Simulations to Test the Causes Of, Effects Of, and Solutions for Heywood Cases in Exploratory Factor Analysis

Allison W. Cooperman & Niels. G Waller

June 18, 2020

In this file, we include the full code to replicate the analyses, tables, and figures in the manuscript "Heywood You Go Away! Examining Causes, Effects, and Treatments for Heywood Cases in Exploratory Factor Analysis."

General Set-Up

```
# Load required libraries
library(pacman)
pacman::p_load(fungible,      # Factor analysis routines and model generation
               dplyr,         # Data manipulation
               ggplot2,       # Graphics
               lattice,       # Graphics
               heplots)       # ANOVA effect sizes

# Set number of trials across simulations
ntrials <- 1000
```

Solutions to Harman's (1960) Data Set

```
# Generate Harman example correlation matrix (p. 125)
R.Harman = matrix(c(1,.945,.84,.735,.630,
                    .945,1,.72,.630,.54,
                    .84,.72,1,.56,.48,
                    .735,.63,.56,1,.420,
                    .63,.54,.48,.42,1),
                  nrow = 5, ncol = 5)

# One-factor model with PAF
paf.out = faMain(R = R.Harman,
                 numFactors = 1,
                 facMethod = "fapa")

# One-factor model with MLE
mle.out = faMain(R = R.Harman,
                 numFactors = 1,
```

```r
                  facMethod = "faml",
                  faControl  = list(maxCommunality =  0.9999999))
```

Table 1

```r
# Print loadings and communality estimates
table1.output = cbind(

  # PAF loadings and communalities
  paf.out$loadings, paf.out$h2,

  # ML loadings and communalities
  mle.out$loadings, mle.out$h2

)
colnames(table1.output) = c("PAF Loadings", "PAF h2",
                             "ML Loadings", "ML h2")
round(table1.output, 2)
```

Study 1: Systematic Study of Eight Causes of Heywood Cases

```r
# Set communality bound for MLE
heybound = 0.998

# Create conditions matrix based on design factors
heycause.conditions = expand.grid(samplesize  = c(50, 150, 300, 500),
                                  nfactors    = c(2, 3, 4),
                                  indperfac   = c(3, 5, 8),
                                  loadpattern = c("Constant High",
                                                  "Constant Low",
                                                  "Sequential",
                                                  "One Strong Loading",
                                                  "One Weak Factor",
                                                  "Two Weak Factors"),
                                  crossload   = c("No", "0.2"),
                                  modelerror  = c("None", "Low", "Moderate"),
                                  modelspec   = c("Correct", "Under", "Over"))

# Number of conditions for full simulation
ncond = nrow(heycause.conditions)

# First select approximate model error parameter values for generating
# population models with moderate levels of model approximation error
# Number of trials
ntrials = 50

# Set maximum W attempts
maxW = 50

# Maximum W loading
```

```r
maxW.load = 0.3

# Maximum number of loadings
maxW.numload = 2

# RMSEA range for moderate fit
rmsea.mod.lower = 0.070
rmsea.mod.upper = 0.089

# Extract number of models
nmodels = which(heycause.conditions$modelerror == "Moderate")

# Create conditions matrix for model error values
merror.conditions = expand.grid(epsTKL = c(0.02, 0.05, 0.1, 0.15, 0.2),
                                error.var = seq(.1, .8, length.out = 8))

# Extract number of model error conditions
nerror = nrow(merror.conditions)

# Create results matrix for "ideal" model error values
results.moderror = matrix(NA, nrow = nmodels, ncol = 8)
colnames(results.mat) = c("epsTKL", "ErrorVar", "RMSEA.mean", "RMSEA.min",
                          "RMSEA.25", "RMSEA.75", "RMSEA.max", "NumNA")

# Begin for-loop across population models
for(iModel in nmodels) {

  # Number of variables in model
  nind <- heycause.conditions$indperfac[iModel]
  nfac <- heycause.conditions$nfactors[iModel]
  nvar <- nfac * nind

  # Factor loading pattern: Loadings vector
  switch(as.character(heycause.conditions$loadpattern[iModel]),
         "Constant High" = {

           simlvec <- c(.8, .8)
           simhvec <- rep(.64, nvar)

         },
         "Constant Low" = {

           simlvec <- c(.3, .3)
           simhvec <- rep(.09, nvar)

         },
         "Sequential" = {

           simlvec <- c(.8, .3)
           simhvec <- NULL

         },
         "One Strong Loading" = {
           simlvec <- c(.8, .5)
           simhvec <- rep(c(.64, rep(.25, nind - 1)), nfac)

         },
         "One Weak Factor" = {

           simlvec <- c(.8, .3)
           simhvec <- c(rep(rep(.64, nind), nfac - 1), rep(.09, nind))

         },
         "Two Weak Factors" = {

           simlvec <- c(.8, .3)
           simhvec <- c(rep(rep(.64, nind), nfac - 2), rep(.09, nind*2))

         })

  # Cross-loadings
  switch(as.character(heycause.conditions$crossload[iModel]),
         "No" = {

           simcross <- 0
           simcrossrange <- c(0, 0)

         },
         "0.2" = {

           simcross <- 1
           simcrossrange <- c(0.2, 0.2)

         })

  # Allocate memory for average RMSEA values across error conditions
  rmsea.avg.error = matrix(NA, nrow = nerror, ncol = 7)
  colnames(rmsea.avg.error) = c("Mean", "SD", "Min", "Max",
                                "25.Quant", "75.Quant", "NumNA")

  # For-loop across error values
  for(iError in 1:nerror) {

    # Set epsTKL value
    epsTKL.val = merror.conditions$epsTKL[iError]

    # Set model error variance value
    merror.val = merror.conditions$error.var[iError]

    # Allocate memory for RMSEA values across trials
    rmsea.trial = numeric(ntrials)

    # For-loop across trials
    for(iTrial in 1:ntrials) {

      # Run simFA
      simout = simFA(Model         = list(NFac           = nfac,
                                          NItemPerFac    = nind,
                                          Model          = "orthogonal"),
                     Loadings      = list(FacLoadDist    = "sequential",
                                          FacLoadRange   = simlvec,
                                          h2             = simhvec),
                     CrossLoadings = list(ProbCrossLoad  = simcross,
                                          CrossLoadRange = simcrossrange),
                     ModelError    = list(ModelError     = TRUE,
                                          ModelErrorType = "U",
                                          ModelErrorVar  = merror.val,
                                          epsTKL         = epsTKL.val,
                                          Wattempts      = maxW,
                                          WmaxLoading    = maxW.load,
                                          NWmaxLoading   = maxW.numload),
                     Seed          = iTrial)

      # Save RMSEA value
      rmsea.trial[iTrial] = simout$ModelErrorFitStats$RMSEA_thetahat

    } # end for iTrial in 1:ntrials

    # Save average RMSEA value for error conditions
    rmsea.avg.error[iError, 1] = mean(rmsea.trial, na.rm = T)
    rmsea.avg.error[iError, 2] = sd(rmsea.trial, na.rm = T)
    rmsea.avg.error[iError, 3] = min(rmsea.trial, na.rm = T)
    rmsea.avg.error[iError, 4] = max(rmsea.trial, na.rm = T)
    rmsea.avg.error[iError, 5] = quantile(rmsea.trial, 0.25, na.rm = T)
    rmsea.avg.error[iError, 6] = quantile(rmsea.trial, 0.75, na.rm = T)
    rmsea.avg.error[iError, 7] = sum(is.na(rmsea.trial))

    # Update console
    cat("\nCompleted model", iModel, "Error Combination", iError)

  } # end for iError in 1:nerror

  # Identify parameter values with minimums and maximums within bounds
  rmsea.avg.error = cbind(merror.conditions$epsTKL,
                          merror.conditions$error.var,
                          rmsea.avg.error)
  potential.values = subset(rmsea.avg.error,
                            (rmsea.avg.error[,3] >= rmsea.mod.lower) &
                              (rmsea.avg.error[,3] <= rmsea.mod.upper))

  # Choose smallest error values with smallest number of NAs
  # to help speed up simulatioon processing time
  which.small = which.min(potential.values[,9])
  results.moderror[iModel, 1] = potential.values[which.small, 1]
  results.moderror[iModel, 2] = potential.values[which.small, 2]
  results.moderror[iModel, 3] = potential.values[which.small, 3]
  results.moderror[iModel, 4] = potential.values[which.small, 5]
  results.moderror[iModel, 5] = potential.values[which.small, 7]
  results.moderror[iModel, 6] = potential.values[which.small, 8]
```

```
  results.moderror[iModel, 7] = potential.values[which.small, 6]
  results.moderror[iModel, 8] = potential.values[which.small, 9]

} # end for iModel in nmodels

# Combine conditions matrices
heycause.conditions = cbind(heycause.conditions, results.moderror)

# Create empty matrices to hold Heywood case prevalence rates during the simulation
heyresultsPAF = heyresultsMLE = matrix(-99, ncol = 4, nrow = ncond)

# For-loop across specified conditions
for(iCond in 1:ncond) {

  # Set model parameters based on condition

      # Number of variables in model
      nind = heycause.conditions$indperfac[iCond]
      nfac = heycause.conditions$nfactors[iCond]
      nvar = nfac * nind

      # Factor loading pattern: Loadings vector
      switch(as.character(heycause.conditions$loadpattern[iCond]),
             "Constant High" = {

               simlvec = c(.8, .8)
               simhvec = rep(.64, nvar)

             },
             "Constant Low" = {

               simlvec = c(.3, .3)
               simhvec = rep(.09, nvar)

             },
             "Sequential" = {

               simlvec = c(.8, .3)
               simhvec = NULL

             },
             "One Strong Loading" = {

               simlvec = c(.8, .5)
               simhvec = rep(c(.64, rep(.25, nind - 1)), nfac)

             },
             "One Weak Factor" = {

               simlvec = c(.8, .3)
               simhvec = c(rep(rep(.64, nind), nfac - 1), rep(.09, nind))

             },
             "Two Weak Factors" = {

               simlvec = c(.8, .3)
               simhvec = c(rep(rep(.64, nind), nfac - 2), rep(.09, nind*2))

             })

      # Cross-loadings
      switch(as.character(heycause.conditions$crossload[iCond]),
             "No" = {

               simcross = 0
               simcrossrange = c(0, 0)

             },
             "0.2" = {

               simcross = 1
               simcrossrange = c(0.2, 0.2)

             })

      # Presence of model error
      if(heycause.conditions$modelerror[iCond] == "None") {

        simerrpres = FALSE
        rmsea.lower = rmsea.upper = NULL

      }
      if(heycause.conditions$modelerror[iCond] == "Low") {

        simerrpres = TRUE
        rmsea.lower = 0.00
        rmsea.upper = 0.05
        simerrepsTKL = 0.2
        simerrvar = 0.1

      }
      if(heycause.conditions$modelerror[iCond] == "Moderate") {

        simerrpres = TRUE
        rmsea.lower = 0.07
        rmsea.upper = 0.089
        simerrepsTKL = heycause.conditions$epsTKL[iCond]
        simerrvar = heycause.conditions$ErrorVar[iCond]

      }

      # Number of factors to extract from sample data
      simnfsamp = switch(as.character(heycause.conditions$modelspec[iCond]),
                         "Correct" = nfac,
                         "Under"   = nfac - 1,
                         "Over"    = nfac + 1)
```

```r
    # Sample size
    simsamplesize = heycause.conditions$samplesize[iCond]

# Empty vectors for saving Heywood results
hey.p = hey.m = rep(-99, ntrials)

# Empty vector for saving whether MLFA converged
ml.conv = rep(-99, ntrials)

# Empty list to save correlation matrices from PAF solutions
cormatlist = as.list(rep(-99, ntrials))

# Begin while-loop to generate sample data and run PAF
# Using a while-loop because if a solution does not converge, we go to the next data set
# Thus we get 1,000 trials of converged (with PAF) solutions
iPAF = 1
iSeed = 0
while(iPAF <= ntrials) {

  # Set seed
  iSeed = iSeed + 1

  # Run simFA to generate sample data based on population model parameters
  simOut = simFA(Model         = list(NFac           = nfac,
                                      NItemPerFac    = nind,
                                      Model          = "orthogonal"),
                 Loadings      = list(FacLoadDist    = "sequential",
                                      FacLoadRange   = simlvec,
                                      h2             = simhvec),
                 CrossLoadings = list(ProbCrossLoad  = simcross,
                                      CrossLoadRange = simcrossrange),
                 ModelError    = list(ModelError     = simerrpres,
                                      ModelErrorType = "U",
                                      ModelErrorVar  = simerrvar,
                                      epsTKL         = simepsTKL),
                 MonteCarlo    = list(NSamples       = 1,
                                      SampleSize     = simsamplesize,
                                      Raw            = TRUE),
                 Seed          = iSeed)

  # Check model fit for model approximation error
  goodfit = (rmsea.lower <= simOut$ModelErrorFitStats$RMSEA_thetahat) &
    (simOut$ModelErrorFitStats$RMSEA_thetahat <= rmsea.upper)
  if(simerrpres == FALSE | (!is.na(goodfit) & goodfit == TRUE)) {

      # Generate correlation matrix
      if(simerrpres == TRUE) {

        Rsamp = cor(simOut$Monte$MCDataME[[1]])

      }
      else {
        Rsamp = cor(simOut$Monte$MCData[[1]])

      }

      # Run PAF factor analysis
      fout.paf = faX(R          = Rsamp,
                     numFactors = simnfsamp,
                     facMethod  = "fapa",
                     faControl  = list(treatHeywood = FALSE))

      # If PAF converged, calculate Heywood proportions and save correlation matrix
      if(fout.paf$faFit$converged == TRUE) {

        # Extract communalities
        h2.paf = fout.paf$h2

        # Compute number of general Heywood cases in factor loadings matrix
        # Communality >= 1
        hey.p[iPAF] = length(which(h2.paf >= 1))

        # Save correlation matrix
        cormatlist[[iPAF]] = Rsamp

        # Go to next trial
        iPAF = iPAF + 1

      }

  }

} # end while(iPAF <= nTrials)

# Begin for-loop to analyze data with MLE
for(iMLE in 1:ntrials) {

  # Factor extraction with MLE
  fout.mle = faX(R          = cormatlist[[iMLE]],
                 numFactors = simnfsamp,
                 facMethod  = "faml",
                 faControl  = list(treatHeywood = FALSE, maxCommunality =  0.9999999))

  # Save convergence results
  ml.conv[iMLE] = if(fout.mle$faFit$converged == TRUE) 1 else 0

  # Extract communalities
  h2.mle = fout.mle$h2

  # Compute number of Heywood cases
  # Communality >= heybound
  hey.m[iMLE] <- length(which(h2.mle >= heybound))

} # end for iMLE in 1:nTrials
```

```r
  # Compute Heywood proportions and save data
  heycause.results.paf = rep(-99, 4)
  heycause.results.paf[1] = iCond
  heycause.results.paf[2] = length(which(hey.p > 0)) / nTrials
  heycause.results.paf[3] = mean(subset(hey.p, hey.p > 0))
  heycause.results.paf[4] = iSeed

  heycause.results.mle = rep(-99, 4)
  heycause.results.mle[1] = iCond
  heycause.results.mle[2] = length(which(hey.m > 0)) / nTrials
  heycause.results.mle[3] = mean(subset(hey.m, hey.m > 0))
  heycause.results.mle[4] = sum(ml.conv) / nTrials

  # Send output to results matrices
  heyresultsPAF[iCond, ] = heycause.results.paf
  heyresultsMLE[iCond, ] = heycause.results.mle

} # end for iCond in 1:ncond

# Convert results matrices into data frames
results.paf = as.data.frame(heyresultsPAF)
results.mle = as.data.frame(heyresultsMLE)

# Add variable names
names(results.paf) = c("Condition", "PropHey", "NHey", "Seed")
names(results.mle) = c("Condition", "PropHey", "NHey", "Conv")

# Add variable for extraction method
results.paf$famethod = "PA"
results.mle$famethod = "ML"

# Merge in conditions matrix
heycause.fullresults.paf = merge(results.paf, heycause.conditions,
                                 by = "Condition")
heycause.fullresults.mle = merge(results.mle, heycause.conditions,
                                 by = "Condition")
heycause.fullresults = rbind(subset(heycause.fullresults.paf,
                                    select = -c(Seed)),
                             subset(heycause.fullresults.mle,
                                    select = -c(Conv)))

# Run ANOVA with two-way interactions
heyanova = aov(PropHey ~ (as.factor(samplesize) + as.factor(nfactors) +
                          as.factor(indperfac) + as.factor(loadpattern) +
                          as.factor(crossload) + as.factor(modelerror) +
                          as.factor(modelspec) + as.factor(famethod))^2,
                 data = heycause.fullresults)
```

Table S1

```r
# Print ANOVA and effect size table, classical effect sizes
print(round(etasq(heyanova, anova = T, partial = F), 2))
```

Figures S1 and S2

```r
# Figure S1 in the online supplement
interaction.plot(x.factor = as.factor(heycause.fullresults$samplesize),
                 trace.factor = as.factor(heycause.fullresults$famethod),
                 response = heycause.fullresults$PropHey,
                 xtick = FALSE,
                 type = "b",
                 pch = 16,
                 trace.label = "Method",
                 xlab = "Sample Size",
                 ylab = "Heywood Case Prevalence Rate",
                 main = "Sample Size by Factor Extraction Method")

# Figure S2 in the online supplement
interaction.plot(x.factor = as.factor(heycause.fullresults$modelspec),
                 trace.factor = as.factor(heycause.fullresults$indperfac),
                 response = heycause.fullresults$PropHey,
                 xtick = FALSE,
                 type = "b",
                 pch = 16,
                 trace.label = "p:k",
                 xlab = "Specification",
                 ylab = "Heywood Case Prevalence Rate",
                 main = "Specification by Indicator-to-Factor Ratio")
```

Table 3

```r
# Summary table for Heywood prevalence rates with PAF
summary.cond <- data.frame(modelspec = c(rep("Correct",12),
                                         rep("Under", 12),
                                         rep("Over", 12)),
                           indperfac = rep(c(rep(3, 4), rep(5, 4), rep(8, 4)), 3),
                           sampsize = rep(c(50, 150, 300, 500), 9))
pafsummary <- NULL
for(i in 1:nrow(summary.cond)) {

  imod <- summary.cond$modelspec[i]
  iindfac <- summary.cond$indperfac[i]
  iss <- summary.cond$sampsize[i]
  aggprop <- (subset(heycause.fullresults.paf,
                     heycause.fullresults.paf$modelspec == imod &
                     heycause.fullresults.paf$indperfac == iindfac &
                     heycause.fullresults.paf$samplesize == iss) %>%
                group_by(loadpattern) %>% summarise(mean = mean(PropHey)))$mean
  pafsummary <- rbind(pafsummary, round(aggprop,4))

}

table3.output <- data.frame(summary.cond, pafsummary)
table3.output <- table3.output[,c(1:4,6:9,5)]
names(table3.output) <- c("Model Specification", "p:k", "N",
                          "Constant High", "Sequential",
                          "One Strong Loading", "One Weak Factor",
                          "Two Weak Factors", "Constant Low")
table3.output[, c(4:9)] <- round(table3.output[, c(4:9)]*100, 2)
print(table3.output)
```

Table 4

```
# Summary table for Heywood prevalence rates with PAF
mlesummary <- NULL
for(i in 1:nrow(summary.cond)) {

  imod <- summary.cond$modelspec[i]
  iindfac <- summary.cond$indperfac[i]
  iss <- summary.cond$sampsize[i]
  aggprop <- (subset(heycause.fullresults.mle,
                     heycause.fullresults.mle$modelspec == imod &
                       heycause.fullresults.mle$indperfac == iindfac &
                       heycause.fullresults.mle$samplesize == iss) %>%
                group_by(loadpattern) %>% summarise(mean = mean(PropHey)))$mean
  mlesummary <- rbind(mlesummary, round(aggprop,4))

}

table4.output <- data.frame(summary.cond, mlesummary)
table4.output <- table4.output[,c(1:4,6:9,5)]
names(table4.output) <- c("Model Specification", "p:k", "N",
                          "Constant High", "Sequential",
                          "One Strong Loading", "One Weak Factor",
                          "Two Weak Factors", "Constant Low")
table4.output[, c(4:9)] <- round(table4.output[, c(4:9)]*100, 2)
print(table4.output)
```

Study 2: Understudied Causes - Factor Score Determinacy

```
# Set model parameters
nfac = 1
nitems = 4
facmethod = "fapa"
sampsize = 100
modtype = "orthogonal"

# Number of FSD iterations
niter = 10

# Starting factor loading patterns
fload.low = c(.35, .35, .35, .35)
fload.seq = c(.45, .35, .3, .25)
fload.one = c(.45, .3, .3, .3)
```

```r
# Set factor loading increase values
constant.val = 0.03

# Set results matrices for each factor loading pattern
results.low = results.seq = results.one = matrix(NA, nrow = niter, ncol = 3)
colnames(results.low) = colnames(results.seq) =
  colnames(results.one) = c("PopDet", "HeyProp", "HeyNum")

# Begin for-loop across FSD increases, constant low condition
for(iIncrease in 1:niter) {

  # Set vectors to hold Heywood case numbers over trials
  heyvec.low = heyvec.numlow = numeric(ntrials)

  # Increase factor loading values
  fload.inc.low = if(iIncrease == 1) fload.low else fload.inc.low + constant.val
  fpattern = matrix(fload.inc.low, ncol = 1, nrow = 4)

  # Begin while-loop across trials
  iTrial = 1
  iSeed = 0
  while(iTrial <= ntrials) {

    # Set seed
    iSeed = iSeed + 1

    # Generate data using simFA
    simout.low = simFA(Model         = list(NFac           = nfac,
                                           NItemPerFac    = nitems,
                                           Model          = modtype),
                       Loadings      = list(FacPattern     = fpattern),
                       ModelError    = list(ModelError     = FALSE),
                       MonteCarlo    = list(NSamples       = 1,
                                           SampleSize     = sampsize,
                                           Raw            = TRUE),
                       Seed          = iSeed)

    # Run PAF extraction
    fout.low = faMain(R = cor(simout.low$Monte$MCData[[1]]),
                      numFactors = 1,
                      facMethod = facmethod,
                      rotate = "none",
                      faControl = list(maxItr = 30000,
                                       epsilon = 1e-4))

    # Check that PAF converged...
    if(fout.low$faFit$convergedX == TRUE) {

      # Save whether there is a Heywood case
      heyvec.low[iTrial] = if(sum(fout.low$h2 >= 1.00) > 0) 1 else 0
      heyvec.numlow[iTrial] = sum(fout.low$h2 >= 1.00)

      # Update console
      cat("\nCompleted Constant Low Increase", iIncrease, "trial", iTrial)

      # Go to next trial
      iTrial = iTrial + 1

    } # end if PAF converged

  } # end while iTrail <= ntrials

  # Save population determinacy
  results.low[iIncrease, 1] = simout.low$Scores$FacInd

  # Save Heywood case prevalence rates
  results.low[iIncrease, 2] = mean(heyvec.low)
  results.low[iIncrease, 3] = mean(heyvec.numlow)

} # end for iIncrease in 1:niter

# Begin for-loop across FSD increases, sequential condition
for(iIncrease in 1:niter) {

  # Set vectors to hold Heywood case numbers over trials
  heyvec.seq = heyvec.numseq = numeric(ntrials)

  # Increase factor loading values
  fload.inc.seq = if(iIncrease == 1) fload.seq else fload.inc.seq + constant.val
  fpattern = matrix(fload.inc.seq, ncol = 1, nrow = 4)

  # Begin while-loop across trials
  iTrial = 1
  iSeed = 0
  while(iTrial <= ntrials) {

    # Set seed
    iSeed = iSeed + 1

    # Generate data using simFA
    simout.seq = simFA(Model         = list(NFac            = nfac,
                                              NItemPerFac     = nitems,
                                              Model           = modtype),
                       Loadings      = list(FacPattern      = fpattern),
                       ModelError    = list(ModelError      = FALSE),
                       MonteCarlo    = list(NSamples        = 1,
                                              SampleSize      = sampsize,
                                              Raw             = TRUE),
                       Seed          = iSeed)

    # Run PAF extraction
    fout.seq = faMain(R = cor(simout.seq$Monte$MCData[[1]]),
                      numFactors = 1,
                      facMethod = facmethod,
                      rotate = "none",
                      faControl = list(maxItr = 30000,
                                      epsilon = 1e-4))

    # Check that PAF converged...
    if(fout.seq$faFit$convergedX == TRUE) {

      # Save whether there is a Heywood case
      heyvec.seq[iTrial] = if(sum(fout.seq$h2 >= 1.00) > 0) 1 else 0
      heyvec.numseq[iTrial] = sum(fout.seq$h2 >= 1.00)

      # Update console
      cat("\nCompleted Sequential Increase", iIncrease, "trial", iTrial)

      # Go to next trial
      iTrial = iTrial + 1

    } # end if PAF converged

  } # end while iTrail <= ntrials

  # Save population determinacy
  results.seq[iIncrease, 1] = simout.seq$Scores$FacInd

  # Save Heywood case prevalence rates
  results.seq[iIncrease, 2] = mean(heyvec.seq)
  results.seq[iIncrease, 3] = mean(heyvec.numseq)

} # end for iIncrease in 1:niter

# Begin for-loop across FSD increases, one stronger loading condition
for(iIncrease in 1:niter) {

  # Set vectors to hold Heywood case numbers over trials
  heyvec.one = heyvec.numone = numeric(ntrials)

  # Increase factor loading values
  fload.inc.one = if(iIncrease == 1) fload.one else fload.inc.one + constant.val
  fpattern = matrix(fload.inc.one, ncol = 1, nrow = 4)

  # Begin while-loop across trials
  iTrial = 1
  iSeed = 0
  while(iTrial <= ntrials) {

    # Set seed
    iSeed = iSeed + 1

    # Generate data using simFA
    simout.one = simFA(Model         = list(NFac            = nfac,
                                            NItemPerFac     = nitems,
                                            Model           = modtype),
                       Loadings      = list(FacPattern      = fpattern),
                       ModelError    = list(ModelError      = FALSE),
                       MonteCarlo    = list(NSamples        = 1,
                                            SampleSize      = sampsize,
                                            Raw             = TRUE),
                        Seed          = iSeed)

    # Run PAF extraction
    fout.one = faMain(R = cor(simout.one$Monte$MCData[[1]]),
                      numFactors = 1,
                      facMethod = facmethod,
                      rotate = "none",
                      faControl = list(maxItr = 30000,
                                       epsilon = 1e-4))

    # Check that PAF converged...
    if(fout.one$faFit$convergedX == TRUE) {

      # Save whether there is a Heywood case
      heyvec.one[iTrial] = if(sum(fout.one$h2 >= 1.00) > 0) 1 else 0
      heyvec.numone[iTrial] = sum(fout.one$h2 >= 1.00)

      # Update console
      cat("\nCompleted One Stronger Increase", iIncrease, "trial", iTrial)

      # Go to next trial
      iTrial = iTrial + 1

    } # end if PAF converged

  } # end while iTrail <= ntrials

  # Save population determinacy
  results.one[iIncrease, 1] = simout.one$Scores$FacInd

  # Save Heywood case prevalence rates
  results.one[iIncrease, 2] = mean(heyvec.one)
  results.one[iIncrease, 3] = mean(heyvec.numone)

} # end for iIncrease in 1:niter

# Combine results matrices
results.mat = cbind(results.low, results.seq, results.one)
colnames(results.mat) = c("PopDet.low", "HeyProp.low", "HeyNum.low",
                          "PopDet.seq", "HeyProp.seq", "HeyNum.seq",
                          "PopDet.one", "HeyProp.one", "HeyNum.one")
```

Figure 1

```
# Create long-format data frame with results across models
# Reshape data long
figure1.output = rbind(

  data.frame("Model"    = "Constant Low",
             "FSD"      = results.mat[,1],
             "HeyProp" = results.mat[,2]),
  data.frame("Model"   = "Sequential",
             "FSD"     = results.mat[,4],
             "HeyProp" = results.mat[,5]),
  data.frame("Model"   = "One Stronger",
             "FSD"     = results.mat[,7],
             "HeyProp" = results.mat[,8])

)

# Graph of Heywood prevalence rates across population determinacy values
# Figure 1
ggplot(figure1.output,
       aes(x = FSD, y = HeyProp, group = Model)) +
  geom_line(aes(linetype = Model)) +
  labs(x = "Population Factor Score Determinacy",
       y = "Heywood Case Prevalence Rate") +
  ggtitle(" ") +
  guides(linetype = guide_legend(title = "Factor Pattern")) +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme(legend.key.width=unit(2,"cm")) +
  xlim(c(.58, 0.90))
```

Study 2: Understudied Causes - Oblique Population Models

```
# Create conditions matrix
conditions.mat = expand.grid(corsize = c(0, .3, .8),
                             merror  = c("None", "Low"),
                             loadpattern = c("Constant High",
                                             "Sequential",
                                             "One Strong Loading",
                                             "One Weak Factor",
                                             "Two Weak Factors",
                                             "Three Weak Factors"))

# Extract number of conditions
ncond = nrow(conditions.mat)

# Set additional model parameters
nfac = 3
nitems = 4
ntotalitems = nfac*nitems
facmethod = "fapa"
sampsize = 100

# Set RMSEA boundaries for low model error
rmsea.lower.low = 0.00
rmsea.upper.low = 0.05

# Set results matrix
results.mat = matrix(NA, nrow = ncond, ncol = 4)
```

```r
colnames(results.mat) = c("HeyProp", "RMSEA.avg", "RMSEA.min", "RMSEA.max")

# Begin for-loop across conditions
for(iCond in 1:ncond) {

  # Set factor correlation
  fcor = conditions.mat$corsize[iCond]
  modtype = if(fcor == 0) "orthogonal" else "oblique"

  # Set factor loading pattern
  switch(as.character(conditions.mat$loadpattern[iCond]),

         "Constant High" = {

           lvec = c(.8, .8)
           hvec = rep(.64, ntotalitems)

         },
         "Sequential" = {

           lvec = c(.8, .3)
           hvec = NULL

         },
         "One Strong Loading" = {

           lvec = c(.8, .5)
           hvec = rep(c(.64, rep(.25, nitems - 1)), nfac)

         },
         "One Weak Factor" = {

           lvec = c(.3, .8)
           hvec = c(rep(.64, ntotalitems - nitems), rep(.09, nitems))

         },
         "Two Weak Factors" = {

           lvec = c(.3, .8)
           hvec = c(rep(.64, ntotalitems - (nitems*2)), rep(.09, nitems*2))

         },
         "Three Weak Factors" = {

           lvec = c(.3, .3)
           hvec = rep(.09, ntotalitems)

         })

  # Vector to save number of Heywood cases across trials
  heywood.vec = numeric(ntrials)

  # Vector to save RMSEA values
```

```r
rmsea.vec = numeric(ntrials)

# While-loop across simulation trials
# to calculate Heywood case prevalence rates
# Use while-loop to ensure that PAF converges
iTrial = 1
iSeed = 1
while(iTrial <= ntrials) {

  # Set seed
  iSeed = iSeed + 1

  # Generate data with no model error
  if(conditions.mat$merror[iCond] == "None") {

    # Generate correlation matrices from population model
    simout = simFA(Model         = list(NFac           = nfac,
                                        NItemPerFac    = nitems,
                                        Model          = modtype),
                   Loadings      = list(FacLoadDist    = "sequential",
                                        FacLoadRange   = lvec,
                                        h2             = hvec),
                   Phi           = list(MaxAbsPhi      = fcor,
                                        PhiType        = "fixed"),
                   ModelError    = list(ModelError     = FALSE),
                   MonteCarlo    = list(NSamples       = 1,
                                        SampleSize     = sampsize,
                                        Raw            = TRUE),
                   Seed          = iSeed)

  } # end if no model error

  # Generate data with low levels of model approximation error
  if(conditions.mat$merror[iCond] != "None") {

    # Set model error variance
    merror.var = 0.1

    # Set initial iteration tracker to FALSE
    goodfit = FALSE
    k = 0

    # Start while loop to find model with RMSEA within specified bounds
    while(goodfit == FALSE) {

      # Update iteration tracker
      k = k + 1

      # Generate correlation matrices from population model
      simout = simFA(Model         = list(NFac           = nfac,
                                          NItemPerFac    = nitems,
                                          Model          = modtype),
                     Loadings      = list(FacLoadDist    = "sequential",
                                          FacLoadRange   = lvec,
                                          h2             = hvec),
                      Phi           = list(MaxAbsPhi      = fcor,
                                          PhiType        = "fixed"),
                      ModelError    = list(ModelError     = TRUE,
                                          NMinorFac      = 150,
                                          ModelErrorType = "U",
                                          ModelErrorVar  = merror.var,
                                          PrintW         = FALSE),
                      MonteCarlo    = list(NSamples       = 1,
                                          SampleSize     = sampsize,
                                          Raw            = TRUE),
                      Seed          = iSeed + (k*100))

      # Check RMSEA
      goodfit = (rmsea.lower.low <= simout$ModelErrorFitStats$RMSEA_thetahat) &
        (simout$ModelErrorFitStats$RMSEA_thetahat <= rmsea.upper.low)

    } # end while goodfit == FALSE

  } # end if low levels of model approximation error

  # Extract sample correlation matrix
  if(conditions.mat$merror[iCond] == "None") {

    rsamp = cor(simout$Monte$MCData[[1]])

  }
  else {

    rsamp = cor(simout$Monte$MCDataME[[1]])

  }

  # EFA with faMain
  fout = faMain(R = rsamp,
                numFactors = nfac,
                facMethod = facmethod,
                rotate = "none",
                faControl = list(maxItr = 30000,
                                 epsilon = 1e-4))

  # If PAF converged...
  if(fout$faFit$convergedX == TRUE) {

    # Save whether there is a Heywood case
    heywood.vec[iTrial] = if(sum(fout$h2 >= 1.00) > 0) 1 else 0

    # Save RMSEA values
    rmsea.vec[iTrial] = simout$ModelErrorFitStats$RMSEA_thetahat

    # Update console
    cat("\nCompleted condition", iCond, "trial", iTrial)
```

```
      # Go to next trial
      iTrial = iTrial + 1

    } # end if PAF converged

  } # End while iTrial <= ntrials

  # Save Heywood case prevalence rate for the condition
  results.mat[iCond, 1] = mean(heywood.vec)

  # Save RMSEA for model
  results.mat[iCond, 2] = if(sum(is.na(rmsea.vec)) == ntrials) NA else mean(rmsea.vec)
  results.mat[iCond, 3] = if(sum(is.na(rmsea.vec)) == ntrials) NA else min(rmsea.vec)
  results.mat[iCond, 4] = if(sum(is.na(rmsea.vec)) == ntrials) NA else max(rmsea.vec)

} # end for iCond in 1:ncond

# Combine results and conditions matrix
results.mat= cbind(conditions.mat, results.mat)
```

Table 5

```
# Sample Heywood case prevalence rates
# across levels of model approximation error
table5.output = data.frame(

  # Loading pattern
  c("A", "B", "C", "D", "E", "F"),

  rbind(

  # Pattern A
  results.mat$HeyProp[results.mat$loadpattern == "Constant High"],

  # Pattern B
  results.mat$HeyProp[results.mat$loadpattern == "Sequential"],

  # Pattern C
  results.mat$HeyProp[results.mat$loadpattern == "One Strong Loading"],

  # Pattern D
  results.mat$HeyProp[results.mat$loadpattern == "One Weak Factor"],

  # Pattern E
  results.mat$HeyProp[results.mat$loadpattern == "Two Weak Factors"],

  # Pattern F
  results.mat$HeyProp[results.mat$loadpattern == "Three Weak Factors"]

))
names(table5.output) = c("Factor Pattern", rep(c("0", "0.3", "0.8"), 2))
```

```
table5.output[,-1] = round(table5.output[,-1], 3)*100
print(table5.output)
```

Study 2: Understudied Causes - Matrix Smoothing

```
# Create function to run simulation for each population model
poc.smoothing <- function(nf,          # Number of common factors
                          itemperfac,  # Number indicators per factor
                          lvec,        # Vector of loading size range
                          hvec = NULL, # Vector of communalities
                          fcor,        # Correlation between factors
                          merr,        # Proportion of model error
                          pmiss,       # Probability of missingness
                          sampsize,    # Sample size
                          fmethod,     # Factor extraction method
                          startSeed,   # Starting seed
                          ntrials) {   # Number of simulation trials

  # Set number of observed variables in the population model
  nvar <- nf*itemperfac

  # Model type (orthogonal or oblique)
  modtype <- if(fcor == 0) "orthogonal" else "oblique"

  # Empty vectors to save Heywood case results
  apa.hey <- by.hey <- kb.hey <- no.hey <- rep(-99, ntrials)

  # Empty vectors to hold RMSE results
  apa.rmse <- by.rmse <- kb.rmse <- no.rmse <- rep(-99, ntrials)
  apa.rmseH <- by.rmseH <- kb.rmseH <- no.rmseH <- rep(-99, ntrials)
  apa.rmseNH <- by.rmseNH <- kb.rmseNH <- no.rmseNH <- rep(-99, ntrials)

  # Set trial number and starting seed
  iTrial <- 1
  iSeed <- startSeed

  # Begin while-loop
  while(iTrial <= ntrials) {

    # Generate simulated data
    iSeed <- iSeed + 1
    simOut <- simFA(Model         = list(NFac          = nf,
                                         NItemPerFac    = itemperfac,
                                         Model          = modtype),
                    Loadings      = list(FacLoadDist    = "sequential",
                                         FacLoadRange   = lvec,
                                         h2             = hvec),
                    ModelError    = list(ModelError     = TRUE,
                                         ModelErrorVar  = merr,
                                         ModelErrorType = "U"),
                    MonteCarlo    = list(NSamples       = 1,
                                          SampleSize     = sampsize,
                                          Raw            = TRUE),
                     Missing       = list(Missing        = TRUE,
                                          Mechanism      = "MCAR",
                                          MSProb         = pmiss),
                     Seed          = iSeed)

    # Compute raw correlation matrix
    Rsamp <- cor(simOut$Monte$MCDataME[[1]], use = "pairwise")

    # If we have a NPD matrix, continue...
    if(min(eigen(Rsamp)$values) < 0) {

      # Smooth NPD matrix with APA
      Rapa <- smoothAPA(R = Rsamp)

      # Smooth NPD matrix with BY
      Rby <- smoothBY(R = Rsamp)

      # Smooth NPD matrix with KB
      Rkb <- smoothKB(R = Rsamp)

      # Set tolerance for zero eigenvalues
      tollevel <- 1e-5

      if(Rapa$convergence == 0 & Rby$convergence == TRUE) {

        # Run factor analysis with APA-smoothed matrix
        fout.apa <- faMain(R = Rapa$RAPA,
                           numFactors = nf,
                           facMethod  = fmethod,
                           rotate = "oblimin",
                           targetMatrix = simOut$loadings,
                           rotateControl = list(numberStarts = 1),
                           faControl  = list(treatHeywood = FALSE))

        # Save convergence information
        fout.apa.conv <- if(fout.apa$faFit$convergedX == TRUE &
                            fout.apa$faFit$convergedR == TRUE) 1 else 0

        # Run factor analysis with BY-smoothed matrix
        fout.by <- faMain(R = Rby$RBY,
                          numFactors = nf,
                          facMethod  = fmethod,
                          rotate = "oblimin",
                          targetMatrix = simOut$loadings,
                          rotateControl = list(numberStarts = 1),
                          faControl  = list(treatHeywood = FALSE))

        # Save convergence information
        fout.by.conv <- if(fout.by$faFit$convergedX == TRUE &
                           fout.by$faFit$convergedR == TRUE) 1 else 0
```

```
# Run factor analysis with KB-smoothed matrix
fout.kb <- faMain(R = Rkb$RKB,
                  numFactors = nf,
                  facMethod  = fmethod,
                  rotate = "oblimin",
                  targetMatrix = simOut$loadings,
                  rotateControl = list(numberStarts = 1),
                  faControl  = list(treatHeywood = FALSE))

# Save convergence information
fout.kb.conv <- if(fout.kb$faFit$convergedX == TRUE &
                   fout.kb$faFit$convergedR == TRUE) 1 else 0

# Run FA with non-smoothed matrix
fout.no <- faMain(R = Rsamp,
                  numFactors = nf,
                  facMethod  = fmethod,
                  rotate = "oblimin",
                  targetMatrix = simOut$loadings,
                  rotateControl = list(numberStarts = 100),
                  faControl  = list(treatHeywood = FALSE))

# Save convergence information
fout.no.conv <- if(fout.no$faFit$convergedX == TRUE &
                   fout.no$faFit$convergedR == TRUE) 1 else 0

# If all factor analyses converged...
if(fout.apa.conv == 1 &
   fout.kb.conv  == 1 &
   fout.by.conv  == 1 &
   fout.no.conv  == 1) {

  # Calculate number of Heywood cases, APA
  h2.apa <- fout.apa$h2
  apa.hey[iTrial] <- length(which(h2.apa >= 1))

  # Calculate RMSD, APA
  apa.rmse[iTrial] <- rmsd(A = simOut$loadings,
                           B = fout.apa$loadings,
                           Symmetric = FALSE)
  apa.rmseH[iTrial] <- if(apa.hey[iTrial] > 0) apa.rmse[iTrial] else NA
  apa.rmseNH[iTrial] <- if(apa.hey[iTrial] == 0) apa.rmse[iTrial] else NA

  # Calculate number of Heywood cases, BY
  h2.by <- fout.by$h2
  by.hey[iTrial] <- length(which(h2.by >= 1))

 # Calculate RMSD, BY
  by.rmse[iTrial] <- rmsd(A = simOut$loadings,
                          B = fout.by$loadings,
                          Symmetric = FALSE)
  by.rmseH[iTrial] <- if(by.hey[iTrial] > 0) by.rmse[iTrial] else NA
  by.rmseNH[iTrial] <- if(by.hey[iTrial] == 0) by.rmse[iTrial] else NA
```

```
        # Calculate number of Heywood cases, KB
        h2.kb <- fout.kb$h2
        kb.hey[iTrial] <- length(which(h2.kb >= 1))

        # Calculate RMSD, K&B
        kb.rmse[iTrial] <- rmsd(A = simOut$loadings,
                                B = fout.kb$loadings,
                                Symmetric = FALSE)
        kb.rmseH[iTrial] <- if(kb.hey[iTrial] > 0) kb.rmse[iTrial] else NA
        kb.rmseNH[iTrial] <- if(kb.hey[iTrial] == 0) kb.rmse[iTrial] else NA

        # Calculate number of Heywood cases, No smoothing
        h2.no <- fout.no$h2
        no.hey[iTrial] <- length(which(h2.no >= 1))

        # Calculate RMSD, no smoothing
        no.rmse[iTrial] <- rmsd(A = simOut$loadings,
                                B = fout.no$loadings,
                                Symmetric = FALSE)
        no.rmseH[iTrial] <- if(no.hey[iTrial] > 0) no.rmse[iTrial] else NA
        no.rmseNH[iTrial] <- if(no.hey[iTrial] == 0) no.rmse[iTrial] else NA

        # Go to next trial
        cat("\nCompleted trial", iTrial)
        iTrial <- iTrial + 1

      } # end if all factor analyses converged

    } # end if all smoothing algorithms converged

  } # end if minimum eigenvalue less than zero

} # end while iTrial < ntrials

# Save results into vector
results.vec <- c(length(which(apa.hey > 0)) / ntrials,
                 mean(subset(apa.hey, apa.hey > 0)),
                 length(which(by.hey > 0)) / ntrials,
                 mean(subset(by.hey, by.hey > 0)),
                 length(which(kb.hey > 0)) / ntrials,
                 mean(subset(kb.hey, kb.hey > 0)),
                 length(which(no.hey > 0)) / ntrials,
                 mean(subset(no.hey, no.hey > 0)))

# Return output
list(heyresults   = results.vec,
     apa.rmse     = apa.rmse,
     by.rmse      = by.rmse,
     kb.rmse      = kb.rmse,
     no.rmse      = no.rmse,
     apa.rmseH    = apa.rmseH,
     by.rmseH     = by.rmseH,
     kb.rmseH     = kb.rmseH,
```

```r
       no.rmseH     = no.rmseH,
       apa.rmseNH   = apa.rmseNH,
       by.rmseNH    = by.rmseNH,
       kb.rmseNH    = kb.rmseNH,
       no.rmseNH    = no.rmseNH,
       endSeed      = iSeed)

} # end poc.smoothing function

# Pattern A: Constant High Loadings
consthigh.miss.01 <- poc.smoothing(nf = 3,
                                   itemperfac = 6,
                                   lvec = c(.8, .8),
                                   fcor = 0,
                                   merr = .1,
                                   pmiss = .25,
                                   sampsize = 100,
                                   fmethod = "fapa",
                                   startSeed = 0,
                                   ntrials = ntrials)
consthigh.miss.03 <- poc.smoothing(nf = 3,
                                   itemperfac = 6,
                                   lvec = c(.8, .8),
                                   fcor = 0,
                                   merr = .3,
                                   pmiss = .25,
                                   sampsize = 100,
                                   fmethod = "fapa",
                                   startSeed = 0,
                                   ntrials = ntrials)

# Pattern B: Sequential Loadings
seq.miss.01 <- poc.smoothing(nf = 3,
                             itemperfac = 6,
                             lvec = c(.8, .3),
                             fcor = 0,
                             merr = .1,
                             pmiss = .25,
                             sampsize = 100,
                             fmethod = "fapa",
                             startSeed = 0,
                             ntrials = ntrials)
seq.miss.03 <- poc.smoothing(nf = 3,
                             itemperfac = 6,
                             lvec = c(.8, .3),
                             fcor = 0,
                             merr = .3,
                             pmiss = .25,
                             sampsize = 100,
                             fmethod = "fapa",
                             startSeed = 0,
                             ntrials = ntrials)
```

```r
# Pattern C: One Strong Loading
sload.miss.01 <- poc.smoothing(nf = 3,
                               itemperfac = 6,
                               lvec = c(.8, .5),
                               hvec = rep(c(.64, rep(.25, 5)), 3),
                               fcor = 0,
                               merr = .1,
                               pmiss = .25,
                               sampsize = 100,
                               fmethod = "fapa",
                               startSeed = 0,
                               ntrials = ntrials)
sload.miss.03 <- poc.smoothing(nf = 3,
                               itemperfac = 6,
                               lvec = c(.8, .5),
                               hvec = rep(c(.64, rep(.25, 5)), 3),
                               fcor = 0,
                               merr = .3,
                               pmiss = .25,
                               sampsize = 100,
                               fmethod = "fapa",
                               startSeed = 0,
                               ntrials = ntrials)

# Pattern D: One Weak Factor
oneweak.miss.01 <- poc.smoothing(nf = 3,
                                 itemperfac = 6,
                                 lvec = c(.8, .3),
                                 hvec = c(rep(.64, 12), rep(.09, 6)),
                                 fcor = 0,
                                 merr = .1,
                                 pmiss = .25,
                                 sampsize = 100,
                                 fmethod = "fapa",
                                 startSeed = 0,
                                 ntrials = ntrials)
oneweak.miss.03 <- poc.smoothing(nf = 3,
                                 itemperfac = 6,
                                 lvec = c(.8, .3),
                                 hvec = c(rep(.64, 12), rep(.09, 6)),
                                 fcor = 0,
                                 merr = .3,
                                 pmiss = .25,
                                 sampsize = 100,
                                 fmethod = "fapa",
                                 startSeed = 0,
                                 ntrials = ntrials)

# Pattern E: Two Weak Factors
twoweak.miss.01 <- poc.smoothing(nf = 3,
                                 itemperfac = 6,
                                 lvec = c(.8, .3),
                                 hvec = c(rep(.64, 6), rep(.09, 12)),
                                 fcor = 0,
                                 merr = .1,
                                 pmiss = .25,
                                 sampsize = 100,
                                 fmethod = "fapa",
                                 startSeed = 0,
                                 ntrials = ntrials)
twoweak.miss.03 <- poc.smoothing(nf = 3,
                                 itemperfac = 6,
                                 lvec = c(.8, .3),
                                 hvec = c(rep(.64, 6), rep(.09, 12)),
                                 fcor = 0,
                                 merr = .3,
                                 pmiss = .25,
                                 sampsize = 100,
                                 fmethod = "fapa",
                                 startSeed = 0,
                                 ntrials = ntrials)

# Pattern F: Three Weak Factors
threeweak.miss.01 <- poc.smoothing(nf = 3,
                                   itemperfac = 6,
                                   lvec = c(.3, .3),
                                   fcor = 0,
                                   merr = .1,
                                   pmiss = .25,
                                   sampsize = 100,
                                   fmethod = "fapa",
                                   startSeed = 0,
                                   ntrials = ntrials)
threeweak.miss.03 <- poc.smoothing(nf = 3,
                                   itemperfac = 6,
                                   lvec = c(.3, .3),
                                   fcor = 0,
                                   merr = .3,
                                   pmiss = .25,
                                   sampsize = 100,
                                   fmethod = "fapa",
                                   startSeed = 0,
                                   ntrials = ntrials)
```

Table 6

```
# Print table of Heywood case prevalence rates
table6.output <- data.frame(rbind(

  # Pattern A: 0.1 Error
  c(consthigh.miss.01$heyresults[c(7,1,3,5)]*100),

  # Pattern B: 0.1 Error
  c(seq.miss.01$heyresults[c(7,1,3,5)]*100),
```

```r
  # Pattern C: 0.1 Error
  c(sload.miss.01$heyresults[c(7,1,3,5)]*100),

  # Pattern D: 0.1 Error
  c(oneweak.miss.01$heyresults[c(7,1,3,5)]*100),

  # Pattern E: 0.1 Error
  c(twoweak.miss.01$heyresults[c(7,1,3,5)]*100),

  # Pattern F: 0.1 Error
  c(threeweak.miss.01$heyresults[c(7,1,3,5)]*100),

  # Pattern A: 0.3 Error
  c(consthigh.miss.03$heyresults[c(7,1,3,5)]*100),

  # Pattern B: 0.3 Error
  c(seq.miss.03$heyresults[c(7,1,3,5)]*100),

  # Pattern C: 0.3 Error
  c(sload.miss.03$heyresults[c(7,1,3,5)]*100),

  # Pattern D: 0.3 Error
  c(oneweak.miss.03$heyresults[c(7,1,3,5)]*100),

  # Pattern E: 0.3 Error
  c(twoweak.miss.03$heyresults[c(7,1,3,5)]*100),

  # Pattern F: 0.3 Error
  c(threeweak.miss.03$heyresults[c(7,1,3,5)]*100)

))
names(table6.output) <- c("No Smoothing", "APA", "BY", "KB")
colavg.01 <- apply(table6.output[1:6,], 2, mean)
colavg.03 <- apply(table6.output[7:12,], 2, mean)
table6.output <- data.frame(
  cbind(

    # Loading Pattern
    rep(c("A", "B", "C", "D", "E", "F", "Avg"), 2),

    # Prevalence Rates
    rbind(

      table6.output[1:6,],
      colavg.01,
      table6.output[7:12,],
      colavg.03

    )

  )

)
```

```r
names(table6.output)[1] <- "Loading Pattern"
table6.output[, -1] <- round(table6.output[, -1], 1)
print(table6.output)
```

Table S2

```r
# Print table of RMSE values for smoothed and non-smoothed matrices
tableS2.output <- data.frame(

  # Loading patterns
  rep(c("A", "B", "C", "D", "E", "F"), 2),

  # No smoothing: All solutions
  c(mean(consthigh.miss.01$no.rmse, na.rm = T),
    mean(seq.miss.01$no.rmse, na.rm = T),
    mean(sload.miss.01$no.rmse, na.rm = T),
    mean(oneweak.miss.01$no.rmse, na.rm = T),
    mean(twoweak.miss.01$no.rmse, na.rm = T),
    mean(threeweak.miss.01$no.rmse, na.rm = T),
    mean(consthigh.miss.03$no.rmse, na.rm = T),
    mean(seq.miss.03$no.rmse, na.rm = T),
    mean(sload.miss.03$no.rmse, na.rm = T),
    mean(oneweak.miss.03$no.rmse, na.rm = T),
    mean(twoweak.miss.03$no.rmse, na.rm = T),
    mean(threeweak.miss.03$no.rmse, na.rm = T)),

  # No smoothing: Heywood solutions
  c(mean(consthigh.miss.01$no.rmseH, na.rm = T),
    mean(seq.miss.01$no.rmseH, na.rm = T),
    mean(sload.miss.01$no.rmseH, na.rm = T),
    mean(oneweak.miss.01$no.rmseH, na.rm = T),
    mean(twoweak.miss.01$no.rmseH, na.rm = T),
    mean(threeweak.miss.01$no.rmseH, na.rm = T),
    mean(consthigh.miss.03$no.rmseH, na.rm = T),
    mean(seq.miss.03$no.rmseH, na.rm = T),
    mean(sload.miss.03$no.rmseH, na.rm = T),
    mean(oneweak.miss.03$no.rmseH, na.rm = T),
    mean(twoweak.miss.03$no.rmseH, na.rm = T),
    mean(threeweak.miss.03$no.rmseH, na.rm = T)),

  # No smoothing: Non-Heywood solutions
  c(mean(consthigh.miss.01$no.rmseNH, na.rm = T),
    mean(seq.miss.01$no.rmseNH, na.rm = T),
    mean(sload.miss.01$no.rmseNH, na.rm = T),
    mean(oneweak.miss.01$no.rmseNH, na.rm = T),
    mean(twoweak.miss.01$no.rmseNH, na.rm = T),
    mean(threeweak.miss.01$no.rmseNH, na.rm = T),
    mean(consthigh.miss.03$no.rmseNH, na.rm = T),
    mean(seq.miss.03$no.rmseNH, na.rm = T),
    mean(sload.miss.03$no.rmseNH, na.rm = T),
    mean(oneweak.miss.03$no.rmseNH, na.rm = T),
    mean(twoweak.miss.03$no.rmseNH, na.rm = T),
    mean(threeweak.miss.03$no.rmseNH, na.rm = T)),

  # APA: All solutions
  c(mean(consthigh.miss.01$apa.rmse, na.rm = T),
    mean(seq.miss.01$apa.rmse, na.rm = T),
    mean(sload.miss.01$apa.rmse, na.rm = T),
    mean(oneweak.miss.01$apa.rmse, na.rm = T),
    mean(twoweak.miss.01$apa.rmse, na.rm = T),
    mean(threeweak.miss.01$apa.rmse, na.rm = T),
    mean(consthigh.miss.03$apa.rmse, na.rm = T),
    mean(seq.miss.03$apa.rmse, na.rm = T),
    mean(sload.miss.03$apa.rmse, na.rm = T),
    mean(oneweak.miss.03$apa.rmse, na.rm = T),
    mean(twoweak.miss.03$apa.rmse, na.rm = T),
    mean(threeweak.miss.03$apa.rmse, na.rm = T)),

  # APA: Heywood solutions
  c(mean(consthigh.miss.01$apa.rmseH, na.rm = T),
    mean(seq.miss.01$apa.rmseH, na.rm = T),
    mean(sload.miss.01$apa.rmseH, na.rm = T),
    mean(oneweak.miss.01$apa.rmseH, na.rm = T),
    mean(twoweak.miss.01$apa.rmseH, na.rm = T),
    mean(threeweak.miss.01$apa.rmseH, na.rm = T),
    mean(consthigh.miss.03$apa.rmseH, na.rm = T),
    mean(seq.miss.03$apa.rmseH, na.rm = T),
    mean(sload.miss.03$apa.rmseH, na.rm = T),
    mean(oneweak.miss.03$apa.rmseH, na.rm = T),
    mean(twoweak.miss.03$apa.rmseH, na.rm = T),
    mean(threeweak.miss.03$apa.rmseH, na.rm = T)),

  # APA: Non-Heywood solutions
  c(mean(consthigh.miss.01$apa.rmseNH, na.rm = T),
    mean(seq.miss.01$apa.rmseNH, na.rm = T),
    mean(sload.miss.01$apa.rmseNH, na.rm = T),
    mean(oneweak.miss.01$apa.rmseNH, na.rm = T),
    mean(twoweak.miss.01$apa.rmseNH, na.rm = T),
    mean(threeweak.miss.01$apa.rmseNH, na.rm = T),
    mean(consthigh.miss.03$apa.rmseNH, na.rm = T),
    mean(seq.miss.03$apa.rmseNH, na.rm = T),
    mean(sload.miss.03$apa.rmseNH, na.rm = T),
    mean(oneweak.miss.03$apa.rmseNH, na.rm = T),
    mean(twoweak.miss.03$apa.rmseNH, na.rm = T),
    mean(threeweak.miss.03$apa.rmseNH, na.rm = T)),

  # BY: All solutions
  c(mean(consthigh.miss.01$by.rmse, na.rm = T),
    mean(seq.miss.01$by.rmse, na.rm = T),
    mean(sload.miss.01$by.rmse, na.rm = T),
    mean(oneweak.miss.01$by.rmse, na.rm = T),
    mean(twoweak.miss.01$by.rmse, na.rm = T),
    mean(threeweak.miss.01$by.rmse, na.rm = T),
    mean(consthigh.miss.03$by.rmse, na.rm = T),
    mean(seq.miss.03$by.rmse, na.rm = T),
```

```r
  mean(sload.miss.03$by.rmse, na.rm = T),
  mean(oneweak.miss.03$by.rmse, na.rm = T),
  mean(twoweak.miss.03$by.rmse, na.rm = T),
  mean(threeweak.miss.03$by.rmse, na.rm = T)),

# BY: Heywood solutions
c(mean(consthigh.miss.01$by.rmseH, na.rm = T),
  mean(seq.miss.01$by.rmseH, na.rm = T),
  mean(sload.miss.01$by.rmseH, na.rm = T),
  mean(oneweak.miss.01$by.rmseH, na.rm = T),
  mean(twoweak.miss.01$by.rmseH, na.rm = T),
  mean(threeweak.miss.01$by.rmseH, na.rm = T),
  mean(consthigh.miss.03$by.rmseH, na.rm = T),
  mean(seq.miss.03$by.rmseH, na.rm = T),
  mean(sload.miss.03$by.rmseH, na.rm = T),
  mean(oneweak.miss.03$by.rmseH, na.rm = T),
  mean(twoweak.miss.03$by.rmseH, na.rm = T),
  mean(threeweak.miss.03$by.rmseH, na.rm = T)),

# BY: Non-Heywood solutions
c(mean(consthigh.miss.01$by.rmseNH, na.rm = T),
  mean(seq.miss.01$by.rmseNH, na.rm = T),
  mean(sload.miss.01$by.rmseNH, na.rm = T),
  mean(oneweak.miss.01$by.rmseNH, na.rm = T),
  mean(twoweak.miss.01$by.rmseNH, na.rm = T),
  mean(threeweak.miss.01$by.rmseNH, na.rm = T),
  mean(consthigh.miss.03$by.rmseNH, na.rm = T),
  mean(seq.miss.03$by.rmseNH, na.rm = T),
  mean(sload.miss.03$by.rmseNH, na.rm = T),
  mean(oneweak.miss.03$by.rmseNH, na.rm = T),
  mean(twoweak.miss.03$by.rmseNH, na.rm = T),
  mean(threeweak.miss.03$by.rmseNH, na.rm = T)),

# KB: All solutions
c(mean(consthigh.miss.01$kb.rmse, na.rm = T),
  mean(seq.miss.01$kb.rmse, na.rm = T),
  mean(sload.miss.01$kb.rmse, na.rm = T),
  mean(oneweak.miss.01$kb.rmse, na.rm = T),
  mean(twoweak.miss.01$kb.rmse, na.rm = T),
  mean(threeweak.miss.01$kb.rmse, na.rm = T),
  mean(consthigh.miss.03$kb.rmse, na.rm = T),
  mean(seq.miss.03$kb.rmse, na.rm = T),
  mean(sload.miss.03$kb.rmse, na.rm = T),
  mean(oneweak.miss.03$kb.rmse, na.rm = T),
  mean(twoweak.miss.03$kb.rmse, na.rm = T),
  mean(threeweak.miss.03$kb.rmse, na.rm = T)),

# KB: Heywood solutions
c(mean(consthigh.miss.01$kb.rmseH, na.rm = T),
  mean(seq.miss.01$kb.rmseH, na.rm = T),
  mean(sload.miss.01$kb.rmseH, na.rm = T),
  mean(oneweak.miss.01$kb.rmseH, na.rm = T),
  mean(twoweak.miss.01$kb.rmseH, na.rm = T),
    mean(threeweak.miss.01$kb.rmseH, na.rm = T),
    mean(consthigh.miss.03$kb.rmseH, na.rm = T),
    mean(seq.miss.03$kb.rmseH, na.rm = T),
    mean(sload.miss.03$kb.rmseH, na.rm = T),
    mean(oneweak.miss.03$kb.rmseH, na.rm = T),
    mean(twoweak.miss.03$kb.rmseH, na.rm = T),
    mean(threeweak.miss.03$kb.rmseH, na.rm = T)),

  # KB: Non-Heywood solutions
  c(mean(consthigh.miss.01$kb.rmseNH, na.rm = T),
    mean(seq.miss.01$kb.rmseNH, na.rm = T),
    mean(sload.miss.01$kb.rmseNH, na.rm = T),
    mean(oneweak.miss.01$kb.rmseNH, na.rm = T),
    mean(twoweak.miss.01$kb.rmseNH, na.rm = T),
    mean(threeweak.miss.01$kb.rmseNH, na.rm = T),
    mean(consthigh.miss.03$kb.rmseNH, na.rm = T),
    mean(seq.miss.03$kb.rmseNH, na.rm = T),
    mean(sload.miss.03$kb.rmseNH, na.rm = T),
    mean(oneweak.miss.03$kb.rmseNH, na.rm = T),
    mean(twoweak.miss.03$kb.rmseNH, na.rm = T),
    mean(threeweak.miss.03$kb.rmseNH, na.rm = T))

)
tableS2.output[,-1] <- round(tableS2.output[,-1], 2)
colnames(tableS2.output) <- c("Loading Pattern",
                              "No Smoothing",
                              "No Smoothing: Heywood",
                              "No Smoothing: Non-Heywood",
                              "APA",
                              "APA: Heywood",
                              "APA: Non-Heywood",
                              "BY",
                              "BY: Heywood",
                              "BY: Non-Heywood",
                              "KB",
                              "KB: Heywood",
                              "KB: Non-Heywood")
print(tableS2.output)
```

Study 3: Effects of Heywood Cases on EFA Model Recovery

```
# Function to compute element-wise RMSE
elrmse <- function(loadarray, poparray) {

  # Set number of variables and factors
  nvar <- nrow(loadarray)
  nfac <- ncol(loadarray)

  # Empty matrix to hold results
  rmsemat <- matrix(-99, nrow = nvar, ncol = nfac)
  # For-loop across elements
  for(rownum in 1:nvar) {

    for(colnum in 1:nfac) {

      l.est <- loadarray[rownum, colnum, ]
      l.pop <- poparray[rownum, colnum, ]
      rmsemat[rownum, colnum] <- sqrt(mean((l.est - l.pop)^2))

    } # end for colnum in 1:nfac

  } # end for rownum in 1:nvar

  # Return matrix of element-wise RMSE values
  return(rmsemat)

} # end elrmse function

# Function to identify proportion of times Heywood case appears for each indicator
whereHey <- function(whichHeyVec, numVar, ntrials) {

  # Empty vector to hold counts
  whereheyVec <- rep(0, numVar)

  # For-loop across trials
  for(iTrial in 1:ntrials) {

    # Extract Heywood variable
    # (indicator which had a communality greater than or equal to one)
    heyvar <- whichHeyVec[[iTrial]]

    # Update vector to hold counts
    whereheyVec[heyvar] <- whereheyVec[heyvar] + 1

  } # end for iTrial in 1:trials

  # Compute proportions vector
  whereheyProp <- whereheyVec / ntrials

  # List output
  list(whereHeyProp = whereheyProp)

} # end whereHey function

# Function to generate lists of seeds that produce the
# Heywood and non-Heywood solution sets
# No model approximation error
heyseeds = function(nFac,                                   # Number of common factors
                    itemPerFac,                             # Items per factor
                    mType = c("orthogonal", "oblique"),     # Model type ("orthogonal)
                    h2vec,                                  # Vector of communalities
                    loadvec,                                # Vector of factor loading sizes
                    fcor,                                   # Factor correlation size
                    sampleSize,                          # Sample size
                    fmethod,                             # Type of factor extraction
                    trials,                              # Number of trials
                    heycase = c("Heywood",               # Communality >= 1
                                "Non-Heywood")) {        # All communalities < 1

  # Empty vectors to hold seeds
  seedlist = rep(-99, trials)

  # While-loop set-up
  iNum = 1
  iSeed = 0

  # Begin while-loop
  while (iNum <= trials) {

    # Calculate trial seed
    iSeed = iSeed + 1

    # Generate simulated data
    out = simFA(Model      = list(NFac          = nFac,
                                  NItemPerFac   = itemPerFac,
                                  Model         = mType),
                Loadings   = list(FacLoadDist   = "sequential",
                                  FacLoadRange  = loadvec,
                                  h2            = h2vec),
                MonteCarlo = list(NSamples      = 1,
                                  SampleSize    = sampleSize,
                                  Raw           = TRUE),
                Seed       = iSeed)

    # Extract sample correlation matrix
    Rsamp = cor(out$Monte$MCData[[1]])

    # Run factor extraction with fapa
    fout = faMain(R = Rsamp,
                  numFactors = nFac,
                  facMethod = fmethod,
                  rotate = "none",
                  faControl = list(treatHeywood = FALSE))

    # Compute type of case (Heywood or non-Heywood)
    if(fout$faFit$convergedX==TRUE) {

      hey = length(which(fout$h2 >= 1))
      iCase = if(hey > 0) "H" else "NH"

      # Save seed depending on condition
      if(heycase == "Heywood" & iCase == "H") {

        seedlist[iNum] = iSeed
        cat("\nCompleted seed", iNum)
        iNum = iNum + 1
```

```r
      }
      if(heycase == "Non-Heywood" & iCase == "NH") {

        seedlist[iNum] = iSeed
        cat("\nCompleted seed", iNum)
        iNum = iNum + 1

      }

    }

  } # end while-loop

  # List output
  list(seeds     = seedlist,
       numTrials = iSeed)

} # end heyseeds function

# Function to generate lists of seeds that produce the
# Heywood and non-Heywood solution sets
# Low levels of model approximation error
heyseeds.err <- function(nFac,
                         itemPerFac,
                         mType = c("orthogonal", "oblique"),
                         h2vec,
                         loadvec,
                         fcor,
                         sampleSize,
                         fmethod,
                         trials,
                         fpattern,
                         heycase = c("Heywood",
                                     "Non-Heywood")) {

  # Empty vectors to hold seeds
  seedlist <- rmsealist <- rep(-99, trials)

  # Phi matrix
  simphi <- matrix(fcor, nFac, nFac)
  diag(simphi) <- 1

  # While-loop set-up
  iNum <- 1
  iSeed <- 0

  # Begin while-loop
  while ( iNum <= trials ) {

    # Calculate trial seed
    iSeed <- iSeed + 1

    # Generate simulated data
    out <- simFA(Model      = list(NFac            = nFac,
                                   NItemPerFac     = itemPerFac,
                                   Model           = mType),
                 Loadings   = list(FacLoadDist     = "sequential",
                                   FacLoadRange    = loadvec,
                                   h2              = h2vec),
                 Phi        = list(PhiType         = "user",
                                   UserPhi         = simphi),
                 ModelError = list(ModelError      = TRUE,
                                   NMinorFac       = 150,
                                   ModelErrorType = "U",
                                   ModelErrorVar   = 0.1,
                                   PrintW          = FALSE),
                 MonteCarlo = list(NSamples        = 1,
                                   SampleSize      = sampleSize,
                                   Raw             = TRUE),
                 Seed       = iSeed)

    # Check RMSEA
    goodfit = (0.00 <= out$ModelErrorFitStats$RMSEA_thetahat) &
      (out$ModelErrorFitStats$RMSEA_thetahat <= 0.05)

    # If RMSEA value under 0.05...
    if(goodfit == TRUE) {

      # Extract sample correlation matrix
      Rsamp <- cor(out$Monte$MCDataME[[1]])

      # Run factor extraction with fapa
      fout <- faMain(R = Rsamp,
                     numFactors = nFac,
                     facMethod = fmethod,
                     rotate = "none",
                     faControl = list(treatHeywood = FALSE))

      # Compute type of case (Heywood or non-Heywood)
      if(fout$faFit$convergedX==TRUE) {

        hey <- length(which(fout$h2 >= 1))
        iCase <- if(hey > 0) "H" else "NH"

        # Save seed depending on condition
        if(heycase == "Heywood" & iCase == "H") {

          seedlist[iNum] <- iSeed
          rmsealist[iNum] <- out$ModelErrorFitStats$RMSEA_thetahat
          cat("\nCompleted", fpattern, "Heywood seed", iNum)
          iNum <- iNum + 1

        }
        if(heycase == "Non-Heywood" & iCase == "NH") {

          seedlist[iNum] <- iSeed
          rmsealist[iNum] <- out$ModelErrorFitStats$RMSEA_thetahat
          cat("\nCompleted", fpattern, "Non-Heywood seed", iNum)
          iNum <- iNum + 1

        }

      }

    }

  } # end while-loop

  # List output
  list(seeds     = seedlist,
       rmsea     = rmsealist,
       numTrials = iSeed)

} # end heyseeds.err function

# Pattern B: Sequential Loadings
for(iCase in c("Heywood", "Non-Heywood")) {

    seedlist = heyseeds(nFac = 3,
                        itemPerFac = 4,
                        mType = "orthogonal",
                        h2vec = NULL,
                        loadvec = c(.8, .3),
                        fcor = 0,
                        sampleSize = 150,
                        fmethod = "fapa",
                        trials = ntrials,
                        heycase = iCase)
    seedlist.err = heyseeds.err(nFac = 3,
                                itemPerFac = 4,
                                mType = "orthogonal",
                                h2vec = NULL,
                                loadvec = c(.8, .3),
                                fcor = 0,
                                sampleSize = 150,
                                fmethod = "fapa",
                                trials = ntrials,
                                heycase = iCase)
    casetype = if(iCase == "Heywood") "hey" else "nohey"
    assign(paste0("seeds.seq.", casetype), seedlist)
    assign(paste0("seeds.seq.", casetype, ".err"), seedlist.err)

} # end for iCase in c("Heywood", "Non-Heywood")

# Pattern C: One Strong Loading
for(iCase in c("Heywood", "Non-Heywood")) {

    seedlist = heyseeds(nFac = 3,
                         itemPerFac = 4,
                         mType = "orthogonal",
                         h2vec =  rep(c(.64, .25, .25, .25), 3),
                         loadvec = c(.8, .5),
                         fcor = 0,
                         sampleSize = 150,
                         fmethod = "fapa",
                         trials = ntrials,
                         heycase = iCase)
    seedlist.err = heyseeds.err(nFac = 3,
                                itemPerFac = 4,
                                mType = "orthogonal",
                                h2vec =  rep(c(.64, .25, .25, .25), 3),
                                loadvec = c(.8, .5),
                                fcor = 0,
                                sampleSize = 150,
                                fmethod = "fapa",
                                trials = ntrials,
                                heycase = iCase)
    casetype = if(iCase == "Heywood") "hey" else "nohey"
    assign(paste0("seeds.sload.", casetype), seedlist)
    assign(paste0("seeds.sload.", casetype, ".err"), seedlist.err)

} # end for iCase in c("Heywood", "Non-Heywood")

# Pattern D: One Weak Factor
for(iCase in c("Heywood", "Non-Heywood")) {

    seedlist = heyseeds(nFac = 3,
                         itemPerFac = 4,
                         mType = "orthogonal",
                         h2vec = c(rep(.64, 8), rep(.09, 4)),
                         loadvec = c(.8, .3),
                         fcor = 0,
                         sampleSize = 150,
                         fmethod = "fapa",
                         trials = ntrials,
                         heycase = iCase)
    seedlist.err = heyseeds.err(nFac = 3,
                                itemPerFac = 4,
                                mType = "orthogonal",
                                h2vec = c(rep(.64, 8), rep(.09, 4)),
                                loadvec = c(.8, .3),
                                fcor = 0,
                                sampleSize = 150,
                                fmethod = "fapa",
                                trials = ntrials,
                                heycase = iCase)
    casetype = if(iCase == "Heywood") "hey" else "nohey"
    assign(paste0("seeds.oneweak.", casetype), seedlist)
    assign(paste0("seeds.oneweak.", casetype, ".err"), seedlist.err)

} # end for iCase in c("Heywood", "Non-Heywood")
```

```r
# Pattern E: Two Weak Factors
for(iCase in c("Heywood", "Non-Heywood")) {

    seedlist = heyseeds(nFac = 3,
                         itemPerFac = 4,
                         mType = "orthogonal",
                         h2vec = c(rep(.64, 4), rep(.09, 8)),
                         loadvec = c(.8, .3),
                         fcor = 0,
                         sampleSize = 150,
                         fmethod = "fapa",
                         trials = ntrials,
                         heycase = iCase)
    seedlist.err = heyseeds.err(nFac = 3,
                                itemPerFac = 4,
                                mType = "orthogonal",
                                h2vec = c(rep(.64, 4), rep(.09, 8)),
                                loadvec = c(.8, .3),
                                fcor = 0,
                                sampleSize = 150,
                                fmethod = "fapa",
                                trials = ntrials,
                                heycase = iCase)
    casetype = if(iCase == "Heywood") "hey" else "nohey"
    assign(paste0("seeds.twoweak.", casetype), seedlist)
    assign(paste0("seeds.twoweak.", casetype, ".err"), seedlist.err)

} # end for iCase in c("Heywood", "Non-Heywood")

# Pattern F: Three Weak Factors
for(iCase in c("Heywood", "Non-Heywood")) {

    seedlist = heyseeds(nFac = 3,
                         itemPerFac = 4,
                         mType = "orthogonal",
                         h2vec = rep(.09, 12),
                         loadvec = c(.3, .3),
                         fcor = 0,
                         sampleSize = 150,
                         fmethod = "fapa",
                         trials = ntrials,
                         heycase = iCase)
    seedlist.err = heyseeds.err(nFac = 3,
                                itemPerFac = 4,
                                mType = "orthogonal",
                                h2vec = rep(.09, 12),
                                loadvec = c(.3, .3),
                                fcor = 0,
                                sampleSize = 150,
                                fmethod = "fapa",
                                trials = ntrials,
                                heycase = iCase)
    casetype = if(iCase == "Heywood") "hey" else "nohey"
    assign(paste0("seeds.threeweak.", casetype), seedlist)
    assign(paste0("seeds.threeweak.", casetype, ".err"), seedlist.err)

} # end for iCase in c("Heywood", "Non-Heywood")

# Save seed lists
save(list = c("seeds.oneweak.hey", "seeds.oneweak.nohey",
              "seeds.twoweak.hey", "seeds.twoweak.nohey",
              "seeds.seq.hey", "seeds.seq.nohey",
              "seeds.sload.hey", "seeds.sload.nohey",
              "seeds.threeweak.hey", "seeds.threeweak.nohey",
              "seeds.oneweak.hey.err", "seeds.oneweak.nohey.err",
              "seeds.twoweak.hey.err", "seeds.twoweak.nohey.err",
              "seeds.seq.hey.err", "seeds.seq.nohey.err",
              "seeds.sload.hey.err", "seeds.sload.nohey.err",
              "seeds.threeweak.hey.err", "seeds.threeweak.nohey.err"),
     file = "./HeywoodEffectSeeds.RData")

# Function to examine effects of Heywood cases across population models
heyeffects = function(nfac,                    # Number of common factors
                      ifac,                    # Number of indicators per factor
                      lvec,                    # Factor loading range
                      hvec,                    # Communality vector
                      fcor,                    # Correlation between factors
                      pasamples,               # Number of samples for parallel analysis
                      sampsize,                # Sample size
                      err = c("None", "Low"), # Level of model approximation error
                      seedlistH,               # Seed list for Heywood solutions
                      seedlistNH,              # Seed list for non-Heywood solutions
                      trials) {                # Number of simulation trials

  # Set number of observed variables
  nvar = nfac * ifac

  # Model type (orthogonal or oblique)
  modtype = if(fcor == 0) "orthogonal" else "oblique"

  # Case types
  heycase = c("Heywood", "Non-Heywood")

  # For loop across Heywood case types
  for(iCase in heycase) {

    # Set Heywood case type
    switch(as.character(iCase),
           "Heywood" = {

             modname = "effects.hey"
             iSeedlist = seedlistH

           },
           "Non-Heywood" = {
```

```r
          modname = "effects.nohey"
          iSeedlist = seedlistNH

        })

  # Empty vectors/lists to save results over trials
  facbias.CI = numfacpa = fullrmse =
    salrmse = localmin = rep(-99, trials)
  detbias = sampload = popload =
    loadbias = phibias = whichheyfac = as.list(rep(-99, trials))

  # For-loop across trials
  for(iTrial in 1:trials) {

    # Set seed (generated in heyseeds function)
    iSeed = iSeedlist[iTrial]

    # No model approximation error
    if(err == "None") {

        # Generate population factor model
        simOut =  simFA(Model         = list(NFac          = nfac,
                                             NItemPerFac   = ifac,
                                             Model         = modtype),
                        Loadings      = list(FacLoadDist   = "sequential",
                                             FacLoadRange  = lvec,
                                             h2            = hvec),
                        MonteCarlo    = list(NSamples      = 1,
                                             SampleSize    = sampsize,
                                             Raw           = TRUE),
                        Seed          = iSeed)

        # Save sample correlation matrix
        sampleMat = simOut$Monte$MCData[[1]]
        Rsamp = cor(sampleMat)

    }

    # Lopw model approximation error
    if(err == "Low") {

      # Generate population factor model
      simOut = simFA(Model      = list(NFac           = nfac,
                                       NItemPerFac    = ifac,
                                       Model          = modtype),
                     Loadings   = list(FacLoadDist    = "sequential",
                                       FacLoadRange   = lvec,
                                       h2             = hvec),
                     Phi        = list(PhiType        = "user",
                                       UserPhi        = simphi),
                     ModelError = list(ModelError     = TRUE,
                                       NMinorFac      = 150,
                                       ModelErrorType = "U",
                                    ModelErrorVar  = 0.1,
                                    PrintW         = FALSE),
                  MonteCarlo = list(NSamples       = 1,
                                    SampleSize     = sampsize,
                                    Raw            = TRUE),
                  Seed       = iSeed)

  # Save sample correlation matrix
  sampleMat = simOut$Monte$MCDataME[[1]]
  Rsamp = cor(sampleMat)

}

# Use faMain to obtain rotated loadings
fout = faMain(R = Rsamp, numFactors = nfac,
              facMethod  = "fapa",
              rotate = "oblimin",
              targetMatrix = simOut$loadings,
              faControl  = list(treatHeywood = FALSE),
              rotateControl = list(numberStarts = 100))

# Save local minima results
localmin[iTrial] = fout$numLocalSets

# Which is the Heywood indicator?
whichheyfac[[iTrial]] = which(fout$h2 >= 1)

# Empty matrix for simulated eigenvalues
eigenMat = matrix(0, nrow = pasamples, ncol = ncol(sampleMat))

# Number of subjects
NSubj = nrow(sampleMat)

# Eigenvalues of observed data matrix
obsEigen = eigen(cor(sampleMat))$values

# Start for-loop for parallel analysis
for(iSample in 1:pasamples) {

  # Set seed
  set.seed(iSample)

  # Sample in item responses based on
  # distributional structure of original data
  simData = apply(sampleMat, 2, sample, size = NSubj, replace = FALSE)

  # Matrix of simulated eigenvalues using Pearson correlations
  eigenMat[iSample, ] = eigen(cor(simData))$values

} # end for iSample in 1:pasamples

# Store upper quantile for simulated eigenvalues
# Using 95th percentile
```

```
    Hi = apply(eigenMat, 2, quantile, prob = 0.95)
    numFactorsPA = sum(cumprod(obsEigen > Hi))
    numfacpa[iTrial] = numFactorsPA

    # Compute factor extraction bias
    facbias.CI[iTrial] = numFactorsPA - nfac

    # Factor score determinacy bias (sample - population)
    detbias[[iTrial]] = fout$facIndeterminacy - simOut$Scores$FacInd

    # Save sample and population loadings
    sampload[[iTrial]] = fout$loadings
    popload[[iTrial]] = simOut$loadings

    # Full-model RMSE
    fullrmse[iTrial] = rmsd(A = simOut$loadings,
                            B = fout$loadings,
                            Symmetric = FALSE)

    # RMSE for salient loadings
    salient = sign(simOut$loadings)
    popsal = matrix(simOut$loadings[which(simOut$loadings != 0)],
                    ncol = 3, nrow = 4)
    sampsal = matrix(fout$loadings[which((salient*fout$loadings) != 0)],
                    ncol = 3, nrow = 4)
    salrmse[iTrial] = rmsd(A = popsal,
                           B = sampsal,
                           Symmetric = FALSE)

    # Compute factor loading bias
    loadbias[[iTrial]] = (fout$loadings - simOut$loadings)
    phibias[[iTrial]] = (fout$Phi - simOut$Phi)

    # Go to next trial
    cat("\nCompleted", iCase, "trial", iTrial)

  } # end for iTrial in 1:trials

  # Loading RMSE and bias
  popload = array(unlist(popload), c(nvar, nfac, trials))
  sampload = array(unlist(sampload), c(nvar, nfac, trials))
  load.elrmse = elrmse(loadarray = sampload, poparray = popload)
  loadbias = array(unlist(loadbias), c(nvar, nfac, trials))
  phibias = array(unlist(phibias), c(nfac, nfac, trials))

  # Determinacy distributions and bias
  detbias = array(unlist(detbias), c(nfac, 1, trials))

  # Calculate proportion of times Heywood Case occurred on each indicator
  whichInd = whereHey(whichHeyVec = whichheyfac, numVar = nvar, ntrials = trials)

  # Save results for treatment method
  assign(modname, list(elRMSE     = load.elrmse,
                         loadBias   = loadbias,
                         phiBias    = phibias,
                         fullRMSE   = fullrmse,
                         salRMSE    = salrmse,
                         detBias    = detbias,
                         facBias    = facbias.CI,
                         numFacPA   = numfacpa,
                         indProp    = whichInd,
                         localMin   = localmin))

  } # end for iCase in heycase

  # List output
  list(facbiasPA    = list(Hey  = effects.hey$facBias,
                          NHey = effects.nohey$facBias),
       numfacPA     = list(Hey  = effects.hey$numFacPA,
                          NHey = effects.nohey$numFacPA),
       detBias      = list(Hey  = effects.hey$detBias,
                          NHey = effects.nohey$detBias),
       fullRMSE     = list(Hey  = effects.hey$fullRMSE,
                          NHey = effects.nohey$fullRMSE),
       salRMSE      = list(Hey  = effects.hey$salRMSE,
                          NHey = effects.nohey$salRMSE),
       elRMSE       = list(Hey  = effects.hey$elRMSE,
                          NHey = effects.nohey$elRMSE),
       loadBias     = list(Hey  = effects.hey$loadBias,
                          NHey = effects.nohey$loadBias),
       phiBias      = list(Hey  = effects.hey$phiBias,
                          NHey = effects.nohey$phiBias),
       indProp      = list(Hey  = effects.hey$indProp),
       localMin     = list(Hey  = effects.hey$localMin,
                          NHey = effects.nohey$localMin))

} # end heyeffects function

# Load saved seed information
load("./HeywoodEffectSeeds.RData")

# Pattern B: Sequential Loadings
seq.effects = heyeffects(nfac = 3,
                         ifac = 4,
                         lvec = c(.8, .3),
                         hvec = NULL,
                         fcor = 0,
                         sampsize = 150,
                         pasamples = 1000,
                         err = "None",
                         seedlistH = seeds.seq.hey$seeds,
                         seedlistNH = seeds.seq.nohey$seeds,
                         trials = ntrials)
seq.effects.err = heyeffects(nfac = 3,
                             ifac = 4,
                             lvec = c(.8, .3),
                             hvec = NULL,
                             fcor = 0,
                             sampsize = 150,
                             pasamples = 1000,
                             err = "Low",
                             seedlistH = seeds.seq.hey.err$seeds,
                             seedlistNH = seeds.seq.nohey.err$seeds,
                             trials = ntrials)

# Pattern C: One Strong Loading
sload.effects = heyeffects(nfac = 3,
                           ifac = 4,
                           lvec = c(.8, .5),
                           hvec = rep(c(.64, .25, .25, .25), 3),
                           fcor = 0,
                           sampsize = 150,
                           pasamples = 1000,
                           err = "None",
                           seedlistH = seeds.sload.hey$seeds,
                           seedlistNH = seeds.sload.nohey$seeds,
                           trials = ntrials)
sload.effects.err = heyeffects(nfac = 3,
                               ifac = 4,
                               lvec = c(.8, .5),
                               hvec = rep(c(.64, .25, .25, .25), 3),
                               fcor = 0,
                               sampsize = 150,
                               pasamples = 1000,
                               err = "Low",
                               seedlistH = seeds.sload.hey.err$seeds,
                               seedlistNH = seeds.sload.nohey.err$seeds,
                               trials = ntrials)

# Pattern D: One Weak Factor
oneweak.effects = heyeffects(nfac = 3,
                             ifac = 4,
                             lvec = c(.8, .3),
                             hvec = c(rep(.64, 8), rep(.09, 4)),
                             fcor = 0,
                             sampsize = 150,
                             pasamples = 1000,
                             err = "None",
                             seedlistH = seeds.oneweak.hey$seeds,
                             seedlistNH = seeds.oneweak.nohey$seeds,
                             trials = ntrials)
oneweak.effects.err = heyeffects(nfac = 3,
                                 ifac = 4,
                                 lvec = c(.8, .3),
                                 hvec = c(rep(.64, 8), rep(.09, 4)),
                                 fcor = 0,
                                 sampsize = 150,
                                 pasamples = 1000,
                                 err = "Low",
```

```r
                                  seedlistH = seeds.oneweak.hey.err$seeds,
                                  seedlistNH = seeds.oneweak.nohey.err$seeds,
                                  trials = ntrials)

# Pattern E: Two Weak Factors
twoweak.effects = heyeffects(nfac = 3,
                              ifac = 4,
                              lvec = c(.8, .3),
                              hvec = c(rep(.64, 4), rep(.09, 8)),
                              fcor = 0,
                              sampsize = 150,
                              pasamples = 1000,
                              err = "None",
                              seedlistH = seeds.twoweak.hey$seeds,
                              seedlistNH = seeds.twoweak.nohey$seeds,
                              trials = ntrials)
twoweak.effects.err = heyeffects(nfac = 3,
                                  ifac = 4,
                                  lvec = c(.8, .3),
                                  hvec = c(rep(.64, 4), rep(.09, 8)),
                                  fcor = 0,
                                  sampsize = 150,
                                  pasamples = 1000,
                                  err = "Low",
                                  seedlistH = seeds.twoweak.hey.err$seeds,
                                  seedlistNH = seeds.twoweak.nohey.err$seeds,
                                  trials = ntrials)


# Pattern F: Three Weak Factors
threeweak.effects = heyeffects(nfac = 3,
                                ifac = 4,
                                lvec = c(.3, .3),
                                hvec = rep(.09, 12),
                                fcor = 0,
                                sampsize = 150,
                                pasamples = 1000,
                                err = "None",
                                seedlistH = seeds.threeweak.hey$seeds,
                                seedlistNH = seeds.threeweak.nohey$seeds,
                                trials = ntrials)
threeweak.effects.err = heyeffects(nfac = 3,
                                    ifac = 4,
                                    lvec = c(.3, .3),
                                    hvec = rep(.09, 12),
                                    fcor = 0,
                                    sampsize = 150,
                                    pasamples = 1000,
                                    err = "Low",
                                    seedlistH = seeds.threeweak.hey.err$seeds,
                                    seedlistNH = seeds.threeweak.nohey.err$seeds,
                                    trials = ntrials)
```

Table 7

```r
# Set empty vectors for holding deviations between true and estimated
# dimensionality from parallel analysis
seq.dev.hey = sload.dev.hey = oneweak.dev.hey =
  twoweak.dev.hey = threeweak.dev.hey = rep(-99, ntrials)
seq.dev.hey.err = sload.dev.hey.err = oneweak.dev.hey.err =
  twoweak.dev.hey.err = threeweak.dev.hey.err = rep(-99, ntrials)
seq.dev.nhey = sload.dev.nhey = oneweak.dev.nhey =
  twoweak.dev.nhey = threeweak.dev.nhey = rep(-99, ntrials)
seq.dev.nhey.err = sload.dev.nhey.err = oneweak.dev.nhey.err =
  twoweak.dev.nhey.err = threeweak.dev.nhey.err = rep(-99, ntrials)

# Set number of population common factors
nfac = 3

# Compute squared deviations for each trial
for(iTrial in 1:ntrials) {

  seq.dev.hey[iTrial] = (seq.effects$numfacPA$Hey[iTrial] - nfac)^2
  seq.dev.nhey[iTrial] = (seq.effects$numfacPA$NHey[iTrial] - nfac)^2
  seq.dev.hey.err[iTrial] = (seq.effects.err$numfacPA$Hey[iTrial] - nfac)^2
  seq.dev.nhey.err[iTrial] = (seq.effects.err$numfacPA$NHey[iTrial] - nfac)^2

  sload.dev.hey[iTrial] = (sload.effects$numfacPA$Hey[iTrial] - nfac)^2
  sload.dev.nhey[iTrial] = (sload.effects$numfacPA$NHey[iTrial] - nfac)^2
  sload.dev.hey.err[iTrial] = (sload.effects.err$numfacPA$Hey[iTrial] - nfac)^2
  sload.dev.nhey.err[iTrial] = (sload.effects.err$numfacPA$NHey[iTrial] - nfac)^2

  oneweak.dev.hey[iTrial] = (oneweak.effects$numfacPA$Hey[iTrial] - nfac)^2
  oneweak.dev.nhey[iTrial] = (oneweak.effects$numfacPA$NHey[iTrial] - nfac)^2
  oneweak.dev.hey.err[iTrial] = (oneweak.effects.err$numfacPA$Hey[iTrial] - nfac)^2
  oneweak.dev.nhey.err[iTrial] = (oneweak.effects.err$numfacPA$NHey[iTrial] - nfac)^2

  twoweak.dev.hey[iTrial] = (twoweak.effects$numfacPA$Hey[iTrial] - nfac)^2
  twoweak.dev.nhey[iTrial] = (twoweak.effects$numfacPA$NHey[iTrial] - nfac)^2
  twoweak.dev.hey.err[iTrial] = (twoweak.effects.err$numfacPA$Hey[iTrial] - nfac)^2
  twoweak.dev.nhey.err[iTrial] = (twoweak.effects.err$numfacPA$NHey[iTrial] - nfac)^2

  threeweak.dev.hey[iTrial] = (threeweak.effects$numfacPA$Hey[iTrial] - nfac)^2
  threeweak.dev.nhey[iTrial] = (threeweak.effects$numfacPA$NHey[iTrial] - nfac)^2
  threeweak.dev.hey.err[iTrial] = (threeweak.effects.err$numfacPA$Hey[iTrial] - nfac)^2
  threeweak.dev.nhey.err[iTrial] = (threeweak.effects.err$numfacPA$NHey[iTrial] - nfac)^2

}

# Print summary statistics of estimated dimensionality from parallel analysis
# between Heywood and non-Heywood solution sets
# Table 7
table7.output = data.frame(

  # Loading pattern
  c("No Error: B",
    "No Error: C",
    "No Error: D",
    "No Error: E",
    "No Error: F",
    "Low Error: B",
    "Low Error: C",
    "Low Error: D",
    "Low Error: E",
    "Low Error: F"),

  # Heywood solutions: Mean estimated dimensionality
  c(mean(seq.effects$numfacPA$Hey),
    mean(sload.effects$numfacPA$Hey),
    mean(oneweak.effects$numfacPA$Hey),
    mean(twoweak.effects$numfacPA$Hey),
    mean(threeweak.effects$numfacPA$Hey),
    mean(seq.effects.err$numfacPA$Hey),
    mean(sload.effects.err$numfacPA$Hey),
    mean(oneweak.effects.err$numfacPA$Hey),
    mean(twoweak.effects.err$numfacPA$Hey),
    mean(threeweak.effects.err$numfacPA$Hey)),

  # Heywood solutions: SD estimated dimensionality
  c(sd(seq.effects$numfacPA$Hey),
    sd(sload.effects$numfacPA$Hey),
    sd(oneweak.effects$numfacPA$Hey),
    sd(twoweak.effects$numfacPA$Hey),
    sd(threeweak.effects$numfacPA$Hey),
    sd(seq.effects.err$numfacPA$Hey),
    sd(sload.effects.err$numfacPA$Hey),
    sd(oneweak.effects.err$numfacPA$Hey),
    sd(twoweak.effects.err$numfacPA$Hey),
    sd(threeweak.effects.err$numfacPA$Hey)),

  # Heywood solutions: Minimum estimated dimensionality
  c(min(seq.effects$numfacPA$Hey),
    min(sload.effects$numfacPA$Hey),
    min(oneweak.effects$numfacPA$Hey),
    min(twoweak.effects$numfacPA$Hey),
    min(threeweak.effects$numfacPA$Hey),
    min(seq.effects.err$numfacPA$Hey),
    min(sload.effects.err$numfacPA$Hey),
    min(oneweak.effects.err$numfacPA$Hey),
    min(twoweak.effects.err$numfacPA$Hey),
    min(threeweak.effects.err$numfacPA$Hey)),

  # Heywood solutions: Maximum estimated dimensionality
  c(max(seq.effects$numfacPA$Hey),
    max(sload.effects$numfacPA$Hey),
    max(oneweak.effects$numfacPA$Hey),
    max(twoweak.effects$numfacPA$Hey),
    max(threeweak.effects$numfacPA$Hey),
    max(seq.effects.err$numfacPA$Hey),
```

```r
  max(sload.effects.err$numfacPA$Hey),
  max(oneweak.effects.err$numfacPA$Hey),
  max(twoweak.effects.err$numfacPA$Hey),
  max(threeweak.effects.err$numfacPA$Hey)),

# Heywood solutions: Estimated dimensionality bias
c(mean(seq.effects$facbiasPA$Hey),
  mean(sload.effects$facbiasPA$Hey),
  mean(oneweak.effects$facbiasPA$Hey),
  mean(twoweak.effects$facbiasPA$Hey),
  mean(threeweak.effects$facbiasPA$Hey),
  mean(seq.effects.err$facbiasPA$Hey),
  mean(sload.effects.err$facbiasPA$Hey),
  mean(oneweak.effects.err$facbiasPA$Hey),
  mean(twoweak.effects.err$facbiasPA$Hey),
  mean(threeweak.effects.err$facbiasPA$Hey)),

# Heywood solutions: Estimated dimensionality RMSE
c(sqrt(sum(seq.dev.hey)/ntrials),
  sqrt(sum(sload.dev.hey)/ntrials),
  sqrt(sum(oneweak.dev.hey)/ntrials),
  sqrt(sum(twoweak.dev.hey)/ntrials),
  sqrt(sum(threeweak.dev.hey)/ntrials),
  sqrt(sum(seq.dev.hey.err)/ntrials),
  sqrt(sum(sload.dev.hey.err)/ntrials),
  sqrt(sum(oneweak.dev.hey.err)/ntrials),
  sqrt(sum(twoweak.dev.hey.err)/ntrials),
  sqrt(sum(threeweak.dev.hey.err)/ntrials)),

# Non-Heywood solutions: Mean estimated dimensionality
c(mean(seq.effects$numfacPA$NHey),
  mean(sload.effects$numfacPA$NHey),
  mean(oneweak.effects$numfacPA$NHey),
  mean(twoweak.effects$numfacPA$NHey),
  mean(threeweak.effects$numfacPA$NHey),
  mean(seq.effects.err$numfacPA$NHey),
  mean(sload.effects.err$numfacPA$NHey),
  mean(oneweak.effects.err$numfacPA$NHey),
  mean(twoweak.effects.err$numfacPA$NHey),
  mean(threeweak.effects.err$numfacPA$NHey)),

# Non-Heywood solutions: SD estimated dimensionality
c(sd(seq.effects$numfacPA$NHey),
  sd(sload.effects$numfacPA$NHey),
  sd(oneweak.effects$numfacPA$NHey),
  sd(twoweak.effects$numfacPA$NHey),
  sd(threeweak.effects$numfacPA$NHey),
  sd(seq.effects.err$numfacPA$NHey),
  sd(sload.effects.err$numfacPA$NHey),
  sd(oneweak.effects.err$numfacPA$NHey),
  sd(twoweak.effects.err$numfacPA$NHey),
  sd(threeweak.effects.err$numfacPA$NHey)),
  # Non-Heywood solutions: Minimum estimated dimensionality
  c(min(seq.effects$numfacPA$NHey),
    min(sload.effects$numfacPA$NHey),
    min(oneweak.effects$numfacPA$NHey),
    min(twoweak.effects$numfacPA$NHey),
    min(threeweak.effects$numfacPA$NHey),
    min(seq.effects.err$numfacPA$NHey),
    min(sload.effects.err$numfacPA$NHey),
    min(oneweak.effects.err$numfacPA$NHey),
    min(twoweak.effects.err$numfacPA$NHey),
    min(threeweak.effects.err$numfacPA$NHey)),

  # Non-Heywood solutions: Maximum estimated dimensionality
  c(max(seq.effects$numfacPA$NHey),
    max(sload.effects$numfacPA$NHey),
    max(oneweak.effects$numfacPA$NHey),
    max(twoweak.effects$numfacPA$NHey),
    max(threeweak.effects$numfacPA$NHey),
    max(seq.effects.err$numfacPA$NHey),
    max(sload.effects.err$numfacPA$NHey),
    max(oneweak.effects.err$numfacPA$NHey),
    max(twoweak.effects.err$numfacPA$NHey),
    max(threeweak.effects.err$numfacPA$NHey)),

  # Non-Heywood solutions: Estimated dimensionality bias
  c(mean(seq.effects$facbiasPA$NHey),
    mean(sload.effects$facbiasPA$NHey),
    mean(oneweak.effects$facbiasPA$NHey),
    mean(twoweak.effects$facbiasPA$NHey),
    mean(threeweak.effects$facbiasPA$NHey),
    mean(seq.effects.err$facbiasPA$NHey),
    mean(sload.effects.err$facbiasPA$NHey),
    mean(oneweak.effects.err$facbiasPA$NHey),
    mean(twoweak.effects.err$facbiasPA$NHey),
    mean(threeweak.effects.err$facbiasPA$NHey)),

  # Non-Heywood solutions: Estimated dimensionality RMSE
  c(sqrt(sum(seq.dev.nhey)/ntrials),
    sqrt(sum(sload.dev.nhey)/ntrials),
    sqrt(sum(oneweak.dev.nhey)/ntrials),
    sqrt(sum(twoweak.dev.nhey)/ntrials),
    sqrt(sum(threeweak.dev.nhey)/ntrials),
    sqrt(sum(seq.dev.nhey.err)/ntrials),
    sqrt(sum(sload.dev.nhey.err)/ntrials),
    sqrt(sum(oneweak.dev.nhey.err)/ntrials),
    sqrt(sum(twoweak.dev.nhey.err)/ntrials),
    sqrt(sum(threeweak.dev.nhey.err)/ntrials))

)
names(table7.output) = c("Loading Pattern",
                         "Heywood: Mean",
                         "Heywood: SD",
                         "Heywood: Min",
                         "Heywood: Max",
                         "Heywood: Bias",
                         "Heywood: RMSE",
                         "Non-Heywood: Mean",
                         "Non-Heywood: SD",
                         "Non-Heywood: Min",
                         "Non-Heywood: Max",
                         "Non-Heywood: Bias",
                         "Non-Heywood: RMSE")
table7.output[,-1] = round(table7.output[,-1], 2)
print(table7.output)
```

Table S3

```
# Print table of full-model RMSE values for Heywood and non-Heywood solutions
tableS3.output = data.frame(

  # Loading pattern
  c("No Error: B",
    "No Error: C",
    "No Error: D",
    "No Error: E",
    "No Error: F",
    "Low Error: B",
    "Low Error: C",
    "Low Error: D",
    "Low Error: E",
    "Low Error: F"),

  # Heywood: Full-model RMSE
  c(mean(seq.effects$fullRMSE$Hey),
    mean(sload.effects$fullRMSE$Hey),
    mean(oneweak.effects$fullRMSE$Hey),
    mean(twoweak.effects$fullRMSE$Hey),
    mean(threeweak.effects$fullRMSE$Hey),
    mean(seq.effects.err$fullRMSE$Hey),
    mean(sload.effects.err$fullRMSE$Hey),
    mean(oneweak.effects.err$fullRMSE$Hey),
    mean(twoweak.effects.err$fullRMSE$Hey),
    mean(threeweak.effects.err$fullRMSE$Hey)),

  # Heywood: Salient loading RMSE
  c(mean(seq.effects$salRMSE$Hey),
    mean(sload.effects$salRMSE$Hey),
    mean(oneweak.effects$salRMSE$Hey),
    mean(twoweak.effects$salRMSE$Hey),
    mean(threeweak.effects$salRMSE$Hey),
    mean(seq.effects.err$salRMSE$Hey),
    mean(sload.effects.err$salRMSE$Hey),
    mean(oneweak.effects.err$salRMSE$Hey),
    mean(twoweak.effects.err$salRMSE$Hey),
    mean(threeweak.effects.err$salRMSE$Hey)),
  # Non-Heywood: Full-model RMSE
  c(mean(seq.effects$fullRMSE$NHey),
    mean(sload.effects$fullRMSE$NHey),
    mean(oneweak.effects$fullRMSE$NHey),
    mean(twoweak.effects$fullRMSE$NHey),
    mean(threeweak.effects$fullRMSE$NHey),
    mean(seq.effects.err$fullRMSE$NHey),
    mean(sload.effects.err$fullRMSE$NHey),
    mean(oneweak.effects.err$fullRMSE$NHey),
    mean(twoweak.effects.err$fullRMSE$NHey),
    mean(threeweak.effects.err$fullRMSE$NHey)),

  # Non-Heywood: Salient loading RMSE
  c(mean(seq.effects$salRMSE$NHey),
    mean(sload.effects$salRMSE$NHey),
    mean(oneweak.effects$salRMSE$NHey),
    mean(twoweak.effects$salRMSE$NHey),
    mean(threeweak.effects$salRMSE$NHey),
    mean(seq.effects.err$salRMSE$NHey),
    mean(sload.effects.err$salRMSE$NHey),
    mean(oneweak.effects.err$salRMSE$NHey),
    mean(twoweak.effects.err$salRMSE$NHey),
    mean(threeweak.effects.err$salRMSE$NHey))

)
names(tableS3.output) = c("Loading Pattern",
                          "Heywood: Full-Model",
                          "Heywood: Salient Loadings",
                          "Non-Heywood: Full-Model",
                          "Non-Heywood: Salient Loadings")
tableS3.output[,-1] = round(tableS3.output[,-1], 2)
print(tableS3.output)
```

Table 8

```
# Print differences in element-wise RMSE
# between Heywood and non-Heywood solutions
table8.output = data.frame(

  # Pattern B
  seq.effects$elRMSE$Hey - seq.effects$elRMSE$NHey,

  # Pattern C
  sload.effects$elRMSE$Hey - sload.effects$elRMSE$NHey,

  # Pattern D
  oneweak.effects$elRMSE$Hey - oneweak.effects$elRMSE$NHey,

  # Pattern E
  twoweak.effects$elRMSE$Hey - twoweak.effects$elRMSE$NHey,
```

```r
  # Pattern F
  threeweak.effects$elRMSE$Hey - threeweak.effects$elRMSE$NHey

)
names(table8.output) = c("B: f1", "B: f2", "B: f3",
                         "C: f1", "C: f2", "C: f3",
                         "D: f1", "D: f2", "D: f3",
                         "E: f1", "E: f2", "E: f3",
                         "F: f1", "F: f2", "F: f3")
print(round(table8.output*100))
```

Table S4

```r
# Print differences in element-wise RMSE
# between Heywood and non-Heywood solutions
# among models with low levels of approximation error
tableS4.output = data.frame(

  # Pattern B
  seq.effects.err$elRMSE$Hey - seq.effects.err$elRMSE$NHey,

  # Pattern C
  sload.effects.err$elRMSE$Hey - sload.effects.err$elRMSE$NHey,

  # Pattern D
  oneweak.effects.err$elRMSE$Hey - oneweak.effects.err$elRMSE$NHey,

  # Pattern E
  twoweak.effects.err$elRMSE$Hey - twoweak.effects.err$elRMSE$NHey,

  # Pattern F
  threeweak.effects.err$elRMSE$Hey - threeweak.effects.err$elRMSE$NHey

)
names(tableS4.output) = c("B: f1", "B: f2", "B: f3",
                          "C: f1", "C: f2", "C: f3",
                          "D: f1", "D: f2", "D: f3",
                          "E: f1", "E: f2", "E: f3",
                          "F: f1", "F: f2", "F: f3")
print(round(tableS4.output*100))
```

Table 9

```r
# Print element-wise RMSE and bias for Heywood solutions
table9.output = data.frame(

  # Pattern B
  seq.effects$elRMSE$Hey[,1],
  apply(seq.effects$loadBias$Hey, 1:2, mean)[,1],
  seq.effects$elRMSE$Hey[,2],
  apply(seq.effects$loadBias$Hey, 1:2, mean)[,2],
  seq.effects$elRMSE$Hey[,3],
  apply(seq.effects$loadBias$Hey, 1:2, mean)[,3],

  # Pattern C
  sload.effects$elRMSE$Hey[,1],
  apply(sload.effects$loadBias$Hey, 1:2, mean)[,1],
  sload.effects$elRMSE$Hey[,2],
  apply(sload.effects$loadBias$Hey, 1:2, mean)[,2],
  sload.effects$elRMSE$Hey[,3],
  apply(sload.effects$loadBias$Hey, 1:2, mean)[,3],

  # Pattern D
  oneweak.effects$elRMSE$Hey[,1],
  apply(oneweak.effects$loadBias$Hey, 1:2, mean)[,1],
  oneweak.effects$elRMSE$Hey[,2],
  apply(oneweak.effects$loadBias$Hey, 1:2, mean)[,2],
  oneweak.effects$elRMSE$Hey[,3],
  apply(oneweak.effects$loadBias$Hey, 1:2, mean)[,3],

  # Pattern E
  twoweak.effects$elRMSE$Hey[,1],
  apply(twoweak.effects$loadBias$Hey, 1:2, mean)[,1],
  twoweak.effects$elRMSE$Hey[,2],
  apply(twoweak.effects$loadBias$Hey, 1:2, mean)[,2],
  twoweak.effects$elRMSE$Hey[,3],
  apply(twoweak.effects$loadBias$Hey, 1:2, mean)[,3],

  # Pattern F
  threeweak.effects$elRMSE$Hey[,1],
  apply(threeweak.effects$loadBias$Hey, 1:2, mean)[,1],
  threeweak.effects$elRMSE$Hey[,2],
  apply(threeweak.effects$loadBias$Hey, 1:2, mean)[,2],
  threeweak.effects$elRMSE$Hey[,3],
  apply(threeweak.effects$loadBias$Hey, 1:2, mean)[,3]

)
names(table9.output) = c("B: f1 RMSE", "B: f1 Bias",
                         "B: f2 RMSE", "B: f2 Bias",
                         "B: f3 RMSE", "B: f3 Bias",
                         "C: f1 RMSE", "C: f1 Bias",
                         "C: f2 RMSE", "C: f2 Bias",
                         "C: f3 RMSE", "C: f3 Bias",
                         "D: f1 RMSE", "D: f1 Bias",
                         "D: f2 RMSE", "D: f2 Bias",
                         "D: f3 RMSE", "D: f3 Bias",
                         "E: f1 RMSE", "E: f1 Bias",
                         "E: f2 RMSE", "E: f2 Bias",
                         "E: f3 RMSE", "E: f3 Bias",
                         "F: f1 RMSE", "F: f1 Bias",
                         "F: f2 RMSE", "F: f2 Bias",
                         "F: f3 RMSE", "F: f3 Bias")
print(round(table9.output*100))
```

Table S5

```r
# Print element-wise RMSE and bias for Heywood solutions
# among models with low levels of approximation error
tableS5.output = data.frame(

  # Pattern B
  seq.effects.err$elRMSE$Hey[,1],
  apply(seq.effects.err$loadBias$Hey, 1:2, mean)[,1],
  seq.effects.err$elRMSE$Hey[,2],
  apply(seq.effects.err$loadBias$Hey, 1:2, mean)[,2],
  seq.effects.err$elRMSE$Hey[,3],
  apply(seq.effects.err$loadBias$Hey, 1:2, mean)[,3],

  # Pattern C
  sload.effects.err$elRMSE$Hey[,1],
  apply(sload.effects.err$loadBias$Hey, 1:2, mean)[,1],
  sload.effects.err$elRMSE$Hey[,2],
  apply(sload.effects.err$loadBias$Hey, 1:2, mean)[,2],
  sload.effects.err$elRMSE$Hey[,3],
  apply(sload.effects.err$loadBias$Hey, 1:2, mean)[,3],

  # Pattern D
  oneweak.effects.err$elRMSE$Hey[,1],
  apply(oneweak.effects.err$loadBias$Hey, 1:2, mean)[,1],
  oneweak.effects.err$elRMSE$Hey[,2],
  apply(oneweak.effects.err$loadBias$Hey, 1:2, mean)[,2],
  oneweak.effects.err$elRMSE$Hey[,3],
  apply(oneweak.effects.err$loadBias$Hey, 1:2, mean)[,3],

  # Pattern E
  twoweak.effects.err$elRMSE$Hey[,1],
  apply(twoweak.effects.err$loadBias$Hey, 1:2, mean)[,1],
  twoweak.effects.err$elRMSE$Hey[,2],
  apply(twoweak.effects.err$loadBias$Hey, 1:2, mean)[,2],
  twoweak.effects.err$elRMSE$Hey[,3],
  apply(twoweak.effects.err$loadBias$Hey, 1:2, mean)[,3],

  # Pattern F
  threeweak.effects.err$elRMSE$Hey[,1],
  apply(threeweak.effects.err$loadBias$Hey, 1:2, mean)[,1],
  threeweak.effects.err$elRMSE$Hey[,2],
  apply(threeweak.effects.err$loadBias$Hey, 1:2, mean)[,2],
  threeweak.effects.err$elRMSE$Hey[,3],
  apply(threeweak.effects.err$loadBias$Hey, 1:2, mean)[,3]

)
names(tableS5.output) = c("B: f1 RMSE", "B: f1 Bias",
                          "B: f2 RMSE", "B: f2 Bias",
                          "B: f3 RMSE", "B: f3 Bias",
                          "C: f1 RMSE", "C: f1 Bias",
                          "C: f2 RMSE", "C: f2 Bias",
                          "C: f3 RMSE", "C: f3 Bias",
                          "D: f1 RMSE", "D: f1 Bias",
                          "D: f2 RMSE", "D: f2 Bias",
                          "D: f3 RMSE", "D: f3 Bias",
                          "E: f1 RMSE", "E: f1 Bias",
                          "E: f2 RMSE", "E: f2 Bias",
                          "E: f3 RMSE", "E: f3 Bias",
                          "F: f1 RMSE", "F: f1 Bias",
                          "F: f2 RMSE", "F: f2 Bias",
                          "F: f3 RMSE", "F: f3 Bias")
print(round(tableS5.output*100))
```

Table S6

```
# Print table of factor loading bias for non-Heywood solutions
tableS6.output = data.frame(

  # Pattern B
  apply(seq.effects$loadBias$NHey, 1:2, mean),

  # Pattern C
  apply(sload.effects$loadBias$NHey, 1:2, mean),

  # Pattern D
  apply(oneweak.effects$loadBias$NHey, 1:2, mean),

  # Pattern E
  apply(twoweak.effects$loadBias$NHey, 1:2, mean),

  # Pattern F
  apply(threeweak.effects$loadBias$NHey, 1:2, mean)

)
names(tableS6.output) = c("B: f1", "B: f2", "B: f3",
                          "C: f1", "C: f2", "C: f3",
                          "D: f1", "D: f2", "D: f3",
                          "E: f1", "E: f2", "E: f3",
                          "F: f1", "F: f2", "F: f3")
print(round(tableS6.output*100))
```

Table S7

```
# Print table of factor loading bias for non-Heywood solutions
# among models with low levels of approximation error
tableS7.output = data.frame(

  # Pattern B
  apply(seq.effects.err$loadBias$NHey, 1:2, mean),

  # Pattern C
  apply(sload.effects.err$loadBias$NHey, 1:2, mean),
```

```r
  # Pattern D
  apply(oneweak.effects.err$loadBias$NHey, 1:2, mean),

  # Pattern E
  apply(twoweak.effects.err$loadBias$NHey, 1:2, mean),

  # Pattern F
  apply(threeweak.effects.err$loadBias$NHey, 1:2, mean)

)
names(tableS7.output) = c("B: f1", "B: f2", "B: f3",
                          "C: f1", "C: f2", "C: f3",
                          "D: f1", "D: f2", "D: f3",
                          "E: f1", "E: f2", "E: f3",
                          "F: f1", "F: f2", "F: f3")
print(round(tableS7.output*100))
```

Table S8

```r
# Proportion of times in which a Heywood Case
# occurred on each factor indicator
tableS8.output = data.frame(

  # Indicator number
  1:12,

  # Pattern B
  seq.effects$indProp$Hey,

  # Pattern C
  sload.effects$indProp$Hey,

  # Pattern D
  oneweak.effects$indProp$Hey,

  # Pattern E
  twoweak.effects$indProp$Hey,

  # Pattern F
  threeweak.effects$indProp$Hey

)
names(tableS8.output) = c("Indicator", "B", "C", "D", "E", "F")
tableS8.output[, -1] = round(tableS8.output[, -1]*100, 1)
print(tableS8.output)
```

Table S9

```r
# Proportion of times in which a Heywood Case
# occurred on each factor indicator
# among models with low levels of approximation error
tableS9.output = data.frame(

  # Indicator number
  1:12,

  # Pattern B
  seq.effects.err$indProp$Hey,

  # Pattern C
  sload.effects.err$indProp$Hey,

  # Pattern D
  oneweak.effects.err$indProp$Hey,

  # Pattern E
  twoweak.effects.err$indProp$Hey,

  # Pattern F
  threeweak.effects.err$indProp$Hey

)
names(tableS9.output) = c("Indicator", "B", "C", "D", "E", "F")
tableS9.output[, -1] = round(tableS9.output[, -1]*100, 1)
print(tableS9.output)
```

Table 10

```r
# Set population factor score determinacy values
seq.det.pop <- rep(.86, 3)
sload.det.pop <- rep(.86, 3)
oneweak.det.pop <- c(.94, .94, .53)
twoweak.det.pop <- c(.94, .53, .53)
threeweak.det.pop <- rep(.53, 3)

# Print factor score determinacy bias for
# Heywood and non-Heywood solution sets
table10.output = data.frame(

  # Loading pattern and factor
  c("B: f1", "B: f2", "B: f3",
    "C: f1", "C: f2", "C: f3",
    "D: f1", "D: f2", "D: f3",
    "E: f1", "E: f2", "E: f3",
    "F: f1", "F: f2", "F: f3"),

  # Population factor score determinacy
  c(seq.det.pop,
    sload.det.pop,
    oneweak.det.pop,
    twoweak.det.pop,
    threeweak.det.pop),
  # Heywood: Bias
  c(apply(seq.effects$detBias$Hey, 1:2, mean),
    apply(sload.effects$detBias$Hey, 1:2, mean),
    apply(oneweak.effects$detBias$Hey, 1:2, mean),
    apply(twoweak.effects$detBias$Hey, 1:2, mean),
    apply(threeweak.effects$detBias$Hey, 1:2, mean)),

  # Heywood: Relative bias
  c(apply(seq.effects$detBias$Hey, 1:2, mean)/seq.det.pop,
    apply(sload.effects$detBias$Hey, 1:2, mean)/sload.det.pop,
    apply(oneweak.effects$detBias$Hey, 1:2, mean)/oneweak.det.pop,
    apply(twoweak.effects$detBias$Hey, 1:2, mean)/twoweak.det.pop,
    apply(threeweak.effects$detBias$Hey, 1:2, mean)/threeweak.det.pop),

  # Non-Heywood: Bias
  c(apply(seq.effects$detBias$NHey, 1:2, mean),
    apply(sload.effects$detBias$NHey, 1:2, mean),
    apply(oneweak.effects$detBias$NHey, 1:2, mean),
    apply(twoweak.effects$detBias$NHey, 1:2, mean),
    apply(threeweak.effects$detBias$NHey, 1:2, mean)),

  # Non-Heywood: Relative bias
  c(apply(seq.effects$detBias$NHey, 1:2, mean)/seq.det.pop,
    apply(sload.effects$detBias$NHey, 1:2, mean)/sload.det.pop,
    apply(oneweak.effects$detBias$NHey, 1:2, mean)/oneweak.det.pop,
    apply(twoweak.effects$detBias$NHey, 1:2, mean)/twoweak.det.pop,
    apply(threeweak.effects$detBias$NHey, 1:2, mean)/threeweak.det.pop)

)
names(table10.output) = c("Loading Pattern and Factor",
                          "Population FSD",
                          "Heywood: Bias",
                          "Heywood: Relative Bias",
                          "Non-Heywood: Bias",
                          "Non-Heywood: Relative Bias")
table10.output[,-1] = round(table10.output[,-1], 2)
print(table10.output)
```

Table S10

```
# Print factor score determinacy bias for
# Heywood and non-Heywood solution sets
# among models with low levels of approximation error
tableS10.output = data.frame(

  # Loading pattern and factor
  c("B: f1", "B: f2", "B: f3",
    "C: f1", "C: f2", "C: f3",
    "D: f1", "D: f2", "D: f3",
    "E: f1", "E: f2", "E: f3",
    "F: f1", "F: f2", "F: f3"),
```

```r
  # Population factor score determinacy
  c(seq.det.pop,
    sload.det.pop,
    oneweak.det.pop,
    twoweak.det.pop,
    threeweak.det.pop),

  # Heywood: Bias
  c(apply(seq.effects.err$detBias$Hey, 1:2, mean),
    apply(sload.effects.err$detBias$Hey, 1:2, mean),
    apply(oneweak.effects.err$detBias$Hey, 1:2, mean),
    apply(twoweak.effects.err$detBias$Hey, 1:2, mean),
    apply(threeweak.effects.err$detBias$Hey, 1:2, mean)),

  # Heywood: Relative bias
  c(apply(seq.effects.err$detBias$Hey, 1:2, mean)/seq.det.pop,
    apply(sload.effects.err$detBias$Hey, 1:2, mean)/sload.det.pop,
    apply(oneweak.effects.err$detBias$Hey, 1:2, mean)/oneweak.det.pop,
    apply(twoweak.effects.err$detBias$Hey, 1:2, mean)/twoweak.det.pop,
    apply(threeweak.effects.err$detBias$Hey, 1:2, mean)/threeweak.det.pop),

  # Non-Heywood: Bias
  c(apply(seq.effects.err$detBias$NHey, 1:2, mean),
    apply(sload.effects.err$detBias$NHey, 1:2, mean),
    apply(oneweak.effects.err$detBias$NHey, 1:2, mean),
    apply(twoweak.effects.err$detBias$NHey, 1:2, mean),
    apply(threeweak.effects.err$detBias$NHey, 1:2, mean)),

  # Non-Heywood: Relative bias
  c(apply(seq.effects.err$detBias$NHey, 1:2, mean)/seq.det.pop,
    apply(sload.effects.err$detBias$NHey, 1:2, mean)/sload.det.pop,
    apply(oneweak.effects.err$detBias$NHey, 1:2, mean)/oneweak.det.pop,
    apply(twoweak.effects.err$detBias$NHey, 1:2, mean)/twoweak.det.pop,
    apply(threeweak.effects.err$detBias$NHey, 1:2, mean)/threeweak.det.pop)

)
names(tableS10.output) = c("Loading Pattern and Factor",
                           "Population FSD",
                           "Heywood: Bias",
                           "Heywood: Relative Bias",
                           "Non-Heywood: Bias",
                           "Non-Heywood: Relative Bias")
tableS10.output[,-1] = round(tableS10.output[,-1], 2)
print(tableS10.output)
```

Study 4: Evaluating Potential Treatments for Heywood Cases

```r
# Function to compute Heywood case prevalence rates across population models
poc.refa = function(nfac,                    # Number of common factors
                    ifac,                    # Number of indicators per factor
                    lvec,                    # Factor loading range
```

```
                     hvec,                        # Vector of communalities
                     fcor,                        # Correlations between factors
                     sampsize,                    # Sample size
                     err = c("None", "Low"), # Level of model approximation error
                     seedlistH,                   # Seed list for Heywood solution set
                     seedlistNH,                  # Seed list for non-Heywood solution set
                     trials) {                    # Number of simulation trials

  # Set number of observed variables
  nvar = nfac * ifac

  # Model type (orthogonal or oblique)
  modtype = if(fcor == 0) "orthogonal" else "oblique"

  # Treatment methods
  heytreat = c("Heywood-PAF", "CLS", "REFA-LS", "REFA-ML",
               "Non-Heywood-PAF", "Non-Heywood-REFA-LS", "Non-Heywood-REFA-ML")

  # For loop across treatment methods
  for(iTreat in heytreat) {

    # Set factor method
    switch(as.character(iTreat),
           "Heywood-PAF" = {

             fmethod = "fapa"
             modname = "results.heyPAF"
             iSeedlist = seedlistH

           },
           "CLS" = {

             fmethod = "fals"
             modname = "results.cls"
             iSeedlist = seedlistH

           },
           "REFA-LS" = {

             fmethod = "faregLS"
             modname = "results.refaLS"
             iSeedlist = seedlistH

           },
           "REFA-ML" = {

             fmethod = "faregML"
             modname = "results.refaML"
             iSeedlist = seedlistH

           },
           "Non-Heywood-PAF" = {
```

```r
          fmethod = "fapa"
          modname = "results.nheyPAF"
          iSeedlist = seedlistNH

        },
        "Non-Heywood-REFA-LS" = {

          fmethod <- "faregLS"
          modname <- "results.nheyRLS"
          iSeedlist <- seedlistNH

        },
        "Non-Heywood-REFA-ML" = {

          fmethod <- "faregML"
          modname <- "results.nheyRML"
          iSeedlist <- seedlistNH

        })

  # Empty vectors/lists to save results over trials
  detdist = detbias = as.list(rep(-99, trials))
  salrmse = fullrmse = rep(-99, trials)

  # For-loop across trials
  for(iTrial in 1:trials) {

    # Set seed
    iSeed = iSeedlist[iTrial]

    # No model approximation error
    if(err == "None") {

      # Generate population factor model
      simOut =  simFA(Model         = list(NFac          = nfac,
                                           NItemPerFac   = ifac,
                                           Model         = modtype),
                      Loadings      = list(FacLoadDist   = "sequential",
                                           FacLoadRange  = lvec,
                                           h2            = hvec),
                      MonteCarlo    = list(NSamples      = 1,
                                           SampleSize    = sampsize,
                                           Raw           = TRUE),
                      Seed          = iSeed)

      # Save sample correlation matrix
      sampleMat = simOut$Monte$MCData[[1]]
      Rsamp = cor(sampleMat)

    }

    # Low model approximation error
    if(err == "Low") {
```

```
    # Generate population factor model
    simOut = simFA(Model      = list(NFac          = nfac,
                                     NItemPerFac   = ifac,
                                     Model         = modtype),
                   Loadings   = list(FacLoadDist   = "sequential",
                                     FacLoadRange  = lvec,
                                     h2            = hvec),
                   Phi        = list(PhiType       = "user",
                                     UserPhi       = simphi),
                   ModelError = list(ModelError    = TRUE,
                                     NMinorFac     = 150,
                                     ModelErrorType = "U",
                                     ModelErrorVar = 0.1,
                                     PrintW        = FALSE),
                   MonteCarlo = list(NSamples      = 1,
                                     SampleSize    = sampsize,
                                     Raw           = TRUE),
                   Seed       = iSeed)

    # Save sample correlation matrix
    sampleMat = simOut$Monte$MCDataME[[1]]
    Rsamp = cor(sampleMat)

  }

  # Run factor analysis with Oblimin rotation
  iHey = if(iTreat == "CLS") TRUE else FALSE
  fout = faMain(R = Rsamp, numFactors = nfac,
                facMethod = fmethod,
                rotate = "oblimin",
                targetMatrix = simOut$loadings,
                faControl = list(treatHeywood = iHey),
                rotateControl = list(numberStarts = 1))

  # Full-model RMSE
  fullrmse[iTrial] = rmsd(A = simOut$loadings,
                          B = fout$loadings,
                          Symmetric = FALSE)

  # RMSE for salient loadings
  salient = sign(simOut$loadings)
  popsal = matrix(simOut$loadings[which(simOut$loadings != 0)],
                  ncol = 3, nrow = 4)
  sampsal = matrix(fout$loadings[which((salient*fout$loadings) != 0)],
                   ncol = 3, nrow = 4)
  salrmse[iTrial] = rmsd(A = popsal, B = sampsal, Symmetric = FALSE)

  # Compute sample factor determinacy
  detdist[[iTrial]] = fout$facIndeterminacy
  detbias[[iTrial]] = detdist[[iTrial]] - simOut$Scores$FacInd

  # Go to next trial
  cat("\nCompleted", iTreat, "trial", iTrial)
```

```
    } # end for iTrial in 1:trials

    # Determinacy distributions and bias
    detdist = array(unlist(detdist), c(nfac, 1, trials))
    detbias = array(unlist(detbias), c(nfac, 1, trials))

    # Save results for treatment method
    assign(modname, list(fullRMSE   = fullrmse,
                         salRMSE    = salrmse,
                         detDist    = detdist,
                         detBias    = detbias))

  } # end for iTreat in heytreat

  # List output
  list(fullRMSE = list(HeyPAF    = results.heyPAF$fullRMSE,
                       CLS       = results.cls$fullRMSE,
                       RLS       = results.refaLS$fullRMSE,
                       RML       = results.refaML$fullRMSE,
                       NHeyPAF   = results.nheyPAF$fullRMSE,
                       NHeyRLS   = results.nheyRLS$fullRMSE,
                       NHeyRML   = results.nheyRML$fullRMSE),
       salRMSE  = list(HeyPAF    = results.heyPAF$salRMSE,
                       CLS       = results.cls$salRMSE,
                       RLS       = results.refaLS$salRMSE,
                       RML       = results.refaML$salRMSE,
                       NHeyPAF   = results.nheyPAF$salRMSE,
                       NHeyRLS   = results.nheyRLS$salRMSE,
                       NHeyRML   = results.nheyRML$salRMSE),
       detDist  = list(HeyPAF    = results.heyPAF$detDist,
                       CLS       = results.cls$detDist,
                       RLS       = results.refaLS$detDist,
                       RML       = results.refaML$detDist,
                       NHeyPAF   = results.nheyPAF$detDist,
                       NHeyRLS   = results.nheyRLS$detDist,
                       NHeyRML   = results.nheyRML$detDist),
       detBias  = list(HeyPAF    = results.heyPAF$detBias,
                       CLS       = results.cls$detBias,
                       RLS       = results.refaLS$detBias,
                       RML       = results.refaML$detBias,
                       NHeyPAF   = results.nheyPAF$detBias,
                       NHeyRLS   = results.nheyRLS$detBias,
                       NHeyRML   = results.nheyRML$detBias))

} # end poc.refa function

# Load saved seed information
load("./HeywoodEffectSeeds.RData")

# Pattern B: Sequential Loadings
seq.reg = poc.refa(nfac = 3,
                   ifac = 4,
                   lvec = c(.8, .3),
                   hvec = NULL,
                   fcor = 0,
                   sampsize = 150,
                   err = "None",
                   seedlistH = seeds.seq.hey$seeds,
                   seedlistNH = seeds.seq.nohey$seeds,
                   trials = ntrials)
seq.reg.err = poc.refa(nfac = 3,
                       ifac = 4,
                       lvec = c(.8, .3),
                       hvec = NULL,
                       fcor = 0,
                       sampsize = 150,
                       err = "Low",
                       seedlistH = seeds.seq.hey.err$seeds,
                       seedlistNH = seeds.seq.nohey.err$seeds,
                       trials = ntrials)

# Pattern C: One Strong Loading
sload.reg = poc.refa(nfac = 3,
                     ifac = 4,
                     lvec = c(.8, .5),
                     hvec = rep(c(.64, .25, .25, .25), 3),
                     fcor = 0,
                     sampsize = 150,
                     err = "None",
                     seedlistH = seeds.sload.hey$seeds,
                     seedlistNH = seeds.sload.nohey$seeds,
                     trials = ntrials)
sload.reg.err = poc.refa(nfac = 3,
                         ifac = 4,
                         lvec = c(.8, .5),
                         hvec = rep(c(.64, .25, .25, .25), 3),
                         fcor = 0,
                         sampsize = 150,
                         err = "Low",
                         seedlistH = seeds.sload.hey.err$seeds,
                         seedlistNH = seeds.sload.nohey.err$seeds,
                         trials = ntrials)

# Pattern D: One Weak Factor
oneweak.reg = poc.refa(nfac = 3,
                       ifac = 4,
                       lvec = c(.8, .3),
                       hvec = c(rep(.64, 8), rep(.09, 4)),
                       fcor = 0,
                       sampsize = 150,
                       err = "None",
                       seedlistH = seeds.oneweak.hey$seeds,
                       seedlistNH = seeds.oneweak.nohey$seeds,
                       trials = ntrials)
oneweak.reg.err = poc.refa(nfac = 3,
                           ifac = 4,
                           lvec = c(.8, .3),
                           hvec = c(rep(.64, 8), rep(.09, 4)),
                           fcor = 0,
                           sampsize = 150,
                           err = "Low",
                           seedlistH = seeds.oneweak.hey.err$seeds,
                           seedlistNH = seeds.oneweak.nohey.err$seeds,
                           trials = ntrials)

# Pattern E: Two Weak Factors
twoweak.reg = poc.refa(nfac = 3,
                       ifac = 4,
                       lvec = c(.8, .3),
                       hvec = c(rep(.64, 4), rep(.09, 8)),
                       fcor = 0,
                       sampsize = 150,
                       err = "None",
                       seedlistH = seeds.twoweak.hey$seeds,
                       seedlistNH = seeds.twoweak.nohey$seeds,
                       trials = ntrials)
twoweak.reg.err = poc.refa(nfac = 3,
                           ifac = 4,
                           lvec = c(.8, .3),
                           hvec = c(rep(.64, 4), rep(.09, 8)),
                           fcor = 0,
                           sampsize = 150,
                           err = "Low",
                           seedlistH = seeds.twoweak.hey.err$seeds,
                           seedlistNH = seeds.twoweak.nohey.err$seeds,
                           trials = ntrials)

# Pattern F: Three Weak Factors
threeweak.reg = poc.refa(nfac = 3,
                         ifac = 4,
                         lvec = c(.3, .3),
                         hvec = rep(.09, 12),
                         fcor = 0,
                         sampsize = 150,
                         err = "None",
                         seedlistH = seeds.threeweak.hey$seeds,
                         seedlistNH = seeds.threeweak.nohey$seeds,
                         trials = ntrials)
threeweak.reg.err = poc.refa(nfac = 3,
                             ifac = 4,
                             lvec = c(.3, .3),
                             hvec = rep(.09, 12),
                             fcor = 0,
                             sampsize = 150,
                             err = "Low",
                             seedlistH = seeds.threeweak.hey.err$seeds,
                             seedlistNH = seeds.threeweak.nohey.err$seeds,
                             trials = ntrials)
```

Table 11

```
# Print table of RMSE values for different extraction methods
table11.output = data.frame(

      # Loading pattern
      c("B", "C", "D", "E", "F"),

      # Non-Heywood PAF: Full-model
      c(mean(seq.reg$fullRMSE$NHeyPAF),
        mean(sload.reg$fullRMSE$NHeyPAF),
        mean(oneweak.reg$fullRMSE$NHeyPAF),
        mean(twoweak.reg$fullRMSE$NHeyPAF),
        mean(threeweak.reg$fullRMSE$NHeyPAF)),

      # Non-Heywood PAF: Salient loadings
      c(mean(seq.reg$salRMSE$NHeyPAF),
        mean(sload.reg$salRMSE$NHeyPAF),
        mean(oneweak.reg$salRMSE$NHeyPAF),
        mean(twoweak.reg$salRMSE$NHeyPAF),
        mean(threeweak.reg$salRMSE$NHeyPAF)),

      # Non-Heywood RCFA-ULS: Full-model
      c(mean(seq.reg$fullRMSE$NHeyRLS),
        mean(sload.reg$fullRMSE$NHeyRLS),
        mean(oneweak.reg$fullRMSE$NHeyRLS),
        mean(twoweak.reg$fullRMSE$NHeyRLS),
        mean(threeweak.reg$fullRMSE$NHeyRLS)),

      # Non-Heywood RCFA-ULS: Salient loadings
      c(mean(seq.reg$salRMSE$NHeyRLS),
        mean(sload.reg$salRMSE$NHeyRLS),
        mean(oneweak.reg$salRMSE$NHeyRLS),
        mean(twoweak.reg$salRMSE$NHeyRLS),
        mean(threeweak.reg$salRMSE$NHeyRLS)),

      # Non-Heywood RCFA-ML: Full-model
      c(mean(seq.reg$fullRMSE$NHeyRML),
        mean(sload.reg$fullRMSE$NHeyRML),
        mean(oneweak.reg$fullRMSE$NHeyRML),
        mean(twoweak.reg$fullRMSE$NHeyRML),
        mean(threeweak.reg$fullRMSE$NHeyRML)),

      # Non-Heywood RCFA-ML: Salient loadings
      c(mean(seq.reg$salRMSE$NHeyRML),
        mean(sload.reg$salRMSE$NHeyRML),
        mean(oneweak.reg$salRMSE$NHeyRML),
        mean(twoweak.reg$salRMSE$NHeyRML),
        mean(threeweak.reg$salRMSE$NHeyRML)),

      # Heywood PAF: Full-model
      c(mean(seq.reg$fullRMSE$HeyPAF),
        mean(sload.reg$fullRMSE$HeyPAF),
        mean(oneweak.reg$fullRMSE$HeyPAF),
        mean(twoweak.reg$fullRMSE$HeyPAF),
        mean(threeweak.reg$fullRMSE$HeyPAF)),

      # Heywood PAF: Salient loadings
      c(mean(seq.reg$salRMSE$HeyPAF),
        mean(sload.reg$salRMSE$HeyPAF),
        mean(oneweak.reg$salRMSE$HeyPAF),
        mean(twoweak.reg$salRMSE$HeyPAF),
        mean(threeweak.reg$salRMSE$HeyPAF)),

      # CLS: Full-model
      c(mean(seq.reg$fullRMSE$CLS),
        mean(sload.reg$fullRMSE$CLS),
        mean(oneweak.reg$fullRMSE$CLS),
        mean(twoweak.reg$fullRMSE$CLS),
        mean(threeweak.reg$fullRMSE$CLS)),

      # CLS: Salient loadings
      c(mean(seq.reg$salRMSE$CLS),
        mean(sload.reg$salRMSE$CLS),
        mean(oneweak.reg$salRMSE$CLS),
        mean(twoweak.reg$salRMSE$CLS),
        mean(threeweak.reg$salRMSE$CLS)),

      # RCFA-ULS: Full-model
      c(mean(seq.reg$fullRMSE$RLS),
        mean(sload.reg$fullRMSE$RLS),
        mean(oneweak.reg$fullRMSE$RLS),
        mean(twoweak.reg$fullRMSE$RLS),
        mean(threeweak.reg$fullRMSE$RLS)),

      # RCFA-ULS: Salient loadings
      c(mean(seq.reg$salRMSE$RLS),
        mean(sload.reg$salRMSE$RLS),
        mean(oneweak.reg$salRMSE$RLS),
        mean(twoweak.reg$salRMSE$RLS),
        mean(threeweak.reg$salRMSE$RLS)),

      # RCFA-ML: Full-model
      c(mean(seq.reg$fullRMSE$RML),
        mean(sload.reg$fullRMSE$RML),
        mean(oneweak.reg$fullRMSE$RML),
        mean(twoweak.reg$fullRMSE$RML),
        mean(threeweak.reg$fullRMSE$RML)),

      # RCFA-ML: Salient loadings
      c(mean(seq.reg$salRMSE$RML),
        mean(sload.reg$salRMSE$RML),
        mean(oneweak.reg$salRMSE$RML),
        mean(twoweak.reg$salRMSE$RML),
        mean(threeweak.reg$salRMSE$RML))
```

```r
)
table11.output[,-1] = round(table11.output[,-1], 2)
names(table11.output) = c("Pattern",
                          "Non-Heywood PAF Full",
                          "Non-Heywood PAF Salient",
                          "Non-Heywood RCFA-ULS Full",
                          "Non-Heywood RCFA-ULS Salient",
                          "Non-Heywood RCFA-ML Full",
                          "Non-Heywood RCFA-ML Salient",
                          "Heywood PAF Full",
                          "Heywood PAF Salient",
                          "CLS Full",
                          "CLS Salient",
                          "RCFA-ULS Full",
                          "RCFA-ULS Salient",
                          "RCFA-ML Full",
                          "RCFA-ML Salient")
print(table11.output)
```

Table S11

```r
# Print table of RMSE values for different extraction methods
# among models with low levels of approximation error
tableS11.output = data.frame(

  # Loading pattern
  c("B", "C", "D", "E", "F"),

  # Non-Heywood PAF: Full-model
  c(mean(seq.reg.err$fullRMSE$NHeyPAF),
    mean(sload.reg.err$fullRMSE$NHeyPAF),
    mean(oneweak.reg.err$fullRMSE$NHeyPAF),
    mean(twoweak.reg.err$fullRMSE$NHeyPAF),
    mean(threeweak.reg.err$fullRMSE$NHeyPAF)),

  # Non-Heywood PAF: Salient loadings
  c(mean(seq.reg.err$salRMSE$NHeyPAF),
    mean(sload.reg.err$salRMSE$NHeyPAF),
    mean(oneweak.reg.err$salRMSE$NHeyPAF),
    mean(twoweak.reg.err$salRMSE$NHeyPAF),
    mean(threeweak.reg.err$salRMSE$NHeyPAF)),

  # Non-Heywood RCFA-ULS: Full-model
  c(mean(seq.reg.err$fullRMSE$NHeyRLS),
    mean(sload.reg.err$fullRMSE$NHeyRLS),
    mean(oneweak.reg.err$fullRMSE$NHeyRLS),
    mean(twoweak.reg.err$fullRMSE$NHeyRLS),
    mean(threeweak.reg.err$fullRMSE$NHeyRLS)),

  # Non-Heywood RCFA-ULS: Salient loadings
  c(mean(seq.reg.err$salRMSE$NHeyRLS),
    mean(sload.reg.err$salRMSE$NHeyRLS),
    mean(oneweak.reg.err$salRMSE$NHeyRLS),
    mean(twoweak.reg.err$salRMSE$NHeyRLS),
    mean(threeweak.reg.err$salRMSE$NHeyRLS)),

  # Non-Heywood RCFA-ML: Full-model
  c(mean(seq.reg.err$fullRMSE$NHeyRML),
    mean(sload.reg.err$fullRMSE$NHeyRML),
    mean(oneweak.reg.err$fullRMSE$NHeyRML),
    mean(twoweak.reg.err$fullRMSE$NHeyRML),
    mean(threeweak.reg.err$fullRMSE$NHeyRML)),

  # Non-Heywood RCFA-ML: Salient loadings
  c(mean(seq.reg.err$salRMSE$NHeyRML),
    mean(sload.reg.err$salRMSE$NHeyRML),
    mean(oneweak.reg.err$salRMSE$NHeyRML),
    mean(twoweak.reg.err$salRMSE$NHeyRML),
    mean(threeweak.reg.err$salRMSE$NHeyRML)),

  # Heywood PAF: Full-model
  c(mean(seq.reg.err$fullRMSE$HeyPAF),
    mean(sload.reg.err$fullRMSE$HeyPAF),
    mean(oneweak.reg.err$fullRMSE$HeyPAF),
    mean(twoweak.reg.err$fullRMSE$HeyPAF),
    mean(threeweak.reg.err$fullRMSE$HeyPAF)),

  # Heywood PAF: Salient loadings
  c(mean(seq.reg.err$salRMSE$HeyPAF),
    mean(sload.reg.err$salRMSE$HeyPAF),
    mean(oneweak.reg.err$salRMSE$HeyPAF),
    mean(twoweak.reg.err$salRMSE$HeyPAF),
    mean(threeweak.reg.err$salRMSE$HeyPAF)),

  # CLS: Full-model
  c(mean(seq.reg.err$fullRMSE$CLS),
    mean(sload.reg.err$fullRMSE$CLS),
    mean(oneweak.reg.err$fullRMSE$CLS),
    mean(twoweak.reg.err$fullRMSE$CLS),
    mean(threeweak.reg.err$fullRMSE$CLS)),

  # CLS: Salient loadings
  c(mean(seq.reg.err$salRMSE$CLS),
    mean(sload.reg.err$salRMSE$CLS),
    mean(oneweak.reg.err$salRMSE$CLS),
    mean(twoweak.reg.err$salRMSE$CLS),
    mean(threeweak.reg.err$salRMSE$CLS)),

  # RCFA-ULS: Full-model
  c(mean(seq.reg.err$fullRMSE$RLS),
    mean(sload.reg.err$fullRMSE$RLS),
    mean(oneweak.reg.err$fullRMSE$RLS),
    mean(twoweak.reg.err$fullRMSE$RLS),
    mean(threeweak.reg.err$fullRMSE$RLS)),
```

```r
  # RCFA-ULS: Salient loadings
  c(mean(seq.reg.err$salRMSE$RLS),
    mean(sload.reg.err$salRMSE$RLS),
    mean(oneweak.reg.err$salRMSE$RLS),
    mean(twoweak.reg.err$salRMSE$RLS),
    mean(threeweak.reg.err$salRMSE$RLS)),

  # RCFA-ML: Full-model
  c(mean(seq.reg.err$fullRMSE$RML),
    mean(sload.reg.err$fullRMSE$RML),
    mean(oneweak.reg.err$fullRMSE$RML),
    mean(twoweak.reg.err$fullRMSE$RML),
    mean(threeweak.reg.err$fullRMSE$RML)),

  # RCFA-ML: Salient loadings
  c(mean(seq.reg.err$salRMSE$RML),
    mean(sload.reg.err$salRMSE$RML),
    mean(oneweak.reg.err$salRMSE$RML),
    mean(twoweak.reg.err$salRMSE$RML),
    mean(threeweak.reg.err$salRMSE$RML))

)
tableS11.output[,-1] = round(tableS11.output[,-1], 2)
names(tableS11.output) = c("Pattern",
                           "Non-Heywood PAF Full",
                           "Non-Heywood PAF Salient",
                           "Non-Heywood RCFA-ULS Full",
                           "Non-Heywood RCFA-ULS Salient",
                           "Non-Heywood RCFA-ML Full",
                           "Non-Heywood RCFA-ML Salient",
                           "Heywood PAF Full",
                           "Heywood PAF Salient",
                           "CLS Full",
                           "CLS Salient",
                           "RCFA-ULS Full",
                           "RCFA-ULS Salient",
                           "RCFA-ML Full",
                           "RCFA-ML Salient")
print(tableS11.output)
```

Table 12

```r
# Print table of factor score determinacy bias for
# four extraction methods
table12.output = data.frame(

  # Loading pattern and factor
  c("B: f1", "B: f2", "B: f3",
    "C: f1", "C: f2", "C: f3",
    "D: f1", "D: f2", "D: f3",
    "E: f1", "E: f2", "E: f3",
    "F: f1", "F: f2", "F: f3"),
  # Non-Heywood PAF
  c(apply(seq.reg$detBias$NHeyPAF, 1:2, mean),
    apply(sload.reg$detBias$NHeyPAF, 1:2, mean),
    apply(oneweak.reg$detBias$NHeyPAF, 1:2, mean),
    apply(twoweak.reg$detBias$NHeyPAF, 1:2, mean),
    apply(threeweak.reg$detBias$NHeyPAF, 1:2, mean)),

  # Non-Heywood RCFA-ULS
  c(apply(seq.reg$detBias$NHeyRLS, 1:2, mean),
    apply(sload.reg$detBias$NHeyRLS, 1:2, mean),
    apply(oneweak.reg$detBias$NHeyRLS, 1:2, mean),
    apply(twoweak.reg$detBias$NHeyRLS, 1:2, mean),
    apply(threeweak.reg$detBias$NHeyRLS, 1:2, mean)),

  # Non-Heywood RCFA-ML
  c(apply(seq.reg$detBias$NHeyRML, 1:2, mean),
    apply(sload.reg$detBias$NHeyRML, 1:2, mean),
    apply(oneweak.reg$detBias$NHeyRML, 1:2, mean),
    apply(twoweak.reg$detBias$NHeyRML, 1:2, mean),
    apply(threeweak.reg$detBias$NHeyRML, 1:2, mean)),

  # Heywood PAF
  c(apply(seq.reg$detBias$HeyPAF, 1:2, mean),
    apply(sload.reg$detBias$HeyPAF, 1:2, mean),
    apply(oneweak.reg$detBias$HeyPAF, 1:2, mean),
    apply(twoweak.reg$detBias$HeyPAF, 1:2, mean),
    apply(threeweak.reg$detBias$HeyPAF, 1:2, mean)),

  # CLS
  c(apply(seq.reg$detBias$CLS, 1:2, mean),
    apply(sload.reg$detBias$CLS, 1:2, mean),
    apply(oneweak.reg$detBias$CLS, 1:2, mean),
    apply(twoweak.reg$detBias$CLS, 1:2, mean),
    apply(threeweak.reg$detBias$CLS, 1:2, mean)),

  # RCFA-ULS
  c(apply(seq.reg$detBias$RLS, 1:2, mean),
    apply(sload.reg$detBias$RLS, 1:2, mean),
    apply(oneweak.reg$detBias$RLS, 1:2, mean),
    apply(twoweak.reg$detBias$RLS, 1:2, mean),
    apply(threeweak.reg$detBias$RLS, 1:2, mean)),

  # RCFA-ML
  c(apply(seq.reg$detBias$RML, 1:2, mean),
    apply(sload.reg$detBias$RML, 1:2, mean),
    apply(oneweak.reg$detBias$RML, 1:2, mean),
    apply(twoweak.reg$detBias$RML, 1:2, mean),
    apply(threeweak.reg$detBias$RML, 1:2, mean))

)
names(table12.output) = c("Loading Pattern and Factor",
                          "Non-Heywood PAF",
                          "Non-Heywood RCFA-ULS",
                          "Non-Heywood RCFA-ML",
                          "Heywood PAF",
                          "CLS",
                          "RCFA-ULS",
                          "RCFA-ML")
table12.output[,-1] = round(table12.output[,-1], 2)
print(table12.output)
```

Table S12

```
# Print table of factor score determinacy bias for
# four extraction methods
# among models with low levels of approximation error
tableS12.output = data.frame(

  # Loading pattern and factor
  c("B: f1", "B: f2", "B: f3",
    "C: f1", "C: f2", "C: f3",
    "D: f1", "D: f2", "D: f3",
    "E: f1", "E: f2", "E: f3",
    "F: f1", "F: f2", "F: f3"),

  # Non-Heywood PAF
  c(apply(seq.reg.err$detBias$NHeyPAF, 1:2, mean),
    apply(sload.reg.err$detBias$NHeyPAF, 1:2, mean),
    apply(oneweak.reg.err$detBias$NHeyPAF, 1:2, mean),
    apply(twoweak.reg.err$detBias$NHeyPAF, 1:2, mean),
    apply(threeweak.reg.err$detBias$NHeyPAF, 1:2, mean)),

  # Non-Heywood RCFA-ULS
  c(apply(seq.reg.err$detBias$NHeyRLS, 1:2, mean),
    apply(sload.reg.err$detBias$NHeyRLS, 1:2, mean),
    apply(oneweak.reg.err$detBias$NHeyRLS, 1:2, mean),
    apply(twoweak.reg.err$detBias$NHeyRLS, 1:2, mean),
    apply(threeweak.reg.err$detBias$NHeyRLS, 1:2, mean)),

  # Non-Heywood RCFA-ML
  c(apply(seq.reg.err$detBias$NHeyRML, 1:2, mean),
    apply(sload.reg.err$detBias$NHeyRML, 1:2, mean),
    apply(oneweak.reg.err$detBias$NHeyRML, 1:2, mean),
    apply(twoweak.reg.err$detBias$NHeyRML, 1:2, mean),
    apply(threeweak.reg.err$detBias$NHeyRML, 1:2, mean)),

  # Heywood PAF
  c(apply(seq.reg.err$detBias$HeyPAF, 1:2, mean),
    apply(sload.reg.err$detBias$HeyPAF, 1:2, mean),
    apply(oneweak.reg.err$detBias$HeyPAF, 1:2, mean),
    apply(twoweak.reg.err$detBias$HeyPAF, 1:2, mean),
    apply(threeweak.reg.err$detBias$HeyPAF, 1:2, mean)),

  # CLS
  c(apply(seq.reg.err$detBias$CLS, 1:2, mean),
    apply(sload.reg.err$detBias$CLS, 1:2, mean),
    apply(oneweak.reg.err$detBias$CLS, 1:2, mean),
    apply(twoweak.reg.err$detBias$CLS, 1:2, mean),
    apply(threeweak.reg.err$detBias$CLS, 1:2, mean)),

  # RCFA-ULS
  c(apply(seq.reg.err$detBias$RLS, 1:2, mean),
    apply(sload.reg.err$detBias$RLS, 1:2, mean),
    apply(oneweak.reg.err$detBias$RLS, 1:2, mean),
    apply(twoweak.reg.err$detBias$RLS, 1:2, mean),
    apply(threeweak.reg.err$detBias$RLS, 1:2, mean)),

  # RCFA-ML
  c(apply(seq.reg.err$detBias$RML, 1:2, mean),
    apply(sload.reg.err$detBias$RML, 1:2, mean),
    apply(oneweak.reg.err$detBias$RML, 1:2, mean),
    apply(twoweak.reg.err$detBias$RML, 1:2, mean),
    apply(threeweak.reg.err$detBias$RML, 1:2, mean))

)
names(tableS12.output) = c("Loading Pattern and Factor",
                           "Non-Heywood PAF",
                           "Non-Heywood RCFA-ULS",
                           "Non-Heywood RCFA-ML",
                           "Heywood PAF",
                           "CLS",
                           "RCFA-ULS",
                           "RCFA-ML")
tableS12.output[,-1] = round(tableS12.output[,-1], 2)
print(tableS12.output)
```