TIMOTHÉE POISOT

# BIPY USER MANUAL

# *Contents*

# *Introduction*

`bipy` is a set of functions written in Python, whose goal is to make it easy to analyze and visualize bipartite networks. This document describes the use of the `bipy`. This documentation is valid only for the versions before the 1.0 release (scheduled to occur sometime in the fall or winter of 2011). The aim of this documentation is to present only the "high-level" functions, i.e. these that the user will need to perform the analyses (although `bipy` was conceived to automate most of the analyses, so there are a very small number of functions that need to be known). In addition, some background about the measures used and the most important references will be given.

This guide should be viewed as a quick walkthrough to the features, and is intended primarily for the people that want to analyze their data without spending much effort in understanding the mechanisms behind the analyses. The "Example scripts" on p. 21 gives some complete scripts to perform the most common analyses.

## *Installing* `bipy`

`bipy` can be obtained on *GitHub* (http://github.com/tpoisot/bipy). There are a number of required additional software. Obviously, `bipy` requires a Python installation, preferentially 2.6 or 2.7. For those of you with a Mac, you need to download the version from python.org (http://www.python.org/getit/).

Once this is done,

## *Do I need to know Python ?*

This is a good question! If you just want to have basic informations about your data, the short answer is **no**. `bipy` was designed from the

beginning to automate most of the calculations, and use the most robust statistics available, so you don't have to play too much with the code. Most of what you will have to do is start python, load your data, and use the summary functions.

If you want to use `bipy` for more complicated analyses, then **yes**, learning python will definitely be a huge plus. But if you are willing to adopt a computational approach, chances are you already know a programming language, and you should be able to pick up python really quick.

### *How do I cite* `bipy` *?*

For the moment, just cite (by replacing the date with the day you downloaded it):

> Poisot, T. (2011) `bipy` – bipartite networks analysis using python. http://github.com/tpoisot/bipy. Downloaded on September 11, 2011.

A paper formally describing `bipy` (along with alternatives in other languages) is currently being written.

# *Loading data*

This chapter describes how to load data in a format that can be understood by `bipy`. There are two different ways to load data in `bipy`: *via* a text file, or *via* the web [1]. We will cover both in this chapter.

[1] Actually, there exists a third way to load data, namely the interaction with the EWDB – given that this database is still highly experimental, and that the server on which it is developed is not ready to support real usage, this will be introduced in future versions of `bipy`.

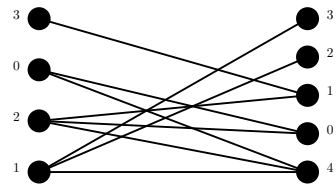
## *What should the data look like ?*

`bipy` assumes that a bipartite network is nothing more than a matrix, in which each square represents the strength of the interaction between a species from the top trophic level, and a species from the bottom trophic level. Thus, a bipartite network is defined by a matrix $\mathbf{M}$, in which $\mathbf{M}_{ij}$ is the strength of the link between upper trophic level species $i$ and lower trophic level species $j$. It is assumed that the rows of $\mathbf{M}$ are the upper trophic level species. This is a notational convention, but we will stick to it through all the documentation.

The advantage of this organization of the data is that it can easily be stored in a text file, with each interaction strength separated by a space, and the different organisms separated by a new line. Thus, the file

```
1 0 0 0 1
0 0 1 1 1
1 1 0 0 1
0 1 0 0 0
```

described the network illustrated in the margin. Of course, the values within the file can be anything (as long as they are numeric): zeroes and ones, number of visits, predation rate, percent of the diet, and so on.




However, an interaction network is much more than the interaction matrix. When an interaction matrix is loaded, `bipy` will automatically initiate the calculation of several important metrics. This has

two advantages. First, these metrics are needed for multiple calculations, so instead of being re-calculated several times, they are calculated only once. This speeds-up calculation time. Second, the object is easier to manipulate, as it keeps all the relevant informations together. For example, the connectance of a `bipartite` object called `w` is `w.connectance`.

Loading data will read the interaction matrix, and generate this `bipartite` object, which will be used in nearly all the analyses after. This requires no input from the user, so getting the data ready to work is really straightforward.

### *Reading data from a local file*

Assuming that your data are in this format, and the file is called `mydata.web`, you can load them

**Listing 1: loading data from a file**

```
1 web = loadweb('mydata.web')
```

At this point, the most difficult part is done. Your data are loaded, and you will be able to see what story they tell using the metrics described in the other chapters.

If your data are not in the "correct" format, i.e. the top level organisms are in columns, do not panic. You just need to pass a supplementary argument `t` to the `loadweb` function:

**Listing 2: loading data from a file and transposing the data**

```
1 web = loadweb('mydata.web',t=True)
```

This will tell `bipy` that your data need to be transposed before going on with the analysis. For a non-console version, just use `loadweb` with no parameters (you can still specificity `t=True` if you want to transpose your data):

**Listing 3: opening a file dialog**

```
1 web = loadweb()
```

`loadweb` will open the file selection dialog of your operating system, so you can pick your dataset.

### *Reading data from the web*

The alternative way is to load data stored on the internet. `bipy` is able to do so by using the `readRemoteWeb` function, which requires two arguments. The first one is the URL of a text file, which stores the interaction matrix. The second tells if the network should be transposed (false by default).

**Listing 4: reading a dataset from the web**

```
web = readRemoteWeb('http://server/mydata.web'),False)
```

This method can be used to load networks made publicly available on the IWDB (http://www.nceas.ucsb.edu/interactionweb/), for example.

### *Adding species names*

`bipy` has the ability to store the names of the species in the network. Currently, the way to do it is to assign a value to the properties `lonames` and `upnames` of a `bipartite` object. An easier way to load the data is currently being implemented.

The code to assign species names is:

**Listing 5: attributing species names**

```
web = bipartite(readweb('mydata.web'))
web.lonames = ['plant1','plant2','plant3',...]
web.upnames = ['herb1','herb2','herb3',...]
```

`bipy` internally takes care of re-ordering the species names when the matrix is sorted (to reflect nestedness or modularity). In addition, the species names are used in some visualizations, and when generating species-level metrics summaries.

If you do not specify species names, then `bipy` will assign number to each of the species, starting with 0 for the first species in your file, and increasing.

### *Adding bibliographical references*

When comparing several datasets, it may be interesting to know where the data come from. `bipy` is able to extract bibliographic information from several identifiers, and to generate a link to the original paper.

If you have a dataset loaded, you can add the original references and see them by doing:

**Listing 6: attributing a reference**

```
web.ref = ref({'doi':'10.1098/rsbl.2010.0774','pmid':'20961886'})})
output_citinfo(w)
```

This uses some NCBI functions to get the original data, so you need to be online to use it.

*Species–level metrics*

# *Network–level metrics*

# *Null models*

# *Graphics*

`bipy` comes with several ways to visualize the data, which are meant to reflect the properties of the networks. There are two broad ways to visualize networks: as a matrix, and as "balls and strings". The visualizations relies on the *Pyx* library, because it produces high quality (publication ready) graphics. This means that the *Pyx* library, along with LaTeX, need to be installed on your system.

In order to simplify the use of `bipy`, all of the plotting is handled by the function `plotWeb`. The number of arguments you can pass to is varies as a function of wether you want your visualization to reflect the nested or modular properties of the data.

### *Nested webs*

### *Modular webs*

In order to plot a network so as to reflect its modularity, you need to pass a second argument to `plotWeb`:

**Listing 7: plotting a modular network**

```
web = loadweb('mydata.web')
mod = findModules(mod,1000)
plotWeb(web,mod,filename='mydata-modular')
```

Note that when plotting a modular network, within each module, the species will be sorted by degree (so within each module, the visualization reflects the nestedness).

*Example scripts*