Method to convert the entrez genes to symbols

Parameters

- **tsv_data** – the dataframe to work on
- **symbol_col** – the column containing the symbols
- **entrez_col** – the column containing the entrez ID
- **geneset** – column containing the entrez to convert

Returns list containing the string names

Example

```
>>> gmt_data = rc.ReadGmt(".gmt", True).get_data()
>>> converted = []
>>> for k, d in gmt_data.items():
>>>     converted[k] = Converters.convert_e2s(d["genes"], tsv_data,entrez_col,
↪ symbol_col)
```

classmethod convert_s2e(*geneset: pandas.core.frame.DataFrame*, *tsv_data: pandas.core.frame.DataFrame*, *entrez_col: str = 'NCBI Gene ID'*, *symbol_col: str = 'Approved symbol'*) → list

Method to convert the genes symbols to entrez id.

Parameters

- **tsv_data** – the dataframe to work on
- **symbol_col** – the column containing the symbols
- **entrez_col** – the column containing the entrez ID
- **geneset** – column containing the strings to convert

Returns list containing the entrez names

Example

```
>>> gmt_data = rc.ReadGmt(".gmt", True).get_data()
>>> converted = []
>>> for k, d in gmt_data.items():
>>>     converted[k] = gmt_data[k]["genes"] = Converters.convert_s2e(d["genes
↪"], tsv_data, entrez_col, symbol_col)
```

class pygna.converters.**CsvToCsvEnriched**(*csv_file: pandas.core.frame.DataFrame*, *conversion: str*, *original_name_col: str*, *new_name_col: str*, *geneset: str*, *entrez_col: str*, *symbol_col: str*, *converter_map_filename: str = 'entrez_name.tsv'*, *output_file: str = None*)

Class that is used to add a column with the entrezID or Symbols to a CSV file

get_data() → pandas.core.frame.DataFrame

Return the conversion result

Returns dataframe with the e2s or s2e added as column

class pygna.converters.**CsvToGmt**(*input_file: str*, *setname: str*, *filter_column: str*, *alternative: str*, *threshold: float*, *output_gmt: str = None*, *output_csv: str = None*, *name_column: str = 'Unnamed: 0'*, *descriptor: str = None*)

This Class converts a csv file to a GMT allowing to filter the elements using the values of one of the columns. The user can specify the column used to retrieve the name of the objects and the filter condition. The output can be either a GMT with the names of the genes that pass the filter or a csv with the whole filtered table, otherwise both can be created.

class pygna.converters.**GmtToGmtEnriched**(*gmt_file: str*, *output_gmt_file: str*, *conversion: str*, *entrez_col: str*, *symbol_col: str*, *converter_map_filename: str = 'entrez_name.tsv'*)

This Class is used to convert a GMT file, adding information about the Entrez ID or the symbol

class pygna.converters.**GroupGmt**(*input_table: str*, *output_gmt: str*, *name_col: str = 'Gene'*, *group_col: str = 'Cancer'*, *descriptor: str = 'cancer_genes'*)

This function generates a GMT file of multiple setnames. From the table file, it groups the names in the group_col (the column you want to use to group them) and prints the genes in the name_col. Set the descriptor according to your needs

5.3 Degree Model

class pygna.degree_model.**DegreeModel**(*network_prob: float = 0.5, vip_prob: float = 1, n_nodes: int = 10, vip_percentage: float = 0.1*)

This class is used to calculate a degree model

create_graph() → None

Create a graph from the nodes

Example

```
>>> dm = DegreeModel()
>>> dm.create_graph()
```

set_nodes(*nodes_names: list*) → None

Set the name of the nodes and calculate the lenght of the list

Parameters **nodes_names** – the list with the nodes name

Example

```
>>> nodes = list("A", "B", "C")
>>> dm = DegreeModel()
>>> dm.set_nodes(nodes)
```

write_genelist(*output_file: str*) → None

Write the GMT gene list on a specific file

Parameters **output_file** – the file path where to save the gene list

Example

```
>>> dm = DegreeModel()
>>> dm.write_genelist("myoutput.gmt")
```

write_network(*output_file: str*) → None

Write on file the network as an edge list

Parameters **output_file** – the file path where to save the network

Example

```
>>> dm = DegreeModel()
>>> dm.write_network("myoutputfile.tsv")
```

pygna.degree_model.**generate_graph_vip**(*n_nodes: int, n_vip: int, network_prob: float = 0.5, vip_prob: float = 1, node_names: list = None*) → networkx.classes.graph.Graph

This function creates a graph with n_nodes number of vertices and a matrix block_model that describes the intra e inter-block connectivity. The nodes_in_block is parameter, list, to control the number of nodes in each cluster

Parameters

- **n_nodes** – number of nodes in the network
- **n_vip** – number of VIP to create
- **network_prob** – probability of connection in the network
- **vip_prob** – probability of connection of the vip
- **node_names** – list of nodes for the network

Example

```
>>> n_nodes = 100
>>> n_vip = 10
>>> network_prob = 0.5
>>> vip_prob = 1
>>> nodes = ["A", "B", "C"]
>>> graph = generate_graph_vip(n_nodes, n_vip, network_prob=network_prob, vip_
↪prob=vip_prob, node_names=nodes)
```

pygna.degree_model.**plot_vip_graph**(*graph: networkx.classes.graph.Graph*, *output_folder: str = None*) → None

Plot the VIP graph in the specific folder

Parameters

- **graph** – the graph to plot
- **output_folder** – the folder path where to save the file

Example

```
>>> g = nx.complete_graph(100)
>>> plot_vip_graph(g, "./")
```

pygna.degree_model.**plot_adjacency**(*graph: networkx.classes.graph.Graph*, *output_folder: str*, *prefix: str*) → None

Plot the adjacency matrix on file

Parameters

- **graph** – the graph to plot
- **output_folder** – the folder where to save the file
- **prefix** – the prefix to give to the file

Example

```
>>> g = nx.complete_graph(100)
>>> plot_adjacency(g, "./","adj_matrix_")
```

5.4 Diagnostic

pygna.diagnostic.**plot_degree**(*degree_object: networkx.classes.graph.Graph*, *output_file: str*)

Diagnosis tool for the degree object. It takes a graph and plot its statistics

Parameters

- **degree_object** – the graph to plot
- **output_file** – the path to save the file

Example

```
>>> graph = nx.complete_graph(100)
>>> plot_degree(nx.degree(graph), "graph_degree")
```

pygna.diagnostic.**plot_connected_components**(*c_components: networkx.algorithms.components.connected.connected_components*, *output_file: str*) → None

Diagnosis tool for the connected components object. Creates the histogram of the components length, to analyse the relationship between the lcc and the other c_components, and prints some overall stats about the connected components

Parameters

- **c_components** – the list of the connected components
- **output_file** – the path to save the file

Example

```
>>> from pygna import diagnostic
>>> import pygna.reading_class as rc
>>> network = rc.ReadTsv("network_file.tsv").get_network()
>>> geneset = rc.ReadGmt("geneset_input_file.gmt").get_geneset("brca")
>>> c_components_figure_file = "components_plot.pdf"
>>> for setname, item in geneset.items():
...     graph = nx.subgraph(network, item)
...     diagnostic.plot_connected_components(nx.connected_components(graph), c_
↪components_figure_file)
```

pygna.diagnostic.**plot_diffusion_matrix**(*nodes: list*, *matrix: numpy.ndarray*, *filename: str*, *show_labels: bool = False*) → None

Diagnosis tool for a diffusion matrix. It shows the weighted adjacency matrix that is the output of a build process

Parameters

- **nodes** – the network nodes
- **matrix** – the diffusion matrix
- **filename** – the path to save the file
- **show_labels** – if labels should be plotted

Example

```
>>> nodes = ["A", "B", "C"]
>>> matrix = np.random.rand(3,2)
>>> plot_diffusion_matrix(nodes, matrix, "diff_matrix.pdf")
```

pygna.diagnostic.**plot_null_distribution**(*null_distribution: list*, *observed: list*, *output_file: str*, *setname: str*, *alternative: str = 'greater'*) → None

Saves the density plot of the null distribution and pinpoints the observed value. Please refer to the CLI for the usage of this function

Parameters

- **null_distribution** – the list with the values from the null distribution
- **observed** – list of the observed genes
- **output_file** – the path to save the file
- **setname** – the name of the gene set
- **alternative** – use "greater" if you want to take the genes with greater than the observed value

Example

```
>>> from pygna import diagnostic
>>> import pygna.statistical_test as st
>>> import pygna.reading_class as rc
>>> network = rc.ReadTsv("network_file.tsv").get_network()
>>> st_test = st.StatisticalTest(st.geneset_total_degree_statistic, network)
>>> geneset = rc.ReadGmt("geneset_file.gmt").get_geneset("brca")
>>> for setname, item in geneset.items():
...     observed, pvalue, null_d, n_mapped, n_geneset = st_test.empirical_
↪pvalue(item, max_iter=500, alternative="greater", cores=10)
...     diagnostic.plot_null_distribution(null_d, observed, "./results/" +␣
↪setname +'_total_degree_null_distribution.pdf', setname=setname)
```

5.5 Elaborators

class pygna.elaborators.**TableElaboration**

This class contains static methods to clean and filter specific columns of a table

static clean_table(*table: pandas.core.frame.DataFrame*, *stat_col: str = 'stat'*) → pandas.core.frame.DataFrame

This function clean the table from the N/A values

Parameters

- **table** – dataframerepresenting the table to be cleaned
- **stat_col** – the column to be cleaned

Returns the table cleaned from the N/A values

Example

```
>>> import numpy as np
>>> table = pd.DataFrame(np.random.randint(0,100,size=(100, 1)), columns=list(
↪'mycol'))
>>> table = TableElaboration.clean_table(table, "mycol")
```

static filter_table(*table: pandas.core.frame.DataFrame*, *filter_column: str = 'padj'*, *alternative: str = 'less'*, *threshold: float = 0.01*) → pandas.core.frame.DataFrame

This method filters a table according to a filter rule èassed as input

Parameters

- **table** – The table to be filtered
- **filter_column** – Column with the values to be filtered
- **alternative** – Alternative to use for the filterK with "less" the filter is applied <threshold; otherwise >= threshold
- **threshold** – Threshold for the filter

Returns The table filtered

Example

```
>>> import numpy as np
>>> table = pd.DataFrame(np.random.randint(0,100,size=(100, 1)), columns=list(
↪'pval'))
>>> table = TableElaboration.filter_table(table, filter_column="pval")
```

5.6 Output

class pygna.output.**Output**(*network_filename: str*, *output_table_results_file: str*, *analysis: str*, *geneset_file: str*, *setnames: list*, *geneset_file_B: str = None*, *setnames_B: list = None*)

This class is used to print different data on files

add_GMT_entry(*key: str*, *descriptor: str*, *gene_list: str*) → None

Add a gmt entry in the GMT file

Parameters

- **key** – the key name to store
- **descriptor** – the descriptor of the gene list
- **gene_list** – the gene list to write

Example

```
>>> geneset = rc.ReadGmt("geneset_file.csv").get_geneset("brca")
>>> setnames = [key for key in geneset.keys()]
>>> import pygna.reading_class as rc
>>> network = rc.ReadTsv("network_file.tsv").get_network()
>>> out = Output("networkfile.tsv", "results.csv", "myanalysis", "genset_a.csv
↪", setnames)
>>> for setname, item in geneset.items():
...     item = set(item)
...     module = nx.subgraph(network, item)
...     lcc = sorted(list(nx.connected_components(module)), key=len,␣
↪reverse=True)[0]
...     out.add_GMT_entry("brca", "topology_module", lcc)
```

close_temporary_table() → None
Remove the temporary file

Example

```
>>> setnames = ["A", "B", "C"]
>>> out = Output("networkfile.tsv", "results.csv", "myanalysis", "genset_a.csv
↪", setnames)
>>> out.create_st_table_empirical()
>>> out.close_temporary_table()
```

create_GMT_output(*output_gmt: str*) → None
Write the GMT line on the GMT file

Parameters **output_gmt** – the GMT to print

Example

```
>>> geneset = rc.ReadGmt("geneset_file.csv").get_geneset("brca")
>>> setnames = [key for key in geneset.keys()]
>>> import pygna.reading_class as rc
>>> network = rc.ReadTsv("network_file.tsv").get_network()
>>> out = Output("networkfile.tsv", "results.csv", "myanalysis", "genset_a.csv
↪", setnames)
>>> for setname, item in geneset.items():
...     item = set(item)
...     module = nx.subgraph(network, item)
...     lcc = sorted(list(nx.connected_components(module)), key=len,␣
↪reverse=True)[0]
...     out.add_GMT_entry("brca", "topology_module", lcc)
>>> out.create_GMT_output("output_lcc.gmt")
```

create_comparison_table_empirical() → None
Write the hadings for the comparison table

Example

```
>>> setnames = ["A", "B", "C"]
>>> out = Output("networkfile.tsv", "results.csv", "myanalysis", "genset_a.csv
↪", setnames)
>>> out.create_comparison_table_empirical()
```

create_st_table_empirical() → None
Create the headings of the table that are going to be wrinnte in the csv file

Example

```
>>> setnames = ["A", "B", "C"]
>>> out = Output("networkfile.tsv", "results.csv", "myanalysis", "genset_a.csv
↪", setnames)
>>> out.create_st_table_empirical()
```

set_diffusion_matrix(*diffusion_matrix_file: str*) → None
Set the diffusion matrix file

Parameters **diffusion_matrix_file** – set the diffusion matrix file to use

Example

```
>>> setnames = ["A", "B", "C"]
>>> out = Output("networkfile.tsv", "results.csv", "myanalysis", "genset_a.csv
↪", setnames)
>>> out.set_diffusion_matrix("diffusion_matrix.csv")
```

update_comparison_table_empirical(*setname_A: str*, *setname_B: str*, *n_geneset_A: int*, *n_mapped_A: int*, *n_geneset_B: int*, *n_mapped_B: int*, *n_overlaps: int*, *number_of_permutations: int*, *observed: int*, *empirical_pvalue: float*, *mean_null: numpy.mean*, *var_null: numpy.var*) → None
Update the content of the comparison table, adding a new row on the file

Parameters

- **setname_A** – the name of the geneset A
- **setname_B** – the name of the geneset B
- **n_geneset_A** – the number of genes in the geneset A
- **n_mapped_A** – the number of mapped genes in geneset A
- **n_geneset_B** – the number of genes in the geneset B
- **n_mapped_B** – the number of mapped genes in geneset B
- **n_overlaps** – the number of overlaps
- **number_of_permutations** – number of performed permutations
- **observed** – number of observed genes
- **empirical_pvalue** – value of the empirical pvalue
- **mean_null** – mean of the null distribution
- **var_null** – variance of the null distribution

Example

```
>>> import itertools
>>> import pygna.command as cm
>>> import pygna.reading_class as rc
>>> import pygna.statistical_comparison as sc
>>> geneset_a = rc.ReadGmt("genset_file").get_geneset("brca")
>>> setnames = [key for key in geneset_a.keys()]
>>> network = rc.ReadTsv("network_file.tsv").get_network()
>>> distance_matrix_filename = "distance_matrix.tsv"
>>> in_memory = True
>>> network = nx.Graph(network.subgraph(max(nx.connected_components(network),␣
↪key=len)))
>>> sp_diz = {"nodes": cm.read_distance_matrix(distance_matrix_filename, in_
↪memory=in_memory)[0],
```

(continues on next page)

(continued from previous page)

```
...             "matrix": cm.read_distance_matrix(distance_matrix_filename, in_
↪memory=in_memory)[1]}
>>> st_comparison = sc.StatisticalComparison(sc.comparison_shortest_path,␣
↪network, diz=sp_diz, n_proc=2)
>>> out = Output("networkfile.tsv", "results.csv", "myanalysis", "genset_a.csv
↪", setnames)
>>> for pair in itertools.combinations(setnames, 2):
...     observed, pvalue, null_d, a_mapped, b_mapped = st_comparison.
↪comparison_empirical_pvalue(set(geneset_a[pair[0]]), set(geneset_
↪a[pair[1]]), max_iter=number_of_permutations)
...     out.update_comparison_table_empirical(pair[0], pair[1],␣
↪len(set(geneset_a[pair[0]])), a_mapped, len(set(geneset_a[pair[1]])), b_
↪mapped, n_overlaps, number_of_permutations, observed, pvalue, np.mean(null_
↪d), np.var(null_d))
```

update_st_table_empirical(*setname: str*, *n_mapped: int*, *n_geneset: int*, *number_of_permutations: int*, *observed: int*, *empirical_pvalue: float*, *mean_null: numpy.mean*, *var_null: numpy.var*) → None

Update the table content, adding a new line to the file

Parameters

- **setname** – the name of the geneset
- **n_mapped** – the number of mapped genes
- **n_geneset** – the number of genesets
- **number_of_permutations** – the number of permutations
- **observed** – value of observed genes
- **empirical_pvalue** – value of the empirical p-value
- **mean_null** – mean of the null distribution
- **var_null** – var of the null distribution

Example

```
>>> setnames = ["A", "B", "C"]
>>> out = Output("networkfile.tsv", "results.csv", "myanalysis", "genset_a.csv
↪", setnames)
>>> out.create_st_table_empirical()
>>> out.update_st_table_empirical(setname, n_mapped, n_geneset, number_of_
↪permutations, observed, pvalue=0.001, mean_null=np.mean(0.11), var_null=np.
↪var(0.2))
```

pygna.output.**print_GMT**(*gmt_dictionary: dict*, *output_file: str*) → None

Save the dictionary on a GMT file

Parameters

- **gmt_dictionary** – the dictionary containing the data
- **output_file** – the file to save the data

Example

```
>>> gmt_dict = {"key": "dict_sets"}
>>> print_GMT(gmt_dict, "mygmt.gmt")
```

pygna.output.**apply_multiple_testing_correction**(*table_file: str*, *pval_col: str = 'empirical_pvalue'*, *method: str = 'fdr_bh'*, *threshold: float = 0.1*) → None

Apply the multiple testing correction and save the file in a csv file

Parameters

- **table_file** – the name of the file to read
- **pval_col** – the name column containing the empirical pvalue
- **method** – the correction method to use
- **threshold** – the threshold to use in the method

Example

```
>>> table_filename = "pygna_comparison_results.csv"
>>> apply_multiple_testing_correction(table_filename, pval_col="empirical_pvalue",
↪ method="fdr_bh", threshold=0.1)
```

pygna.output.**write_graph_summary**(*graph: networkx.classes.graph.Graph*, *output_file: str*, *net_name: str = None*) → None

This function takes a graph as input and writes the network properties in a text file

Parameters

- **graph** – the graph to print
- **output_file** – the name of the file to print
- **net_name** – the name of the network

Example

```
>>> import pygna.reading_class as rc
>>> text_output = "My summary stats"
>>> network = rc.ReadTsv("mynetwork.tsv").get_network()
>>> write_graph_summary(network, text_output, "mynetwork.tsv")
```

5.7 Plots

class pygna.plots.**PygnaFigure**

Abstract class that implements a general figure in Pygna. It has a class attribute representing the default height/width ratios for the figures

class pygna.plots.**VolcanoPlot**(*df: pandas.core.frame.DataFrame*, *output_file: str*, *loc: int = 2*, *p_col: str = 'empirical_pvalue'*, *id_col: str = 'setname_B'*, *plotting_col: str = 'observed'*, *x_threshold: float = 0.1*, *y_threshold: float = 0.1*, *y_label: str = '-log10(pvalue)'*, *x_label: str = 'z-score'*, *annotate: bool = False*, *size: int = 2*)

This class represent a Volcano Plot. It saves the value in the dataframe on a file.

5.8 Reading Data

class pygna.reading_class.**ReadingData**

Abstract class used to read different types of file. You can implement your own reading method, but remember that each subclass must implement the **readfile** and **get_data** methods

get_data()

Get the data from the reading class. This method must be always overridden

class pygna.reading_class.**ReadTsv**(*filename: str*, *pd_table: bool = False*, *int_type: int = None*)

This class is used to read and parse a network file in a tab-separated format (tsv).

get_data() → pandas.core.frame.DataFrame

Returns the data of the tsv file

Returns list representing the genes read in the file

Example

```
>>> tsvdata = ReadTsv("mydata.tsv").get_data()
```

get_network() → networkx.classes.graph.Graph

Returns the nx.graph object of the network

Returns graph containing the network information

Example

```
>>> tsvdata = ReadTsv("mydata.tsv").get_network()
```

class pygna.reading_class.**ReadGmt**(*filename: str*, *read_descriptor: bool = False*)

This class is used to read a gmt file, which contains information about the genes with a setname and separated by a comma

get_data() → dict

Returns the data of the gmt file

Returns dict representing the genes list

Example

```
>>> gmtdata = ReadGmt("mydata.gmt").get_data()
```

get_geneset(*setname: str = None*) → dict

Returns the geneset from the gmt file

Parameters **setname** – the setname to extract

Returns the geneset data

Example

```
>>> gmtdata = ReadGmt("mydata.gmt").get_geneset("brca")
```

class pygna.reading_class.**ReadCsv**(*filename: str*, *sep: str = '\t'*, *use_cols: list = None*, *column_to_fill: str = None*)

This class is used to read a csv file.

get_data() → pandas.core.frame.DataFrame

Returns the data of the csv file

Returns dataframe representing the data read inside the .csv

Example

```
>>> csvdata = ReadCsv("mydata.csv").get_data()
```

class pygna.reading_class.**ReadTxt**(*filename: str*)

This class reads a txt file containing a single gene per line

get_data() → pandas.core.frame.DataFrame

Get the dataframe from the class

Returns dataframe object from the file read

Example

```
>>> txtdata = ReadTxt("mydata.txt").get_data()
```

class pygna.reading_class.**ReadDistanceMatrix**(*filename: str*, *in_memory: bool = False*)

This class read a distance matrix in the HDF5 format

get_data() → [<class 'list'>, <class 'numpy.matrix'>]

Return the data of the HDF5 Matrix

Returns table data, the data of the HDF5 Matrix and table nodes, the nodes of the HDF5 Matrix

Example

```
>>> nodes, data = ReadDistanceMatrix("mydata.hdf5").get_data()
```

5.9 Statistical Comparison

class pygna.statistical_comparison.**StatisticalComparison**(*comparison_statistic*, *network: networkx.classes.graph.Graph*, *n_proc: int = 1*, *diz: dict = {}*, *degree_bins=1*)

This class implements the statistical analysis comparison between two genesets. Please refer to the single method documentation for the returning values

comparison_empirical_pvalue(*genesetA: set*, *genesetB: set*, *alternative: str = 'less'*, *max_iter: int = 100*, *keep: bool = False*) → [<class 'int'>, <class 'float'>, <class 'float'>, <class 'int'>, <class 'int'>]

Calculate the empirical value between two genesets

Parameters

- **genesetA** – the first geneset to compare
- **genesetB** – the second geneset to compare
- **alternative** – the pvalue selection of the observed genes
- **max_iter** – the maximum number of iterations
- **keep** – if the geneset B should not be kept

Return observed, pvalue, null_distribution, len(mapped_genesetA), len(mapped_genesetB)
the list with the data calculated

get_comparison_null_distribution(*genesetA: list*, *genesetB: list*, *n_samples: int*, *keep: bool*, *sampling_p_a=None*, *sampling_p_b=None*) → list

Calculate the null distribution between two genesets with single CPU

Parameters

- **genesetA** – the first geneset to compare
- **genesetB** – the second geneset to compare
- **n_samples** – the number of samples to be taken
- **keep** – if the geneset B should not be kept

Returns the random distribution calculated

get_comparison_null_distribution_mp(*genesetA: list*, *genesetB: list*, *max_iter: int = 100*, *keep: bool = False*, *sampling_p_a=None*, *sampling_p_b=None*) → numpy.ndarray

Calculate the null distribution between two genesets with multiple CPUs

Parameters

- **genesetA** – the first geneset to compare
- **genesetB** – the second geneset to compare
- **max_iter** – maximum number of iteration to perform
- **keep** – if the geneset B should not be kept
- **sampling_p_a** – random sampling probability for geneset a
- **sampling_p_b** – random sampling probability for geneset b

Returns the array with null distribution

pygna.statistical_comparison.**comparison_shortest_path**(*network: networkx.classes.graph.Graph*, *genesetA: set*, *genesetB: set*, *diz: dict*) → float

Evaluate the shortest path between two genesets

Parameters

- **network** – the graph representing the network
- **genesetA** – the first geneset list

- **genesetB** – the second geneset list
- **diz** – the dictionary containing the nodes name and index

pygna.statistical_comparison.**calculate_sum**(*n: numpy.ndarray*, *m: numpy.ndarray*, *diz: dict*) → numpy.ndarray

Evaluate the sum of the columns of two matrices

Parameters

- **n** – the first column
- **m** – the second column
- **diz** – the dictionary containing the data

pygna.statistical_comparison.**comparison_random_walk**(*network: networkx.classes.graph.Graph*, *genesetA: list*, *genesetB: list*, *diz: dict = {}*) → float

Evaluate the random walk on two genesets

Parameters

- **network** – the graph representing the network
- **genesetA** – the first geneset list
- **genesetB** – the second geneset list
- **diz** – the dictionary containing the nodes name and index

5.10 Statistical Diffusion

class pygna.statistical_diffusion.**DiffusionTest**(*test_statistic*, *nodes: list*, *diffusion_matrix: numpy.matrix*, *weights_table: pandas.core.frame.DataFrame*, *names_col: str = 'name'*, *weights_col: str = 'stat'*, *diz: dict = {}*)

This class elaborates the diffusion statistics. It elaborates the diffusion tests over the given network. Please refer to the single method documentation for the returning values

empirical_pvalue(*geneset: list*, *alternative: str = 'less'*, *max_iter: int = 100*, *cores: int = 1*) → [<class 'int'>, <class 'float'>, <class 'float'>, <class 'int'>, <class 'int'>]

Calculate the empirical pvalue on the genes list

Parameters

- **geneset** – the geneset to elaborate
- **alternative** – the pvalue selection of the observed genes
- **max_iter** – the number of iterations to be performed
- **cores** – the number of cores to be used

Return observed, pvalue, null_distribution, len(mapped_genesetA), len(mapped_genesetB)
the list with the data calculated

get_null_distribution(*geneset_index: list*, *n_samples: int*, *randomize: str = 'index'*) → list

Calculate the null distribution over the geneset

Parameters

- **geneset_index** – the geneset id that points to the geneset to be used
- **n_samples** – the number of samples to be taken

Returns the random distribution calculated for each element

get_null_distribution_mp(*geneset_index: list, iter: int = 100, n_proc: int = 1*) → numpy.ndarray

Calculate the null distribution with multiple cores on the geneset

Parameters

- **geneset_index** – the geneset id that point to the geneset to be used
- **iter** – the number of iterations to perform
- **n_proc** – the number of cpu to use for the elaboration

Returns the array with null distribution

pygna.statistical_diffusion.**map_table2matrix**(*node_list, x*)

pygna.statistical_diffusion.**weights_diffusion_statistic**(*matrix: numpy.matrix, weights: numpy.matrix, geneset_index: list, diz: dict = {}, observed_flag: bool = False*) → float

Not in use. This statistic reweights the original weights and returns the average reweighted statistic.

pygna.statistical_diffusion.**hotnet_diffusion_statistic**(*matrix: numpy.matrix, weights: numpy.matrix, geneset_index: list, diz: dict = {}, observed_flag: bool = False*) → numpy.ndarray

HOTNET2 like diffusion. Applies the diagonal matrix of weights and gets all rows and columns according to the genelist

Parameters

- **matrix** – the matrix corresponding to the graph
- **weights** – the matrix of which it will be created the diagonal matrix
- **geneset_index** – the gene list index
- **observed_flag** – TBD

5.11 Statistical Test

class pygna.statistical_test.**StatisticalTest**(*test_statistic, network: networkx.classes.graph.Graph, diz: dict = {}, degree_bins=1*)

This class implements the statistical analysis performed by Pygna. It performs the statistical tests on the given network, elaborates the number of observed genes, the pvalue etc. Please refer to the single method documentation for the returning values

empirical_pvalue(*geneset: set, alternative: str = 'less', max_iter: int = 100, cores: int = 1*) → [<class 'int'>, <class 'float'>, <class 'float'>, <class 'int'>, <class 'int'>, <class 'int'>]

Calculate the empirical pvalue on the genes list

Parameters

- **geneset** – the geneset to elaborate
- **alternative** – the pvalue selection of the observed genes
- **max_iter** – the number of iterations to be performed
- **cores** – the number of cores to be used

Return observed, pvalue, null_distribution, len(mapped_genesetA), len(mapped_genesetB)
the list with the data calculated

get_null_distribution(*geneset: list*, *n_samples: int*, *sampling_p=None*)
Calculate the null distribution over the geneset

Parameters

- **geneset** – the geneset to be used
- **n_samples** – the number of samples to be taken

Returns the random distribution calculated

get_null_distribution_mp(*geneset: list*, *iter: int = 100*, *n_proc: int = 1*, *sampling_p=None*)
Calculate the null distribution with multiple cores on the geneset

Parameters

- **geneset** – the geneset to be used
- **iter** – the number of iterations to perform
- **n_proc** – the number of cpu to use for the elaboration

Returns the array with null distribution

pygna.statistical_test.**geneset_localisation_statistic_median**(*network: networkx.classes.graph.Graph*, *geneset: set*, *diz: dict = {}*, *observed_flag: bool = False*) → float
Calculate the median shortest path for each node

Parameters

- **network** – the network used in the analysis
- **geneset** – the geneset to analyse
- **diz** – the dictionary containing the genes name
- **observed_flag** – whether the gene has been already observed

pygna.statistical_test.**geneset_localisation_statistic**(*network: networkx.classes.graph.Graph*, *geneset: set*, *diz: dict = {}*, *observed_flag: bool = False*) → float
Identify the genes in a geneset

Parameters

- **network** – the network used in the analysis
- **geneset** – the geneset to analyse
- **diz** – the dictionary containing the genes name

- **observed_flag** – whether the gene has been already observed

pygna.statistical_test.**geneset_module_statistic**(*network: networkx.classes.graph.Graph*, *geneset: set*, *diz: dict = {}*, *observed_flag: bool = False*) → float

Evaluate the length of a observed network

Parameters

- **network** – the network used in the analysis
- **geneset** – the geneset to analyse
- **diz** – the dictionary containing the genes name
- **observed_flag** – whether the gene has been already observed

pygna.statistical_test.**geneset_total_degree_statistic**(*network: networkx.classes.graph.Graph*, *geneset: set*, *diz: dict = {}*, *observed_flag: bool = False*) → float

Total degree of the geneset: average total_degree

Parameters

- **network** – the network used in the analysis
- **geneset** – the geneset to analyse
- **diz** – the dictionary containing the genes name
- **observed_flag** – whether the gene has been already observed

pygna.statistical_test.**geneset_internal_degree_statistic**(*network: networkx.classes.graph.Graph*, *geneset: set*, *diz: dict = {}*, *observed_flag: bool = False*) → float

Internal degree ratio: average of the ratio internal_degree/total_degree

Parameters

- **network** – the network used in the analysis
- **geneset** – the geneset to analyse
- **diz** – the dictionary containing the genes name
- **observed_flag** – whether the gene has been already observed

pygna.statistical_test.**geneset_RW_statistic**(*network: networkx.classes.graph.Graph*, *geneset: set*, *diz: dict = {}*, *observed_flag: bool = False*) → numpy.ndarray

Poisson binomial probability, sum of interaction probabilities for the genes in the geneset

Parameters

- **network** – the network used in the analysis
- **geneset** – the geneset to analyse
- **diz** – the dictionary containing the genes name
- **observed_flag** – whether the gene has been already observed

CHAPTER 6

Changelog

6.1 Latest Version

6.1.1 v3.3.0

Added node degree corrected sampling

6.1.2 v3.1.2

Fixing conda CI

6.1.3 v3.0.0

New version with conda CI.

6.1.4 v2.0.7

Updated requirements (python >=3.6)

6.1.5 v2.0.5

Solved problem in function from table to gmt.

6.1.6 v2.0.4

Fully working version with documentation for basic functionalities. Complete API docs missing.

6.1.7 v2.0.1

Documentation changed, likely to change in the near future. Bear with us!

6.1.8 v2.0.0

IO conventions changed, introduced temporary files. Now the package should be fully compatible with snakemake conventions.

6.2 Older Version

6.2.1 v1.0.0

All names changed to test_topology and test_association convention

6.2.2 07/06/2019

Single geneset analysis, added dataset dimension filter.

6.2.3 v0.3.1

Changes to network summary function, writes text rst formatted

6.2.4 v0.3.0

Diffusion weights test deseq support

6.2.5 v0.2.1

Temporary or redundant functions removed New utility function for reading csv files(deseq)

6.2.6 v0.2.0

Name refactoring for new repo

6.2.7 v0.0.0 - Initial release

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Index

H

M

O

P

R

S

T

U

V

W