appendix

Feng Ji

June 19, 2018

#functions for validation  
  
#' Automatically calculate the accuracy of one condition.  
#' @export  
#' @param Dep The names of variables you would like to condition on.  
#' @param Ind The names of variables of your interest.  
#' @param d Dataset created by using simStandarized.  
#' @return Correlation between the conditional Mahalanobis distance calculated by using the true scores and the conditional Mahalanobis calculated by using estimated factor scores  
#' @examples  
#' SimModel <- "  
#' Gc =~ 0.85 \* Gc1 + 0.68 \* Gc2 + 0.8 \* Gc3  
#' Gf =~ 0.8 \* Gf1 + 0.9 \* Gf2 + 0.8 \* Gf3  
#' Gs =~ 0.7 \* Gs1 + 0.8 \* Gs2 + 0.8 \* Gs3  
#' Read =~ 0.66 \* Read1 + 0.85 \* Read2 + 0.91 \* Read3  
#' Math =~ 0.4 \* Math1 + 0.9 \* Math2 + 0.7 \* Math3  
#' Gc ~ 0.6 \* Gf + 0.1 \* Gs  
#' Gf ~ 0.5 \* Gs  
#' Read ~ 0.4 \* Gc + 0.1 \* Gf  
#' Math ~ 0.2 \* Gc + 0.3 \* Gf + 0.1 \* Gs"  
#' d\_demo <- simStandardized(SimModel, 10)  
#' CM\_cor(Dep = c("Math", "Read"),Ind = c("Gc", "Gf", "Gs") , d\_demo)  
  
CM\_cor <- function(Dep, Ind = NULL, d){  
  
 # extract simulated data  
 R <- d[["R"]]  
 cor\_data <- d[["Data"]]  
  
 # get the true CMahalanobis  
 TrueCM <- CMahalanobis(Dep = Dep, Ind = Ind, R = R, d = cor\_data)  
  
 # get the CMahalanobis of FS  
 EstCM <- CMahalanobis\_FS(Dep = Dep, Ind = Ind, R = R, d = cor\_data)  
 TrueCM <- TrueCM[["dCM"]]  
 EstCM <- EstCM[["dCM"]]  
  
 # calculate the reliability  
 cor <- cor(TrueCM, EstCM)  
 cor  
}  
  
#' Function that removes all fixed values in a lavaan model  
#' @export  
#' @param m Population relations among variables represented by Lavaan Syntax.  
#' @return Lavaan object without specified coefficients  
#' @examples  
#' SimModel <- "  
#' Gc =~ 0.85 \* Gc1 + 0.68 \* Gc2 + 0.8 \* Gc3  
#' Gf =~ 0.8 \* Gf1 + 0.9 \* Gf2 + 0.8 \* Gf3  
#' Gs =~ 0.7 \* Gs1 + 0.8 \* Gs2 + 0.8 \* Gs3  
#' Read =~ 0.66 \* Read1 + 0.85 \* Read2 + 0.91 \* Read3  
#' Math =~ 0.4 \* Math1 + 0.9 \* Math2 + 0.7 \* Math3  
#' Gc ~ 0.6 \* Gf + 0.1 \* Gs  
#' Gf ~ 0.5 \* Gs  
#' Read ~ 0.4 \* Gc + 0.1 \* Gf  
#' Math ~ 0.2 \* Gc + 0.3 \* Gf + 0.1 \* Gs"  
#' Sim2Free(SimModel)  
  
Sim2Free <- function(m){  
 m %>%  
 lavaanify(fixed.x = F) %>%  
 filter(.data$lhs != .data$rhs) %>%  
 group\_by(.data$lhs, .data$op) %>%  
 summarise(rhs = paste(.data$rhs, collapse = " + ")) %>%  
 arrange(desc(.data$op)) %>%  
 tidyr::unite("l", .data$lhs, .data$op, .data$rhs, sep = " ") %>%  
 pull(.data$l) %>%  
 paste(collapse = "\n")  
}  
  
#' Function to make a lavaan formula into a model for simulation that takes a vector of parameters as input  
#' @export  
#' @param SimModel Population relations among variables represented by Lavaan Syntax.  
#' @return Lavaan object without specified coefficients  
#' @examples  
#' SimModel <- "  
#' Gc =~ 0.85 \* Gc1 + 0.68 \* Gc2 + 0.8 \* Gc3  
#' Gf =~ 0.8 \* Gf1 + 0.9 \* Gf2 + 0.8 \* Gf3  
#' Gs =~ 0.7 \* Gs1 + 0.8 \* Gs2 + 0.8 \* Gs3  
#' Read =~ 0.66 \* Read1 + 0.85 \* Read2 + 0.91 \* Read3  
#' Math =~ 0.4 \* Math1 + 0.9 \* Math2 + 0.7 \* Math3  
#' Gc ~ 0.6 \* Gf + 0.1 \* Gs  
#' Gf ~ 0.5 \* Gs  
#' Read ~ 0.4 \* Gc + 0.1 \* Gf  
#' Math ~ 0.2 \* Gc + 0.3 \* Gf + 0.1 \* Gs"  
#' sim2glue(SimModel)  
  
sim2glue <- function(SimModel){  
 SimModel %>%  
 lavaanify(fixed.x = F) %>%  
 filter(.data$lhs != .data$rhs) %>%  
 rowid\_to\_column(var = "ID") %>%  
 mutate(ustart = paste0("{para",  
 "[",  
 sprintf( "%02d", .data$ID ),  
 "]",  
 "} \* ")) %>%  
 group\_by(.data$op, .data$lhs) %>%  
 summarise(rhs = paste(.data$ustart, .data$rhs, collapse = " + ")) %>%  
 arrange(desc(.data$op)) %>%  
 arrange(.data$rhs) %>%  
 unite("l", .data$lhs, .data$op, .data$rhs, sep = " ") %>%  
 pull(.data$l) %>%  
 paste(collapse = "\n")  
}  
  
#' Function to stuck parameters to the lavaan model  
#' @export  
#' @param SimModel Population relations among variables represented by Lavaan Syntax.  
#' @param para Parameters for simulation studies.  
#' @return A list of lavaan object with specified parameters.  
  
glue2simf <- function(SimModel, para){  
 glue(  
 sim2glue(SimModel)  
 )  
}  
  
  
#' Create a tibble to hold everything for simulation  
#' @export  
#' @param SimModel Population relations among variables represented by Lavaan Syntax  
#' @param dPar Parameters we would like to test in simulation studies  
#' @param n The number of cases  
#' @return all parameters, a lavaan object with specified parameters and data generated for each condition by calling simStandarized  
#' @examples  
#' SimModel <- "  
#' Gc =~ 0.85 \* Gc1 + 0.68 \* Gc2 + 0.8 \* Gc3  
#' Gf =~ 0.8 \* Gf1 + 0.9 \* Gf2 + 0.8 \* Gf3  
#' Gs =~ 0.7 \* Gs1 + 0.8 \* Gs2 + 0.8 \* Gs3  
#' Read =~ 0.66 \* Read1 + 0.85 \* Read2 + 0.91 \* Read3  
#' Math =~ 0.4 \* Math1 + 0.9 \* Math2 + 0.7 \* Math3  
#' Gc ~ 0.6 \* Gf + 0.1 \* Gs  
#' Gf ~ 0.5 \* Gs  
#' Read ~ 0.4 \* Gc + 0.1 \* Gf  
#' Math ~ 0.2 \* Gc + 0.3 \* Gf + 0.1 \* Gs"  
#' beta\_measure1 <- makePara(1000, 9000, n = 10, k = 15)  
#' beta\_structure1 <- makePara(1000, 1000, n = 10, k = 8)  
#' cbind(beta\_measure1, beta\_structure1)-> par\_beta1  
#' colnames(par\_beta1) <- paste0("x", 1:23)  
#' simMaha(par\_beta1, SimModel, n = 10)  
  
simMaha <- function(dPar, SimModel, n = 100){  
 by\_row(dPar, glue2simf, SimModel = SimModel) %>%  
 mutate(data = map(.data$.out,  
 simStandardized,  
 n = n))  
}  
  
#' Simulate data and cor between true conditional Mahalanobis distance and the estimated  
#' @export  
#' @param SimModel Population relations among variables represented by Lavaan Syntax  
#' @param dPar Parameters we would like to test in simulation studies  
#' @param Dep The names of variables you would like to condition on  
#' @param Ind The names of variables of your interest  
#' @param n The number of cases  
#' @return all parameters, lavaan object with specified parameters and data generated for each condition by calling simStandarized and the accuracy index for each condition  
#' @examples  
#' SimModel <- "  
#' Gc =~ 0.85 \* Gc1 + 0.68 \* Gc2 + 0.8 \* Gc3  
#' Gf =~ 0.8 \* Gf1 + 0.9 \* Gf2 + 0.8 \* Gf3  
#' Gs =~ 0.7 \* Gs1 + 0.8 \* Gs2 + 0.8 \* Gs3  
#' Read =~ 0.66 \* Read1 + 0.85 \* Read2 + 0.91 \* Read3  
#' Math =~ 0.4 \* Math1 + 0.9 \* Math2 + 0.7 \* Math3  
#' Gc ~ 0.6 \* Gf + 0.1 \* Gs  
#' Gf ~ 0.5 \* Gs  
#' Read ~ 0.4 \* Gc + 0.1 \* Gf  
#' Math ~ 0.2 \* Gc + 0.3 \* Gf + 0.1 \* Gs"  
#' beta\_measure1 <- makePara(1000, 9000, n = 10, k = 15)  
#' beta\_structure1 <- makePara(1000, 1000, n = 10, k = 8)  
#' par\_beta1 <- cbind(beta\_measure1, beta\_structure1)  
#' colnames(par\_beta1) <- paste0("x", 1:23)  
#' simCor(dPar = par\_beta1, SimModel, Dep = c("Read", "Math"),Ind = c("Gc", "Gs", "Gf"), n = 10)  
  
simCor <- function(dPar, SimModel, Dep, Ind = NULL, n = 100){  
 dPar %>%  
 simMaha(SimModel, n) %>%  
 mutate(cor = map\_dbl(.data$data,  
 CM\_cor,  
 Dep = Dep,  
 Ind = Ind))  
}  
  
#' function to make a set of parameters that follows beta distribution  
#' @export  
#' @param from starting point for the parameters  
#' @param to ending point for the parameters  
#' @param n number of cases  
#' @param k number of variables.  
#' @return all parameters, lavaan object with specified parameters and data generated for each condition by calling simStandarized and the accuracy index for each condition  
#' @examples  
#' makePara(1000, 9000, n = 10, k = 15)  
  
makePara <- function(from, to, n, k){  
 list1 <- map(seq(from = from,  
 to = to,  
 length.out = n),  
 function(p) rbeta(k, p, 10000 - p))  
 data.frame(t(sapply(list1, c)))  
}  
  
#' confidence interval of the reliability (accuracy index)  
#' @export  
#' @param SimModel Population relations among variables represented by Lavaan Syntax  
#' @param Dep The names of variables you would like to condition on  
#' @param Ind The names of variables of your interest  
#' @param size The number of trials  
#' @param n The number of cases  
#' @return simulated 95% confidence interval  
#' @examples  
#' SimModel <- "  
#' Gc =~ 0.85 \* Gc1 + 0.68 \* Gc2 + 0.8 \* Gc3  
#' Gf =~ 0.8 \* Gf1 + 0.9 \* Gf2 + 0.8 \* Gf3  
#' Gs =~ 0.7 \* Gs1 + 0.8 \* Gs2 + 0.8 \* Gs3  
#' Read =~ 0.66 \* Read1 + 0.85 \* Read2 + 0.91 \* Read3  
#' Math =~ 0.4 \* Math1 + 0.9 \* Math2 + 0.7 \* Math3  
#' Gc ~ 0.6 \* Gf + 0.1 \* Gs  
#' Gf ~ 0.5 \* Gs  
#' Read ~ 0.4 \* Gc + 0.1 \* Gf  
#' Math ~ 0.2 \* Gc + 0.3 \* Gf + 0.1 \* Gs"  
#' boot(SimModel, Dep = c("Math", "Read"),Ind = c("Gc", "Gf", "Gs"), size = 100, n = 100)  
  
boot <- function(SimModel, Dep, Ind = NULL, size = 100, n = 100){  
 con\_cor <- function(SimModel, Dep, Ind, size){  
 d <- simStandardized(SimModel, size)  
 CM\_cor(Dep, Ind, d)  
 }  
 out <- replicate(n, con\_cor(SimModel, Dep, Ind, size))  
 stats::quantile(out, probs = c(0.025, 0.975))  
}  
  
  
#' Extract names from a lavaan syntax object.  
#' @export  
#' @param m Lavaan Syntax Object.  
#' @return Names of observed variables, latent variables.  
#' @examples  
#' m = "LatantVariable =~ ObservedVar1 + ObservedVar2 + Observed3"  
#' mahaName(m)  
  
  
mahaName <- function(m){  
 # Parameter Table  
 pt <- lavParTable(m, fixed.x = F)  
  
 # Variable Names  
 vObserved <- lavNames(pt, "ov")  
 vLatent <- lavNames(pt, "lv")  
 vLatentExogenous <- lavNames(pt, "lv.x")  
 vLatentEndogenous <- lavNames(pt, "lv.nox")  
 vObservedExogenous <- lavNames(pt, "ov.x")  
 vObservedEndogenous <- lavNames(pt, "ov.nox")  
 if (length(vLatentEndogenous) > 0) {  
 vDisturbance <- paste0("d\_", vLatentEndogenous)  
 } else {  
 vDisturbance <- character(0)  
 }  
  
 if (length(vObservedEndogenous) > 0) {  
 vError <- paste0("e\_", vObservedEndogenous)  
 } else {  
 vError <- character(0)  
 }  
  
 # Names for A and S matrices  
 vA <- c(vLatentExogenous,  
 vLatentEndogenous,  
 vObservedExogenous,  
 vObservedEndogenous)  
 vS <- c(vLatentExogenous,  
 vDisturbance,  
 vObservedExogenous,  
 vError)  
 vnewS <- c(vLatentExogenous,  
 vLatentEndogenous,  
 vObservedExogenous,  
 vObservedEndogenous,  
 vDisturbance,  
 vError)  
 mahanamelist <- list(vA = vA,  
 vS = vS,  
 vObserved = vObserved,  
 vLatent = vLatent,  
 vLatentExogenous = vLatentExogenous,  
 vLatentEndogenous = vLatentEndogenous,  
 vObservedExogenous = vObservedExogenous,  
 vObservedEndogenous = vObservedEndogenous,  
 vError = vError)  
 mahanamelist  
  
}  
  
#' Estimate factor scores for a given profile and population model.  
#' @export  
#' @param m Lavaan Syntax Object.  
#' @param d Observed Z scores.  
#' @return Standardized estimated factor scores.  
#' @examples  
#' m = "LatentVariable =~ 0.8 \* ObservedVar1 + 0.8 \* ObservedVar2 + 0.8 \* Observed3"  
#' d <- simStandardized(m, 1000) #simulate 1000 cases (profiles)  
#'  
#' # Pick up the first three cases for illustration  
#' demo\_individual\_exp <- d$Data[1:3,]  
#'  
#' # Estimate factor scores based on the three cases as well as the population model in m  
#' estStandardized(m, demo\_individual\_exp)  
estStandardized <- function(m, d){  
 # Parameter Table  
 pt <- lavParTable(m, fixed.x = F)  
 # Variable Names  
 vObserved <- lavNames(pt, "ov")  
 vLatent <- lavNames(pt, "lv")  
 vLatentExogenous <- lavNames(pt, "lv.x")  
 vLatentEndogenous <- lavNames(pt, "lv.nox")  
 vObservedExogenous <- lavNames(pt, "ov.x")  
 vObservedEndogenous <- lavNames(pt, "ov.nox")  
 if (length(vLatentEndogenous) > 0) {  
 vDisturbance <- paste0("d\_", vLatentEndogenous)  
 } else {  
 vDisturbance <- character(0)  
 }  
  
 if (length(vObservedEndogenous) > 0) {  
 vError <- paste0("e\_", vObservedEndogenous)  
 } else {  
 vError <- character(0)  
 }  
  
 # Names for A, S and newS matrices  
 vA <- c(vLatentExogenous,  
 vLatentEndogenous,  
 vObservedExogenous,  
 vObservedEndogenous)  
 vS <- c(vLatentExogenous,  
 vDisturbance,  
 vObservedExogenous,  
 vError)  
 vnewS <- c(vLatentExogenous,  
 vLatentEndogenous,  
 vObservedExogenous,  
 vObservedEndogenous,  
 vDisturbance,  
 vError)  
 # Number of Variables  
 k <- length(vA)  
  
 # Initialize A matrix and exogenous correlation matrix  
 ExoCor <- A <- matrix(0, k, k, dimnames = list(vA, vA))  
  
 # Assign loadings to A  
 for (i in pt[pt[, "op"] == "=~", "id"] ) {  
 A[pt$rhs[i], pt$lhs[i]] <- pt$ustart[i]  
 }  
  
 # Assign regressions to A  
 for (i in pt[pt[, "op"] == "~", "id"]) {  
 A[pt$lhs[i], pt$rhs[i]] <- pt$ustart[i]  
 }  
  
 # Assign correlations to ExoCor  
 diag(ExoCor) <- 1  
 for (i in pt[pt[, "op"] == "~~", "id"]) {  
 if (pt$lhs[i] != pt$rhs[i]) {  
 ExoCor[pt$lhs[i], pt$rhs[i]] <- ifelse(is.na(pt$ustart[i]),  
 0,  
 pt$ustart[i])  
 ExoCor[pt$rhs[i], pt$lhs[i]] <- ExoCor[pt$lhs[i], pt$rhs[i]]  
 }  
  
 }  
  
 #Solving for error variances and correlation matrix  
  
 #Column of k ones  
 v1 <- matrix(1, k)  
  
 #Initial estimate of error variances  
 varS <- as.vector(v1 - (A \* A) %\*% v1)  
 S <- diag(varS) %\*% ExoCor %\*% diag(varS)  
  
 #Initial estimate of correlation matrix  
 R <- solve(diag(k) - A) %\*% S %\*% t(solve(diag(k) - A))  
  
 # Set interaction count at 0  
 iterations <- 0  
  
 # Find values for S matrix  
 while ((round(sum(diag(R)), 10) != k) \* (iterations < 100)) {  
 iA <- solve(diag(k) - A)  
 R <- iA %\*% S %\*% t(iA)  
 sdS <- diag(diag(S) ^ 0.5)  
 S <- diag(diag(diag(k) - R)) + (sdS %\*% ExoCor %\*% sdS)  
 iterations <- iterations + 1  
 }  
 if (iterations == 100) {  
 warning(paste("Maximum iterations reached (100).",  
 "Results might not be trustworthy."))  
 }  
  
 # Assign variable names to S  
 dimnames(S) <- list(vS, vS)  
  
 # Created extended A matrix  
 extendA <- diag(diag(S) ^ 0.5)  
 dimnames(extendA) <- list(c(vLatentExogenous,  
 vLatentEndogenous,  
 vObservedExogenous,  
 vObservedEndogenous),  
 c(vLatentExogenous,  
 vDisturbance,  
 vObservedExogenous,  
 vError))  
 # Remove exogenous variables  
 extendA <- extendA[, c(vDisturbance, vError)]  
  
 # bind A and extended A  
 extCol <- cbind(A, extendA)  
 # Append zeros so that new A will be square  
 extRow <- matrix(0,  
 nrow = ncol(extendA),  
 ncol = ncol(A) + ncol(extendA))  
 newA <- rbind(extCol, extRow)  
 dimnames(newA) <- list(c(colnames(A), colnames(extendA)),  
 c(colnames(A), colnames(extendA)))  
  
 # build a S matrix with 1s and 0s on the diag  
 newS <- diag(c(rep(1, length(vLatentExogenous)),  
 rep(0, length(vLatentEndogenous)),  
 rep(1, length(vObservedExogenous)),  
 rep(0, length(vObservedEndogenous)),  
 rep(1, length(vDisturbance)),  
 rep(1, length(vError)))  
 )  
  
 dimnames(newS) <- list(vnewS, vnewS)  
  
 # Insert all off-diagonal covariances  
 ExoCor <- newS[c(vLatentExogenous,  
 vObservedExogenous,  
 vDisturbance,  
 vError),  
 c(vLatentExogenous,  
 vObservedExogenous,  
 vDisturbance,  
 vError)]  
  
 R <- solve(diag(nrow(newA)) - newA) %\*%  
 newS %\*%  
 t(solve(diag(nrow(newA)) - newA))  
  
  
 Rxx <- R[vObserved, vObserved]  
 Rxy <- R[vObserved, c(vLatent, vDisturbance, vError)]  
 iRxx <- solve(Rxx)  
 l <- list(Data = d,  
 vObserved = vObserved,  
 vError = vError)  
 if (length(vLatent) > 0) {  
 FScoef <- iRxx %\*% Rxy  
 FactorScores <- as.matrix(d[, vObserved]) %\*% FScoef  
 colnames(FactorScores) <- paste0(c(vLatent,  
 vDisturbance,  
 vError),  
 "\_FS")  
 # add factor scores to the R matrix  
 FS\_name <- c(vLatent, vDisturbance, vError)  
 R\_right <- R[, FS\_name]  
 colnames(R\_right) <- paste0(FS\_name, "\_FS")  
 cbind(R, R\_right)  
 R\_down <- R[FS\_name, ]  
 rownames(R\_down) <- paste0(FS\_name, "\_FS")  
 R\_central <- R[FS\_name, FS\_name]  
 R\_down\_central <- cbind(R\_down, R\_central)  
 R\_all <- rbind(cbind(R, R\_right), R\_down\_central)  
  
 FSValidity <- diag(t(FScoef) %\*%  
 R[vObserved,  
 c(vLatent,  
 vDisturbance,  
 vError)])  
 FSStandardError <- sqrt(rep(1, length(c(vLatent,  
 vDisturbance,  
 vError))) - FSValidity)  
 names(FSStandardError) <- paste0("se.", names(FSStandardError))  
 l$Data <- cbind(d, FactorScores)  
 l$vLatent <- vLatent  
 l$vDisturbance <- vDisturbance  
 l$vFactorScores <- colnames(FactorScores)  
 l$FactorScoreCoef <- FScoef  
 l$R <- R  
 l$R\_all <- R\_all  
 l$R\_FS <- cov2cor(t(FScoef) %\*% Rxx %\*% FScoef)  
 l$FactorScoreValidity <- FSValidity  
 l$FactorScoreSE <- FSStandardError  
 }  
 l  
}  
  
#' Function that takes a lavaan model with standardized parameters and simulates latent scores, errors, disturbances, and observed scores  
#'  
#'@export  
#' @param m Lavaan Syntax Object.  
#' @param n Number of simulated cases.  
#' @param ObservedOnly Return only observed data  
#' @return Latent scores, errors, disturbances, and observed scores.  
#' @examples  
#' # Lavaan model  
#' m = "LatantVariable =~ 0.8 \* ObservedVar1 + 0.8 \* ObservedVar2 + 0.8 \* Observed3"  
#'  
#' # simulate 100 cases  
#' d <- simStandardized(m, n = 100)  
simStandardized <- function(m, n = 1000, ObservedOnly = FALSE){  
 # Parameter Table  
 pt <- lavParTable(m, fixed.x = F)  
  
 # Variable Names  
 vObserved <- lavNames(pt, "ov")  
 vLatent <- lavNames(pt, "lv")  
 vLatentExogenous <- lavNames(pt, "lv.x")  
 vLatentEndogenous <- lavNames(pt, "lv.nox")  
 vObservedExogenous <- lavNames(pt, "ov.x")  
 vObservedEndogenous <- lavNames(pt, "ov.nox")  
 if (length(vLatentEndogenous) > 0) {  
 vDisturbance <- paste0("d\_", vLatentEndogenous)  
 } else {  
 vDisturbance <- character(0)  
 }  
  
 if (length(vObservedEndogenous) > 0) {  
 vError <- paste0("e\_", vObservedEndogenous)  
 } else {  
 vError <- character(0)  
 }  
  
 # Names for A, S and new S matrices  
 vA <- c(vLatentExogenous,  
 vLatentEndogenous,  
 vObservedExogenous,  
 vObservedEndogenous)  
 vS <- c(vLatentExogenous,  
 vLatentEndogenous,  
 vObservedExogenous,  
 vObservedEndogenous)  
 vnewS <- c(vLatentExogenous,  
 vLatentEndogenous,  
 vObservedExogenous,  
 vObservedEndogenous,  
 vDisturbance,  
 vError)  
 # Number of Variables  
 k <- length(vA)  
  
 # Initialize A matrix and exogenous correlation matrix  
 ExoCor <- A <- matrix(0, k, k, dimnames = list(vA, vA))  
  
 # Assign loadings to A  
 for (i in pt[pt[, "op"] == "=~", "id"] ) {  
 A[pt$rhs[i], pt$lhs[i]] <- pt$ustart[i]  
 }  
  
 # Assign regressions to A  
 for (i in pt[pt[, "op"] == "~", "id"]) {  
 A[pt$lhs[i], pt$rhs[i]] <- pt$ustart[i]  
 }  
  
 # Assign correlations to ExoCor  
 diag(ExoCor) <- 1  
 for (i in pt[pt[, "op"] == "~~", "id"]) {  
 if (pt$lhs[i] != pt$rhs[i]) {  
 ExoCor[pt$lhs[i], pt$rhs[i]] <- ifelse(is.na(pt$ustart[i]),  
 0,  
 pt$ustart[i])  
 ExoCor[pt$rhs[i], pt$lhs[i]] <- ExoCor[pt$lhs[i], pt$rhs[i]]  
 }  
  
 }  
  
 #Solving for error variances and correlation matrix  
  
 #Column of k ones  
 v1 <- matrix(1, k)  
  
 #Initial estimate of error variances  
 varS <- as.vector(v1 - (A \* A) %\*% v1)  
 S <- diag(varS) %\*% ExoCor %\*% diag(varS)  
  
 #Initial estimate of the correlation matrix  
 R <- solve(diag(k) - A) %\*% S %\*% t(solve(diag(k) - A))  
  
 # Set interaction count at 0  
 iterations <- 0  
  
 # Find values for S matrix  
 while ((round(sum(diag(R)), 10) != k) \* (iterations < 100) ) {  
 iA <- solve(diag(k) - A)  
 R <- iA %\*% S %\*% t(iA)  
 sdS <- diag(diag(S) ^ 0.5)  
 S <- diag(diag(diag(k) - R)) + (sdS %\*% ExoCor %\*% sdS)  
 iterations <- iterations + 1  
 }  
 if (iterations == 100) {  
 warning(paste("Maximum iterations reached (100).",  
 "Results might not be trustworthy."))  
 }  
  
  
  
  
  
  
 # Assign variable names to S  
 dimnames(S) <- list(vS, vS)  
  
 # Generate data frame  
  
 # Exogenous data  
 u <- rmvnorm(n, sigma = S)  
 colnames(u) <- vS  
  
 v <- u %\*% t(iA)  
 #Simulated dataset  
 d <- as\_tibble(cbind(v,  
 u[, c(-1 \* match(vLatentExogenous, vS),  
 -1 \* match(vObservedExogenous, vS))]))  
 colnames(d) <- vnewS  
  
 dimnames(S) <- list(vS, vS)  
  
 # Created extended A matrix  
 extendA <- diag(diag(S) ^ 0.5)  
 dimnames(extendA) <- list(c(vLatentExogenous,  
 vLatentEndogenous,  
 vObservedExogenous,  
 vObservedEndogenous),  
 c(vLatentExogenous,  
 vDisturbance,  
 vObservedExogenous,  
 vError))  
 # Remove exogenous variables  
 extendA <- extendA[, c(vDisturbance, vError)]  
  
 # bind A and extended A  
 extCol <- cbind(A, extendA)  
 # Append zeros so that new A will be square  
 extRow <- matrix(0,  
 nrow = ncol(extendA),  
 ncol = ncol(A) + ncol(extendA))  
 newA <- rbind(extCol, extRow)  
 dimnames(newA) <- list(c(colnames(A), colnames(extendA)),  
 c(colnames(A), colnames(extendA)))  
  
 # build a S matrix with 1s and 0s on the diag  
 newS <- diag(c(rep(1, length(vLatentExogenous)),  
 rep(0, length(vLatentEndogenous)),  
 rep(1, length(vObservedExogenous)),  
 rep(0, length(vObservedEndogenous)),  
 rep(1, length(vDisturbance)),  
 rep(1, length(vError)))  
 )  
  
 dimnames(newS) <- list(vnewS, vnewS)  
  
 # Insert all off-diagonal covariances  
 ExoCor <- newS[c(vLatentExogenous,  
 vObservedExogenous,  
 vDisturbance,  
 vError),  
 c(vLatentExogenous,  
 vObservedExogenous,  
 vDisturbance,  
 vError)]  
  
 R <- solve(diag(nrow(newA)) - newA) %\*%  
 newS %\*%  
 t(solve(diag(nrow(newA)) - newA))  
  
 Rxx <- R[vObserved, vObserved]  
 Rxy <- R[vObserved, c(vLatent, vDisturbance, vError)]  
 iRxx <- solve(Rxx)  
 if (ObservedOnly) {  
 d[, vObserved]  
 } else {  
 l <- list(Data = d,  
 vObserved = vObserved,  
 vError = vError)  
 if (length(vLatent) > 0) {  
 FScoef <- iRxx %\*% Rxy  
 FactorScores <- as.matrix(d[, vObserved]) %\*% FScoef  
 colnames(FactorScores) <- paste0(c(vLatent,  
 vDisturbance,  
 vError),  
 "\_FS")  
 # Add factor scores to the R matrix  
 FS\_name <- c(vLatent, vDisturbance, vError)  
 R\_right <- R[, FS\_name]  
 colnames(R\_right) <- paste0(FS\_name, "\_FS")  
 cbind(R, R\_right)  
 R\_down <- R[FS\_name, ]  
 rownames(R\_down) <- paste0(FS\_name, "\_FS")  
 R\_central <- R[FS\_name, FS\_name]  
 R\_down\_central <- cbind(R\_down, R\_central)  
 R\_all <- rbind(cbind(R, R\_right), R\_down\_central)  
  
 FSValidity <- diag(t(FScoef) %\*%  
 R[vObserved, c(vLatent,  
 vDisturbance,  
 vError)])  
 FSStandardError <- sqrt(rep(1,  
 length(c(vLatent,  
 vDisturbance,  
 vError))) - FSValidity)  
 paste0("se.", names(FSStandardError)) -> names(FSStandardError)  
 l$Data <- cbind(d, FactorScores)  
 l$vLatent <- vLatent  
 l$vDisturbance <- vDisturbance  
 l$vFactorScores <- colnames(FactorScores)  
 l$FactorScoreCoef <- FScoef  
 l$R <- R  
 l$R\_all <- R\_all  
 l$R\_FS <- cov2cor(t(FScoef) %\*% Rxx %\*% FScoef)  
 l$FactorScoreValidity <- FSValidity  
 l$FactorScoreSE <- FSStandardError  
 l$Model <- m  
 }  
 l  
 }  
}  
#  
#' Calculate the conditional Mahalanobis distance based on factor scores.  
#' @export  
#' @param R Conditional correlation among variables.  
#' @param Dep The names of variables you would like to condition on.  
#' @param Ind The names of variables of your interest.  
#' @param d Profiles of interest.  
#' @return conditional Mahalanobis distance, percentiles for each case based on the Chi-square distribution formed by conditional Mahalanobis distance and predicted Deps based on Inds.  
#' @examples  
#' SimModel <- "  
#' Gc =~ 0.85 \* Gc1 + 0.68 \* Gc2 + 0.8 \* Gc3  
#' Gf =~ 0.8 \* Gf1 + 0.9 \* Gf2 + 0.8 \* Gf3  
#' Gs =~ 0.7 \* Gs1 + 0.8 \* Gs2 + 0.8 \* Gs3  
#' Read =~ 0.66 \* Read1 + 0.85 \* Read2 + 0.91 \* Read3  
#' Math =~ 0.4 \* Math1 + 0.9 \* Math2 + 0.7 \* Math3  
#' Gc ~ 0.6 \* Gf + 0.1 \* Gs  
#' Gf ~ 0.5 \* Gs  
#' Read ~ 0.4 \* Gc + 0.1 \* Gf  
#' Math ~ 0.2 \* Gc + 0.3 \* Gf + 0.1 \* Gs"  
#' d\_demo <- simStandardized(SimModel, 10)  
#' CMahalanobis\_FS(c("Math", "Read"),c("Gf", "Gs", "Gc"),d\_demo$R\_all,d\_demo$Data)  
  
CMahalanobis\_FS <- function(Dep, Ind = NULL, R, d){  
 # change the names for calculating the CMahalanobis of factor scores  
 Ryy <- R[Dep, Dep]  
 DepKeys <- (rownames(R) %in% Dep) \* 1  
  
 if (!is.null(Ind)) {  
 Rxx <- R[Ind, Ind]  
 Rxy <- R[Ind, Dep]  
 Ryx <- R[Dep, Ind]  
  
 iRxx <- solve(Rxx)  
  
 RegBeta <- iRxx %\*% Rxy  
 R2 <- colSums(RegBeta \* Rxy)  
  
 #change the name to select cases  
 Dep\_FS <- paste0(Dep, "\_FS")  
 Ind\_FS <- paste0(Ind, "\_FS")  
 PredictedSubtests <- as.matrix(d[, Ind\_FS]) %\*% RegBeta  
 SubtestDeviations <- d[, Dep\_FS, drop = F] - PredictedSubtests  
 #conditional variance  
 CondCov <- Ryy - Ryx %\*% iRxx %\*% Rxy  
  
 dCM <- (((as.matrix(SubtestDeviations) %\*%  
 solve(CondCov)) \* as.matrix(SubtestDeviations)) %\*%  
 t(t(rep(1, sum(DepKeys)))))  
 df <- length(Dep)  
 p <- pchisq(dCM, df)  
 list(dCM = dCM,  
 df = df,  
 p = p,  
 PredictedSubtests = PredictedSubtests,  
 R2 = R2)  
 }else{  
 Dep\_FS <- paste0(Dep, "\_FS")  
 dCM <- (((as.matrix(d[, Dep\_FS, drop = FALSE]) %\*%  
 solve(Ryy)) \* as.matrix(d[, Dep\_FS, drop = FALSE])) %\*%  
 t(t(rep(1, sum(DepKeys)))))  
 df <- length(Dep)  
 p <- pchisq(dCM, df)  
 list(dCM = dCM,  
 df = df,  
 p = p)  
 }  
}  
  
  
#' Calculate the conditional Mahalanobis distance for any variables.  
#' @export  
#' @param R Conditional correlation among variables.  
#' @param Dep The names of variables you would like to condition on.  
#' @param Ind The names of variables of your interest.  
#' @param d Profiles of interest.  
#' @return conditional Mahalanobis distance, percentiles for each case based on the Chi-square distribution formed by conditional Mahalanobis distance and predicted Deps based on Inds.  
#' @examples  
#' SimModel <- "  
#' Gc =~ 0.85 \* Gc1 + 0.68 \* Gc2 + 0.8 \* Gc3  
#' Gf =~ 0.8 \* Gf1 + 0.9 \* Gf2 + 0.8 \* Gf3  
#' Gs =~ 0.7 \* Gs1 + 0.8 \* Gs2 + 0.8 \* Gs3  
#' Read =~ 0.66 \* Read1 + 0.85 \* Read2 + 0.91 \* Read3  
#' Math =~ 0.4 \* Math1 + 0.9 \* Math2 + 0.7 \* Math3  
#' Gc ~ 0.6 \* Gf + 0.1 \* Gs  
#' Gf ~ 0.5 \* Gs  
#' Read ~ 0.4 \* Gc + 0.1 \* Gf  
#' Math ~ 0.2 \* Gc + 0.3 \* Gf + 0.1 \* Gs"  
#' d\_demo <- simStandardized(SimModel, 10)  
#' CMahalanobis(c("Math", "Read"),c("Gf", "Gs", "Gc"),d\_demo$R\_all,d\_demo$Data)  
  
CMahalanobis <- function(Dep, Ind = NULL, R, d) {  
  
 if (is.list(Dep)) Dep <- unlist(Dep)  
 if (is.list(Ind)) Ind <- unlist(Ind)  
  
 Ryy <- R[Dep, Dep]  
 DepKeys <- (rownames(R) %in% Dep) \* 1  
  
 if (!is.null(Ind)) {  
 Rxx <- R[Ind, Ind]  
 Rxy <- R[Ind, Dep]  
 Ryx <- R[Dep, Ind]  
  
 iRxx <- solve(Rxx)  
  
 RegBeta <- iRxx %\*% Rxy  
 R2 <- colSums(RegBeta \* Rxy)  
  
 # change the name to select cases  
 PredictedSubtests <- as.matrix(d[, Ind]) %\*% RegBeta  
 SubtestDeviations <- d[, Dep, drop = F] - PredictedSubtests  
  
 #conditional variance  
 CondCov <- Ryy - Ryx %\*% iRxx %\*% Rxy  
  
 dCM <- (((as.matrix(SubtestDeviations) %\*%  
 solve(CondCov)) \* as.matrix(SubtestDeviations)) %\*%  
 t(t(rep(1, sum(DepKeys)))))  
  
 df <- length(Dep)  
 p <- pchisq(dCM, df)  
 list(dCM = dCM,  
 df = df,  
 p = p,  
 PredictedSubtests = PredictedSubtests,  
 R2 = R2)  
 } else {  
 dCM <- (((as.matrix(d[, Dep, drop = F]) %\*%  
 solve(Ryy)) \* as.matrix(d[, Dep, drop = F])) %\*%  
 t(t(rep(1, sum(DepKeys)))))  
 df <- length(Dep)  
 p <- pchisq(dCM, df)  
 list(dCM = dCM, df = df, p = p)  
 }  
  
}  
  
#' Wrapper for finding out Mahalanobis distance between variables: this one gives everything for practitioners to use when they only have population relations and their clients' data  
#' @export  
#' @param Model Population relations among variables represented by Lavaan Syntax.  
#' @param Dep The names of variables you would like to condition on.  
#' @param Ind The names of variables of your interest.  
#' @param d Profiles of interest.  
#' @return conditional Mahalanobis distance, percentiles for each case based on the Chi-square distribution formed by conditional Mahalanobis distance and predicted Deps based on Inds.  
#' @examples  
#' SimModel <- "  
#' Gc =~ 0.85 \* Gc1 + 0.68 \* Gc2 + 0.8 \* Gc3  
#' Gf =~ 0.8 \* Gf1 + 0.9 \* Gf2 + 0.8 \* Gf3  
#' Gs =~ 0.7 \* Gs1 + 0.8 \* Gs2 + 0.8 \* Gs3  
#' Read =~ 0.66 \* Read1 + 0.85 \* Read2 + 0.91 \* Read3  
#' Math =~ 0.4 \* Math1 + 0.9 \* Math2 + 0.7 \* Math3  
#' Gc ~ 0.6 \* Gf + 0.1 \* Gs  
#' Gf ~ 0.5 \* Gs  
#' Read ~ 0.4 \* Gc + 0.1 \* Gf  
#' Math ~ 0.2 \* Gc + 0.3 \* Gf + 0.1 \* Gs"  
#' d\_demo <- simStandardized(SimModel, 10)  
#' maha(SimModel, Dep = c("Math", "Read"),Ind = c("Gc", "Gf", "Gs") , d\_demo$Data)  
maha <- function(Model, Dep, Ind = NULL, d) {  
 Output <- estStandardized(Model, d = d)  
 CMahalanobis(  
 Dep = Dep,  
 Ind = Ind,  
 R = Output$R\_all,  
 d = Output$Data  
 )  
}