**Fit a binary configuration model to a bipartite network using the code Fit\_BiCM.py.**

The key reference for the Python code **Fit\_BiCM.py** is the published Python package bicm2.0. which is fully explained and tested at

<https://pypi.org/project/bicm/>

For all the details check:

<https://meh.imtlucca.it/codes/bicm-2-0-python3-5>

Before running the script **Fit\_BiCM.py** follows all the instructions given at <https://meh.imtlucca.it/codes/bicm-2-0-python3-5>

to run the basic routines of bicm2.0. All the information on the Python version and dependencies necessary to run bicm2.0 are provided at that link, including basic testing of the routines.

The script **Fit\_BiCM.py** simply implements a specific application of bicm2.0. It first calls ‘bicm’ and ‘numpy’ with the starting lines (below, script lines in Italics):

*import sys*

*sys.path.append('..')*

*# from BipartiteGraph import \**

*# import bicm*

*from bicm import \**

*import numpy as np*

which provides all the functions to compute the model. Then the code loads the matrix 'bicm\_mat.csv' which encodes a specific plant pollinator bipartite network:

*biad\_mat\_names = np.loadtxt('bicm\_mat.csv', delimiter=',',dtype=str)*

The code codifies the structure of the matrix in terms of the two layers:

*plants = biad\_mat\_names[1:,0]*

*pollinators = biad\_mat\_names[0, 1:]*

*biad\_mat = biad\_mat\_names[1:, 1:].astype(np.ubyte)*

*plants\_dict = dict(enumerate(np.unique(plants)))*

*plants\_inv\_dict = {v:k for k,v in plants\_dict.items()}*

*pollinators\_dict = dict(enumerate(np.unique(pollinators)))*

*pollinators\_inv\_dict = {v:k for k,v in pollinators\_dict.items()}*

and also in terms of an edge list:

*edgelist = edgelist\_from\_biadjacency(biad\_mat)[0]*

*edgelist\_names = [(plants\_dict[edge[0]], pollinators\_dict[edge[1]]) for edge in edgelist]*

Now the data are ready for the application of the basic functionalities of bicm to fit the bipartite, binary configuration model with the lines:

*myGraph = BipartiteGraph(edgelist=edgelist\_names)*

*myGraph = BipartiteGraph()*

*myGraph.set\_edgelist(edgelist\_names)*

*myGraph.solve\_tool()*

*dict\_x, dict\_y = myGraph.get\_bicm\_fitnesses()*

*print('Yielded data type is:', type(dict\_x))*

*myGraph.loglikelihood*

*x = myGraph.x*

*y = myGraph.y*

*rows\_dict = myGraph.rows\_dict*

*cols\_dict = myGraph.cols\_dict*

*avg\_mat = myGraph.get\_bicm\_matrix()*

*print(avg\_mat[0, 0] == x[0] \* y[0] / (1 + x[0] \* y[0]))*

*avg\_mat = bicm\_from\_fitnesses(x, y)*

*print(avg\_mat[0, 0] == x[0] \* y[0] / (1 + x[0] \* y[0]))*

*myGraph.check\_sol(biad\_mat, avg\_mat)*

The final lines of the script will samples the desired number of matrices. Note that in order for the sampling to work, it is necessary to manually create a folder named “samples” in the working directory.

See also notes in the script in the last lines:

*################################*

*################################*

*##Before running the next lines, which sample the ensemble, manually create the folder "samples" in the working Directory############*

*#####the next lines will sample 999 matrices. Change "range" as desired for fewer or more matrices###################*

*####for example, substitute "for sample\_i in range(999)" with "for sample\_i in range(10)" to sample just 10 matrices###############*

*sample\_with\_names = np.zeros\_like(biad\_mat\_names)*

*sample\_with\_names[:, 0] = biad\_mat\_names[:,0]*

*sample\_with\_names[0, :] = biad\_mat\_names[0,:]*

*for sample\_i in range(999):*

*sampled\_network = sample\_bicm(avg\_mat)*

*sample\_with\_names[1:,1:] = sampled\_network*

*np.savetxt('samples/sample\_' + repr(sample\_i) + '.csv', sample\_with\_names, fmt='%s', delimiter=',')*

At the end of the process, the final output will look like the folder called “samples” in this repository.