Reference Manual for Package "graphMC"

J. Ray, A. Pinar and C. Safta
Sandia National Laboratories, Livermore, CA
{jairay,apinar,csafta}@sandia.gov

1 Introduction

This is a manual for the package "graphMC", a statistical package for testing the independence of graphs generated by a Markov chain Monte Carlo method. It contains statistical software that can be used to investigate whether a series of graphs generated by Markov chain (where the joint-degree-distribution is held constant) resemble a first-order Markov process or whether they seem to resemble independent draws from a distribution. We observe that if we follow an edge between two labeled nodes in the graph, we get a binary time-series indicating whether the edge existed or not in the graph realizations. A necessary condition for the sequence of graphs to be first-order Markov is that the time-series of each of the edges should resemble a first order Markov process. The same hold true if we desire to generate independent realizations of graphs from a Markov chain ("MC") on the space of graphs.

The package supplies the software used in the paper "A stopping criterion for Markov chains when generating independent random graphs", by J. Ray, A. Pinar and C. Seshadhri, being submitted to Physical Review E. The paper contains the theory and justification for using these methods with graphs. A version of the paper can also be found on arXiv [1].

The contents of the package are:

1. `libMIX` : A library that implements a graph re-wiring scheme that preserves the joint degree distribution of the graph.
2. `libgibbsit`: A statistical library containing tests of first-order Markovianity, independence and calculation of Markov transition probabilities. This is a stripped down and modified version of the FORTRAN code `gibbsit.f` [2], and translated into C++. We are grateful to StatLib [3] for making this code available to the public.
3. `libgraphDB`: A utility library, containing a graph data base. This is used for storing realization of graphs generated by an MC.
4. `ex01`: An example that illustates how to use the tests of first-order Markovianity and independence, in `libgibbsit`, with a binary time-series.
5. `ex02`: A example of how to use the tests when one has a MC on graphs. This example uses `libMIX` to rewire and generate graph realizations inside a MC and also uses `libgraphDB` to store them, prior to analyzing them

Below we describe the "public" functions in these libraries. We provide some description of the examples, but the reader is advised to read the code in `ex01/` and `ex02/`.

2 Description of libraries

This section contains description of the "public" functions in the 3 libraries distributed with this package.

2.1 `libMIX`

`libMIX` contains a scheme for rewiring graphs while preserving the joint degree distribution. [4] describes this re-wiring scheme being used within a MC. The public functions, whose use is demonstrated in `ex02/ex02.cpp` are:

1. `struct graph`: This is a struct that is used to store a graph in a sparse representation format. The definition is in `include/graph.h`.
2. `int check_simple(struct graph *G)`: Checks whether the graph is simple - no double edges between nodes, no self-loops and no disconnected sub-graphs.
3. `void mix_jdd(struct graph *G, int X)`: Given a graph G, re-wire it X times and put it back in the same struct.
4. `edges2gr(char *fname, struct graph *G)`: Given an edge-list stored in a file `fname`, read and fill up the `struct graph`. The function will allocate all the arrays in `struct graph` but it is the caller's responsibility to de-allocate the memory.
5. `int make_JDD(struct graph *G, int ***J)`: A function that calculates the joint degree distribution of a graph. The function allocates a 2D matrix for J and the caller is responsible for deallocating this 2D matrix.

2.2 `libgibbsit`

`libgibbsit` contains a set of tests for independence, and a function `gibbsmain()` that takes a binary time-series, thins it repeatedly, applies the tests of independence and first-order Markovianity. It also tracks whether the time-series is long enough of us to compute these tests with any degree of precision.

The individial tests `mctest()` and `indtest()` are used in `ex01/ex01.cpp`, as is the function to estimate transition probabilities (`mcest()`). `gibbsmain()` is used in `ex02/convergence.cpp`. The theory for this test is in Chapter 2.2.3, 3.1.2 and 7.2 and 7.3 in [5] and [6].

1. `void mctest(int *data, int n, double *g2, double *bic)`: This function takes in an array of zeros and ones (`*data`), of length `n`, and fits a second-order Markov model and first-order Markov model to it. Bascially, we will in a $2 \times 2 \times 2$ contingency table of different types of 3-step transions e.g.$\{0,0,0\}, \{1,0,1\}$ etc. and predict their expected values using log-linear models for second- and first-order Markov processes. It returns $\Delta\text{BIC} = \text{BIC}_{\text{second-order}} - \text{BIC}_{\text{first-order}}$ in `bic` and the G2 statistic (ratio of likelihoods) [5] in `g2`. A negative `bic` indicates that the first-order Markov model is better fitted to the data. If not, the data displays longer range correlations i.e., `data[i+1]` is dependent on `data[i], data[i-1], ....` and you should consider thinning the time-series to make it first-order Markov. See illustation in `ex01/ex01.cpp`.

2. `void mcest(int *data, int n, double *alpha, double *beta)`: Function to estimate the 0-1 and 1-0 transition probabilities in a binary time-series, *assuming that the series is first-order Markov*. Use `mctest()` to make sure that it is. See illustation in `ex01/ex01.cpp`

3. `void indtest(int *data, int n, double *g2, double *bic)` : This function performs a test of independence, by fitting log-linear models, similar to `mctest()`. See illustation in `ex01/ex01.cpp`.

4. `void gibbmain(double *original, int n, double q, double r,double s, double epsilon, double *dwrk, int *iwrk, int *nmin, int *kthin, int *nburn, int *nprec, int *kmind, int *r15)`: This is a function that (1) ensure that a time-series is long enough to do a test of independence, and if not, provides an estimated length `nprec` and (2) returns the thinning factors `kthin` (to turn the time-series into first-order Markov) and `kmind`, that turns it into independent draws from a distribution. An illustration is in `ex02/convergence.cpp`.

   `original` is the binary time series of length `n`. `q` is set to -1, indicating the the function should calculate an edge-mean using the original time-series. We desire a time-series that, when thinned to a first-order Markov series will still provide edge-mean estimates which are with a tolerance of `r` with confidence `q`. `dwrk, iwrk` are work arrays supplied by the caller.

2.3 `libutils`

This is a rather simple library which is an implementation of a graphical database. This database is used to store graphs as they are generated by an MC; they are then retrieved when performing tests of independence etc. The header for the `graphDB` object is `include/graphDB.h` which provides exhaustive documentation for the member functions. The functions are

1. `graphDB::insertGraph(struct graph *pG)`: Store a graph into a database

2. `graphDB::getNumGraphsInDB()`: Number of graphs in the database

3. `struct graph * graphDB::extractGraph(int i)`: Extract graph `i` from the database and return a pointer to it.

4. `struct graph * graphDB::extractCopyOfGraph(int i)`: Same as above but make a copy of the graph first. It is the caller's responsibility to delete this copy.

5. `int graphDB::writeGraphsToFiles(std::string filename)`: Self-explanatory

6. `int graphDB::createThinnedVersionOfDatabase(graphDB &ndb, int thin_factor)`: Create a decimated/thinned version of the database by retaining every `thin_factor`-th instance.

7. `graphDB::clear()`: Delete all graphs in the database and clear out.

3 Description of example problems

We provide 2 examples, in `ex01/` and `ex02/`, to illustrate the use of the software tools in this package.

3.1 ex01

This example simply demonstrates the use of tests of first-order Markovianity and independence, operating on a binary time-series provided in `timeSeries.dat`. The example demonstrates the use of `mctest(), indtest()` and `mcest()`, from `libgibbsit`.

The program is executed simply as `./ex01.exe`. The code output shows that one requires a thinning by a factor of 4 before first-order Markovianity sets in; it needs a further thinning, by a factor of 7, for become independent. Higher levels of thinning preserve a better fit of first-order Markov model (versus a second-order one) and a better fit of an independence model (versus a first-order Markov model).

3.2 ex02

`ex02` is an example of a Markov chain on graphs. It is initialized using a real graph (`lesmis.elist`). It first turns the edge-list into a graph using `edges2gr()` and then performs a random walk, using `mix_jdd()` to rewire the graph while preserving the joint degree distribution. The description of this method is in [4]. Every $|E|$ iterations, it stores the graph in the database (`db.insertGraph(&G)`. At the end of the run, `checkConvergence()` is invoked to check for independence of realizations.

`checkConvergence()` is implemented in `convergence.cpp`. It extracts the time-histories of edges and examines them using `gibbsmain()`. It insists on having a MC long enough that edge-means can be estimated with tolerance `r = 0.01` with confidence `s = 95%`. The functioning of `checkConvergence()` is best understood by reading through the source-code.

The example is executed as `./ex02.exe`, which prints out a help screen. Also, one can run the code using `ex02/runit.sh`.

Acknowledgements

This work was funded by the United States Department of Energy, Office of Science and by an Early Career Award from the Laboratory Directed Research & Development (LDRD) program at Sandia National Laboratories. Sandia National Laboratories is a multiprogram laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] J. Ray, A. Pinar, and C. Seshadhri. A stopping criterion for markov chains when generating independent random graphs. arXiv:1210.8184, 2012.

[2] Adrian E. Raftery and Steven M. Lewis. Fortran code for gibbsit. http://lib.stat.cmu.edu/general/gibbsit.

[3] Statlib: Data, software and news from the statistic community. http://lib.stat.cmu.edu/.

[4] Isabelle Stanton and Ali Pinar. Constructing and sampling graphs with a prescribed joint degree distribution using Markov chains. *ACM Journal of Experimental Algorithmics*. to appear.

[5] Y. M. Bishop, S. E. Fienberg, and P. W. Holland. *Discrete multivariate analysis: Theory and practice*. Springer-Verlag, New York, NY, 2007.

[6] A. E. Raftery and S. M. Lewis. How many iterations in the Gibbs sampler? In J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, editors, *Bayesian Statistics*, volume 4, pages 765–766. Oxford University Press, 1992.