# Online Supplement 2: Full simulation code

The simulation code consists of two files. pameters.R defines the simulation designs and simulations.R implements the actual Monte Carlo simulation. The simulation design was full-factorial with 720 experimental conditions. Each condition contained 1000 independent replications, which were run as 10 sets of 100 to speed up the computations.

## parameters.R

designMatrix <- expand.grid(sample = c(50, 100, 250, 1000),  
 indicators = c(3,6,9),  
 factorCorrelation = c(.7, .8, .9, 1),  
 loadings = list(c(0.9, 0.8, 0.8),  
 c(0.8, 0.7, 0.7),  
 c(0.5, 0.4, 0.4),  
 c(0.9, 0.6, 0.3),  
 c(0.8, 0.8, 0.8)),   
 crossLoadings = c(0,1,2))  
  
CUTOFFS <- c(0.85, 0.9, 0.95, 1)  
  
REPLICATIONS\_PER\_SET <- 100

## simulations.R

library(lavaan)  
library(MASS)  
library(semTools)  
library(gdata)  
library(boot)   
library(plyr)  
  
# Read the experimental conditions  
  
source("parameters.R")  
  
  
  
# This file is designed to be run on a computer cluster where each condition  
# split into ten replication sets is  
# run as a separate job. Read the condition number from the command line if  
# given. Otherwise run all conditions and all replication sets.  
  
args <- commandArgs(trailingOnly = TRUE)  
  
if(length(args) == 0){  
 designNumbers <- 1:nrow(designMatrix)  
 replicationSets <- 1:10  
} else {  
 n <- as.numeric(args[1])  
 designNumbers <- floor((n-1)/10)  
 replicationSets <- n - 10\*designNumbers  
}  
  
  
############################################################################  
#  
# Main program  
#  
############################################################################  
  
  
# Loop over designs  
  
for(designNumber in designNumbers){  
   
 design <- designMatrix[designNumber, ]  
   
 print(paste("Starting design number", designNumber))  
 print(design)  
   
 # Create the population covariance matrix  
 Phi <- matrix(c(1, design$factorCorrelation, .5,   
 design$factorCorrelation, 1, .5,   
 .5, .5, 1), 3, 3)  
 loadings <- rep(design$loadings[[1]], each = design$indicators/3)  
 Lambda <- diag(3)[rep(1:3, each = design$indicators), ]\*loadings  
   
 # Add crossloadings  
   
 if(design$crossLoadings != 0){  
 for(i in 1:(design$crossLoadings\*design$indicators/3)){  
 j <- design$indicators+i   
 Lambda[j, 1:2] <- Lambda[j, 2]/sqrt(sum(Phi[1:2, 1:2]))  
 }  
 }  
   
 Sigma <- Lambda %\*% Phi %\*% t(Lambda)  
 colnames(Sigma) <- rownames(Sigma) <- paste("x", 1:nrow(Sigma), sep = "")  
 diag(Sigma) <- 1  
   
 # CFA model specifications for the estimated models.  
 # This creates a total of 10 estimated models as different combinations  
 # for the kind of constraints corresponding to chi2(1), chi2(merge), and  
 # chi2(ref) with different cutoff values  
   
 modelSpec <- rbind(expand.grid(type = 1:2, # 1: Equality constraint,   
 #2: less than constraint  
 cutoff = CUTOFFS),   
 expand.grid(type = 3:4, # 3: No constraints,   
 # 4: One factor for F1 and F2  
 cutoff = NA))  
   
 # We use the population values as the starting values to speed up  
 # the simulation  
   
 models <- apply(modelSpec, 1, function(spec){  
   
 # Factor loadings  
 modelStr <- paste(rep(c("F1", ifelse(spec[1]==4, "F1", "F2"), "F3"),   
 each = design$indicators),   
 " =~ start(", loadings, ")\*x", 1:(design$indicators\*3),   
 sep="", collapse="\n")  
   
 # Factor correlations  
 if(spec[1] == 4){  
 modelStr <- paste(modelStr, "\nF1 ~~ start(.5)\*F3")  
 }  
 else{  
 if(spec[1] == 1){  
 modelStr <- paste(modelStr, "\nF1 ~~ ", spec[2], "\*F2", sep="")  
   
 }  
 else if(spec[1] == 2){  
 modelStr <- paste(modelStr, "\nF1 ~~ c\*F2  
 c <", spec[2])  
 }  
 else if(spec[1] == 3){  
 modelStr <- paste(modelStr, "\nF1 ~~ start(",   
 design$factorCorrelation, ")\*F2", sep="")  
 }  
 modelStr <- paste(modelStr, "\nF1 ~~ start(.5)\*F3\nF2 ~~ start(.5)\*F3")  
 }  
   
 # Error variances  
 modelStr <- paste(modelStr, paste("x", 1:(design$indicators\*3),   
 " ~~ start(", 1-loadings^2, ")\*x",   
 1:(design$indicators\*3),   
 collapse="\n", sep=""), sep="\n")  
   
 lavaanify(modelStr, model.type = "cfa",   
 std.lv = TRUE, auto.cov.lv.x = TRUE, auto.var = TRUE)  
 })  
   
 #  
 # This is the main loop for doing replications. Each repliction starts by  
 # generating a dataset from a population Sigma and then applying all  
 # techniques to that dataset  
 #   
   
 for(replicationSet in replicationSets){  
   
 # Run only if the file does not exists already  
 filename <- paste("Design ", designNumber, "-", replicationSet, ".Rdata", sep="")  
   
 if(!file.exists(filename)){  
 set.seed(designNumber\*10+replicationSet)  
   
 results <- do.call(rbind, lapply(1:REPLICATIONS\_PER\_SET, function(replication){  
   
 print(paste("Design:", designNumber, "Replication:", replication))  
 start\_time <- Sys.time()  
   
 # Draw a sample  
 d <- mvrnorm(design$sample, mu = rep(0, nrow(Sigma)), Sigma)  
   
 # Estimate the CFA models  
 fits <- lapply(models, cfa, data = d)  
   
 # Extract the required results from the CFA runs  
   
 fl <- lapply(fits, function(fit){  
 est <- parameterEstimates(fit)  
 names <- paste(apply(est[, 1:3], 1, paste, collapse=""),   
 rep(names(est)[-5:0+ncol(est)], each = nrow(est)))  
 est <- unlist(est[, -5:0+ncol(est)])  
 names(est) <- names  
   
 if(inspect(fit, "converged")) fitInd <- fitMeasures(fit)  
 else fitInd <- NULL  
   
 rel <- reliability(fit)  
   
 t(c(converged = inspect(fit, "converged"),   
 est,   
 fitInd,   
 unmatrix(rel)))  
 })  
   
 cfaResults <- cbind(modelSpec, rbind.fill.matrix(fl))  
   
 # The actual model comparisons are done in summarizeResults.R  
   
 # HTMT is the average of between factor correlations  
 # divided by the arithmentic mean of mean of within factor  
 # correlations  
   
 boots <- boot(d, function(data, indices){  
 S <- cor(data[indices, 1:(design$indicators\*2)])  
 i <- 1:design$indicators  
 mean(S[i, -i]) /   
 sqrt(mean(S[i, i][lower.tri(S[i, i])]) \*  
 mean(S[-i, -i][lower.tri(S[-i, -i])]))  
 }, R = 1000)  
   
 ci <- boot.ci(boots, type ="perc")$percent[4:5]  
   
 htmtResults <- data.frame(type = 5, "F1~~F2 est" = boots$t0,   
 "F1~~F2 se" = sd(boots$t),   
 "F1~~F2 ci.lower" = ci[1],   
 "F1~~F2 ci.upper" = ci[2],   
 check.names = FALSE)  
   
 # Correlation and disattenuated correlation between unit weighted  
 # scale scores (i.e. sums of standardized indicators).   
   
 ds <- scale(d)  
   
 C <- cor(rowSums(ds[, 1:design$indicators]),   
 rowSums(ds[, 1:design$indicators +design$indicators]))  
   
 # Composite reliabilities / omega reliabilities  
   
 r1p <- cfaResults[cfaResults$type == 3, "alpha:F1"]  
 r2p <- cfaResults[cfaResults$type == 3, "alpha:F2"]  
 r1c <- cfaResults[cfaResults$type == 3, "omega:F1"]  
 r2c <- cfaResults[cfaResults$type == 3, "omega:F2"]  
   
 est <- c(C,   
 C/sqrt(r1p\*r2p),   
 C/sqrt(r1c\*r2c))  
   
 # The CIs are calculated based using boostrap  
   
 boots <- boot(d, function(data, indices){  
 d <- data[indices, ]  
 # Disable standard error calculation because it is not needed  
 fit <- cfa(models[[9]], d, se = "none")  
   
 rel <- reliability(fit)  
   
 d <- scale(d)  
 C <- cor(rowSums(d[, 1:design$indicators]),   
 rowSums(d[, 1:design$indicators +design$indicators]))  
   
 # Disattenuate C with alpha and omega  
 c(C/sqrt(rel[1, 1]\*rel[1, 2]),   
 C/sqrt(rel[2, 1]\*rel[2, 2]))  
   
   
 }, R = 1000)  
   
 cisDTR <- boot.ci(boots, type ="perc", index = 1)$percent[4:5]  
 cisDCR <- boot.ci(boots, type ="perc", index = 2)$percent[4:5]  
   
 # 95% percentile intervals  
 ci.lower <- c(NA, cisDTR[1], cisDCR[1])  
 ci.upper <- c(NA, cisDTR[2], cisDCR[2])  
   
 correlationResults <- cbind(type=6:8,   
 "F1~~F2 est" = est,   
 "F1~~F2 ci.lower" = ci.lower,   
 "F1~~F2 ci.upper" = ci.upper,   
 cfaResults[cfaResults$type == 3,   
 c("avevar:F1", "avevar:F2")])  
   
 end\_time <- Sys.time()  
 print(end\_time - start\_time)  
   
 rbind.fill(cfaResults, htmtResults, correlationResults)  
 }))  
   
 save(results, file = filename)  
 } # End of if for running only non-existing files  
 } # End of looping replication sets  
} # End of looping conditions