BiMat : Start Guide

Cesar O. Flores, Timothée Poisot, and Joshua S. Weitz
http://ecotheory.biology.gatech.edu

February 16, 2014

1 Description

1.1 Main Goal

The main goal of BiMat is to facilitate the analysis of nestedness and modularity of bipartite ecological networks.

1.2 System Requirements

- MATLAB® 2009b or superior. BiMat may work in previous versions, but BiMat was not tested on them.
- The user is expected to have basic MATLAB® knowledge.
- (Optional)MATLAB® Parallel Computing Toolbox, in case the user want to work with the parallel BiMat version.

1.3 Functionality

BiMat is a MATLAB® library whose main function is the analysis of modularity and nestedness in bipartite ecological networks. Its main features are:

- Modularity and nestedness analysis.
- Diversity analysis using Shannon and/or Simpson's indexes.
- Different null models for the creation of random bipartite networks.
- Statistics values for helping the user to make inference about the structure of their networks (i.e. percentile,z-score).
- Internal statistics of the modules (multi-scale analysis).
- Meta-Statistics analysis (useful when the user need to compare and analyze many bipartite networks).
- Drawing of bipartite networks in both matrix and graph layout.

1.4 Workflow

The workflow of the BiMat package can be visualized in Figure 1.

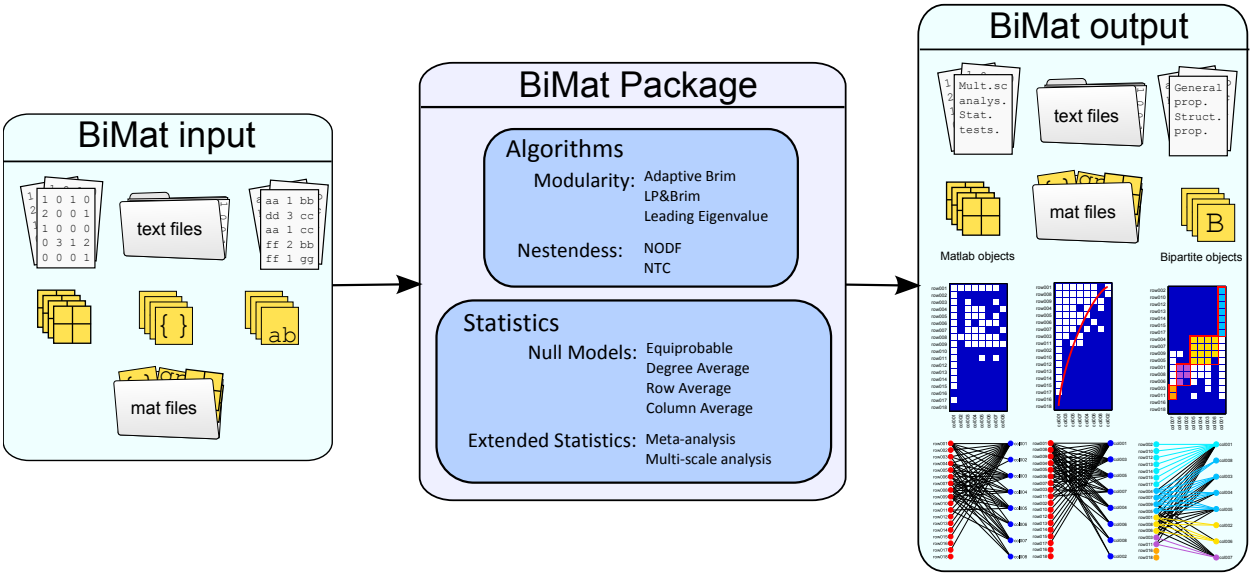


Figure 1: `BiMat` Workflow. The figure shows the main scheme of the `BiMat` package. `BiMat` can take matlab objects or text files as main input. The input is analyzed mainly around modularity and nestedness using a variety of null models. The user may also perform an additional multi-scale analysis on the data, or if he have more than one matrix to perform a meta-analysis in the entire data. Finally, the user can observe the results via matlab objects, text files, and plots.

2 Installation

2.1 Downloading BiMat

BiMat comes in two versions: (i) un-parallel and (ii) parallel version. While both have exactly the same functions, the last one has been coded for using more than one cpu core for performing the statistical tests, and therefore increase the speed of simulations. If you chose to use the parallel version be sure that the user have the Parallel Computing Toolbox installed in MATLAB® . The user can check if he has this toolbox by running in the MATLAB® command line:

```
>>ver
```

After entering this command, MATLAB® will display information about the version of MATLAB® the user is running, including a list of all toolboxes installed on the user's system.

No matter the version the user want to use, both versions can be downloaded from the main developer website: http://ecotheory.biology.gatech.edu/~cflores. Equally possible, the user can download the last available version from the main developer GitHub website: https://github.com/cesar7f/BiMat.

2.2 Installing BiMat and adding it to the MATLAB® path

To install BiMat , copy the downloaded zip file to a directory of interest and unzip it. Next, you will need to add BiMat to the MATLAB® path either temporally or permanently:

- **Temporal path:** Add the BiMat directory (and sub-directories) to the MATLAB® path. You can do that by typing in the MATLAB® command line:

  ```
  >>g=genpath('bimat_directory_location');
  >>addpath(g);
  ```

  You should replace bimat_directory_location with the full path to the directory in which you installed BiMat .

- **Permanent path:** Alternatively, the user can update permanently (also temporally) by accessing the MATLAB® path configuration. The path configuration can be accesed via menu *File –>Set Path*.

2.3 BiMat configuration: Options.m file

Most of the BiMat functions can work without the need of parameters by the user. However, if the user does not specify the required arguments, BiMat will assume that default values will be used. These default values are specified on the file main/Options.m that the user can modified according to his needs. A description of each parameter with its default value is indicated below:

- *Statistical Significance:* A two-tail test is the default way of testing for significance in BiMat . Notice that the user can perform a one-tail test by just duplicating the values below:
  - P_VALUE = 0.05: The p-value for testing statistical significance using a percentile test approach. Anything above the percentile $100*(1-p/2)$ will be significant, while anything below the percentile $100*(p/2)$ will be anti-significant.
  - Z_SCORE = 1.96: The z-score for testing statistical significance using a z-test approach. Anything above $|z|$ will be considered significant, while anything below $-|z|$ will be considered anti-significant. $z = 1.96$ has been chosen in order to correspond to $p = 0.05$.
- *Null Models:*

- `DEFAULT_NULL_MODEL = @NullModels.EQUIPROBABLE`: The default function for creating random networks.
- `ALLOW_ISOLATED_NODES = true`: When the network is sparse, a random network may be created with nodes with no links at all (matrix with empty rows or columns). `BiMat` by default allow this kind of random networks for performing the statistical test. However, the user may want to change this value to `false` and like this avoid the creation of this kind of random networks. However, the user must be aware that the time required for creating a random network without empty nodes will growth with the sparsity of the matrix.
- `TRIALS_FOR_NON_EMPTY_NODES = 1000`: This value is only used when the user changes the value of the previous parameter to `false`. In some extreme cases (a very sparse network), `BiMat` will not be able to find a random network without empty nodes. Hence, in order to avoid infinite loops, `BiMat` will stop looking for them after the number of trials specified in this parameter. If `BiMat` can not create a random network without empty nodes before this number of trials, `BiMat` will just create a random network without this constraint and will print the next message in the `MATLAB`® command line:

  ```
  Warning: Not possible to create a matrix with non isolated nodes.
  The random matrix was created without this constraint instead.
  Consider to modify Options.ALLOW_ISOLATED_NODES and/or Options.INCLUDE_EMPTY_NODES
  ```

- `INCLUDE_EMPTY_NODES = true`: Sometimes the user may have data with empty nodes (a matrix with empty rows and/or columns). Depending on the value of this parameter `BiMat` will chose between keeping these nodes (`true`) or deleting them from the adjacency matrix (`false`). Further, the user must be aware that including or not empty nodes will have an effect during the statistical tests of his data.
- `REPLICATES = 100`: The amount of replicates that `BiMat` performs in order to test for statistical significance. The value of `100` was chosen with the idea of getting quick results. However, the user must be aware that this value is no appropriate for accurate testing. The right value will depend on the kind of network (or networks) that the user is analyzing. It will depend mostly in two quantities: the fill and the size of the adjacency matrix. Experience from the developers indicate that if matrices are small $\sim 10 \times 10$ the appropiate number is $\sim 10,000$, while for big matrices $\sim 200 \times 200$, the appropriate number is $\sim 1,000$. However, the right way for testing the appropiate value is by looking and how the variance decrease as the number of replicates increase. The variance stops decreasing considerably with the number of replicates, increasing this last number does not have any effect on the statistical results.

- *Algorithms:* All the next parameters refer to the modularity algorithms behavior. The user can change the values here, or he can change the parameters dynamically by modifying the corresponding properties in the `BipartiteModularity` instance.

  - `OPTIMIZE_COMPONENTS = false`: Modularity is a function that depends in the global information of the network. However, sometimes, the user may have a network which is not connected (it has isolated components). By using the default value `false`, `BiMat` will optimize the modularity value at the entire adjacency matrix, while by using the value `true`, `BiMat` will optimize the modularity at the component level. Optimizing at the component level may decrease the global modularity value, thought the number of communities may increase and be more finner.
  - `MODULARITY_ALGORITHM = @AdaptiveBrim`: `BiMat` has three algorithms for optimizing the modularity equation and hence find the module configuration of the network.
  - `TRIALS_MODULARITY = 20`: The results of the modularity algorithms depends strongly in some initial random assignment of the communities. Therefore, `BiMat` restart the algorithm using this amount of times.

2.4 Getting help

At any moment you can access help from the command line using any of the next commands:

- `help class_name`: For a summary of the class file (i.e. `help StatisticalTest`). This will summarize all public and static methods and properties of the class. If you want to see private and/or protected methods you can use the `doc` instead of the `help` command.
- `help class_name.method_name`: For a summary of what the methods does and what kind of arguments it gets (i.e. `help StatisticalTest.DoNulls`).
- `help class_name.property_name`: For a summary of the property (i.e. `help StatisticalTest.replicates`).

You can always replace `help` by the `doc` command.

3 Examples

This section include three different examples to introduce the user to the main features of `BiMat` . All the code and data file can be found on the `examples` directory.

- `creating_networks.m`. It shows and explains the required input for `BiMat` .
- `moebus_study.m`. An analysis of the Moebus phage-bacteria bipartite network. It shows how to use the most important functions that are available to analyze a single matrix. This analysis include how to calculate most of the results published on [14].
- `group_matrices.m`. An analysis of a group of matrices that shows how to perform an analysis in many matrices at the same time. This example reproduce some of the results published on [13]. However, using this template all the results can be reproduced with a little extra effort.

3.1 BiMat - Creating networks

This example will introduce the user to the input of `BiMat` . It explains what input is required and how it is used by `BiMat` . This example is located on `examples/creating_networks.m` and make use of `examples/input_adja.txt` and `examples/input_matrix.txt` files.

3.1.1 Contents

- Add the source to the `MATLAB`® path
- Bipartite class and main input
- Optional input
- Creating input for Bipartite class
- Creating a Bipartite object from `MATLAB`® data
- Creating a Bipartite object from text files

3.1.2 Add the source to the `MATLAB`® path

```
5 % BiMat software.
6
7 %% Add the source to the matlab path
8 %Assuming that you run this script from examples directory
```

3.1.3 Bipartite class (main class)

The `Bipartite` is the fundamental class of the `BiMat` software. This class works as a communication bridge between all the available classes. Therefore, in order to work with BiMat we will always need to instantiate at least an object of this class.

3.1.4 Required input

The required input of the `Bipartite` class is a `MATLAB`® `matrix`, where the rows will represent the node set R and the columns the node set C, such that if the element `matrix(i,j)>0` a link between node r_i and c_j exist. This matrix input can contain only non-negative integers $\{0, 1, 2, 3...\}$. However, at present, **values greater than 1 are only used for plotting purposes** (e.g. color interactions according to weight) and not in the existing algorithms (which only work using the boolean version of the matrix).

3.1.5 Optional input

`BiMat` has two different types of optional input. The first type is for node labeling and the main use of it will be for labeling row and column nodes during plotting. The input must be encoded in a cell of strings for each set R and C nodes, such that each string in a cell corresponds to the label of a node. The size of such cells must corresponds to the number of nodes.

The second type of input consist of the type of node for either row and column nodes. For an example of type of nodes consider a bipartite network where R and C represent pollinators and plants respectively. In turn pollinators can be classified in birds and insects, which will be the classification for set R. The information of this classification is useful to explain modularity in terms of node classification. You can consult the Moebus study example for additional details. The classification input must be vectors of the same size than the number of nodes in rows and columns. The values must be positive integers $\{1, 2, 3, ...\}$ that represents the classification class of each node.

3.1.6 Creating input for Bipartite class

Here will show an example of the simplest way of creating a `Bipartite` object. We will create a bipartite networks using a `MATLAB®` matrix as input of the `Bipartite` object. This synthetic data matrix represents the interactions between a set of pollinators (rows) and a set of plants (columns). `matrix(i,j)>0` means that pollinator i pollinates plant j with strength `matrix(i,j)`.

```
% Here will show an example of the simplest way of creating Bipartite
% object:
%Creating the data
matrix = [2 0 2 2;...
          1 2 2 1;...
          2 0 0 2;...
          0 1 2 2;...
          0 0 1 0];
% For the next variables observe that the size of matrix 5x4 correlates with
% them
row_labels = {'insect 1', 'insect 2', 'insect 3', 'bird 1', 'bird 2'};
col_labels = {'flower 1', 'flower 2', 'grass 1', 'gras 2'};
%Notice that as long as each kind is represented by a diferented positive
%integer you will be fine.
row_ids = [1 1 1 3 3];
```

3.1.7 Creating a Bipartite object from `MATLAB®` data

Using the data we just created we can now create our `Bipartite` object:

```
%% Creating a Bipartite object from matlab data
% Using the data we just created we can now create or Bipartite object:
bp = Bipartite(matrix);
bp.row_labels = row_labels;
bp.col_labels = col_labels;
```

3.1.8 Creating a Bipartite object from text files

An additional way of creating data is by using the static functions from the `Reading.m` class. Currently two different formats are available. The first input format will contain only the information of the adjacency matrix (you will need to add row/column labels and classification id's if you need). A file example for creating the last data is on `examples/input_matrix.txt`, which contains:

```
2 0 2 2
1 2 2 1
2 0 0 2
0 1 2 2
0 0 1 0
```

The last format input can be called using:

```
%
% The last format input can be called using:
bp = Reader.READ_BIPARTITE_MATRIX('input_matrix.txt');
% We need to add labels and classification ids by ourselves
bp.row_labels = row_labels;
bp.col_labels = col_labels;
```

The second input format consist on writing the adjacency list. This input format will read also the row and column node labels. However if you need ids for the classification you will need to add by yourself. An example for the last data format is located on `examples/input_adja.txt` and is shown below:

```
insect_1 2 flower_1
insect_1 2 grass_1
insect_1 2 grass_2
insect_2 1 flower_1
insect_2 2 flower_2
insect_2 2 grass_1
insect_2 1 grass_2
insect_3 2 flower_1
insect_3 2 grass_2
bird_1 1 flower_2
bird_1 2 grass_1
bird_1 2 grass_2
bird_2 1 grass_1
```

The middle column is optional. If it is not used, the reading function will assume that is composed of ones only. We can now just call:

```
% The middle columns is optional. If it is not used, the reading function
% will assume that is composed of ones only. We can now just call:
bp = Reader.READ_ADJACENCY_LIST('input_adja.txt.');
% Wee need to add classification ids by ourselves
```

Now that you know how to create a network object, you can proceed to the next example that shows how to perform a complete analysis in a bipartite network.

3.2 BiMat Use case using Moebus cross-infection matrix data

This example will introduce the user to the most basic features of the `BiMat` Software. In order to do that we will calculate some of the results presented on the Flores et al 2012 paper (Multi-scale structure and geographic drivers of cross-infection within marine bacteria and phages) [14]. We will show how to plot, evaluate modularity and nestedness, and perform some statistics at the global and internal modular structure.

This example is located on `examples/moebus_study.m` and makes use of `examples/moebus_use_case.mat` data file.

3.2.1 Contents

- Add the source to the MATLAB® path
- Creating the `Bipartite` network object
- Calculating Modularity
- Calculating Nestedness
- Plotting in Matrix Layout
- Statistical analysis in the entire network
- Statistical analysis of the internal modules

3.2.2 Add the source to the MATLAB® path

```
10 %Assuming that you run this script from examples directory
11 g = genpath('../'); addpath(g);
12 close all; %Close any open figure
```

We need also to load the data from which we will be working on:

```
15 load moebus_use_case.mat;
```

```
load moebus_use_case.mat;
```

The loaded data contains the bipartite adjacency matrix of the Moebus and Nattkemper study [22], where 1's and 2's in the matrix represent either clear or turbid lysis spots. It also contains the labels for both bacteria and phages and their geographical location from which they were isolated across the Atlantic Ocean.

3.2.3 Creating the Bipartite network object

```
23 bp = Bipartite(moebus.weight_matrix);    % Create the main object
24 bp.row_labels = moebus.bacteria_labels;  % Updating node labels
25 bp.col_labels = moebus.phage_labels;
26 bp.row_class = moebus.bacteria_stations;   % Updating node ids
27 bp.col_class = moebus.phage_stations;
```

We can print the general properties of the network with:

```
30 bp.printer.PrintGeneralProperties();
```

```
General Properties
     Number of species:            501
     Number of row species:        286
     Number of column species:     215
     Number of Interactions:      1332
     Size:                       61490
     Connectance or fill:        0.022
```

3.2.4 Calculating Modularity

The modularity algorithm is encoded in the property `modules` of the `Bipartite` object (`bp.modules`). Tree algorithms are available:

1. Adaptive BRIM (`AdaptiveBrim.m`)
2. LP&BRIM (`LPBrim.m`)
3. Leading Eigenvector (`NewmanAlgorithm.m`)

Each algorithm optimizes the same modularity equation [3] for bipartite networks using different approaches. Only the Newman algorithm may return the same result. The other two perform at some point random module pre-assigments, and by consequence they may return the same result in each call. The default algorithm is specified on `Options.MODULARITY_ALGORITHM`. However, we can assign another algorithm dynamically. Here, for example, we will use the Newman's algorithm (Leading eigenvector):

```
47 bp.modules = LeadingEigenvector(bp.matrix);
48 % The next flag is exclusive of Newman Algorithm and what it does is to
49 % performn a final tuning after each sub-division (see Newman 2006).
50 bp.modules.DoKernighanLinTunning = false; %Very slow, so we turn off.
```

We need to calculate the modularity explicitly by calling:

```
53 bp.modules.Detect();
```

If we are interested only in node module indexes we can use `bp.modules.row_modules` and `bp.modules.col_modules`. However for modularity values we need to call `bp.modules.Qb` or `bp.modules.Qr` as follows:

```
60 fprintf('The modularity value Qb is %f\n', bp.modules.Qb);
61 fprintf('The fraction inside modules Qr is %f\n',bp.modules.Qr);
```

```
The modularity value Qb is 0.770942
The fraction inside modules Qr is 0.917417
```

Because `AdaptiveBrim` is not deterministic you may get a different result. In order to improve the result you may increase the parameter `Options.TRIALS_MODULARITY`, which specify how many random restarts will perform the algorithm.

The value $0 \leq Q_b \leq 1$ is calculated using the standard bipartite modularity function (introduced by Barber) [3] while the value $0 \leq Q_r \leq 1$ represents the fraction of interactions that fall inside modules.

3.2.5 Calculating Nestedness

Two algorithms exist for calculating nestedness. Contrary to the case of modularity, in this case there is no need to change the algorithm because all the algorithms have an independent property in the Bipartite object. These algorithms are:

- NODF (Nestedness metric based on overlap and decreasing filling). With value in the interval [0,1] [1].
- NTC (Nestedness Temperature Calculator) With value in the interval [0 1] [2].

The first algorithm is runned during the creation of the Bipartite object, but because the NTC algorithm is slow, you need to run the algorithm explicitally:

```
85 bp.ntc.Detect();
```

Finally to show the values of the two algorithms you need to call::

```
88 % The same value will be printed all the times
89 fprintf('The nestedness NODF value is %f\n', bp.nodf.N);
90 % Because the value depends in the sorting of rows and columns, it may
91 % variate from trial to trial.
92 fprintf('The nestedness NTC value is %f\n', bp.ntc.N);
```

```
The nestedness NODF value is 0.034053
The nestedness NTC value is 0.954287
```

We can print all the structure values by just calling:

```
95 bp.printer.PrintStructureValues();
```

```
Modularity
    Used algorithm:                 NewmanModularity
    N (Number of modules):                        49
    Qb (Standard metric):                     0.7709
    Qr (Ratio of int/ext inter):              0.8979
Nestedness
    NODF metric value:                        0.0341
    NTC metric value:                         0.9543
```

3.2.6 Plotting in Matrix Layout

You can print the layout of the original, nestedness, and modular sorting. If you matrix is weighted in a categorical way using integers (0,1,2...) you can visualize a different color for each interaction, where 0 is no interaction. For using this functionality you need to assign a color for each interaction and specifically indicate that you want a color for each interaction before calling the plot function:

```
103 figure(1);
104 % Matlab command to change the figure window;
105 set(gcf,'Position',[0   72   932   922]);
106 bp.plotter.font_size = 2.0; %Change the font size of the rows and labels
107 % Use different color for each kind of interaction
108 bp.plotter.use_type_interaction = true; %
109 bp.plotter.color_interactions(1,:) = [1 0 0]; %Red color for clear lysis
110 bp.plotter.color_interactions(2,:) = [0 0 1]; %Blue color for turbid spots
111 bp.plotter.back_color = 'white';
112 % After changing all the format we finally can call the plotting function.
113 bp.plotter.PlotMatrix();
```

For plotting the nestedness matrix you may decide to use or not an isocline. The nestedness pattern is just the matrix sorted in decreasing degree for row and column nodes.

```
118 figure(2);
119 % Matlab command to change the figure window;
120 set(gcf,'Position',[0+200    72   932   922]);
121 bp.plotter.use_isocline = true; %The isocline is used in the NTC algorithm.
122 bp.plotter.isocline_color = 'red'; %Decide the color of the isocline.
123 bp.plotter.PlotNestedMatrix();
```

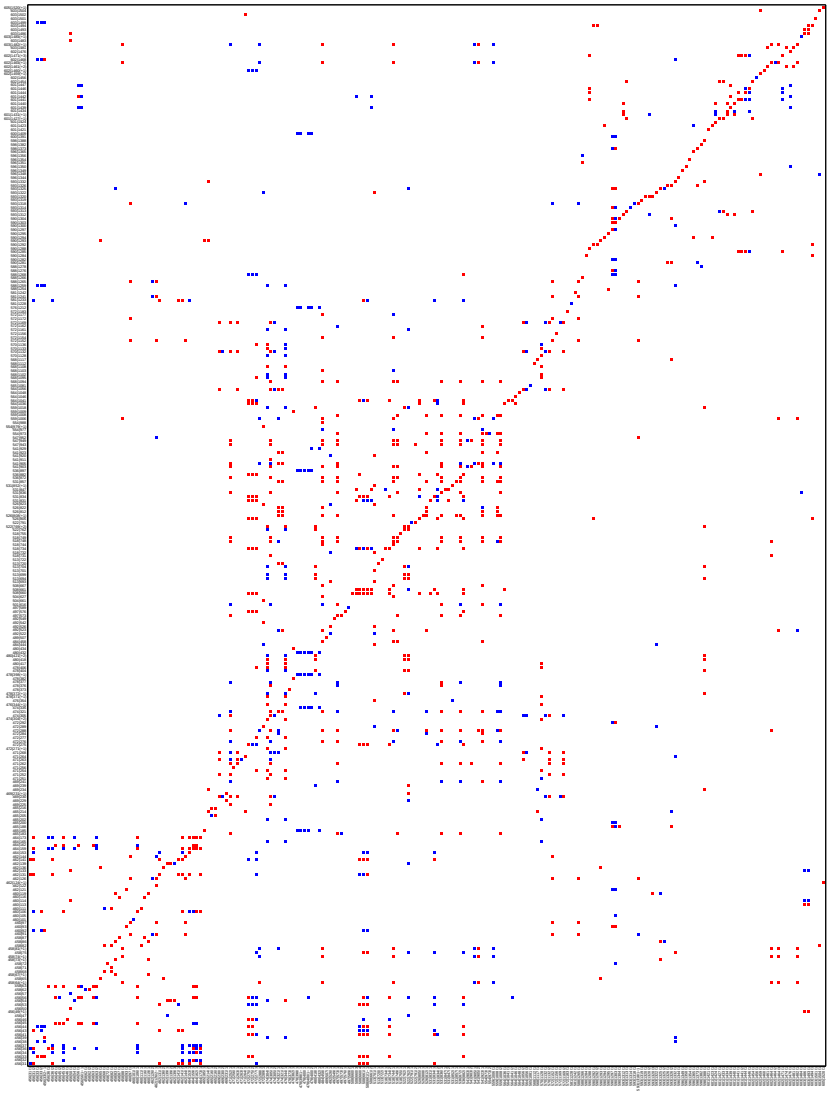


Figure 2: Original sorted matrix. Blue and red cells represent different strengths of infection between virus and bacteria. Rows and columns represent bacteria and phages, respectively.

For plotting the modularity sort, the plot function will calculate the modularity (call `bp.modules.Detect()`) if you have not previously called it.

```
figure(3);
% Matlab command to change the figure window;
set(gcf,'Position',[0+400    72   932   922]);
% This will plot isoclines inside modules.
bp.plotter.plot_iso_modules = true;
bp.plotter.PlotModularMatrix();
```

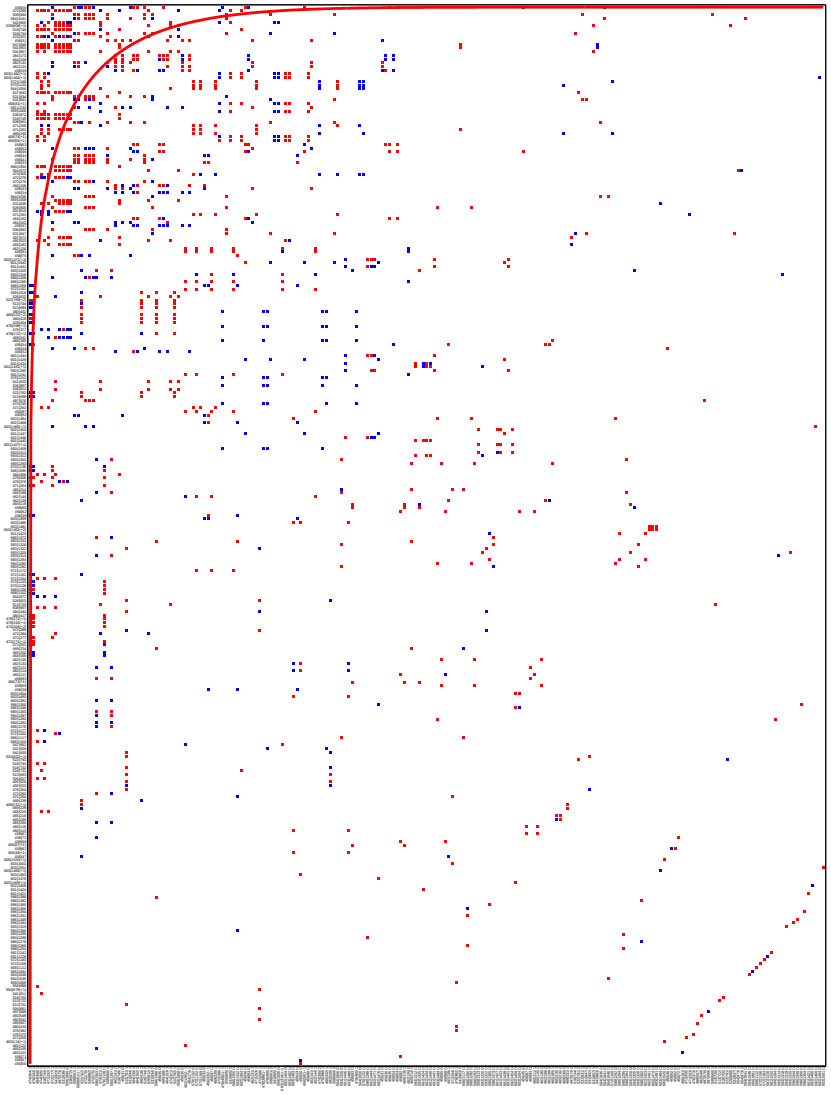


Figure 3: Nested sorted matrix. Blue and red cells represent different strengths of infection between virus and bacteria. In a perfectly nested pattern of the same fill than the current matrix, all the interaction cells will lay above the isocline (red line).

3.2.7 Plotting in graph layout

Plotting in graph layout use the same three functions than matrix layout. You just need to replace the part `Matrix` in the function name by `Graph`. For example, for plotting the graph layout of modularity we will need to type:

```
figure(4);
% Matlab command to change the figure window;
set(gcf,'Position',[19+600    72   932   922]);
bp.plotter.PlotModularGraph();
```

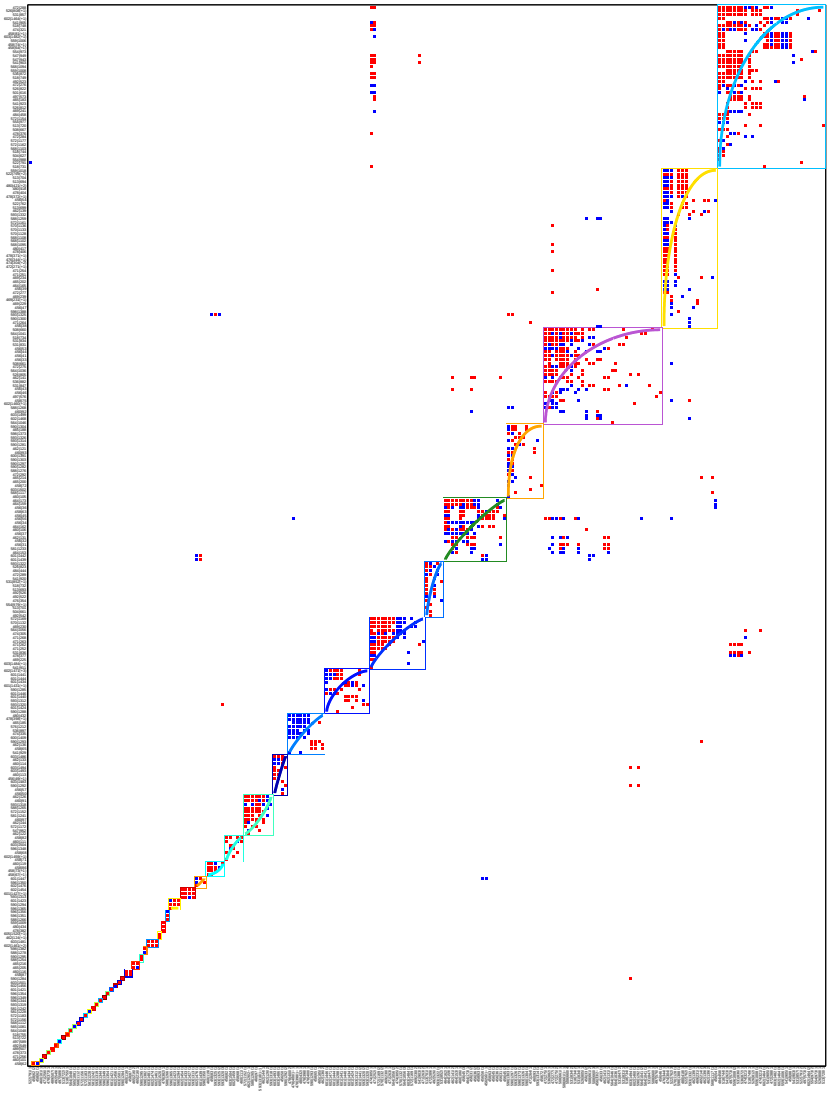


Figure 4: Modular sorted matrix. Blue and red cells represent different strengths of infection between virus and bacteria. Each block represent a different module.

3.2.8 Statistical analysis in the entire network

We can perform an statistical analysis in the entire network for nestedness and modularity. In order to make an statistical analysis of the structure values we need to decide how many replicates we will need and what null model is more convenient for what we need. File `NullModels.m` contain all the available null models, while file `StatisticalTest.m` contains all the functions required for performing this analysis. The current null models are:

- `NullModels.EQUIPROBABLE`: A random matrix in which all the interactions are uniformelly permuted. Another common name for this matrix is Bernoulli Matrix.
- `NullModels.AVERAGE`: A random matrix in which each element has an interaction with probability that depends on the sum of both the row and column to which the cell belongs to.
- `NullModels.AVERAGE_ROWS`: A random matrix in which each element has an interaction with proba-

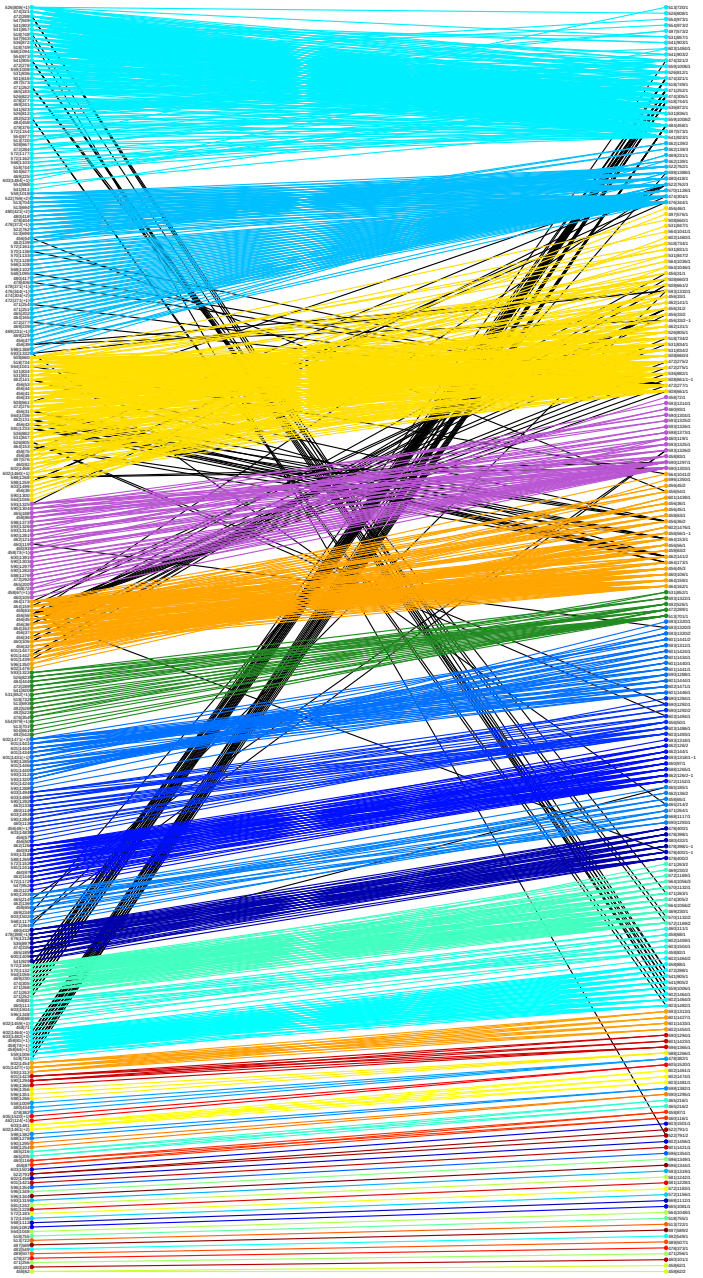


Figure 5: Modular graph layout. Nodes and interactions are colored according to the module they belong to. Black color is used for interaction across modules. Left and right side nodes represent bacteria and phages, respectively.

bility that depends on the sum of the row to which the cell belongs to.

- `NullModels.AVERAGE_COLS`: A random matrix in which each element has an interaction with probability that depends on the sum of the column to which the cell belongs to.

To perform the statistical analysis of all the structure values we can just type `bp.statistics.DoCompleteAnalysis()`, which will perform an analysis using the default number of

random matrices (`Options.REPLICATES`) and the default null model (`Options.DEFAULT_NULL_MODEL`). However, here we will chose directly those parameters:

```
% Do an analysis of modularity, nodf, and ntc values using 100 random
% matrices and the EQUIPROBABLE (Bernoulli) null model.
bp.statistics.DoCompleteAnalysis(100, @NullModels.EQUIPROBABLE);
```

```
Creating 100 null random matrices...
Performing NODF statistical analysis...
Performing Modularity statistical analysis...
Performing NTC statistical analysis...
```

The last function call printed information about the current status of the simulation. For printing the results we need to call:

```
bp.printer.PrintStructureStatistics(); %Print the statistical values
```

```
Modularity
     Used algorithm:                 NewmanModularity
     Null model:               NullModels.EQUIPROBABLE
     Replicates:                                  100
     Qb value:                                 0.7709
     z-score:                                 47.5933
     percent:                                100.0000
NODF Nestedness
     Null model:               NullModels.EQUIPROBABLE
     Replicates:                                  100
     NODF value:                               0.0341
     z-score:                                 18.4316
     percent:                                100.0000
NTC Nestedness
     Null model:               NullModels.EQUIPROBABLE
     Replicates:                                  100
     NTC value:                                0.9539
     z-score:                                 15.9336
     percent:                                100.0000
```

The printed information is as follows:

1. `Null Model`: The null model used to created the random matrices.
2. `Replicates`: The number of random matrices that were created for performing the statistical analysis.
3. `Value`: The value in the tested real matrix (Qb, NTC, NODF).
4. `z-score`: The z-score of the real matrix using the values of the random matrices.
5. `percent`: The percent of random matrices than have an smaller value than the matrix that is being evaluated.

Additional information that can be acceded via code includes the mean, standard deviation, and t-test results. Be aware that the number of replicates is especially critical parameter for the results of the statistical analysis. To chose this number consider the size and fill of the matrix. As a rule of thumb, 100 works fine as quick analysis, and 10,000 for a more accurate result (up to a matrix size of 300 by 300).

3.2.9 Statistical Analysis of the internal modules

In addition to be able to perform structure analysis in the entire network, we may be able (depending in the size and module configuration of the tested matrix) to perform a structural analysis in the internal modules. We will show next (i) how to do an analysis of modularity and nestedness in the internal modules and (ii) how to test for a possible correlation between node labeling and module configuration. All the functions for performing this kind of analysis is encoded in file `InternalStatistics.m`. For calculating the statistical structure of the internal modules we just need to call:

```
% 100 random matrices using the EQUIPROBABLE null model.
bp.internal_statistics.TestInternalModules(100,@NullModels.EQUIPROBABLE);
```

The last function call will print information about what is the current matrix (module) that is being evaluated. Like this, the user knows at every moment the current status of the analysis:

```
Testing Matrix: 1 . . .
Testing Matrix: 2 . . .
Testing Matrix: 3 . . .
Testing Matrix: 4 . . .
Testing Matrix: 5 . . .
Testing Matrix: 6 . . .
Testing Matrix: 7 . . .
. . . and so on . . .
```

Finally, to print the results we just need to call:

```
bp.printer.PrintStructureStatisticsOfModules(); % Print the results
```

```
Network,     Qb,Qb mean,Qb z-score,Qb percent,   NODF,NODF mean,NODF z-score,NODF percent,    NTC,NTC mean,NTC z-score,NTC percent
      1,0.38961,0.27568,   13.8964,       100,0.46477,  0.24068,     23.5413,         100,0.86108, 0.43876,    13.6499,        100
      2,0.42241,0.33384,    7.6765,       100,0.29082,   0.2388,      3.5568,         100,0.77327, 0.51871,     5.7448,        100
      3,0.31017,0.29024,    1.9282,        97,0.40425,  0.25719,     13.2868,         100,0.76455, 0.46967,      8.866,        100
      4,0.39294,0.39632,  -0.14158,        45,0.37092,  0.23815,      4.6112,         100,0.75558, 0.64589,     1.4784,         93
      5,  0.274,0.22035,    5.0719,       100,0.51256,  0.44014,      2.9949,         100,0.63604, 0.44693,     4.0164,        100
      6,0.37778,0.30769,    2.0488,        97,0.38468,  0.36383,      0.39141,         63,0.73545, 0.62046,     1.4505,         91
      7,0.20003,0.24335,   -3.8697,         0, 0.6963,  0.42384,      7.9444,         100,0.87392, 0.47566,     6.7485,        100
      8,0.47396,0.31748,    7.6554,       100,0.27045,   0.3541,     -1.6833,           2,0.56773, 0.57891,   -0.12247,         46
      9,0.30859,0.25506,    2.9189,       100,0.34967,  0.44545,     -1.9416,           2,0.54532, 0.55009,  -0.067587,         48
     10,0.31293, 0.2756,     1.227,        91,0.36885,  0.43368,    -0.85541,          20,0.65236, 0.66395,   -0.14178,         48
     11,0.16272,0.18304,   -1.4246,         6,0.76951,  0.57219,      3.5834,         100,0.86983, 0.61222,     3.4562,        100
     12,    0.4,0.29951,    2.3661,        99,0.41398,  0.40155,      0.13433,         58,0.66161, 0.71362,   -0.49594,         29
     13,0.18343,0.19118,  -0.27446,        33, 0.8125,  0.59937,      1.4436,          90,0.96298, 0.85172,    0.92978,         89
     14,   0.32, 0.2488,   0.67173,        32,0.66667,     0.46,      1.1346,          71,0.87346, 0.87029,   0.028445,         50
     15,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       1,       1,   -0.99499,          0
     16,0.16327,0.14041,    1.1225,        56,0.66667,  0.66667,     0.99499,          0,0.99999, 0.99999,    -1.9481,          0
     17,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     18,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     19,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     20,      0,      0,       NaN,         0,      0,        0,         NaN,           0,0.99999, 0.99999,   -0.99499,          0
     21,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     22,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     23,      0,      0,       NaN,         0,      0,        0,         NaN,           0,0.99999, 0.99999,   -0.99499,          0
     24,      0,      0,       NaN,         0,      0,        0,         NaN,           0,0.99999, 0.99999,   -0.99499,          0
     25,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     26,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     27,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     28,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     29,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     30,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     31,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     32,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     33,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     34,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     35,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     36,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     37,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     38,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     39,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     40,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     41,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     42,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     43,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     44,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     45,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
     46,      0,      0,       NaN,         0,      0,        0,         NaN,           0,       0,       0,        NaN,          0
```

```
47,     0,     0,     NaN,       0,     0,       0,       NaN,         0,     0,     0,       NaN,        0
48,     0,     0,     NaN,       0,     0,       0,       NaN,         0,     0,     0,       NaN,        0
49,   NaN,   NaN,     NaN,       0,     0,       0,       NaN,         0,     0,     0,       NaN,        0
```

The module indexing is in the same order that the plotted modularity matrix, in which Network 1 corresponds to the one located at the top right of Figure 4. This last created table shows the same values previouslly described. However it is specially usefull for describing some of the possible results that the user may get at some point. What follows summarize some of the important points:

- `NaN` values appear in many of the z-scores. The reason of those values is because they are fully connected and mostly composed of only one node of each type (a matrix of size 1×1. Therefore only one permutation of the matrix exist and by consequence all the random matrices are of the same size than the one being analyzed. This makes the standard deviation to be 0, and therefore the z-score to be $0/0 =$ `NaN`.
- Some of the matrices that have `NaN` values dot not have on the `NTC` values. However, they are also fully connected. The reason for which the z-score is not `NaN` (as theoretically predicted) is a computational error. In other words, for the z-score we are dividing a very small number over another very small number.
- The last module (49) has `NaN` values not only in the z-score values, but also in the values of modularity. The reason of these results, is because this module is compose of only one column (phage) node and the matrix is of size 0×1.

We can also study if a correlation exists between the row labeling and the module configuration. For performing this analysis we always will need a classification for rows and/or columns that group them in different sets. In this case we have as labeling the station number from which the bacteria and phages were extracted. Therefore what we will study is if there exist a correlation between the station location (geography) and the module configuration. We will use the same method that was used in Flores et al 2012 [14]. Given the labeling this method calculates the diversity index of the labeling inside each module and compare it with random permutations of the labeling across the matrix.

```
%Using the labeling of bp and 1000 random permutations
bp.internal_statistics.TestDiversityRows(1000);
% Using specific labeling and Shannon index
bp.internal_statistics.TestDiversityColumns( ...
    1000,moebus.phage_stations,@Diversity.SHANNON_INDEX);
%Print the information of column diversity
bp.printer.PrintColumnModuleDiversity();
```

```
Diversity index:       Diversity.SHANNON_INDEX
Random permutations:                      1000
Module,index value, zscore,percent
     1,      2.662,-1.7609,    5.7
     2,     2.3959,-0.2332,   31.1
     3,     2.3421,-5.1204,      0
     4,     1.6434,-3.3856,    0.4
     5,     1.2622,-9.1915,      0
     6,     1.6094,0.54671,   25.6
     7,     2.0262,-2.8726,    0.6
     8,     1.0751,-8.3588,      0
     9,     1.4979,-4.2423,    0.3
    10,     1.0397,     -2,    0.8
    11,     1.4942,-2.9839,      1
```

```
12,     1.3322,-1.2265,    2.6
13,    0.95027,-3.7459,    0.1
14,     1.0986,0.32725,   10.3
15,     1.0397,-1.9639,    0.5
16,     1.0986,0.31089,      9
17,          0,    NaN,      0
18,          0,    NaN,      0
19,          0,    NaN,      0
20,    0.63651, -2.982,      0
21,          0,    NaN,      0
22,          0,    NaN,      0
23,          0,-4.7735,      0
24,    0.69315, 0.1633,    2.6
25,          0,    NaN,      0
26,          0,    NaN,      0
27,          0,    NaN,      0
28,          0,    NaN,      0
29,          0,    NaN,      0
30,          0,    NaN,      0
31,          0,    NaN,      0
32,          0,    NaN,      0
33,          0,    NaN,      0
34,          0,    NaN,      0
35,          0,    NaN,      0
36,          0,    NaN,      0
37,          0,    NaN,      0
38,          0,    NaN,      0
39,          0,    NaN,      0
40,          0,    NaN,      0
41,          0,    NaN,      0
42,          0,    NaN,      0
43,          0,    NaN,      0
44,          0,    NaN,      0
45,          0,    NaN,      0
46,          0,    NaN,      0
47,          0,    NaN,      0
48,          0,-5.0991,      0
49,          0,    NaN,      0
```

Using a two tailed p-value of 0.05 we can see that 3, 4, 5, 7, ... are not as diverse as random labeling and conclude that those modules have phages that were isolated from similar locations. The module indexing is in the same order that the plotted modularity matrix, in which module 1 corresponds to the one located at the top of the plot. The `NaN` values happens because such modules have only a single phage and therefore the standard deviation used for calculating the z-score is 0.

3.3 BiMat - Group Statistics Example

This example will introduce the user to the features about how to perform an statistical analysis of a group of bipartite networks (matrices). For doing that we will use the data from Flores et Al 2011. This data consist of 38 bipartite adjacency matrices of different sizes. Each matrix is named according to the first author paper from which it was extracted. We will perform an analysis of modularity and nestedness in the entire set.

This example is located on `examples/group_matrices.m` and make use of `examples/group_testing_data.mat` data file.

3.3.1 Contents

- Add the source to the MATLAB® path
- Creating a Group Testing object
- Perform an statististical analysis in the set of matrices
- Using a `GroupStatistics` object to create your own plots

3.3.2 Add the source to the MATLAB® path

```
11 %Assuming that you run this script from examples directory
12 g = genpath('../'); addpath(g);
13 close all; %close all open figures
```

We need also to load the data from which we will be working on

```
16 load group_testing_data.mat;
```

The loaded data is a set of 38 matrices together with a name that refer to the first author and year from the paper from which the matrix was extracted. These matrices were published by Flores et Al 2011 [13].

3.3.3 Creating a Group Statistics object

If the number of random matrices and the null model are not assigned, 100 and AVERAGE are used as default. Here we will use 100 random matrices with the EQUIPROBABLE null model

```
22 gp = MetaStatistics(grouptesting.matrices);    % Create the main object
```

3.3.4 Perform an statistical analysis in the set of matrices

Suppose that we are interested in calculating the modularity and nestedness using the NTC algorithm as Flores et Al 2011 did. In addition, following the approach of Flores et Al 2011 [13], we want to use the equiprobable model as null model in our random networks. The way to perform this analysis is by running the next lines:

```
30 gp.replicates = 100; %How many random networks we want for each matrix
31 gp.null_model = @NullModels.EQUIPROBABLE; %Our Null model
32 gp.modularity_algorithm = @AdaptiveBrim; %Algorithm for modularity.
33 gp.do_ntc = 1; % Perform NTC analysis (default)
34 gp.do_modul = 1; % Perform Modularity analysis (default)
35 gp.do_nodf = 1; % Perform Modularity analysis (default)
36 gp.names = grouptesting.name;
37 gp.DoMetaAnalyisis(); % Perform the analysis.
```

```
Testing Matrix: 1 . . .
Testing Matrix: 2 . . .
Testing Matrix: 3 . . .
Testing Matrix: 4 . . .
Testing Matrix: 5 . . .
```

```
Testing Matrix: 6 . . .
Testing Matrix: 7 . . .
. . . and so on . . .
```

Notice that `DoGroupTesting` method prints information about the current networks that is being analyzed, such that the user will know at every moment the current status of the analysis. After the analysis is finished a simple statistical measure to say that a matrix is nested and/or modular is to chose a two tail p-value = 0.05 as Flores et al 2011 did. Therefore, the next lines of code will show how many matrices are found nested and/or modular

```
46 fprintf('Number of NTC nested matrices: %i\n',sum(gp.ntc_values.percentile ≥ 97.5));
47 fprintf('Number of NODF nested matrices: %i\n',sum(gp.nodf_values.percentile ≥ 97.5));
48 fprintf('Number of modular matrices: %i\n',sum(gp.qb_values.percentile ≥ 97.5));
```

```
Number of nested matrices: 29
Number of modular matrices: 6
```

Because we only did 100 random matrices you may get different results. For a more accurate result you may try 1.000 or even 10,000. Finally we can show detailed results for the entire set of matrices:

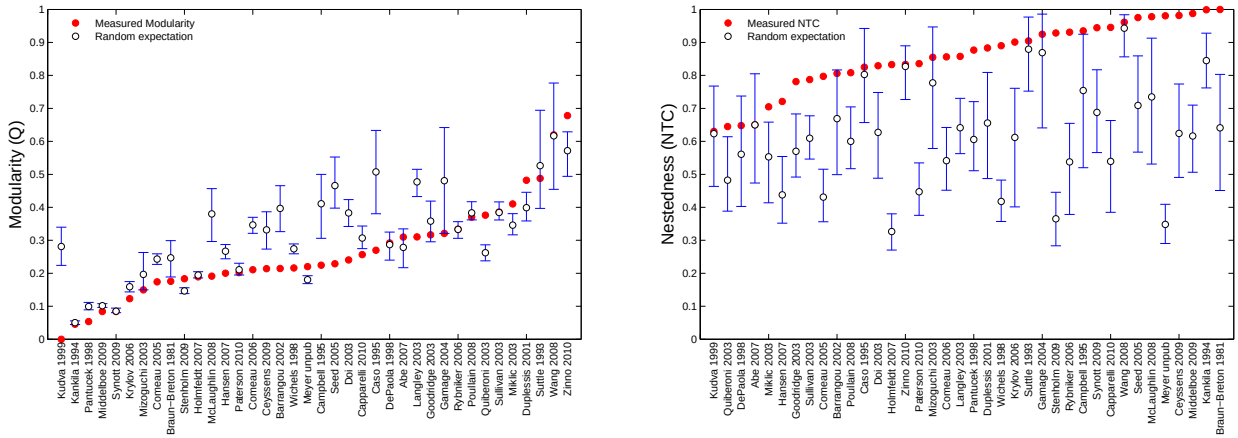
```
53 gp.Print();
```

```
Network,      Qb, Qb mean,Qb z-score,Qb percent,    NODF,NODF mean,NODF z-score,NODF percent,    NTC,NTC mean,NTC z-score,NTC percent
      1, 0.30992, 0.27149,    1.2134,        93, 0.28197,  0.45503,     -2.2113,           2,0.65025, 0.66945,   -0.21996,         38
      2,  0.2144, 0.40165,   -5.1897,         0, 0.19418,  0.27106,     -1.4879,           6,0.80601, 0.68326,     1.451,         95
      3, 0.17556,  0.2437,   -2.0928,         3, 0.47436,  0.43916,     0.62112,          73,0.99985, 0.64117,    4.6862,        100
      4, 0.22449, 0.40286,   -3.5167,         0,  0.2971,  0.26748,     0.45136,          72, 0.9352, 0.76571,    1.7461,         98
      5, 0.25652, 0.30552,   -2.7321,         1,  0.6114,  0.37957,       5.288,         100,0.94524,  0.5527,    5.2917,        100
      6,  0.2699, 0.50685,   -3.6399,         0,0.030888, 0.072604,     -2.9556,           0,0.82486, 0.78899,    0.41664,        60
      7, 0.21403, 0.33322,   -4.1449,         0, 0.29565,  0.34784,     -1.0767,          13,0.98175,  0.6181,    4.2708,        100
      8,   0.174, 0.24526,   -6.8079,         0, 0.46282,  0.40415,       2.588,         100,0.79783, 0.43154,    6.9821,        100
      9, 0.21395, 0.34563,  -10.4795,         0, 0.28559,  0.24741,      2.4581,          99,0.85564, 0.54169,    7.8469,        100
     10, 0.29191, 0.27974,   0.57535,        74,  0.5081,   0.4438,      1.0616,          85,0.64376, 0.59709,    0.6057,         68
     11, 0.24033, 0.38446,   -6.2761,         0, 0.17111,  0.28159,     -3.0679,           0,0.82933, 0.61602,    3.0949,        100
     12,  0.4821, 0.40186,      2.83,       100,  0.2197,  0.26988,     -1.1726,           9,0.86948, 0.65688,    2.4762,        100
     13, 0.30864,  0.4842,   -2.3729,         1, 0.19444,  0.15741,      0.7342,          75,0.92493, 0.88036,    0.65076,        69
     14, 0.31667, 0.34872,  -0.94887,        12,0.097822, 0.089063,     0.80359,          85,0.77249, 0.56627,     4.645,        100
     15, 0.20023, 0.26613,   -6.5474,         0, 0.37647,  0.37026,     0.26789,          54,0.72051, 0.42937,    6.2751,        100
     16, 0.18696, 0.19292,   -1.2011,        10, 0.61153,  0.40412,     17.0309,         100,0.83459, 0.33253,    17.5724,       100
     17,0.045608,0.050021,   -1.7491,         1, 0.60563,  0.70271,     -3.2974,           2,0.99875, 0.84967,    3.6965,        100
     18,  0.1231, 0.15739,    -4.343,         0, 0.67625,  0.61862,      1.221,           88,0.90103, 0.60926,    3.7284,        100
     19,       0,  0.2781,   -8.3767,         0,       0,   0.3749,     -6.5682,           0,0.63018, 0.63498,  -0.069048,        47
     20, 0.31027, 0.47264,   -7.6013,         0, 0.11655,   0.1328,     -1.7354,           7,0.85639, 0.64653,    4.9957,        100
     21, 0.19136, 0.37914,   -4.1428,         0, 0.34694,  0.31734,     0.45722,          64,0.97793, 0.73159,    2.5262,        100
     22, 0.22015, 0.18003,    7.0829,       100, 0.85688,  0.46691,     25.9477,         100, 0.9807, 0.34648,    20.3579,       100
     23, 0.08406, 0.10161,   -5.1781,         0, 0.54985,  0.68441,     -4.4116,           0,0.98762, 0.60969,    6.0492,        100
     24,  0.4102, 0.34576,    3.7243,       100, 0.19474,   0.2912,     -3.2609,           0,0.70409, 0.54842,    2.4715,         99
     25, 0.14966, 0.19488,   -2.3858,         1, 0.58971,   0.5887,   0.0099181,          49,0.85489, 0.77292,    0.85917,        79
     26,0.053624,0.099352,   -8.5646,         0, 0.27302,  0.60459,    -21.8246,           0,0.87649, 0.61087,    6.0426,        100
     27, 0.20209, 0.21213,     -1.15,        14, 0.51298,  0.50088,     0.62949,          75,0.83643, 0.43449,    14.4161,       100
     28, 0.35584, 0.38299,   -1.6314,         4, 0.30619,  0.20129,      6.7904,         100,0.80589, 0.59973,    3.7898,        100
     29, 0.37622, 0.26251,    9.0951,       100, 0.41751,  0.41328,     0.14669,          58,0.64478, 0.47417,    3.3102,        100
     30, 0.33347, 0.33582,  -0.12742,        46, 0.53377,  0.31019,      7.2935,         100,0.93098, 0.53837,    5.6463,        100
     31, 0.22893, 0.47055,   -6.6212,         0, 0.18127,   0.1555,       1.21,           87,0.97517,  0.7038,    3.4536,        100
     32, 0.18341,  0.1465,    9.2354,       100, 0.71354,  0.55809,     8.6466,         100,0.92835, 0.36665,    15.2208,       100
     33, 0.38847, 0.38518,   0.26049,        63, 0.22991,  0.17142,     4.7962,         100,0.78597, 0.61381,    4.3825,        100
     34,  0.4876, 0.51298,  -0.34778,        35, 0.18421,  0.13342,     1.1952,          89,0.86017, 0.89187,   -0.53745,         25
     35,0.084203,0.084955,  -0.29141,        42,  0.7063,  0.71613,    -0.29018,          36,0.94762,  0.6857,    4.2754,        100
     36, 0.61983, 0.63331,  -0.15015,        40, 0.04023, 0.029799,     0.73559,          79,0.98846, 0.94324,    1.3188,        100
     37, 0.21079, 0.27282,   -6.8961,         0, 0.70141,  0.26149,     49.6956,         100,0.88983, 0.42266,    14.2301,       100
     38,  0.6743, 0.56996,    3.3928,       100, 0.12467, 0.096073,     2.2108,          100,0.83768, 0.82511,    0.32553,        58
```

3.3.5 Plotting results

The user can visualize the results of the last output in a graphical way. For example for visualizing the results of modularity and NTC nestedness value, the user can type:

```
58 gp.plotter.p_value = 0.05; %p-value for error bars
59 gp.plotter.font_size = 10; %Size for x-labels.
60 gp.plotter.PlotModularValues();
61 gp.plotter.PlotNTCValues();
```



(a) Modularity statistics

(b) Nestedness (NTC) statistics

Figure 6: Visual representation of the statistical tests in the set of matrices. Red circles represent the value of the analyzed networks. White circles represent the mean of the null model, while the error bars represent the networks that falls inside a two-tailed version of the random null model values. The margin of the error bars are (p-value,1-p-value), where p-value can is an optional argument of the plot functions.

```
gp.plotter.p_value = 0.05; %p—value for color labeling
% Plot of nested graphs
gp.plotter.bead_color_rows = 'blue'; %Color of row nodes
gp.plotter.bead_color_columns = 'red'; %Color of column nodes
gp.plotter.link_width = 0.5; %Edge width
gp.plotter.nest_test = 2; %Use NODF for color labels (1 is for NTC).
gp.plotter.PlotModularMatrices(5,8);
%Plot of modular matrices
gp.plotter.PlotNestedGraphs(5,8);
```

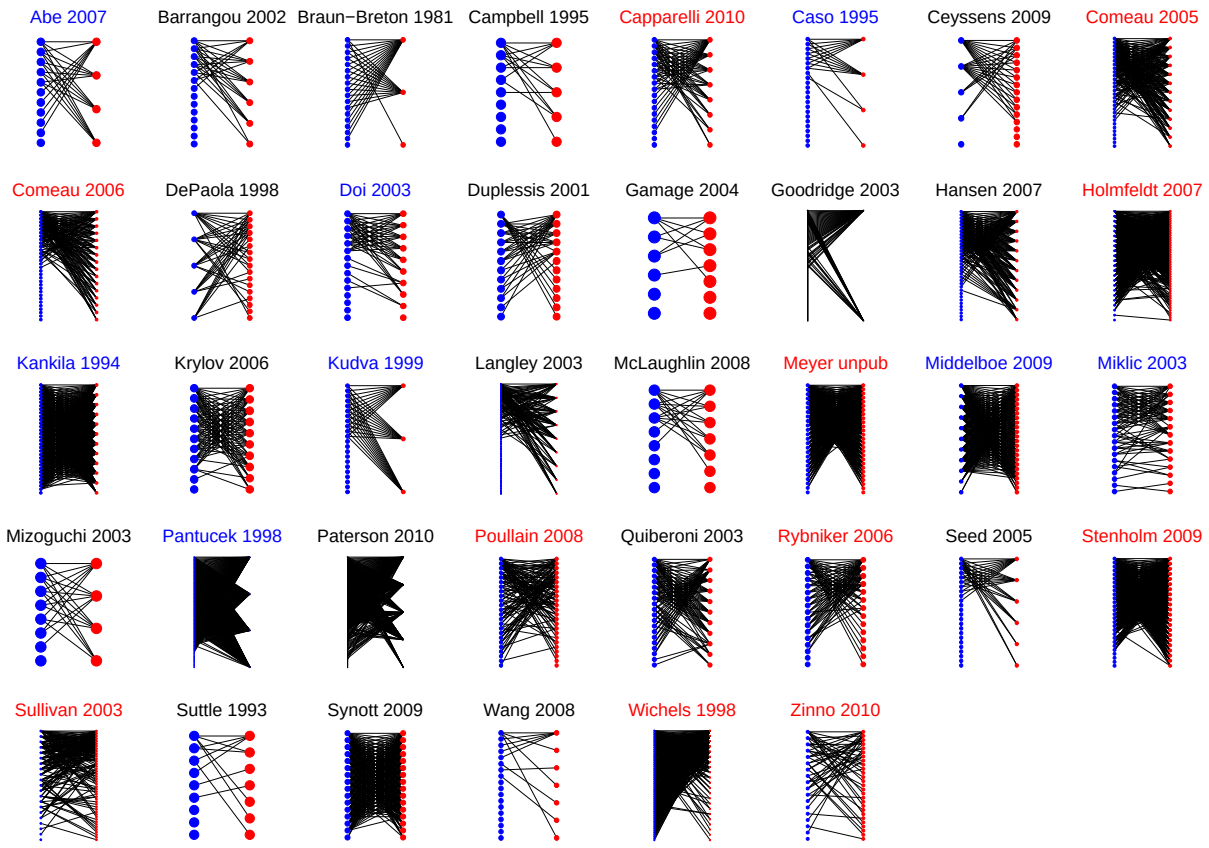


Figure 7: NODF nestedness values of a set of 38 matrices of phage-bacteria networks. A two-tail p-value of 0.05 was used for labeling the names. Blue and and red lebels represent anti and statistical significance, respectively. Notice that this Figure shows an smaller number of nested matrices than the NTC plot of the previous figure.

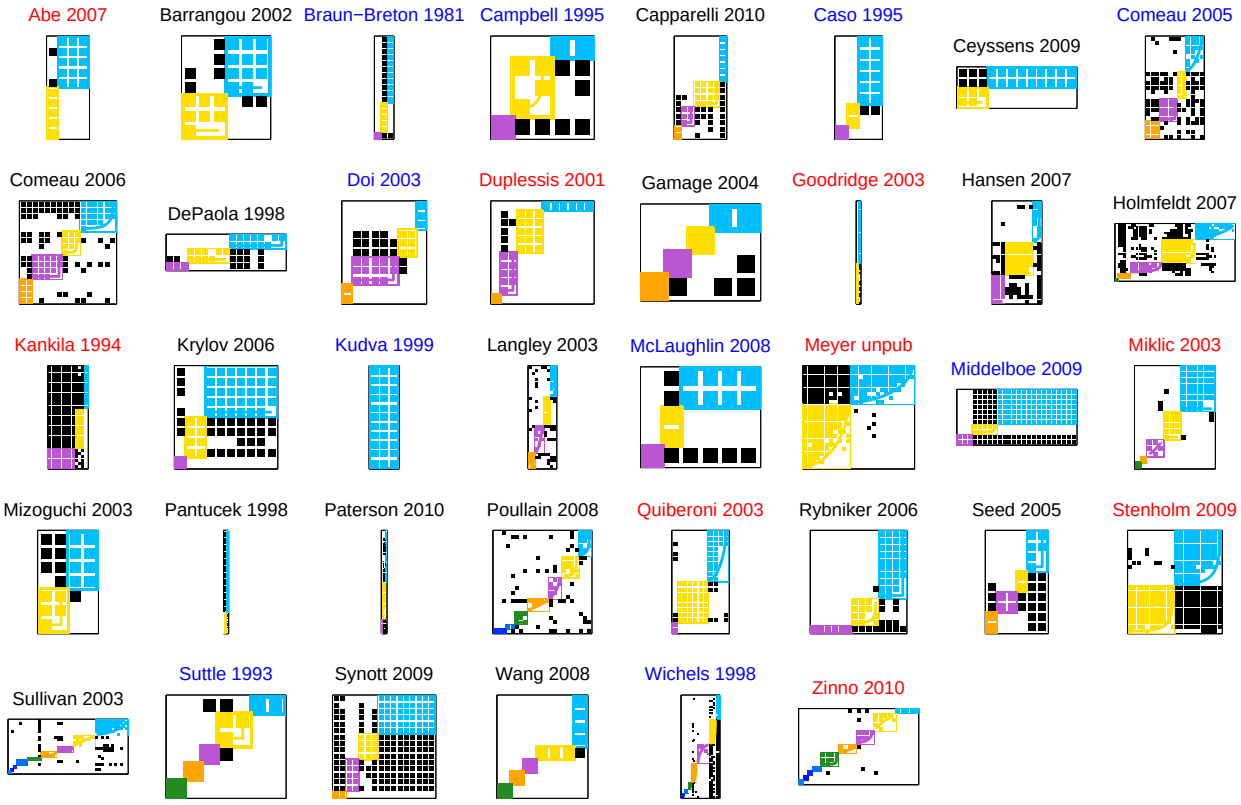


Figure 8: The meta-set collected on Flores et al [13] plotted using the modularity algorithm of the `BiMat` library. Red and blue labels represent significant modularity ($p \geq 0.975$) and anti-modularity ($p \leq 0.275$), respectively. For bibliographic information about these matrices see [13].

References

[1] Mário Almeida-Neto, Paulo Guimaraes, Paulo R Guimarães, Rafael D Loyola, and Werner Ulrich. A consistent metric for nestedness analysis in ecological systems: reconciling concept and measurement. *Oikos*, 117(8):1227–1239, 2008.

[2] Wirt Atmar and Bruce D Patterson. The measure of order and disorder in the distribution of species in fragmented habitat. *Oecologia*, 96:373–382, 1993.

[3] Michael Barber. Modularity and community detection in bipartite networks. *Physical Review E*, 76:066102, 2007.

[4] Michael J Barber, Margarida Faria, Ludwig Streit, and Oleg Strogan. Searching for communities in bipartite networks. *arXiv preprint arXiv:0803.2854*, 2008.

[5] Jordi Bascompte and Pedro Jordano. Plant-animal mutualistic networks: the architecture of biodiversity. *Annu. Rev. Ecol. Evol. Syst.*, 38:567–593, 2007.

[6] Jordi Bascompte, Pedro Jordano, Carlos J Melián, and Jens M Olesen. The nested assembly of plant–animal mutualistic networks. *Proceedings of the National Academy of Sciences of the United States of America*, 100:9383–9387, 2003.

[7] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: an open source software for exploring and manipulating networks. In *ICWSM*, 2009.

[8] Ugo Bastolla, Miguel A Fortuna, Alberto Pascual-García, Antonio Ferrera, Bartolo Luque, and Jordi Bascompte. The architecture of mutualistic networks minimizes competition and increases biodiversity. *Nature*, 458(7241):1018–1020, 2009.

[9] Gary Chartrand. Introductory graph theory. 1985.

[10] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5), 2006.

[11] C. F. Dormann, B. Gruber, and J. Fruend. Introducing the bipartite package: Analysing ecological networks. *R News*, 8(2):8–11, 2008.

[12] Jennifer A Dunne. The network structure of food webs. *Ecological networks: linking structure to dynamics in food webs*, pages 27–86, 2006.

[13] Cesar O Flores, Justin R Meyer, Sergi Valverde, Lauren Farr, and Joshua S Weitz. Statistical structure of host–phage interactions. *Proceedings of the National Academy of Sciences*, 108(28):E288–E297, 2011.

[14] Cesar O Flores, Sergi Valverde, and Joshua S Weitz. Multi-scale structure and geographic drivers of cross-infection within marine bacteria and phages. *The ISME journal*, 7(3):520–532, 2013.

[15] Miguel A Fortuna, Daniel B Stouffer, Jens M Olesen, Pedro Jordano, David Mouillot, Boris R Krasnov, Robert Poulin, and Jordi Bascompte. Nestedness versus modularity in ecological networks: two sides of the same coin? *Journal of Animal Ecology*, 79(4):811–817, 2010.

[16] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[17] Javier Galeano, Juan M Pastor, and Jose M Iriondo. Weighted-interaction nestedness estimator (wine): A new estimator to calculate over frequency matrices. *Environmental Modelling & Software*, 24(11):1342–1346, 2009.

[18] Paulo R Guimaraes Jr and Paulo Guimarães. Improving the analyses of nestedness for large sets of matrices. *Environmental Modelling & Software*, 21(10):1512–1513, 2006.

[19] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Laboratory (LANL), 2008.

[20] Lucas N Joppa, Jordi Bascompte, Jose M Montoya, Ricard V Sole, Jim Sanderson, and Stuart L Pimm. Reciprocal specialization in ecological networks. *Ecology letters*, 12(9):961–969, 2009.

[21] Xin Liu and Tsuyoshi Murata. Community detection in large-scale bipartite networks. In *Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT '09. IEEE/WIC/ACM International Joint Conferences on*, volume 1, pages 50 –57, sept. 2009.

[22] K Moebus and H Nattkemper. Bacteriophage sensitivity patterns among bacteria isolated from marine waters. *Helgoländer Meeresuntersuchungen*, 34(3):375–385, 1981.

[23] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[24] Timothée Poisot. An a posteriori measure of network modularity. *F1000Research*, 2, 2013.

[25] Timothée Poisot, Gildas Lepennetier, Esteban Martinez, Johan Ramsayer, and Michael E Hochberg. Resource availability affects the structure of a natural bacteria–bacteriophage community. *Biology letters*, 7(2):201–204, 2011.

[26] Timothée Poisot, Peter H Thrall, and Michael E Hochberg. Trophic network structure emerges through antagonistic coevolution in temporally varying environments. *Proceedings of the Royal Society B: Biological Sciences*, 279(1727):299–308, 2012.

[27] Robert Poulin. Network analysis shining light on parasite ecology and diversity. *Trends in parasitology*, 26(10):492–498, 2010.

[28] Stephen R Proulx, Daniel EL Promislow, and Patrick C Phillips. Network thinking in ecology and evolution. *Trends in Ecology & Evolution*, 20(6):345–353, 2005.

[29] M A Rodríguez-Gironés and L Santamaría. A new algorithm to calculate the nestedness temperature of presence-absence matrices. *Journal of Biogeography*, 33:924–935, 2006.

[30] EN Sawardecker, CA Amundsen, M Sales-Pardo, and LAN Amaral. Comparison of methods for the detection of node group membership in bipartite networks. *The European Physical Journal B*, 72(4):671–677, 2009.

[31] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498–2504, 2003.

[32] Phillip PA Staniczenko, Jason C Kopp, and Stefano Allesina. The ghost of nestedness in ecological networks. *Nature communications*, 4:1391, 2013.

[33] Daniel B Stouffer and Jordi Bascompte. Compartmentalization increases food-web persistence. *Proceedings of the National Academy of Sciences*, 108(9):3648–3652, 2011.

[34] Elisa Thebault and Colin Fontaine. Does asymmetric specialization differ between mutualistic and trophic networks? *Oikos*, 117(4):555–563, 2008.

[35] Elisa Thébault and Colin Fontaine. Stability of ecological communities and the architecture of mutualistic and trophic networks. *Science*, 329(5993):853–856, 2010.

[36] Werner Ulrich, Mário Almeida-Neto, and Nicholas J Gotelli. A consumer's guide to nestedness analysis. *Oikos*, 118(1):3–17, 2009.

[37] Werner Ulrich and Nicholas J Gotelli. Null model analysis of species nestedness patterns. *Ecology*, 88(7):1824–1831, 2007.