

# The development of the networksem Package

The package networksem includes various functions to analyze network data in the SEM framework. networksem has two major components.

- Functions started with sem.net: These are functions used to convert network data into SEM compatible formats and then embed those data in lavaan-based SEM analyses.
- Other functions: functions such as summary() and path.networksem() calculates values of interest such as mediation effect post-hoc.

## List of functions

The following functions are available in the packages. The exported functions are:

- path.networksem
- sem.net
- sem.net.edge
- sem.net.edge.lsm
- sem.net.lsm
- summary.networksem

The following functions are not imported and are used by other functions.

- sem.net.addvar.stat
- sem.net.addvar.influential
- sem.net.addvar

## sem.net

This function uses a two-stage method to obtain network statistics from networks and then fit the SEM model with them.

```
#' Fit a sem model with network data using node statistics as variables. User-specified network statistics will be
calculated and used as variables instead of the networks themselves in the SEM.
#' @param model a model specified in lavaan model syntax.
#' @param data a list containing the observed non-network nodal variables and the network data
#' @param netstats a user-specified list of network statistics to be calculated and used in the SEM, e.g.,
c("degree", "betweenness"), available options include "degree", "betweenness", "closeness", "evcent",
```

```

"stresscent", and "infocent" from the "sna" package and "ivi", "hubness.score", "spreading.score" and
"clusterRank" from the "influential" package
#' @param netstats.options a user-specified named list with element names corresponding to the network
statistics names and element values corresponding to other lists. The list corresponding to each network
statistics name has element names being the argument names for calculating the network statistics, and values
being the argument values, as used in the corresponding functions in the "sna" or "influential" packages. e.g.,
netstats.options=list("degree"=list("cmode"="freeman"), "closeness"=list("cmode"="undirected"),
"clusterRank"=list("directed"=FALSE))
#' @param netstats.rescale a list of logical value indicating whether to rescale network statistics to have mean 0
and sd 1.
#' @param data.rescale whether to rescale the whole dataset (with restructured network and nonnetwork data)
to have mean 0 and standard deviation 1 when fitting it to SEM, default to FALSE
#' @param ordered parameter same as "ordered" in the lavaan sem() function; whether to treat data as ordinal
#' @param sampling.weights parameter same as "sampling.weights" in the lavaan sem() function; whether to
apply weights to data
#' @param group parameter same as "group" in the lavaan sem() function; whether to fit a multigroup model
#' @param cluster parameter same as "cluster" in the lavaan sem() function; whether to fit a cluster model
#' @param constraints parameter same as "constraints" in the lavaan sem() function; whether to apply
constraints to the model
#' @param WLS.V parameter same as "WLS.V" in the lavaan sem() function; whether to use WLS.V estimator
#' @param NACOV parameter same as "NACOV" in the lavaan sem() function; whether to use NACOV estimator
#' @param ... optional arguments for the sem() function
#' @return the updated model specification with the network statistics as variables and a lavaan object which is
the SEM results
#' @import lavaan
#' @import sna
#' @import igraph
#' @import influential
#' @import latentnet
#' @import ergm
#' @import network
#' @export
#' @examples
#' \donttest{
#' set.seed(100)
#' nsamp = 20
#' net <- ifelse(matrix(rnorm(nsamp^2), nsamp, nsamp) > 1, 1, 0)
#' mean(net) # density of simulated network
#' lv1 <- rnorm(nsamp)
#' lv2 <- rnorm(nsamp)

```

```

#' nonnet <- data.frame(x1 = lv1*0.5 + rnorm(nsamp),
#'                      x2 = lv1*0.8 + rnorm(nsamp),
#'                      x3 = lv2*0.5 + rnorm(nsamp),
#'                      x4 = lv2*0.8 + rnorm(nsamp))
#'
#' model <- '
#'   lv1 =~ x1 + x2
#'   lv2 =~ x3 + x4
#'   net ~ lv2
#'   lv1 ~ net + lv2
#' '
#' data = list(network = list(net = net), nonnetwork = nonnet)
#' set.seed(100)
#' res <- sem.net(model = model, data = data, netstats = c('degree'))
#' summary(res)
#' }

sem.net <- function(model=NULL, data=NULL, netstats=NULL,
                    ordered = NULL, sampling.weights = NULL, data.rescale = FALSE,
                    netstats.rescale = FALSE, group = NULL, cluster = NULL,
                    constraints = "", WLS.V = NULL, NACOV = NULL,
                    netstats.options=NULL, ...){
  ## checking proper input
  if(is.null(model)){
    stop("required argument model is not specified.")
  }
  if(is.null(data)){
    stop("required argument data is not specified.")
  }

  params <- c(as.list(environment()), list(...))

  ## get the variable names in the model
  model.info <- lavaan::lavParseModelString(model)
  model.var <- unique(c(model.info$lhs, model.info$rhs))

  ## non-network data variable names
  data.nonnetwork.var <- names(data$nonnetwork)

```

```

## network data variable names
if (!is.null(data$network)){
  data.network.var <- names(data$network)
}

## find the network variables in the model
model.network.var <- data.network.var[data.network.var %in% model.var]

## create variables for network data and model
## add network data variables to the non-network data
model.network.stat.var.list <- list()
if (length(model.network.var) > 0){
  if (is.null(netstats)){
    ## loop through the statistics
    for (i in 1:length(model.network.var)){
      ## call helper function, which loops over all target statistics to be used
      res.tmp <- sem.net.addvar(model.network.stat.var.list, data, c("degree"), model.network.var[i])
      model.network.stat.var.list <- res.tmp[[1]]
      data$nonnetwork <- res.tmp[[2]]
    }
  }else{
    ## loop through the variables and statistics
    for (i in 1:length(model.network.var)){
      res.tmp <- sem.net.addvar(model.network.stat.var.list, data, netstats, model.network.var[i],
netstats.rescale, netstats.options)
      model.network.stat.var.list <- res.tmp[[1]]
      data$nonnetwork <- res.tmp[[2]]
    }
  }
}

## reconstruct the path model with the network variables
## replace the network variable name with the network variable stats name

## lavaanify the model
model.lavaanify <- lavaan::lavaanify(model)

## get the use specified model information
model.user <- model.lavaanify[model.lavaanify$user==1, ]

```

```

## now process each part of the user specified model
model.to.remove.index <- NULL # row index of the model items to remove
model.to.add <- ""
for (i in 1:nrow(model.user)){
  ## check if the variable on the lhs is a network variable
  if (model.user$lhs[i] %in% model.network.var && !(model.user$rhs[i] %in% model.network.var)){
    ## if it is, record the index i and create new model items for te network
    model.to.remove.index <- c(model.to.remove.index, i)
    model.stat.var.to.add <- model.network.stat.var.list[[model.user$lhs[i]]]
    for (j in 1:length(model.stat.var.to.add)){
      model.temp <- paste0("\n", model.stat.var.to.add[j], model.user$op[i], model.user$rhs[i])
      model.to.add <- paste0(model.to.add, model.temp)
    }
  }

  ## check if the variable on the rhs is a network variable and the lhs is not
  if (model.user$rhs[i] %in% model.network.var && !(model.user$lhs[i] %in% model.network.var)){
    ## record the index i and create new model items
    model.to.remove.index <- c(model.to.remove.index, i)
    model.stat.var.to.add <- model.network.stat.var.list[[model.user$rhs[i]]]
    for (j in 1:length(model.stat.var.to.add)){
      model.temp <- paste0("\n", model.user$lhs[i], model.user$op[i], model.stat.var.to.add[j])
      model.to.add <- paste0(model.to.add, model.temp)
    }
  }

  ## check if both lhs and rhs are network variables
  if (model.user$rhs[i] %in% model.network.var && model.user$lhs[i] %in% model.network.var){
    ## if it is, record the index i and create new model items
    model.to.remove.index <- c(model.to.remove.index, i)
    model.stat.var.to.add.rhs <- model.network.stat.var.list[[model.user$rhs[i]]]
    model.stat.var.to.add.lhs <- model.network.stat.var.list[[model.user$lhs[i]]]
    for (j in 1:length(model.stat.var.to.add.rhs)){
      for (k in 1:length(model.stat.var.to.add.lhs)){
        model.temp <- paste0("\n", model.stat.var.to.add.lhs[j], model.user$op[i], model.stat.var.to.add.rhs[k])
        model.to.add <- paste0(model.to.add, model.temp)
      }
    }
  }
}

```

```

model.remove.network.var <- model.user[-model.to.remove.index, ] # remove initial model specification
# add altered model specification
model.non.network.var <- ""
if (nrow(model.remove.network.var) > 0 ){
  for (i in 1:nrow(model.remove.network.var)){
    model.non.network.var.temp <- paste0(paste0(model.remove.network.var[i, c('lhs', 'op', 'rhs')], collapse = '
'))
    model.non.network.var <- paste0(model.non.network.var.temp, "\n", model.non.network.var)
  }
}

model.full <- paste0(model.non.network.var, "\n", model.to.add)

# if(!is.null(community)){
#   communities_clust <- cutree(sna::equiv.clust(network)$cluster, k=community)
#   data["communities_clust"]<-communities_clust
# }
#

# if(!is.null(community) || !is.null(group)){
#   group <- ifelse(!is.null(community), "communities_clust", group)
# }
#

lavparams <- list()
for (i in 1:length(params)){
  if (names(params)[i] %in% names(lavaan::lavOptions())){
    lavparams[[names(params)[i]]] <- params[[i]]
  }
}

if (data.rescale){
  for (i in 1:ncol(data$nonnetwork)){
    if (is.numeric(data$nonnetwork[,i])){
      data$nonnetwork[,i] <- scale(data$nonnetwork[,i], center = TRUE, scale = TRUE)
    }
  }
}

```

```

    }
  }

  lavparams[["data"]] <- data$nonnetwork
  lavparams[["model"]] <- model.full
  lavparams[["ordered"]] <- ordered
  lavparams[["sampling.weights"]] <- sampling.weights
  lavparams[["group"]] <- group
  lavparams[["cluster"]] <- cluster
  lavparams[["constraints"]] <- constraints
  lavparams[["WLS.V"]] <- WLS.V
  lavparams[["NACOV"]] <- NACOV

  model.res <- do.call(what="sem", args=c(lavparams))

  obj <- list(model=model.full, estimates=model.res, data=data)
  class(obj) <- "networksem"
  return(obj)
}

```

## sem.net.edge

This function uses a two-stage method to get latent positions and/or network statistics as variables and then fit the SEM model with them.

```

#' Fit a sem model with network data using node latent positions and/or network statistics as variables. User-
specified network statistics will be calculated and used as variables instead of the networks themselves in the
SEM.

#' @param model a model specified in lavaan model syntax.
#' @param data a list containing both the non-network and network data
#' @param netstats.rescale a logical value indicating whether to rescale network statistics or variables to have
mean 0 and sd 1
#' @param data.rescale whether to rescale the whole dataset (with restructured network and nonnetwork data)
to have mean 0 and standard deviation 1 when fitting it to SEM, default to FALSE
#' @param ordered parameter same as "ordered" in the lavaan sem() function; whether to treat data as ordinal
#' @param sampling.weights parameter same as "sampling.weights" in the lavaan sem() function; whether to
apply weights to data

```

```

#' @param group parameter same as "group" in the lavaan sem() function; whether to fit a multigroup model
#' @param cluster parameter same as "cluster" in the lavaan sem() function; whether to fit a cluster model
#' @param constraints parameter same as "constraints" in the lavaan sem() function; whether to apply
constraints to the model
#' @param WLS.V parameter same as "WLS.V" in the lavaan sem() function; whether to use WLS.V estimator
#' @param NACOV parameter same as "NACOV" in the lavaan sem() function; whether to use NACOV estimator
#' @param latent.dim number of network latent dimensions to use
#' @param ... optional arguments for the sem() function
#' @return the updated model specification with the network statistics as variables and a lavaan object which is
the SEM results
#' @export
#' @examples
#' \dontrun{
#' \donttest{
#' set.seed(10)
#' nsamp = 20
#' net <- ifelse(matrix(rnorm(nsamp^2), nsamp, nsamp) > 1, 1, 0)
#' mean(net) # density of simulated network
#' lv1 <- rnorm(nsamp)
#' lv2 <- rnorm(nsamp)
#' nonnet <- data.frame(x1 = lv1*0.5 + rnorm(nsamp),
#'                      x2 = lv1*0.8 + rnorm(nsamp),
#'                      x3 = lv2*0.5 + rnorm(nsamp),
#'                      x4 = lv2*0.8 + rnorm(nsamp))
#'
#' model <- '
#'   lv1 =~ x1 + x2
#'   lv2 =~ x3 + x4
#'   net ~ lv2
#'   lv1 ~ net + lv2
#' '
#' data = list(network = list(net = net), nonnetwork = nonnet)
#' set.seed(100)
#' res <- sem.net.lsm(model = model, data = data, latent.dim = 2)
#' summary(res)
#' }}
sem.net.lsm <- function(model=NULL, data=NULL, latent.dim = 2,
                        ordered = NULL, sampling.weights = NULL, data.rescale=FALSE,
                        netstats.rescale=FALSE, group = NULL, cluster = NULL,
                        constraints = "", WLS.V = NULL, NACOV = NULL, ...){

```



```

## checking proper input
if(is.null(model)){
  stop("required argument model is not specified.")
}
if(is.null(data)){
  stop("required argument data is not specified.")
}

params <- c(as.list(environment()), list(...))

## get the variable names in the model
model.info <- lavaan::lavParseModelString(model)
model.var <- unique(c(model.info$lhs, model.info$rhs))

## non-network data variable names
data.nonnetwork.var <- names(data$nonnetwork)

## network data variable names
if (!is.null(data$network)){
  data.network.var <- names(data$network)
}
latent.network = data.network.var

## find the network variables in the model
model.network.var <- data.network.var[data.network.var %in% model.var]

## create variables for network data and model
## add network data variables to the non-network data

latent.vars <- list()
model.lavaanify <- lavaan::lavaanify(model)

## get the use specified model information
model.user <- model.lavaanify[model.lavaanify$user==1, ]

## estimate network latent positions
lsm.fits <- list()
for (i in 1:length(latent.network)){

```

```

fit <- latentnet::ergmm(network::network(data$network[[latent.network[i]]]) ~ euclidean(d = latent.dim))
lsm.fits[[i]] <- fit
latent.vars[[latent.network[i]]] <- c()
for (dimind in 1:latent.dim){
  data$nonnetwork[paste0(latent.network[i], ".Z", dimind)] <- fit$mcmc.mle$Z[,dimind]
  if (netstats.rescale){
    data$nonnetwork[paste0(latent.network[i], ".Z", dimind)] <- scale(fit$mcmc.mle$Z[,dimind], center =
TRUE, scale = TRUE)
  }
  latent.vars[[latent.network[i]]] <- c(latent.vars[[latent.network[i]]], paste0(latent.network[i], ".Z", dimind))
}
}

# print(lsm.fits)

## reconstruct the path model with the network variables
## replace the network variable name with the network variable stats name

## lavaanify the model
model.lavaanify <- lavaan::lavaanify(model)

## get the use specified model information
model.user <- model.lavaanify[model.lavaanify$user==1, ]

## now process each part of the user specified model
model.to.remove.index <- NULL
model.to.add <- ""
for (i in 1:nrow(model.user)){
  ## check if left is network with LSM, remake
  if (model.user$lhs[i] %in% latent.network){
    model.to.remove.index <- c(model.to.remove.index, i)
    model.stat.var.to.add <- latent.vars[[model.user$lhs[i]]]
    for (j in 1:length(model.stat.var.to.add)){
      model.temp <- paste0("\n ", model.stat.var.to.add[j], model.user$op[i], model.user$rhs[i])
      model.to.add <- paste0(model.to.add, model.temp)
    }
  }
}

```

```

}

## check if right is network with LSM and left is other variables
if (model.user$rhs[i] %in% latent.network){
  model.to.remove.index <- c(model.to.remove.index, i)
  model.stat.var.to.add <- latent.vars[[model.user$rhs[i]]]
  for (j in 1:length(model.stat.var.to.add)){
    model.temp <- paste0("\n ", model.user$lhs[i], model.user$op[i], model.stat.var.to.add[j])
    model.to.add <- paste0(model.to.add, model.temp)
  }
}

}

if (!is.null(model.to.remove.index)){
  model.remove.network.var <- model.user[-model.to.remove.index, ]
}

model.non.network.var <- ""
if(nrow(model.remove.network.var) > 0){
  for (i in 1:nrow(model.remove.network.var)){
    model.non.network.var.temp <- paste0(paste0(model.remove.network.var[i, c('lhs', 'op', 'rhs')], collapse = '
'))
    model.non.network.var <- paste0(model.non.network.var.temp, "\n", model.non.network.var)
  }
}

model.full <- paste0(model.non.network.var, "\n", model.to.add)

lavparams <- list()
for (i in 1:length(params)){
  if (names(params)[i] %in% names(lavOptions())){
    lavparams[[names(params)[i]]] <- params[[i]]
  }
}

if (data.rescale){
  for (i in 1:ncol(data$nonnetwork)){
    if (is.numeric(data$nonnetwork[,i])){

```

```

      data$nonnetwork[,i] <- scale(data$nonnetwork[,i], center = TRUE, scale = TRUE)
    }
  }
}

lavparams[["data"]] <- data$nonnetwork
lavparams[["model"]] <- model.full
lavparams[["ordered"]] <- ordered
lavparams[["sampling.weights"]] <- sampling.weights
lavparams[["group"]] <- group
lavparams[["cluster"]] <- cluster
lavparams[["constraints"]] <- constraints
lavparams[["WLS.V"]] <- WLS.V
lavparams[["NACOV"]] <- NACOV

model.res <- do.call(what="sem", args=c(lavparams))

obj <- list(model=model.full, estimates=list(sem.es=model.res,lsm.es=lsm.fits), data=data)
class(obj) <- "networksem"
return(obj)
}

```

## sem.net.lsm

This function will first get the edge information and the fit a sem model.

```

#' Fit a sem model with network data using edges as variables. User-specified network statistics will be
calculated and used as variables instead of the networks themselves in the SEM.
#' @param model a model specified in lavaan model syntax.
#' @param data a list containing both the non-network and network data
#' @param type "difference" for using the difference between the network statistics of the two actors as the
edge covariate; "average" for using the average of the network statistics of the two actors as the edge covariate
#' @param ordered parameter same as "ordered" in the lavaan sem() function; whether to treat data as ordinal
#' @param data.rescale whether to rescale the whole dataset (with restructured network and nonnetwork data)
to have mean 0 and standard deviation 1 when fitting it to SEM, default to FALSE
#' @param netstats.rescale a logical value indicating whether to rescale network statistics or variables to have
mean 0 and sd 1

```

```

#' @param sampling.weights parameter same as "sampling.weights" in the lavaan sem() function; whether to
apply weights to data
#' @param group parameter same as "group" in the lavaan sem() function; whether to fit a multigroup model
#' @param cluster parameter same as "cluster" in the lavaan sem() function; whether to fit a cluster model
#' @param constraints parameter same as "constraints" in the lavaan sem() function; whether to apply
constraints to the model
#' @param WLS.V parameter same as "WLS.V" in the lavaan sem() function; whether to use WLS.V estimator
#' @param NACOV parameter same as "NACOV" in the lavaan sem() function; whether to use NACOV estimator
#' @param ... optional arguments for the sem() function
#' @return the updated model specification and a lavaan object which is the SEM results, and the data
generated
#' @export
#' @examples
#' \dontrun{
#' \donttest{
#' set.seed(100)
#' nsamp = 20
#' net <- data.frame(ifelse(matrix(rnorm(nsamp^2), nsamp, nsamp) > 1, 1, 0))
#' mean(net) # density of simulated network
#' lv1 <- rnorm(nsamp)
#' lv2 <- rnorm(nsamp)
#' nonnet <- data.frame(x1 = lv1*0.5 + rnorm(nsamp),
#'                      x2 = lv1*0.8 + rnorm(nsamp),
#'                      x3 = lv2*0.5 + rnorm(nsamp),
#'                      x4 = lv2*0.8 + rnorm(nsamp))
#'
#' model <- '
#'   lv1 =~ x1 + x2
#'   lv2 =~ x3 + x4
#'   lv1 ~ net
#'   lv2 ~ lv1
#' '
#' data = list(network = list(net = net), nonnetwork = nonnet)
#' set.seed(100)
#' res <- sem.net.edge(model = model, data = data, type = 'difference')
#' summary(res)
#' }}
sem.net.edge <- function(model = NULL, data = NULL, type = "difference",
                        ordered = NULL, sampling.weights = NULL, data.rescale = FALSE,
                        group = NULL, cluster = NULL, netstats.rescale = FALSE,

```

```

        constraints = "", WLS.V = NULL, NACOV = NULL,
        ...){
## checking proper input
if(is.null(model)){
  stop("required argument model is not specified.")
}
if(is.null(data)){
  stop("required argument data is not specified.")
}

params <- c(as.list(environment()), list(...))

## get the variable names in the model
model.info <- lavParseModelString(model)
model.var <- unique(c(model.info$lhs, model.info$rhs))

## non-network data variable names
data.nonnetwork.var <- names(data$nonnetwork)

## network data variable names
if (!is.null(data$network)){
  data.network.var <- names(data$network)
}

## find the network variables in the model
model.network.var <- data.network.var[data.network.var %in% model.var]

## create variables for network data and model
## add network data variables to the non-network data
model.network.stat.var.list <- list()
data_edge = data.frame(row_actor=rep(NA, nrow(data$nonnetwork)^2), col_actor=rep(NA,
nrow(data$nonnetwork)^2))
for (i in 1:length(model.network.var)){
  data_edge[model.network.var[i]]=NA
}
if (length(model.network.var)>0){
  for (i in 1:nrow(data$nonnetwork)){
    for (j in 1:nrow(data$nonnetwork)){

```

```

data_edge[j+(i-1)*nrow(data$nonnetwork), "row_actor"]=i
data_edge[j+(i-1)*nrow(data$nonnetwork), "col_actor"]=j
for (netind in 1:length(model.network.var)){
  data_edge[j+(i-1)*nrow(data$nonnetwork), model.network.var[netind]]=data$network[[netind]][i,j]

}
}
}

if(netstats.rescale){
  for (netind in 1:length(model.network.var)){
    data_edge[model.network.var[netind]]=scale(data_edge[model.network.var[netind]], center=TRUE,
scale=TRUE)
  }
}

#print(model.network.stat.var.list)
## reconstruct the path model with the network variables
## replace the network variable name with the network variable stats name

## lavaanify the model
model.lavaanify <- lavaanify(model)

## get the use specified model information
model.user <- model.lavaanify[model.lavaanify$user==1, ]

## now process each part of the user specified model
model.to.remove.index <- NULL
model.to.add <- ""

variables.to.change=c()
for (i in 1:nrow(model.user)){
  ## check if the variable on the lhs is a nonnetwork variable
  if (model.user$lhs[i] %in% colnames(data$nonnetwork) && !model.user$lhs[i] %in% model.network.var){
    variables.to.change <- c(variables.to.change, model.user$lhs[i])
  }
  if (model.user$rhs[i] %in% colnames(data$nonnetwork) && !model.user$rhs[i] %in% model.network.var){

```

```

    variables.to.change <- c(variables.to.change, model.user$rhs[i])
  }
}

for (i in 1:length(variables.to.change)){
  data_edge[variables.to.change[i]]=NA
}

if (length(variables.to.change)>0){
  for (vind in 1:length(variables.to.change)){
    v_row <- rep(data$nonnetwork[variables.to.change[vind]][[1]], each=nrow(data$nonnetwork))
    v_col <- rep(data$nonnetwork[variables.to.change[vind]][[1]], nrow(data$nonnetwork))
    if (type=="difference"){
      data_edge[variables.to.change[vind]] <- v_row - v_col
    }else if (type=="average"){
      data_edge[variables.to.change[vind]] <- (v_row + v_col)/2
    }

  }
}

lavparams <- list()
for (i in 1:length(params)){
  if (names(params)[i] %in% names(lavOptions())){
    lavparams[[names(params)[i]]] <- params[[i]]
  }
}

if (data.rescale){
  for (i in 1:ncol(data_edge)){
    if (is.numeric(data_edge[,i])){
      data_edge[,i] <- scale(data_edge[,i], center = TRUE, scale = TRUE)
    }
  }
}

```



```

lavparams[["data"]] <- data_edge
lavparams[["model"]] <- model
lavparams[["ordered"]] <- ordered
lavparams[["sampling.weights"]] <- sampling.weights
lavparams[["group"]] <- group
lavparams[["cluster"]] <- cluster
lavparams[["constraints"]] <- constraints
lavparams[["WLS.V"]] <- WLS.V
lavparams[["NACOV"]] <- NACOV

model.res <- do.call(what="sem", args=c(lavparams))

obj <- list(model=model, estimates=model.res, data=data_edge)
class(obj) <- "networksem"
return(obj)
}

```

## sem.net.edge.lsm

This function fits a sem model with network data using latent distances between actors as variables.

```

#' Fit a sem model with network data using latent distances between actors as variables
#' @param model a model specified in lavaan model syntax.
#' @param data a list containing both the non-network and network data
#' @param type "difference" for using the difference between the network statistics of the two actors as the
edge covariate; "average" for using the average of the network statistics of the two actors as the edge covariate
#' @param ordered parameter same as "ordered" in the lavaan sem() function; whether to treat data as ordinal
#' @param sampling.weights parameter same as "sampling.weights" in the lavaan sem() function; whether to
apply weights to data
#' @param data.rescale whether to rescale the whole dataset (with restructured network and nonnetwork data)
to have mean 0 and standard deviation 1 when fitting it to SEM, default to FALSE
#' @param netstats.rescale a logical value indicating whether to rescale network statistics or variables to have
mean 0 and sd 1
#' @param group parameter same as "group" in the lavaan sem() function; whether to fit a multigroup model
#' @param cluster parameter same as "cluster" in the lavaan sem() function; whether to fit a cluster model
#' @param constraints parameter same as "constraints" in the lavaan sem() function; whether to apply
constraints to the model

```

```

#' @param WLS.V parameter same as "WLS.V" in the lavaan sem() function; whether to use WLS.V estimator
#' @param NACOV parameter same as "NACOV" in the lavaan sem() function; whether to use NACOV estimator
#' @param latent.dim number of network latent dimensions to use
#' @param ... optional arguments for the sem() function
#' @return the updated model specification with the network statistics as variables and a lavaan object which is
the SEM results, also the data generated
#' @export
#' @examples
#' \dontrun{
#' \donttest{
#' set.seed(10)
#' nsamp = 20
#' lv1 <- rnorm(nsamp)
#' net <- ifelse(matrix(rnorm(nsamp^2) , nsamp, nsamp) > 1, 1, 0)
#' lv2 <- rnorm(nsamp)
#' nonnet <- data.frame(x1 = lv1*0.5 + rnorm(nsamp),
#'                      x2 = lv1*0.8 + rnorm(nsamp),
#'                      x3 = lv2*0.5 + rnorm(nsamp),
#'                      x4 = lv2*0.8 + rnorm(nsamp))
#'
#' model <- '
#'   lv1 =~ x1 + x2
#'   lv2 =~ x3 + x4
#'   net ~ lv1
#'   lv2 ~ net
#' '
#' data = list(network = list(net = net), nonnetwork = nonnet)
#' set.seed(100)
#' res <- sem.net.edge.lsm(model = model, data = data, latent.dim = 1)
#' summary(res)
#' }
sem.net.edge.lsm <- function(model=NULL, data=NULL, type="difference",
                             latent.dim = 2, data.rescale = FALSE,
                             ordered = NULL, sampling.weights = NULL,
                             group = NULL, cluster = NULL, netstats.rescale = FALSE,
                             constraints = "", WLS.V = NULL, NACOV = NULL,
                             ...){
  ## checking proper input
  if(is.null(model)){
    stop("required argument model is not specified.")
  }

```

```

}
if(is.null(data)){
  stop("required argument data is not specified.")
}

```

```

params <- c(as.list(environment()), list(...))

```

```

## get the variable names in the model
model.info <- lavParseModelString(model)
model.var <- unique(c(model.info$lhs, model.info$rhs))

```

```

## non-network data variable names
data.nonnetwork.var <- names(data$nonnetwork)

```

```

## network data variable names
if (!is.null(data$network)){
  data.network.var <- names(data$network)
}

```

```

## find the network variables in the model
model.network.var <- data.network.var[data.network.var %in% model.var]
latent.network <- model.network.var

```

```

data_edge = data.frame(row_actor=rep(NA, nrow(data$nonnetwork)^2), col_actor=rep(NA,
nrow(data$nonnetwork)^2))

```

```

for (i in 1:length(model.network.var)){
  data_edge[model.network.var[i]]=NA
}
if (length(model.network.var)>0){
  for (i in 1:nrow(data$nonnetwork)){
    for (j in 1:nrow(data$nonnetwork)){
      data_edge[j+(i-1)*nrow(data$nonnetwork), "row_actor"]=i
      data_edge[j+(i-1)*nrow(data$nonnetwork), "col_actor"]=j
    }
  }
}

```

```

for (netind in 1:length(model.network.var)){
  data_edge[j+(i-1)*nrow(data$nonnetwork),model.network.var[netind]]=data$network[[netind]][i,j]
}
}
}
}

```

```

latent.vars <- list()
lsm.fits <- list()
fit.prev <- NULL
cov.mani <- list()
edgeatt <- list()

```

```

model.lavaanify <- lavaan::lavaanify(model)

```

```

## get the use specified model information
model.user <- model.lavaanify[model.lavaanify$user==1, ]

```

```

## change nonnetwork variable to be pairwise
variables.to.change=c()
for (i in 1:nrow(model.user)){
  ## check if the variable on the lhs is a nonnetwork variable
  if (model.user$lhs[i] %in% colnames(data$nonnetwork) && !model.user$lhs[i] %in% model.network.var){
    variables.to.change <- c(variables.to.change, model.user$lhs[i])
  }
  if (model.user$rhs[i] %in% colnames(data$nonnetwork) && !model.user$rhs[i] %in% model.network.var){
    variables.to.change <- c(variables.to.change, model.user$rhs[i])
  }
}

```

```

for (i in 1:length(variables.to.change)){
  data_edge[variables.to.change[i]]=NA
}

```

```

if (length(variables.to.change)>0){
  for (vind in 1:length(variables.to.change)){

```

```

v_row <- rep(data$nonnetwork[variables.to.change[vind]][[1]], each = nrow(data$nonnetwork))
v_col <- rep(data$nonnetwork[variables.to.change[vind]][[1]], nrow(data$nonnetwork))
if (type=="difference"){
  data_edge[variables.to.change[vind]] <- v_row - v_col
}else if (type=="average"){
  data_edge[variables.to.change[vind]] <- (v_row + v_col)/2
}

}

}

## estimate network latent positions
lsm.fits <- list()
for (i in 1:length(latent.network)){
  fit <- latentnet::ergmm(network::network(data$network[[latent.network[i]]]) ~ euclidean(d = latent.dim))
  lsm.fits[[i]] <- fit
  latent.vars[[latent.network[i]]] <- c()
  for (dimind in 1:latent.dim){
    distsum <- 0
    for (dimind in 1:latent.dim){
      distsum = distsum + outer(fit$mcmc.mle$Z[,dimind], fit$mcmc.mle$Z[,dimind], "-")^2
    }
    dists <- array(t(sqrt(distsum)))

    data_edge[paste0(model.network.var[i], ".dists")] <- dists
    if (netstats.rescale){
      data_edge[paste0(model.network.var[i], ".dists")] <- scale(dists, center = TRUE, scale = TRUE)
    }
    latent.vars[[model.network.var[i]]] <- c(paste0(model.network.var[i], ".dists"))
  }
}

# print(lsm.fits)

#print(model.network.stat.var.list)
## reconstruct the path model with the network variables
## replace the network variable name with the network variable stats name

```

```

## lavaanify the model
model.lavaanify <- lavaan::lavaanify(model)

## get the use specified model information
model.user <- model.lavaanify[model.lavaanify$user==1, ]

## now process each part of the user specified model
model.to.remove.index <- NULL
model.to.add <- ""
model.to.remove.index <- NULL
model.to.add <- ""
for (i in 1:nrow(model.user)){
  ## check if left is network with LSM, remake
  if (model.user$lhs[i] %in% latent.network){
    model.to.remove.index <- c(model.to.remove.index, i)
    model.stat.var.to.add <- latent.vars[[model.user$lhs[i]]]
    for (j in 1:length(model.stat.var.to.add)){
      model.temp <- paste0("\n ", model.stat.var.to.add[j], model.user$op[i], model.user$rhs[i])
      model.to.add <- paste0(model.to.add, model.temp)
    }
  }
  ## check if right is network with LSM and left is other variables
  if (model.user$rhs[i] %in% latent.network){
    model.to.remove.index <- c(model.to.remove.index, i)
    model.stat.var.to.add <- latent.vars[[model.user$rhs[i]]]
    for (j in 1:length(model.stat.var.to.add)){
      model.temp <- paste0("\n ", model.user$lhs[i], model.user$op[i], model.stat.var.to.add[j])
      model.to.add <- paste0(model.to.add, model.temp)
    }
  }
}

model.remove.network.var <- model.user[-model.to.remove.index, ]
model.non.network.var <- ""
if (nrow(model.remove.network.var)>0){
  for (i in 1:nrow(model.remove.network.var)){
    model.non.network.var.temp <- paste0(paste0(model.remove.network.var[i, c('lhs', 'op', 'rhs')], collapse = '
'))
  }
}

```

```
    model.non.network.var <- paste0(model.non.network.var.temp, "\n", model.non.network.var)
  }
}
```

```
model.full <- paste0(model.non.network.var, "\n", model.to.add)
```

```
lavparams <- list()
for (i in 1:length(params)){
  if (names(params)[i] %in% names(lavOptions())){
    lavparams[[names(params)[i]]] <- params[[i]]
  }
}
```

```
if (data.rescale){
  for (i in 1:ncol(data_edge)){
    if (is.numeric(data_edge[,i])){
      data_edge[,i] <- scale(data_edge[,i], center = TRUE, scale = TRUE)
    }
  }
}
```

```
lavparams[["data"]] <- data_edge
lavparams[["model"]] <- model.full
lavparams[["ordered"]] <- ordered
lavparams[["sampling.weights"]] <- sampling.weights
lavparams[["group"]] <- group
lavparams[["cluster"]] <- cluster
lavparams[["constraints"]] <- constraints
lavparams[["WLS.V"]] <- WLS.V
lavparams[["NACOV"]] <- NACOV
```

```
model.res <- do.call(what="sem", args=c(lavparams))
```

```

obj <- list(model=model.full, estimates=list(sem.es=model.res,lsm.es=lsm.fits), data = data_edge)
class(obj) <- "networksem"
return(obj)
}

```

## summary.networksem

This method function works for the networksem object to print nice information from the analysis.

```

#' Summarize output from networksem
#' @param object a networksem output
#' @param ... optional arguments
#' @return a summary of the output
#' @export
summary.networksem <- function(object,...){

  res = object
  otype = "obs"
  if (class(res$estimates)=="list"){
    otype = "lsm"
  }

  #  cat("The reconstructed model:\n")
  #  cat(res$model)
  #  cat("\n\n")

  if (otype == "obs"){
    lvsummary <- getMethod("summary",signature(object="lavaan"))
    cat("The SEM output:\n")
    print(lvsummary(res$estimates, fit = T))
  }else{
    lvsummary <- getMethod("summary",signature(object="lavaan"))

    # just fit info
    semsum <- lvsummary(res$estimates$sem.es)
    teststats <- semsum$test$standard$stat
    df <- semsum$test$standard$df
    pval <- semsum$test$standard$pval
    cat("Model Fit Information")
    cat("SEM Test statistics: ", teststats, "on", df, "df with p-value: ", pval, "\n")
  }
}

```



```

for (i in 1:length(res$estimates$lsn.es)){
  lsmsum <- summary(res$estimates$lsn.es[[i]])
  bic <- lsmsum$bic$overall
  cat("network", i, "LSM BIC: ", bic, "\n")
}
cat("=====\n")
cat("=====\n\n")

# full output
cat("The SEM output:\n")
print(lvsuamary(res$estimates$sem.es, fit = T))
cat("The LSM output:\n")
for (lsnout in res$estimates$lsn.es){
  print(summary(lsnout))
}
}
}
.S3method("summary", "networksem", "summary.networksem")

```

## path.networksem

This function can be used to calculate a mediation effect.

```

#' Calculate a mediation effect from a networksem model
#' @param res a networksem output file
#' @param predictor a character string of the predictor variable
#' @param mediator a character string of the mediator variable
#' @param outcome a character string of the outcome variable
#' @return a target path, associated estimates, and z-score
#' @export
path.networksem <- function(res, predictor, mediator, outcome){
  # only allow 1 predictor and 1 outcome
  if (length(predictor) > 1 | length(outcome) > 1){
    stop("Only 1 predictor and 1 outcome are allowed")
  }
}

```

```

otype = "obs"
if (class(res$estimates)=="list"){
  otype = "lsm"
}

if (otype == "obs"){
  pars <- parameterEstimates(res$estimates)
  covm <- vcov(res$estimates)
}else{
  pars <- parameterEstimates(res$estimates$sem.es)
  covm <- vcov(res$estimates$sem.es)
}

effect_table <- expand.grid("predictor" = predictor,
                           "mediator" = mediator,
                           "outcome" = outcome)
effect_table$apath = NA; effect_table$bpath = NA; effect_table$indirect = NA;
effect_table$indirect_se = NA; effect_table$indirect_z = NA;

for (i in 1:nrow(effect_table)){

  # estimates of a and b paths
  effect_table$apath[i] <- pars[pars$rhs == effect_table$predictor[i] & pars$lhs == effect_table$mediator[i],
"est"]
  effect_table$bpath[i] <- pars[pars$rhs == effect_table$mediator[i] & pars$lhs == effect_table$outcome[i],
"est"]
  effect_table$indirect[i] <- effect_table$apath[i]*effect_table$bpath[i]

  # indirect effect se usinh sobel
  headera = paste0(effect_table$mediator[i], "~", effect_table$predictor[i])
  headerb = paste0(effect_table$outcome[i], "~", effect_table$mediator[i])
  abcov = covm[c(headera, headerb), c(headera, headerb)]
  a <- effect_table$apath[i]
  b <- effect_table$bpath[i]
  ab_vector <- c(a, b)
  var_ab <- t(ab_vector) %*% abcov %*% ab_vector
  effect_table$indirect_se[i] <- sqrt(var_ab)

```

```

    effect_table$indirect_z[i] <- effect_table$indirect[i]/effect_table$indirect_se[i]

  }

  return(effect_table)
}

```

## sem.net.addvar.stat

```

#' Compute a list of user-specified network statistics values using the "sna" package and add them to the non-
network data.
#' @param data a list containing both the non-network and network data
#' @param model.network.stat.var.list a list of elements with names corresponding to the network names and
values corresponding to lists of network statistics that will be calculated for the corresponding network
#' @param model.network.var.i an index indicating a specific network within all networks
#' @param stats a network statistics that can be calculated using package "sna"
#' @param statsname name of the network statistics
#' @param netstats.rescale a logical value indicating whether to rescale network statistics to have mean 0 and
sd 1
#' @param netstats.options a list with names being the argument names for calculating the network statistics,
and values being the argument values
#' @return a list with the first value being the list of network statistics names and the second value being the
data frame with added network statistics
sem.net.addvar.stat <- function(model.network.stat.var.list, data, model.network.var.i, stats, statsname,
netstats.rescale, netstats.options=NULL){
  degree <- sna::degree
  betweenness <- sna::betweenness
  closeness <- sna::closeness
  ## create the network stats variable name
  model.network.stat.var <- paste0(model.network.var.i, ".", statsname)

  ## add network statistics to the variable list
  model.network.stat.var.list[[model.network.var.i]] <- c(model.network.stat.var.list[[model.network.var.i]],
model.network.stat.var)

  ## using do.call to calculate the network statistics in variable list, add statistics to nonnetwork data
  args <- list("dat"=data$network[[model.network.var.i]])

```

```

args <- c(args, netstats.options)
data$nonnetwork[[model.network.stat.var]] <- do.call(what=stats, args=args)

# scale
if(netstats.rescale){
  data$nonnetwork[model.network.stat.var] <- scale(data$nonnetwork[model.network.stat.var])
}

return(list(model.network.stat.var.list, data$nonnetwork))
}

```

## sem.net.addvar.influential

```

#' Compute a list of user-specified network statistics using the "influential" package and add it to the existing
data.
#' @param model.network.stat.var.list a list of elements with names corresponding to the network names and
values corresponding to lists of network statistics that will be calculated for the corresponding network
#' @param data a list containing both the non-network and network data
#' @param model.network.var.i an index indicating a specific network within all networks
#' @param stats a network statistics that can be calculated using package "influential"
#' @param statsname name of the network statistics
#' @param netstats.rescale a logical value indicating whether to rescale network statistics to have mean 0 and
sd 1
#' @param netstats.options a list with names being the argument names for calculating the network statistics,
and values being the argument values
#' @return a list with the first value being the list of network statistics names and the second value being the
data frame with added network statistics
sem.net.addvar.influential <- function(model.network.stat.var.list, data, model.network.var.i, stats, statsname,
netstats.rescale, netstats.options=NULL){
  ## create the network stats variable name
  model.network.stat.var <- paste0(model.network.var.i, ".", statsname)

  ## add network statistics to the variable list
  model.network.stat.var.list[[model.network.var.i]] <- c(model.network.stat.var.list[[model.network.var.i]],
model.network.stat.var)

```

```

## using do.call to calculate the network statistics in variable list, add statistics to nonnetwork data
args <- list("graph"=graph_from_adjacency_matrix(data$network[[model.network.var.i]]))
args <- c(args, netstats.options)
data$nonnetwork[[model.network.stat.var]] <- do.call(what=stats, args=args)

# scale
if(netstats.rescale){
  data$nonnetwork[model.network.stat.var] <- scale(data$nonnetwork[model.network.stat.var])
}

return(list(model.network.stat.var.list, data$nonnetwork))
}

```

## sem.net.addvar

```

#' Compute user-specified network statistics for a specific network.
#' @param model.network.stat.var.list a list of elements with names corresponding to the network names and
values corresponding to lists of network statistics that will be calculated for the corresponding network
#' @param data a list containing both the non-network and network data
#' @param netstats a list of user-specified network statistics
#' @param model.network.var.i the index of a network within all networks
#' @param netstats.rescale a logical value indicating whether to rescale network statistics to have mean 0 and
sd 1
#' @param netstats.options a list with element names corresponding to the network statistics and element
values corresponding to another list. The list corresponding to each network statistics has element names being
the argument names for calculating the network statistics, and values being the argument values
#' @return a list with the first value being the list of network statistics names and the second value being the
data frame with added network statistics variables
sem.net.addvar <- function(model.network.stat.var.list=NULL, data=NULL, netstats=NULL,
model.network.var.i=NULL, netstats.rescale=TRUE, netstats.options=NULL){
  res.list<-list()
  for (stat in netstats){

    if( stat %in% c("degree", "closeness", "betweenness", "evcent", "stresscent", "infocent")){
      # sna
      res.list<-sem.net.addvar.stat(model.network.stat.var.list, data, model.network.var.i, stats=stat,
statsname=stat, netstats.rescale, netstats.options[[stat]])
    }else{

```

```
# influential
res.list<-sem.net.addvar.influential(model.network.stat.var.list, data, model.network.var.i, stats=stat,
statsname=stat, netstats.rescale, netstats.options[[stat]])
}
model.network.stat.var.list <- res.list[[1]]
data$nonnetwork <- res.list[[2]]
}
return(res.list)
}
```

---

Revision #16

Created 27 October 2024 21:38:46 by Admin

Updated 30 December 2024 22:13:37 by Admin