Package 'CCMnet'

July 2, 2014

Version 0.0-2

Date 2014-01-14

Title Simulate Congruence Class Model for Networks

Author Ravi Goyal, with contributions from Mark S. Handcock (ergm R package), David R. Hunter (ergm R package), Carter T. Butts (ergm R package), Steven M. Goodreau (ergm R package), Pavel N. Krivitsky (ergm R package), Martina Morris (ergm R package), Nicole Bohme Carnegie (ergm-user term degmix)

Maintainer Ravi Goyal <ravi.goyal@mail.harvard.edu>

Depends sna, network, ergm, R(>= 2.12.2)

Description Tools to simulate networks based on Congruence Class models.

License GPL-3 + file LICENSE

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-02-14 21:46:52

R topics documented:

CCMnet *Methods to Simulate Networks from the Congruence Class Model.*

Description

CCMnet is a collection of methods to simulate networks from the Congruence Class Model (CCM) for networks. For details regarding CCM see the paper by Goyal et al. in the reference section below.

For a list of functions type: help(package='CCMnet')

To cite this package see the citation in `citation(package="CCMnet")`.

Details

The `CCMnet_constr` function simulates networks based on the CCM for networks; see the examples section of the `CCMnet_constr` function for a few demonstrations.

This package relies on the **network** package, which allows networks to be represented in R. Markov Chain Monte Carlo methods are from the **ergm** package; **CCMnet** is partially based on C code from the **ergm** package (see reference by Handcock et al. below). The **SNA** package allows for additional methods relating to social network analysis. For detailed information regarding these packages go to the **statnet** website: `http://statnet.org`. This package incorporates an ergm-term coded by Nicole Bohme Carnegie (carnegie@hsph.harvard.edu).

Author(s)

Ravi Goyal, with contributions from Mark S. Handcock (ergm R package), David R. Hunter (ergm R package), Carter T. Butts (ergm R package), Steven M. Goodreau (ergm R package), Pavel N. Krivitsky (ergm R package), Martina Morris (ergm R package), Nicole Bohme Carnegie (ergm-user term degmix)

References

Goyal R, Blitzstein J, and De Gruttola V. Sampling Networks from Their Posterior Predictive Distribution. Network Science. In press.

Handcock M, Hunter D, Butts C, Goodreau S, Krivitsky P, and Morris M (2012). ergm: Fit, Simulate and Diagnose Exponential-Family Models for Networks. Seattle, WA. Version 3.0-1. Project home page at http://www.statnet.org.

CCMnet_constr *Congruence Class Model for Networks*

Description

Simulate networks from the Congruence Class Model (CCM) for networks.

Usage

```
CCMnet_constr(Network_stats,
              Prob_Distr,
              Prob_Distr_Params,
              samplesize = 5000,
              burnin=1000,
              interval=1000,
              statsonly=TRUE,
              P=NULL,
              population,
              covPattern = NULL,
              remove_var_last_entry = FALSE
   )
```

Arguments

Network_stats	A vector of network statistics used to generate networks. Currently implemented network statistics are the following: "Density" (Network Density), "DegreeDist" (Degree Distribution), "DegMixing" (Degree Mixing Matrix), and "Triangles" (Number of 3-cycles).
Prob_Distr	A vector of probability distributions; one distribution for each network statistic in Network_stats. Currently implemented probability distributions are: "Normal" and "DirMult". DirMult (Dirichlet-multinomial) is currently only implemented for degree distribution.
Prob_Distr_Params	A list of lists. Each list in the list corresponds to a network statistic in Network_stats. Each list gives the parameters for the associated probability distribution in Prob_Distr. The normal distribution requires a list of length 2 containing a mean vector and covariance matrix. The Dirichlet-multinomial requires a list of length 1 containing the alpha vector.
samplesize	Number of networks simulated.
burnin	MCMC burnin.
interval	Thinning of MCMC.
statsonly	If TRUE, returns only the network statistics of the simulated network, plus the last simulated network. If FALSE returns the network statistics and simulated networks.

`P` If P is specified, it will be used as the initial starting network for the MCMC; otherwise, a random graph will be used as the initial starting network.

`population` Number of nodes in simulated networks.

`covPattern` Covariate Pattern of each node in network. Currently only a maximum of two covariate patterns, labeled 1 and 2, are allowed.

`remove_var_last_entry`
Removes the last row and column of the covariance matrix.

Details

Simulates networks of fixed network size given a probability distribution on network statistics. Uses an algorithm from Goyal et al. and partially based on C code from Handcock et al. (2012). Also incorporates an ergm-term coded by Nicole Bohme Carnegie (carnegie@hsph.harvard.edu).

Not all combinations of network statistics and probability distributions are currently implemented; additional combinations will be implemented in later releases. Below we list all implemented combinations.

Density: (Normal)

DegreeDist: (Normal) and (DirMult)

DegMixing: (Normal)

DegMixing and Clustering: (Normal, Normal)

Value

A list of length 2 containing:

`Network Statistics`
a matrix of network statistics

`Network(s)` List of returned network(s) of type network

References

Goyal R, Blitzstein J, and De Gruttola V. Sampling Networks from Their Posterior Predictive Distribution. Network Science. In press.

Handcock M, Hunter D, Butts C, Goodreau S, Krivitsky P, and Morris M (2012). ergm: Fit, Simulate and Diagnose Exponential-Family Models for Networks. Seattle, WA. Version 3.0-1. Project home page at http://www.statnet.org.

Examples

```
set.seed(1)
#
#Example: Density (Normal)
#
CCMnet_Result = CCMnet_constr(Network_stats= c('Density'),
                              Prob_Distr=c('Normal'),
                              Prob_Distr_Params=list(list(.04, .0001)),
                              samplesize = 5000,
```

```
                              burnin=1000,
                              interval=100,
                              statsonly=TRUE,
                              P=NULL,
                              population=100,
                              covPattern = NULL,
                              remove_var_last_entry = FALSE)
statsmatrix = CCMnet_Result[[1]]
G_list = CCMnet_Result[[2]]

#Mean Network Density (Simulated Networks)
mean(statsmatrix)
#Variance of Network Density (Simulated Networks)
var(statsmatrix)

## Not run:
#
#Example: Degree Distribution (Dirichlet-multinomial)
#
CCMnet_Result = CCMnet_constr(Network_stats='DegreeDist',
                              Prob_Distr='DirMult',
                              Prob_Distr_Params=list(list(c(2,21,15,12))),
                              samplesize = 10000,
                              burnin=100000,
                              interval=1000,
                              statsonly=TRUE,
                              P=NULL,
                              population=500,
                              covPattern = NULL,
                              remove_var_last_entry = FALSE)
statsmatrix = CCMnet_Result[[1]]
G_list = CCMnet_Result[[2]]

#Mean Degree Distribution (Simulated Networks)
apply(statsmatrix, 2, mean)
#Variance of Degree Distribution (Simulated Networks)
apply(statsmatrix, 2, var)

#
#Example: Degree Distribution (Normal)
#
Prob_Distr_Params=list(NS_Multinomial(G_list[[1]],
                                      Network_stats = 'DegreeDist',
                                      mean_inflate = .05,
                                      var_inflate = 1.05))

CCMnet_Result = CCMnet_constr(Network_stats='DegreeDist',
                              Prob_Distr='Normal',
                              Prob_Distr_Params=Prob_Distr_Params,
                              samplesize = 50000,
                              burnin=100000,
                              interval=1000,
                              statsonly=TRUE,
```

```
                              P=NULL,
                              population=500,
                              covPattern = NULL,
                              remove_var_last_entry = FALSE)
statsmatrix = CCMnet_Result[[1]]
G_list = CCMnet_Result[[2]]

#Mean Degree Distribution (Simulated Networks)
apply(statsmatrix, 2, mean)
#Variance of Degree Distribution (Simulated Networks)
apply(statsmatrix, 2, var)

#
#Example: Degree Mixing (Normal)
#
Prob_Distr_Params=list(NS_Multinomial(G_list[[1]],
                                      Network_stats = 'DegMixing',
                                      mean_inflate = .05,
                                      var_inflate = 1.05))

CCMnet_Result = CCMnet_constr(Network_stats='DegMixing',
                              Prob_Distr='Normal',
                              Prob_Distr_Params=Prob_Distr_Params,
                              samplesize = 50000,
                              burnin=100000,
                              interval=1000,
                              statsonly=TRUE,
                              P=NULL,
                              population=500,
                              covPattern = NULL,
                              remove_var_last_entry = FALSE)
statsmatrix = CCMnet_Result[[1]]
G = CCMnet_Result[[2]]

#Mean Degree Mixing (Simulated Networks)
apply(statsmatrix, 2, mean)
#Variance of Degree Mixing (Simulated Networks)
apply(statsmatrix, 2, var)


#
#Example: Degree Mixing and Triangles (Normal, Normal)
#
Prob_Distr_Params=list(NS_Multinomial(G_list[[1]],
                                      Network_stats = 'DegMixing',
                                      mean_inflate = .05,
                                      var_inflate = 1.05),
                       list(6,2))

CCMnet_Result = CCMnet_constr(Network_stats=c('DegMixing', 'Triangles'),
                              Prob_Distr=c('Normal', 'Normal'),
                              Prob_Distr_Params=Prob_Distr_Params,
                              samplesize = 50000,
```

```
                              burnin=100000,
                              interval=1000,
                              statsonly=TRUE,
                              P=NULL,
                              population=500,
                              covPattern = NULL,
                              remove_var_last_entry = FALSE)
statsmatrix = CCMnet_Result[[1]]
G = CCMnet_Result[[2]]

#Mean Degree Mixing and Number of Triangles (Simulated Networks)
apply(statsmatrix, 2, mean)
#Variance of Degree Mixing and Number of Triangles (Simulated Networks)
apply(statsmatrix, 2, var)

## End(Not run)
```

NS_Multinomial *Calculate network statistic and covariance matrix.*

Description

Calculate network statistic and covariance matrix, which is based on a multinomial distribution. Each unit (either node or edge) in the network is assumed to be sampled from a multinomial distribution based on probabilities associated with the network statistic.

Usage

```
NS_Multinomial(g,
   Network_stats,
   mean_inflate = 0,
   var_inflate = 1
 )
```

Arguments

g	a network object.
Network_stats	Either 'DegreeDist' or 'DegMixing'.
mean_inflate	Add small amount to remove zero values from degree mixing matrix entries.
var_inflate	Multiply the variance by a constant. Used to avoid signular covariance matrices.

Value

A list of length 2 containing:

Network Statistic	Network statistic of the inputted network.
Covariance	Covariance matrix for the network statistic; assumes each unit (either node or edge) is sampled from a multinomial distribution based on probabilities derived from the network statistic.

Examples

```
g = as.network(rgraph(n=500, m=1, tprob=.01,
           mode='graph', diag=FALSE,
                     replace=FALSE, tielist=NULL,
                     return.as.edgelist=FALSE),
               directed = FALSE)

Prob_Distr_Params=list(NS_Multinomial(g,
  Network_stats = 'DegreeDist',
                       mean_inflate = .05,
                       var_inflate = 1.05))

Prob_Distr_Params=list(NS_Multinomial(g,
                       Network_stats = 'DegMixing',
                       mean_inflate = .05,
                       var_inflate = 1.05))
```

Index