Simulación Factorial Confirmatorio

Manuel Torres Acosta

Índice

Objetivo

La tarea actual consiste en utilizar simulaciones para evaluar el funcionamiento del análisis factorial confirmatorio en una serie de circunstancias.

En primer lugar, debemos simular una estructura factorial determinada y evaluar el ajuste del modelo correcto frente al de otros modelos incorrectos. A continuación debemos realizar un análisis factorial multi grupo a partir de una base de datos dada para evaluar si la estructura factorial está presente en todos los grupos de la misma forma (equivalencia métrica).

Un único grupo

El primer apartado de la práctica consiste en desarrollar una simulación que involucre al análisis factorial confirmatorio sin tener en cuenta ningún tipo de grupo. Para éste apartado, realizaremos una simulación en la que iremos introduciendo una cantidad progresivamente mayor de errores de especificación. Entonces evaluaremos cómo evolucionan distintos indicadores de ajuste.

Empezaremos ajustando una serie de modelos sin errores de especificación para obtener un indicador del ajuste de los modelos correctos. A continuación, cambiaremos un ítem de factor, luego dos, tres… etc. Para ello tendremos que desarrollar algunas funciones que nos permitan manipular los modelos.

```
manipulaModelo <- function(correct,
                           nErrors){
  distorted <- correct
  nFactors <-  length(correct)
  factorNames <- names(correct)
  registry <- c() #Para registrar los cambios hechos

  for(i in 1:nErrors){
    if(nErrors == 0) break
    #Introducimos errores de especificacion cambiando items de
    #factor
    rFact <- sample.int(nFactors, 1) #Escogemos un factor al azar
    #Escogemos un item al azar del factor
    rItem <- sample.int(length(distorted[[rFact]]), 1)

    #Si hemos seleccionado un item ya seleccionado, escogemos
    #otro hasta encontrar uno que no haya sido seleccionado
    while(distorted[[rFact]][rItem] %in% registry){
       rFact <- sample.int(3, 1)
       rItem <- sample.int(length(distorted[[rFact]]), 1)
    }

    #Escogemos el factor de destino al azar
    dest <- sample(setdiff(1:nFactors, rFact), 1)


    #Metemos al item en el otro factor
    distorted[[dest]] = c(distorted[[dest]],
                          distorted[[rFact]][rItem])

    #Añadimos el item al registro para no volver a intentar
    #moverlo de sitio
```

```
    registry <- c(registry, distorted[[rFact]][rItem])

    #Eliminamos el item del factor original
    distorted[[rFact]] = setdiff(distorted[[rFact]],
                                 distorted[[rFact]][rItem])
  }
  return(distorted)
}

#Otra funcion que recibe una lista con los items que componen
#cada factor y la convierte en un modelo (string)
modeloString <- function(modelo){
  modelString <- unlist(lapply(modelo,
                               function(x){
                                  paste(x, collapse = " + ")
                                }))
  modelString <- paste(names(modelString), " =~ ",
                       modelString, sep = "")
  modelString <- paste(modelString, collapse = "\n")

  return(modelString)
}
```

La función `manipulaModelo()` recibe como input una lista que contiene las especificaciones del modelo **correcto**. Cada elemento de ésta lista representa un factor, y está formado por un vector que contiene los nombres de los ítems que pertenecen al mismo.

La función tiene un segundo parámetro, que determina el número de errores de especificación a introducir. Si vale cero, no hace ningún cambio al modelo. Si adopta un valor mayor que cero, selecciona ése número ítems al azar y los cambia cada uno a otro de los factores (el factor de destino también se selecciona al azar para cada ítem). Durante el proceso, se garantiza que el factor de destino no sea el mismo que el de origen, y que un mismo ítem solo se cambie de factor una vez.

El resultado de ésta función si `nErrores` es mayor que cero será una lista que contiene los mismos factores que el modelo original pero con algunos ítems cambiados de factor. Sin embargo, el paquete lavaan requiere que el modelo esté especificado en formato de texto.

Para ello, creamos una segunda función `modeloString()`, que recibe como input un modelo en forma de lista y lo devuelve como una cadena de caractéres válida para lavaan. Veamos si ambas funciones muestran el comportamiento esperado.

```
set.seed(123)
#Simulamos una estructura con varios factores
nSujetos = 1000
nItems = 5
```

```r
Datos <- list(F1 = simDF(nSujetos, nItems, lambda = 0.7))

Datos$F2 <- simDF(nSujetos, nItems, lambda = 0.7)
Datos$F3 <- simDF(nSujetos, nItems, lambda = 0.7)
Datos$F4 <- simDF(nSujetos, nItems, lambda = 0.7)
Datos$F5 <- simDF(nSujetos, nItems, lambda = 0.7)

Datos <- do.call(cbind, Datos)

#Creamos una lista que los items que pertenecen a cada factor
modeloCorrecto = list(
  F1 = grep("F1", colnames(Datos), value = TRUE),
  F2 = grep("F2", colnames(Datos), value = TRUE),
  F3 = grep("F3", colnames(Datos), value = TRUE),
  F4 = grep("F4", colnames(Datos), value = TRUE),
  F5 = grep("F5", colnames(Datos), value = TRUE)
)

modeloCorrecto
```

```
## $F1
## [1] "F1.V1" "F1.V2" "F1.V3" "F1.V4" "F1.V5"
##
## $F2
## [1] "F2.V1" "F2.V2" "F2.V3" "F2.V4" "F2.V5"
##
## $F3
## [1] "F3.V1" "F3.V2" "F3.V3" "F3.V4" "F3.V5"
##
## $F4
## [1] "F4.V1" "F4.V2" "F4.V3" "F4.V4" "F4.V5"
##
## $F5
## [1] "F5.V1" "F5.V2" "F5.V3" "F5.V4" "F5.V5"
```

```
#Añadimos un error de especificación
modelo <- manipulaModelo(modeloCorrecto, 1)
rm(.Random.seed)
modelo
```

```
## $F1
## [1] "F1.V1" "F1.V2" "F1.V3" "F1.V4" "F1.V5"
##
## $F2
## [1] "F2.V1" "F2.V2" "F2.V3" "F2.V5"
##
## $F3
## [1] "F3.V1" "F3.V2" "F3.V3" "F3.V4" "F3.V5" "F2.V4"
##
## $F4
## [1] "F4.V1" "F4.V2" "F4.V3" "F4.V4" "F4.V5"
##
## $F5
## [1] "F5.V1" "F5.V2" "F5.V3" "F5.V4" "F5.V5"
```

Podemos ver que un ítem del segundo factor ha pasado al tercero, lo que implica que el modelo contiene un error de especificación. Podemos introducir los que queramos. A continuación, utilizamos la otra función para convertir ésta lista a un modelo que pueda aceptar lavaan (texto).

```
modelo <- modeloString(modelo)
fitCorrecto <- cfa(modeloString(modeloCorrecto),
                   Datos, estimator = "GLS")
#Vemos que según chi cuadrado el modelo ajusta
fitMeasures(fitCorrecto)[3:5]
```

```
##       chisq          df      pvalue
## 264.6026937 265.0000000   0.4953323
```

```
fit <- cfa(modelo, Datos, estimator = "GLS")

#Al introducir el error de especificacion el modelo no ajusta
fitMeasures(fit)[3:5]
```

```
##    chisq       df   pvalue
## 524.5482 265.0000   0.0000
```

Podemos ver que `cfa()` acepta el input que le pasamos.

Simulación

```
simEspecif <- function(idx, seed, repet, colNames){
  #Simulamos una estructura con varios factores
  nSujetos <- 1000
  nItems <- 10

  result <- data.frame()
  alertas <- c()

  for(i in 1:repet){
    set.seed(seed[[idx]][i])
    Datos <- list(F1 = simDF(nSujetos, nItems, lambda = 0.3))

    Datos$F2 <- simDF(nSujetos, nItems, lambda = 0.3)
    Datos$F3 <- simDF(nSujetos, nItems, lambda = 0.3)

    Datos <- do.call(cbind, Datos)

    modeloCorrecto = list(
      F1 = grep("F1", colnames(Datos), value = TRUE),
      F2 = grep("F2", colnames(Datos), value = TRUE),
      F3 = grep("F3", colnames(Datos), value = TRUE)
    )

    modelo <- manipulaModelo(modeloCorrecto, nErrores[idx])
    modelo <- modeloString(modelo)

    resultIter <- c()
    #A veces los modelos pueden no converger
    try({
      fit <- cfa(modelo, Datos, estimator = "GLS",
                 control = list(iter.max = 200))
      #Capturamos posibles errores
      alerta <- lavInspect(fit, "optim")$warn.txt
      alertas <- c(alertas, alerta)

      resultIter <- c(nErrores[idx], fitMeasures(fit))
    })

    if(length(resultIter) == 0){
      resultIter <- c(nErrores[idx], rep(NA, 36))
    }
    result <- rbind(result, resultIter)
```

```
  }

  colnames(result) <- c("nErrores", colNames)


  return(
    list(
      Resultados = result,
      Alertas = alertas
    )
  )
}

nErrores <- 5:0
rep <- 10
colNames <- names(fitMeasures(fit))
set.seed(114)
semillas <- as.list(1:length(nErrores))
semillas <- lapply(semillas, function(x) sample.int(100000000,
                                                    rep))
rm(.Random.seed)

library(parallel)
Ncores <- round(detectCores()/2, 0)
if(Ncores < 1) Ncores = 1
#Creamos un cluster de procesamiento
clust <- makeCluster(Ncores, type = "FORK")

#Aplicamos la funcion en paralelo
Resultados <- parLapply(clust,
                        1:length(nErrores),
                        simEspecif,
                        semillas,
                        rep,
                        colNames)
stopCluster(clust) #Detenemos el cluster

#Ya tenemos los resultados de todas las iteraciones, es decir
#se ha ejecutado la simulacion rep veces para cada nErrores
#Ahora hacemos la media de todas las repeticiones para cada
#nErrores
Temp <- lapply(Resultados, function(x) x$Resultados)
Temp <- do.call(rbind, Temp)
```

```r
Temp <- data.frame(Temp) -> dfResultados
#Quitamos los resultados de los modelos que no ajustaron
Temp <- Temp[rowSums(is.na(Temp)) == 0, ]

resumeResultados <- function(x){
  tapply(x, Temp$nErrores, mean)
}

Resumen <- data.frame(lapply(Temp, resumeResultados))
Alertas <- unlist(lapply(Resultados, function(x) x$Alertas))
```

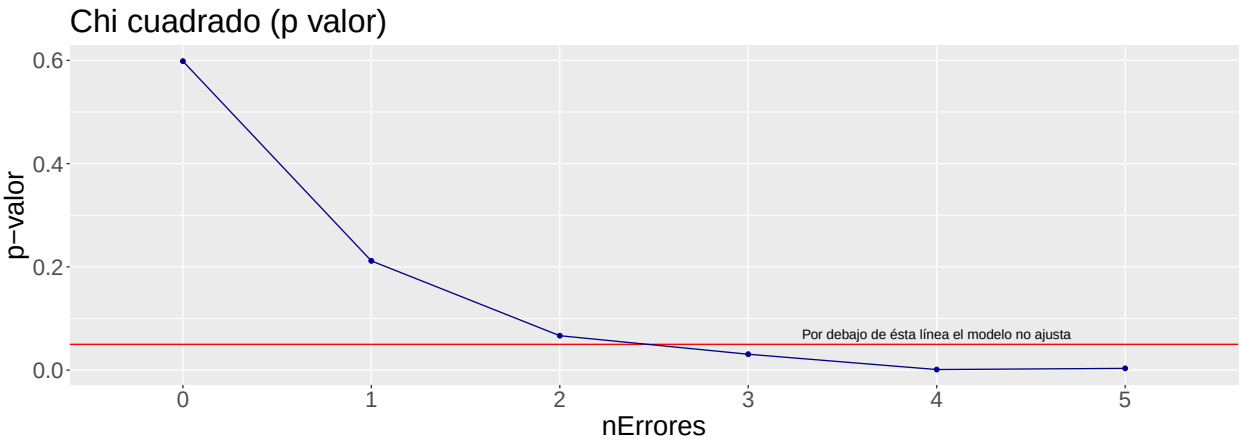
Interpretación de los resultados

A continuación, podemos representar los resultados de la simulación. A la hora de interpretar los resultados seguiremos las directrices encontradas en una de las lecturas recomendadas (Hooper, Coughlan, and Mullen 2008). No obstante, los elementos que marcan el comportamiento de un buen indicador de ajuste no son tan claros como cabría esperar.

Podríamos considerar que un buen indicador de ajuste es aquel que considera que existe ajuste en el modelo correcto, y que no existe ajuste en el momento en el que introduzcamos un error de especificación. Claro que también podría argumentarse que tal indicador es demasiado sensible.

Además, sería conveniente que nos permita distinguir entre modelos con más o menos errores (es decir, que haya variabilidad en sus valores para la simulación), y que su evolución siga un patrón regular a medida que aumenta el número de errores (siempre decreciente para los indicadores de buen ajuste y creciente para los de mal ajuste).

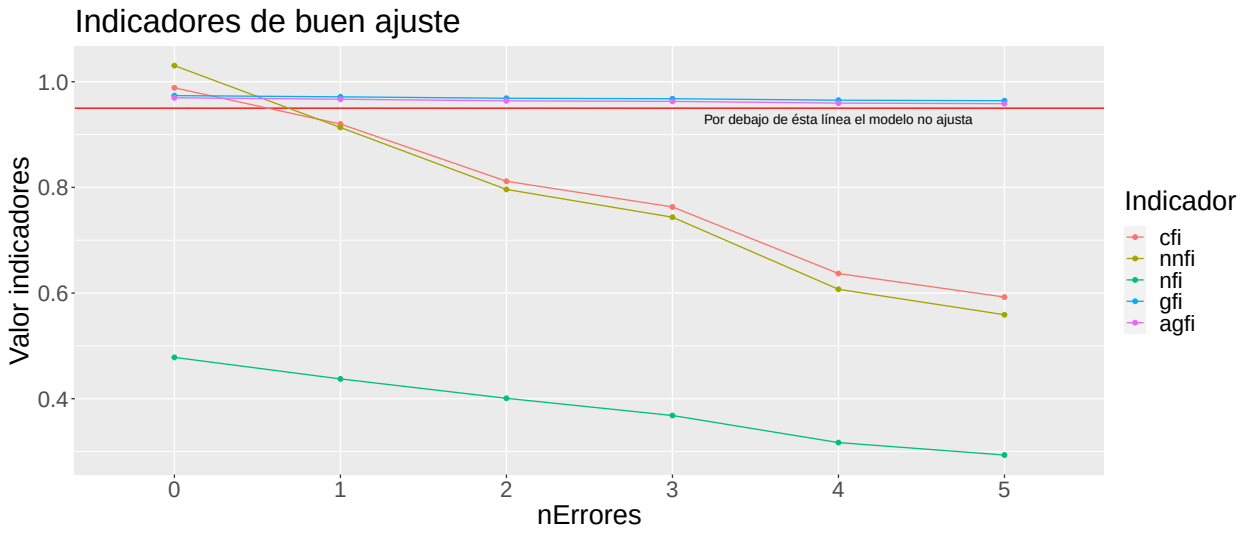
Chi cuadrado



Empezamos por el p-valor asociado al χ^2. En el gráfico podemos ver que con tres errores de especificación considera que el modelo no ajusta correctamente. Uno de los inconvenientes de χ^2 es que es sensible al tamaño muestral, de forma que para muestras grandes casi nunca indicaría que el modelo ajusta.

En nuestro caso no tiene problemas para identificar el ajuste del modelo sin errores. A juzgar por el p-valor incluso seguramente podría identificarlo con un tamaño muestral bastante mayor, aunque seguramente marcaría que no ajusta con un número menor de errores.

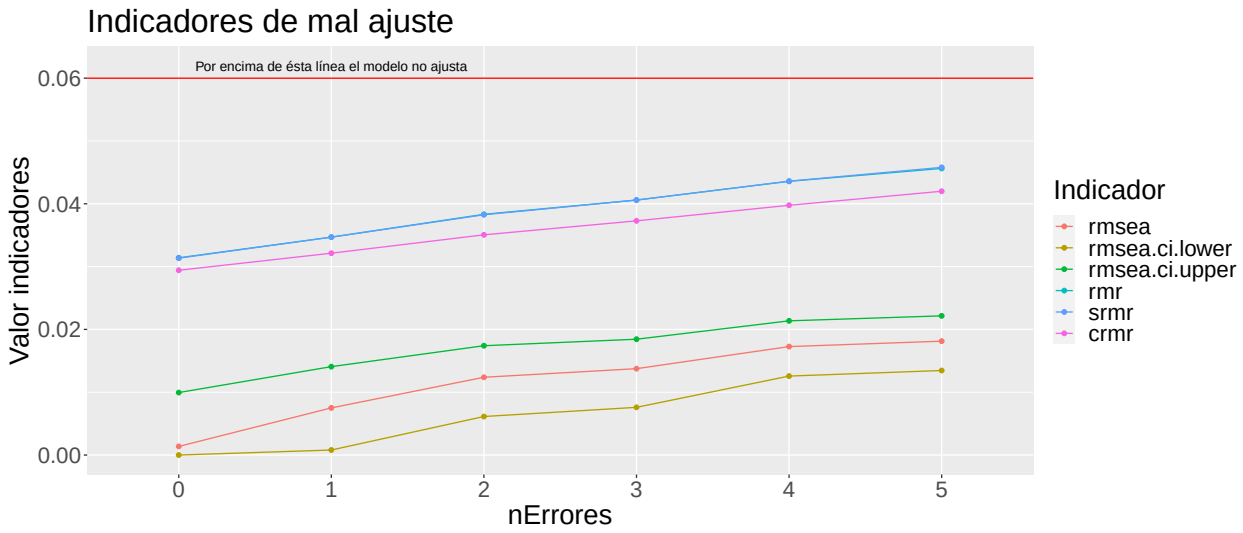
Indicadores de buen ajuste



En el gráfico observamos que tanto GFI como AGFI consideran que el modelo no pierde nunca el ajuste, ya que sus valores a penas se reducen. En cambio, según el NFI, el modelo nunca llegaría a ajustar. Según la literatura vista en clase, el NFI tiende a infraestimar el ajuste para muestras pequeñas (inferiores a 200), pero no es nuestro caso. Resulta extraño por tanto observar valores tan bajos para éste indicador.

El CFI y el NNFI (ambos derivados del NFI) por su parte indican que el modelo sin errores ajusta, pero basta un único error de especificación para que pierda el ajuste. Quizás son éstos dos indicadores los que mejor funcionan para ésta categoría ya que discriminan entre el modelo correcto y los incorrectos. Además, los valores decrecen todavía más a medida que vamos introduciendo más errores de especificación.

Indicadores de mal ajuste



En nuestra simulación parece ser que los indicadores de mal ajuste toleran un número bastante grande de errores de especificación, ya que todos indicarían que existe ajuste independientemente de la convención que utilicemos como punto crítico (el límite de aceptabilidad oscila entre 0.05 y 0.08).

Vemos que a penas hay diferencias entre el RMR y el SRMR. Sería interesante ver si ésta distancia aumenta al introducir ítems medidos en escalas diferentes, ya que el RMR depende de las unidades de medida de los ítems.

Conclusiones

A partir de los resultados los indicadores que muestran un mejor funcionamiento en la simulación son el CFI y el NFI, ambos indicadores de buen ajuste. Únicamente consideran que existe ajuste para el modelo sin errores de especificación. En el momento en el que hay un error, los valores caen por debajo de 0.95. Además sus valores siguen cayendo a medida que aumenta el número de errores, por lo que permiten discriminar entre modelos mejores o peores.

χ^2 y los indicadores de mal ajuste han mostrado una tolerancia mayor a los errores de especificación. Para el caso de χ^2 hacen falta tres errores de especificación para que dictamine que se pierde el ajuste. En el caso de los indicadores de mal ajuste, en todo momento consideran que los modelos son adecuados. Habría que añadir más errores para ver si consideran que no existe ajuste.

Cabe destacar que entre los modelos con 5 errores de especificación es frecuente que no lleguen a ajustar nunca. Por eso no exploramos escenarios con más errores.

Limitaciones

La simulación realizada presenta algunos inconvenientes. Ya comentamos que a medida que van aumentando los errores de especificación es frecuente que los modelos empiecen a dejar de converger.

Ésto se ve agravado por el hecho de que para facilitar que las simulaciones se ejecuten en un período de tiempo razonable debemos reducir el número de iteraciones máximas. De lo contrario, éstos modelos que no llegan a converger pueden tardarían demasiado en terminar, y teniendo en cuenta la cantidad de modelos que hay que ajustar, a medida que aumentamos los errores de especificación el tiempo de ejecución sería de varias horas.

De las 10 iteraciones con 5 errores de especificación, entorno a 4-6 no convergen, por lo que con 6 estimamos que será todavía más probable.

Otro elemento que limita la simulación es el hecho de que conviene que los factores tengan por lo menos 4 ítems cada uno (con menos nuevamente encontramos que los modelos no convergen). Ésto limita la cantidad de errores que podemos introducir con escalas de pocos ítems, ya que es posible que se quiten varios ítems de una escala y ninguno del resto.

Una forma de parchear éste problema sería añadir más factores, de modo que sea menos probable que se seleccionen demasiados ítems del mismo factor.

El problema es que al añadir más factores (y más ítems), nuevamente el tiempo de ejecución aumenta bastante. Con 5 escalas de 10 ítems cada una tardaba mucho más en ajustar. Para introducir más errores de especificación intentamos generar estructuras de 5 factores con 20 ítems cada uno (100 ítems en total), pero el tiempo de ejecución era demasiado alto.

Recordamos que en la simulación hemos ajustado cerca de 60 modelos (130 en el caso de la estructura con 20 ítems ya que introducimos hasta 12 errores de especificación), lo cual puede parecer que no es tanto. No obstante hay que recordar que a medida que se añaden errores de especificación los modelos tardan más en converger, de modo que el tiempo de ejecución no aumenta linealmente.

Otro elemento que sería interesante explorar es si en vez de cambiar los ítems de sitio se colocan en un factor nuevo. Con el código que hemos creado sería sencillo hacer que los ítems se coloquen en un factor nuevo en vez de cambiarlos de factor, incluso podría hacerse de forma probabilística de tal modo que el 80 % de los ítems que se mueven de sitio vayan a un factor nuevo y el otro 20 % a factores ya existentes. Como la práctica ya tiene una extensión considerable, se decidió omitir éste escenario.

Igualmente por brevedad no se ha explorado qué ocurre si los factores están correlacionados entre sí, o lo que ocurriría al utilizar datos dicotómicos, pero ambos escenarios son fáciles de simular con el código del que disponemos.

Análisis multigrupo

Normalidad y unidimensionalidad

Para la segunda parte de la práctica debemos aplicar el AFC sobre un fichero dado para comprobar si el intrumento presenta invarianza de medida entre hombres y mujeres. Antes que nada debemos comprobar si se cumple el supuesto de normalidad multivariante.

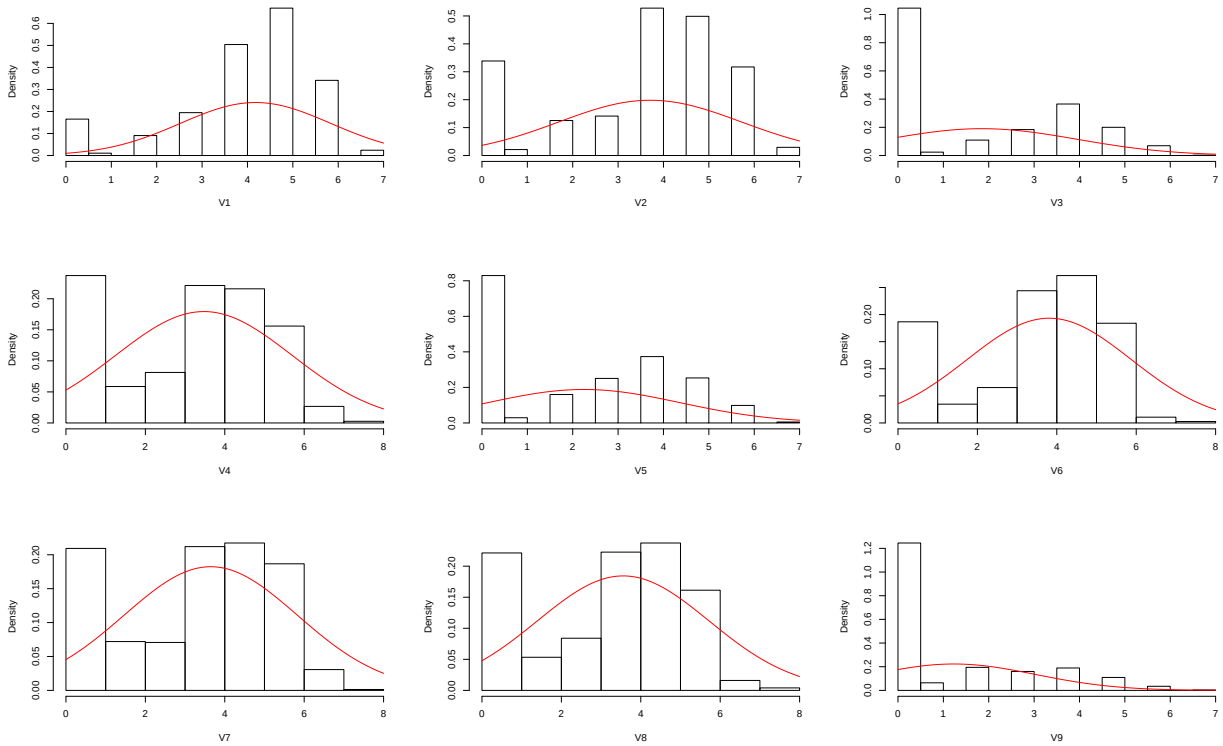
```
library(foreign)
library(psych)
Datos <- read.spss(Rutas$Datos,
                   to.data.frame = TRUE)
SoloItems <- Datos[, 1:9]
skewKurt <- describe(SoloItems)
range(skewKurt$skew) #Minimo y maximo simetria
```

```
## [1] -1.171996  1.116818
```

```
range(skewKurt$kurtosis) #Minimo y maximo apuntamiento
```

```
## [1] -1.4679805  0.9389532
```

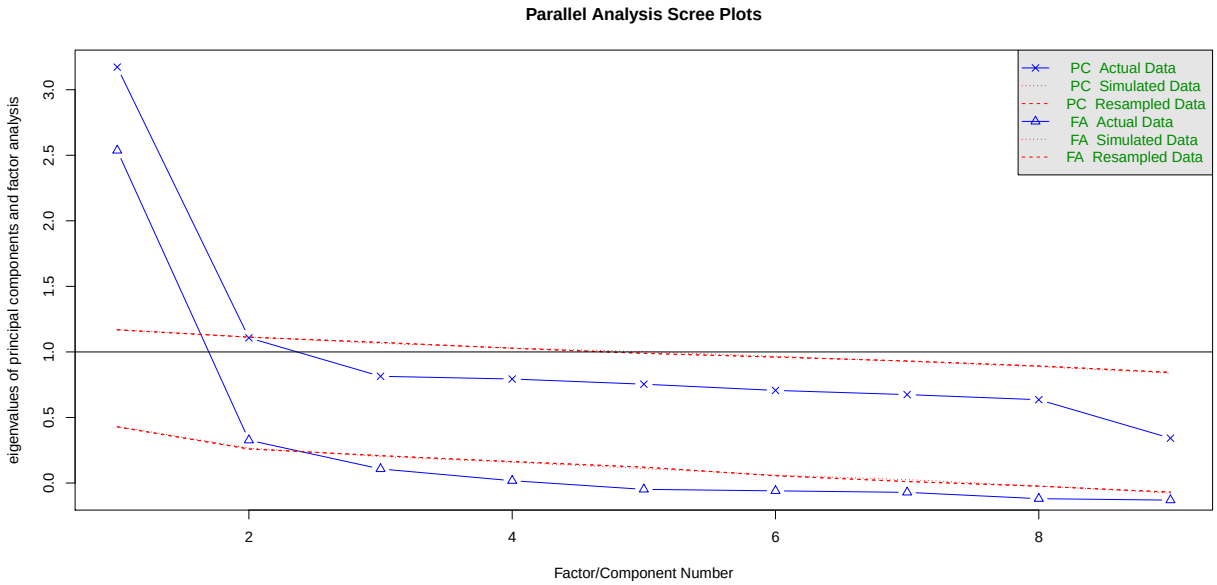
```
library(MVN)
Normalidad <- mvn(data = SoloItems, mvnTest = "mardia",
                  univariatePlot = "histogram",
                  univariateTest = "SW")
```




```
pValores <- levels(Normalidad$multivariateNormality$`p value`)
pValores <- round(as.numeric(pValores), 3)
```

Según mardia, no tendríamos problemas de kurtosis (p = 0.161), pero sí con la simetría (p < 0.001). En los gráficos podemos ver que la distribución de algunas de las variables claramente no es normal. Comprobamos si existe unidimensionalidad tal y como se indica que debería ser el caso.

```
#Debemos comprobar si existe unidimensionalidad
paralelo <- fa.parallel(SoloItems, fm = "gls")
```



Observamos que según el paralelo con análisis factorial en los datos hay dos factores, aunque componentes principales indica que efectivamente hay uno. En condiciones reales en éste punto deberíamos replantearnos si realmente conviene utilizar éste instrumento. Sin tener información sobre el contenido de los ítems vemos que en muchos ítems la opción más seleccionada es el 0 (que bien podría representar datos perdidos).

También podemos aplicar KMO para evaluar si la matriz de correlaciones reune los requisitos necesarios para aplicar sobre ella análisis factorial.

```
kmo <- KMO(SoloItems)
kmo$MSA #MSA general
```

```
## [1] 0.8369317
```

```
round(kmo$MSAi, 2) #MSA por item
```

```
##   V1   V2   V3   V4   V5   V6   V7   V8   V9
## 0.78 0.78 0.88 0.86 0.89 0.88 0.88 0.90 0.84
```

El MSA sí que muestra valores aceptables, tanto en general como para cada uno de los ítems. Aún con todo, no nos encontramos en las mejores condiciones para realizar un AFC. Para el desarrollo de la práctica continuaremos, siempre procurando usar técnicas robustas frente al incumplimiento de los supuestos.

Ajuste del modelo

```
modelo <- '
  F1 =~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9
'
fit <- cfa(modelo, SoloItems, estimator = "ULSMV")
fitMeasures(fit, c("pvalue.scaled", "cfi", "nnfi", "rmsea"))
```

```
## pvalue.scaled           cfi          nnfi         rmsea
##         0.000         0.972         0.963         0.216
```

Encontramos que incluso utilizando los métodos robustos, el modelo no ajusta.

Mejoras para el ajuste

Consultamos las posibles mejoras que podríamos hacer sobre el modelo y las vamos aplicando.

```
#Observamos las recomendaciones para mejorar el modelo
mejoras <- modindices(fit, sort. = TRUE)
mejoras <- mejoras[mejoras$mi > 8, ] #Solo mejoras signif.

#Convertimos las mejoras a texto para poder ir añadiendolas
#al modelo de forma automática
mejoras <- paste(mejoras$lhs, " ",
                 mejoras$op, " ", mejoras$rhs,
                 "\n", sep = "")

#Utilizamos un bucle para ir añadiendo las mejoras al modelo
#Registramos los resultados para poder representarlos
modelo2 <- modelo
Resultados <- data.frame()
Temp <- c("pvalue.scaled", "cfi", "nnfi", "rmsea")

for(i in 1:length(mejoras)){
  fit1 <- cfa(modelo2, SoloItems, estimator = "ULSMV")
  modelo2 <- paste(modelo2, mejoras[i])

  #Comprobamos si el modelo ajusta
  fit2 <- cfa(modelo2, SoloItems, estimator = "ULSMV")
  ajuste <- fitMeasures(fit2, Temp)
  mejoraAjuste <- anova(fit1, fit2)$`Pr(>Chisq)`[2]
  Resultados <- rbind(Resultados, c(ajuste, mejoraAjuste))
```

```
  #En cuanto chi cuadrado determine que hay ajuste, paramos
  if(ajuste["pvalue.scaled"] > 0.05) break
}

Resultados <- round(Resultados, 4)
colnames(Resultados) <- c(Temp, "incrChi")
rownames(Resultados) <- paste("Iteración", 1:nrow(Resultados))
Resultados$Modificacion <- head(gsub("\\n", "", mejoras),
                                nrow(Resultados))
```

En la tabla a continuación podemos visualizar los resultados de aplicar las mejoras. El código va incorporando las mejoras al modelo hasta que un indicador adopte un valor determinado. En éste caso hemos seleccionado el p-valor de χ^2,[1] parando cuando éste adopte un valor mayor a 0.05. También se muestra el CFI, NNFI y RMSEA. La columna incrChi muestra si el incremento en el χ^2 es estadísticamente significativo (p-valor del ANOVA).

	pvalue.scaled	cfi	nnfi	rmsea	incrChi	Modificacion
Iteración 1	0.0000	0.9786	0.9704	0.1933	0.0000	V1 ~~ V2
Iteración 2	0.0008	0.9841	0.9771	0.1699	0.0001	V7 ~~ V9
Iteración 3	0.0062	0.9870	0.9805	0.1571	0.0058	V3 ~~ V4
Iteración 4	0.0225	0.9891	0.9830	0.1467	0.0200	V4 ~~ V6
Iteración 5	0.0229	0.9894	0.9826	0.1481	0.2346	V2 ~~ V4
Iteración 6	0.0850	0.9916	0.9855	0.1353	0.0103	V4 ~~ V5

En condiciones reales tendríamos que evaluar la consistencia de los cambios que hemos hecho sobre el modelo con respecto a nuestras teorías, pero como no tenemos información sobre el instrumento simplemente las aplicamos para mejorar el ajuste y las damos por buenas.

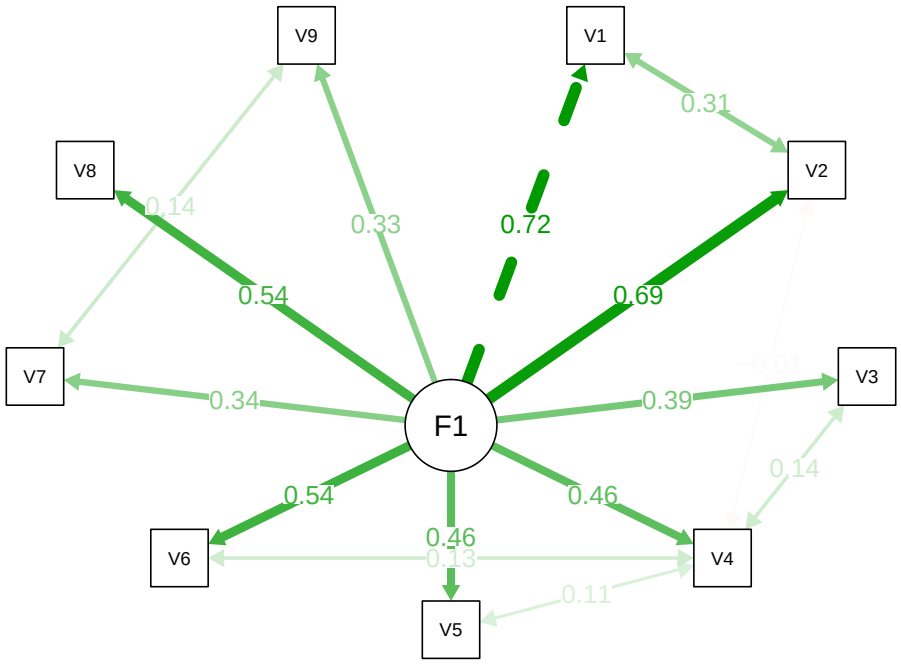
Otra opción para mejorar el ajuste con éstos datos sería aplicar el factorial sobre la matriz de **correlaciones policóricas** en vez de las tradicionales correlaciones de pearson. No obstante, éste procedimiento asume normalidad, de modo que quizás no sería adecuado para nuestros datos (añadiría más distorsiones). Además, el paquete psych no permite calcular las correlaciones policóricas para variables con más de 8 niveles.

Como método de estimación hemos escogido la versión robusta de ULS, dado que no tenemos normalidad multivariante en nuestros datos.

[1] Se utilizó como criterio por las instrucciones de la práctica. Podríamos seguir hasta que todos los indicadores marquen ajuste. En éste caso, RMSEA tardaba mucho más.

Ya tenemos una solución unifactorial que se ajusta a los datos. Podemos visualizar el modelo con el paquete `semPlot`.

```
modelo <- modelo2
fit <- fit2
semPaths(fit, "std", layout = "circle",
         edge.label.cex=1, residuals = FALSE,
         curvePivot = TRUE)
```



Observamos que las saturaciones de todos los ítems son elevadas en el factor, aunque los valores de los pesos no son los mismos para todos los ítems: es posible que no haya τ equivalencia.

Vemos también que con la excepción de las correlaciones entre V1 y V2, las correlaciones entre ítems son bajas. Seguramente la correlación más alta entre V1 y V2 se deba a que las correlaciones de ambas variables con el factor son elevadas.

Análisis de invarianza

```
config <- cfa(modelo, Datos, group = "SEXO")
weak <- cfa(modelo, Datos, group = "SEXO",
            group.equal = "loadings")
strong<- cfa(modelo, Datos, group ="SEXO",
             group.equal = c("loadings", "intercepts"))
strict<- cfa(modelo, Datos, group = "SEXO",
             group.equal = c("loadings", "intercepts",
                             "residuals"))
anova(config, weak, strong, strict)
```

```
## Chi-Squared Difference Test
##
##        Df   AIC   BIC  Chisq Chisq diff Df diff Pr(>Chisq)
## config 42 27505 27810 57.754
## weak   50 27492 27760 61.203     3.4492       8    0.90310
## strong 58 27490 27721 74.748    13.5443       8    0.09444 .
## strict 67 27482 27672 85.484    10.7363       9    0.29422
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Observamos que tenemos invarianza de medida estricta (ninguno de los desajustes es significativo).

Conclusiones

Tras aplicar una serie de correcciones a la solución factorial conseguimos un modelo que presenta **invarianza de medida estricta**. Con éste tipo de invarianza, es legítimo realizar cualquier tipo de comparación entre los grupos de la variable estudiada (Sexo) para las puntuaciones del instrumento. Con la invarianza débil podemos establecer comparaciones entre las relaciones de las distintas variables latentes entre los distintos grupos. La invarianza escalar nos permite comparar las medias de las variables latentes. Por su parte, la invarianza estricta nos da más garantías en todas éstas comparaciones.

Referencias

Hooper, D., J. Coughlan, and M. R. Mullen. 2008. “Structural Equation Modelling: Guidelines for Determining Model Fit.”