dble edge documentation

version .1

Bailey Fosdick, Daniel Larremore, Joel Nishimura, Johan Ugander

August 02, 2016

Contents

dbl_edge_mcmc documentation

Overview

About

For a fixed degree sequence and a given graph space, a configuration model is a uniform distribution of graphs with that degree sequence in that space. This code package implements Markov chain Monte Carlo methods to sample from configuration models, as discussed in the associated paper [1]. Given an input graph (and its implicit degree sequence), these methods can sample graphs from the configuration model on the space of (either stub or vertex-labeled): simple graphs, multigraphs or loopy multigraphs.

For those interested in sampling loopy graphs (graphs which may have self-loops but not multiedges): While the methods in this package will perform normally if asked to sample from the space of loopy graphs, there are relatively rare degree sequences for which double edge swaps will not be able to reach every loopy graph.

If this code is utilized in work contributing to an academic publication please cite the associated paper [1].

Package Contents

This package is composed of 3 small Python modules and one Jupyter notebook. The notebook **dbl_edge_worksheet.ipynb** provides a good overview of this package's content, demonstrating simple examples of how to: sample graphs with the same degree sequence; calculate statistics for different graph spaces; and use a simple test script to help verify the uniformity of the sampling.

Please use the Jupyter notebook in conjunction with the documentation of the following modules:

1. **dbl_edge_MCMC.py**

   This stores the main methods used to sample graphs.

2. **dist_verification.py**

   Samples small graphs and plots the output to allow for visual inspection of the distribution the MCMC methods sample.

3. **sample_assortativity.py**

   Utilizes the methods in dbl_edge_MCMC to sample the assortativity of graphs drawn uniformly from a graph space. Running this as a script computes the assortativity for simple graphs with the same degree sequence as Zachary's karate club.

Taking MCMC steps

The module **dbl_edge_MCMC.py** contains the core methods used to uniformly sample graphs with fixed degree sequences via a double edge swap Markov chain Monte Carlo sampler. This section provides a brief overview of the different ways this module can be used. For a more detailed discussion of arguments and outputs please consult the module's documentation.

There are two primary ways to utilize this module, either though providing and maintaining a networkx graph/multigraph object as in MCMC_class, or by providing and maintaining an adjacency matrix and an edge list as in the functions MCMC_step_stub and MCMC_step. In either approach it is necessary to specify the desired graph space by stating whether self-loops and/or multiedges are allowed as well deciding whether the graph is stub- or vertex-labeled.

MCMC_step and MCMC_step_stub

Functions MCMC_step_stub and MCMC_step perform a single stub-labeled and vertex-labeled (respectively) double edge swap and correspond to Algorithms 1 and 3 in the accompanying paper. These functions modify a full (non-sparse) graph adjacency matrix, a list of edges, and a length 4 list, all in place. Both take the same arguments (as detailed below). Consider the following example.

```
import numpy as np
A = np.array([[0,1,0],[1,2,2],[0,2,0 ]])
edge_list = np.array([[0,1],[1,2],[1,2],[1,1]])
swaps = [0,0,0,0]
MCMC_step_stub(A, edge_list, swaps, loops = True, multi = True)
```

This performs a single MCMC step on a stub-labeled loopy multigraph, potentially updating A, edge_list and swap with new, post-swap values.

Both functions return a boolean, which is true only if the Markov chain step altered the adjacency matrix (as opposed to resampling the current graph). If the adjacency matrix is altered the swaps argument will be changed in place, storing the nodes that were swapped.

MCMC_class

The MCMC_class is initialized with a Networkx graph, along with the three choices that define the graph space. Calling the class function 'get_graph' advances the Markov chain and returns the current Networkx graph. Consider the following example.

```
import networkx as nx
G = nx.Graph()
G.add_path([0,1,2,3,4])
MC = MCMC_class(G, loops = True, multi = True, v_labeled = False)
G2 = MC.get_graph()
```

This takes a path graph on 4 nodes, instantiates a MCMC_class based on this graph and returns a pointer to a graph G2 which differs from G by one double edge swap. Notice that this samples from the space of stub-labeled loopy multigraphs, but can be easily adjusted to other spaces.

Notes

In general, directly calling MCMC_step or MCMC_step_stub is faster than using MCMC_class, since updating Networkx data structures doesn't benefit from numba acceleration.

For large graphs, the full adjacency matrix may not be able to be stored in memory. If so, the '@nb.jit' function decorator can be deleted and a sparse matrix can be passed into these functions as an argument, though at a significant cost in speed.

We use the convention that a self-loop (u,u) contributes 2 to the diagonal of an adjacency matrix, not 1 as in Networkx.

References

[1] Bailey K. Fosdick, Daniel B. Larremore, Joel Nishimura, Johan Ugander. Configuring Random Graph Models with Fixed Degree Sequences (2016)

dbl_edge_mcmc.py

Created on Tue Jul 19 07:43:30 2016

@author: Joel Nishimura

This module contains the core methods used to uniformly sample graphs with fixed degree sequences via a double edge swap Markov chain Monte Carlo sampler.

class `dbl_edge_mcmc.`**`MCMC_class`** (`G`, `loops`, `multi`, `v_labeled=True`)

MCMC_class stores the objects necessary for MCMC steps. This implementation maintains a networkx version of the graph, though at some cost in speed.

Args:
G (networkx_class): This graph initializes the Markov chain. All sampled graphs will have the same degree sequence as G.
loops (bool): True only if loops allowed in the graph space.

multi (bool): True only if multiedges are allowed in the graph space.
v_labeled (bool): True only if the graph space is vertex-labeled. True by default.

Returns:
None
MCMC_class copies the instance of the graph used to initialize it. This class supports loopy graphs, but depending on the degree sequence, it may not be able to sample from all loopy graphs.

`get_graph ()`
The Markov chains will attempt a double edge swap, after which the next graph/multigraph in the chain is returned.

Args:
None

Returns:
The Markov chain's current graph.
Modifying the returned graph will cause errors in repeated calls of this function.

`dbl_edge_mcmc.`**`MCMC_step`**
Performs a vertex-labeled double edge swap.

Args:
A (nxn numpy array): The adjacency matrix. Will be changed inplace.
edge_list (nx2 numpy array): List of edges in A. Node names should be the integers 0 to n-1. Will be changed inplace. Edges must appear only once.
swaps (length 4 numpy array): Changed inplace, will contain the four nodes swapped if a swap is accepted.
loops (bool): True only if loops allowed in the graph space.
multi (bool): True only if multiedges are allowed in the graph space.

Returns:
bool: True if swap is accepted, False if current graph is resampled.
This method currently requires a full adjacency matrix. Adjusting this to work a sparse adjacency matrix simply requires removing the '@jit' decorator. This method supports loopy graphs, but depending on the degree sequence, it may not be able to sample from all loopy graphs.

`dbl_edge_mcmc.`**`MCMC_step_stub`**
Performs a stub-labeled double edge swap.

Args:
A (nxn numpy array): The adjacency matrix. Will be changed inplace.
edge_list (nx2 numpy array): List of edges in A. Node names should be the integers 0 to n-1. Will be changed inplace. Edges must appear only once.
swaps (length 4 numpy array): Changed inplace, will contain the four nodes swapped if a swap is accepted.
loops (bool): True only if loops allowed in the graph space.
multi (bool): True only if multiedges are allowed in the graph space.

Returns:
bool: True if swap is accepted, False if current graph is resampled.
This method currently requires a full adjacency matrix. Adjusting this to work a sparse adjacency matrix simply requires removing the '@nb.jit' decorator. This method supports loopy graphs, but depending on the degree sequence, it may not be able to sample from all loopy graphs.

`dbl_edge_mcmc.`**`flatten_graph`** `(graph, loops, multi)`
Takes an input graph and returns a version w/ or w/o loops and multiedges.

Args:
G (networkx_class): The original graph.
loops (bool): True only if loops are allowed in output graph.
multi (bool): True only if multiedges are allowed in output graph.

Returns:
A graph with or without multiedges and/or self-loops, as specified.

sample_assortativity.py

Created on Tue Jul 19 10:06:33 2016

@author: Joel Nishimura

This module contains functions to sample the assortativity values of graphs with the same degree sequence as an input graph. The functions use the dbl_edge_mcmc module to perform double edge swaps.

Running this module as a script samples the assortativity of simple graphs with the same degree sequence as Zachary's karate club at 50k different graphs spaced over 5 million double edge swaps.

Running the function 'sample_geometers' performs a more resource intensive MCMC sampling of a collaboration network of geometers.

`sample_assortativity.`**`calc_r`**
Calculates the assortativity r based on a network's edgelist and degrees.

Args:
degree (dict): Keys are node names, values are degrees.
edges (list): A list of the edges (u,v) in the graph.

Returns:
(float) The assortativity of the graph.

`sample_assortativity.`**`load_geometers`** ()
This loads the geometers graph from file and returns a networkx multigraph.

`sample_assortativity.`**`r_sample_MCMC`** (`G`, `loops`, `multi`, `v_uniform=True`, `its=10000`, `n_recs=100`, `filename='temp'`)
Samples the graph assortativity of graphs in a specified graph space with the same degree sequence as the input graph. Output is saved in subdirectory 'output/'.

Args:
G (networkx graph or multigraph): Starts the MCMC at graph G. Node names be the integers 0 to n.
loops (bool): True only if loops allowed in the graph space.
multi (bool): True only if multiedges are allowed in the graph space.
uniform (bool): True if the space is vertex labeled, False for stub-labeled.
its (int): The total number of MCMC steps
n_recs (int): The number of samples from the MCMC sampler, spaced out evenly over the total number of its.
filename (str): the name for the output file.

Returns:
(array) An array recording the assortativity at n_recs number of sampled graphs.

`sample_assortativity.`**`sample_geometers`** ()
This calculates the assortativity on a collaboration network of geometers, on each of the 7 possible graphs spaces which allow/disallow self-loops, multiedges, and are either stub or vertex-labeled. 10 thousand samples are drawn over the course of 5 billion double edge swaps. Output is saved in the 'output' subdirectory with the name 'geo'.
References: Bill Jones. Computational geometry database (http://jeffe.cs.illinois.edu/compgeom/biblios.html), 2002.

dist_verification.py

Created on Tue Jul 19 09:20:04 2016

@author: Joel Nishimura

This module contains functions to test the uniformity of the MCMC sampling in dbl_edge_mcmc.py.

Running this as a script performs a test on the path graph with degree sequence 1,2,2,2,1. Output is saved to subdirectory 'verification'.

A more thorough, though time-consuming test, is available in the function 'test_sampling_seven_node'.

`dist_verification.`**`determine_relative_freq`** (`G`)
Returns the ratio of stub-matchings for the input graph divided by the number of stub-matchings for a simple graph with the same degree sequence.

Args:
G (networkx_class): The input graph.

Returns:
The likelihood of the input graph relative to a simple graph with the same degree sequence.

`dist_verification.`**`plot_vals`** (`samples`, `uniform`, `name`)
Plots the output of test_sampling as a histogram of the number of times each graph was visited in the MCMC process. Creates a figure in subdirectory 'verification/'.

Args:
samples (dict): Output from test_sampling. Has a length 2 list as values corresponding to [num_samples,sampling_weight].
uniform (bool): True if the space is vertex labeled, False for stub-labeled.
name (str): Name for output.

Returns:
None

`dist_verification.`**`test_sampling`** (`G`, `self_loops=False`, `multi_edges=False`, `sample_uniformly=True`, `its=100000`)
Tests the uniformity of the MCMC sampling on an input graph.

Args:
G (networkx graph or multigraph): The starting point of the mcmc double edges swap method.
self_loops (bool): True only if loops allowed in the graph space.
multi_edges (bool): True only if multiedges are allowed in the graph space.
sample_uniformly (bool): True if the space is vertex labeled, False for stub-labeled.
its (int): The number of samples from the MCMC sampler.

Returns:
dict: Keys correspond to each visited graph, with values being a list giving the number of times the graph was sampled along with a weight proportional to the expected number of samplings (relevant for stub-labeled samplings)

`dist_verification.`**`test_sampling_five_node`** ()
This tests the MCMC's ability to sample graphs uniformly, on degree seq. 1,2,2,2,1. Output is saved to subdirectory verification with name beginning in 'FiveNode'.

`dist_verification.`**`test_sampling_seven_node`** ()
This tests the MCMC's ability to sample graphs uniformly, on degree seq. 5,3,2,2,2,1,1. Output is saved to subdirectory verification with name beginning in 'SevenNode'.

Indices

- **genindex**
- **modindex**
- **search**

Index

Python Module Index

d

s