BiMat : Start Guide

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

July 16, 2013

1 System Requirements

- MATLAB® 2009b or superior. The software may work in previous versions, however we could not test in those versions.
- The user is expected to have basic MATLAB® knowledge.
- If parallel computing will be used, the Parallel computing toolbox need to be installed in the user MATLAB® installation.

2 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 calculation.
- Diversity calculation using Shannon and Simpson's indexes.
- Different null models for the creation of random bipartite networks.
- Statistics of the network.
- Internal statistics of the modules (multi-scale analysis).
- Group statistical analysis (analysis of many networks).
- Parallel processing for improving the speed during a statistical analysis.
- Plotting in matrix or graph layouts.

3 Downloading and installing BiMat

You can also download the last stable version from: https://www.dropbox.com/s/eiblsvlb1hq2xkz/bimat_stable.zip. To install BiMat , copy the bimat_stable.zip file to a directory of interest and unzip it. Note the directory name of the BiMat library, as you will need to use it for configuration.
Additionally, you can have access to the most updated version from GitHub: https://github.com/cesar7f/BiMat or ask me (cesar.flores@gatech.edu) to send you an invitation to share my dropbox directory.

4 Pre-configuring BiMat

Before using BiMat you need to do an small configuration of the framework which consists of:

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

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

  You should replace bimat_directory_location with the full path to the directory in which you installed BiMat . You can also permanently add BiMat to your MATLAB® path using the menu File –>Set Path.

- (Recommended) Update the default parameters in the file main/Options.m according to the needs of the user. These parameters are used in the code whenever the user do not specify an analogous parameter in a function or property of a class instance. For example, for evaluating the statistics of the structure (modularity/nestedness) of a network, if the user do not specify the number of random matrices to be used for the null model, the code automatically will assume that the value will be Options.REPLICATES. Keep in mind that the current values were designed for getting quick results, which may not be the most accurate ones.

- (Optional) If you want to use the Parallel Computing features just type in the MATLAB® command line matlabpool;, which will open a certain number of workers according to your local MATLAB® configuration (See Parallel Computing documentation of your MATLAB® version for more information).

5 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).

- `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`).

6 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 [5].

- `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 [4]. However, using this template all the results can be reproduced with a little extra effort.

6.1 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.

6.1.1 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

6.1.2 Add the source to the `MATLAB`® path

```
%% Add the source to the matlab path
%Assuming that you run this script from examples directory
g = genpath('../'); addpath(g);
close all;
```

6.1.3 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.

6.1.4 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).

6.1.5 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 respectivally. 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.

6.1.6 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];
```

6.1.7 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;
```

6.1.8 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:

```
bp = Reading.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;
bp.row_class = row_ids;
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/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:

```
bp = Reading.READ_ADJACENCY_LIST('input_adja.txt.');
% Wee need to add classification ids by ourselves
bp.row_class = row_ids;
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.

6.2 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) [5]. 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.

6.2.1 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

6.2.2 Add the source to the MATLAB® path

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

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

```
load moebus_use_case.mat;
```

```
load moebus_use_case.mat;
```

The loaded data contains the bipartite adjacency matrix of the Moebus and Nattkemper study [6], 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.

6.2.3 Creating the Bipartite network object

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

We can print the general properties of the network with:

```
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
```

6.2.4 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):

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

We need to calculate the modularity explicitly by calling:

```
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:

```
fprintf('The modularity value Qb is %f\n', bp.modules.Qb);
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.

6.2.5 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:

```
bp.ntc.CalculateNestedness();
```

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

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

```
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
```

6.2.6 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:

```
figure(1);
% Matlab command to change the figure window;
set(gcf,'Position',[19    72   932   922]);
bp.plotter.font_size = 2.0; %Change the font size of the rows and labels
% Use different color for each kind of interaction
bp.plotter.use_type_interaction = true; %
bp.plotter.color_interactions(1,:) = [1 0 0]; %Red color for clear lysis
bp.plotter.color_interactions(2,:) = [0 0 1]; %Blue color for turbid spots
bp.plotter.back_color = 'white';
% After changing all the format we finally can call the plotting function.
bp.plotter.PlotMatrix();
```

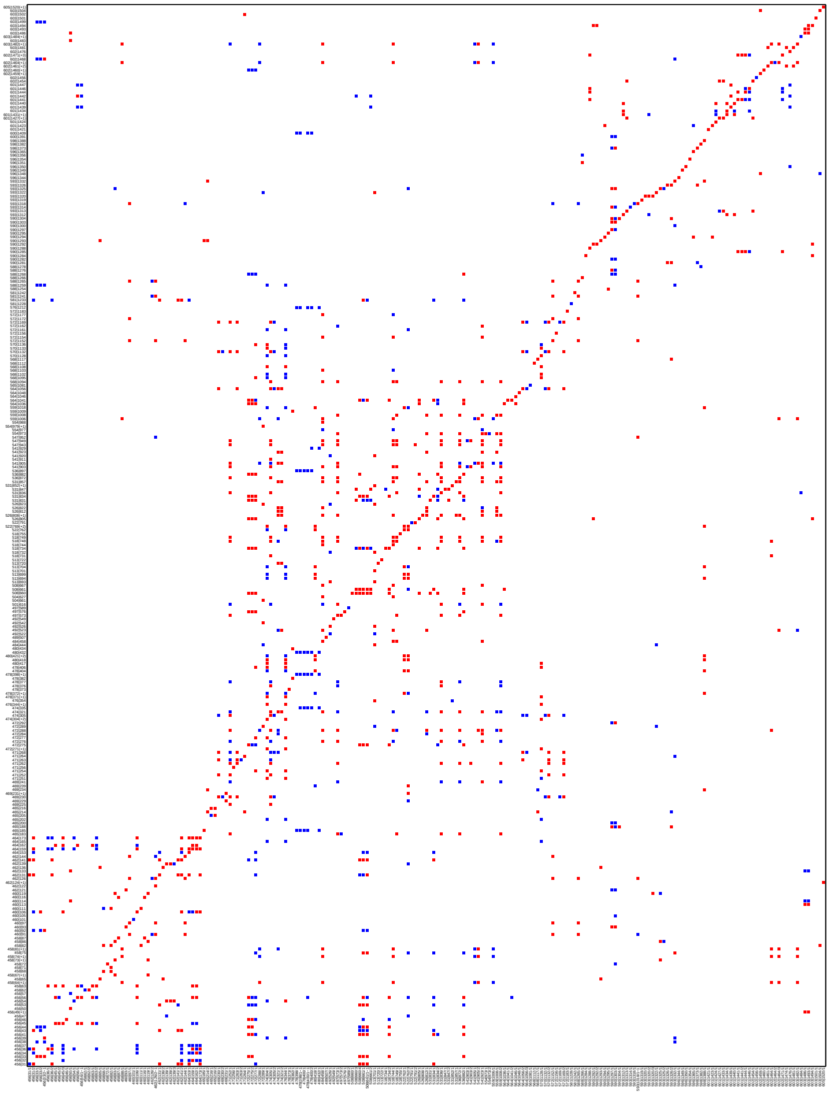

Figure 1: Original sorted matrix. Blue and red cells represent different strengths of infection between virus and bacteria. Rows and columns represent bacteria and phages, respectively.

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',[19+200    72    932    922]);
bp.plotter.use_isocline = true; %The isocline is used in the NTC algorithm.
bp.plotter.isocline_color = 'red'; %Decide the color of the isocline.
bp.plotter.PlotNestedMatrix();
```

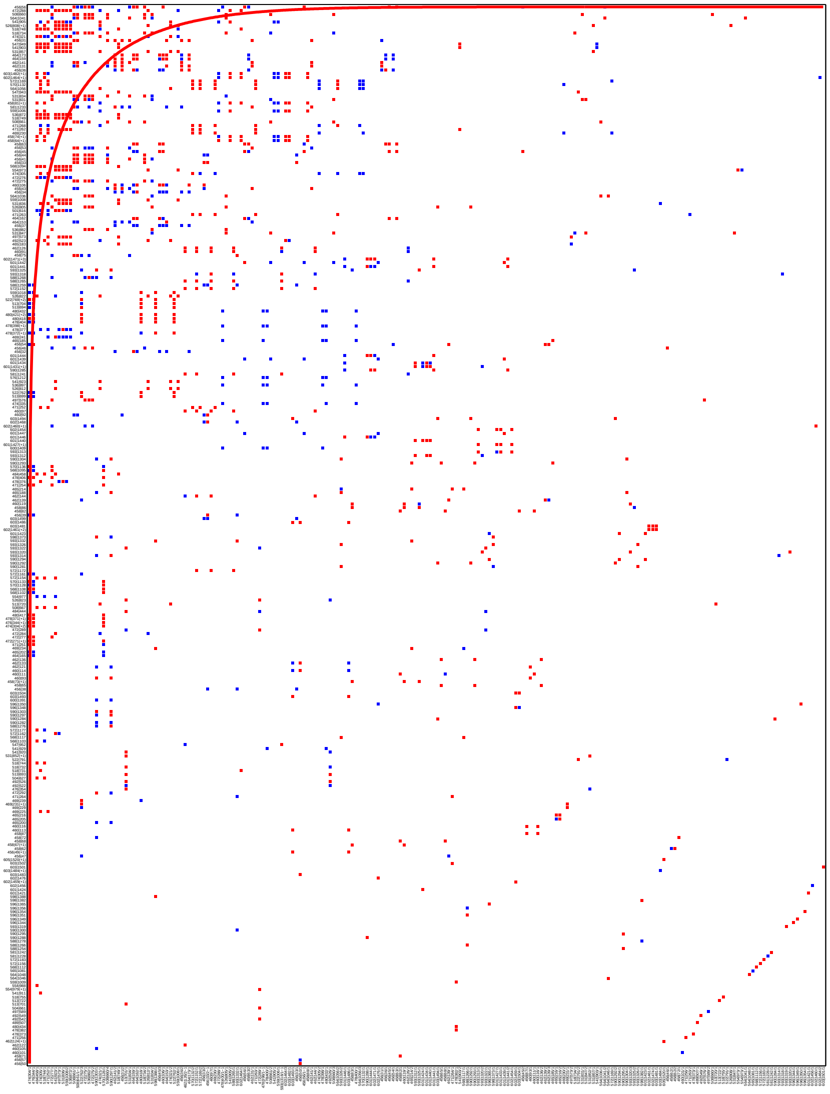

Figure 2: Nested sorted matrix. Blue and red cells represent different strengths of infection between virus and bacteria. In a perfectly nested pattern of the same fill than the current matrix, all the interaction cells will lay above the isocline (red line).

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',[19+400    72    932    922]);
% This will plot isoclines inside modules.
bp.plotter.plot_iso_modules = true;
bp.plotter.PlotModularMatrix();
```

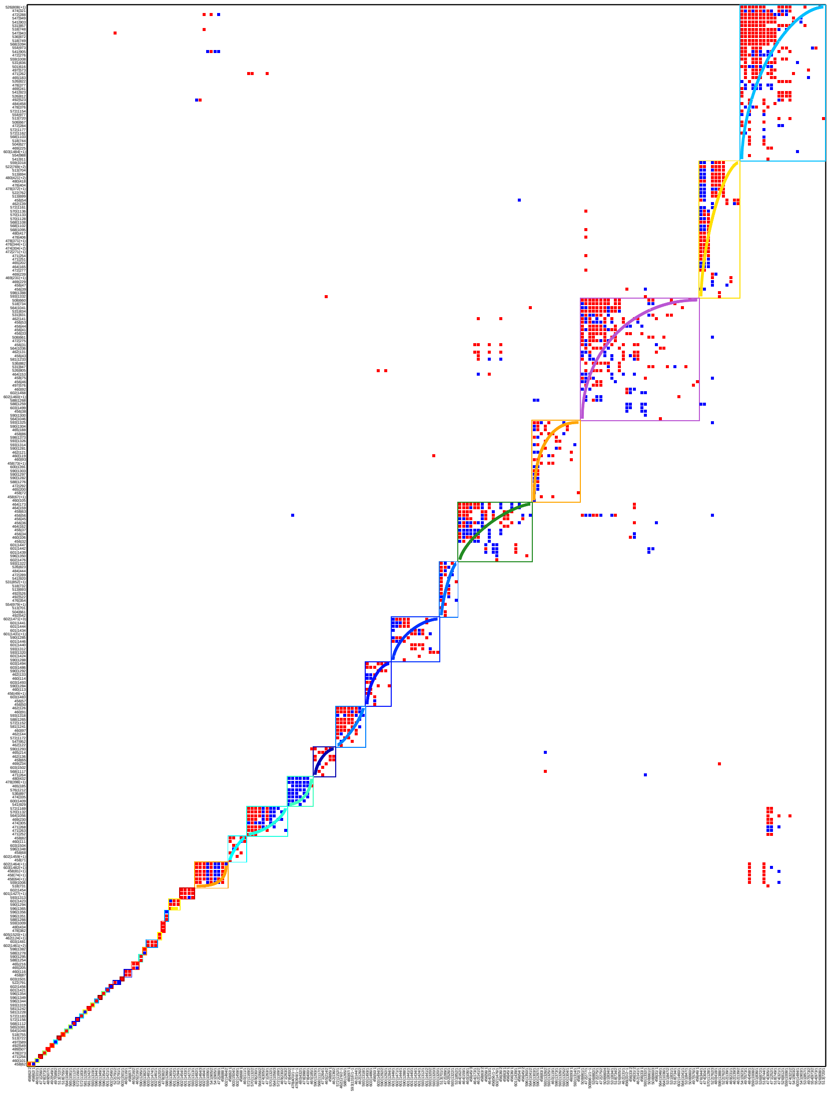

Figure 3: Modular sorted matrix. Blue and red cells represent different strengths of infection between virus and bacteria. Each block represent a different module.

6.2.7 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();
```

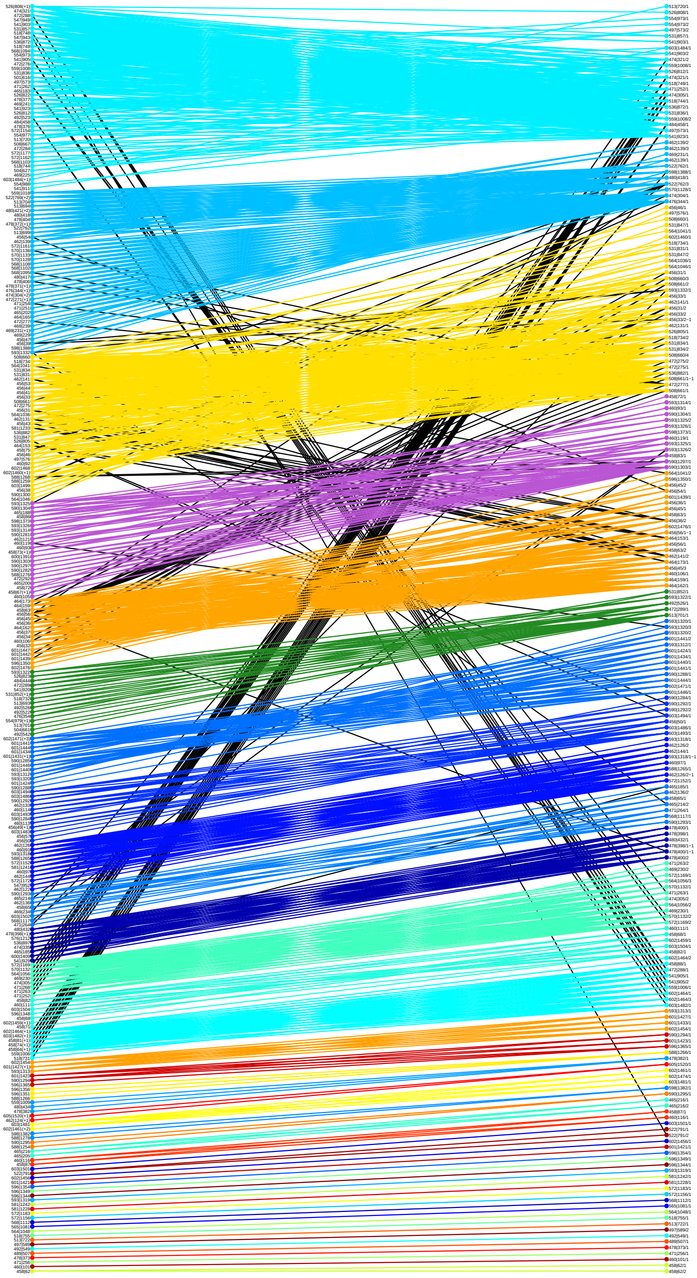

Figure 4: Modular graph layout. Nodes and interactions are colored according to the module they belong to. Black color is used for interaction across modules. Left and right side nodes represent bacteria and phages, respectively.

6.2.8 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 probability 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).

6.2.9 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 3. 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 have NaN values only 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 this results, is because this module is compose of only one column (phage) node and the matrix is of size 0×1.

NaN values are reported because the standard deviation used for calculating the z-score is 0. This happens when you have a fully connected network.

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 [5]. 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.

6.3 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.

6.3.1 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

6.3.2 Add the source to the MATLAB® path

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

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

```
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 [4].

6.3.3 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

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

6.3.4 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 [4], 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:

```
gp.replicates = 100; %How many random networks we want for each matrix
gp.null_model = @NullModels.EQUIPROBABLE; %Our Null model
gp.modul_class = @AdaptiveBrim; %Algorithm for modularity.
gp.do_temp = 1; % Perform NTC analysis (default)
gp.do_modul = 1; % Perform Modularity analysis (default)
gp.do_nest = 0; % Do not perform NODF analysis
gp.names = grouptesting.name;
gp.DoGroupTesting(); % 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

```
fprintf('Number of nested matrices: %i\n',sum(gp.tempvals.percent ≥ 97.5));
fprintf('Number of modular matrices: %i\n',sum(gp.qb_vals.percent ≥ 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:

```
gp.PrintResults();
```

```
Network,      Qb, Qb mean,Qb z-score,Qb percent,    NTC,NTC mean,NTC z-score,NTC percent
      1, 0.30992, 0.27824,    1.2157,        87,0.65025, 0.64688,   0.036295,         46
      2,  0.2144, 0.39928,   -5.6089,         0,0.80601, 0.69052,     1.3941,         87
      3, 0.17556, 0.23938,   -2.2402,         1,0.99985, 0.66045,     4.4259,        100
      4, 0.21429, 0.39515,   -3.4546,         0, 0.9352, 0.76606,      1.959,        100
      5, 0.25652, 0.30519,   -2.6785,         0,0.94378, 0.54503,     5.3161,        100
      6,  0.2699, 0.50294,   -3.5986,         0,0.82486, 0.80259,    0.29428,         58
      7, 0.21403, 0.32955,   -3.9932,         0,0.98175, 0.63525,     4.0427,        100
      8,   0.174, 0.24596,   -8.6057,         0,0.79901, 0.42449,     7.4005,        100
      9, 0.20425, 0.34843,  -10.4438,         0,0.85648,   0.528,     6.6521,        100
     10, 0.29191, 0.28168,   0.45852,        70,0.63681, 0.57901,    0.70123,         78
     11, 0.24033, 0.38286,   -5.6806,         0, 0.8345, 0.62663,     2.5466,        100
     12,  0.4821, 0.40291,    3.0713,        99,0.88313, 0.66411,     3.3618,        100
     13, 0.32099, 0.47753,   -1.9894,         1,0.92493, 0.88425,    0.53952,         57
     14, 0.31667, 0.34269,  -0.76401,        20,0.77919, 0.57298,     3.9353,        100
     15, 0.20023, 0.26494,   -6.1184,         0,0.72057, 0.43735,     6.6741,        100
     16, 0.18921, 0.19229,  -0.62367,        28,0.83436, 0.33867,    16.4959,        100
     17,0.045608,0.050023,   -1.6475,         3,0.99912, 0.84848,     4.2089,        100
     18,  0.1231,  0.1568,   -3.6118,         0,0.90103, 0.61776,     3.2653,        100
     19,       0, 0.28287,   -8.8635,         0,0.63018, 0.62053,    0.13265,         57
     20, 0.31027, 0.47356,   -7.2774,         0,0.85753, 0.64455,     5.1049,        100
     21, 0.19136, 0.37488,   -5.3425,         0,0.97793, 0.74547,     2.7151,        100
     22, 0.22015, 0.18026,    7.3218,       100,0.98088, 0.35149,    17.5923,        100
     23, 0.08406,  0.1012,   -4.7627,         0,0.98762, 0.60741,     6.2826,        100
     24,  0.4102, 0.34538,    3.6954,       100,0.69601, 0.55701,     1.9437,        100
     25, 0.14966, 0.19494,   -2.4697,         2,0.85489, 0.79752,    0.56717,         73
     26,0.053624,  0.1001,   -7.2554,         0,0.87653, 0.61682,     5.1154,        100
     27, 0.20209, 0.21157,   -1.0718,        11,0.83383, 0.43861,    13.4504,        100
     28, 0.36414,  0.3859,   -1.3404,         9,0.81175,  0.5975,     4.7904,        100
     29, 0.37622, 0.25992,   10.2604,       100,0.64478, 0.48281,     2.8797,        100
     30, 0.33347, 0.33171,   0.10515,        57,0.93098, 0.53762,     6.0094,        100
     31, 0.22893, 0.46612,   -5.7954,         0,0.97517, 0.70022,     3.7424,        100
     32, 0.18341, 0.14675,    7.5267,       100,0.92843, 0.36295,    13.9571,        100
     33, 0.38326, 0.38331, -0.003301,        55,0.78858, 0.61045,     5.0939,        100
     34,  0.4876,  0.5186,  -0.37801,        25,0.89365, 0.87697,    0.20438,         47
     35,0.084203,0.085248,   -0.3478,        39,0.94707, 0.68499,     3.9904,        100
     36, 0.61983, 0.62645, -0.076019,        42,0.98837, 0.94205,     1.4191,         98
     37, 0.21195, 0.27354,   -7.2782,         0,0.88919, 0.41435,    16.0852,        100
     38, 0.67805, 0.57405,    3.6546,       100,0.81999, 0.82626,   -0.16788,         42
```

6.3.5 Using a GroupStatistics object to create your own plots

We can use a `GroupStatistics` object (`gp` in this case) to create specific plots. Suppose we are interested in plotting all the matrices in modular sorting, such that the lables in the matrix are in red if the matrix is modular and in blue if it is antimodular. A simple script for performing this task will be:

```
n_rows = 5;
n_cols = 8;
modular_indices = gp.qb_vals.percent ≥ 97.5;
no_modular_indices = gp.qb_vals.percent ≤ 2.5;
```

```
figure(1);
for i = 1:gp.n_networks
    subplot(n_rows, n_cols, i);
    gp.networks{i}.plotter.use_labels = 0; %Do not show row/column labels
    gp.networks{i}.plotter.plot_iso_modules = 0;%No isocline inside modules
    gp.networks{i}.plotter.PlotModularMatrix();
    col = 'black'; % Color for not significance
    if(modular_indices(i) == 1) % Color for significant modularity
        col = 'red';
    elseif(no_modular_indices(i) == 1)%Color for significant antimodularity
        col = 'blue';
    end
    title(gp.names{i},'Color',col, 'FontSize',10);
end
set(gcf,'Position', [148          213          1142          746]);
```

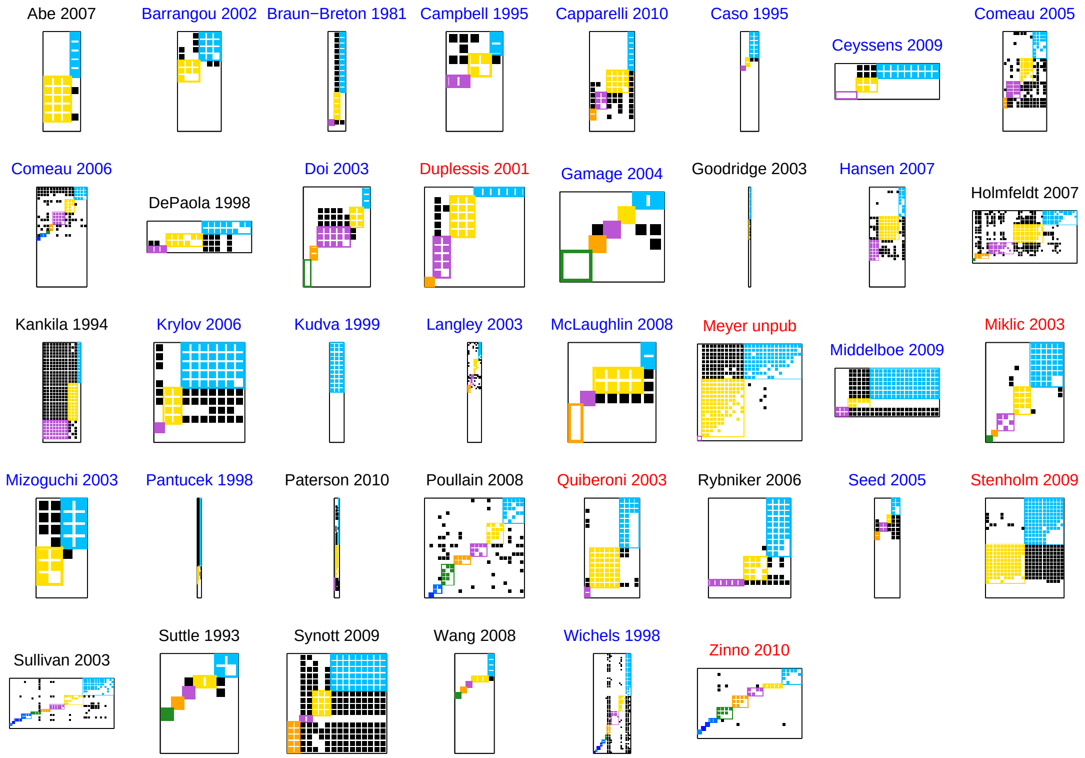

Figure 5: Modular sorting of the entire set of matrices. Red and blue are for significant modularity and anti-modularity, respectively.

We may also want to create a plot that compare the values of the networks with the random values of the null model. The next lines will show how to create such plot for the case of the NTC results

```
ntc_vals = gp.tempvals.value;
[¬,sorted_indexes] = sort(ntc_vals); % I will plot in increasing NTC value

%Get random values and sort according to sorted_indexes
```

```
ntc_vals = ntc_vals(sorted_indexes);
mean_random_vals = gp.tempvals.mean(sorted_indexes);
random_values = gp.tempvals.random_values; %variable already sorted in rows
random_values = random_values(sorted_indexes,:); %sort in rows
names = gp.names(sorted_indexes);

%Find the limits of the error bars using two tail p-value=0.05
sup_bound = random_values(:,round(gp.replicates * 0.975));
low_bound = random_values(:,round(gp.replicates * 0.025));

%Plot the data of the real matrices
figure(2);
plot(1:gp.n_networks, ...
    ntc_vals,'o','MarkerFaceColor','red','MarkerEdgeColor','red');
hold on;
%Plot the data of the random values
errorbar(1:gp.n_networks, mean_random_vals, ...
    mean_random_vals - low_bound, sup_bound - mean_random_vals, ...
    'o','MarkerFaceColor','white','MarkerEdgeColor','black');
hold off;

%Write the labels
set(gca,'xticklabel',[]);
for i=1:gp.n_networks
    tmph=text(i,-0.01,names(i));
    set(tmph,'HorizontalAlignment','right');
    set(tmph,'rotation',90);
    set(tmph,'fontsize',10);
end

%Labels in title, y-axis and legends
tmplh = legend('Measured modularity','Random ...
    expectation',1,'Location','NorthWest');
legend('boxoff')
title('Nestedness in Bacteria-Phage Networks','fontsize',20);
ylabel('Nestedness (NTC)','fontsize',16);

%Give format to the matrix
xlim([1 1+gp.n_networks]);
ylim([0 1]);

%Give appropiate size to the figure window
set(gcf,'Position',[91   135   859   505]);
set(gca,'Units','pixels');
set(gcf,'Position', [91   135   859   505+150])
apos = get(gca,'position');
apos(2) = apos(2) + 82;
set(gca,'position',apos);
set(gcf,'position',[91   135   859   596]);
```

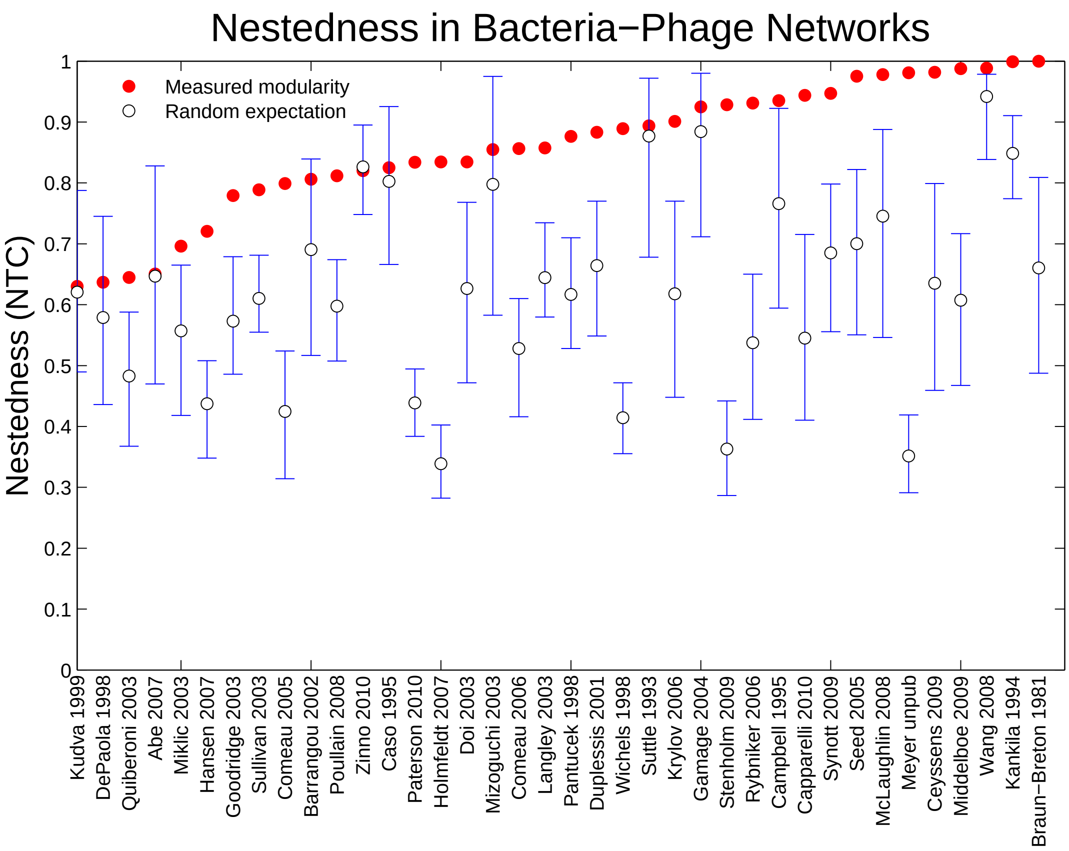

Figure 6: NTC nestedness values of a set of 38 matrices of phage-bacteria networks. A two-tail p-value of 0.05 was used for the error bars.

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] 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.

[5] 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.

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

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

[8] 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.