Package 'netmediate'

June 19, 2024

Type Package

Title Micro-Macro Analysis for Social Networks

Version 1.0.1

Maintainer Scott Duxbury <duxbury@email.unc.edu>

Description Estimates micro effects on macro structures (MEMS) and average micro mediated effects (AMME).
URL: <https://github.com/sduxbury/netmediate>.
BugReports: <https://github.com/sduxbury/netmediate/issues>.
Robins, Garry, Phillipa Pattison, and Jodie Woolcock (2005) <doi:10.1086/427322>.
Snijders, Tom A. B., and Christian E. G. Steglich (2015) <doi:10.1177/0049124113494573>.
Imai, Kosuke, Luke Keele, and Dustin Tingley (2010) <doi:10.1037/a0020761>.
Duxbury, Scott (2023) <doi:10.1177/00811750231209040>.
Duxbury, Scott (2024) <doi:10.1177/00811750231220950>.

License GPL (>= 2)

Imports MASS, btergm, stats, ergm, RSiena, sna, network, ergMargins, VGAM, plyr, lme4, plm, gam, intergraph

Suggests Matrix, igraph, relevent, statnet, statnet.common

Encoding UTF-8

NeedsCompilation no

Author Scott Duxbury [aut, cre, cph]

Repository CRAN

Date/Publication 2024-06-19 08:40:07 UTC

Contents

AMME *Function to estimate the average micro mediated effect (AMME).*

Description

AMME implements parametric and nonparametric estimation routines to estimate the average mediated micro effect. It requires two models. The first is a generative network model (i.e., a model where the dyad, dyad-time period, or dyad-group is the unit of analysis) of the form $f(A_{ij}|T_{ij}, Z_{ij})$, where A is a cross-sectional or longitudinal network or group of longitudinal or cross-sectional networks, T is the possibly endogenous network selection process of interest and Z is a matrix of possibly endogenous confounding selection mechanisms.

The second model is a cross-sectional or longitudinal macro model (i.e., a model where the unit of analysis is a node, subgraph, or network or a combination of nodes, subgraphs, and networks measured collected from multiple settings [such as distinct schools or organizations]) of the form $g(Y_i|M_i, X_i, T_i)$, where Y_i is the outcome variable, M_i is the mediating macro variable, X_i is a matrix of control variables that possibly vary as a function of selection process T_{ij}, and T_i is the optional unit-level measure of T_{ij}. The AMME is the change in Y_i when T_{ij} allowed to vary versus set to 0 because of an associated change in M_i. The AMME is given by

$$AMME = \frac{1}{2n} y_i(T_i(t), M_i(T_{ij}), X_i(t)) - y_i(T_i(t), M_i(0), X_i(t))$$

, where n is the number of observations and $t = 0, T_{ij}$. AMME currently accepts the following micro models: `glm`, `glmer`, `ergm`, `btergm`, `sienaFit`, `rem.dyad`, and `netlogit` objects. The following macro model objects are accepted: `lm`, `glm`, `lmer`, `glmer`, `gam`, `plm`, and `lnam` objects. Pooled estimation for multiple network models is also implemented for `ergm` and `sienaFit` micro models. Both parametric and nonparametric estimation are available.

Usage

```
AMME(micro_model,
      macro_model,
      micro_process,
      mediator,
      macro_function,
      link_id,
      object_type=NULL,
      controls=NULL,
      control_functions=NULL,
      interval=c(0,1),
      nsim=500,
      algorithm="parametric",
      silent=FALSE,
      full_output=FALSE,
      SAOM_data=NULL,
      SAOM_var=NULL,
      time_interval=NULL,
```

```
covar_list=NULL,
edgelist=NULL,
net_logit_y=NULL,
net_logit_x=NULL,
group_id=NULL,
node_numbers=NULL)
```

Arguments

`micro_model` the micro-model. Currently accepts `glm`, `glmer`, `ergm`, `btergm`, `sienaFit`, `rem.dyad`, and `netlogit` objects. Pooled estimation for multiple network models is also implemented for `ergm` and `sienaFit` objects. To implement pooled estimation, `model` should be provided as a list of `ergm` or `sienaFit` objects.

`macro_model` the macro model. Currently accepts `lm`, `glm`, `lmer`, `glmer`, `gam`, `plm`, and `lnam` objects.

`micro_process` a character string containing the name of the micro process of interest. The character string should exactly match the relevant coefficient name in `micro_model` output.

`mediator` a character string containing the name of the mediating variable of interest. The character string should exactly match the relevant coefficient name in `macro_model` output.

`macro_function` a `function` that calculates `mediator` on the simulated networks. Currently accepts user defined functions as well as functions inherent in the `igraph` and `statnet` packages for R.

`link_id` a required vector of IDs used to link the `micro_model` output to the `macro_model` input. If calculating a network-level mediator, this should be the network identifier or network-group/network-time period identifier. If calculating a node-level mediator, this should be the node ID or node-time-period/node-group identifier. Observations should correspond exactly to rows in the `macro_model` data matrix. If calculating multiple network statistics at different levels of analysis when `controls` are included, `link_id` may be provided as an ordered list of identifiers. In this case, each entry in the list is a vector of IDs corresponding to the unique entries of the relevant statistics. If provided as a list, the first entry should correspond to `macro_function` (i.e., the mediator) and the remaining entries should correspond to `control_functions` (i.e., the controls).

`controls` a vector of character strings listing the control variables in `macro_model` that may vary as a function of `micro_process`. Each element in `controls` should correspond exactly to a coefficient in `macro_model` output. If `controls` is left `NULL`,then the AMME is calculated without controlling for confounding network variables.

`control_functions`
a list of functions used to calculate `controls`. The elements in `control_functions` should correspond exactly to the elements in `controls` and should be provided in the same order. If `micro_process` appears as an independent variable in `macro_model`, then this can be specified by specifying the `netmediate` helper function `identity_function` to `control_functions`.

object_type
: A character string or vector of character strings that tells netmediate the type of object to apply the macro_function and control_functions to. If controls are included into the AMME call, then object_type should be provided as a vector of character strings where the first element is the object_type for macro_function and the remaining elements are the ordered object_type for control_functions. Currently accepts igraph and network objects. If left NULL, network objects are assumed. Can be over-ridden to use other object types with a user-function by defining a function that accepts either a network or igraph object and returns a numeric value or vector of numeric values (see examples).

interval
: Tuning parameters to vary the strength of θ. Should be provided as a vector of numeric values with 2 entries.

nsim
: The number of simulations or bootstrap samples to use during estimation.

algorithm
: The estimation algorithm to be used. Currently accepts "parametric" and "nonparametric". If "parametric", estimation is obtained with Monte Carlo sampling. If "nonparametric", estimation uses bootstrap resampling.

silent
: logical parameter. Whether to provide updates on the progress of the simulation or not.

full_output
: logical parameter. If set to TRUE, the entire distribution of simulated statistics will be provided as part of the model output.

SAOM_data
: required when micro_model is a sienaFit object; ignored otherwise. If a sienaFit object is provided, SAOM_data should be the siena object that contains the data for SAOM estimation. If using pooled estimation on multiple sienaFit objects (i.e., providing a list of sienaFit objects), then SAOM_data should be provided as an ordered list with each entry containing the siena object corresponding to list of sienaFit objects.

SAOM_var
: optional parameter when micro_model is a sienaFit object. SAOM_var is a list of of the varCovar and varDyadCovar objects used to assign time varying node and dyad covariates when calling sienaDataCreate. If provided, netmediate assigns the varying node covariates and dyad covariates to each simulated network. This parameter is required when macro_function computes a statistic that varies as a function of time varying node or dyad covariates (i.e., network segregation, assortativity). Time invariant characteristics (coCovar and coDyadCovar) are handled internally by MEMS and should not be provided. When providing a list of sienaFit objects for pooled estimation, SAOM_var should be provided as a list of lists, where each entry in the list contains a list of varCovar and varDyadCovar objects associated with corresponding sienaFit object.

time_interval
: an optional parameter to be used when micro_model is a rem.dyad object. May be provided as a numeric vector or the character string "aggregate". If a numeric vector is provided unique network snapshots at each interval. For example, time_interval=c(0,2,3) would induce two networks, one for the 0 - 2 time period and one for the 2 - 3 time period. If specified as "aggregate", the AMME is calculated by creating an aggregated cross-sectional representation of the entire event sequence. If left NULL, defaults to |"aggregate". Note that time_interval must correspond to the time periods observed in macro_model. That is, time_interval must be set to "aggregate" when macro_model is

cross-sectional and the entries in time_interval must correspond to the time periods observed in the repeated measurement data when macro_model is longitudinal.

covar_list
: an optional list of sender/receiver covariates used in rem.dyad estimation. Only required when a rem.dyad object is the micro_model and covariates are in the rem.dyad call. The list format should correspond to the format required by rem.dyad.

edgelist
: an optional three column edgelist providing the sender, receiver, and time of event occurrence when micro_model is a rem.dyad object. Only required when time_interval is set to NULL or "aggregate". Ignored for other types of models.

net_logit_y
: the dependent variable when micro_model is a netlogit object. Should be provided as a vector.

net_logit_x
: the matrix of independent variables when micro_model is a netlogit object

group_id
: optional vector of group identifiers to use when micro_model is a glm or glmer on grouped data (i.e., multiple time periods, multiple networks). When specified, AMME will induce unique networks for each grouping factor. If left unspecified, all groups/time periods are pooled. If using glmer, the grouping factor does not have to be provided as part of the model or used as a random effect. If specified, the entries in the macro_model model matrix are assumed to be sequentially ordered by unit_id-group_id.

node_numbers
: a numeric vector containing the number of nodes in each group_id when using glm or glmer. If estimating AMME aggregated over all networks (i.e., group_id=NULL), this shoud be the total number of nodes in all networks. Required when using glm or glmer, ignored otherwise.

Details

Estimates the AMME over the provided intervals. Standard errors and confidence intervals are based on the sampling distribution of simulated values, which are calculated either parametrically or nonparametrically according to algorithm. Parametric estimation is typically faster, but cannot be used for nonparametric network models (e.g., quadratic assignment procedure).

macro_function and control_functions make up the core utilites of AMME. macro_function calculates the mediating variable of interest, while control_functions calculates all control variables that vary as a function of micro_process and potentially confound the effect of mediator. When controls are left NULL, then AMME estimates the AMME without accounting for confounding variables. Specifying controls and control_functions ensures that estimates of the AMME account for alternative pathways from micro_process to the outcome variable in macro_model. In cases where micro_process is included as a predictor variable in macro_model, this can be specified by including the netmediate helper function identity_function into control_functions.

netmediate currently supports functions calculated on igraph and network objects, which should be specified using the object_type argument. These may be functions inherent to the statnet and igraph software package or they may be functions from other packages that accept network/igraph objects. The functions provided to macro_function and control_functions may also be user-defined functions that accept network or igraph objects as inputs and return a numeric value or vector of numeric values as output. It is also possible to over-ride the network and igraph object requirements within a user function. To do so, set the object_type argument (or relevant element within the object_type argument when object_type is a list) to either network or igraph and then define a user-function that accepts a network or igraph object as its input, converts the object to the desired data structure, calculates the statistic of interest, and returns a numeric value or vector of numeric values. See examples below for an illustration.

By default, the AMME is calculated by averaging over the distribution of simulated values. If full_output is set to TRUE, the distribution of simualted statistics is returned. This may be useful when the median or mode of the simulated distribution is required or if the researcher wants to inspect the distributional shape of simulated values.

AMME also supports pooled estimation for when multiple ergm or sienaFit objects are used as the micro_model. To use pooled estimation, the model parameter should be specified as a list of ergm or sienaFit objects. If using sienaFit, the SAOM_data argument will also need to be specified as an ordered list with elements corresponding to entries in the list of sienaFit objects. Similarly, the SAOM_var parameter will need to be specified as a list of lists, where each entry in the list is, itself, a list containing all varCovar and varDyadCovar objects used to calculate macro statistics of interest. Note that SAOM_var should not be provided if the macro statistic of interest is not a function of the variables contained in varCovar and varDyadCovar.

Value

If full_output=FALSE, then a table is returned with the AMME, its standard error, confidence interval, and p-value.

If full_output=TRUE, then a list is returned with the following three elements.

summary_dat is the table of summary output ucontaining the AMME, its standard error, confidence interval, and p-value.

AMME_obs is vector of observations where each entry is the AMME for a single simulation trial.

prop_explained_obs
is vector containing the proportion explained values for each simulation trial.

Author(s)

Duxbury, Scott W. Associate Professor, University of North Carolina–Chapel Hill, Department of Sociology.

References

Duxbury, Scott W. 2024. "Micro-macro Mediation Analysis in Social Networks." Sociological Methodology.

See Also

MEMS ergm.mma mediate

Examples

```
##############################
#   Basic AMME specifications
#############################


####create ERGM generative model
library(statnet)
data("faux.mesa.high")
ergm_model<-ergm(faux.mesa.high~edges+
                   nodecov("Grade")+
                   nodefactor("Race")+
                   nodefactor("Sex")+
                   nodematch("Race")+
                   nodematch("Sex")+
                   absdiff("Grade"))


###create node-level data for second stage analysis with
node_level_data<-data.frame(grade=faux.mesa.high%v%"Grade",
                            race=faux.mesa.high%v%"Race",
                            sex=faux.mesa.high%v%"Sex",
                            degree=degree(faux.mesa.high))

node_level_data$senior<-0
node_level_data$senior[node_level_data$grade==max(node_level_data$grade)]<-1
node_level_data$v_id<-1:network.size(faux.mesa.high) #define ID for each observation

probit_model<-glm(senior~race+sex+degree,
                data=node_level_data,
                family=binomial(link="probit"))

###estimate the indirect effect of grade homophily on senior status acting through degree centrality
  #in a model with no network control variables
AMME(micro_model=ergm_model,
     macro_model=probit_model,
     micro_process="absdiff.Grade",
     mediator="degree",
     macro_function=degree,
     link_id=node_level_data$v_id, #specify vertex IDs
     object_type="network",
     interval=c(0,1),
     nsim=50,
     algorithm="parametric",
     silent=FALSE)

#use nonparametric estimation for a generalized additive model
library(gam)

gam_model<-gam(senior~race+sex+s(degree),
               data=node_level_data)
```

```
AMME(micro_model=ergm_model,
     macro_model=gam_model,
     micro_process="absdiff.Grade",
     mediator="s(degree)",
     macro_function=degree,
     link_id=node_level_data$v_id,
     object_type="network",
     interval=c(0,1),
     nsim=50,
     algorithm="nonparametric",
     silent=FALSE)



###estimate AMME with linear network autocorrelation model

lnam_model<-lnam(node_level_data$grade,
                x=as.matrix(node_level_data[,4:5]),
                W1=as.sociomatrix(faux.mesa.high))


AMME(micro_model=ergm_model,
     macro_model=lnam_model,
     micro_process="absdiff.Grade",
     mediator="degree",
     macro_function=degree,
     link_id=node_level_data$v_id,
     object_type="network",
     interval=c(0,1),
     nsim=50,
     algorithm="parametric",
     silent=FALSE)




############################
#   Including controls
###########################

##single control
node_level_data<-data.frame(grade=faux.mesa.high%v%"Grade",
                            race=faux.mesa.high%v%"Race",
                            sex=faux.mesa.high%v%"Sex",
                            degree=degree(faux.mesa.high),
                            betweenness=betweenness(faux.mesa.high))

node_level_data$senior<-0
node_level_data$senior[node_level_data$grade==max(node_level_data$grade)]<-1
node_level_data$v_id<-1:network.size(faux.mesa.high) #define ID for each observation

probit_model<-glm(senior~race+sex+degree+betweenness,
```

```
                  data=node_level_data,
                  family=binomial(link="probit"))


AMME(micro_model=ergm_model,
     macro_model=probit_model,
     micro_process="absdiff.Grade",
     mediator="degree",
     macro_function=degree,
     link_id=node_level_data$v_id, #specify vertex IDs
     controls="betweenness", #should match model output exactly
     control_functions=betweenness,
     object_type="network",
     interval=c(0,1),
     nsim=50,
     algorithm="parametric",
     silent=FALSE)



##multiple controls
##include an AR 1 parameter to make it a nonlinear network autocorrelation model
node_level_data$AR1<-as.sociomatrix(faux.mesa.high)%*%node_level_data$senior
probit_model<-glm(senior~race+sex+degree+betweenness+AR1,
                  data=node_level_data,
                  family=binomial(link="probit"))

#specify user function
ar_function<-function(x){
  return(as.sociomatrix(x)%*%node_level_data$senior)
}


AMME(micro_model=ergm_model,
     macro_model=probit_model,
     micro_process="absdiff.Grade",
     mediator="degree",
     macro_function=degree,
     link_id=node_level_data$v_id,
     controls=c("betweenness","AR1"), #should match model output exactly
     control_functions=list(betweenness,ar_function), #provide functions as a list
     object_type="network",
     interval=c(0,1),
     nsim=50,
     algorithm="parametric",
     silent=FALSE)



##using identity_function when micro_process has a direct effect on y
  #to use identity_function, the control and micro_process need to have the same
  #name and the macro control variable has to be numeric
```

```
node_level_data$Sex<-as.numeric(as.factor(node_level_data$sex))
logit_model<-glm(senior~race+Sex+degree+betweenness+AR1,
                  data=node_level_data,
                  family=binomial)



AMME(micro_model=ergm_model,
     macro_model=logit_model,
     micro_process="nodefactor.Sex.M",
     mediator="degree",
     macro_function=degree,
     link_id=node_level_data$v_id,
     controls=c("betweenness","AR1","Sex"), #should match model output exactly
     control_functions=list(betweenness,ar_function,identity_function),
     object_type="network",
     interval=c(0,1),
     nsim=50,
     algorithm="parametric",
     silent=FALSE)





################################
#   More complex data structures
###############################


###############################
# AMME with longitudinal data
##############################

#bootstrap TERGM and panel data model
library(btergm)
library(plm)
data(alliances)

ally_data<-list(LSP[[1]],
                LSP[[2]],
                LSP[[3]])

#fit bootstrap TERGM with 200 replications
bt_model<-btergm(ally_data~edges+
                   gwesp(.7,fixed=T)+
                   mutual,R=200)


#create node data
ally_node_data<-data.frame(outdeg=c(rowSums(LSP[[1]]),rowSums(LSP[[2]]),rowSums(LSP[[3]])),
                          indeg=c(colSums(LSP[[1]]),colSums(LSP[[2]]),colSums(LSP[[3]])))
```

```
ally_node_data$v_id<-rep(rownames(LSP[[1]]),3) #create node IDS
ally_node_data$t_id<-c(rep(1, nrow(ally_data[[1]])), #create time IDS
                       rep(2, nrow(ally_data[[1]])),
                       rep(3, nrow(ally_data[[1]])))
ally_node_data$link_id<-paste(ally_node_data$v_id,ally_node_data$t_id)#create node-panel identifiers

ally_node_data$v_id<-as.factor(as.character(ally_node_data$v_id))

#estimate a linear model with node fixed effects
lm_model<- lm(outdeg~indeg +v_id,
          data = ally_node_data)



AMME(micro_model=bt_model,
     macro_model=lm_model,
     micro_process="gwesp.OTP.fixed.0.7",
     mediator="indeg",
     macro_function=function(x){degree(x,cmode="indegree")},
     link_id=ally_node_data$link_id, #provide node-panel identifiers
     object_type="network",
     interval=c(0,1),
     nsim=11,
     algorithm="nonparametric",
     silent=FALSE)


##include controls at different units of analysis
  #include global transitivity statistic at each network panel
transitivity_list<-c(gtrans(as.network(LSP[[1]])),
                     gtrans(as.network(LSP[[2]])),
                     gtrans(as.network(LSP[[3]])))


ally_node_data$transitivity<-c(rep(transitivity_list[1],nrow(LSP[[1]])),
                               rep(transitivity_list[2],nrow(LSP[[2]])),
                               rep(transitivity_list[3],nrow(LSP[[3]])))



lm_model<- lm(outdeg~indeg+transitivity +v_id,
              data = ally_node_data)



AMME(micro_model=bt_model,
     macro_model=lm_model,
     micro_process="gwesp.OTP.fixed.0.7",
     mediator="indeg",
     macro_function=function(x){degree(x,cmode="indegree")},
   link_id=list(ally_node_data$link_id,ally_node_data$t_id),#list of IDs for nodes and time
     controls="transitivity",
     control_functions = gtrans,
```

```
    object_type="network",
    interval=c(0,1),
    nsim=11,
    algorithm="nonparametric",
    silent=FALSE)
```

```
#SAOM and panel data model with PLM package
library(RSiena)
#specify 3 wave network panel data as DV
network_list<-array(c(s501,s502,s503),dim = c(50,50,3))

Network<-sienaDependent(network_list)
Smoking<-varCovar(s50s)
Alcohol<-varCovar(s50a)
SAOM.Data<-sienaDataCreate(Network=Network,Smoking,Alcohol)

#specify
SAOM.terms<-getEffects(SAOM.Data)
SAOM.terms<-includeEffects(SAOM.terms,egoX,altX,sameX,interaction1="Alcohol")
SAOM.terms<-includeEffects(SAOM.terms,egoX,altX,sameX,interaction1="Smoking")
SAOM.terms<-includeEffects(SAOM.terms,transTies,inPop)


create.model<-sienaAlgorithmCreate(projname="netmediate",
                                   nsub=5,
                                   n3=2000)


##estimate the SAOM
SAOM_model<-siena07(create.model,
                        data=SAOM.Data,
                        effects=SAOM.terms,
                        verbose=TRUE)


##create node-level data
node_level_data<-data.frame(smoking=s50s[,1], #smoking behavior for DV
                            alcohol=s50a[,1],
                            v_id=rownames(s501), #unique node IDS
                            wave="Wave 1",       #unique time IDS
                            outdegree=rowSums(s501),
                            indegree=colSums(s501),
                            AR1=s501%*%s50s[,1],  #assign network autocorrelation
                            gcc=gtrans(as.network(s501)))

node_level_data<-rbind(node_level_data,data.frame(smoking=s50s[,2],
                                                  alcohol=s50a[,2],
                                                  v_id=rownames(s502),
                                                  wave="Wave 2",
                                                  outdegree=rowSums(s502),
```

```
                                                     indegree=colSums(s502),
                                                     AR1=s502%*%s50s[,2],
                                                     gcc=gtrans(as.network(s502))))



node_level_data<-rbind(node_level_data,data.frame(smoking=s50s[,3],
                                                     alcohol=s50a[,3],
                                                     v_id=rownames(s503),
                                                     wave="Wave 3",
                                                     outdegree=rowSums(s503),
                                                     indegree=colSums(s503),
                                                     AR1=s503%*%s50s[,3],
                                                     gcc=gtrans(as.network(s503))))


##create unique identifiers for node-panel
node_level_data$unique_ids<-paste(node_level_data$v_id,node_level_data$wave)

##estimate one-way fixed effects model with PLM
library(plm)
FE_model<-plm(smoking~alcohol+outdegree+indegree+AR1+gcc,
               data=node_level_data,
               index=c("v_id","wave"))



##create AR function to provide to AMME
ar_function<-function(x){return(as.sociomatrix(x)%*%(x%v%"Smoking"))}


AMME(micro_model=SAOM_model,
     macro_model=FE_model,
     micro_process="reciprocity",
     mediator="indegree",
     macro_function=function(x){degree(x,cmode="indegree")},
     link_id=list(node_level_data$unique_id,node_level_data$unique_id,
                     node_level_data$unique_id,node_level_data$wave),
     object_type="network",
     controls=c("outdegree","AR1","gcc"),
    control_functions=list(function(x){degree(x,cmode="outdegree")},ar_function,gtrans),
     interval=c(0,.1),
     nsim=500,
     algorithm="parametric",
     silent=FALSE,
     SAOM_data = SAOM.Data,
     SAOM_var=list(Smoking=Smoking,Alcohol=Alcohol)) #provide var_list
```

```
################################
# AMME with pooled ERGM and SAOM
################################



#pooled ERGM
  #fit two ERGMs to two networks
data("faux.mesa.high")
model1<-ergm(faux.mesa.high~edges+
               nodecov("Grade")+
               nodefactor("Race")+
               nodefactor("Sex")+
               nodematch("Race")+
               nodematch("Sex")+
               absdiff("Grade"))

data("faux.magnolia.high")
model2<-ergm(faux.magnolia.high~edges+
               nodecov("Grade")+
               nodefactor("Race")+
               nodefactor("Sex")+
               nodematch("Race")+
               nodematch("Sex")+
               absdiff("Grade"))


#create node level data
node_level_data<-data.frame(grade=faux.mesa.high%v%"Grade",
                            sex=faux.mesa.high%v%"Sex",
                            degree=degree(faux.mesa.high),
                            betweenness=betweenness(faux.mesa.high),
                            gcc=gtrans(faux.mesa.high),
                            net_id="Mesa")

node_level_data$senior<-0
node_level_data$senior[node_level_data$grade==max(node_level_data$grade)]<-1
node_level_data$v_id<-1:network.size(faux.mesa.high)


node_level_data2<-data.frame(grade=faux.magnolia.high%v%"Grade",
                            sex=faux.magnolia.high%v%"Sex",
                            degree=degree(faux.magnolia.high),
                            betweenness=betweenness(faux.magnolia.high),
                            gcc=gtrans(faux.magnolia.high),
                            net_id="Magnolia")

node_level_data2$senior<-0
node_level_data2$senior[node_level_data$grade==max(node_level_data2$grade)]<-1
node_level_data2$v_id<-206:(network.size(faux.magnolia.high)+205)
node_level_data<-rbind(node_level_data,node_level_data2)
```

```
#estimate glm macro model with an AR 1 process
probit_model<-glm(senior~sex+degree+betweenness+gcc,
                data=node_level_data,
                family=binomial(link="probit"))



AMME(micro_model=list(model1,model2),
     macro_model=probit_model,
     micro_process="nodematch.Sex",
     mediator="degree",
     macro_function=degree,
     link_id=list(node_level_data$v_id,node_level_data$v_id,node_level_data$net_id),
     object_type="network",
     controls=c("betweenness","gcc"),
     control_functions=list(betweenness,gtrans),
     interval=c(0,1),
     nsim=50,
     algorithm="parametric",
     silent=FALSE)



##pooled SAOM with control functions using time varying covariates

library(RSiena)
#specify 3 wave network panel data as DV
network_list<-array(c(s501,s502,s503),dim = c(50,50,3))

Network<-sienaDependent(network_list)
Smoking<-varCovar(s50s)
Alcohol<-varCovar(s50a)
SAOM.Data<-sienaDataCreate(Network=Network,Smoking,Alcohol)

#specify
SAOM.terms<-getEffects(SAOM.Data)
SAOM.terms<-includeEffects(SAOM.terms,egoX,altX,sameX,interaction1="Alcohol")
SAOM.terms<-includeEffects(SAOM.terms,egoX,altX,sameX,interaction1="Smoking")
SAOM.terms<-includeEffects(SAOM.terms,transTies,inPop)


create.model<-sienaAlgorithmCreate(projname="netmediate",
                                   nsub=5,
                                   n3=2000)


##estimate the SAOM
SAOM_model<-siena07(create.model,
                    data=SAOM.Data,
                    effects=SAOM.terms,
                    verbose=TRUE)
```

```
##create node-level data
node_level_data<-data.frame(smoking=s50s[,1], #smoking behavior for DV
                            alcohol=s50a[,1],
                            v_id=rownames(s501), #unique node IDS
                            wave="Wave 1",       #unique time IDS
                            outdegree=rowSums(s501),
                            indegree=colSums(s501),
                            AR1=s501%*%s50s[,1],  #assign network autocorrelation
                            gcc=gtrans(as.network(s501)))

node_level_data<-rbind(node_level_data,data.frame(smoking=s50s[,2],
                                                  alcohol=s50a[,2],
                                                  v_id=rownames(s502),
                                                  wave="Wave 2",
                                                  outdegree=rowSums(s502),
                                                  indegree=colSums(s502),
                                                  AR1=s502%*%s50s[,2],
                                                  gcc=gtrans(as.network(s502))))



node_level_data<-rbind(node_level_data,data.frame(smoking=s50s[,3],
                                                  alcohol=s50a[,3],
                                                  v_id=rownames(s503),
                                                  wave="Wave 3",
                                                  outdegree=rowSums(s503),
                                                  indegree=colSums(s503),
                                                  AR1=s503%*%s50s[,3],
                                                  gcc=gtrans(as.network(s503))))


#recycle the same model for illustrative purposes
node_level_data$net_ID<-"Model 1"
node_level_data<-rbind(node_level_data,node_level_data)
node_level_data$net_ID[151:300]<-"Model 2"

##create unique identifiers for node-panel
  #ID for node-panel-model
node_level_data$unique_id<-paste(node_level_data$v_id,node_level_data$wave,node_level_data$net_ID)
  #ID for panel-model
node_level_data$unique_waves<-paste(node_level_data$wave,node_level_data$net_ID)

#estimate a linear network autocorrelation model with node fixed effects
FE_model<-lm(smoking~alcohol+outdegree+indegree+AR1+gcc+v_id,
              data=node_level_data)



##create user function calculate AR1 process on time varying node attributes
ar_function<-function(x){return(as.sociomatrix(x)%*%(x%v%"Smoking"))}

##estimate AMME
AMME(micro_model=list(SAOM_model,SAOM_model), #provide list of sienaFit objects
```

```
      macro_model=FE_model,
      micro_process="reciprocity",
      mediator="indegree",
      macro_function=function(x){degree(x,cmode="indegree")},
      link_id=list(node_level_data$unique_id,node_level_data$unique_id,
                         node_level_data$unique_id,node_level_data$unique_waves),
      object_type="network",
      controls=c("outdegree","AR1","gcc"),
     control_functions=list(function(x){degree(x,cmode="outdegree")},ar_function,gtrans),
      interval=c(0,.1),
    nsim=100,            #parametric estimation requires more simulations than coefficients
      algorithm="parametric",
      silent=FALSE,
      SAOM_data = list(SAOM.Data,SAOM.Data), #list of siena objects
      SAOM_var=list(list(Smoking=Smoking,Alcohol=Alcohol),#provide var_list
                   list(Smoking=Smoking,Alcohol=Alcohol)))




#################################
# AMME with nested data
################################

####create dyad-level data

library(lme4)
library(btergm)
##use small data to simplify estimation
glm_dat<-edgeprob(model1)
glm_dat$net_id<-"mesa"
glm_dat2<-edgeprob(model2)
glm_dat2$net_id<-"magnolia"
glm_dat<-rbind(glm_dat,glm_dat2[,-c(4)])


##estimate micro model as glm for btoh networks using pooled ERGM data
net_glm<-glm(tie~nodecov.Grade+
                 nodefactor.Race.Hisp+
                 nodefactor.Race.NatAm+
                 nodefactor.Race.Other+
                 nodefactor.Sex.M+
                 nodematch.Race+
                 nodematch.Sex+
                 absdiff.Grade,
               data=glm_dat)


#create macro data
node_level_data<-data.frame(grade=faux.mesa.high%v%"Grade",
                            sex=faux.mesa.high%v%"Sex",
                            degree=degree(faux.mesa.high),
```

```
                            betweenness=betweenness(faux.mesa.high),
                            gcc=gtrans(faux.mesa.high),
                            net_id="Mesa")

node_level_data$senior<-0
node_level_data$senior[node_level_data$grade==max(node_level_data$grade)]<-1
node_level_data$v_id<-1:network.size(faux.mesa.high)


node_level_data2<-data.frame(grade=faux.magnolia.high%v%"Grade",
                             sex=faux.magnolia.high%v%"Sex",
                             degree=degree(faux.magnolia.high),
                             betweenness=betweenness(faux.magnolia.high),
                             gcc=gtrans(faux.magnolia.high),
                             net_id="Magnolia")

node_level_data2$senior<-0
node_level_data2$senior[node_level_data$grade==max(node_level_data2$grade)]<-1
node_level_data2$v_id<-206:(network.size(faux.magnolia.high)+205)
node_level_data<-rbind(node_level_data,node_level_data2)


#estimate glm macro model
probit_model<-glm(senior~sex+degree+betweenness+gcc,
                  data=node_level_data,
                  family=binomial(link="probit"))




AMME(micro_model=net_glm,
     macro_model=probit_model,
     micro_process="nodematch.Sex",
     mediator="degree",
     macro_function=degree,
     link_id=list(node_level_data$v_id,node_level_data$v_id,node_level_data$net_id),
     object_type="network",
     controls=c("betweenness","gcc"),
     control_functions=list(betweenness,gtrans),
     interval=c(0,1),
     nsim=50,
     algorithm="parametric",
     silent=FALSE,
     group_id=glm_dat$net_id,
     node_numbers = c(network.size(faux.mesa.high),
                      network.size(faux.magnolia.high)))




###using glmer for micro model
net_glmer<-glmer(tie~nodecov.Grade+
                 nodefactor.Race.Hisp+
```

```
                 nodefactor.Race.NatAm+
                 nodefactor.Race.Other+
                 nodefactor.Sex.M+
                 nodematch.Race+
                 nodematch.Sex+
                 absdiff.Grade+
                 (1|net_id),
               data=glm_dat)

probit_glmer<-glm(senior~sex+degree+betweenness+gcc,
                data=node_level_data,
                family=binomial(link="probit"))


AMME(micro_model=net_glm,
     macro_model=probit_glmer,
     micro_process="nodematch.Sex",
     mediator="degree",
     macro_function=degree,
     link_id=list(node_level_data$v_id,node_level_data$v_id,node_level_data$net_id),
     object_type="network",
     controls=c("betweenness","gcc"),
     control_functions=list(betweenness,gtrans),
     interval=c(0,1),
     nsim=50,
     algorithm="parametric",
     silent=FALSE,
     group_id=glm_dat$net_id,
     node_numbers = c(network.size(faux.mesa.high),
                      network.size(faux.magnolia.high)))
```

`compare_MEMS` *Function to compare micro effect on macro structure (MEMS) estimates between models.*

Description

`compare_MEMS` implements parametric and nonparametric routines to compare `MEMS` estimate between models. When compared between nested models, `compare_MEMS` results can be interpreted as the portion of a MEMS explained by a mediating or confounding variable. When compared between models with distinct functional forms and the same specification, `compare_MEMS` results can be interpreted as the sensitivity of `MEMS` results to decision about model functional form.

The difference in MEMS is the change in MEMS after one or more micro-processes are included into a model or, in the case of sensitivity tests, when the functional form is changed. Let $MEMS_p$ represent the MEMS obtained from a model that omits one or more intervening variables and $MEMS_f$ be the MMES obtained from a model that includes the intervening variable(s). The change in MEMS is given

$$\Delta MEMS = MEMS_p - MEMS_f$$

. $MEMS_p$ and $MEMS_f$ may also be have the same specification but use distinct functional forms or other modeling decisions in the case of sensitivity tests. Tuning parameters can be assigned to toggle the strength of θ in model-implied estimates of $MEMS$. `MEMS` currently accepts `glm`, `glmer`, `ergm`, `btergm`, `sienaFit`, `rem.dyad`, and `netlogit` objects and implements both parametric and nonparametric estimation. Pooled estimation for multiple network models is also implemented for `ergm` and `sienaFit` objects.

Usage

```
compare_MEMS(partial_model,
      full_model,
      micro_process,
      macro_function,
      object_type=NULL,
      interval=c(0,1),
      nsim=500,
      algorithm="parametric",
      silent=FALSE,
      full_output=FALSE,
      SAOM_data=NULL,
      SAOM_var=NULL,
      time_interval=NULL,
      covar_list=NULL,
      edgelist=NULL,
      net_logit_y=NULL,
      net_logit_x=NULL,
      group_id=NULL,
      node_numbers=NULL,
      mediator=NULL,
      link_id=NULL,
      controls=NULL,
      control_functions=NULL)
```

Arguments

`partial_model` the micro-model excluding one or more intervening or confounding variables of interest. May also be a fully specified model with a distinct functional form in the case of sensitivity tests. Currently accepts `glm`, `glmer`, `ergm`, `btergm`, `sienaFit`, `rem.dyad`, and `netlogit` objects. Pooled estimation for multiple network models is also implemented for `ergm` and `sienaFit` objects. To implement pooled estimation, `model` should be provided as a list of `ergm` or `sienaFit` objects.

full_model
: the micro-model including one or more intervening or confounding variables of interest. May also be a fully specified model with a distinct functional form in the case of sensitivity tests. Currently accepts `glm`, `glmer`, `ergm`, `btergm`, `sienaFit`, `rem.dyad`, and `netlogit` objects. Pooled estimation for multiple network models is also implemented for `ergm` and `sienaFit` objects. To implement pooled estimation, `model` should be provided as a list of `ergm` or `sienaFit` objects.

micro_process
: a character string containing the name of the micro process of interest. The character string should exactly match coefficient names in `model` output.

macro_function
: a `function` that calculates the macro statistic of interest. Currently accepts user defined functions as well as functions inherent in the `igraph` and `statnet` packages for R.

object_type
: A character string that tells netmediate the type of object to apply the `macro_function` to. Currently accepts `igraph` and `network` objects. If left `NULL`, `network` objects are assumed. Can be over-ridden to use other object types with a user-function by defining a function that accepts either a `network` or `igraph` object and returns a numeric value or vector of numeric values (see examples).

interval
: The value of tuning parameters to assign to θ. Should be provided as a vector of numeric values with 2 entries.

nsim
: The number of simulations or bootstrap samples to use during estimation.

algorithm
: The estimation algorithm to be used. Currently accepts `"parametric"` and `"nonparametric"`. If `"parametric"`, estimation is obtained with Monte Carlo sampling. If `"nonparametric"`, estimation uses bootstrap resampling.

silent
: logical parameter. Whether to provide updates on the progress of the simulation or not.

full_output
: logical parameter. If set to `TRUE`, `compare_MEMS` will return all sampled statistics and complete results for $MEMS_p$ and $MEMS_f$.

SAOM_data
: required when the model is a `sienaFit` object; ignored otherwise. If a `sienaFit` object is provided, `SAOM_data` should be the `siena` object that contains the data for SAOM estimation. If using pooled estimation on multiple `sienaFit` objects (i.e., providing a list of `sienaFit` objects), then `SAOM_data` should be provided as an ordered list with each entry containing the `siena` object corresponding to list of `sienaFit` objects.

SAOM_var
: optional parameter when the model is a `sienaFit` object. `SAOM_var` is a list of of the `varCovar` and `varDyadCovar` objects used to assign time varying node and dyad covariates when calling `sienaDataCreate`. If provided, `netmediate` assigns the varying node covariates and dyad covariates to each simulated network. This parameter is required when `macro_function` computes a statistic that varies as a function of time varying node or dyad covariates (i.e., network segregation, assortativity). Time invariant characteristics (`coCovar` and `coDyadCovar`) are handled internally by `MEMS` and should not be provided. When providing a list of `sienaFit` objects for pooled estimation, `SAOM_var` should be provided as a list of lists, where each entry in the list contains a list of `varCovar` and `varDyadCovar` objects associated with corresponding `sienaFit` object.

time_interval
: an optional parameter to be used with rem.dyad objects. May be provided as a numeric vector or the character string "aggregate". If a numeric vector is provided unique network snapshots at each interval. For example, time_interval=c(0,2,3) would induce two networks, one for the 0 - 2 time period and one for the 2 - 3 time period. If specified as "aggregate", the MEMS is calculated by creating an aggregated cross-sectional representation of the entire event sequence. If left NULL, defaults to |"aggregate".

covar_list
: an optional list of sender/receiver covariates used in rem.dyad estimation. Only required for rem.dyad objects when covariates are included. The list format should correspond to the format required by rem.dyad

.

edgelist
: an optional three column edgelist providing the sender, receiver, and time of event occurrence when using rem.rem.dyad. Only required when time_interval is set to NULL or "aggregate". Ignored for other types of models.

net_logit_y
: the dependent variable for netlogit objects. Should be provided as a vector. Only required when model is a netlogit object.

net_logit_x
: the matrix of independent variables for netlogit type objects. Only required when model is a netlogit object.

group_id
: optional vector of group identifiers to use when estimating a glm or glmer on grouped data (i.e., multiple time periods, multiple networks). When specified, MEMS will induce unique networks for each grouping factor. If left unspecified, all groups/time periods are pooled. If using glmer, the grouping factor does not have to be provided as part of the model or used as a random effect.

node_numbers
: a numeric vector containing the number of nodes in each group_id when using glm or glmer. If estimating MEMS aggregated over all networks (i.e., group_id=NULL), this shoud be the total number of nodes in all networks. Required when using glm or glmer, ignored otherwise.

mediator
: a character string detailing the mediator of interest. Intended for internal use with the AMME function; not intended for end users.

link_id
: a vector or list of vectors corresponding to unique identifiers. Intended for internal use with the AMME function; not intended for end users.

controls
: a vector of character strings listing the controls to be calculated when using AMME. Intended for internal use with the AMME function; not intended for end users.

control_functions
: a list of functions to calculate the macro control variables provided in controls. Intended for internal use with the AMME function; not intended for end users.

Details

Compares MEMS estimates between two models. If one or more confounding or intervening variables are excluded or included between models, the change in MEMS can be interpreted as the portion of the MEMS explained by one or more confounding or intervening variable. If two models are provided with the same specification but a distinct functional form, the change in MEMS is a sensitivty test of how much the MEMS estimate changes because of a model decision. This can be useful, for example, when comparing TERGM and SAOM estimates as each models make distinct assumptions about sources of network change and the temporal ordering of tie changes.

`compare_MEMS` functionality inherits directly from the `MEMS` command. See the `MEMS` page for more details.

Value

If `full_output=FALSE`, then a table is returned with the change MEMS, its standard error, confidence interval, and p-value, and the same results for the partial and complete MEMS.

If `full_output=TRUE`, then a list is returned with the following three elements.

`diff_MEMS_results`
: is the table of summary output containing the MEMS, its standard error, confidence interval, and p-value, and a list of the simulated values of the change in MEMS.

`p_MEMS_results`
: contains the summary statistics for the partial MEMS along with all simulated statistics.

`f_MEMS_results`
: contains the summary statistics for the full MEMS along with all simulated statistics.

Author(s)

Duxbury, Scott W. Associate Professor, University of North Carolina–Chapel Hill, Department of Sociology.

References

Duxbury, Scott W. 2024. "Micro Effects on Macro Structure in Social Networks." Sociological Methodology.

Wertsching, Jenna, and Scott W. Duxbury. Working paper. "Comparing Micro Effects on Macro Structure between Nested Models."

See Also

AMME MEMS ergm.mma mediate

Examples

```
##############
# Not run
###############
library(statnet)
library(igraph)
data("faux.mesa.high")

  #how much of the effect of racial homophily on transitivity
```

```
    #is explained by triadic closure effects?

model<-ergm(faux.mesa.high~edges+nodecov("Grade")+nodefactor("Race")+
               nodefactor("Sex")+nodematch("Race")+nodematch("Sex")+absdiff("Grade"))

model2<-ergm(faux.mesa.high~edges+nodecov("Grade")+nodefactor("Race")+
               nodefactor("Sex")+nodematch("Race")+nodematch("Sex")+absdiff("Grade")+
               gwesp(.5,fixed=TRUE))


compare_MEMS(partial_model=model,
              full_model=model2,
              micro_process="nodematch.Race",
             macro_function=transitivity,
             object_type = "igraph",
             silent=FALSE,
             algorithm="parametric")
```

---

identity_function — *Function to map micro_process onto macro_model within calls to* `AMME`.

Description

A function to control for a node-level `micro_process` in `AMME` estimation.

Usage

```
identity_function(x)
```

Arguments

`x`	a network object used to transfer `micro_process`.

Value

No return value, used internally with `AMME`

MEMS *Function to estimate the micro effect on macro structure (MEMS).*

Description

MEMS implements parametric and nonparametric estimation routines to estimate the micro effect on macro structure when using a generative network model (i.e., a model where the dyad, dyad-time period, or dyad-group is the unit of analysis). The MEMS is defined in postestimation as a function of the possibly endogenous micro process X, which is assumed to be a predictor in the micro model of the form $A = f(\theta X + \gamma^T Z)$, where Z is a matrix of possibly endogenous controls and A is the network of interest. The MEMS when θ changes from 0 to 1 is given by

$$MEMS = \sum_i \frac{M(\theta, X, \gamma, Z)_i - M(\gamma, Z)_i}{n}$$

, for n observations. Tuning parameters can be assigned to toggle the strength of θ in model-implied estimates of $MEMS$. MEMS currently accepts glm, glmer, ergm, btergm, sienaFit, rem.dyad, and netlogit objects and implements both parametric and nonparametric estimation. Pooled estimation for multiple network models is also implemented for ergm and sienaFit objects.

Usage

```
MEMS(model,
      micro_process,
      macro_function,
      object_type=NULL,
      interval=c(0,1),
      nsim=500,
      algorithm="parametric",
      silent=FALSE,
      full_output=FALSE,
      SAOM_data=NULL,
      SAOM_var=NULL,
      time_interval=NULL,
      covar_list=NULL,
      edgelist=NULL,
      net_logit_y=NULL,
      net_logit_x=NULL,
      group_id=NULL,
      node_numbers=NULL,
      mediator=NULL,
      link_id=NULL,
      controls=NULL,
      control_functions=NULL)
```

Arguments

- `model` — the micro-model to be analyzed. Currently accepts `glm`, `glmer`, `ergm`, `btergm`, `sienaFit`, `rem.dyad`, and `netlogit` objects. Pooled estimation for multiple network models is also implemented for `ergm` and `sienaFit` objects. To implement pooled estimation, `model` should be provided as a list of `ergm` or `sienaFit` objects.
- `micro_process` — a character string containing the name of the micro process of interest. The character string should exactly match coefficient names in `model` output.
- `macro_function` — a `function` that calculates the macro statistic of interest. Currently accepts user defined functions as well as functions inherent in the `igraph` and `statnet` packages for R.
- `object_type` — A character string that tells netmediate the type of object to apply the `macro_function` to. Currently accepts `igraph` and `network` objects. If left `NULL`, `network` objects are assumed. Can be over-ridden to use other object types with a user-function by defining a function that accepts either a `network` or `igraph` object and returns a numeric value or vector of numeric values (see examples).
- `interval` — The value of tuning parameters to assign to θ. Should be provided as a vector of numeric values with 2 entries.
- `nsim` — The number of simulations or bootstrap samples to use during estimation.
- `algorithm` — The estimation algorithm to be used. Currently accepts `"parametric"` and `"nonparametric"`. If `"parametric"`, estimation is obtained with Monte Carlo sampling. If `"nonparametric"`, estimation uses bootstrap resampling.
- `silent` — logical parameter. Whether to provide updates on the progress of the simulation or not.
- `full_output` — logical parameter. If set to `TRUE`, the entire distribution of simulated statistics will be provided as part of the model output.
- `SAOM_data` — required when the model is a `sienaFit` object; ignored otherwise. If a `sienaFit` object is provided, `SAOM_data` should be the `siena` object that contains the data for SAOM estimation. If using pooled estimation on multiple `sienaFit` objects (i.e., providing a list of `sienaFit` objects), then `SAOM_data` should be provided as an ordered list with each entry containing the `siena` object corresponding to list of `sienaFit` objects.
- `SAOM_var` — optional parameter when the model is a `sienaFit` object. `SAOM_var` is a list of of the `varCovar` and `varDyadCovar` objects used to assign time varying node and dyad covariates when calling `sienaDataCreate`. If provided, `netmediate` assigns the varying node covariates and dyad covariates to each simulated network. This parameter is required when `macro_function` computes a statistic that varies as a function of time varying node or dyad covariates (i.e., network segregation, assortativity). Time invariant characteristics (`coCovar` and `coDyadCovar`) are handled internally by `MEMS` and should not be provided. When providing a list of `sienaFit` objects for pooled estimation, `SAOM_var` should be provided as a list of lists, where each entry in the list contains a list of `varCovar` and `varDyadCovar` objects associated with corresponding `sienaFit` object.
- `time_interval` — an optional parameter to be used with `rem.dyad` objects. May be provided as a numeric vector or the character string `"aggregate"`. If a numeric vector is provided unique network snapshots at each interval. For example, `time_interval=c(0,2,3)`

would induce two networks, one for the 0 - 2 time period and one for the 2 - 3 time period. If specified as "aggregate", the MEMS is calculated by creating an aggregated cross-sectional representation of the entire event sequence. If left NULL, defaults to l"aggregate".

covar_list
: an optional list of sender/receiver covariates used in rem.dyad estimation. Only required for rem.dyad objects when covariates are included. The list format should correspond to the format required by rem.dyad

.

edgelist
: an optional three column edgelist providing the sender, receiver, and time of event occurrence when using rem.rem.dyad. Only required when time_interval is set to NULL or "aggregate". Ignored for other types of models.

net_logit_y
: the dependent variable for netlogit objects. Should be provided as a vector. Only required when model is a netlogit object.

net_logit_x
: the matrix of independent variables for netlogit type objects. Only required when model is a netlogit object.

group_id
: optional vector of group identifiers to use when estimating a glm or glmer on grouped data (i.e., multiple time periods, multiple networks). When specified, MEMS will induce unique networks for each grouping factor. If left unspecified, all groups/time periods are pooled. If using glmer, the grouping factor does not have to be provided as part of the model or used as a random effect.

node_numbers
: a numeric vector containing the number of nodes in each group_id when using glm or glmer. If estimating MEMS aggregated over all networks (i.e., group_id=NULL), this shoud be the total number of nodes in all networks. Required when using glm or glmer, ignored otherwise.

mediator
: a character string detailing the mediator of interest. Intended for internal use with the AMME function; not intended for end users.

link_id
: a vector or list of vectors corresponding to unique identifiers. Intended for internal use with the AMME function; not intended for end users.

controls
: a vector of character strings listing the controls to be calculated when using AMME. Intended for internal use with the AMME function; not intended for end users.

control_functions
: a list of functions to calculate the macro control variables provided in controls. Intended for internal use with the AMME function; not intended for end users.

Details

Estimates the MEMS over the provided intervals. If the macro statistic is calculated on the node or subgraph levels or on multiple network observations, the aMEMS is provided instead. Standard errors and confidence intervals are based on the sampling distribution of simulated values, which are calculated either parametrically or nonparametrically according to algorithm. Parametric estimation is typically faster, but cannot be used for nonparametric network models (e.g., quadratic assignment procedure).

macro_function is the workhorse component of MEMS. The function should calculate the macro statistic of interest. netmediate currently supports functions calculated on igraph and network objects, which should be specified as using the object_type argument. These may be functions inherent to the statnet and igraph software package or they may be functions from other packages that accept network/igraph objects. They may also be user-defined functions that accept network or igraph objects as input and return a numeric value or vector of numeric values as output. It is also possible to over-ride the network and igraph object requirements within a user function. To do so, set the object_type argument to either network or igraph and then define a user-function that accepts a network or igraph object as its input, converts the object to the desired data structure, calculates the statistic of interest, and finally returns a numeric value or vector of numeric values. See examples below for an illustration.

By default, the MEMS is provided by averaging over the distribution of simulated values. If full_output is set to TRUE, the entire distribution of simualted statistics is returned. This may be useful when the median or mode of the simulated distribution is required or if the researcher wants to inspect the distributional shape of simulated values.

MEMS also supports pooled estimation for multiple ergm or sienaFit objects. To use pooled estimation, the model parameter should be specified as a list of ergm or sienaFit objects. If using sienaFit, the SAOM_data argument will also need to be specified as an ordered list with elements corresponding to entries in the list of sienaFit objects. Similarly, the SAOM_var parameter will need to be specified as a list of lists, where each entry in the list is, itself, a list containing all varCovar and varDyadCovar objects used to calculate macro statistics of interest. Note that SAOM_var should not be provided if the macro statistic of interest is not a function of the variables contained in varCovar and varDyadCovar.

When estimating a relational event model with a rem.dyad object, time_interval can be specified to provide exact time intervals over which to induce unique networks. This utility is often useful when combining rem.dyad estimation with AMME when the macro_model is panel data with coarse timing information. The same behavior can be obtained when estimating a relational event model using glm or glmer by assigning the desired time intervals in the model matrix and then providing the vector of time intervals to the group_id parameter when calling MEMS.

Value

If full_output=FALSE, then a table is returned with the MEMS, its standard error, confidence interval, and p-value.

If full_output=TRUE, then a list is returned with the following three elements.

summary_dat
: is the table of summary output containing the MEMS, its standard error, confidence interval, and p-value.

output_data
: is a matrix where each row is a simulated draw of the MEMS (or a simulation draw for a specific network in the case of temporal data or pooled estimation) and each column corresponds to a unique value provided in the interval argument.

mems_samples
: is vector matrix corresponding where each row is a simulated draw of the MEM (or a simulation draw for a specific network in the case of temporal data or pooled estimation) and each column represents the differences in MEMS/aMEMS when subtracting the value of a macro statistic at one interval level from the next highest interval level.

Author(s)

Duxbury, Scott W. Associate Professor, University of North Carolina–Chapel Hill, Department of Sociology.

References

Duxbury, Scott W. 2024. "Micro Effects on Macro Structure in Social Networks." Sociological Methodology.

See Also

AMME ergm.mma mediate

Examples

```
########################################
# ERGM examples and basic utilities
#######################################


####start with a simple model
library(statnet)

data("faux.mesa.high")

model1<-ergm(faux.mesa.high~edges+
               nodecov("Grade")+
               nodefactor("Race")+
               nodefactor("Sex")+
               nodematch("Race")+
               nodematch("Sex")+
               absdiff("Grade"))



##calculate the MEMS when the absolute difference in grade is changed from an interval of 0 to 1
  #with default specifications for gtrans
MEMS(model1,
     micro_process="absdiff.Grade",
     macro_function = gtrans,
     object_type = "network",
     nsim=100,
     interval=c(0,1),
     silent=FALSE,
     algorithm = "parametric")

#call an argument from gtrans by specifying it as a function
  #use nonparametric estimation
MEMS(model1,
     micro_process="absdiff.Grade",
```

```
     macro_function = function(x){gtrans(x,measure="strongcensus")},
     object_type = "network",
     nsim=100,
     interval=c(0,1),
     silent=FALSE,
     algorithm = "nonparametric")




####calculate the MEMS using igraph
MEMS(model1,
     micro_process="absdiff.Grade",
     macro_function = function(x){igraph::transitivity(x,type="local")},
     object_type = "igraph",
     nsim=100,
     interval=c(0,1),
     silent=FALSE,
     algorithm = "parametric")



##specify a user function that counts the number of communities
community_counts<-function(x){
  walktrap<-igraph::walktrap.community(x) #use walktrap community detection
  return(length(unique(walktrap$membership))) #return the number of communities
}

MEMS(model1,
     micro_process="absdiff.Grade",
     macro_function = community_counts,
     object_type = "igraph",
     nsim=100,
     interval=c(0,1),
     silent=FALSE,
     algorithm = "parametric")



##calculate a function using exogenous node attributes
assortativity_grade<-function(x){
  require(igraph)
  return(assortativity_nominal(x,V(x)$Grade))
}

MEMS(model1,
     micro_process="absdiff.Grade",
     macro_function = assortativity_grade,
     object_type = "igraph",
     nsim=100,
     interval=c(0,1),
     silent=FALSE,
     algorithm = "parametric")
```

```
##specify a user function that does not depend on either igraph or statnet
  #assuming a network input object, we have
manual_user_function<-function(x){
  x<-as.sociomatrix(x)
  return(colSums(x))
}

MEMS(model1,
     micro_process="absdiff.Grade",
     macro_function = manual_user_function,
     object_type = "network",
     nsim=100,
     interval=c(0,1),
     silent=FALSE,
     algorithm = "parametric")







####estimation for POOLED ERGM
data("faux.magnolia.high")

model2<-ergm(faux.magnolia.high~edges+
               nodecov("Grade")+
               nodefactor("Race")+
               nodefactor("Sex")+
               nodematch("Race")+
               nodematch("Sex")+
               absdiff("Grade"))



MEMS(list(model1,model2),
     micro_process="absdiff.Grade",
     macro_function = assortativity_grade,
     object_type = "igraph",
     nsim=50,
     interval=c(0,1),
     silent=FALSE,
     algorithm = "parametric")



#################################
#   Estimation with GLM and GLMER
#################################
library(btergm)

#use models 1 and 2 from examples above
```

```
glm_dat<-edgeprob(model1)
glm_dat2<-edgeprob(model2)
glm_dat2<-glm_dat2[,-c(4)]


##create stacked dataset for the purposes of grouped estimation
glm_dat$net_id<-"mesa" #specify ID for each network
glm_dat2$net_id<-"magnolia"
glm_dat<-rbind(glm_dat,glm_dat2)


##estimate as a linear probability model
net_glm<-glm(tie~nodecov.Grade+
               nodefactor.Race.Hisp+
               nodefactor.Race.NatAm+
               nodefactor.Race.Other+
               nodefactor.Sex.M+
               nodematch.Race+
               nodematch.Sex+
               absdiff.Grade,
             data=glm_dat)



MEMS(net_glm,
     micro_process="nodematch.Race", #should be written as in netlogit output
     macro_function = function(x){gtrans(x)},
     object_type = "network",
     nsim=100,
     interval=c(0,.5),
     silent=FALSE,
     full_output = FALSE,
     algorithm = "parametric",
     group_id=glm_dat$net_id, #provide network ID for estimation
   node_numbers =c(network.size(faux.mesa.high), #provide the number of nodes in each network
                   network.size(faux.magnolia.high)))


##estimate as a multilevel model
library(lme4)
net_glmer<-glmer(tie~nodecov.Grade+
               nodefactor.Race.Hisp+
               nodefactor.Race.NatAm+
               nodefactor.Race.Other+
               nodefactor.Sex.M+
               nodematch.Race+
               nodematch.Sex+
               absdiff.Grade+
                 (1|net_id),
             data=glm_dat,
             family=gaussian)
```

```
MEMS(net_glmer,
     micro_process="nodematch.Race", #should be written as in netlogit output
     macro_function = function(x){gtrans(x)},
     object_type = "network",
     nsim=50,
     interval=c(0,.5),
     silent=FALSE,
     full_output = FALSE,
     algorithm = "parametric",
     group_id=glm_dat$net_id,
     node_numbers =c(203,974))



##############################################
##nonparametric estimation for bootstrap TERGM
##############################################

library(btergm)
data(alliances)
ally_data<-list(LSP[[1]],
                LSP[[2]],
                LSP[[3]])

bt_model<-btergm(ally_data~edges+
                   gwesp(.7,fixed=T)+
                   mutual,R=200)



MEMS(bt_model,
     micro_process="gwesp.OTP.fixed.0.7",
     macro_function = gtrans,
     object_type = "network",
     nsim=50,
     interval=c(0,1),
     silent=FALSE,
     algorithm = "nonparametric")





################################
# Parametric estimation using SAOM
##################################
library(RSiena)
#specify 3 wave network panel data as DV
network_list<-array(c(s501,s502,s503),dim = c(50,50,3))

Network<-sienaDependent(network_list)
```

```
Smoking<-varCovar(s50s)
Alcohol<-varCovar(s50a)
SAOM.Data<-sienaDataCreate(Network=Network,Smoking,Alcohol)

#specify
SAOM.terms<-getEffects(SAOM.Data)
SAOM.terms<-includeEffects(SAOM.terms,egoX,altX,sameX,interaction1="Alcohol")
SAOM.terms<-includeEffects(SAOM.terms,egoX,altX,sameX,interaction1="Smoking")
SAOM.terms<-includeEffects(SAOM.terms,transTies,inPop)


create.model<-sienaAlgorithmCreate(projname="netmediate",
                                   nsub=5,
                                   n3=2000)


##estimate the model using siena07
SAOM_model<-siena07(create.model,
                        data=SAOM.Data,
                        effects=SAOM.terms,
                        verbose=TRUE)


SAOM_model




##basic specification for reciprocity effects on outdegree distribution
MEMS(SAOM_model,
     micro_process="reciprocity", #should be written as in SIENA output
     macro_function = function(x){igraph::degree(x,mode="out")},
     object_type = "igraph",
     interval=c(0,.5),
     SAOM_data=SAOM.Data,
     silent=FALSE,
     algorithm = "parametric")



##include user functions on time varying covariates
assortativity_smoking<-function(x){
  return(assortativity_nominal(x,V(x)$Smoking))
}


MEMS(SAOM_model,
     micro_process="reciprocity",
     macro_function = assortativity_smoking,
     object_type = "igraph",
     interval=c(0,.5),
     SAOM_data=SAOM.Data,
   SAOM_var=list(Smoking=Smoking,Alcohol=Alcohol), #Smoking and Alcohol are varCovar objects
```

```
     silent=FALSE,
     full_output = FALSE,
     algorithm = "parametric")




###Pooled SAOM
MEMS(list(SAOM_model,SAOM_model),
     micro_process="reciprocity",
     macro_function = gtrans,
     object_type = "network",
     interval=c(0,.5),
     SAOM_data=list(SAOM.Data,SAOM.Data),
     silent=FALSE,
     full_output = FALSE,
     nsim=100,
     algorithm = "parametric")


#Pooled SAOM with user functions and time varying attributes
assortativity_smoking<-function(x){
  return(assortativity_nominal(x,V(x)$Smoking))
}



MEMS(list(SAOM_model,SAOM_model),
     micro_process="reciprocity",
     macro_function = assortativity_smoking,
     object_type = "igraph",
     interval=c(0,.5),
     SAOM_data=list(SAOM.Data,SAOM.Data),
     SAOM_var=list(list(Smoking=Smoking,Alcohol=Alcohol),
                   list(Smoking=Smoking,Alcohol=Alcohol)),
     silent=FALSE,
     full_output = FALSE,
     nsim=100,
     algorithm = "parametric")







################################################
## Selection and Influence in SAOM when analyzing
## co-evolution of networks and behavior
################################################
```

```
##Example Moran decomposition
library(RSiena)

###run the model--taken from RSiena scripts

# prepare first two waves of s50 data for RSiena analysis:
(thedata <- sienaDataCreate(
  friendship = sienaDependent(array(
    c(s501,s502),dim=c(50,50,2))),
  drinking = sienaDependent(s50a[,1:2])
))

# specify a model with (generalised) selection and influence:
themodel <- getEffects(thedata)
themodel <- includeEffects(themodel,name='friendship',gwespFF)
themodel <- includeEffects(themodel,name='friendship',simX,interaction1='drinking')
themodel <- includeEffects(themodel,name='drinking',avSim,interaction1='friendship')
themodel



# estimate this model:
estimation.options <- sienaAlgorithmCreate(projname='results',cond=FALSE,seed=1234567)
(theresults <- siena07(estimation.options,data=thedata,effects=themodel))



##calculate MEMS for selection effect
  #Uses Moran_dv--a function internally called by netmediate
  #to calculate change in amount of network autocorrelation
  #as a function of both endogenous behavior and network dependent
  #variables

MEMS(theresults,
     micro_process="drinking similarity",
     macro_function =Moran_dv,
     object_type = "network",
     SAOM_data = thedata,
     silent=FALSE,
     nsim=50)

#just influence
MEMS(theresults,
     micro_process="drinking average similarity",
     macro_function =Moran_dv,
     object_type = "network",
     SAOM_data = thedata,
     silent=FALSE,
     nsim=50)

##joint effect of selection and influence
MEMS(theresults,
     micro_process=c("drinking similarity","drinking average similarity"),
```

```
     macro_function =Moran_dv,
     object_type = "network",
     SAOM_data = thedata,
     silent=FALSE,
     nsim=500)
```

```
#######################################
# Relational event models using relevent
#######################################
set.seed(21093)
library(relevent)
##generate a network with 15 discrete time periods
  #example based on relevent rem.dyad example
library(relevent)
roweff<-rnorm(10) #Build rate matrix
roweff<-roweff-roweff[1] #Adjust for later convenience
coleff<-rnorm(10)
coleff<-coleff-coleff[1]
lambda<-exp(outer(roweff,coleff,"+"))
diag(lambda)<-0
ratesum<-sum(lambda)
esnd<-as.vector(row(lambda)) #List of senders/receivers
erec<-as.vector(col(lambda))
time<-0
edgelist<-vector()
while(time<15){ # Observe the system for 15 time units
  drawsr<-sample(1:100,1,prob=as.vector(lambda)) #Draw from model
  time<-time+rexp(1,ratesum)
  if(time<=15) #Censor at 15
    edgelist<-rbind(edgelist,c(time,esnd[drawsr],erec[drawsr]))
  else
    edgelist<-rbind(edgelist,c(15,NA,NA))
}
effects<-c("CovSnd","FERec")



##estimate model
fit.time<-rem.dyad(edgelist,10,effects=effects,
                   covar=list(CovSnd=roweff),
                   ordinal=FALSE,hessian=TRUE)


###aggregate estimation
MEMS(fit.time,
     micro_process="CovSnd.1", #should be written as in relevent output
     macro_function = function(x){sna::degree(x)},
```

```
     object_type = "network",
     nsim=10,
     interval=c(0,.5),
     silent=FALSE,
     covar_list=list(CovSnd=roweff), #covariate effects
     time_interval="aggregate", ##aggregated estimation
     edgelist=edgelist,
     algorithm = "parametric")


##time interval estimation
##estimation with time intervals
MEMS(fit.time,
     micro_process="CovSnd.1",
     macro_function = function(x){igraph::degree(x)},
     object_type = "igraph",
     nsim=10,
     interval=c(0,.1),
     silent=TRUE,
     covar_list=list(CovSnd=roweff),
    time_interval=c(0,5,10,15), #specify three time intervals, 0 - 5, 5 - 10, and 10 - 15
     algorithm = "parametric")






########################################################
# Network regression with quadratic assignment procedure
########################################################
library(sna)
##generate network data
set.seed(21093)
x<-rgraph(20,4)
y.l<-x[1,,]+4*x[2,,]+2*x[3,,]
y.p<-apply(y.l,c(1,2),function(a){1/(1+exp(-a))})
y<-rgraph(20,tprob=y.p)

nl<-netlogit(y,x,reps=100)
summary(nl)



MEMS(nl,
     micro_process="x2", #should be written as in netlogit output
     macro_function = function(x){degree(x)},
     object_type = "igraph",
     nsim=20,
     interval=c(0,1),
     silent=FALSE,
     full_output = FALSE,
```

```
        net_logit_y=y,
        net_logit_x=x,
        algorithm = "nonparametric")
```

Moran_dv *Function to calculate Moran's first order network autocorrelation in co-evolution SAOM.*

Description

A function to calculate Moran's first order network autocorrelation in co-evolution SAOM using both the endogenous dependent variables (behavior and network functions).

Usage

```
Moran_dv(network)
```

Arguments

`network`	a network object used to calculate autocorrelation.

Value

No return value, used internally with `MEMS` and `AMME`

Index