Impacto del Sesgo de Ítems en la Comparabilidad de Grupos: Simulación con Puntuaciones Promediadas, Factor Scores y Modelos MIMIC

Brian Norman Peña-Calero

20 mayo, 2025

Tabla de contenidos

1 Introducción

Este proyecto de simulación investigará cómo la presencia de sesgo en los interceptos de los ítems afecta la comparabilidad de las puntuaciones entre dos grupos. Se examinará el impacto de diferentes magnitudes de sesgo de ítem y diferentes magnitudes de diferencias verdaderas en las medias latentes sobre tres métodos comunes de comparación: puntuaciones promediadas, factor scores derivados de Análisis Factorial Confirmatorio (AFC), y estimaciones de medias latentes de modelos MIMIC (Multiple Indicators Multiple Causes). El estudio también considerará el efecto del tamaño muestral en la precisión de estas comparaciones.

> **!** Importante
>
> Una versión de este informe puede ser encontrada en línea en https://brianmsm.github.io/sesgo-interceptos-comp-grupos/informe.html.

2 Objetivos

2.1 Objetivo General

Evaluar cuantitativamente cómo diferentes magnitudes de sesgo en los interceptos de los ítems y diferentes niveles de diferencia verdadera en las medias latentes entre dos grupos afectan la comparabilidad de las puntuaciones estimadas mediante promedio, factor scores y modelos MIMIC, bajo distintas condiciones de tamaño muestral.

2.2 Objetivos Específicos

a) Determinar el grado de sesgo (en términos de d de Cohen) introducido en la comparación de medias grupales por diferentes niveles de sesgo en los interceptos de los ítems cuando no existe una diferencia real en la media latente entre los grupos.
b) Evaluar cómo el sesgo en los interceptos de los ítems distorsiona la estimación de diferencias de medias latentes verdaderas conocidas (pequeñas, medianas y grandes) entre los grupos.
c) Comparar la robustez y precisión de las puntuaciones promediadas, los factor scores (obtenidos de `lavaan` asumiendo invarianza escalar) y las estimaciones de medias latentes de modelos MIMIC frente al sesgo de los ítems.
d) Investigar la influencia del tamaño muestral (N=400 vs. N=800 total) en la magnitud del sesgo observado y la precisión de las estimaciones de las diferencias de medias.

3 Diseño de Simulación

1. Modelo de Generación de Datos Base:

- Estructura Factorial: 1 factor común.
- Número de Ítems: 10 ítems continuos con distribución normal.
- Cargas Factoriales (λ): Promedio de 0.7 +- 0.10.
- Varianza del Factor Latente (Ψ): Fijada a 1 en el grupo de referencia y para la generación de datos
- Interceptos de los Ítems (ν) Base: Fijados a 0 para todos los ítems en el grupo de referencia.
- Varianzas de Error de los Ítems ( ): Se simula en situación estandarizada por lo que $\theta_j = 1 - \lambda_j^2$.

2. Variables Independientes (Factores de la Simulación):

Factor	Niveles	Descripción del Nivel
1. Diferencia Real Medias Latentes ($\Delta\theta$)	4 niveles: 0, 0.2, 0.5, 0.8	Nula, Pequeña, Mediana, Grande (en unidades de d de Cohen)
2. Magnitud Sesgo Intercepto Ítem ($\Delta\nu$)*	4 niveles: 0, +0.2, +0.5, +0.8	Nulo, Pequeño, Mediano, Grande (en unidades de DE del ítem)
3. Tamaño Muestral Total (N)	2 niveles: 400, 800	200 por grupo, 400 por grupo

- **Nota sobre** $\Delta\nu$: El sesgo en el intercepto se introducirá en el **30% de los ítems (es decir, 3 ítems)**. Estos 3 ítems serán siempre los mismos a lo largo de todas las condiciones. El valor de $\Delta\nu$ indica la magnitud de la diferencia en el intercepto de estos 3 ítems para el Grupo Focal en comparación con el Grupo de Referencia (favoreciendo al Grupo Focal, $\nu_{item,Focal} = \nu_{item,Ref} + \Delta\nu$.

3. Número Total de Condiciones de Simulación: 4 ($\Delta\theta$) x 4 ($\Delta\nu$) x 2 (N) = **32 condiciones**.

4. Número de Replicaciones por Condición: 500

5. Análisis para evaluación:

- Para Puntuaciones Sumadas (o Promedio): d de Cohen observada para la diferencia de medias entre el Grupo Focal y el Grupo de Referencia.
- Para Factor Scores (de `lavaan`): d de Cohen observada para la diferencia de medias de los factor scores entre grupos (estimados de un AFC multigrupo que asume invarianza escalar).

- Para Modelos MIMIC: d de Cohen observada para la diferencia estimada en la media del factor latente entre grupos.
- Métrica de Sesgo Primaria: Para todas las d observadas: $Sesgo = d_{observada} - d_{verdadera}$ (donde $d_{verdadera}$ es el $\Delta\theta$ de la condición).

4 Modelo poblacional y manipulación del sesgo

La variable observada del ítem j para la persona i se genera a partir de un modelo de medición unidimensional:

$$X_{ij} \;=\; \tau_j^{(g_i)} \;+\; \lambda_j\,\theta_i^{(g_i)} \;+\; \varepsilon_{ij}, \qquad j = 1, \ldots, 10,\; i = 1, \ldots, n,$$

donde

Símbolo	Significado	Manipulación en la simulación
$\theta_i^{(g)} \sim \mathcal{N}(\mu_\theta^{(g)}, 1)$	puntuación latente del individuo i en el grupo g	la media se fija a $\mu_\theta^{(0)} = 0$ y se varía $\Delta\theta = \mu_\theta^{(1)} - \mu_\theta^{(0)} \in \{0, 0.2, 0.5, 0.8\}$
λ_j	carga factorial del ítem j	muestreadas una vez de $U(0.60, 0.80)$ y **no se manipulan**
$\tau_j^{(g)}$	intercepto del ítem j en el grupo g	para los ítems $3, 6, 9$ del **grupo focal** se añade $\Delta\nu \in \{0, 0.2, 0.5, 0.8\}$ $\Rightarrow \tau_j^{(1)} = \tau_j^{(0)} + \Delta\nu$
ε_{ij}	error residual, $\varepsilon_{ij} \sim \mathcal{N}(0,\ \sigma_{\varepsilon j}^2)$	$\sigma_{\varepsilon j}^2 = 1 - \lambda_j^2$ para que $\mathrm{Var}(X_{ij}) = 1$

La manipulación experimental se concentra, por tanto, **exclusivamente en los interceptos** de tres ítems — provocando *Differential Intercept Functioning* — y en la media latente $\Delta\theta$. Todos los demás parámetros permanecen invariantes entre los dos grupos.

5 Estadísticos de evaluación

Para cada réplica se calculan:

- Sesgo de d:

$$e_r = \hat{d}_r - d_{\text{true}}$$

- Raíz del error cuadrático medio (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{R}\sum_{r=1}^{R} e_r^2}$$

y sus **intervalos empíricos al 95 %** se obtienen como los cuantiles 2.5 % y 97.5 % de la distribución de e_r (o $|e_r|$ para RMSE) a lo largo de las $R = 500$ réplicas por condición.

6 Simulación

La codificación de la simulación se organizó en **módulos** dentro de la carpeta `03_functions/`. El árbol completo se muestra a continuación:

```
fs::dir_tree("03_functions/")
```

```
03_functions/
+-- 00_config.R
+-- 01_helpers.R
+-- 02_generator.R
+-- 03_estimators.R
+-- 04_sim_one.R
\-- 05_parallel.R
```

Además, todo ello se encuentra almacenado en el repositorio Github: https://github.com/brianmsm/sesgo-interceptos-comp-grupos.

6.1 Config

Define **todas** las condiciones de simulación (niveles de $\Delta\theta$ y $\Delta\nu$), tamaños muestrales y réplicas), crea la estructura de carpetas del proyecto y detecta los núcleos disponibles.
Finalmente guarda el objeto `config_sim.rds`, que actúa como punto único de referencia en los demás scripts.

```
# ------------------------------------------------------------------
# 00_config.R  •  Constantes y rutas del estudio de simulación
# Ubicación: /03_functions/00_config.R
# ------------------------------------------------------------------

# === 1. PARÁMETROS DEL DISEÑO ===========================================
```

```r

n_items        <- 10                          # número de ítems totales
biased_items   <- c(3, 6, 9)                  # ítems con sesgo (30 %)

# Escalas de magnitud (SD de la latente)
Delta_theta_levels <- c(0, 0.2, 0.5, 0.8)     # diferencia de medias latentes
Delta_nu_levels    <- c(0, 0.2, 0.5, 0.8)     # sesgo intercepto en 3 ítems

n_total_levels <- c(400, 800)                 # muestra total (50 %/grupo)

n_reps <- 500                                  # réplicas Monte-Carlo


# === 2. GESTIÓN DE DIRECTORIOS ======================================

suppressPackageStartupMessages(library(here))
base_dir <- here::here()                       # directorio raíz del proyecto

# Estructura definida por el usuario
raw_dir      <- file.path(base_dir, "01_data", "raw")
proc_dir     <- file.path(base_dir, "01_data", "processed")
plots_dir    <- file.path(base_dir, "02_output", "plots")
tables_dir   <- file.path(base_dir, "02_output", "tables")
reports_dir  <- file.path(base_dir, "02_output", "reports")

# === 3. GRADE DESIGN GRID ===========================================

design_grid <- expand.grid(n_total     = n_total_levels,
                           Delta_theta = Delta_theta_levels,
                           Delta_nu    = Delta_nu_levels,
                           KEEP.OUT.ATTRS = FALSE)


# === 4. RECURSOS DE CPU =============================================

n_cores <- max(1, parallel::detectCores())


# === 5. OBJETO CONFIG UNIFICADO =====================================

config <- list(
  # dimensiones y diseño
```

```
  n_items      = n_items,
  biased_items = biased_items,
  Delta_theta_levels = Delta_theta_levels,
  Delta_nu_levels    = Delta_nu_levels,
  n_total_levels     = n_total_levels,
  n_reps       = n_reps,
  design_grid  = design_grid,
  # paths
  dirs = list(raw = raw_dir,
              processed = proc_dir,
              plots = plots_dir,
              tables = tables_dir,
              reports = reports_dir),
  # recursos
  n_cores      = n_cores
)

saveRDS(config, file = file.path(base_dir, "config_sim.rds"))

rm(base_dir)
```

El archivo de configuración contiene:

```
config
```

```
$n_items
[1] 10

$biased_items
[1] 3 6 9

$Delta_theta_levels
[1] 0.0 0.2 0.5 0.8

$Delta_nu_levels
[1] 0.0 0.2 0.5 0.8

$n_total_levels
[1] 400 800

$n_reps
```

```
[1] 500

$design_grid
   n_total Delta_theta Delta_nu
1      400         0.0      0.0
2      800         0.0      0.0
3      400         0.2      0.0
4      800         0.2      0.0
5      400         0.5      0.0
6      800         0.5      0.0
7      400         0.8      0.0
8      800         0.8      0.0
9      400         0.0      0.2
10     800         0.0      0.2
11     400         0.2      0.2
12     800         0.2      0.2
13     400         0.5      0.2
14     800         0.5      0.2
15     400         0.8      0.2
16     800         0.8      0.2
17     400         0.0      0.5
18     800         0.0      0.5
19     400         0.2      0.5
20     800         0.2      0.5
21     400         0.5      0.5
22     800         0.5      0.5
23     400         0.8      0.5
24     800         0.8      0.5
25     400         0.0      0.8
26     800         0.0      0.8
27     400         0.2      0.8
28     800         0.2      0.8
29     400         0.5      0.8
30     800         0.5      0.8
31     400         0.8      0.8
32     800         0.8      0.8

$dirs
$dirs$raw
[1] "D:/Github/sesgo-interceptos-comp-grupos/01_data/raw"

$dirs$processed
[1] "D:/Github/sesgo-interceptos-comp-grupos/01_data/processed"
```

```
$dirs$plots
[1] "D:/Github/sesgo-interceptos-comp-grupos/02_output/plots"

$dirs$tables
[1] "D:/Github/sesgo-interceptos-comp-grupos/02_output/tables"

$dirs$reports
[1] "D:/Github/sesgo-interceptos-comp-grupos/02_output/reports"


$n_cores
[1] 12
```

6.2 Helpers

Funciones utilitarias comunes:

- `make_lambda()` — genera las cargas $\lambda_j \sim U(0.60, 0.80)$.
- `cohen_d()` y `z_score()` — estadísticas básicas.
- `safe_cfa()` / `safe_sem()` — wrappers que silencian 'warnings' de **lavaan**.

```r
# ------------------------------------------------------------------
# 01_helpers.R  •  Funciones genéricas reutilizadas en la simulación
# Ubicación: project/03_functions/01_helpers.R
# ------------------------------------------------------------------

# ---- 1. CARGAR CONFIGURACIÓN ---------------------------------------

suppressPackageStartupMessages({
  library(tidyverse)
  library(lavaan)
})

config <- readRDS(here::here("config_sim.rds"))

# Atajo para rutas (por legibilidad):
DIRS <- config$dirs

# ---- 2. GENERADORES AUXILIARES -------------------------------------
```

```r

# 2.1  make_lambda(): vector de cargas   ~ U(0.60, 0.80)
make_lambda <- function(n = config$n_items,
                        min_lambda = 0.60,
                        max_lambda = 0.80) {
  runif(n, min_lambda, max_lambda)
}

# 2.2  cohen_d(): diferencia estandarizada entre dos grupos (0/1)
cohen_d <- function(x, g) {
  if (!is.numeric(x)) x <- as.numeric(x)
  g <- as.integer(as.character(g))
  if (!all(sort(unique(g)) == c(0, 1)))
    stop("El vector de grupos debe contener exactamente 0 y 1")
  x1 <- x[g == 0]
  x2 <- x[g == 1]
  s_pooled <- sd(c(x1, x2))
  (mean(x2) - mean(x1)) / s_pooled
}

# 2.3  z_score(): estandariza un vector (media 0, sd 1)
z_score <- function(v) {
  (v - mean(v)) / sd(v)
}

# ---- 3. WRAPPERS PARA AJUSTES LAVAAN ------------------------------

# 3.1  safe_cfa(): CFA multigrupo con control de warnings/errores
safe_cfa <- function(model, data, ...) {
  tryCatch(
    cfa(model, data = data, warn = FALSE, std.lv = TRUE, ...),
    error   = function(e) NULL,
    warning = function(w) invokeRestart("muffleWarning")
  )
}

# 3.2  safe_sem(): para MIMIC
safe_sem <- function(model, data, ...) {
  tryCatch(
    sem(model, data = data, warn = FALSE, std.lv = TRUE, ...),
    error   = function(e) NULL,
    warning = function(w) invokeRestart("muffleWarning")
```

```
  )
}

# ---- 4. LOGGING / MENSAJES ----------------------------------------

log_info <- function(msg) {
  cat(sprintf("[%s] %s\n", format(Sys.time(), "%H:%M:%S"), msg))
}
```

6.3 Generator

Implementa dos motores de simulación:

1. **`manual_generator()`** — más rápido y transparente: añade el sesgo ($\Delta\nu$) sólo a los ítems 3-6-9 del grupo focal.

2. **`lavaan_generator()`** — usa `simulateData()` por si se quiere reproducir exactamente la matriz poblacional.

Función principal `generate_data()`, con argumento `engine = "manual" | "lavaan"`.

```
# ------------------------------------------------------------------
# 02_generator.R  •  Creación de datos poblacionales y simulados
# Ubicación: project/03_functions/02_generator.R
# ------------------------------------------------------------------

# Este módulo provee una única función pública:
#   generate_data()
# que regresa un data.frame con las variables simuladas para una réplica.

source(here::here("03_functions/01_helpers.R"))

# ---- 1. FUNCIÓN AUXILIAR: RESIDUAL VARIANCE ----------------------

compute_resid_var <- function(lambda_vec) {
  1 - lambda_vec^2
}

# ---- 2. GENERADOR MANUAL (más rápido y flexible) -----------------

manual_generator <- function(n_total, Delta_theta, Delta_nu, lambda_vec,
                             bias_idx) {
```

```r
  n_per_group <- n_total / 2
  stopifnot(n_per_group %% 1 == 0)

  # -- Paso 1: generar latentes  ------------------------------------
  theta <- c(rnorm(n_per_group, 0, 1),
             rnorm(n_per_group, Delta_theta, 1))
  grp   <- rep(c(0, 1), each = n_per_group)

  # -- Paso 2: interceptos  -----------------------------------------
  tau_base <- numeric(config$n_items)
  tau_g1   <- tau_base
  tau_g1[bias_idx] <- tau_g1[bias_idx] + Delta_nu

  # -- Paso 3: varianza residual ------------------------------------
  resid_var <- compute_resid_var(lambda_vec)

  # -- Paso 4: ensamblar matriz de ítems ----------------------------
  X <- matrix(NA_real_, n_total, config$n_items)
  for (j in seq_len(config$n_items)) {
    current_tau <- ifelse(grp == 1, tau_g1[j], tau_base[j])
    X[, j] <- current_tau + lambda_vec[j] * theta +
      rnorm(n_total, 0, sqrt(resid_var[j]))
  }
  colnames(X) <- paste0("Item", seq_len(config$n_items))

  tibble::tibble(grp = factor(grp), theta_true = theta) |>
    dplyr::bind_cols(as.data.frame(X))
}

# ---- 3. GENERADOR CON lavaan::simulateData --------------

lavaan_generator <- function(n_total, Delta_theta, Delta_nu, lambda_vec,
                             bias_idx) {
  n_per_group <- n_total / 2

  # *Modelo poblacional*
  model_lines <- c(
    paste0("F =~ ", paste(paste0(lambda_vec, "*Item", seq_len(config$n_items)), collapse = "
    # interceptos base = 0, luego añadimos sesgo a los 3 ítems
    paste0("Item", bias_idx, " ~ start(0) + ", Delta_nu, "*1"),
    paste0("Item", setdiff(seq_len(config$n_items), bias_idx), " ~ 0*1"),
    "F ~~ 1*F"  # varianza latente = 1
```

```r
  )
  pop_model <- paste(model_lines, collapse = "\n")

  dat <- lavaan::simulateData(model = pop_model,
                              model.type = "cfa",
                              meanstructure = TRUE,
                              group.label = c("0", "1"),
                              sample.nobs = c(n_per_group, n_per_group),
                              group.w.free = FALSE,
                              std.lv = TRUE) |>
    as.data.frame()

  # simulateData no permite medias latentes diferentes fácilmente;
  # ajustamos manualmente para el grupo 1
  idx_g1 <- seq_len(n_per_group) + n_per_group
  dat$F[idx_g1] <- dat$F[idx_g1] + Delta_theta

  dat$grp <- factor(c(rep(0, n_per_group), rep(1, n_per_group)))
  dat$theta_true <- dat$F
  dat$F <- NULL

  dat[c("grp", paste0("Item", seq_len(config$n_items)), "theta_true")]
}

# ---- 4. Función generadora ------------------------------------------

#' generate_data
#'
#' @param n_total     Tamaño muestral total (parejo entre grupos)
#' @param Delta_theta Diferencia de medias latentes (SD units)
#' @param Delta_nu    Sesgo de intercepto aplicado a los ítems sesgados
#' @param lambda_vec  Vector de cargas; se genera con make_lambda() si NULL
#' @param bias_idx    Índices de ítems sesgados (default config$biased_items)
#' @param engine      "manual" (default) o "lavaan" para usar simulateData
#'
#' @return data.frame con: grp (factor 0/1), Item1:Item10, theta_true
#' @export
generate_data <- function(n_total, Delta_theta, Delta_nu,
                          lambda_vec = NULL,
                          bias_idx   = config$biased_items,
                          engine     = c("manual", "lavaan")) {
  engine <- match.arg(engine)
```

```r
  if (is.null(lambda_vec)) lambda_vec <- make_lambda()

  switch(engine,
         manual = manual_generator(n_total, Delta_theta,
                                   Delta_nu, lambda_vec, bias_idx),
         lavaan = lavaan_generator(n_total, Delta_theta,
                                   Delta_nu, lambda_vec, bias_idx))
}

# ---- 5. PRUEBA RÁPIDA ---------------

if (interactive()) {
  log_info("Test rápido de generate_data...")
  test_df <- generate_data(400, 0.5, 0.2)
  print(head(test_df))
  log_info("OK!")
}
```

Ejemplo:

```r
data_ejemplo <- generate_data(400, 0.5, 0.2)
data_ejemplo
```

```
# A tibble: 400 x 12
   grp   theta_true   Item1   Item2   Item3   Item4    Item5  Item6  Item7
   <fct>       <dbl>   <dbl>   <dbl>   <dbl>   <dbl>    <dbl>  <dbl>  <dbl>
 1 0           0.150  0.0768  0.302  -0.0628 -0.112   0.428   -0.521  0.728
 2 0          -0.573  1.17    0.106  -1.06   -0.663   0.842   -0.645  0.320
 3 0          -0.881 -0.434  -0.718  -0.439  -0.868  -0.435   -1.45  -0.603
 4 0           0.198  0.568  -0.907  -0.380  -0.452  -0.331   -0.360  0.243
 5 0           1.13   1.81    1.22    0.972   1.41   -0.00100  1.34   0.159
 6 0           0.605  0.342   1.22   -1.10   -0.0277  0.496    0.420  0.518
 7 0          -0.387 -0.156  -0.0958  0.181  -0.377  -0.0183  -0.876 -0.483
 8 0          -0.137  0.721  -0.842  -0.125  -0.0989 -0.375   -0.439 -0.237
 9 0          -0.517 -0.387  -0.652  -0.545   0.241   0.455   -0.788 -0.564
10 0          -0.751 -0.286  -0.713  -1.19   -0.0703 -0.478    0.576  0.615
# i 390 more rows
# i 3 more variables: Item8 <dbl>, Item9 <dbl>, Item10 <dbl>
```

6.4 Estimators

- Ajusta un **CFA** para recuperar la diferencia de medias latentes — devuelta como `d_fscore`.
- Calcula `d_mean` (promedio de ítems estandarizado) y `d_mimic` (pendiente *grupo* → *factor* en modelo MIMIC).
- La función de alto nivel `analyse_data()` empaqueta todo en un `list`.

```r
# Este módulo proporciona funciones para obtener:
#   • Sum score estandarizado
#   • Factor scores vía CFA multigrupo
#   • Pendiente grupo→factor del modelo MIMIC
# Y calcula los correspondientes Cohen d.
# ------------------------------------------------------------------
# 03_estimators.R  •  Ajustes de modelos y métricas derivadas
# Ubicación: project/03_functions/03_estimators.R
# ------------------------------------------------------------------

source(here::here("03_functions/01_helpers.R"))
source(here::here("03_functions/02_generator.R"))

# ---- 1. MEAN SCORE ---------------------------------------------

mean_score <- function(items_mat) {
  rowMeans(items_mat)
}

# ---- 2. FACTOR SCORES (CFA) ------------------------------

fit_cfa_simple <- function(data_df) {
  model <- paste0("F =~ ", paste(paste0("Item", 1:config$n_items),
                                 collapse = " + "))

  safe_cfa(model, data = data_df)
}

get_factor_scores <- function(fit, n_obs) {
  # Devuelve vector numérico con length == n_obs (orden original de casos)
  if (is.null(fit)) return(rep(NA_real_, n_obs))

```

```r
  fs <- lavPredict(fit, method = "Bartlett")
  # lavaan devuelve lista por grupo si group.arg=NULL y hay grupos
  if (is.list(fs)) {
    fs <- do.call(rbind, fs)  # concatena
  }
  if (is.matrix(fs)) fs <- fs[, 1]
  fs <- as.numeric(fs)

  # Chequeo defensivo
  if (length(fs) != n_obs) {
    warning("Factor scores length (", length(fs),
            ") != n_obs (", n_obs, ") - se rellenará con NA")
    fs <- rep(NA_real_, n_obs)
  }
  fs
}

# ---- 3. MODELO MIMIC ------------------------------------------- MODELO MIMIC ----------

fit_mimic <- function(data_df) {
  model <- paste0("F =~ ", paste(paste0("Item", 1:config$n_items),
                                 collapse = " + "),
                  "\nF ~ grp")
  safe_sem(model, data = data_df)
}

get_mimic_d <- function(fit) {
  if (is.null(fit)) return(NA_real_)
  parameterEstimates(fit) %>%
    dplyr::filter(lhs == "F", op == "~", rhs == "grp") %>%
    dplyr::pull(est) %>%
    dplyr::first()
}

# ---- 4. MÉTRICAS DE COMPARABILIDAD ----------------------------

compute_metrics <- function(data_df, f_scores, mimic_d) {
  items_mat <- as.matrix(data_df[ , grep("^Item", names(data_df))])

  d_mean <- cohen_d(z_score(mean_score(items_mat)), data_df$grp)
  d_fscore <- if (all(is.na(f_scores))) NA_real_ else cohen_d(f_scores,
                                                              data_df$grp)
```

```r

  list(d_mean = d_mean,
       d_fscore = d_fscore,
       d_mimic = mimic_d)
}

# ---- 5. FUNCIÓN DE ALTO NIVEL: analyse_data -----------------------

#' analyse_data
#'
#' @param data_df  data.frame generado por generate_data()
#'
#' @return list con: d_true y los d estimados (mean, fscore, mimic)
#' @export
analyse_data <- function(data_df) {
  n_obs <- nrow(data_df)

  # d verdadero a partir de theta_true
  d_true <- cohen_d(data_df$theta_true, data_df$grp)

  # CFA & factor scores
  fit_cfa <- fit_cfa_simple(data_df)
  f_scores <- get_factor_scores(fit_cfa, n_obs)

  # MIMIC
  fit_mim <- fit_mimic(data_df)
  d_mimic <- get_mimic_d(fit_mim)

  # Métricas
  metrics <- compute_metrics(data_df, f_scores, d_mimic)

  c(list(d_true = d_true), metrics)
}

# ---- 6. TEST INTERACTIVO ------------------------------------------

if (interactive()) {
  log_info("Test rápido de analyse_data() ...")
  tmp <- generate_data(500, 0.5, 0.2)
  print(analyse_data(tmp))
}
```

Ejemplo:

```
analisis_ejemplo <- analyse_data(data_ejemplo)
analisis_ejemplo
```

```
$d_true
[1] 0.5002165

$d_mean
[1] 0.5532032

$d_fscore
[1] 0.5723635

$d_mimic
[1] 0.6262107
```

6.5 Sim One

Integra **una réplica** completa:

1. Fija semilla.
2. Llama a `generate_data()` y `analyse_data()`.
3. Devuelve un `tibble` con `d_true`, `d_mean`, `d_fscore`, `d_mimic`.
4. (Opcional) guarda los datos crudos en `01_data/raw/`.

```
# ------------------------------------------------------------------
# 04_sim_one.R  •  Ejecuta UNA réplica de la simulación
# Ubicación: project/03_functions/04_sim_one.R
# ------------------------------------------------------------------
# Proporciona la función pública:  sim_one()
# ------------------------------------------------------------------

source(here::here("03_functions/01_helpers.R"))
source(here::here("03_functions/02_generator.R"))
source(here::here("03_functions/03_estimators.R"))

# ---- 1. FUNCIÓN PRINCIPAL ----------------------------------------
```

```r

#' sim_one
#'
#' Ejecuta una réplica de la simulación Monte-Carlo.
#'
#' @param seed        Entero semilla para reproducibilidad.
#' @param n_total     Tamaño muestral total (parejo entre grupos).
#' @param Delta_theta Diferencia verdadera de medias latentes (SD).
#' @param Delta_nu    Magnitud del sesgo de intercepto en los ítems sesgados.
#' @param save_raw    Lógico. ¿Guardar los datos simulados en 01_data/raw?
#' @param engine      "manual" (predeterminado) o "lavaan" para generar datos.
#'
#' @return Un tibble con columnas de condiciones + métricas
#'   (d_true, d_mean, d_fscore, d_mimic).
#' @export
sim_one <- function(seed, n_total, Delta_theta, Delta_nu,
                    save_raw = FALSE,
                    engine = c("manual", "lavaan")) {
  engine <- match.arg(engine)
  set.seed(seed)

  # --- Generar datos ------------------------------------------------
  data_df <- generate_data(n_total = n_total,
                           Delta_theta = Delta_theta,
                           Delta_nu = Delta_nu,
                           engine = engine)

  # --- Analizar datos -----------------------------------------------
  metrics_list <- analyse_data(data_df)

  # --- Resultado ----------------------------------------------------
  res <- tibble::tibble(seed = seed,
                        n_total = n_total,
                        Delta_theta = Delta_theta,
                        Delta_nu = Delta_nu,
                        !!!metrics_list)

  # --- Guardado opcional --------------------------------------------
  if (isTRUE(save_raw)) {
    file_tag <- sprintf("n%d_dt%.2f_dn%.2f_seed%09d.rds",
                        n_total, Delta_theta, Delta_nu, seed)
    fpath <- file.path(DIRS$raw, file_tag)
```

```
    saveRDS(list(data = data_df, metrics = res), fpath)
  }

  res
}

# ---- 2. TEST RÁPIDO ------------------------------------------------

if (interactive()) {
  log_info("Test rápido de sim_one() ...")
  out <- sim_one(seed = 2025, n_total = 400, Delta_theta = 0.5, Delta_nu = 0.2)
  print(out)
}
```

Ejemplo:

```
sim_one(seed = 2025, n_total = 400,
        Delta_theta = 0.5, Delta_nu = 0.2)
```

```
# A tibble: 1 x 8
   seed n_total Delta_theta Delta_nu d_true d_mean d_fscore d_mimic
  <dbl>   <dbl>       <dbl>    <dbl>  <dbl>  <dbl>    <dbl>   <dbl>
1  2025     400         0.5      0.2  0.520  0.597    0.599   0.658
```

6.6 Parallel

Expande el grid (32 condiciones × 500 réplicas), asigna **una semilla única por fila** y lanza el cálculo en paralelo con **future + progressr**:

- Guarda `sim_results_raw.rds`

```
# ------------------------------------------------------------------
# 05_parallel.R  •  Lanza todas las réplicas en paralelo y resume
# Ubicación: project/03_functions/05_parallel.R
# ------------------------------------------------------------------

source(here::here("03_functions/01_helpers.R"))
source(here::here("03_functions/04_sim_one.R"))

suppressPackageStartupMessages({
  library(future.apply)
```

```r
  library(progressr)      # barra de progreso compatible con future
  library(tidyverse)
})

# ---- 1. CONFIGURAR PLAN DE FUTURE ---------------------------------

plan(multisession, workers = config$n_cores)
log_info(sprintf("Usando %d núcleos", config$n_cores))

# ---- 2. EXPANDIR GRID × RÉPLICAS ----------------------------------

grid <- config$design_grid
big_grid <- purrr::map_dfr(seq_len(config$n_reps), ~ grid) %>%
  dplyr::mutate(rep = rep(seq_len(config$n_reps), each = nrow(grid)))

# --- Semilla única por fila (estrategia B) -------------------------
set.seed(42)  # reproducibilidad del vector de semillas
big_grid$seed <- sample.int(.Machine$integer.max, nrow(big_grid))

n_jobs <- nrow(big_grid)
log_info(sprintf("Total de réplicas a ejecutar: %d", n_jobs))

# ---- 3. BARRA DE PROGRESO + EJECUCIÓN -----------------------------

handlers("progress")

results <- with_progress({
  p <- progressor(steps = n_jobs)
  future_lapply(seq_len(n_jobs),
                future.seed=TRUE,
                function(i) {
    row <- big_grid[i, ]
    res <- sim_one(seed = row$seed,
                   n_total = row$n_total,
                   Delta_theta = row$Delta_theta,
                   Delta_nu = row$Delta_nu)
    # Mensaje con la condición actual
    cond_msg <- sprintf("n=%d, Δ =%.1f, Δ =%.1f",
                        row$n_total, row$Delta_theta,
                        row$Delta_nu)
    p(message = cond_msg)
    res
```

```
  })
}) %>% dplyr::bind_rows()

# ---- 4. GUARDAR RESULTADOS CRUDOS ---------------------------------

saveRDS(results, file = file.path(DIRS$processed, "sim_results_raw.rds"))

log_info("Simulación completada y guardada.")
```

Una vez cargado y configurado, se procedió a realizar la simulación:

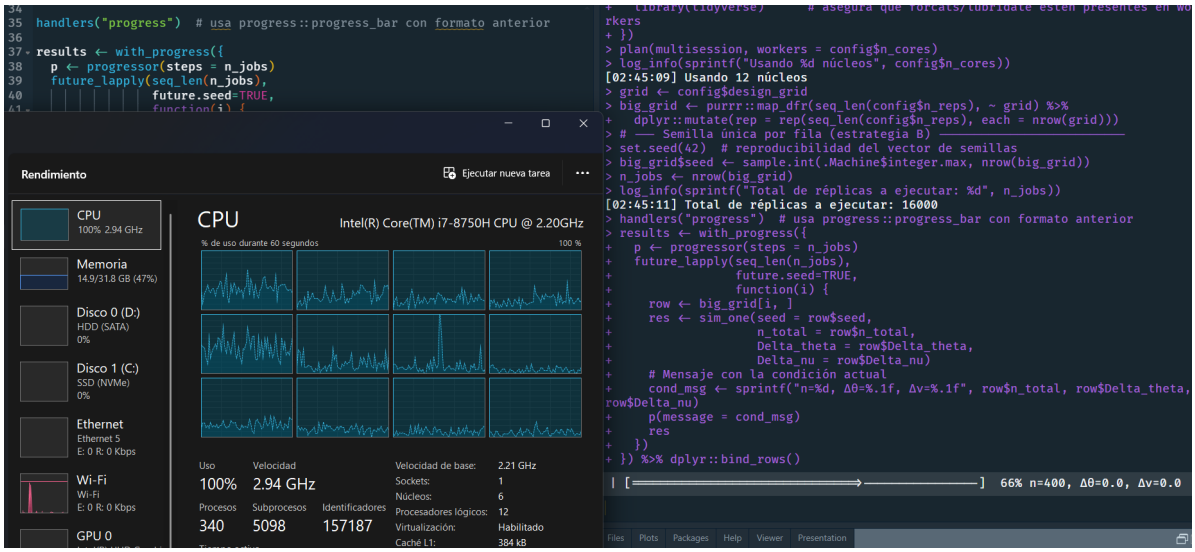


Figura 1: Simulación usando paralelización con todos los núcleos activos

> **Nota**
>
> Con esta estructura modular resulta sencillo depurar o sustituir solo la parte interesada (p. ej., cambiar el generador o añadir un nuevo estimador) sin afectar al resto del flujo.

7 Análisis

Realizaremos primero un resumen de las simulaciones enfocados en *bias* y *rmse*.

```r
config <- readRDS(here::here("config_sim.rds"))
DIRS   <- config$dirs
raw_rds <- file.path(DIRS$processed, "sim_results_raw.rds")

results <- readRDS(raw_rds)

# ---- RESUMEN ---------

bias_cols <- c("d_mean", "d_fscore", "d_mimic")

summary_tbl <- results %>%
  group_by(n_total, Delta_theta, Delta_nu) %>%
  summarise(
    across(all_of(bias_cols),
           ## cada función devuelve un escalar
           list(
             bias_mean  = ~ mean(.x - d_true, na.rm = TRUE),
             bias_low   = ~ quantile(.x - d_true,  .025, na.rm = TRUE),
             bias_high  = ~ quantile(.x - d_true,  .975, na.rm = TRUE),
             rmse_mean  = ~ sqrt(mean((.x - d_true)^2, na.rm = TRUE)),
             rmse_low   = ~ quantile(abs(.x - d_true), .025, na.rm = TRUE),
             rmse_high  = ~ quantile(abs(.x - d_true), .975, na.rm = TRUE)
           ),
           .names = "{.col}_{.fn}")
  ) %>%
  ungroup()
```

```
`summarise()` has grouped output by 'n_total', 'Delta_theta'. You can override
using the `.groups` argument.
```

```r
summary_tbl
```

```
# A tibble: 32 x 21
   n_total Delta_theta Delta_nu d_mean_bias_mean d_mean_bias_low
     <dbl>       <dbl>    <dbl>            <dbl>           <dbl>
 1     400         0        0          -0.000966        -0.0633
 2     400         0        0.2         0.0821           0.0216
 3     400         0        0.5         0.202            0.134
 4     400         0        0.8         0.322            0.259
 5     400         0.2      0          -0.00930         -0.0668
 6     400         0.2      0.2         0.0729           0.0159
```

```
 7     400         0.2      0.5         0.185            0.127
 8     400         0.2      0.8         0.300            0.234
 9     400         0.5      0          -0.0204          -0.0758
10     400         0.5      0.2         0.0538          -0.00407
# i 22 more rows
# i 16 more variables: d_mean_bias_high <dbl>, d_mean_rmse_mean <dbl>,
#   d_mean_rmse_low <dbl>, d_mean_rmse_high <dbl>, d_fscore_bias_mean <dbl>,
#   d_fscore_bias_low <dbl>, d_fscore_bias_high <dbl>,
#   d_fscore_rmse_mean <dbl>, d_fscore_rmse_low <dbl>,
#   d_fscore_rmse_high <dbl>, d_mimic_bias_mean <dbl>, d_mimic_bias_low <dbl>,
#   d_mimic_bias_high <dbl>, d_mimic_rmse_mean <dbl>, ...
```

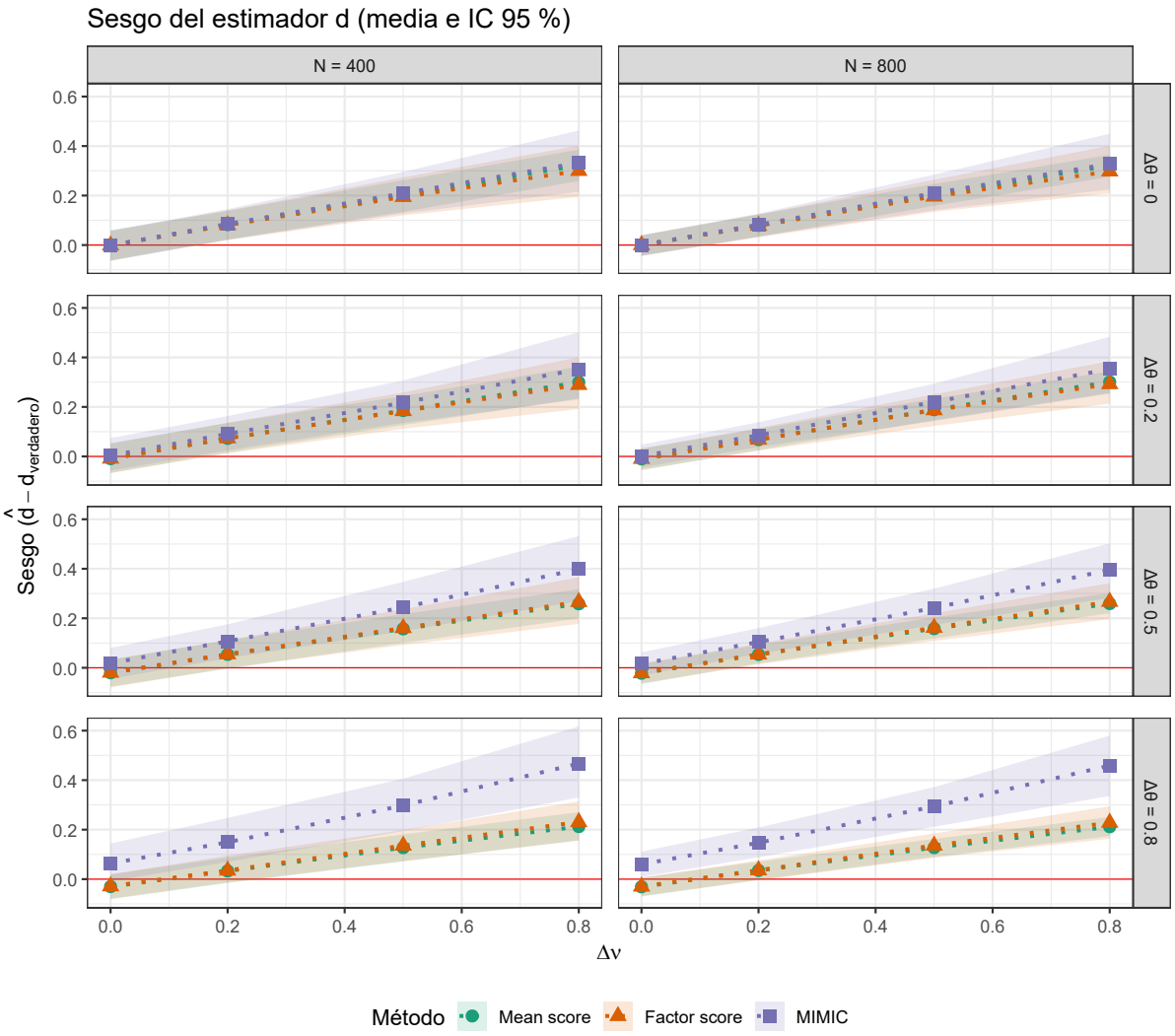
7.1 Bias

En el gráfico de sesgo observamos cuánto se desplaza, en promedio, la estimación de la diferencia de medias ($\hat{d}$) con respecto al valor poblacional verdadero. Cuando el sesgo de intercepto es nulo ($\Delta\nu = 0$) las tres curvas de los métodos se sitúan prácticamente sobre la línea roja de referencia, confirmando que el procedimiento es imparcial en ausencia de distorsión. Sin embargo, basta incrementar gradualmente $\Delta\nu$ para que todas las líneas asciendan con una pendiente casi perfecta: cuanto mayor es la distorsión en los interceptos, mayor la sobre-estimación de la diferencia entre grupos. El método basado en MIMIC es el que muestra más rápidamente esa desviación, seguido por el promedio de ítems y por los factor scores; este orden se mantiene en todos los tamaños muestrales y en los cuatro escenarios de diferencia latente ( ). Las bandas sombreadas, que representan el intervalo empírico del 95 %, se vuelven apenas más estrechas al pasar de N=400 a N=800N, lo que indica que aumentar la muestra reduce muy poco la incertidumbre cuando el sesgo sistemático domina el error.

```r
bias_long <- summary_tbl %>%
  pivot_longer(cols = matches("_bias_(mean|low|high)$"),
               names_to  = c("method", "stat"),
               names_pattern = "(d_[^_]+)_bias_(.*)",
               values_to = "value") %>%
  pivot_wider(names_from = stat, values_from = value) %>%
  mutate(method = factor(method,
                         levels = c("d_mean","d_fscore","d_mimic"),
                         labels = c("Mean score","Factor score","MIMIC")))
```

```r
ggplot(bias_long,
                 aes(x = Delta_nu, y = mean,
                     colour = method, fill = method,
                     shape = method, group = method)) +
  geom_hline(yintercept = 0, colour = "red", linewidth = .3) +
```

```r
  geom_ribbon(aes(ymin = low, ymax = high), alpha = .15, colour = NA) +
  geom_line(linewidth = 1, linetype = "dotted") +
  geom_point(size = 3) +
  facet_grid(Delta_theta ~ n_total,
             labeller = labeller(
               Delta_theta = \(x) sprintf("Δ  = %s", x),
               n_total     = \(x) sprintf("N = %s", x))) +
  scale_colour_brewer(palette = "Dark2", name = "Método") +
  scale_fill_brewer(palette = "Dark2",  name = "Método") +
  scale_shape_manual(values = c(16, 17, 15), name = "Método") +
  labs(title = "Sesgo del estimador d (media e IC 95 %)",
       x = expression(Delta*nu),
       y = expression("Sesgo ("*hat(d) - d[verdadero]*")")) +
  theme_bw(base_size = 12) +
  theme(legend.position = "bottom",
        panel.spacing.y  = unit(.8, "lines"),
        panel.spacing.x  = unit(.6, "lines"))
```

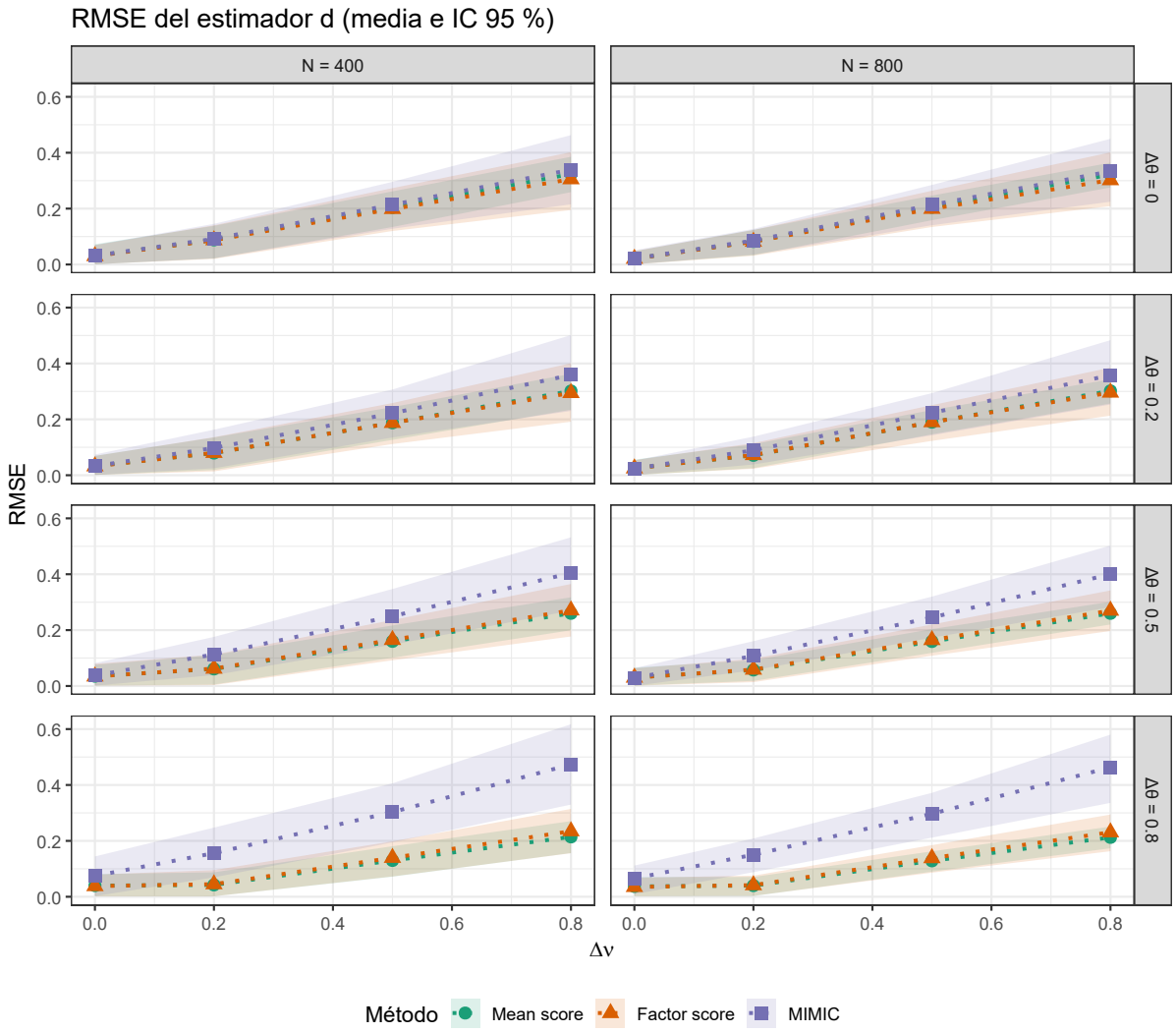


7.2 RMSE

El gráfico de RMSE confirma esa lectura. El RMSE combina varianza entre réplicas y sesgo al cuadrado; sin embargo, las curvas resultan casi superponibles a las del sesgo medio porque la variabilidad de las estimaciones es muy pequeña en comparación con la magnitud del sesgo introducido por $\Delta\nu$. En otras palabras, los estimadores son muy precisos pero potencialmente muy inexactos: convergen de forma consistent hacia un valor equivocado cuando los interceptos están contaminados. De nuevo, las diferencias entre métodos son claras: MIMIC acumula el mayor error total, mientras que el promedio y los factor scores muestran un crecimiento más moderado.

```r
rmse_long <- summary_tbl %>%
  pivot_longer(cols = matches("_rmse_(mean|low|high)$"),
               names_to  = c("method", "stat"),
               names_pattern = "(d_[^_]+)_rmse_(.*)",
               values_to = "value") %>%
  pivot_wider(names_from = stat, values_from = value) %>%
  mutate(method = factor(method,
                         levels = c("d_mean","d_fscore","d_mimic"),
                         labels = c("Mean score","Factor score","MIMIC")))
```

```r
ggplot(rmse_long,
                 aes(x = Delta_nu, y = mean,
                     colour = method, fill = method,
                     shape = method, group = method)) +
  geom_ribbon(aes(ymin = low, ymax = high), alpha = .15, colour = NA) +
  geom_line(linewidth = 1, linetype = "dotted") +
  geom_point(size = 3) +
  facet_grid(Delta_theta ~ n_total,
             labeller = labeller(
               Delta_theta = \(x) sprintf("Δ  = %s", x),
               n_total     = \(x) sprintf("N = %s", x))) +
  scale_colour_brewer(palette = "Dark2", name = "Método") +
  scale_fill_brewer(palette = "Dark2",  name = "Método") +
  scale_shape_manual(values = c(16, 17, 15), name = "Método") +
  labs(title = "RMSE del estimador d (media e IC 95 %)",
       x = expression(Delta*nu), y = "RMSE") +
  theme_bw(base_size = 12) +
  theme(legend.position = "bottom",
        panel.spacing.y  = unit(.8, "lines"),
        panel.spacing.x  = unit(.6, "lines"))
```



8 Conclusiones

En conjunto, los análisis muestran que la fiabilidad de una prueba de diferencia de medias entre grupos depende mucho más de la presencia de sesgo en los ítems que del tamaño de la muestra manejado en este estudio. Aún duplicando el número de participantes, la sobre-estimación permanece prácticamente intacta si no se corrige el problema de los interceptos o al menos se diagnostica para controlarlo adecuadamente.

También es relevante el sesgo que se introduce con el modelo mimic que parte de evaluar las diferencias en un contexto de modelamiento de ecuaciones estructurales (SEM) cuando no se verifica apropiadamente un análisis de invarianza escalar (interceptos) o análisis DIF de

ítems. Emplear este método puede incrementar aún más el sesgo en las diferencias de grupos empleadas.

Por último, la pequeña diferencia entre el método `Factor score` y `Mean score` podría deberse a que en simulación únicamente se manipuló el parámetro de intercepto (ν) y no las cargas factoriales diferentes entre grupos (λ).