BiMat : Start Guide

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

April 9, 2015

1 Description

This document contains the start guide for the BiMat library. An extended documentation of this library can be located on: http://bimat.github.io/

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® 2011 or superior. BiMat may work in previous versions, but BiMat was not tested on them.
- The user is expected to have basic MATLAB® knowledge.

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.

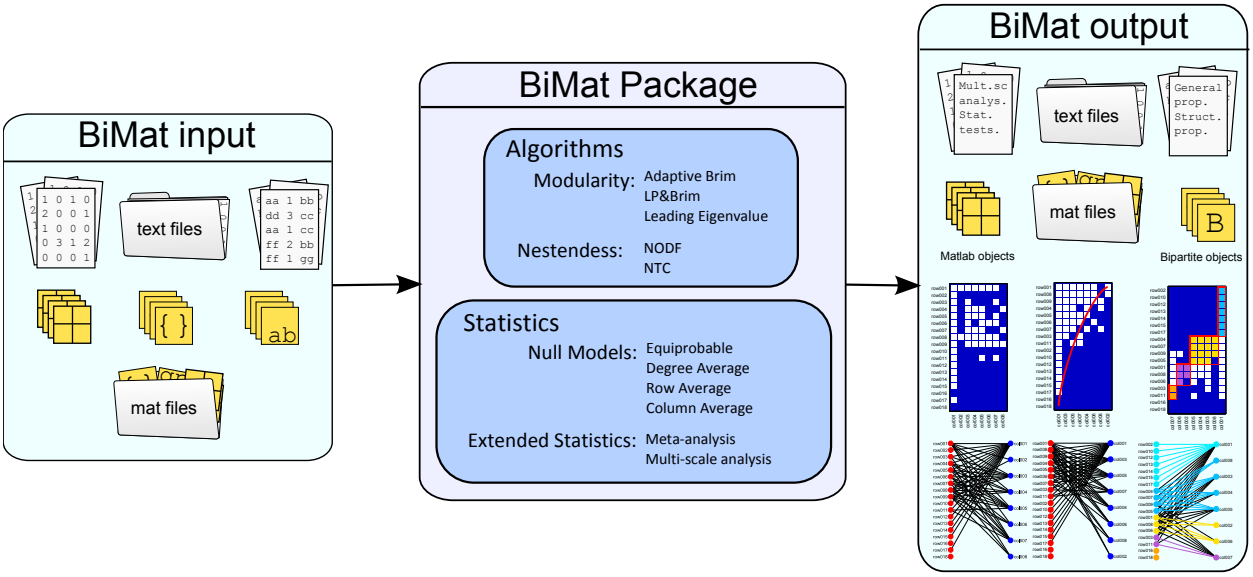


Figure 1: `BiMat` Workflow. The figure shows the main scheme of the `BiMat` package. `BiMat` can take matlab objects or text files as main input. The input is analyzed mainly around modularity and nestedness using a variety of null models. The user may also perform an additional multi-scale analysis 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 can be downloaded from the main developer website: http://bimat.github.io/.

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.
- SWAP_FIXED_FACTOR = 100: This swap factor S_f is used for creating random networks using the FIXED null model. The amount of performed random swaps in the matrix is $S_f E$, were E is the number of edges.
- 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 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 or Nestedness.
  - 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.
  - NESTEDNESS_ALGORITHM = @NestednessNODF: BiMat has two metrics for evaluating nestedness.
  - 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. For another the description of other examples included in the same directory, please visit `http://bimat.github.io/`

- `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 [3].
- `phage_bacteria_meta_analysis.m`. An analysis of a group of matrices that shows how to perform a meta-statistics analysis. This example reproduce some of the results published on [2]. 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/data/input_adja.txt` and `examples/data/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 %% Add the source to the matlab path
6 %Assuming that you run this script from examples directory
7 g = genpath('../'); addpath(g);
8 close all;
```

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)`.

```
%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];
%Notice that 1 in col_ids not necessearly corresponds to 1's in row_ids.
col_ids = [1 1 5 5];
```

3.1.7 Creating a Bipartite object from `MATLAB®` data

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

```
bp = Bipartite(matrix);
bp.row_labels = row_labels;
bp.col_labels = col_labels;
bp.row_class = row_ids;
bp.col_class = col_ids;
```

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/data/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:

```
84  bp = Reader.READ_BIPARTITE_MATRIX('input_matrix.txt');
85  % We need to add labels and classification ids by ourselves
86  bp.row_labels = row_labels;
87  bp.col_labels = col_labels;
88  bp.row_class = row_ids;
89  bp.col_class = col_ids;
```

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/data/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:

```
112  bp = Reader.READ_ADJACENCY_LIST('input_adja.txt.');
113  % Wee need to add classification ids by ourselves
114  bp.row_class = row_ids;
115  bp.col_class = col_ids;
```

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) [3]. 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/data/moebus_data.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_data.mat;
```

The loaded data contains the bipartite adjacency matrix of the Moebus and Nattkemper study [4], 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 `community` of the `Bipartite` object (`bp.community`). 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 [1] for bipartite networks using different approaches. Only the Newman algorithm return the same result. The other two perform at some point random module pre-assigments, and by consequence they may not 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.community = 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.community.DoKernighanLinTunning = true; % Default value
```

We need to calculate the modularity explicitly by calling:

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

If `Options.PRINT_RESULTS` is true, the last call will print the next lines:

```
Modularity:
    Used algorithm:                  LeadingEigenvector
    N (Number of modules):                           48
    Qb (Standard metric):                        0.7956
    Qr (Ratio of int/ext inter):                 0.8348
```

If we are interested in node module indexes too, we can use `bp.community.row_modules` and `bp.community.col_modules`. We can also access directly the modularity values by calling `bp.community.Qb` or `bp.community.Qr` as the next example:

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

```
The modularity value Qb is 0.795611
The fraction inside modules Qr is 0.834835
```

The value $0 \leq Q_b \leq 1$ is calculated using the standard bipartite modularity function (introduced by Barber) [1] while the value Q_r is an a posteriori represents the fraction of interactions that fall inside modules [5].

3.2.5 Calculating Nestedness

The nestedness algorithm is encoded in the property `nestedness` of the `Bipartite` object (`bp.nestedness`). Currently, two algorithms (metrics) are available:

1. Nestedness Temperatur Calculator NTC (`NestednessNTC.m`)
2. NODF (`NestednessNODF.m`)

Contrary to modularity (where each algorithm optimizes the same metric), these algorithms use different metrics to calculate nestedness. Therefore, the statistical significance of a network will depend not only in which null model but also in which metric (algorithm) is used. As the modularity case, the default

nestedness algorithm that `BiMat` uses is specified in `Options.NESTEDNESS_ALGORITHM`. The user can also switch the algorithm dinamically as we show for modularity. However, here we will just use the default algorithm by calling:

```
86 bp.nestedness.Detect();
```

As the modularity case, `BiMat` will return the next output if `Options.PRINT_RESULTS` is true:

```
Nestedness NODF:
    NODF (Nestedness value):                    0.0341
    NODF (Rows value):                          0.0368
    NODF (Columns value):                       0.0293
```

Finally the user can access directly the value of nestedness as in the following line:

```
93 fprintf('The Nestedness value is %f\n', bp.nestedness.N);
```

```
The Nestedness value is 0.034053
```

To finish this section, we can summarize both modularity and nestedness results by calling:

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

```
Modularity:
    Used algorithm:                  LeadingEigenvector
    N (Number of modules):                           48
    Qb (Standard metric):                        0.7956
    Qr (Ratio of int/ext inter):                 0.8348
Nestedness NODF:
    NODF (Nestedness value):                     0.0341
    NODF (Rows value):                           0.0368
    NODF (Columns value):                        0.0293
```

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 (otherwise default colors will be used):

```
105 figure(1);
106 % Matlab command to change the figure window;
107 set(gcf,'Position',[0   72   1751   922]);
108 bp.plotter.font_size = 2.0; %Change the font size of the rows and labels
109 % Use different color for each kind of interaction
110 bp.plotter.use_type_interaction = true; %
111 bp.plotter.color_interactions(1,:) = [1 0 0]; %Red color for clear lysis
112 bp.plotter.color_interactions(2,:) = [0 0 1]; %Blue color for turbid spots
113 bp.plotter.back_color = 'white';
114 % After changing all the format we finally can call the plotting function.
115 bp.plotter.PlotMatrix();
```

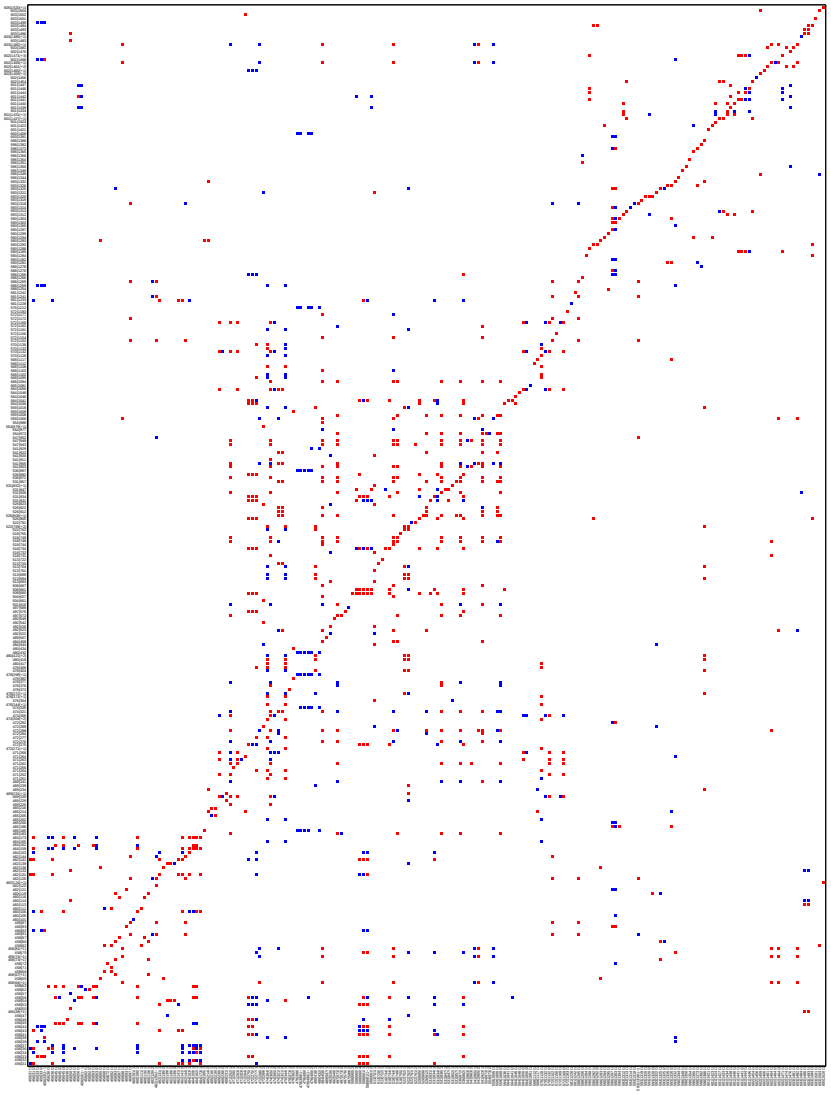


Figure 2: Original sorted matrix. Blue and red cells represent different strengths of infection between virus and bacteria. Rows and columns represent bacteria and phages, respectively.

For plotting the 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.

```
figure(2);
% Matlab command to change the figure window;
set(gcf,'Position',[0+50    72   932   922]);
bp.plotter.use_isocline = true; %The NTC isocline will be plotted.
bp.plotter.isocline_color = 'red'; %Decide the color of the isocline.
bp.plotter.PlotNestedMatrix();
```

For plotting the modularity sort, lets use the example to introduce the user to an interesting modularity property which is `optimize_by_component`. This property forces the modularity algorithms to optimize modularity in each component:

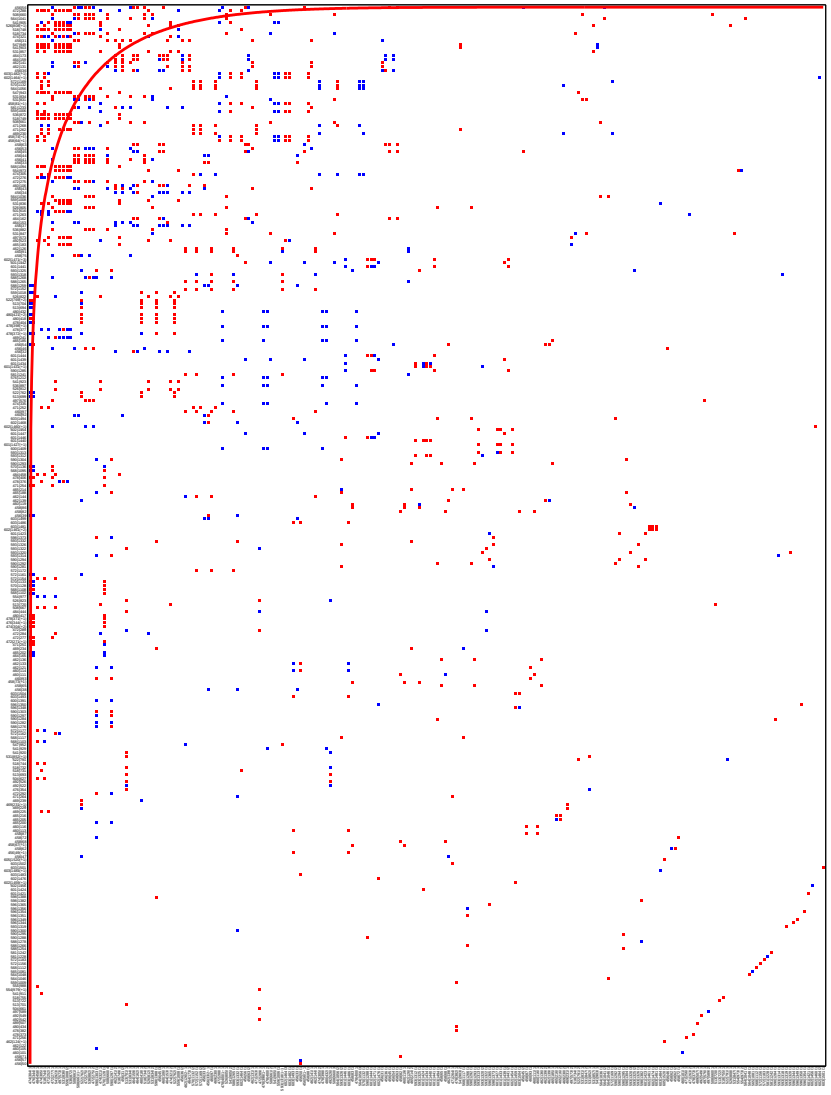


Figure 3: Nested sorted matrix. Blue and red cells represent different strengths of infection between virus and bacteria. In a perfectly nested pattern of the same fill than the current matrix, all the interaction cells will lay above the isocline (red line).

```
% independently of each other:
figure(3);
% Matlab command to change the figure window;
set(gcf,'Position',[0+100    72   1754    922]);
% First, lets optimize at the total matrix (default behavior)
subplot(1,2,1);
bp.community = LPBrim(bp.matrix); %Uses LPBrim algorithm
bp.plotter.use_isocline = true; %Although true is the default value
bp.plotter.PlotModularMatrix();
title(['$Q = $',num2str(bp.community.Qb),' $c = $', num2str(bp.community.N)],...
    'interpreter','latex','fontsize',23);
%
%Now, we will optimize at the graph component level.
```

```
subplot(1,2,2);
bp.community = LPBrim(bp.matrix);
bp.community.optimize_by_component = true; % optimize by components
bp.plotter.PlotModularMatrix();
title(['$Q = $',num2str(bp.community.Qb),' $c = $', num2str(bp.community.N)],...
    'interpreter','latex','fontsize',23);
% Move right panel to the left
set(gca,'position',get(gca,'position')-[0.07 0 0 0]);
```

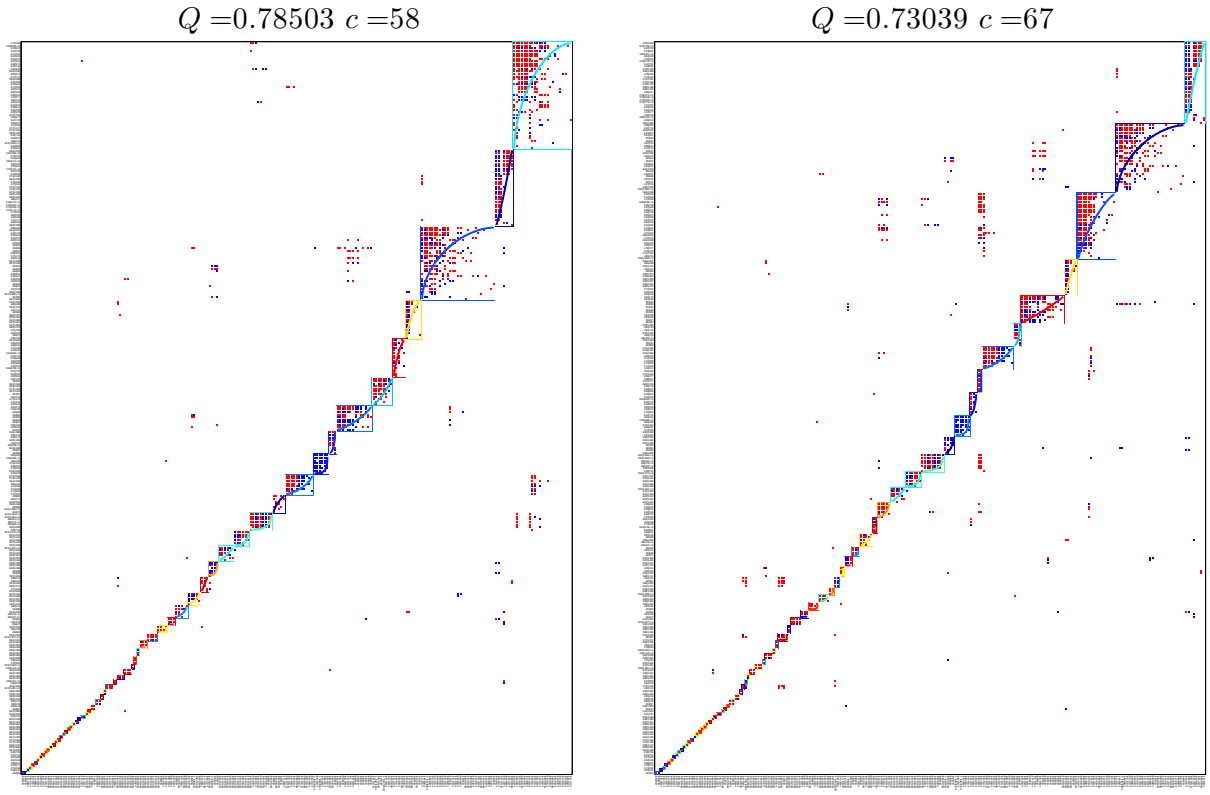


Figure 4: Modular sorting in matrix layout. Blue and red cells represent different strengths of infection between virus and bacteria. Each block represent a different module. Left panel shows the default behavior (optimize at the total matrix), while right panel shows the component optimization. Generally the second case will have better resolution but smaller global modularity value. LPBrim was used for optimizing the modularity function in both cases.

Finally, the user can play with `use_isocline`, `use_type_interactions`, `use_type_interaction`, and `use_module_format` to create interesting visualizations:

```
figure(4);
set(gcf,'Position',[0+150    72   1754    922]);
% First, lets come back to use the LeadingEigenvector algorithm
bp.community = LeadingEigenvector(bp.matrix);
%
subplot(1,2,1);
bp.plotter.use_isocline = false;
bp.plotter.use_type_interaction = false;
bp.plotter.PlotModularMatrix();
%
```

```
subplot(1,2,2);
% Isocline and divisions will not have the same color than modules
bp.plotter.use_module_format = false;
bp.plotter.use_isocline = true;
bp.plotter.isocline_color = 'red';
bp.plotter.division_color = 'red';
bp.plotter.back_color = [0 100 180]/255;
bp.plotter.cell_color = 'white';
bp.plotter.PlotModularMatrix();
% Move right panel to the left
set(gca,'position',get(gca,'position')-[0.07 0 0 0]);
```

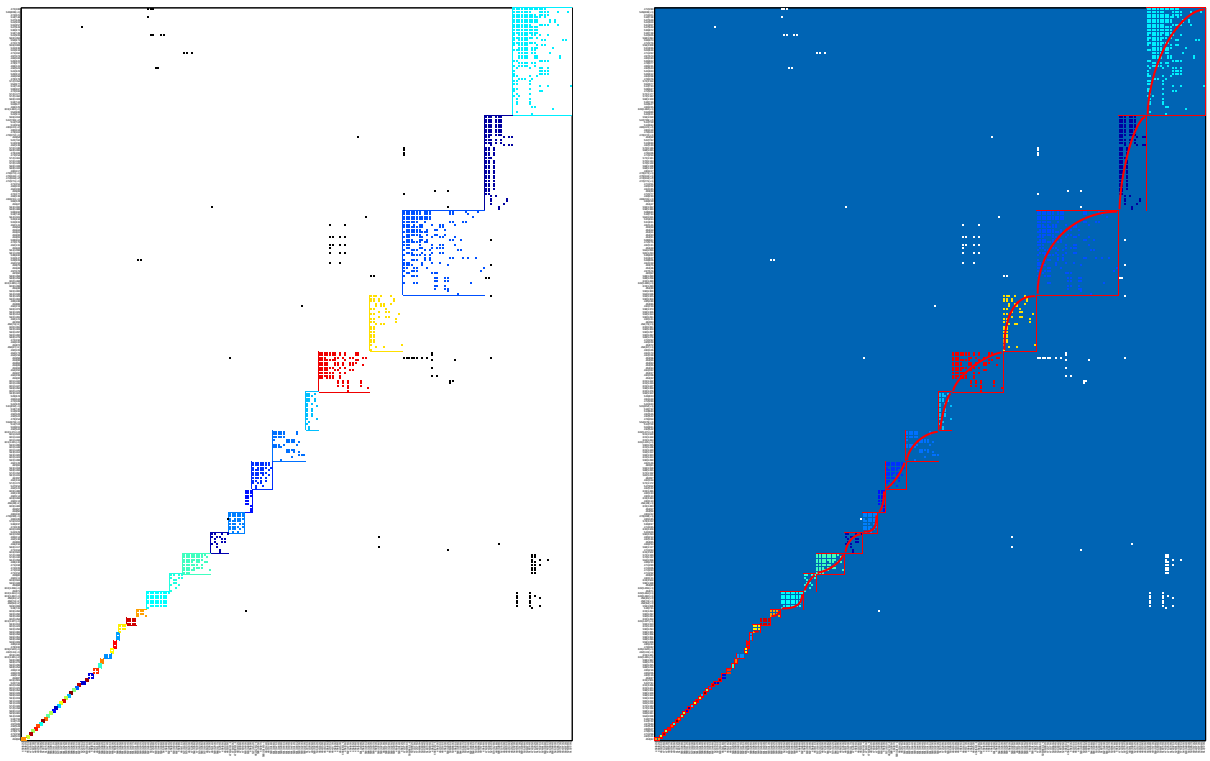


Figure 5: Modular sorting in matrix layout. The user can play with the plotter properties in order to create interesting matrix layout formats. LeadingEigenvector was used for optimizing the modularity function in both cases.

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(6);
% Matlab command to change the figure window;
set(gcf,'Position',[19+800    72   932   922]);
bp.plotter.PlotModularGraph();
```

Figure 6: 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.

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 , $P_{ij} = E/(mn)$ – the connectance of the network is respected, but not the number of interactions in which each node is involved.

NullModels.AVERAGE , $P_{ij} = (k_i/n + d_j/m)/2$ – the connectance, and the expected number of interactions in which each node is involved, are respected

NullModels.AVERAGE_COLS , $P_{ij} = k_i/n$ – the connectance, and the expected number of interactions of row nodes, are respected

NullModels.AVERAGE_ROWS , $P_{ij} = d_j/m$ – the connectance, and the expected number of interactions of column nodes, are respected

NullModels.FIXED - this model creates random matrices that respect the total sums of each row and column of the bipartite adjacency matrix. It uses a random swapping algorithm.

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 and nestedness 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:

```
% Both calls print the same information
bp.printer.PrintStructureStatistics(); %Print the statistical values
bp.statistics.Print(); %Print the statistical values
```

```
Modularity
     Used algorithm:                LeadingEigenvector
     Null model:                NullModels.EQUIPROBABLE
     Replicates:                                   100
     Qb value:                                  0.7951
         mean:                                  0.4403
         std:                                   0.0050
         z-score:                              71.2951
         percentil:                           100.0000
     Qr value:                                  0.8333
         mean:                                  0.1082
         std:                                   0.0214
         z-score:                              33.8450
         percentil:                           100.0000
Nestedness
```

```
Used algorithm:                    NestednessNODF
Null model:               NullModels.EQUIPROBABLE
Replicates:                                  100
Nestedness value:                         0.0341
    mean:                                 0.0240
    std:                                  0.0006
    z-score:                             16.5544
    percentil:                          100.0000
```

The printed information is as follows:

- `Used algorithm`: The algorithm that was used for calculating the metric.
- `value`: value to be tested (*e.g.* nestedness or modularity).
- `replicates`: number of replicates used during testing.
- `mean`: mean of the replicate values.
- `std`: standard deviation of the replicate values (note that distributions of network values are not necessarily well described by a normal distribution).
- `zscore`: The z-score of `value` assuming that the replicate values represent the entire population.
- `percentile`: The percentage of replicate values that are smaller than `value`.

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:

```
250 % 100 random matrices using the EQUIPROBABLE null model.
251 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:

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

```
Network,     Qb,Qb mean,Qb z-score,Qb percent,       Qr,   Qr mean,Qr z-score,Qr percent,      N, N mean,N z-score,N percent
      1,0.28198,0.25424,     3.052,       100,   0.4052,-0.0090706,    8.6901,       100,0.53237,0.29431,  18.7592,      100
      2,0.38344,0.28069,      8.11,       100,  0.59091,  0.082727,    8.7795,       100,0.37935,0.33732,   1.8029,       96
      3,0.36091, 0.2856,    8.3161,       100,   0.3361,  0.022075,    5.3604,       100,0.36098,0.24427,   9.3941,      100
      4,  0.495,0.40179,      4.61,       100,  0.66667,   0.29133,    5.7683,       100,0.34029,0.22005,   5.0641,      100
      5,0.34295,0.25917,    6.5907,       100,   0.5614,  0.042982,    9.5962,       100,0.42548,0.37412,   1.9607,       98
      6,0.37778,  0.311,    2.1714,       100,      0.6,   0.28533,    2.0792,        98,0.38468,0.37225,  0.21945,       55
      7,0.49146,0.33043,    7.2185,       100,  0.91837,   0.20245,     8.548,       100,0.25255,0.33147,   -1.695,        3
      8,0.16272,0.18724,   -1.7281,         3,  0.30769,      0.15,    1.1627,        83,0.76951,0.56351,   3.8483,      100
      9,0.22222,0.17667,    1.5673,        90,  0.55556,   0.38444,    1.4824,        96,0.46154,0.56869,  -1.0421,       14
     10,0.10526,0.11098,  -0.45415,        16,  0.21053,   0.21895,   -0.1065,         7,0.66822, 0.6694,-0.013412,       45
     11,0.48148,0.37346,    2.1981,        99,  0.77778,   0.41222,    2.5832,        98, 0.2483,0.31196,  -1.0607,       10
     12,0.13173,0.15876,   -2.3413,         1,-0.052632,    0.1193,   -1.2618,         6,0.74994,0.60985,   2.4645,      100
     13,    0.4,0.30636,    1.9125,        97,  0.86667,   0.40667,    2.7075,        99,0.41398,0.40417, 0.092677,       55
     14,0.06414,0.11336,   -4.8795,         0,  0.55102,    0.2102,    3.7813,       100,0.38596,0.66925,  -3.7248,        1
     15,   0.25,0.20938,   0.97224,        81,      0.5,      0.31,   0.73365,        44,0.44444,0.61944, -0.84654,       11
     16,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     17,0.16327,0.14531,   0.88196,        44,  0.42857,   0.42857,       NaN,         0,0.66667,0.66667,  0.99499,        0
     18,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     19,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     20,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     21,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     22,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     23,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     24,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     25,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     26,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     27,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     28,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     29,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     30,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     31,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     32,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     33,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     34,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     35,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     36,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     37,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     38,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     39,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     40,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     41,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     42,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     43,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     44,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     45,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     46,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     47,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     48,      0,      0,       NaN,         0,        1,         1,       NaN,         0,      0,      0,       NaN,        0
     49,      0,      0,       NaN,         0,        1,         1,       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 5. 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 have the same structure than the one being analyzed. This makes the standard deviation to be 0, and therefore the z-score to be $0/0 =$ `NaN`.

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 [3]. 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.

```
266  %Using the labeling of bp and 1000 random permutations
267  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,     2.4873, -1.9465,    2.6
     2,     1.9722, -1.5477,    4.6
     3,     2.2497, -5.9225,      0
     4,     1.4791, -6.0072,      0
     5,     1.8174, -5.9413,      0
     6,     1.6094, 0.57569,   27.3
     7,     1.0906, -8.7094,      0
     8,     1.0042,  -6.007,      0
     9,     1.4942, -2.9247,    0.5
    10,     1.7479,-0.41875,   12.4
    11,    0.45056, -8.4223,      0
    12,     1.7678, -2.8444,    0.5
    13,     1.3322, -1.2948,    2.3
    14,      1.677, -2.4976,    0.5
    15,     1.0397, -2.0919,    0.6
    16,     1.0986, 0.28398,    7.8
    17,          0,     NaN,      0
    18,          0,     NaN,      0
    19,          0,     NaN,      0
    20,    0.63651, -2.8875,      0
    21,          0,     NaN,      0
    22,          0,     NaN,      0
    23,          0, -5.6834,      0
    24,    0.69315, 0.20402,      4
    25,          0,     NaN,      0
    26,          0, -4.8339,      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, -5.8889,      0
```

Using a one tailed p-value of 0.05 we can see that 1-5,7-9,11-15 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.

Before finishing this example, we must say that in order to analyze the statistical significance of nestedness and modularity of the internal modules, what `BiMat` is really performing is a meta analysis. This functionality is encoded in the class `MetaStatistics`. This class makes use of the class `MetaStatisticsPlotter` to create interesting visualization of the matrices being analyzed. We show how to use this functionality in the next example.

3.3 BiMat - Meta-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/phage_bacteria_meta_analysis.mat` data file.

3.3.1 Contents

- Add the source to the `MATLAB®` path
- Creating a MetaStatistics object
- Perform an statististical analysis in the set of matrices
- Using a `MetaStatistics` 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 phage_bacteria_matrices.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 [2].

3.3.3 Creating a MetaStatistics 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 mstat = MetaStatistics(phage_bacteria_matrices.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 [2], 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 mstat.replicates = 100; %How many random networks we want for each matrix
31 mstat.null_model = @NullModels.EQUIPROBABLE; %Our Null model
32 mstat.modularity_algorithm = @AdaptiveBrim; %Algorithm for modularity.
33 mstat.nestedness_algorithm = @NestednessNTC; %Algorithm for nestedness.
34 mstat.do_community = 1; % Perform Modularity analysis (default)
35 mstat.do_nestedness = 1; % Perform Nestedness analysis (default)
36 mstat.names = phage_bacteria_matrices.name;
37 mstat.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 `DoMetaAnalysis` 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 nested matrices: %i\n',sum(mstat.N_values.percentile ≥ 97.5));
47 fprintf('Number of modular matrices: %i\n',sum(mstat.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 mstat.Print();
```

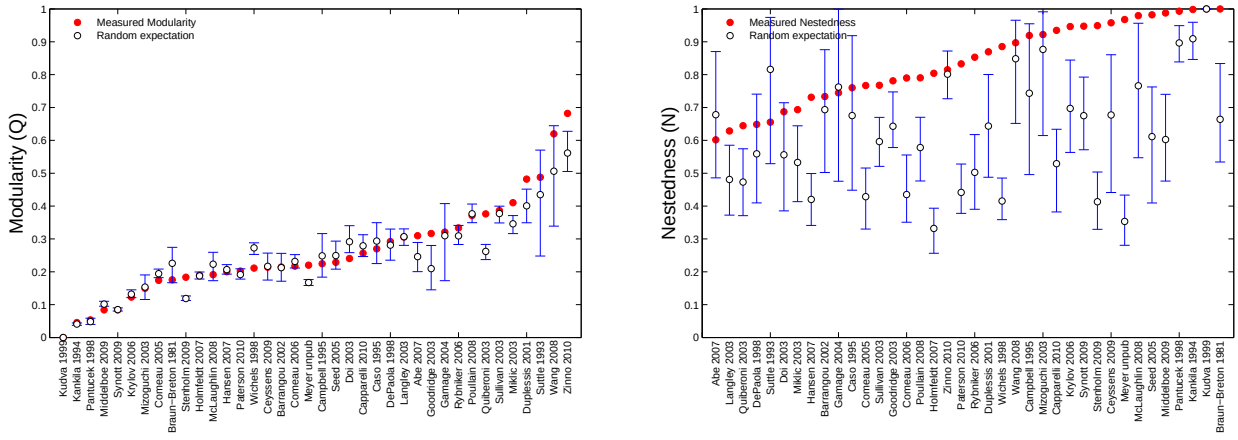
```
Network,      Qb, Qb mean,Qb z-score,Qb percent,       Qr,  Qr mean,Qr z-score,Qr percent,      N, N mean,N z-score,N percent
      1, 0.30992, 0.24599,    2.5886,        99,  0.81818,  0.16636,    3.9627,       100,0.60166,  0.678, -0.75945,       24
      2,  0.2144, 0.21253,   0.10036,        54,     0.44,   0.1024,    1.9222,        92,0.73351,0.69341,  0.37149,       66
      3, 0.17556, 0.22576,   -1.6619,         4,  0.13333,  0.12467,  0.055595,        55,0.99999,0.66398,   3.9396,      100
      4, 0.22449, 0.24862,  -0.86772,        12, -0.14286,  0.20143,   -1.9187,         0,0.91927,0.74326,   1.3936,       91
      5, 0.25652, 0.27901,   -1.2848,         8, 0.074074,  0.11037,  -0.35654,        32,0.93491,0.52914,   6.1977,      100
      6,  0.2699,  0.2936,  -0.68086,        26,  0.76471,     0.24,    3.0418,        99,0.76003,0.67532,  0.72106,       79
      7, 0.21403, 0.21669,    -0.143,        40,  0.58621,  0.10552,    2.9076,        99,0.95766,0.67737,   2.5286,      100
      8,   0.174, 0.19425,   -3.1699,         0, -0.15789,-0.064737,   -1.2478,         1,0.76686,0.42858,   7.2945,      100
      9, 0.21718, 0.23151,   -1.4067,         9,-0.033898,-0.028136, -0.075524,        31,0.78982,0.43517,   6.8795,      100
     10, 0.29191,  0.2811,   0.48576,        67,  0.28205,  0.17436,   0.85949,        70,0.64872,0.55895,   1.2204,       91
     11, 0.24033, 0.29148,   -2.5914,         0,  0.21951,  0.16439,   0.44398,        64,0.68685,0.55609,   1.6116,       93
```

```
12,  0.4821, 0.40073,    2.8901,      100,  0.67568,  0.27568,    3.8592,      100,0.86948,0.64357,   2.7302,      100
13, 0.32099, 0.30988,   0.20222,       45,  0.11111,  0.34222,    -1.057,        3,0.74467,0.76221, -0.13176,       46
14, 0.31667, 0.20973,    3.2725,      100,  0.63333,  0.38867,    4.0298,      100,0.78116,  0.643,   3.0432,      100
15, 0.20023, 0.20669,  -0.79464,       26,  0.20548,-0.044384,    3.3436,      100, 0.7313,0.42032,   6.9832,      100
16, 0.18956, 0.18757,   0.37716,       65,-0.033493, -0.18445,    2.4474,       99,0.80411,0.33195,   14.855,      100
17,0.045608,0.040801,    2.5875,       99, -0.43931,-0.024509,    -6.2235,       0,0.99822,0.90924,   3.0789,      100
18,  0.1231, 0.13176,   -1.3919,        8,  0.17808,-0.050411,     1.773,       88,0.94654,0.69728,   2.9167,      100
19,       0,       0,       NaN,       0,       1,         1,       NaN,        0,0.99998,0.99998, -0.99499,        0
20, 0.30568, 0.30689, -0.093846,       47, 0.070707, 0.065455,   0.071881,      56,0.62854,0.48084,   2.6413,       99
21, 0.19136, 0.22327,    -1.452,        7,  0.22222,  0.14889,    0.42528,      66,0.97959,0.76606,   1.8373,       99
22, 0.22015, 0.16726,    10.8245,     100,  0.36306, -0.15618,    9.6282,      100,0.96778,0.35335,  15.2339,      100
23, 0.08406, 0.10169,   -4.5703,        0,  0.23762, -0.12762,    4.6519,      100,0.98762,0.60238,   6.0343,      100
24,  0.4102, 0.34589,    4.2458,      100,  0.82857,  0.13457,    8.9292,      100,0.69338,0.53292,   2.6264,       99
25, 0.14966, 0.15358,  -0.24209,       32,  0.2381,      0.1,    0.65528,      51,0.92211,0.87686,  0.42745,       50
26,0.053624,0.048243,    1.1569,       89, 0.062112, 0.058261,   0.040003,       6,0.99319,0.89658,   3.6763,      100
27, 0.20209, 0.19155,    1.2751,       90, 0.093633, 0.064644,   0.52737,       67,0.83274,0.44163,  10.5648,      100
28, 0.37139,  0.3764,  -0.31136,       45,  0.14019, 0.095701,    0.5921,       67,0.79046,0.57808,   4.1357,      100
29, 0.37622, 0.26208,    9.0336,      100,  0.70787, 0.039101,     8.168,      100,0.64449,0.47324,   2.7723,      100
30, 0.33347, 0.30927,    1.7178,       96,  0.54286, 0.095143,    5.5966,      100,0.85292,0.50253,   6.0303,      100
31, 0.22893, 0.24931,  -0.99388,       14,-0.096774,  0.13032,    -1.8803,       0,0.98194,0.61131,   4.2027,      100
32, 0.18341, 0.11878,   18.4957,      100,  0.28161, -0.15667,    5.4021,      100,0.94908,0.41311,  12.7962,      100
33,  0.3866, 0.37729,   0.76166,       77,  0.32432, 0.059189,    3.8055,      100,0.76762,0.59612,   4.2834,      100
34,  0.4876, 0.43471,   0.66412,       71,  0.45455,  0.47273, -0.091987,       26,0.65529,0.81606,  -1.3744,       14
35,0.084203,0.085013,    -0.312,       40, -0.24638, -0.14589,  -0.93777,       0,0.94752,0.67504,   4.6206,      100
36, 0.61983,  0.5062,    1.4912,       93,  0.81818,  0.59818,    1.3411,       80,0.89709, 0.8485,  0.56436,       68
37, 0.21137, 0.27294,   -7.4452,        0, -0.20755, -0.12006,    -1.6681,       4, 0.8854,0.41533,  14.0797,      100
38, 0.68222, 0.56131,    3.9645,      100,  0.71429,  0.4302,     3.8722,      100,0.81505,0.80138,  0.35774,       60
```

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  mstat.plotter.font_size = 10; %Size for x-labels.
59  figure(1);
60  mstat.plotter.PlotModularValues();
61  figure(2);
62  mstat.plotter.PlotNestednessValues();
```



(a) Modularity statistics

(b) Nestedness (NTC) statistics

Figure 7: 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.

In addition the user can also plot the data in either graph or matrix layout. Here we show for graph nested layout and modular matrix layout. As in the case of a single network, is possible to specify some of the most fundamental format properties.

```
68  mstat.plotter.p_value = 0.05; %p-value for color labeling
69  % Plot of nested graphs
70  mstat.plotter.bead_color_rows = 'blue'; %Color of row nodes
71  mstat.plotter.bead_color_columns = 'red'; %Color of column nodes
72  mstat.plotter.link_width = 0.5; %Edge width
73  mstat.plotter.use_isocline = false; %Do no show isocline inside modules
74  figure(3);
75  mstat.plotter.PlotModularMatrices(5,8); %Use a grid of 5 x 8
76  %Plot of modular matrices
77  figure(4);
78  mstat.plotter.PlotNestedGraphs(5,8);
```

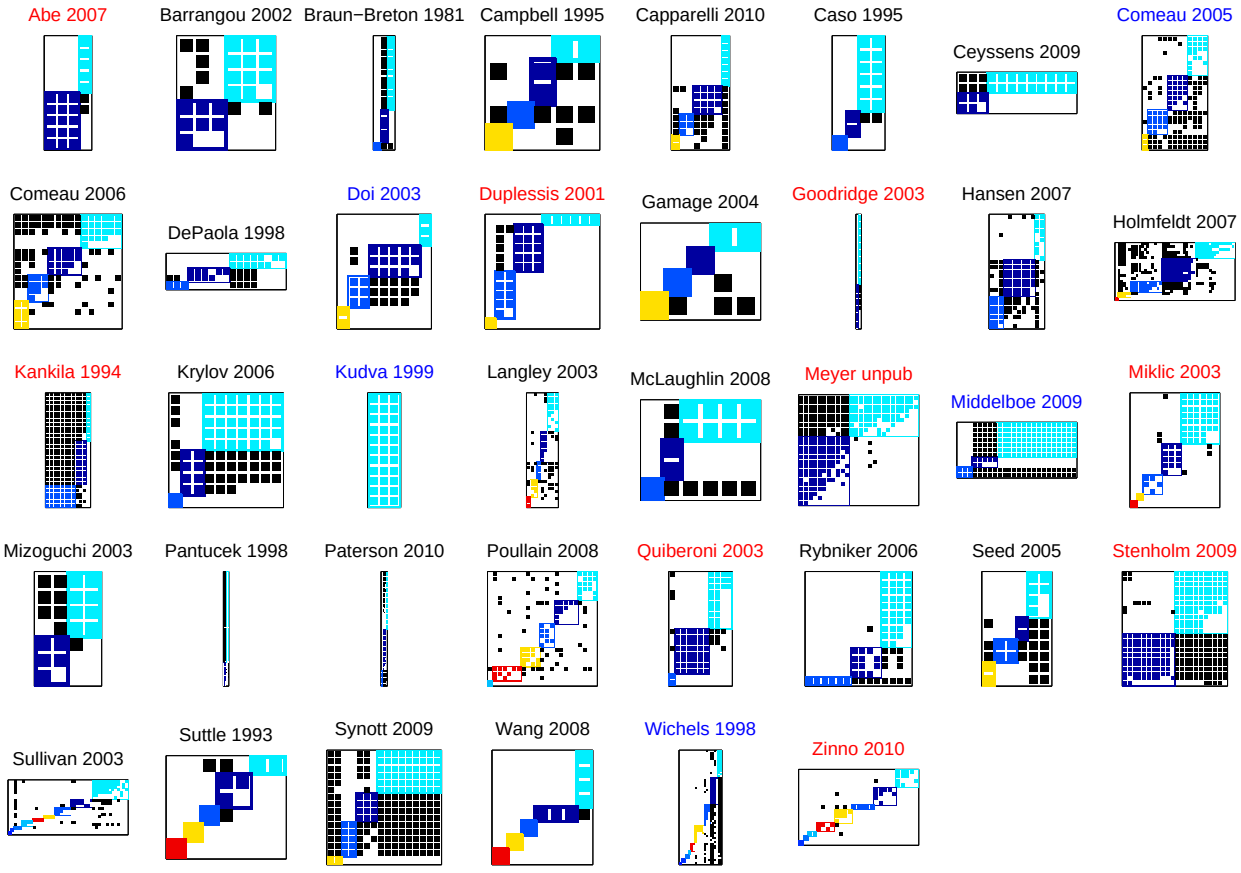


Figure 8: The meta-set collected on Flores et al [2] plotted using the modularity algorithm of the `BiMat` library. Red and blue labels represent significant modularity ($p \geq 0.975$) and anti-modularity ($p \leq 0.275$), respectively. For bibliographic information about these matrices see [2].

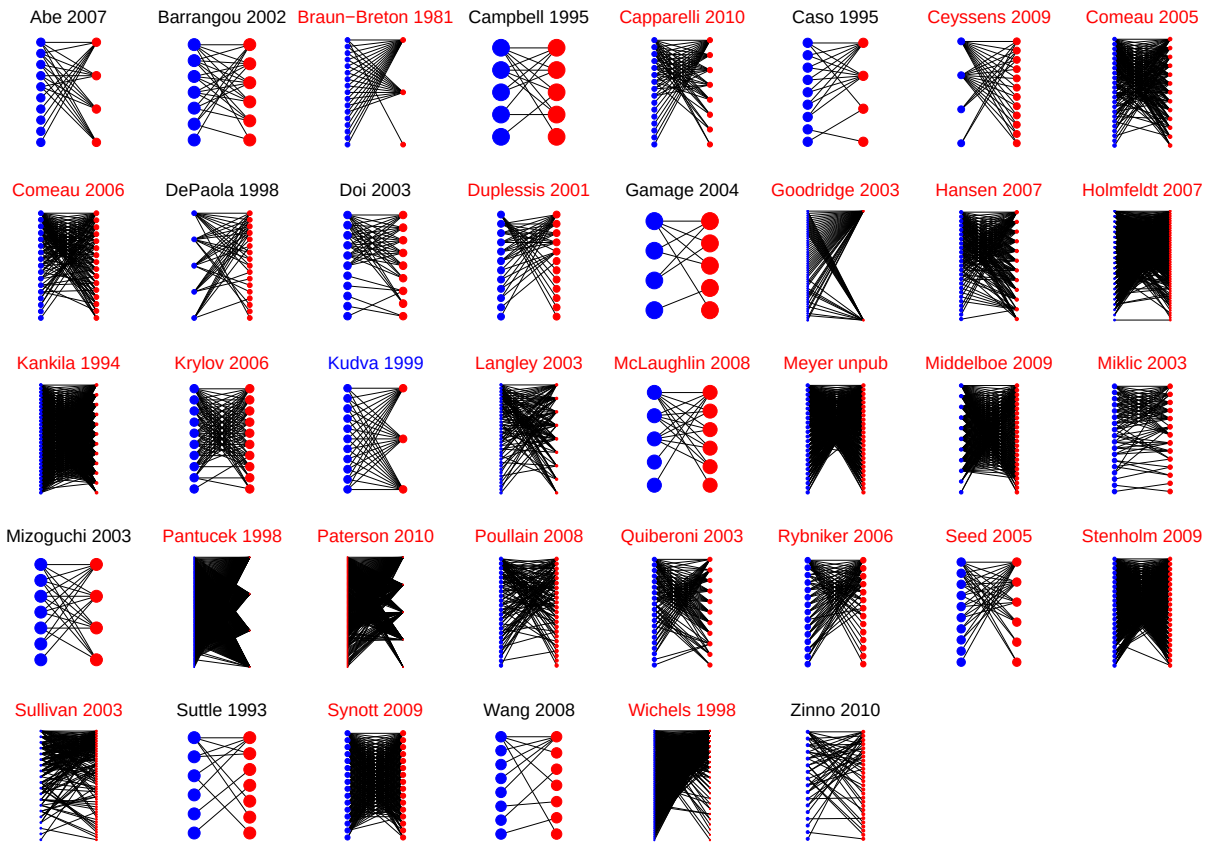


Figure 9: NODF nestedness values of a set of 38 matrices of phage-bacteria networks. A two-tail p-value of 0.05 was used for labeling the names. Blue and and red lebels represent anti and statistical significance, respectively. Notice that this Figure shows an smaller number of nested matrices than the NTC plot of the previous figure.

References

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

[2] 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.

[3] 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.

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

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